A System for Microcode Reduction

Reinhard Rauscher
University of Hamburg
Department of Computer Science
rauscher@informatik.uni-hamburg.de

Manfred Koegst
Fraunhofer-Institut für Integrierte Schaltungen
EAS Dresden
koegst@eas.iis.fhg.de

Abstract

The design flow in the development of microcontrolled devices deals with a large number of design decisions which subsequently influence the realization costs. Some of these decisions are arbitrary choices between design alternatives with hardly any impact on the hardware structure. An example for such design alternatives is the ordering of data inputs at a multiplexor, which can be permutated without any costs in the data path. But such a reordering changes the control-codes for the multiplexor and may possibly allow a reduction of the size of the microcode. This paper presents an algorithm for microcode optimization which uses recoding of arbitrarily coded fields of the microinstruction. A set of design examples is used to demonstrate improvements achieved with our approach.

1 Introduction

Although recently a trend towards RISC-architectures has been observed, the last decades have seen an ongoing interest in the field of microprogramming. So, there still exists the need for microprogrammed architectures and dedicated tools to support their design (see e.g. [6]).

This is an important result of the work of Paulin [14]. Paulin reported that a lot of design efforts have been made in microcode generation and also in microcode optimization. In addition, our design experience also proves the importance of this task.

Most of the single chip microprocessors/microcontrollers are microprogrammed [4], especially special purpose chips (e.g. for interfaces [17]) are best suited for the concept of microprogramming.

Microcode optimization (see also [1]) deals with the following issues:

- Decrease of the chip area used for the microprogram storage (RAM, ROM, PLA)
- Reduce of access time of the microcode storage (e.g. access time depends on the size of the ROM).
- Avoid problems which occur within the used CAD-tool (in our applications most often the SOLO 1400 system [5]). These problems occur not only due to bugs in the CAD tool, but also due to restrictions of the macro cell generators. E.g. in SOLO 1400 we had to restrict the width of the microinstruction word to at most 32 bits because of restrictions within the design tool.

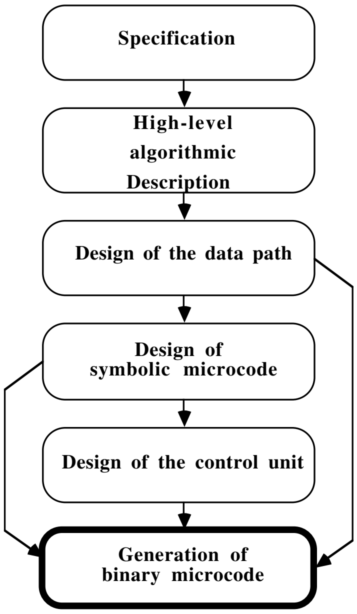

Figure 1: Design flow

The need of an additional microcode optimization tool has been seen from several practical designs at the University of Hamburg (e.g. [9], [11], [12], [13]). The approach considered in this paper is due to the usual design flow.

This design flow is sketched in Figure 1. Starting with a specification, the data path is designed along with so called *symbolic microprogram* controlling the data path. This program can be considered to be an algorithmic description on the register-transfer level. It is used as a specification for the design of the control unit.

Thus, as a first milestone there exists a description of the data path and a symbolic microprogram. Lateron, the control unit will be developed. Finally, the binary microcode will be generated and (if necessary) optimized.

Our approach tries to revise early design decisions made e.g. while designing the data path. An example for this is the ordering of the data inputs at multiplexors.

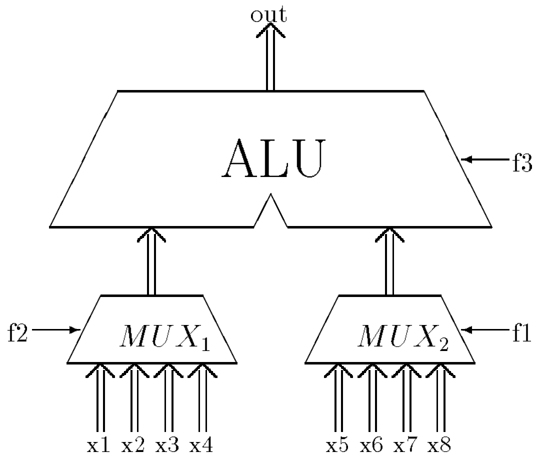

Figure 2: Part of a data path

f1	f2	f3	f4	...
12	34	5678	9	
01	00	0000	1	
00	**	1111	1	
**	**	1001	1	
10	01	000*	0	

Figure 3: Example for the contents of the ROM

2 Basic Approach for Microcode Width Minimization

The basic approach of this work makes use of the observation that the generated microcode often contains a lot of *don't cares*. Additionally, most high-level-synthesis systems generate horizontal microprogram formats.

So, a trivial approach would be to examine whether two or more columns of the microprogram ROM can be merged. E.g., in Figure 3 the columns 6 and 7 are identical. Additionally, the columns numbered 4 and 8 can be merged. These optimizations can be done without any efforts for additional hardware.

Further optimizations could be for example to realize column 9 by "OR"-connecting columns 2 and 5. These optimizations suffer from the fact, that additional hardware (in the example above an OR-gate) is needed and the access time to the microprogram word will increase ([18]).

The new idea in our approach is to change the values of some columns to allow additional optimizations as described above.

If some bits of the microprogram word are exclusively used for controlling multiplexors, their values can be arbitrarily exchanged, if the multiplexor inputs are rearranged correspondingly. Such bits are called *freely exchangeable*. So, if the value table of the values of $f2$ in Figure 3 is changed as described in Figure 4, the values of $f1$ and $f2$ are identical. So, one of the fields can be eliminated completely without any additional hardware.

f2_old	f2_new
00	01
01	10
10	00
11	11

Figure 4: Changes of the value table of $f2$

It should be remarked that exchanging the value table for $f2$ results *only* in a permutation of the data inputs at multiplexor MUX_1 in Figure 2. This local exchange in the data path can be determined directly from the table above. For example, the first line of Figure 4 indicates, that the first data input (x_1) has to be exchanged with the second input (x_2), and so on.

2.1 Basic definitions

A binary microprogram (MP) itself can be considered as a $n \times m$-matrix C with components $c_i^j \in \{0, 1, \star\}$.

A row c_i (output bits) of M is called a *(microprogram) word (W)*.

A column c^j of M is called *freely exchangeable* or *free*, if it is used for controlling of multiplexors only. For example, if it is assumed that the fields $f1, f2, f3$ in Figure 3 are used only for controlling the three components of the data path sketched in Figure 2, at least the bits contained in $f1$ and $f2$ are freely exchangeable.

The set of free columns is abbreviated by using the symbol FC. The bits, which are not free, are called *fixed*. The corresponding set is described by GC.

This classification into both sets FC and GC can be performed using a mapping *kla*.

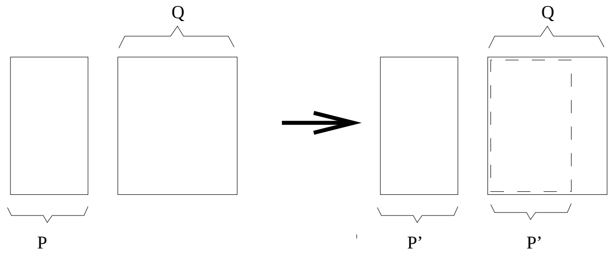

Figure 5: Desired mapping of P and Q into P' and Q'

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	$\star$	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	0	0
$\star$	1	0	1	0	0	1	0	0	0	1	0	1	0	0	$\star$	$\star$	1	$\star$	0
0	0	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	1	1
1	$\star$	0	1	1	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0
$\star$	0	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	1	0
0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	0	1	1	0	0
1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	1	0	1	1	1

Figure 6: Example microcode C

In Figure 6 an example microcode is given. After a first analysis of the data path the Figure 7 results. In Figure 7 describes the multiplexors controlled by the microinstruction word. The second row describes the correspondence to the different columns of the original microcode. In the third row the classification is described. As an example, bit 11 is used for controlling the multiplexor *MUX3* and is part of FC.

It should be noticed, that in this example the bit number 13 is used twice. On one hand for controlling *MUX4*, on the other hand for controlling *MUX6*.

Using a set of indices $V(P) = \{k_1, ..., k_{mp}\}$ and $V(Q) = \{l_1, ..., l_{mq}\}$ subsets of the columns of MP can be described. These subsets are denotes as *fields* $P = (p_i^j)$ or $Q = (q_i^j)$, respectively.

A field FF is called *free*, if the following conditions hold:

1. FF contains only freely exchangeable columns.
2. There exists no multiplexor, which is controlled by a subset X of columns with: $\phi \subset X \subset FF$.

As an example, the columns 14 and 15 are forming a free field, while 13, 14, and 15 is not a free field.

The value of column i of field P is denoted by
$p_i = (p_i^1, p_i^2, ..., p_i^{mp})$.

In considering the main goal, to reduce the width of the microprogram storage, we can define a subgoal. Such a subgoal is to determine for each pair of fields (P, Q) with $mp \leq mq$, whether there exists a mapping *map*, which changes the values of P and Q to values P' and Q', so that P' is a part of Q'. This subgoal is illustrated in Figure 5.

In Figure 9 the resulting microcode after performing all optimization steps is illustrated.

For the determination of such a mapping *map* first the set $R(P, Q) = \{(p_i, q_i), 1 \leq i \leq n\}$, with $p_i \in P$, $q_i \in Q$ is determined.

For two given elements $r_i = (r_i^1, ..., r_i^{mp+mq})$ and $r_k = (r_k^1, ..., r_k^{mp+mq})$ from $R(P, Q)$ a partial order $\leq$ can be defined as follows:

$r_i \leq r_k \leftrightarrow \forall j \in \{1, .., mp + mq\} \rightarrow r_k^j \in \{\star, r_i^j\})$
Based on this, a compatibility relation can be defined:

$r_i \simeq r_k \leftrightarrow \exists r \in \{0, 1, \star\}^{mp+mq} : r \leq r_i \wedge r \leq r_k$.

A set of R(P,Q) is called *compatibility class* (CC), if the following conditions hold:

- $\phi \subset CC \subseteq R(P, Q)$
- $\forall r_i, r_k \in CC : r_i \simeq r_k$

The set of all compatibility classes for two given fields P and Q is called $SVK(P, Q)$, if:

- each $x \in R(P, Q)$ concerns to one unique class of $SVK(P, Q)$.

As an example considering Figure 7 in case of $V(P) = \{1\}$ and $V(Q) = \{2\}$ $R(P, Q) = \{0\star, 11, \star 1, 00, 1\star, \star 0\}$.
There exist the following compatibility classes:
$VK1 = \{0\star, 00, \star 0\}$, $VK2 = \{\star 1, 11, 1\star\}$,
$VK3 = \{0\star, \star 1\}$, $VK4 = \{1\star, \star 0\}$

The classes $VK1$ and $VK2$ build a complete, disjunctive $SVK(P, Q)$.

Each compatibility class VK can be represented by one unique *representant rep(VK)* of the classes follows:

- $\forall x \in VK : rep(VK) \leq x$
- $\neg \exists y \neq rep(VK) : rep(VK) \leq y$ and
  $\forall x \in VK : y \leq x$

For a set of compatibility classes for two given fields P and Q the set of representants is called $SRE(P, Q)$.
As an example the representant of
$VK = \{01\star\star, \star 110\}$ is 0110.

For the pair of fields (P,Q) in Figure 6 with $V(P) = \{1\}$ and $V(Q) = \{2\}$ $R(P, Q)$ can be determined to be:
$R(P, Q) = \{0\star, 11, \star 1, 00, 1\star, \star 0\}$ and
$SVK(P, Q) = \{VK1, VK2\}$ with
$VK1 = \{0\star, 00, \star 0\}$ and $VK2 = \{\star 1, 11, 1\star\}$.

The representants of $VK1$ and $VK2$ are *00* and *11*, respectively. So, $SRE(P, Q) = \{00, 11\}$.

Obviously, the cardinality of SRE describes the amount of different values of the pair (P,Q).

If there exists a y with : $|SRE| \leq 2^y$, only y bits are sufficient for coding.

Given two fields P and Q. A mapping *corr* is defined by $SRE(P, Q)$ as follows: For each $r =$

							MUX1		MUX2		MUX3		MUX4			MUX5		MUX6			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	13	18	19	20
kla(i)	G	G	G	G	G	G	(F	F)	(F	F)	(F	F)	(F)	(F	F)	(F	F)	(F)	(F	F	F)
	0	$\star$	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1	0	0	1	0
	1	1	1	1	1	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	0
	$\star$	1	0	1	0	0	1	0	0	0	1	0	1	0	0	$\star$	$\star$	1	1	$\star$	0
	0	0	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	1
	1	$\star$	0	1	1	0	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0
	$\star$	0	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1	1	0
	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	0	0
	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	1	0	1	1	1	1

Figure 7: Microprogram MP including freely exchangeable fields and the mapping *kla*

							MUX1		MUX2		MUX3		MUX4			MUX5		MUX6			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	13	18	19	20
ind(i)	1	1	3	4	5	6	3	4	9	10	9	10	13	5	6	16	17	13	16	17	20
kla(i)	G		G	G	G	G			(F	F)			(F)			(F	F)				(F)
	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0
	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0
	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	0
	0	0	1	0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0
	1	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1
	0	0	1	1	1	1	1	1	0	1	0	1	0	1	1	1	0	0	1	0	1
	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1	1	0	1	1
	1	1	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	1	1	1	1

Figure 8: Microcode representation after execution of strategies 1 to 5

$(p, q) \in SRE(P, Q)$: $corr(p) = q$. The domain of $corr$ is denoted by $P^\star$.

This mapping $corr$ is named *r-ambiguous*, if for each p there are at most r different values for $corr(p)$. In case of $r = 1$ the mapping is unique.

Additionally, for each p the *ambiguousity value w(p)* is defined as follows:
$w(p) = |\{q | (p, q) \in SRE(P, Q)\}|$

Obviously, for each p the relation $w(p) \leq r$ holds.

2.2 Approaches for comparing fields

In order for a comparison of two given fields P and Q with width mp and mq a set of approaches has been implemented. Additionally, V(P) and V(Q) have to be disjunct.

2.2.1 Meshing

Premise: $mp = mq = 1$
Condition: $SVK(P, Q) \subseteq \{00, 11\}$.
If condition fulfilled: One column can be erased.

For example, see Figure 8, columns 1 and 2.

2.2.2 Strategy 1

Premise: $mp = mq$, $Q \subset FC$
Condition: $corr$ is a one to one mapping.
If condition fulfilled: The field Q can be erased including only small changes in the data path.

2.2.3 Strategy 2

Premise: $mp = mq$, Q is a free field.
Condition: $corr$ is unique.
If condition fulfilled: The field Q can be erased including only small changes in the data path. The multiplexor structure has to be changed slightly.

Considering the example in Figure 7 the fields specified by $V(P) = \{3, 4\}$ and $V(Q) = \{7, 8\}$ result in $SRE(P, Q) = \{0001, 0110, 1010, 1111\}$. So the mapping $corr$ can be defined as follows:

P	Q
00	01
01	10
10	10
11	11

Obviously, $corr$ is unique, so the condition is fulfilled. The resulting code of Q is substituted by the code of P. This is shown in Figure 8. Within Figure 8 it is indicated in row "kla", that both columns (7 and 8) are not longer used.

2.2.4 Strategy 3

Premise: $mp = mq$, P and Q are free fields.
Condition: $|SRE(P, Q)| \leq 2^{mp}$
If condition fulfilled: Either P or Q can be erased. For

1	3	4	5	6	9	10	13	16	17	20
0	0	0	0	0	0	1	0	0	1	0
1	1	1	1	1	1	0	1	0	0	0
1	0	1	0	0	0	0	1	1	1	0
0	1	0	0	1	1	1	0	1	0	0
1	0	1	1	0	0	0	0	0	0	1
0	1	1	1	1	0	1	0	1	0	1
0	0	0	0	1	0	0	1	0	1	1
1	1	0	1	0	1	0	1	1	1	1

Figure 9: Optimized microcode C after execution of strategies 1 to 5.

both fields the multiplexor structure has to be changed slightly.

Considering the example in Figure 7 the fields specified by $V(P) = \{9, 10\}$ and $V(Q) = \{11, 12\}$ result in $SRE(P, Q) = \{0010, 0111, 0100, 1000\}$.

Obviously, the condition is fulfilled. The resulting code of Q is substituted by the code of P. This is shown in Figure 8. Within Figure 8 it is indicated in row "kla", that both columns (11 and 12) are not longer used.

For improvement of the efficiency of the strategies and also for reduction of computation time the strategies 2 and 3 have been generalized.

2.2.5 Strategy 4

Premise: $mp \leq mq$, Q is a free field.
Condition: $corr$ is 2^{mq-mp}-ambiguous.
If condition fulfilled: From the field Q there can be exactly mp columns erased. The multiplexor structure of Q has to be changed slightly.

Considering the example in Figure 8 the fields specified by $V(P) = \{5, 6\}$ and $V(Q) = \{13, 14, 15\}$ result in the following mapping $corr$:

P	Q
00	000
00	100
01	001
01	111
10	011
10	111
11	101
11	010

From this follow the values of $w(00)...w(11)$:
$w(00) = |\{000, 100\}| = 2,\ \ w(01) = |\{001, 111\}| = 2,$
$w(10) = |\{011, 111\}| = 2,\ \ w(11) = |\{101, 010\}| = 2.$

The resulting code is shown in Figure 8.

2.2.6 Strategy 5

Premise: $mp \leq mq$, P and Q are free fields.
Condition: $\sum_{p \in P^\star} \lfloor w(p) \star 2^{mp-mq} \rfloor$.
If condition fulfilled: From the field Q there can be exactly mp columns erased. The multiplexor structures for both fields P and Q have to be changed slightly.

Considering the example in Figure 8 the fields specified by $V(P) = \{16, 17\}$ and $V(Q) = \{18, 19, 20\}$ result in the following representation of $SRE(P, Q)$ after a well suited substitution of all don't cares in the third row results:

$SRE(P, Q) =$
$\{00000, 00100, 01010, 01100, 10110, 10011, 10111\}$.

From this follow the values of $w(00)...w(11)$:
$w(00) = |\{000, 100\}| = 2,\ w(01) = |\{010, 100\}| = 2,$
$w(10) = |\{110, 011, 111\}| = 3.$

So, $\sum_{p \in P^\star} \lfloor w(p) \star 2^{mp-mq} \rfloor \ = \ \lfloor w(00) \star 2^{-1} \rfloor + \lfloor w(01) \star 2^{-1} \rfloor + \lfloor w(10) \star 2^{-1} \rfloor = 1 + 1 + 2.$

The resulting internal representation of the microcode is shown in Figure 8. The result of all optimization steps is shown in Figure 9.

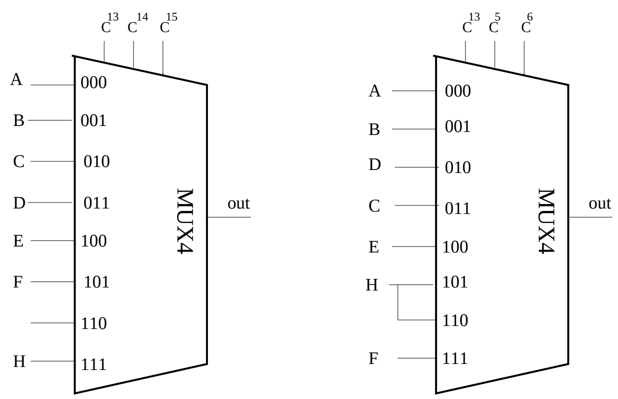

Figure 10: Modification of multiplexor environment

2.3 Modification of the data path

As mentioned above, some local changes of the data may be necessary. These changes consist on permutation of the data inputs of the multiplexers, changing the control inputs, and partially modifying the internal multiplexor structure.

As an example, this is illustrated for the multiplexor MUX4 in Figure 10. On the left side the original multiplexor is sketched, on the right one the multiplexor and

field {13,14,15} of *old(C')*	field {13,14,15} of new(C')	data input
000	000	A
001	001	B
010	011	D
100	100	E
011	010	C
101	111	H
110	111	H
111	101	F

i	ind(i)
13	13
14	5
15	6

Figure 11: Modification of data and control inputs

its environment after modification.

The kind of modification is described in Figure 11. On top of the figure the correspondence between data inputs and the values of the control lines is depicted. On bottom of Figure 11 the modification of the control lines is indicated using the mapping *ind*.

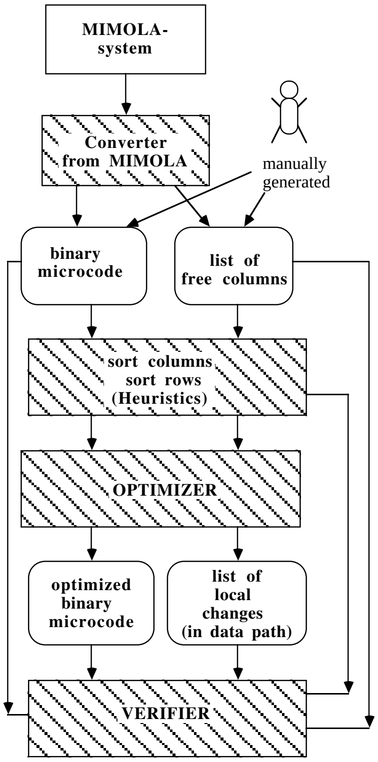

Figure 12: System overview

3 Implementation

In Figure 12 the system environment is illustrated. As the discussion of the usual design flow pointed out our system needs two inputs and two outputs. The inputs are the contents of the microprogram ROM on one hand, and a list of freely exchangeable bits of the microinstruction word. The outputs are similiar, the first part is the new (optimized) binary microprogram. The second part describes the list of local changes within the data path.

Figure 12 also shows the most important components of the system (in dashed rectangles). As mentioned above, alternatively there can be chosen a handcrafted or a automatically generated design. In the latter case a converter into the internal format has to be used. In our first version only a converter from MIMOLA [10] is available.

The next component contains the realization of the set of heuristics to decrease the problem size. After this, the optimization procedure as explained above takes place.

Finally, the automatically regenerated microcode can be verified against the initial microcode. The last component has been very useful in searching for bugs while developing the system. Nevertheless it still makes sense to prove that all optimizations have been correct ([3]).

The system has been implemented on a SUN workstation using C++. Additionally, a DOS-version for 80386 and successors is available.

For evaluation of the efficiency of our approach we analyzed a set of examples. Here we had to distinguish between handcrafted designs and automatically generated designs, e.g. done by a silicon compiler. The handcrafted designs have been: A Scheme-Processor [13], a variant of the DEMOCOM [12], an accelerator for linear programming [11], a microprogrammed version of a delta modulator [15].

For each of the handcrafted designs the designers have been asked to determine which bits are defined as freely exchangeable.

In addition, we used the well known MIMOLA-System [10] for automatical generation of a set of additional examples. After generation of the netlist and the microcode description both outputs of the MIMOLA system are used as inputs to our system. A system component (see Figure 12) converts the microprogram descriptions and the netlists into an internal format. It should be noticed, that MIMOLA does not produce a really horizontal microinstruction format.

Thus, the system component (see Figure 12) generates the list of freely exchangeable bits by analyzing the netlist. It includes only such bits of the microprogram word into the list of freely exchangeable bits if the current bit is used to control multiplexors *only*.

One remarkable result of our activities was, that the sequence of the the strategies performed is of crucial relevance in the results and the computation time.

The disadvantage of an intuitive ordering is, that eventually a few columns are determined to be identical to others, while they also could be used for one of the strategies performed later (meshing fields). Starting with the strategies could result in a time wasting procedure. A comparison for different orders is given in Figure 13. In this figure for the example "synth" five different orderings are compared. It can be seen, that the time used and the results differ!

A compromise (in consuming time and efficiency of the whole procedure we determined, that the following sequence would be best suited:

- Strategy 2
- Strategy 3
- Strategy 1
- Meshing
- Constants

Considering this sequence of operations we determined the following results:

The results are listed in Figure 14 showing the results for several examples. For the discussion of these examples it should be pointed out, that not only fields with a width of 2 bits have been considered. In an earlier activity (see [16]) this restriction has been made because of the complexity aspects mentioned above.

In the first column an identification for the corresponding design is given.

The second column specifies the problem size (number of rows, number of columns). The third column shows the number of freely exchangeable bits. The next five columns identify the number of bits which can be substituted by a constant (const), which can be meshed with other columns (mesh) and be eliminated by the strategies.

The last three columns describe the summarized number of bits which can be eliminated (sum), the percentage of reduction (red), and the CPU-time on a usual PC (i80386) (in seconds) used for this example.

Now, the examples themselves will be discussed in some detail. The first three examples are standard examples within the MIMOLA-system [10]. The first one realizes a prime factor factorization. The second is the description of a RISC-Processor, and the last one is a description of a PDP8. The next three examples are small academic examples. The first one determines the value of *a mod b*, the second one the faculty of a given integer n.

The examples named "LP" are three versions of a hardware accelerator [11] for an algorithm to solve systems of inequalities. The first example of them is a handcrafted design, while the both remaining examples have been generated using the MIMOLA-System.

The examples named "DEMOC" are examples concerning the DEMOCOM [12], a 16-bit CISC processor designed for educational purposes. Again, the example with the extension "man" is a handcrafted design, while all the others are generated using MIMOLA.

The examples named "PC" refer to a student's project, in which a pocket calculator [8] has been developed using the MIMOLA system. The last four examples are designs for the garbage collection of the SCHEME processor [13], a design realizing most of the features of the I8051, a digital filter [9] and a second version of a delta modulator [15].

The results show that also for quite large examples an impressive reduction of the microcode ROM can be achieved. E.g. considering the average number of bits which can be eliminated using the strategies only, there has been a reduction of the size of microcode of *at least* 10 %.

Further results concerning the set of examples can be found in [2].

This result is only based on the rearrangement of freely exchangeable bits. Using "usual" optimizing strategies (e.g. [7], [18], [1]) the size of microcode can be further reduced.

4 Further activities

First, other heuristics for determination of the fields to be compared will be developed and evaluated using additional examples. Especially, for fields of a width of at least 3 bits there is a crucial need for such heuristics.

Additionally, some efforts will be made in implementing an estimation procedure, which determines which sequence of the different optimization strategies is best suited. This implies an interactive component, in which the user of the system imposes a limit on the CPU time.

From Figure 13 and Figure 14 it can be seen, that in most cases nearly all free columns are eliminated. An improvement of the overall strategy would be not to restrict only multiplexors to be base for free exchangeable columns, but also to consider other components (e.g. control lines of the ALU). Such an improvement could be done using an interactive procedure in which the user himself could widen the set of freely exchangeable columns.

References

[1] M. Andrews: Principles of Firmware Engineering; Pitman Publishing Limited, London, 1981

[2] Chr. Brüschke: Entwurf und Implementierung eines CAD-Tools zur Nachoptimierung von mikroprogrammierten VLSI-Steuerwerksstrukturen, Diploma thesis, University of Hamburg, 1995

[3] W. Carter; W. Joyner; D. Brand: Microprogram Verification Considered Necessary, IBM Thomas Watson Research Center, Report RC 7073, 1978

ordering	problem size	FC	const	mesh	S1	S2	S3	sum	red	time	remarks
S2,S3,S1,Mesh.,Const.	109 × 54	15	1	10	0	12	0	23	43	1533	
S3,S2,S1,Mesh.,Const.	109 × 54	15	1	8	0	6	7	22	42	4141	
S1,S2,S3,Mesh.,Const.	109 × 54	15	0	0	4	0	0	4	8	5788	time out
Const.,Mesh.,S1,S2,S3	109 × 54	15	8	6	0	10	0	24	45	10295	time out
Mesh.,Const.,S1,S2,S3	109 × 54	15	1	12	0	3	0	16	30	20834	time out

Figure 13: Comparison of some orderings

name	problem size	FC	const	mesh	S1	S2	S3	sum	red	time	remarks
ref	60 × 55	9	1	25	0	8	0	34	63	122	
risc	89 × 50	14	1	2	0	14	0	17	35	333	
synth	109 × 54	15	1	10	0	12	0	22	43	1533	
amodb	13 × 55	3	1	42	0	2	0	45	81	21	
faculty	11 × 55	5	2	43	0	4	0	49	89	8	
pdp8.2	91 × 50	13	1	6	0	12	0	19	39	160	
LP.1	708 × 108	8	1	60	0	7	0	68	63	4325	1 free column remaining
LP.2	954 × 108	8	2	37	0	8	0	47	44	5242	
DEMOC.2	323 × 56	7	2	21	0	5	0	28	51	813	
PC.1	652 × 74	26	0	0	0	10	0	10	14	25274	time out
GarbC	123 × 56	15	1	1	0	14	0	16	29	9044	1 free column remaining
I8051	843 × 64	10	1	15	0	10	0	26	41	175	
filter	48 × 56	6	0	26	0	4	0	30	53	104	
delta m.	171 × 51	5	1	2	0	2	0	5	10	892	3 free columns remaining

Figure 14: Results

[4] G. Chroust: Mikroprogrammierung und Rechnerentwurf; Oldenbourg Verlag, München, 1989

[5] European Silicon Structures Ltd., Solo 1400 Release 3.1 Reference Manual, Berkshire, 1991

[6] M. McFarland, A. Parker, R. Camposano: The High-Level Synthesis of Digital Systems; Proc. of the IEEE, Vol. 78, 1990, pp 301-318

[7] G. Goosens, J. Rabaey, J. Vandevalle, H. De Man: An Efficient Microcode-Compiler for Custom DSP-Processors; Proc. IEEE IACCAD Conf., Santa Clara, Nov. 1987

[8] N. Hendrich, K. Lagemann, A. Maeder: Teaching High-Level Design; Proc. 4th EUROCHIP Workshop, Toledo, Spain, Sept. 1993, 48-53

[9] N. Hendrich, J. Lohse, R. Rauscher: Silicon Compilation and Rapid Prototyping of Microprogrammed VLSI-Circuits with MIMOLA and SOLO 1400; Proc. Euromicro 92, Paris, Sept.1992, 287-294

[10] R. Jöhnk, P. Marwedel: MIMOLA Reference Manual V. 3.45, Report No. 8902, University of Kiel, 1989

[11] A. Klindworth; B. Schütz: A VLSI-Chip-Set for a Hardware-Accelerator for the Simplex-Method; Proc. 5th Ann. IEEE Intern. ASIC Conference, Rochester, NY, Sept. 1992, pp. 553-559

[12] K. Lagemann: Rechnerstrukturen, Springer Verlag, 1987

[13] J. Lohse: Konzeption, Entwurf und Realisierung eines Prozessors für die funktionale Programmiersprache SCHEME als VLSI- Chip in Standard- und Makrozellentechnik, Ph.-D. thesis, University of Hamburg, 1990

[14] P. Paulin: Design Tool Requirements for the Nineties: An Industrial Perspective, Proc. Sixth Intern. Workshop on High-Level Synthesis, 1992, Laguna Niguel, CA, pp. 1-26

[15] R. Rauscher: Entwurf und Implementierung eines ICs zur krümmungsadaptiven Deltamodulation, University of Hamburg, internal Report Nr. 210, Sept. 1992

[16] R. Rauscher: A new Approach in Microcode Optimization, Proc. EUROMICRO 94, Liverpool, Sept. 1994, pp 27-34

[17] K. Schmidt; W. Wildegger; F. Pätzold; G. Schwarz: Mikroprogrammierbare Schnittstellen, Leitfäden der angewandten Informatik, Teubner, Stuttgart, 1984

[18] G. Zeuner: Entwicklung, Implementierung und Untersuchung eines Algorithmus zur Wortlängenreduktion von horizontalen Mikroprogrammen unter technischen Randbedingungen, Diploma Thesis, Univ. Hamburg, 1988