Chapter 4

Preliminary Study

In this chapter, a preliminary study of applying GA to circuit synthesis will be presented. The study, conducted in software, concentrated on the synthesis of sequential circuits from partial input/output sequences. This chapter is divided into 2 sections based on the circuit representation. Section 4.1 presents an initial work in which the circuit was represented by the configuration bits of programmable logic device (PLD). Section 4.2 presents another work in which the circuit was represented by a finite-state machine (FSM).

4.1 Evolving the Configuration Bits of GAL Structure

A sequential circuit can be constructed from the understanding of its behavior. Each state must be identified to define the state transition function and the output function. Given a behavioral description, the desired circuit can be synthesised by many conventional methods. In contrast, this work proposed a different approach: a sequential circuit was synthesised not from a behavioral description but from *single* partial input/output sequence. The paper was published in 1998 (Manovit, 1998). I present only the contents related to this thesis.

4.1.1 Input/output Sequence

A partial input/output sequence, used as a circuit specification, was attained by the following steps.

- reset the target circuit to the start state (state 0)
- fed a random input sequence to the target circuit
- recorded the corresponding output sequence

To be a general approach, the input sequence was generated at random with uniform distribution (i.e. at any time, the probability that the input bit be "0" and "1" were equal). See Figure 4.1 for example.

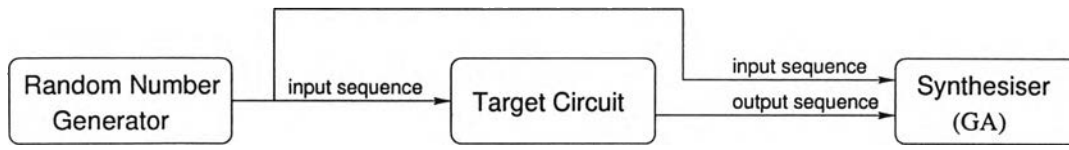

Figure 4.1: Generating an input/output sequence.

4.1.2 Circuit Representation

The circuit was encoded to the 256-bit configuration of GAL structure, composed of rows of two-level sum-of-product programmable array logic (PAL) connected to D flip-flop. The GAL structure, shown in Figure 4.2, consisted of 4-bit input and 4-bit state. It should be noticed that the GAL structure only performed a state-transition function, the output-mapping function was not included.

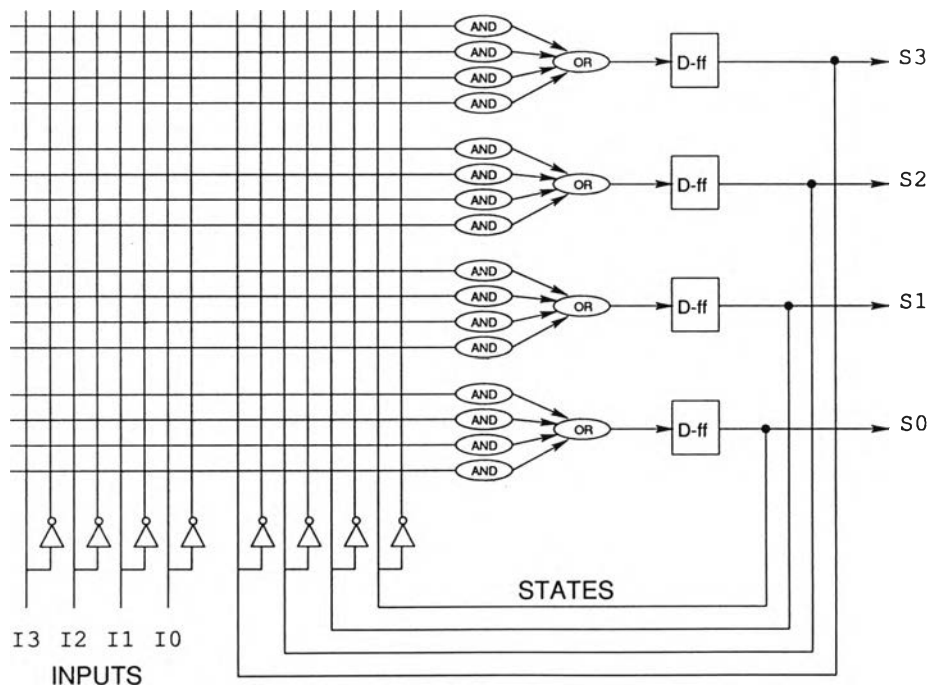

Figure 4.2: The GAL structure.

4.1.3 Genetic Operators

The simple GA was tailored to solve the circuit synthesis problem. The genetic operators were defined as follows.

- **Selection:** The 10 fittest individuals, ordered by combined rank method (Winston, 1992), were selected to survive in the next generation. The combined rank method can be succinctly summarised as selecting the individuals which were fit and different from the others.
- **Crossover:** All possible pairs among 10 already selected individual were used to produced new offspring using uniform crossover. Given two parents p_1, p_2 and a random string $mask$. The uniform crossover produced two children: $(p_1 \ \& \ mask) \mid (p_2 \ \& \sim mask)$ and $(p_1 \ \& \sim mask) \mid (p_2 \ \& \ mask)$.
- **Mutation:** The 10 selected individuals were mutated to produce 10 offspring. The mutation changed exactly 5 bits of each individual.

In addition, some constraints were taken into account. First, a configuration resulting a product term to always be "0" was not allowed (e.g. A & ~A). Second, the unused input signals were left unwired. These constraints reduced the search space of the problem and let the program concentrated on the connection points that indeed affected the function of the circuit.

4.1.4 Fitness Function

An individual was evaluated by the following steps.

1. fed an input to the circuit, then clocked the circuit
2. next state of the circuit was mapped with the corresponding output, recorded the number of times the state was mapped to the output "0" and "1" independently.
3. repeated step 1 and 2 until the end of sequence.

After the sequence was completed, the fitness value of the individual, F, was computed by:

$$F = \sum_{i=0}^{S-1} f_i \qquad \text{where} \qquad f_i = \begin{cases} +\max(p_i, q_i) & ; \quad p_i = 0 \text{ or } q_i = 0 \\ -\min(p_i, q_i) & ; \quad \text{otherwise} \end{cases} \tag{4.1}$$

where

- f_i denoted the fitness value of state i
- p_i denoted the number of times in which state i was mapped with output "0"
- q_i denoted the number of times in which state i was mapped with output "1"
- S denoted the number of states

Based on Moore's model, a state of an FSM must be mapped to only one output value. Therefore any state that was mapped to both "0" and "1" caused the penalty in the fitness value as shown in the formula. For Mealy's model, the fitness evaluation was similar to Moore's model. The output of Mealy's machine were constructed from both state and input, and therefore f_i denoted the fitness value of $(state, input)_i$ and S denoted $(number\ of\ states \times number\ of\ inputs)$.

4.1.5 Experiment Results

The GA was applied to synthesise a number of sequential circuits shown in Table 4.1. In the experiment, the state diagrams of the target circuits were known. Therefore the verification was done by comparing state diagrams of the evolved circuit and the target circuit. It was noticed that the evolved circuit sometimes was not *functionally* equivalent to the target circuit. The evolved circuit, that was functionally similar to the target circuit, was called *complete solution*. The *incomplete solution* referred to the evolved circuit that was able to perform according to a given input/output sequence but its function was not similar to the target circuit.

The correctness percentage was defined as:

$$\text{correctness percentage} = \frac{\text{number of runs yielding complete solutions}}{\text{number of runs yielding solutions}} \times 100 \tag{4.2}$$

A number of GA executions were run to obtain the correctness percentage. The experiment shows that the correctness percentage increased with the length of input/output sequence. The graph in Figure 4.3 shows that the longer input/output sequence gave a

Table 4.1: The behaviors of the tested circuits.

Circuits	#input (bit)	#output (bit)	#state		Circuit description
			Moore	Mealy	
Frequency divider	0	1	8	8	gave square wave output of frequency input divided by 8
Odd parity detector	1	1	2	2	gave an output "1" when the number of "1" in inputs is odd
Modulo-5 detector	1	1	6	5	gave an output "1" when the number of "1" in inputs can be divided by five.
Serial adder	2	1	4	2	gave the sum of 2 inputs feeding from LSB to MSB

better result. However, the correctness levelled off when the sequence length exceeded a value called *upperbound length*. The upperbound length, $\mathcal{U}$, was computed by:

$$\mathcal{U} = E(S,S) \times E(I,I) \tag{4.3}$$

where

$E(n,n)$	denoted $n(\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n})$
S	denoted the number of states (that was 16 for GAL structure)
I	denoted the number of inputs (that was 4 for serial adder)

The $E(S,S)\times E(I,I)$ was an approximation of sequence length needed to walk thoroughly in a Mealy's FSM which had S states and I inputs. For serial adder, the upperbound length was $E(16,16) \times E(4,4) = 451$. The result shows that we can calculate the upperbound length for a small sequential circuit. However, the Equation 4.3 cannot be used to predict the upperbound length of a larger circuit.

4.2 Evolving the Finite-State Machine (FSM)

In the later study, the circuit representation was changed from GAL structure to FSM due to the following reasons.

- The GAL structure was inefficient since it possibly produced the meaningless circuit (e.g. a product terms consisting of A & ~A). To avoid such circuit, the time to perform constraint check must be spent.

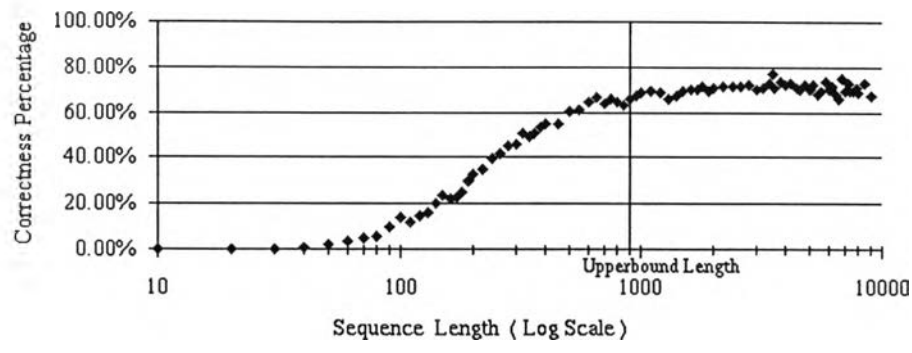

Figure 4.3: The correctness percentage of serial adder, Mealy's model. (Each point on the graph was taken from 100 independent runs)

- It was hard to determined whether the number of product terms in GAL structure was sufficient to synthesise a particular circuit or not.
- The simulation of GAL structure took a great deal of computational time. Although there was a considerable attempt to optimise the GAL simulator, it still consumed 90% of the GA execution.
- The GAL simulator was lack of scalability. It was rather complicated to program a scalable and efficient GAL simulator.

Due to the change of circuit representation, this work can be regarded as *FSM Inference* – drawing a FSM from its partial input/output sequences. The paper was published in 1999 (Chongstitvatana, 1999).

4.2.1 Circuit Representation

An FSM was encoded to a binary string to perform genetic operations. The encoding scheme was illustrated in Figure 4.4. The binary string was constructed by concatenating the next states and the outputs of all rows together. The string length, L, was computed by:

$$L = 2^s \times 2^i \times (s + o) \tag{4.4}$$

where

L denoted string length

s denoted the number of state bits

i denoted the number of input bits

o denoted the number of output bits

For real world applications, the number of internal states needed to produced a complete solution might be unknown. Therefore the number of states of an individual would be larger than the target FSM.

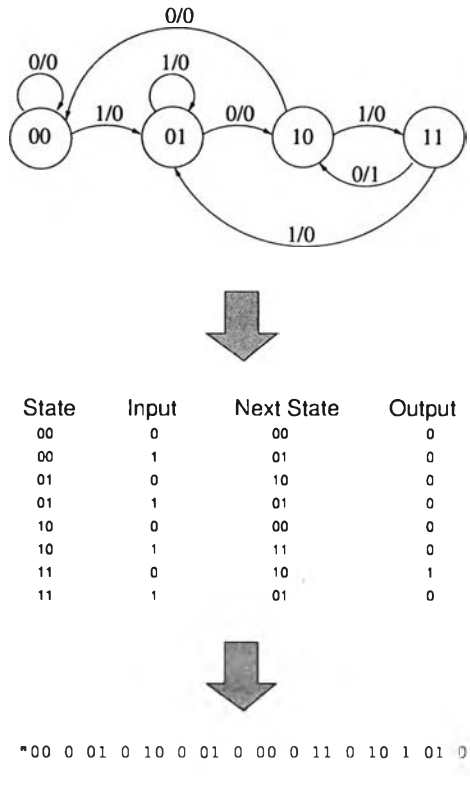

Figure 4.4: Encoding an FSM to a binary string.

4.2.2 Multiple Input/output Sequences

In previous work, the correctness levelled off after the sequence length exceeded the upperbound length. To improve the correctness, the multiple input/output sequences were proposed. The main idea was that the performance of learning system will get better with more examples. The multiple input/output sequences should yield higher correctness than single input/output sequence of the same length. Generating an input/output sequence was similar to the previous work.

4.2.3 Genetic Operators

The GA was customised to the problem. The main algorithm is presented in Algorithm 4.1. The maximum number of generations was set at 50,000. The P, Q, and R

Algorithm 4.1 Genetic Algorithms.

```
line  1: generation = 0;
line  2: initialize P individuals;
line  3: while termination conditions not met do
line  4:     produce Q individuals using crossover;
line  5:     produce R individuals using mutation;
line  6:     select  P individuals from (P, Q, R);
line  7:     generation = generation + 1;
line  8: endwhile
```

were set at 100, 200, 100 respectively. The genetic operators – crossover, mutation, and selection – were defined as follows:

- **Crossover:** A pair of parents were selected randomly from P individuals to produce two offspring using single-point crossover
- **Mutation:** A parent was selected randomly from P individuals. The selected parent was mutated to produce an offspring with the mutation probability $P_m = 0.01$.
- **Selection:** The best P individuals were selected from $(P \cup Q \cup R)$ individuals to the next generation using combined rank method (fitness rank + diversity rank).

4.2.4 Fitness function

The fitness of an individual was evaluated by the following steps:

1. fitness, $\mathcal{F}$, = 0
2. reset the individual (FSM) to start state
3. fed a given input sequence to the individual to get the corresponding output sequence

4. compared the corresponding output sequence with the given output sequence, $\mathcal{F} = \mathcal{F}$ + *number of similar output bits*

5. repeated steps 2-4 for the remaining input/output sequences

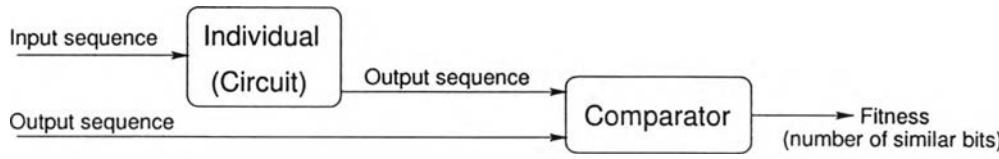

Figure 4.5: Fitness evaluation.

4.2.5 Experiment Results

A number of sequential circuits, used in the experiment, is shown in Figure 4.6. The parameters in the experiment were fixed for each circuit except the number of available internal states. We let the number of state of an individual to be larger than the number of state in the target FSM. The solution may contain redundant states and unreachable states. A conventional method can be used to optimise them. The number of available internal states for each circuit is presented in Table 4.2.

The experiment result in Table 4.3 shows that the correctness increased with the number of input/output sequences. The correctness can be raised to 100% using 100 input/output sequences. The synthesis of serial adder was analysed in order to understand this improvement. A complete solution was shown in Figure 4.7. The FSM consisted of a redundant state and an unreachable state; the state "10" was equivalent to the state "11" and the state "01" was unreachable. An incomplete solution produced by using single input/output sequence was shown in Figure 4.8. The FSM consisted of 3 parts : the initial state "00", the part A, and the part B. Part A produced incorrect outputs. Part B produced correct outputs. The first few bits in the input sequence determined which part the subsequent states belonged to. Using a single sequence, the FSM that walked into part B was indistinguishable from a complete solution. Using multiple sequences increased the possibility of exercising part A and hence identified this FSM as an incomplete solution. In other words, the multiple sequences had a better discrimination between complete and incomplete solutions.

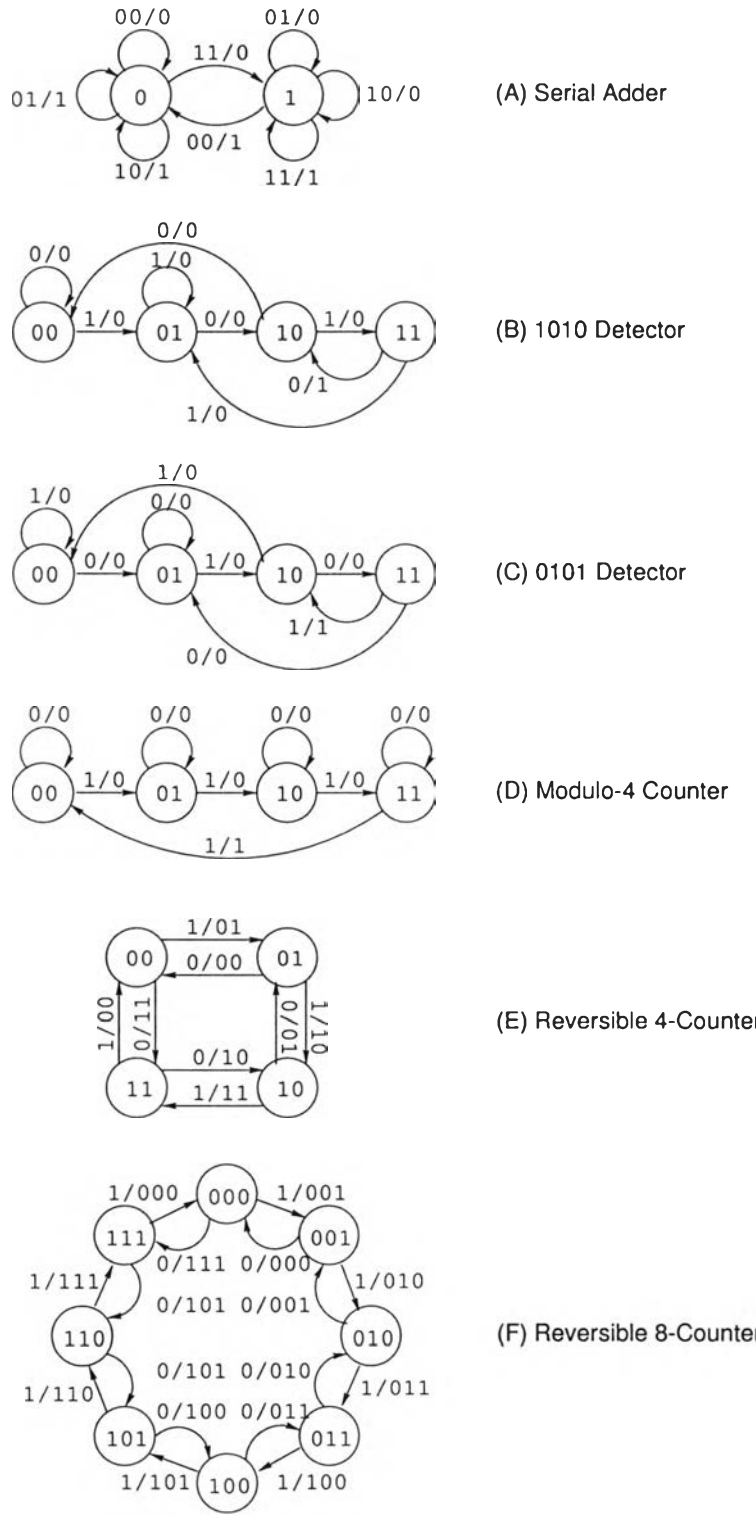

Figure 4.6: The sequential circuits used in the experiment.

Table 4.2: The number of available internal states

Tested circuits	Input (bits)	Output (bits)	The number of internal states	The number of available internal states
Serial Adder	2	1	2	4
1010 Detector	1	1	4	8
0101 Detector	1	1	4	8
Modulo-4 Counter	1	1	4	8
Reversible 4-Counter	1	2	4	8
Reversible 8-Counter	1	3	8	32

Table 4.3: Correctness Percentage

Number of Sequences	Correctness Percentage (Sequence Length = 100)					
	Serial Adder	1010 Detector	0101 Detector	Modulo-4 Counter	Reversible 4-Counter	Reversible 8-Counter
1	60.0	0.0	10.0	42.8	20.0	0.00
5	70.0	40.0	10.0	83.6	100.0	57.1
10	80.0	90.0	80.0	100.0	100.0	71.4
25	100.0	55.5	90.0	100.0	100.0	100.0
50	100.0	90.0	100.0	100.0	100.0	100.0
75	100.0	100.0	100.0	100.0	100.0	100.0
100	100.0	100.0	100.0	100.0	100.0	100.0

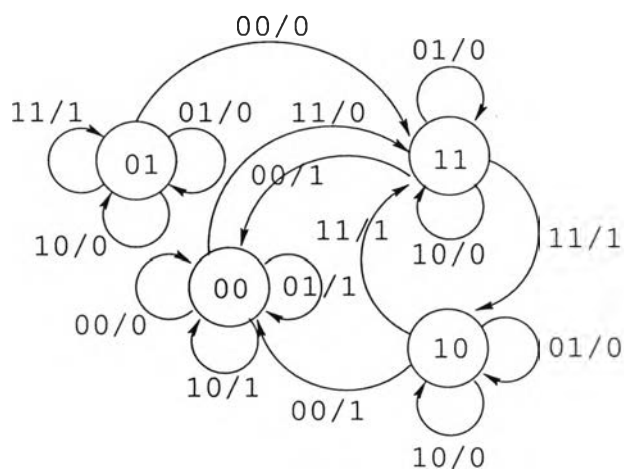

Figure 4.7: A complete solution (serial adder)

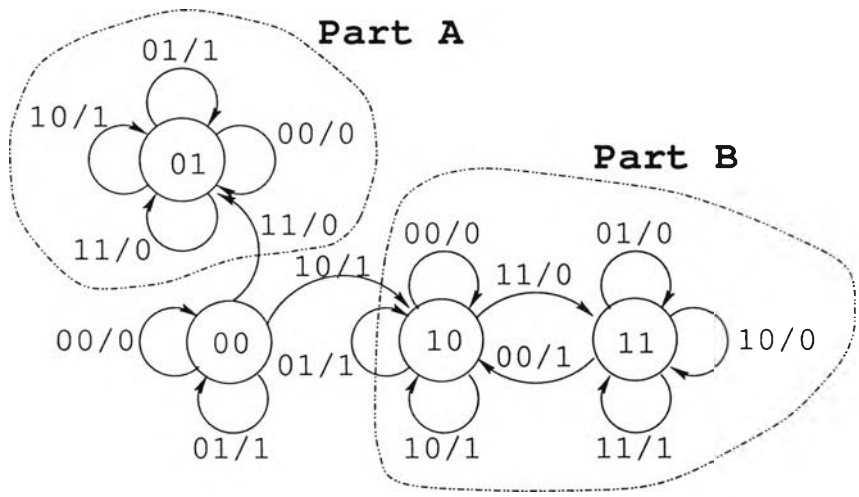

Figure 4.8: An incomplete solution (serial adder)