HEURISTIC OPTIMIZATION OF MICROCONTROLLERS

by

Andrew Nagle

DRC - 18 - 34 - 81

August, 1981

Heuristic Optimization of Microcontrollers

Andrew Nagle

Bell Laboratories
Holmdel, New Jersey 07733

ABSTRACT

We describe a heuristic method for constructing and optimizing a microprogrammed controller. The input is a control flow graph. The output is a specification of a microcontroller including layout of the microword and the contents of the microprogram memory. The optimization performed uses a clustering technique to decide which control signals should appear together in a single microword. As more signals are clustered together, more parallelism becomes possible. We initially assume no parallelism and correspondingly small signal clusters. This corresponds to a highly encoded(vertical) microcontroller. Using a scheme of "attraction weights" we then merge clusters together until the desired degree of parallelism, or the maximum width of control word is reached, whichever comes first. The controller in which all clusters are merged corresponds to a horizontal microcontroller. We describe a canonical control structure, the clustering algorithm, the computer implementation, and some results.

1. INTRODUCTION

The goal of the research described here[1] has been to specify a formal method for automating the design of a microprogram based controller. The method was to be applicable after completion of the data path design. It was to be capable of producing a variety of implementations at different points in the cost/speed design space by adapting to the cost and speed constraints supplied by the designer. Since microcode optimization with a given micro-engine had been shown to be NP complete, and this problem

1. This work is part of the author's graduate study in association with Carnegie-Mellon University under Bell Laboratories Graduate Tuition Reimbursement Program.

adds yet another degree of freedom, the method was to be heuristic. And finally, although we were working in the context of completely automatic synthesis, the method was to have general applicability for use in interactive design aids.

The method we selected meets these constraints by dealing with the microprogram control word format. Briefly, we divide the total number of control signals into groups and then design a microword format that can produce one or more of these groups at a time. If we constrain the microword to be as narrow as possible, the microword will produce only one group at a time. This allows only one operation at a time, keeping down the cost while driving up the speed. As we allow the microword to contain more groups, more and more parallelism can be implemented. A completely horizontal microword permits the maximum degree of parallelism. Thus we can manage the degree of parallelism (speed) and the width of the control word (cost) simply by managing the grouping of control signals into microword formats. Furthermore we can trade one for the other iteratively until we reach a satisfactory solution.

2. BACKGROUND

We begin by surveying the work that has been reported in microprogram optimization in the past.

Optimization of the two dimensions of a microprogram memory, the number of words in the memory and the number of bits per word, has been addressed separately in the literature on microcode optimization. Since the individual algorithms for the reduction of each dimension have computational complexity problems, no attention has been paid to the more complex problem addressed here: the influence of one dimension on the other. Before beginning our discussion of this problem, we will briefly review the previous work done in these two fields.

2.1 Control Word Width Reduction

Automatic reduction in the bit dimension of microprogram memories has been handled by direct encoding of "compatible" columns of the microprogram memory. (e.g. if there are 7 columns in which at any one time zero or exactly one column is a logic ONE, then these seven columns can be encoded by three.) All of the algorithms that have been presented require the same form of input data: a complete specification of the memory contents before reduction.

Seven different authors have reported techniques for bit dimension reduction in the literature: Schwarz[SCHW68], Grassilli and Montanari[GRAS70], Das et al[DAS73] Montangero[MONT74], Jayasri and Basu[JAYA76], Halatsis and Gaitanis[HALA78], and Baer and Koyama[BAER79]. They share a common control store model: each bit in the control word is postulated to encode a single micro-operation. The logic ONES in any given word stipulate which micro-operations are to be evoked in parallel when that word is executed. Each of the optimization techniques proposes a different algorithm for encoding micro-operations that are never used in parallel. This problem has recently been shown to be NP complete[ROBE79]. We review the published algorithms briefly below.

Schwartz introduced the notion of encoding groups of micro-operations, no two of which ever occur together in the same word. His algorithm first finds all groups of encodable signals. Then it postulates that a solution exists with a number of groups equal to the number of micro-operations evoked in the densest word. The algorithm searches for a valid grouping while enumerating all combinations of this number of groups. Failing to find one, it increments the number of groups and searches again.

Grasselli and Montanari pointed out that Schwartz's minimum group solution does not guarantee a minimum word width, however. They formalized Schwartz's notion by defining a compatibility relation between micro-operations and then showed that the minimum word width solution can be obtained by solving a covering table of the prime implicant type.

Das et al propose a slightly different method for determining the minimum solution based on the compatibility relation defined by Grasselli and Montanari. They prune the original cover table and generate all possible solutions to the pruned version. From each of these solutions they build another smaller cover table and find solutions for it. Using these to augment the solutions of the first cover table, they find the final solution set. After evaluation of all members of this set, they choose the minimum solution.

Jayasri and Basu propose yet another method for finding the minimum solution. After an initial analysis based on the same compatibility relation as the previous two, they compute a theoretical lower bound for the width of the control word. Based on this lower bound and some other statistics measured from the microprogram, they compute the number of groups and the size of the groups needed to achieve the lower bound. They then test for the existence of this optimal solution to the given problem. If they find it, they quit, naturally; if they don't, they increment the lower bound, postulate new solution characteristics and test for the existence of that solution. This continues until a solution is found that matches the postulated characteristics.

Halatsis and Gaitanis do not use the compatibility relation of Grasselli and Montanari. Instead they introduce the notion of "minimum AND/OR dependence sets," and they propose to store only this minimum AND/OR dependence set in the control store. By the nature of these dependence sets, any micro-operation not stored as part of this set is a logic AND or logic OR function of the members of the set. Thus all control signals are either directly available from the control store, or through a single AND or OR gate. To find a minimum AND/OR dependence set, they first write a boolean equation specifying each bit in the control word in terms of all the others. Then they find the prime implicants of each of these equations. They combine these into a new boolean equation which is then reduced to prime implicants. This provides the basis for their solution.

Montangero introduces an extra degree of freedom to the problem: he permits the micro-operations in each word to be varied, within limits, to obtain the minimal width control word. He assumes a maximally parallel microprogram. With this constraint, some micro-operations still can be assigned to one of several micro-instructions. Montangero recognized that the manner in which this assignment is made affects the success of Grasselli and Montanari's coding scheme. He then proposed an exhaustive search strategy for finding the assignment that results in the minimum width microword. As he admits in his paper, exhaustive search would prove to be far too complex in any practical design.

Baer and Koyama propose a branch and bound method for encoding the microword with Grasselli and Montanari's compatibility classes. Although, like the rest, their algorithm is exponential in the worst case, they can stop at any time with a sub-optimal solution. They argue convincingly that they have a reasonably efficient method for a complex problem.

Our own method for optimizing the control word performs the same task as Montangero's. The difference is that we have introduced a different encoding scheme that can handle larger problems by using a heuristic based search algorithm.

2.2 Microprogram Length Reduction

The literature on microcode optimization explores thoroughly the packing of compatible micro-operations into micro-instructions. Davidson and Shriver [DAVI79] provided an extensive update and summary of the work reported in this field. Here we will only describe the problem addressed in the literature along with some details of the associated microprogram model. The interested reader is referred to Davidson and Shriver for more detail.

Mallett[MALL78] described a unifying model of a microprogram. Each micro-operation in his model comprises a six-tuple: name, sources, destinations, required resources, clock phases, and microword fields used. The name is just that: a way to refer to each micro-instruction. Sources and destinations are the registers or memories used in the data transfer. Required resources include the intermediate operators and links required. Clock phase requirements designate the minor cycle within the instruction cycle used by each operation. And the micro-instruction fields are the bits of the microword required to evoke this operation.

Davidson and Shriver describe the many algorithms from the literature for using this model or a similar one to compose micro-operations into micro-instructions. These algorithms work within the bounds of straight line segments, sequences of micro-operations without intervening branches. They assign micro-operations to micro-instructions, attempting to create the fewest possible instructions. This problem has been shown to be NP-complete, but effective heuristic techniques have been reported that reach near-optimal results. Hence Davidson and Shriver say this problem is solved.

2.3 Combining the Two Reductions into One Algorithm

Our goal is to combine word and bit dimension reduction into one algorithm that accounts for the influence of one on the other. We begin with a specification of the design that permits both reordering of microprogram steps and regrouping of micro-operations into word formats. For the former, we adopt a representation from the literature on word optimization: the control flow graph. For the latter, we build a list of all the micro-operations that are used at one time or another. We then iteratively cluster the operations and order the microprogram steps to meet cost/speed constraints in the manner described in the remainder of the paper. In the end we have a completely specified design of a microcontroller.

2.4 Organization of the Paper

We must develop several ideas before we can present the heuristic that is our primary contribution. In the next section, we present a technique for encoding a microprogram word that can be used for both horizontal and vertical microstores. Following that we develop a definition of micro-operations and show how they relate to our control signal clusters. Then we define the attraction weights on which the clustering heuristic is based.

Finally, we present the clustering heuristic, along with the evaluation functions that drive it.

3. <u>AN ENCODING TECHNIQUE FOR MICROWORDS</u>

In this section we define an encoding scheme for microprogram words and discuss the nature of the implementing microcontroller.

We begin with a few pertinent definitions.

> <u>Define</u> <u>a</u> "<u>correct</u>" <u>microcontroller</u> informally as a microcontroller that is capable of providing the control signals necessary to activate the micro-instructions in a control flow graph.

Let C be the set of all signals that must originate in the controller and terminate at the control ports of data path modules. (We will discuss the signals that constitute this set in more detail later.)

> <u>Define</u> <u>a</u> "<u>horizontal</u>" <u>microcontroller</u> as a microcontroller that contains one bit in the control word for each member of C.

A horizontal microcontroller is by definition a correct microcontroller.

> <u>Define</u> "<u>active</u>" <u>control</u> <u>signals</u> as those members of C that either affect the value of data to be latched or effect the latching of data.

Each time a micro-instruction from the control store is executed, only a subset of the data path's control signals are active in most systems. This is what makes it possible to encode control words. In the past, many different encoding techniques have been employed by human designers. One has been automated: the direct encoding of bits in the word. We propose to use a different one: a bit steering field. The function of a bit steering field in a control word is to determine the functions that other bits in the control word must perform. That is, a bit steering field "steers" other control word bits to their proper destinations in the data paths. The HP21MX microword opcode field provides an example of this technique.

Suppose that all the control signals of a particular design could be partitioned into disjoint subsets such that no member of any one subset was ever active in the same micro-instruction as any member of any other subset. Then a bit-steering field could select the active subset, and the control word would need to be only as wide as the largest subset plus the width of the bit-steering field. Decoders outside the microstore could "steer" the control bits to the correct place.

The following partition provides the basis for such a microcontroller. Let W be the set of all micro-instruction words, and let $w_i \in W$ be the set of all c_j which are active in word i.

Define a partition P on C such that for each micro-instruction in W there is a corresponding block of the partition that contains this micro-instruction. And for each pair of micro-instructions that share control signals there is a corresponding block of the partition that contains this pair. Stated formally, this is:

$$\forall w_i \in W, \exists p_i \in P \ni w_i \leq p_i \text{, and}$$

$$\forall w_i, w_j \in W \ni w_i \cap w_j \neq \Phi, \exists p_k \in P \ni w_i \leq p_k, w_j \leq p_k.$$

This means that all micro-instructions that overlap are placed in the same block of the partition. In the remainder of this paper, we will use "p" to denote a block of this partition P.

Define a bit steering decoder as a logic element that passes one bit of data through to one of n different outputs depending on the value of some data selecting inputs. The remaining n-1 outputs are inactive.

Define a bit steering microcontroller as a microcontroller with a memory as wide as the largest p plus a bit steering field, which is log (total number of p's) wide. Each bit in the control word except those in the bit steering field is passed into a bit steering decoder. The directing inputs of the bit steering decoder are fed by the bit steering field of the control word. It has a number of data outputs equal to the total number of p's in the P partition. At each micro-instruction, the bit steering field selects which p contains the current word w and the control signals emanate from the bit steering decoders to the control inputs in the data paths.

Note that the definition of the partition places each microinstruction into a single block of the partition. Therefore only a single p needs to be active at any given time. Thus a bit steering microcontroller is a correct microcontroller.

The basic structure of a bit steering microcontroller (BSM) is shown in figure 1.

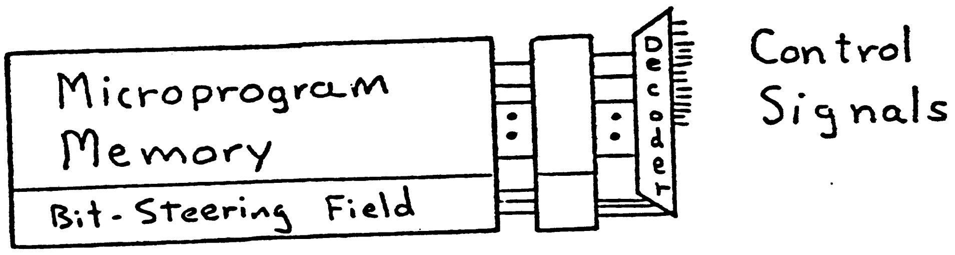


Figure 1. Structure of a Bit Steering Micro-controller.

This is the basis of the encoding scheme that is used throughout this paper. In the degenerate cases this scheme looks like a conventional horizontal or vertical control word. If any control signal could occur at the same time as any other, the bit steering field is 0 bits wide and the partition has one block as wide as a horizontal word. If the activation of signals is mutually exclusive, the bit steering field is Log (n = total control signals) wide and the partition has n blocks with one control signal in each.

3.1 Active and Inactive Signals

We now improve on the BSM by recognizing some properties of control signals and realizing that some of the bit steering decoders are not necessary. To begin, we look more closely at what is meant by "active" and "inactive."

Active and inactive denote opposite states of a control signal, not in a logic sense but rather in a dynamic sense. An active signal may take on any of four values: zero, one, positive edge, or negative edge. It need not be the same each time it is active. An inactive signal, on the other hand, may take on one of only two values: zero or one. The proper value is determined by the function it performs on the module. In some circumstances, to be defined later, the value does not matter. The important distinction is that active signals must be specified from within the microstore because their value is a function of the desired operation. On the other hand, inactive signals can be driven to the proper state by other logic circuits, such as the unselected output of a bit steering decoder. We made use of this in the BSM when we said one of n outputs of a bit steering decoder would be selected (active) and the other n-1 would be inactive.

3.2 Select and Evoke Signals

Taking a closer look at an active signal, we find that it can fall into one of two roles: It can "select" a path or a function without causing any state changes, or it can directly "evoke" a state change in the data paths. This is an important distinction which can be used to show that bit steering decoders are not necessary in all cases.

<u>Define</u> <u>an</u> "<u>evoke</u>" <u>signal</u> <u>as</u> a module control input that is capable of directly changing the contents of that module's internal memory.

<u>Define</u> <u>a</u> "<u>select</u>" <u>signal</u> <u>as</u> a module control input that is incapable of directly changing the contents of that module's internal memory.

Using these two definitions we can show that any microprogram output that performs a select function in all word formats requires no decoder on its output. The memory bit can be wired directly to the one control port in each p to which it is assigned. More specifically, we can show that:

- Select signals from one p cannot affect the outcome of operations controlled by a different p.
- Inactive select signals need not be held constant at any level in order to preserve the state of the data paths.
- Evoke signals from one p <u>can</u> affect the outcome of operations controlled by a different p.
- Inactive evoke signals in all unselected p's must be held constant at a known inactive level to avoid undesired state changes.

- Hence any bit in the control word of a BSM that performs a select role in each p to which it is assigned requires no bit steering decoder for proper operation.

  Define a "Modified Bit Steering Microcontroller" (MBSM) as a BSM that applies this last list item to eliminate decoders.

Finally, using the results listed, we can show that a MBSM is a correct microcontroller.

The MBSM is diagramed in figure 2.

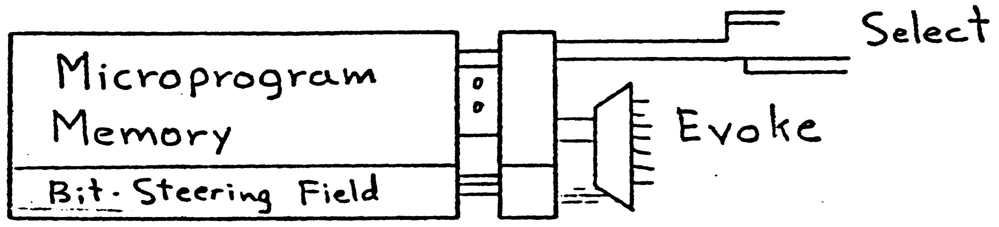


Figure 2. A Modified Bit-Steering Micro-controller. Notice that only a subset of the control signals are fed through the bit steering decoders. The remainder are the select signals. This is the encoding scheme we will use as the focus of our heuristic. To complete the definition of the microcontrol, we will now describe efficient timing signals for this controller.

3.3 Clock Waveforms and Next-State Logic

The clock that the MBSM uses is a simple two phase clock. Phase one evokes a controller state change, enabling new control signals. Phase two evokes data path operations. The controller state change on phase one can either increment the microprogram counter or directly load it. The choice depends on control signals from the control store and status signals from the data paths. The data path operations depend on which "evoke" signals are enabled through the bit steering decoders.

The waveform diagram in figure 3 shows the relationship between the two clock phases and the activity they evoke.

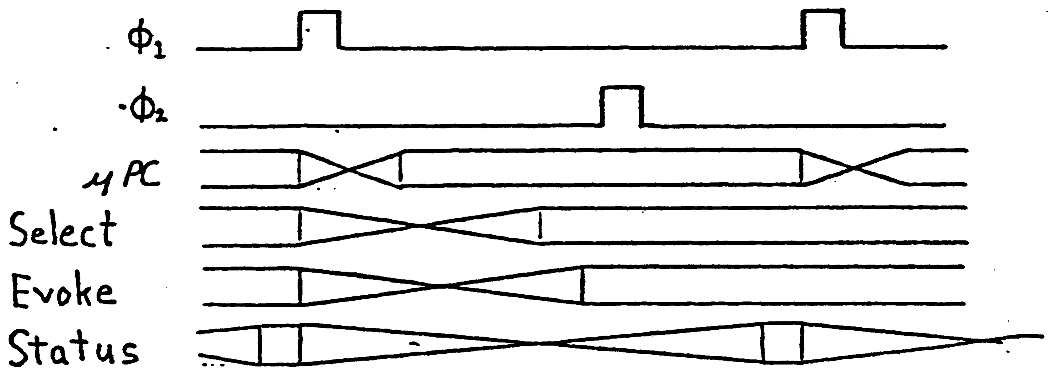


Figure 3. Timing waveforms for a MBSM.

Phase one evokes a change in the microprogram counter. This invalidates the control store outputs for a while. When they become valid, the data path select signals are valid and data begins to settle to the proper values to be latched. Meanwhile, the enabling evoke signals are making their way through the bit steering decoders. The phase two clock pulse comes after the data settles and the evoke enable signals are valid. Note that the active evoke enable signals are a level (a logic ONE) which gets ANDed or NANDed with the phase two clock pulse. AND and NAND gates are used respectively to generate positive and negative going pulses. Inactive evoke-enable signals are logic ZEROs so that they block the phase two pulse from going through the AND or NAND. See figure 4. After the data path evoke signal, the

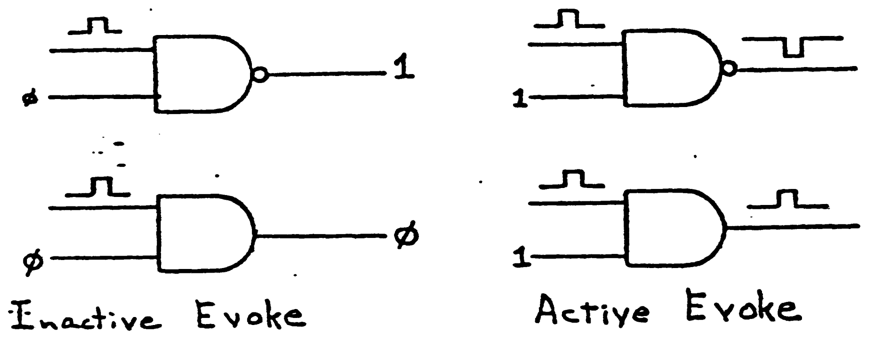


Figure 4. Enabling gates for Evoke Signals.

new data values are permitted to settle so that status signals arising from the newly latched data can be allowed to influence the next state.

Figure 5 shows the same timing diagram for a controller in which the fetch and execute cycles are overlapped.

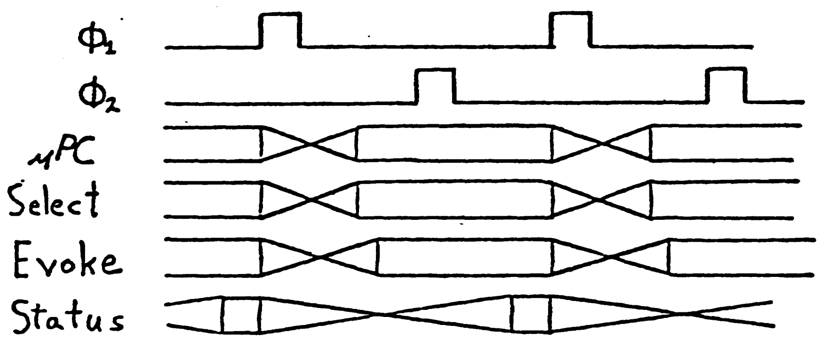


Figure 5. Timing waveforms for overlapped fetch/execute cycles.

This is a common technique used to speed up the execution of microcontrollers. It fetches the next micro-instruction during execution of the current one. Thus execution need not be held up during the memory fetch. Its single disadvantage arises during a branch operation, since an extra step is required between the execution of branch and the time it takes effect. To fill this empty time in some cases a NOOP instruction must be inserted. This wastes space in the microstore, but normally the increased speed justifies this cost. The waveforms are given here to show that the proposed microcontroller covers this case.

4. CODABLE UNITS OF A MICROPROGRAM

In this section we decompose a microprogram and discuss its parts. We describe a way to arrive at a partitioning of the control signals that matches the formal partition defined in the previous section. We show how blocks of this partition can be merged to form new blocks, and discuss the effect this has on the microcontroller.

4.1 Micro-operations and signal clusters

Define a micro-operation (MO) as a state change which is called for by the microprogram, which involves an evoke operation on a single module, which may require select operations on any number of modules, and which is completely specified by a four-tuple: a name, the modules to be used, the operation which each must perform, and the list of MO's that precede it.

This definition of a micro-instruction is similar to many others in the literature on microcode compaction. The most general of these models is in Mallet[MALL78], which defined a micro-operation as a six-tuple (name, sources, destinations, resources, clock phases, microword fields). In our definition we have tried to be more restrictive in some respects, and less restrictive in others. We require a name in the same way as Mallett. We do not require the source list because we assume that source conflicts are accounted for when the control flow graph is built. That is why we require a list of micro-operations that precede each one. We require destinations to be listed, since they are the modules that must perform the evoke operation. Here we are slightly more restrictive: we insist on a single evoke operation per micro-operation. This gives us the maximum degree of flexibility with ordering and encoding. The resource information we require is similar to that required in Mallett's model: a list of the modules used and the operation each performs. We do not require clock phase information as we assume that all data operations occur on a single phase, and all control operations occur on a single separate phase. Finally, we do not require a list of the microword fields that are used because we design that when we derive the word formats. Thus we are more restrictive by limiting the amount of activity that may occur in a single micro-operation and by considering only two-phase clock systems. And we are less restrictive by not limiting the word format to one already specified.

We have said that our heuristic for deriving the word format uses clustering of control signals. By defining micro-operations we have now identified the smallest useful group of control signals: those active during a single micro-operation. The algorithm defined later begins with these groups of control signals as the fields of the control word and clusters the groups into sets that appear together in the same micro-instruction word. But before introducing the algorithm, we must discuss some properties of these groups of signals, especially as they relate to the modules they control, and show that these groups define a P partition on the set of control signals C. Having shown that, we will be able to adopt the modified BSM as our controller.

4.2 Modules and Submodules

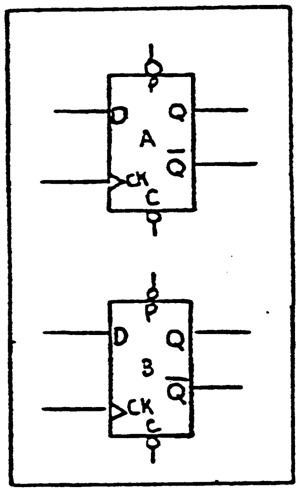
Two or more micro-operations can share control signals; in fact, they could use exactly the same set of control signals, only with different values. For example, the signals used to shift a general purpose shift register left or right are the same: usually a clock and some function select inputs. The only difference from one micro-operation to the other is the value of the function select inputs. Therefore it is appropriate to define the set of signals independent of the operation they perform.

> Define a micro-operation signal set (MS set) as the set of all control signals active during a given micro-operation.

MS sets can be further decomposed into sets of signals according to the module or submodule they control. A module can be broken into submodules if it performs functions that are controlled by disjoint subsets of its control signals.

Define a submodule as the portion of a module whose micro-operations are activated by a subset of the module's control signals, such that all submodules for a given module are disjoint, and only active signals are associated with each micro-operation.

Example. The submodules of a 7474 Dual D-Type positive edge triggered flip-flop with preset and clear are shown in figure 6.

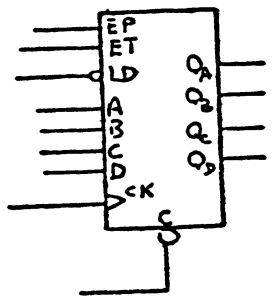


Submodule	signal	operation	active state	type
1	preset A	set A	0	evoke
2	clock A	load A	pos trans	evoke
3	clear A	clear A	0	evoke
4	preset B	set B	0	evoke
5	clock B	load B	pos trans	evoke
6	clear B	clear B	0	evoke

Figure 6. Submodules of a 7474 Dual D-type Flip Flop.

Notice that each signal falls into a separate submodule because each operation of the module requires only one active signal. We acknowledge that the values of the inactive signals in each case must be controlled because they are all evoke signals. See section 3.2.

Example. The submodules of a 74163 synchronous counter with synchronous clear are shown in figure 7. Notice that all signals are in a single submodule because all signals are active during each operation. A 74161 counter with asynchronous clear would have two submodules; the clear signal would be separated out in a submodule of its own, and it would be an evoke signal.



Submodule	signal	operation	active state	type
1	enable P	count	1	select
	enable T	count	1	select
	load	load	0	select
	clear	clear	0	select
	clock	all above	pos trans	evoke

Figure 7. Submodules of a 74163 synchronous counter with synchronous clear.

The useful feature of modules and submodules is that they embody a level of abstraction for dealing with control signals. By listing the submodules used to complete a given micro-operation, we implicitly identify the MS set required. If we further specify the register transfer level operation to be performed by each listed submodule, and if we keep in a data base the rules for implementing each register transfer operation with specific control signal values, then we have a complete hierarchical description of a micro-operation.

We find it useful to discuss the submodules used in micro-operations, so we give them a name.

<u>Define a micro-operation set (MOPSET) as</u> the set of submodules with active signals in a given micro-operation.

4.3 <u>Minimum signal clusters</u>

MOPSETS need not be disjoint. For example, an ALU whose output feeds more than one destination register will be a member of a MOPSET with each of these registers. Now suppose we merge these sets, and all sets that intersect like them. Disjoint sets of submodules would remain. Each set would comprise one or more MOPSETs and should therefore be capable of implementing one or more micro-operations. We give these sets a formal definition.

<u>Define an OPSET as</u> a set O of submodules such that every member of O is also a member of at least one MOPSET which is contained in O.

If the submodules are partitioned into the maximum number of OPSETs, then no proper subset of any OPSET O can be removed from O without breaking up a MOPSET. Conversely, a set of two or more OPSETs is still an OPSET.

<u>Define OPSET signals</u> as the control signals that belong to the submodules in an OPSET.

These control signals for the maximum OPSET partition are the beginning clusters for the heuristic.

Now we can unify our earlier discussion of the modified bit-steering microcontroller with our current discussion of OPSETs.

> Let the submodules in a system be partitioned into OPSETs; then the OPSET signal sets constitute blocks of a P partition on the set C of control signals.

This is an important result because it permits us to build a modified BSM based on the OPSET signal sets.

We consider first the case in which each microword activates a single micro-operation, and the submodules in the system are partitioned into the maximum number of OPSETs.

4.4 <u>Architecture of a vertical MBSM based on minimum size clusters</u>

Figure 2 shows the general architecture of a MBSM. Here we give more details on its size in the simple case in which each microword activates a single micro-operation.

The width of the micro-instruction word contains two components: the bit steering field and the actual control bits. The control bit field must be as wide as the number of control signals in the largest OPSET signal set. The bit steering field must be wide enough to select which OPSET is active. In the worst case this must be log n, where n is the number of OPSETs. But this can often be improved upon by using a bin-packing algorithm to fit all of the OPSETs into the minimum number of formats of the same size, where the size is the width of the largest OPSET signal

to form a new OPSET to gain the most cost/speed tradeoff advantage.

To start, we assume that the maximum OPSET partition has been made. This means we have partitioned the control flow graph into maximally serial steps. Then we compute attraction weights between each pair of OPSETs by an algorithm described below. We remember the highest and the lowest, assess the effect of each on cost and speed, and let an overseer decide whether to merge the OPSETs or exclude their merger. This process is repeated computing new attraction weights (for the remaining OPSETs if a merger has occurred) each time through the loop until the overseer is satisfied that no other candidate pairs need to be considered.

To describe the details of this process, first we define attraction weights and their computation. Then we discuss the effects on the control flow graph of merging two OPSETs together or of excluding such a merger. Finally we discuss the overseer that directs the decisions to merge or exclude each pair.

5.1 Computation of Attraction Weights

Attraction weights measure the relative advantage that would result from the merger of each pair of OPSETs. As such, they attempt to answer the question: if these two OPSETs were merged, how many micro-instructions in the microprogram would make use of the merger by activating one micro-operation from each of the old OPSETs? If the answer is zero, then the merger would be a waste. The pair with the highest weight, if the measure is a good one,

in [AGER76].

In order to compute the attraction weights, we make the following assumption: it is equally likely that any given micro-operation will be executed during any of the slots in its range. Then the probability that a given micro-operation will fall into any one slot is the inverse of the number of slots in its range. The probability that two micro-operations will fall together into the same slot is the conditional probability that the second will fall there given that the first is there already. This is the product of the individual probabilities that they will fall into this slot. For the purpose of computing attraction weights, these probabilities are attached to the OPSETs associated with the micro-operations rather than to the micro-operations themselves.

The attraction weights for a given pair of OPSETs is computed as the sum over all segments of the individual probabilities that these two OPSETs will be used together in the same slot given the random assignment assumption. Notice that this accounts for all uses of the OPSETs from the beginning to the end of the microprogram. It is not restricted to pairing of micro-operations. There may be several micro-operations associated with each OPSET. Sections of the microprogram that are weighted more heavily than another can have their attraction weights multiplied by the weighting factor. Thus the attraction weights in the weighted section will have a heavier bearing on the selection of the OPSET pair to be considered for merging.

5.2 A simple example

Consider the control flow graph in figure 8.

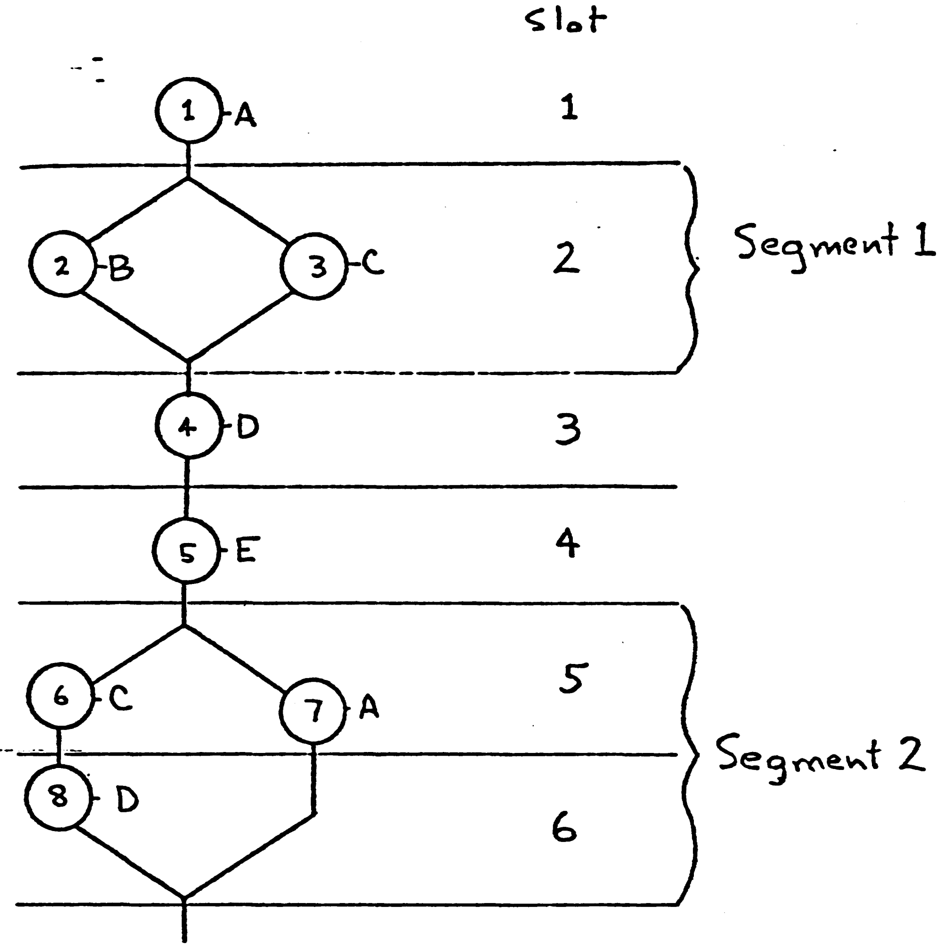


Figure 8. Example control flow graph

It refers to a system that consists of five OPSETs: A, B, C, D, and E. Each node in the control graph shows which OPSET it uses. The critical path through this control graph, six steps long, is shown next to the control graph. There are two segments: the first comprises nodes 2 and 3, the second comprises nodes 6, 7 and 8. Computation of attraction weights proceeds as follows. In the first segment, we note that nodes 2 and 3 each have a range of one. The probability that node 2 will fall in slot 2 is one; similarly for node 3. Thus the probability that nodes 2 and 3, and therefore OPSET's B and C, will fall into the same slot in a minimal length microprogram is one. This is the attraction weight between OPSETs B and C for this segment. In the next segment, we see that node 7 has a range of 2. It therefore has an equal probability, 0.5, of falling into slot 5 or 6. Nodes 6 and 8 each have a range of one and a probability of one that they will fall into slots 5 and 6 respectively. Therefore the A - C attraction weight is 0.5, and the A - D attraction weight is 0.5. The "strongest" attraction weight in this example is between B and C. We infer from this that merging these two into one will have a desirable effect, and we can see that this is true. If they are merged, nodes 2 and 3 can be executed together. If they are not merged, two steps are needed.

The user could have chosen to give the second segment a weighting factor to increase its importance relative to the other segment. A weighting factor of 5.0 would increase the A - C and A - D attraction weights to 2.5 each. Note, too, that if node 2 used OPSET A instead of B, then the A - C attraction weight would be the sum of the weight from each segment.

The object of computing attraction weights is to select an OPSET pair to be merged. In the example, we would have chosen to merge B with C first. We now consider the effect of merging or separating two OPSETs in general.

5.3 The effect of merging OPSETs

When two OPSETs are merged into one it is necessary to reconsider the placement of micro-operations within segments. Whenever two MOPSETs belonging to the same OPSET can potentially fall into the same slot they should be so fixed. If together they can fall into a range of slots then their range should be modified to be equal. If one can also fall into a slot outside the other's range, then this potential should be eliminated by reducing the range of the first one. Naturally when the range of one micro-operation on one leg of a parallel fork is changed this affects the ranges of the preceding and succeeding nodes as well. Thus the ranges of all nodes in all segments must be checked for consistency after merging two OPSETs.

Merging two OPSETs and modifying the micro-operation ranges invalidates the attraction weights just computed. Micro-operations that could fall into the same slot before range modification might be completely separated afterwards. Their OPSETs will therefore no longer be attracted to one another. On the other hand by reducing the ranges of nodes we are increasing the likelihood that they will fall in any of the slots that remain within their range. Thus some attraction weights will need to be increased. Furthermore all attraction weights between the remaining OPSETs and the new OPSET will need to be computed. For all these reasons it is essential to recompute the attraction weights after each two OPSETs are merged. A new candidate pair can then be selected.

In more general terms, the merger of two OPSETs restricts some of the parallelism potential in the control graph. It also can increase the minimum width of the control word in the MBSM. Recall that the minimum width of the control word is the sum of the largest OPSET signal set plus log (number of formats). If the new OPSET is larger than all existing ones, then it forms the basis of the control word width.

5.4 The effect of excluding the merger of two OPSETs

The overseer which we will discuss in an upcoming section may decide to exclude two OPSETs from being merged. It would normally make this decision because the resultant new OPSET would contain too many submodules, or too many signals or both. Merging them would increase the microstore width and the control store cost. But deciding not to merge them also restricts the ranges of micro-operations.

It is possible for a merger exclusion of this nature to necessitate an additional slot in the critical path of some segments. If two parallel micro-operations each have a range of one slot, and if they each use one of the excluded OPSETs, then an extra step will be necessary. The slot that is within their range can be used to execute only one of them. The other one will require a new slot before or after the existing one, further serializing the microprogram.

In more general terms, excluding two OPSETs from merger keeps the width of the control word down, but can create additional steps in the microprogram. An overseer might decide to do this if merging two OPSETs causes an unacceptable length for the control word. It might also simply want to select for exclusion the OPSET pair that would cause the least additional number of words. The pair thus selected ought to have the least interaction of all the existing OPSET pairs. Thus it should have the lowest attraction weight.

5.5 Summary of the attraction weight heuristic

If we are building a modified bit steering microcontroller then we can manipulate the trade-off between word width and microprogram parallelism by manipulating the OPSET partition. In the attraction weight heuristic we begin with the maximal OPSET partition, assuming that each OPSET can activate one micro-operation at a time. We then compute attraction weights as defined above. Based on these an overseer can select a candidate OPSET pair to be merged together or excluded from merger. The effects of this decision can then be accounted for, new attraction weights can be computed and the loop can be repeated. An overseer can stop the loop at any time.

5.6 Structure of an overseer module

The overseer module requires cost and speed evaluations to decide which direction to push the design in. It has the tools at its disposal to increase the word width to keep the number of microwords to a minimum, or to increase the number of microwords to keep the control word width to a minimum. These tools are the attraction weights. But four other evaluations are also needed:

- The cost of the system as it is currently represented
- The incremental cost of merging or excluding
- The speed of the system as it is currently represented
- The incremental speed change resulting from a merger or exclusion

In addition, the overseer needs to know the user's goals and constraints. These may be expressed as limits on the control word width or limits on the total number of words to be used for a given segment of the program, for example.

Each time a pair of OPSETs is selected as candidates to be merged, the overseer module looks at all of the evaluations above and makes a decision to merge or to exclude this pair. To do this it uses the concept of a preferred direction and hard constraints. It will always choose to push the design in the (user specified) preferred direction unless doing so would violate some (also user specified) hard constraint.

In the next section we discuss briefly the computer implementation of the algorithm described above and suggest how the research will proceed.

6. IMPLEMENTATION AND PLANNED RESEARCH

We constructed a computer implementation of our algorithm to test its performance on some examples. The program was written over a period of about one year by one full-time graduate student and another part-time graduate student. It is implemented in BLISS on a PDP-10. The code that implements the part described in this paper consists of about 2400 lines of code and comments. It runs on the data base of the RT-CAD project at Carnegie-Mellon University. We have run a very simple example through the program (a design of the Mark1 Computer) with favorable results. The heuristic optimizer was able to build microprograms with dimensions ranging from 11 bits by 41 words to 28 bits by 32 words.

Our future plans call for more examples and more documentation. We plan to process at least one example that will allow comparison with an existing microprogrammed architecture. Other examples will be chosen to determine the limits of the technique, and to collect some run-time statistics. Having done this we plan to write a formal specification of the technique, with a more complete treatment of the overall structure of the control, in the form of a PhD Thesis.

7. CONCLUSION

We have described a method for computerizing the optimization of a microprogram memory in two dimensions. We have shown the correspondence of the technique to a simple microcontrol structure with a simple two-phase clock. And we have introduced a heuristic that drives the two dimensional optimization. This technique could have an important impact if reduction to hardware from behavioral descriptions ever becomes a common design method. The examples we plan to do next should certify the feasibility of the technique for large scale designs.

ACKNOWLEDGEMENT

The author wishes to thank Dr. Alice Parker, Assistant Professor of Electrical Engineering, Carnegie-Mellon University, for her help in developing the ideas in this paper.

REFERENCES

[AGER76] T. Agerwala, "Microprogram Optimization: A Survey," IEEE Trans. Comp., vol. C-25, No. 10, Oct. 1976, pp.962-973.

[BAER79] J. Baer and B. Koyama, "On the Minimization of the Width of the Control Memory of Microprogrammed Processors," IEEE Trans. Comp., Vol C-28, No. 4, April 1979, pp. 310-316.

[DAS73] S. R. Das, D. K. Banerji, and A. Chattopadhyay, "On control memory minimization in microprotrammed digital computers," IEEE Trans. Comp. vol. C-22, No. 9, Sept 1973, pp.845-848.

[GRAS70] A. Grasselli and U. Montanari, "On the minimization of read-only memories in microprogrammed digital computers," IEEE Trans. Comp., Nov 1970, pp 1111-1114.

[HALA78] C. Halatsis and N. Gaitanis, "On the Minimization of the Control Store in Microprogrammed Computers," IEEE Trans. Comp. Vol. C-27, No. 12, Dec 1978, pp. 1189-1192.

[JAYA76] T. Jayasri and D. Basu, "An Approach to Organizing Microinstructions which Minimizes the Width of Control Store Words," IEEE Trans. Comp. Vol C-25, No. 5, May 1976, pp. 514-521.

[MALL78] P. Mallett, "Methods of Compacting Microprograms," PhD Dissertation, Univ of Southwestern Louisiana, Dec 1978

[MONT74] C. Montangero, "An Approach to the Optimal Specification of Read-Only Memories in Microprogrammed Digital Computers," IEEE Trans. Comp., Vol C-23, No. 4, Apr 1974, pp. 375-389.

[ROBE79] E. Robertson, "Microcode Bit Optimization is NP Complete," IEEE Trans. Comp., Vol. C-28, No. 4, Apr 1979, pp. 316-319.

[SCHW68] S. J. Schwartz, "An Algorithm for Minimizing Read Only Memories for Machine Control," Proc. 10th Annu. IEEE Symp. Switching and Automata Theory, pp. 28-33, 1968.

[DAVI79] S. Davidson and B. Shriver, "Firmware Engineering: An Extensive Update," Technical Report, Computer Science Department, Univ of Southwestern Louisiana, Dec 1979.