State assignment of finite state machines using a genetic algorithm

A.E.A. Almaini
J.F. Miller
P. Thomson
S. Billina

Indexing terms: State assignment, Finite state machines, Genetic algorithm

Abstract: The use of genetic algorithms for the generation of optimal state assignments for synchronous finite state machines (FSM) is proposed. Results are presented to show that, in all examples attempted, the resulting state assignments are better than or at least as good as those produced by SPECTRAL, NOVA and MUSTANG and also closed partition assignments. On average, the genetic algorithm produced assignments with 33% less logic than the best produced by other algorithms.

1 Introduction

Attempts at solving the state assignment problem spreads over five decades. Amongst the early attempts are those due to Armstrong [1], Dolotta and McCluskey [2] and the decomposition technique due to Hartmanis and Stearns [3, 4]. The state assignment refers to the allocation of unique binary codes to the states of sequential switching circuits. The discussion is limited to the state assignment of synchronous sequential machines that employ minimum number of state variables. It is known that all valid assignment codes produce workable circuits. The problem is that the resulting circuits may vary vastly in terms of their complexity, and there is no known method of predicting the optimum assignment for the states. Exhaustive search is only possible for relatively small state machines (e.g eight states). The partitioning technique developed by Hartmanis and Stearns [3, 4] is arguably the most systematic study of the subject. One of its drawbacks is that not all state machines have well behaved closed partitions.

Recent work has tended to look for assignments that can be easily automated and give good though not necessarily optimum results. Some of these are now incorporated into commercial ECAD tools and may be targeted at two levels [5] or multilevel [6] implementations.

This paper describes a technique which tackles the problem in an alternative way to the analytical approach of partitioning, or the heuristics of commercially available software. Using a genetic algorithm, all possible assignments for a given problem may be considered as a search space of potential solutions. The genetic algorithm, then, essentially searches this space in a systematic manner whilst employing various methods for the avoidance of settling upon localised optima. It is by this structured search that a good solution for the state assignment may be found.

Paper 1885E (C1), first received 15th August 1994 and in revised form 31st January, 1995

The authors are with the Department of Electrical, Electronic and Computer Engineering, Napier University, 219 Colinton Road, Edinburgh EH14 1DJ, United Kingdom

2 Genetic algorithms

Genetic algorithms (GAs) [7, 8] present a nondeterministic approach to problem solving. The technique originated in Holland's study of adaptation in natural and artificial systems has now been successfully applied to a large number of varied NP-hard problems.

The basic method is to create a population of randomly selected potential solutions to the problem. These solutions are encoded as 'chromosomes' (abstract representations of problem solutions), and each chromosome is subjected to an evaluation function that assigns a 'fitness' depending upon how well the solution it encodes solves the problem at hand.

The chromosomes are recombined by a process called 'crossover'. The rationale behind this is that good solutions will contain good building blocks (partial solutions), the rearrangement of which may produce even better solutions. Further, a 'mutation' process makes random changes to a few randomly selected 'genes' (partial solutions) within chromosomes, preventing premature convergence by maintaining the genetic diversity of the population. The population is then iterated through many generations and each individual's survival depends upon its fitness; hence, the best genes tend to be preserved, and the average fitness of the population should increase from generation to generation.

The mechanism for the selection of parent chromosomes is crucial as this is the driving force behind the gradual increase in average fitness of each generation as the GA evolves. A well known way to achieve this has been the so-called 'roulette wheel selection'. Also known in statistics as the 'Monte Carlo' method, this is simply an algorithm for selecting parent chromosomes in proportion to their fitness. This means that particularly fit chromosomes will be chosen as parent chromosomes a number of times. They will then be able to 'pass on' genetic subfeatures to a number of offspring.

A GA has many parameters to be fine tuned for the problem at hand, and it is still a matter of some debate as to what these settings should be, and how they should be

established. The parameters in question are: (i) the population size, (ii) the crossover rate: the number of chromosomes which will breed in one generation, and (iii) the mutation rate: the percentage of genes in the population that will be altered.

The mechanisms for parent selection and crossover within GAs are very much a research topic in their own right, and many alternatives have been developed [9]. The most effective choice of these mechanisms may be problem related as indeed are the particular values of the GA parameters.

3 Genetic algorithm for the state assignment problem

An FSM with m states requires a minimum of s state variables for the assignment where $s = \lceil \log_2 m \rceil$ and $\lceil g \rceil$ is the smallest integer equal to or greater than g. The number of logically unique [10] assignments for an FSM N is given by

$$N = \frac{(2^s - 1)!}{(2^s - m)!s!}$$

The chromosome representation chosen was a 'pick list'. For example, the 'pick list' 3 4 2 1 1 would map the states 0 1 2 3 4 to the assignment 2 4 1 0 3. This comes about as follows: pick the third state, which gives 2, then remove this from the list. Next, pick the fourth state, which gives 4, then remove. This procedure is then continued for all five states. The reason for this apparently complicated chromosome representation is that it avoids the generation of invalid assignments which may otherwise have arisen by (i) random selection in the initial population and (ii) the processes of crossover and mutation. If the assignment itself were used as the chromosome, the duplication of code assignments to particular states would be unavoidable. This would hamper the functioning of the genetic search because each chromosome would have to be repaired to re-establish their validity as problem solutions.

The genetic algorithm's search space is given by N above. The fitness $f(c_i)$ of a chromosome c_i in a generation having an ESPRESSO multioutput minimised literal count l_i was given by $f(c_i) = \max\{l_j\} - l_i$, where $\max\{l_j\}$ is the maximum literal count found in the generation. For each chromosome, a Berkeley standard PLA file is created, this is then passed to ESPRESSO for minimisation. It is the literal count for this minimisation that forms the fitness for that chromosome. Parent chromosomes were randomly selected with a frequency proportional to their relative fitness (windowed roulette wheel selection). Pairs were then formed ready for crossover in such a way as to avoid the production of duplicate children. Once the new generation is created, elitism is employed to promote the best parent from the previous generation into the new generation. The algorithm used is described in Fig. 1.

From experience the following GA parameters were chosen: population size = 30, breeding rate = 40%, minimum mutation rate = 8%. The mutation rate was variable and increased with each generation if there had been no improvement in the literal count of the best chromosome. It then dropped back to 8% if improvement had occurred. The number of generations over which the GA was run was a maximum of 500. However, it was observed that, for machines with less than nine states, the number of generations required for the GA to

Step 1. Create *num_chromosomes* randomly generated chromosomes c_i;

Step 2. Evaluate the fitness of each chromosome $f(c_i)$;

Step 3. *best_chromosome*=c_i such that $f(c_i)=\max\{f(c_j)\}$;

Step 4. *num_generations=1*;

Step 5. *num_children=(percent_breeding*num_chromosomes)*;

Step 6. while (*max_num_generations* not exceeded) do

```
begin
   select num_children parent chromosomes by windowed roulette wheel;
   apply one-point crossover to create num_children new chromosomes;
   replace num_children parents with new chromosomes (children);
   mutate percent_mutation genes of total genes;
   replace randomly selected chromosome with best_chromosome (elitism);
   evaluate the fitness of new chromosomes f(c_j);
   if (f(c_j) > max{f(c_i)})
      best_chromosome=c_j;
   num_generations=num_generations+1;
end;
```

Fig. 1 *Genetic algorithm for state assignment*

operate effectively was lower at somewhere between 50 and 100.

4 Closed partitions state assignment

The existence of closed partitions on the states of a FSM gives an insight into the structure of the circuit. If the state assignment code is based on closed partitions, the circuit is decomposed into smaller units operating in parallel, serial or as a composite structure. The structure can be predicted by simple inspection of the lattice diagram of the partitions. The reduced dependency between the state variables after decomposition normally results in simpler logic equations. It should be pointed out that not all state machines have closed partitions though, in theory, they can be made to do so by state splitting.

It is felt that a comparison between the performances of the genetic algorithm and those obtained by industry standard algorithm, namely NOVA, MUSTANG and SPECTRAL should give a clear indication of the potential of this algorithm. A comparison with the partitions assignments should give a better understanding of the algorithm and its capability of finding what are known to be good solutions. First, however, a brief review of the relevant aspects of partition theory is given.

A partition π on the states of a sequential machine is called a closed partition with respect to the machine, if for any two states S_i and S_j belonging to the same block of π and any input x, the next states $S_i x$ and $S_j x$ are also in a common block of π. The existence of a partition τ and a closed partition π on the set of states of a sequential machine such that $\pi . \tau = \pi(0)$ is a sufficient condition for the machine to be decomposed into two components operating in series. The predecessor consists of $\lceil \log_2 \#(\pi) \rceil$ state variables to distinguish the blocks of π. These variables are independent of the remaining $\lceil \log_2 \#(\tau) \rceil$ variables assigned to distinguish between the blocks of τ. This can be generalised to a cascade of more than two components [3, 4, 11]. Similarly, if these exist n closed partitions such that $\pi_1, \pi_2, \ldots, \pi_n = \pi(0)$, the machine can be decomposed into n components operating in parallel and independent of each other. If $\pi_1 + \pi_2 + \cdots + \pi_n \neq \pi(I)$, a component may be factored out to produce a composite structure.

5 Results

Due to space limitations, only selected examples are considered in detail. In every case the logic was minimised using the multioutput logic minimiser, ESPRESSO. The codes for the state assignments are given in Table 3. The

Table 1: Comparison of algorithms (including output logic)

KISS2 file	Number of states	Genetic Algorithm	Nova	Mustang	Spectral	Partitioning
bbtas	6	20 / 12 / 7	34 / 22 / 9	28 / 18 / 7	49 / 34 / 14	-
dk14	7	96 / 70 / 55	105 / 77 / 50	131 / 99 / 53	150 / 115 / 68	-
dk15	4	68 / 50 / 33	68 / 50 / 33	87 / 66 / 44	74 / 55 / 33	-
dk16	27	257 / 201 / 104	371 / 297 / 113	423 / 342 / 113	383 / 306 / 115	-
dk27	7	16 / 9 / 6	22 / 14 / 7	22 / 14 / 5	22 / 13 / 7	25 / 15 / 6
dk512	14	43 / 28 / 19	65 / 47 / 25	80 / 60 / 26	69 / 49 / 19	85 / 63 / 26
shiftreg	8	4 / 0 / 0	18 / 10 / 4	10 / 4 / 2	35 / 23 / 12	4 / 0 / 0
tav	4	32 / 22 / 4	35 / 24 / 5	35 / 24 / 5	35 / 24 / 5	35 / 24 / 5

Table 2: Comparison of algorithms (ignoring output logic)

KISS2 file	Number of states	Genetic Algorithm	Nova	Mustang	Spectral	Partitioning
donfile	24	104 / 75 / 32	207 / 160 / 60	236 / 184 / 57	248 / 197 / 79	-
bbtas	6	15 / 9 / 6	31 / 20 / 8	20 / 12 / 7	34 / 22 / 13	-
dk14	7	39 / 26 / 19	78 / 59 / 26	70 / 52 / 17	82 / 63 / 20	-
dk15	4	19 / 12 / 8	21 / 14 / 6	25 / 16 / 7	22 / 15 / 5	-
dk16	27	223 / 176 / 89	279 / 225 / 92	344 / 278 / 92	377 / 304 / 96	-
dk27	7	11 / 5 / 6	19 / 12 / 5	15 / 9 / 3	20 / 12 / 5	19 / 11 / 5
dk512	14	24 / 14 / 11	53 / 38 / 19	71 / 53 / 21	51 / 35 / 14	71 / 52 / 20
modulo12	12	15 / 7 / 5	31 / 19 / 10	38 / 25 / 12	32 / 19 / 9	20 / 11 / 5
shiftreg	8	3 / 0 / 0	18 / 11 / 7	6 / 2 / 1	30 / 20 / 10	3 / 0 / 0
tav	4	2 / 0 / 0	5 / 2 / 1	5 / 2 / 1	5 / 2 / 1	5 / 2 / 1
test	8	11 / 5 / 3	23 / 15 / 7	26 / 17 / 6	27 / 17 / 8	14 / 7 / 4
table1	16	22 / 13 / 7	75 / 55 / 24	65 / 46 / 17	76 / 56 / 25	31 / 19 / 8
table2	9	14 / 7 / 5	37 / 25 / 13	36 / 24 / 10	35 / 23 / 11	19 / 10 / 5
table3	6	19 / 12 / 4	36 / 24 / 13	52 / 38 / 13	51 / 36 / 14	19 / 11 / 5
swma1	8	9 / 4 / 2	17 / 10 / 5	13 / 7 / 3	20 / 12 / 5	13 / 6 / 4
swma2	8	7 / 3 / 2	14 / 8 / 4	10 / 5 / 2	29 / 19 / 7	7 / 3 / 1
swma3	16	67 / 46 / 20	115 / 88 / 41	152 / 88 / 49	141 / 107 / 41	74 / 53 / 17

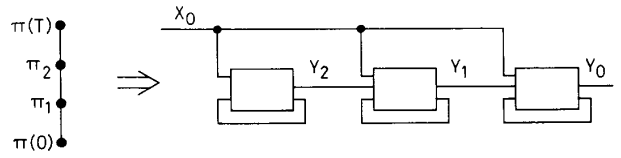
KISS2 files for the examples which are not benchmarks are given in Table 4. These were taken from References 11–13.

The partition assignment, for example, 'dk27' resulted in a predictable serial decomposition as shown in Fig. 2. The GA, which is biased towards minimising the number of literals, generated an assignment which required the smallest number of literals. This bias can be adjusted, if desired, to favour the minimisation of the number of cubes or the number of AND/OR gates. The other algorithms produced comparable assignments.

The example 'test' had three useful closed partitions and resulted in two components in parallel with a predecessor as can be predicted from the lattice diagram in Fig. 3. The parallel components are independent of each other. The GA produced the best result with the partition assignment a close second, both much better than all the others.

The example 'swma2' assignment was based on three partitions of the same block size, one of which is also an input consistent partition [11]. The result is probably the best possible, three components operating in parallel independently of each other one of which is independent of the external primary input. The GA solution is comparable but with more dependence. MUSTANG produced the next best solution followed by NOVA and SPECTRAL. These are given in Fig. 4.

The results shown in Tables 1 and 2 show the performance, in terms of the number of two input gates required, for the various state assignment algorithms considered. The Tables show, for each algorithm, a breakdown of the number of literals and the number of two input AND gates and OR gates, respectively.



$\pi_1 = \overline{2}\ ;\ \overline{1,3};\ \overline{5,6};\ \overline{4,7}$

$\pi_2 = \overline{1,2,5,6}\ ;\ \overline{2,4,7}$

Partition equations:

$$Y_2 = \bar{y}_2 x_0$$

$$Y_1 = \bar{y}_2 y_1 x_0 + y_2 \bar{y}_1 + y_2 \overline{x_0} + \bar{y}_1 \overline{x_0}$$

$$Y_0 = \bar{y}_1 \bar{y}_0 \overline{x_0} + \bar{y}_1 y_0 x_0 + y_2 x_0$$

GA equations:

$$Y_2 = \bar{y}_1 \bar{y}_0 x_0 + \bar{y}_2 y_1 y_0 + \overline{x_0}$$

$$Y_1 = \bar{y}_2 \overline{x_0} + \bar{y}_2 y_1 y_0 + \bar{y}_1$$

$$Y_0 = \bar{y}_1 \bar{y}_0 x_0 + \bar{y}_2 y_1 y_0 + y_2$$

NOVA equations:

$$Y_2 = y_2 y_0 \overline{x_0} + y_2 y_1 \overline{y_0} + \overline{y_1} x_0$$

$$Y_1 = \bar{y}_2 \bar{y}_0 x_0 + \bar{y}_1 \bar{y}_0 + \bar{y}_1 x_0$$

$$Y_0 = \overline{y_1} y_0 x_0 + y_1 \overline{y_0} x_0$$

MUSTANG equations:

$$Y_2 = y_1 x_0$$

$$Y_1 = \bar{y}_2 \bar{y}_1 + \bar{y}_2 y_0 x_0$$

$$Y_0 = y_2 y_0 \overline{x_0} + \bar{y}_2 \bar{y}_1 x_0 + y_1 y_0$$

SPECTRAL equations:

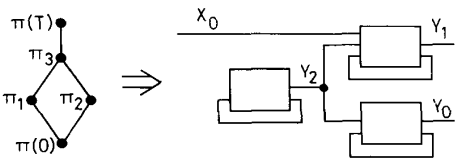
$$Y_2 = y_2 y_1 \overline{x_0} + y_1 y_0 + y_2 y_0$$

$$Y_1 = \overline{y_1} x_0$$

$$Y_0 = y_2 \bar{y}_0 x_0 + \bar{y}_2 \bar{y}_0 \overline{x_0} + \bar{y}_2 y_0 x_0 + y_1 x_0$$

Fig. 2 *Equations for benchmark dk27*

In terms of numbers of literals, the genetic algorithm performed 42% better than NOVA. It should be noted that only the logic for the state variables was considered, as this gives a useful comparison against the partitioning method. Observing the progress of the genetic algorithm, it was noted that a state assignment similar to the one produced by NOVA was found after 50 generations, on average. NOVA generally performed better than MUSTANG and SPECTRAL. When the output logic was considered, the genetic algorithm performed, on average, 33% better than NOVA. On the whole, the partition method and the genetic algorithm produced similar results if good partitions exist. However, if no partitions exist (as some state tables possess no useful closed partitions), the genetic algorithm still performs well as its operation is not dependent upon the existence or non-existence of this property. The CPU time consumed by running the GA is longer than that required by other methods, but this mainly due to the fact that the GA has to communicate with ESPRESSO to ascertain the fitness of chromosomes. This had to be done by creating files as



$\pi_1 = \overline{1,2}\ ;\ \overline{3,4};\ \overline{6,8};\ \overline{5,7}$
$\pi_2 = \overline{1,8}\ ;\ \overline{3,7};\ \overline{4,5};\ \overline{2,6}$
$\pi_3 = \overline{1,2,6,8}\ ;\ \overline{3,4,5,7}$

Partition equations:

$Y_2 = \overline{y_2}$

$Y_1 = \bar{y}_2 \bar{y}_1 \overline{x_0} + y_1 x_0 + y_2 x_0 + y_2 y_1$

$Y_0 = \bar{y}_2 \bar{y}_0 + y_2 y_0$

GA equations:

$Y_2 = \overline{y_2}$

$Y_1 = \overline{y_0} + y_2 x_0$

$Y_0 = \bar{y}_2 \bar{y}_1 x_0 + y_1 \overline{x_0} + y_2 y_1$

NOVA equations:

$Y_2 = \bar{y}_2 \bar{y}_1 x_0 + y_2 y_0 \overline{x_0} + \bar{y}_2 y_1$

$Y_1 = \bar{y}_2 \bar{y}_1 x_0 + y_2 y_0 \overline{x_0} + \bar{y}_1 y_0 \overline{x_0} + y_2 y_1 \bar{y}_0$

$Y_0 = \bar{y}_2 \bar{y}_1 \overline{x_0} + \bar{y}_1 y_0 \overline{x_0} + y_1 y_0 x_0$

MUSTANG equations:

$Y_2 = \overline{y_2}$

$Y_1 = \bar{y}_2 y_1 \bar{y}_0 \overline{x_0} + y_2 y_0 + \overline{y_1} y_0 \overline{x_0}$

$Y_0 = y_2 y_1 \bar{y}_0 x_0 + y_2 y_1 y_0 \overline{x_0} + \bar{y}_1 y_0 x_0 + \bar{y}_1 \bar{y}_0 \overline{x_0} + \bar{y}_2 \bar{y}_1$

SPECTRAL equations:

$Y_2 = \overline{y_2} y_1 \overline{y_0} + \overline{y_1} y_0 + y_2 \overline{y_1}$

$Y_1 = y_1 y_0 \overline{x_0} + y_2 x_0 + \bar{y}_1 \bar{y}_0 x_0 + y_2 \bar{y}_1$

$Y_0 = y_1 y_0 x_0 + y_2 y_0 \overline{x_0} + \bar{y}_2 \bar{y}_0 \overline{x_0} + y_2 \bar{y}_0 x_0$

Fig. 3 *Equations for example 'test'*

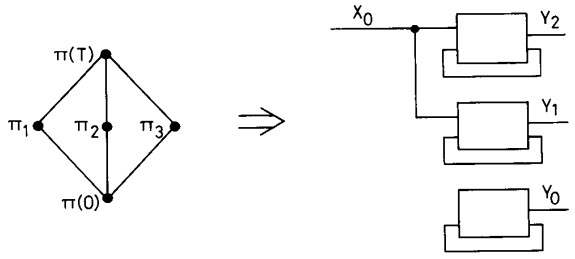
ESPRESSO is not embedded in the GA. Even allowing for this overhead, the times taken to run the GA were not excessive. For example, the time taken to run 'donfile' (with 24 states) over 500 generations was about 30 minutes on an HP-720 workstation. Further, with the improving processing speeds of modern workstations, this is likely to become less significant in the future.

At this point it should be noted that the genetic algorithm is not a random search. To support this, tests were made where 1000 different chromosomes were generated at random for 'donfile'. The GA started with 10 chromosomes and ran over 100 generations. The results have shown that on average, over ten tests, the genetic algorithm produced a 40% better result than the random search. In every test run, the GA found a better assignment than that produced by any random search.

6 Conclusions

A genetic algorithm for finding good assignments for synchronous sequential state machines has been written in C, and described above. Tests as to the effectiveness of this approach to the problem were conducted by comparison of performance against commercially available software when operating upon various widely used benchmarks. A number of smaller examples were also included by way of demonstration.

It may be concluded that the state assignments as found by the genetic algorithm were at least as good, but in most cases better, than those derived by the best available methods. In examples where there is a good structure like 'modulo-12', 'shiftreg', 'test', 'table 1-3' and 'swma3', the GA and the partitioning method produced significantly simpler next-state equations.



$\pi_1 = \overline{1,4,6,8}\ ;\ \overline{2,3,5,7} = \tau_i$

$\pi_2 = \overline{1,2,3,4}\ ;\ \overline{5,6,7,8}$

$\pi_3 = \overline{1,2,7,8}\ ;\ \overline{3,4,5,6}$

Partition equations:

$Y_2 = \overline{y_2} x_0$

$Y_1 = \bar{y}_1 \overline{x_0} + y_1 x_0$

$Y_0 = \overline{y_0}$

GA equations:

$Y_2 = \overline{y_2} x_0 + y_2 \overline{x_0}$

$Y_1 = \overline{y_1}$

$Y_0 = y_0 x_0 + y_2 \overline{x_0}$

NOVA equations:

$Y_2 = \bar{y}_2 \bar{y}_1 \overline{x_0} + y_1 y_0 \overline{x_0} + \bar{y}_0 x_0$

$Y_1 = \overline{y_1}$

$Y_0 = y_2 \bar{y}_1 \overline{x_0} + y_1 y_0 \overline{x_0} + \bar{y}_2 x_0$

MUSTANG equations:

$Y_2 = \overline{y_2} x_0$

$Y_1 = \overline{y_1}$

$Y_0 = y_2 \bar{y}_0 \overline{x_0} + y_0 x_0 + \bar{y}_2 y_0$

SPECTRAL equations:

$Y_2 = y_2 \bar{y}_1 \bar{y}_0 x_0 + \bar{y}_2 y_1 \overline{x_0} + y_1 y_0 x_0 + \bar{y}_2 y_0 x_0$

$Y_1 = y_2 \bar{y}_0 \overline{x_0} + \bar{y}_1 x_0$

$Y_0 = \bar{y}_2 y_1 y_0 x_0 + \bar{y}_1 \bar{y}_0 \overline{x_0} + y_2 \bar{y}_0 + y_2 \bar{y}_1$

Fig. 4 *Equations for example 'swma2'*

Table 3: State assignments for each algorithm

donfile:	GA:	2, 3, 0, 4, 1, 5, 6, 7, 10, 11, 8, 9, 12, 13, 14, 15, 21, 23, 17, 16, 18, 19, 25, 27
	NOVA:	0, 2, 25, 9, 3, 11, 5, 1, 21, 22, 4, 6, 16, 27, 7, 31, 26, 15, 18, 8, 13, 12, 10, 14
	Mu:	6, 7, 14, 22, 10, 12, 2, 18, 3, 11, 19, 15, 13, 4, 5, 21, 20, 29, 0, 1, 16, 8, 17, 9
	Sp:	0, 4, 6, 8, 10, 11, 1, 5, 14, 16, 18, 19, 2, 9, 12, 17, 22, 23, 3, 7, 13, 15, 20, 21
bbtas:	GA: 7, 1, 6, 0, 4, 5	NOVA: 2, 3, 1, 0, 5, 6 Mu: 6, 0, 5, 1, 3, 2 Sp: 0, 1, 2, 3, 4, 5
dk14:	GA: 5, 7, 1, 3, 6, 0, 4	NOVA: 1, 4, 0, 2, 7, 5, 3 Mu: 0, 1, 2, 6, 3, 4, 7 Sp: 0, 5, 1, 4, 2, 3, 6
dk15:	GA: 2, 3, 0, 1	NOVA: 1, 0, 3, 2 Mu: 0, 1, 3, 2 Sp: 0, 1, 2, 3
dk16:	GA:	17, 29, 0, 1, 28, 25, 5, 4, 15, 7, 9, 26, 16, 21, 23, 14, 8, 31, 11, 27, 20, 13, 6, 3, 12, 30, 2
	NOVA:	12, 7, 1, 3, 4, 10, 24, 23, 5, 27, 15, 16, 11, 6, 0, 20, 31, 2, 13, 25, 21, 14, 18, 19, 30, 17, 22
	Mu:	6, 4, 3, 18, 10, 8, 5, 12, 19, 14, 24, 13, 0, 1, 3, 22, 27, 7, 9, 11, 25, 23, 16, 20, 21, 17, 15
	Sp:	0, 5, 1, 8, 9, 10, 11, 6, 7, 2, 3, 4, 21, 22, 19, 12, 13, 16, 17, 18, 14, 15, 23, 24, 25, 20, 26
dk27:	GA: 5, 1, 2, 3, 6, 7, 0	NOVA: 2, 6, 5, 3, 4, 0, 7 Mu: 2, 3, 7, 4, 1, 0, 5 Sp: 0, 3, 5, 2, 4, 1, 6
	Pa: 0, 6, 1, 4, 2, 3, 5	
dk512:	GA:	4, 3, 11, 15, 14, 9, 12, 7, 2, 1, 0, 10, 13, 8 NOVA: 1, 10, 8, 0, 3, 7, 15, 6, 14, 12, 13, 9, 4, 11
	Mu:	9, 11, 1, 0, 4, 3, 5, 2, 14, 10, 12, 6, 7, 8 Sp: 0, 3, 5, 4, 6, 7, 12, 1, 2, 8, 9, 10, 11,13
	Pa:	0, 8, 1, 2, 3, 9, 10, 4, 11, 12, 13, 5, 6, 7
modulo12:	GA:	11, 14, 4, 3, 10, 12, 7, 2, 8, 15, 6, 0 NOVA: 0, 15, 1, 14, 2, 13, 3, 12, 4, 11, 5, 10
	Mu:	0, 1, 9, 8, 2, 3, 4, 5, 7, 15, 6, 14 Sp: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
	Pa:	0, 10, 5, 12, 2, 9, 4, 14, 1, 8, 6, 13
shiftreg:	GA:	0, 1, 6, 2, 7, 3, 4, 5 NOVA: 0, 4, 3, 2, 5, 1, 6, 7 Mu: 0, 2, 6, 4, 1, 3, 7, 5
	Sp:	0, 2, 3, 5, 1, 4, 6, 7 Pa: 0, 1, 2, 3, 4, 5, 6, 7
tav:	GA:	3, 1, 0, 2 NOVA: 0, 3, 1, 2 Mu: 0, 1, 2, 3 Sp: 0, 1, 2, 3 Pa: 0, 2, 1, 3
test:	GA:	0, 2, 6, 7, 5, 3, 4, 1 NOVA: 7, 0, 6, 1, 5, 2, 4, 3
	Mu:	6, 4, 0, 1, 2, 5, 3, 7 Sp: 0, 3, 1, 2, 6, 4, 7, 5
	Pa:	0, 2, 7, 5, 4, 3, 6, 1
table1:	GA:	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 10, 11, 14, 15 NOVA: 15, 4, 10, 13, 0, 3, 14, 1, 8, 7, 9, 6, 11, 5, 2, 12
	Mu:	4, 0, 3, 7, 1, 5, 6, 2, 10, 8, 9, 11, 12, 13, 14, 15 Sp: 0, 3, 6, 9, 10, 7, 4, 0, 14, 12, 13, 15, 8, 11, 2, 5
	Pa:	0, 8, 4, 12, 1, 9, 5, 13, 2, 10, 6, 14, 3, 11, 7, 15
table2:	GA:	4, 11, 8, 5, 7, 9, 15, 12, 13 NOVA: 10, 12, 1, 14, 5, 2, 13, 0, 3
	Mu:	7, 6, 4, 15, 3, 2, 5, 0, 1 Sp: 0, 2, 1, 3, 4, 5, 8, 6, 7
	Pa:	0, 1, 4, 6, 9, 10, 13, 12, 14
table3:	GA: 6, 3, 2, 0, 4, 1	NOVA: 6, 0, 2, 5, 4, 7 Mu: 4, 1, 3, 5, 2, 0 Sp: 0, 4, 1, 2, 5, 3 Pa: 0, 3, 2, 6, 4, 7
swma1:	GA: 2, 5, 0, 7, 1, 6, 4, 3	NOVA: 0, 4, 5, 3, 2, 6, 1, 7 Mu: 0, 7, 2, 6, 1, 3, 5, 4 Sp: 0, 2, 4, 6, 1, 5, 7, 3
	Pa: 2, 0, 3, 4, 6, 7, 1, 5	
swma2:	GA: 7, 6, 2, 4, 5, 0, 3, 1	NOVA: 0, 3, 6, 5, 2, 4, 7, 1 Mu: 0, 3, 2, 1, 7, 4, 6, 5 Sp: 0, 3, 1, 4, 7, 6, 2, 5
	Pa: 0, 1, 3, 2, 7, 6, 5, 4	
swma3:	GA:	6, 15, 14, 10, 8, 0, 2, 4, 12, 7, 9, 11, 3, 1, 5, 13 NOVA: 10, 12, 13, 9, 3, 1, 5, 15, 14, 7, 0, 4, 6, 2, 11, 8
	Mu:	5, 12, 15, 11, 8, 3, 0, 14, 13, 9, 1, 2, 7, 4, 10, 6 Sp: 0, 3, 4, 5, 9, 11, 1, 12, 13, 2, 6, 7, 10, 8, 14, 15
	Pa:	0, 10, 4, 14, 12, 1, 2, 11, 15, 8, 3, 6, 13, 9, 5, 7

Table 4: KISS2 files for 'nonbenchmark' examples

test.kiss2	0 st4 st9 0	1 st2 st4 0	00 st4 st4 0	.s 8	00 st4 st11 0
.i 1	1 st4 st10 0	0 st3 st1 0	01 st4 st3 0	0 st0 st2 0	01 st4 st10 0
.o 1	0 st5 st8 0	1 st3 st5 0	10 st4 st2 0	1 st0 st6 0	10 st4 st12 0
.p 16	1 st5 st11 0	0 st4 st5 0	11 st4 st1 0	0 st1 st3 0	00 st5 st10 0
.s 8	0 st6 st11 0	1 st4 st7 0	00 st5 st3 0	1 st1 st7 0	01 st5 st13 0
0 st0 st2 0	1 st6 st8 0	0 st5 st4 0	01 st5 st5 0	0 st2 st0 0	10 st5 st7 0
1 st0 st3 0	0 st7 st10 0	1 st5 st8 0	10 st5 st1 0	1 st2 st5 0	00 st6 st10 0
0 st1 st3 0	1 st7 st9 0	0 st6 st8 0	11 st5 st0 0	0 st3 st1 0	01 st6 st12 0
1 st1 st2 0	0 st8 st1 0	1 st6 st7 0		1 st3 st4 0	10 st6 st8 0
0 st2 st5 0	1 st8 st4 0	0 st7 st7 0		0 st4 st0 0	00 st7 st5 0
1 st2 st5 0	0 st9 st0 0	1 st7 st6 0	**swma1.kiss2**	1 st4 st3 0	01 st7 st14 0
0 st3 st7 0	1 st9 st5 0	0 st8 st6 0	.i 1	0 st5 st1 0	10 st7 st9 0
1 st3 st5 0	0 st10 st3 0	1 st8 st8 0	.o 1	1 st5 st2 0	00 st8 st6 0
0 st4 st0 0	1 st10 st6 0		.p 16	0 st6 st3 0	01 st8 st15 0
1 st4 st1 0	0 st11 st2 0		.s 8	1 st6 st0 0	10 st8 st9 0
0 st5 st4 0	1 st11 st7 0	**table3.kiss2**	0 st0 st4 0	0 st7 st2 0	00 st9 st2 0
1 st5 st6 0	0 st12 st7 0	.i 2	1 st0 st0 0	1 st7 st1 0	01 st9 st0 0
0 st6 st1 0	1 st12 st2 0	.o 1	0 st1 st7 0		10 st9 st8 0
1 st6 st1 0	0 st13 st6 0	.p 24	1 st1 st0 0		00 st10 st10 0
0 st7 st6 0	1 st13 st3 0	.s 6	0 st2 st5 0	**swma3.kiss2**	01 st10 st4 0
1 st7 st4 0	0 st14 st5 0	00 st0 st0 0	1 st2 st2 0	.i 2	10 st10 st3 0
	1 st14 st0 0	01 st0 st2 0	0 st3 st6 0	.o 1	00 st11 st10 0
	0 st15 st4 0	10 st0 st3 0	1 st3 st4 0	.p 48	01 st11 st3 0
table1.kiss2	1 st15 st1 0	11 st0 st5 0	0 st4 st0 0	.s 16	10 st11 st4 0
.i 1		00 st1 st2 0	1 st4 st4 0	00 st0 st6 0	00 st12 st15 0
.o 1		01 st1 st1 0	0 st5 st2 0	01 st0 st9 0	01 st12 st6 0
.p 32	**table2.kiss2**	10 st1 st5 0	1 st5 st5 0	10 st0 st1 0	10 st12 st4 0
.s 16	.i 1	11 st1 st4 0	0 st6 st3 0	00 st1 st0 0	00 st13 st14 0
0 st0 st7 0	.o 1	00 st2 st0 0	1 st6 st2 0	01 st1 st2 0	01 st13 st5 0
1 st0 st14 0	.p 18	01 st2 st1 0	0 st7 st1 0	10 st1 st9 0	10 st13 st3 0
0 st1 st6 0	.s 9	10 st2 st5 0	1 st7 st5 0	00 st2 st0 0	00 st14 st5 0
1 st1 st15 0	0 st0 st2 0	11 st2 st3 0		01 st2 st1 0	01 st14 st7 0
0 st2 st5 0	1 st0 st1 0	00 st3 st4 0	**swma2.kiss2**	10 st2 st9 0	10 st14 st13 0
1 st2 st12 0	0 st1 st3 0	01 st3 st5 0	.i 1	00 st3 st10 0	00 st15 st6 0
0 st3 st4 0	1 st1 st0 0	10 st3 st1 0	.o 1	01 st3 st11 0	01 st15 st8 0
1 st3 st13 0	0 st2 st0 0	11 st3 st2 0	.p 16	10 st3 st13 0	10 st15 st12 0

It has been demonstrated that the genetic algorithm is valid method of finding a good state assignment; it is competitive with commercial software, and is not dependent on any particular feature of the sequential machine. Since the fitness function of the genetic algorithm may be easily modified, certain other aspects of machine design such as propagation time, the use of flip–flops other than D-type, or the use of one gate type as opposed to another may be taken into account. Further, it is believed that the effectiveness of the genetic algorithm itself may be enhanced, possibly by better chromosome encoding and more effective parent selection, and that, as a consequence, improved results may be obtained.

7 References

1 ARMSTRONG, D.B.: 'A programmed algorithm for assigning internal codes to sequential machines', *IRE Trans.*, 1962, **EC-11**, pp. 466–472

2 DOLOTTA, T.A., and McCLUSKEY, E.: 'The coding of internal states of sequential machines', *IEEE Trans.*, 1964, **EC-13**, pp. 549–562

3 HARTMANIS, J.: 'On the state assignment problem for sequential machines I', *IRE Trans.*, 1961, **EC-10**, pp. 157–165

4 STEARNS, R.E., and HARTMANIS, J.: 'On the state assignment problem for sequential machines II', *IRE Trans.*, 1961, **EC-10**, pp. 593–603

5 VILLA, T., and SANGIOVANNI-VINCENTELLI, A.: 'NOVA: state assignment of finite state machines for optimal two level logic implementation', *IEEE Trans.*, 1990, **C-9**, pp. 905–924

6 DEVADAS, S., MA, H.T., NEWTON, A.R., and SANGIOVANNI-VINCENTELLI, A.: 'MUSTANG: state assignment of finite state machines for optimal multilevel logic implementations'. ICCAD, 1987

7 HOLLAND, J.M.: 'Adaptation in natural and artificial system' (Ann Arbor-University of Michigan Press, 1975)

8 GOLDBERG, D.E.: 'Genetic algorithms in search, optimization and machine learning' (Addison-Wesley, 1989)

9 HANCOCK, P.J.: 'An empirical comparison of selection methods in evolutionary algorithms'. AISB workshop on evolutionary computing, Leeds University, UK, 1994

10 McCLUSKEY, E.J., and UNGER, S.H.: 'A note on the number of internal variable assignments for sequential switching circuits', *IRE Trans.*, 1959, **EC-8**, pp. 439–440

11 ALMAINI, A.E.A.: 'Electronic logic systems' (Prentice-Hall, 1994, 3rd edn.)

12 PERIN, J.P., DENOUETTE, M., and DACLIN, E.: 'Switching machines' (D. Reidel, 1972). Vol. 2

13 KOHAVI, Z.: 'Switching and finite state automata theory' (McGraw-Hill, 1970)