Chapter 3

Literature Review

3.1 Evolvable Hardware (EHW)

In 1992, the idea of applying genetic algorithms (GA) (Goldberg, 1989) to the design of electronic circuits was first investigated by T. Higuchi (Higuchi, 1992). The first experiment was to evolve the configuration bits of GAL16V8, which is a programmable logic device (PLD), to perform 6-multiplexor function (see Figure 3.1 for example). The fitness evaluation was done in GAL16V8 simulator by comparing the outputs of evolved circuit and the required outputs.

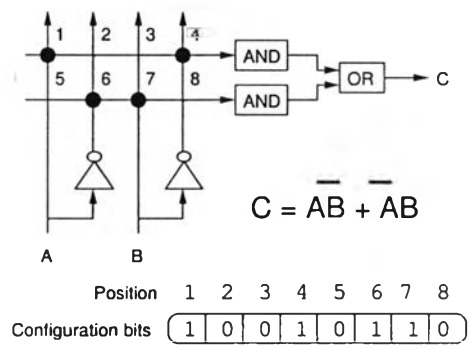

Figure 3.1: An example of PLD.

The experiment on GAL16V8 was continued by evolving sequential circuits (Higuchi, 1993; 1994). The target circuit was set at 4-state, 1-input, 1-output finite-state machine (FSM). The configuration bits were 157 bits in length. The fitness was calculated by comparing the output sequence of evolved circuit, generated by random inputs, with the required output sequence. The result showed that the evolved circuit can perform a given input/output sequence; however, the circuit function was different from the target circuit. The result implied that the input/output sequence cannot completely describe the behavior of the target circuit.

Next I will show the three categories of EHW combined with the promising works in recent years. The classification of EHW is slightly different from (Sipper, 1997a).

3.1.1 Evolutionary Design of Electronic Circuits

In this category, the common property is the *off-line* process. The GA is applied to automated circuit design, and the fitness is evaluated in the circuit simulator. Following that, the evolved solution could be further implemented in the actual circuit. This approach is named as *off-line EHW* or sometimes called *evolutionary circuit design* since the key idea is applying the evolutionary algorithms such as GA to circuit design. For a particular problem (e.g. analog circuit), the off-line EHW has potential to be competitive to human design. The difference of EHWs in this category is the circuit representation. A number of various representations are presented as follows.

Sakanashi applied *genetic programming (GP)* to search for a *binary decision diagram (BDD)* representing a desired circuit (Sakanashi, 1996). It is clear that the BDD is a standard representation since it represents a logical structure in unique and compact form. Moreover, the BDD can be verified quickly. In this work, GP is more suitable than GA because GP searches an infinite set of tree structures (Koza, 1992). The logic functions, used in the experiment, were Even {3,4,5,6}-Parity, {6-11}-Multiplexer, {4,5,6}-Symmetry, and 6-bit random. The goal was to find the compact BDDs expressing the logic functions. The result showed that GP was able to search for the optimal BDDs for 8 functions, and 2 functions was not successful.

Mizoguchi introduced hardware description language (HDL) for circuit representation (Mizoguchi, 1994). An individual represented a derivation tree of HDL grammar. The derivation tree consisted of a start symbol, non-terminal symbols, and terminal symbols. With this representation, the crossver and mutation produced only grammatically correct HDL program. The fitness of an HDL program was evaluated in a commercial HDL simulator. The goal was to generate HDL program controlling an artificial ant to follow a trail in a 32×32 block-world. The ant had one sensory input for determining whether the trail in front of it was exist or not. The ant had two outputs for three actions, move forward, turn left, and turn right. The result showed that the ant controller was achievable.

T. Higuchi suggested that evolving a very large circuit from the *gate-level* such as

AND gates, OR gates, and NOT gates is impractical for industrial applications (Higuchi, 1997). The size of circuit generated at gate-level was not very large because GA execution took a long time to evolve large circuit. In contrast, the *function-level EHW*, of which the basic components were `Adders`, `Subtractors`, and `Sine` generators, was proposed. The applications of function-level EHW will be presented later in the last category.

J. Koza proposed an automatic design of analog circuit by means of genetic programming (Koza, 1997). The tree represented a program used to construct both topology and components of analog circuit. Next the circuit was constructed and translated into a netlist simplified by removing wires, dangling components, isolated subcircuits, and consolidating series and parallel combinations of like components. Following that, the simplified netlist was evaluated in the SPICE (Simulation Program with Integrated Circuit Emphasis) (Quarles, 1994). A number of analog circuits were experimented: lowpass filter, crossover filter, source identification circuit, computational circuit, time-optimal controller circuit, amplifier, temperature-sensing circuit, and voltage reference circuit. It can be seen that the evolved circuits met the elementary requirements. Koza suggested that the commercial requirements could be reached by adding more constraints to fitness function and increasing more computational power.

J. Lohn introduced a circuit representation technique for analog circuits (Lohn, 1999). The analog circuit was constructed from an abstract machine composed of five instructions. It can be shown that any sequence of instructions produced a synthesisable analog circuit. This helpfully reduced the time to translate the circuit to a valid netlist. An individual represented a sequence of instructions generating an analog circuit. The crossover operator was modified to manipulate the variable-length strings resulted by the unfixed number of instructions. The fitness was evaluated in the public domain Berkeley SPICE running on a cluster of workstations. The result showed that three analog filters and two amplifiers that met the specifications can be achieved.

3.1.2 Intrinsic Hardware Evolution

A. Thompson introduced the terms *intrinsic hardware evolution* which means evaluating the fitness of an individual in the actual circuit (Thompson, 1998). This is contrary to the *extrinsic hardware evolution* in which the behavior of the circuit can be perfectly simulated. In tradition, the design is based on an abstract model which guarantees the

behavior of physical components. For example, the digital circuit design is based on basic gates such as AND, OR, and NOT gates. Thompson hypothesised that removing those constraints will allow more exploitation of the physical medium, increasing the chance that the better circuit which is beyond the scope of conventional design could be found. The framework of intrinsic EHW, shown in Figure 3.2, consists of an ordinary PC and a fitness evaluator connected to the PC. The PC executes the GA, except the fitness calculation done in the actual circuit.

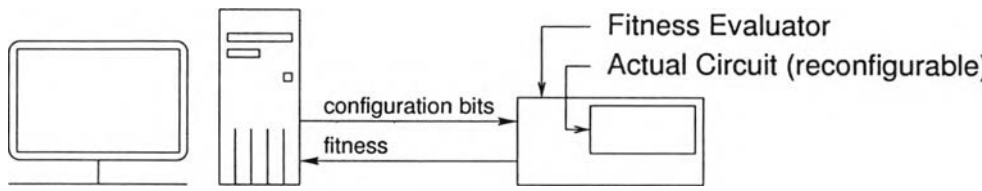

Figure 3.2: A framework of intrinsic EHW.

A robot controller was presented as an example of intrinsic EHW (Thompson, 1998: 48-56; 1999). A two-wheeled mobile robot was placed in a 2.9m×4.2m area. The task was to evolve a robot controller performing wall-avoidance/room-centering behavior. There were two sonar sensors and two DC motors on the robot. The sonars fired five times a second. When the sonar fired, its output switched from logic "0" to logic "1". Until the first echo was sensed, the output returned to logic "0". The controller, shown in Figure 3.3, was a RAM-based FSM of which two inputs were the pulses from sonar sensors without signal preprocessing, and the two outputs controlling the motors. The G.L. denoted *a bank of genetic latches* determining whether synchronous or asynchronous signal passing. This architecture, combining synchronous and asynchronous circuits, was called *Dynamic State Machine (DSM)*. In addition, the global clock was also evolved by GA. The fitness was evaluated in the actual circuit. After the final solution was downloaded into the real robot, the robot was able to perform wall-avoidance behavior.

Another demonstration is a frequency discriminator implemented on an FPGA without external clock signal (Thompson, 1998: 73-85; 1999). The task was to evolve a circuit in a 10×10 corner of an XC6216 Xilinx FPGA to discriminate 1 kHz and 10 kHz square waves. The output was "1" when the input frequency was 1 kHz, and the output became "0" when the input frequency was 10 kHz. To setup the experiment, the GA was executed on a personal computer in which an ISA card was directly plugged. The ISA card contained an FPGA employed to evaluate the fitness of an individual. Since there was

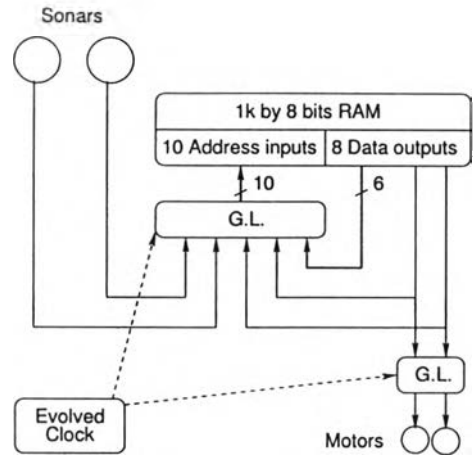

Figure 3.3: Robot controller.

no external clock, the evolved circuit exploited the gate delays to discriminate two frequencies. This resulted in an unconventional circuit; however, it showed the design space which was not yet explored.

Due to the fabrication variation, downloading the configuration bits to the other areas or the same area on the other chips degraded the performance of the frequency discriminator. In addition, the range of operational temperature was relatively narrow. It was said that the evolved circuit was *brittle*. To improve the robustness of evolved circuit, the fitness was actually evaluated in five operational environments differing in temperature, voltage source, fabrication factory, and configured area on the chip (Thompson, 1999). After 8,000 generations, the robust solution was emerged. This implied that there was a robust structure for a particular circuit.

A real world application of EHW in this category is a microwave circuit (Kasai, 2000). Since the operational frequency is higher than 1 GHz, the parasitic capacitances and inductances are unpredictable in the design stage. In conventional design, a calibration circuit is included in the microwave circuit in oder to adjust the performance to an acceptable level after the fabrication. Engineers have to balance the tradeoff between the size of calibration circuit and the adjustment complexity. The small calibration circuit causes the complex adjustment while the large calibration circuit causes the high-priced products. The circuit adjustment is a labor task and requires experienced engineers. It is obvious that to apply GA to this task, it is necessary to evaluate the fitness in the actual circuit because it is impossible to simulate the parasitic components. An image-rejection

mixer processing the received signal to an intermediate frequency and suppressing the mirror-image frequency was experimented. The result showed that GA was more effective than the iterated hill-climbing method and the manual adjustment. This profitably reduced the size of calibration circuit and the cost of experienced engineers.

The fabrication variation also causes a low yield rate in digital circuits, especially the high-speed VLSI such as Pentium III (500 MHz) or DEC Alpha (600 MHz). About 90% of total chips are discarded due to the reason that the timing delays are not met the design specification (this problem is called *clock skew*). This causes the high-priced products. To increase the yield rate, the delay-adjustable components should be included in the circuits in order to adjust the delay values by GA (Higuchi, 1999). The experiment was set as follows. After the fabrication stage, a binary string represented the delay values was loaded into a chip to perform conventional chip test. Following that, the fitness was calculated by extracting all flip-flop values from the chip to compare with the required values. The simulation of memory test pattern generator showed that GA can improve the yield rate by 50% at 800 MHz. Higuchi believed that the clock-timing adjusting architecture will be an important technology for high-speed digital systems.

3.1.3 On-line Evolvable Hardware

The EHW in this category is able to change its structure to suit the operational environment. This property is indispensable for an application of which the function cannot be predefined or the function needs to be changed during its lifetime. The on-line EHW consists of 2 parts: a GA-engine and a functional unit (see Figure 3.4). The GA-engine, executing GA, searches for a solution which performs effectively in current environment by evaluating the candidate solutions in the evaluator unit. The functional unit is periodically updated with the best solution.

M. Sipper demonstrated the on-line EHW called *Firefly Machine* (Sipper, 1997b). The firefly machine is based on *cellular automata (CA)*. The cellular automaton is composed of an array of cells. The array is n-dimensional, where n = 1, 2, 3 is used in practice. Each cell holds a state in which the number of states is finite. The state of a cell is updated synchronously by a rule. In uniform CA, the rules of each cell are similar, but the rules can be different in non-uniform CA. The rule is a combinational logic of which the inputs are the states of neighbouring cells and the state of itself. The number of

neighbours on each side (left/right) is determined by a parameter r. An one-dimensional, 2-state, non-uniform, r=1 CA is shown in Figure 3.5A.

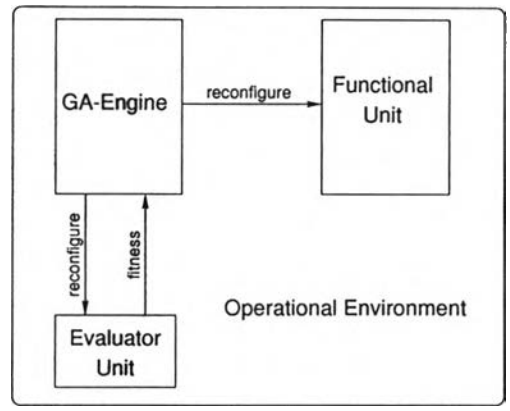

Figure 3.4: On-line EHW.

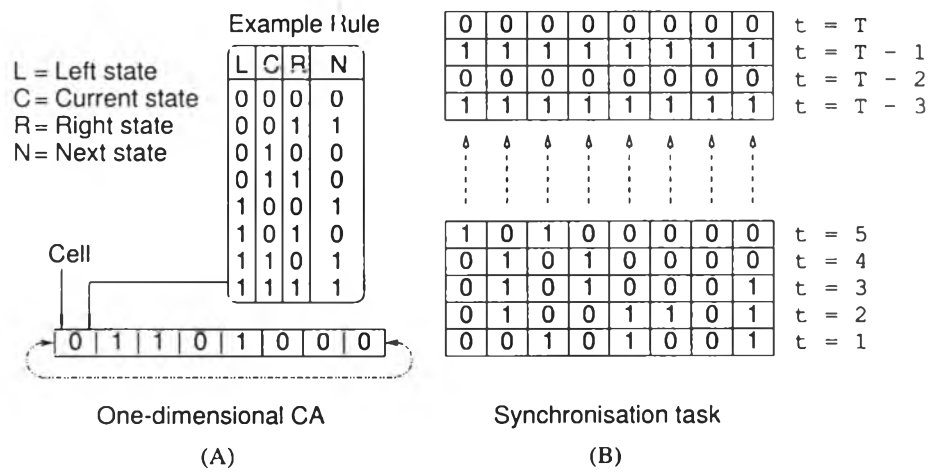

Figure 3.5: An one-dimensional, 2-state, non-uniform, r=1 cellular automata (CA).

Given an initial configuration (initial states, where $t = 1$), the task was to synchronise the states to a sequence of $0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \ldots \rightarrow 0 \rightarrow 1$ (see Figure 3.5B for example). The synchronisation task is a hard problem. In nature, the fireflies come together and gradually blink synchronously. There is no clear answer how the synchronisation can be occurred. The *cellular programming* - an algorithm used to program the CA to perform a task - was prototyping in FPGA technology. The firefly machine was able to evolve a set of rules performing synchronisation task from an initial configuration. It can be seen that the rules was not designed in advance. The rules was adapted to the current initial configuration which was manually changeable.

The myoelectric prosthetic hand controller is a prominent application of on-line E-HW (Higuchi, 1999). The artificial hand is designed for a disabled person. The prosthetic hand is used to capture a signal (electromyography signal, EMG) generated from the muscular movements, then perform a hand action according to the received signal. Usually the disabled persons have to train with the prosthetic hand at least one month before they skilfully control the muscular movements generating the desired hand actions. The EMG signals vary greatly from person to person. Moreover, for a particular person, the EMG signal may change over a short period due to the physical conditions. Thus a design of prosthetic hand in advance causes the burdensome training. In contrast, the on-line EHW is able to adapt the translation of EMG signal to a desired hand action. The use of prosthetic hand was carried out by the following steps. First, the EMG signal for six actions were collected as training data. Second, the hand controller, translating EMG signal to a desired action, was evolved by GA. All steps was done in a single VLSI chip embedded in the prosthetic hand.

Next I will show the applications of function-level EHW. The adaptive equalization, used in digital mobile communication, was evolved (Higuchi, 1997). Normally the linear transversal filter is employed to improve the received signal corrupted by the channel distortion and noise. In digital mobile communication, the topography of the operational area (e.g. buildings) differently influences the channel. Accordingly, the adaptive equalizer is preferable. To establish a communication, the transmitter sent a known training sequence to the receiver, then the EHW-based equalizer adapted itself to minimise the bit error rate for the current environment. Another application of function-level EHW is a lossy compression based on *non-linear predictive coding* (Higuchi, 1997). A template, used for pixel prediction of a particular image, was evolved using GA. The compression circuit and decompression circuit were reconfigured according to the evolved template. I suppose this work was further developed to a lossless compression chip used in electrophotographic printer (Tanaka, 1998; Higuchi, 1999).

3.2 Hardware-based Genetic Algorithms

Due to the extensive computation of GA (Goldberg, 1989), a myriad of hardware-based GAs has been put forward. Here we cite only the more recent works.

In 1995, Scott introduced an implementation of simple GA on field-programmable gate array (FPGA) (Scott, 1995). Scott proposed the hardware framework to speedup GA.

The implementation by means of FPGAs allowed the changeability which was necessary for the fitness evaluation module. The hardware-based genetic algorithm (HGA) was presented as a proof-of-concept system. It was designed using VHSIC hardware description language (VHDL) for the scalability. The HGA was experimented on linear, quadratic, and cubic functions. In terms of clock cycles, the speedup achieved was 2-3 orders of magnitude compared to software-based GA running on Silicon Graphics 4D/440 with four MIPS R3000 CPUs. The full details, included VHDL code, can be seen in his master thesis (Scott, 1994).

Graham used the Splash 2, a reconfigurable computer, to solve a 24-city TSP (Graham, 1995). The Splash2 consisted of a collection of processor array boards connected to Sun Sparc via an interface card. The workstation supported reconfiguration, control, and data exchange with the Splash 2 board. The simple GA, using 4 FGPAs and 4 separated memories, was 10.6 times faster than software running on 125 MHz HP PA-RISC workstation. The *island model* - distributed version of GA - was investigated. Four simple versions of GA were conducted simultaneously with periodic migration of solutions among the islands. The result showed that the island model converged to the optimum solution quicker than the other configurations.

Graham subsequently analysed the different performance between hardware and software version of GA (Graham, 1996). It can be seen that the hardware was more efficient because it employed the benefits of fine-grained parallelism, custom address generation, and well-organised memory hierarchy. In addition, the random number generator in software dominated the execution time of crossover and mutation, and therefore only hardware contributed to produce the random numbers can remarkably improve the performance.

Sitkoff used the Armstrong III Machine to solve a 500-component chip partitioning problem (Sitkoff, 1995). The Armstrong III was a MIMD multicomputer with reconfigurable resources. It was composed of an array of processor boards. Each processor board consisted of a 33MHz AMD AM29050, 32 Mbytes DRAM, and three Xilinx XC4010 FPGAs. The software profiling indicated that 95% of the execution time was spent on fitness evaluation. For that reason, the reconfigurable resources were contributed to speedup the fitness evaluation. The result showed that Armstrong III was 2.98 times faster than the software executing on 60MHz SPARC 20 Model 61. In addition, 8.32× speedup was reported in case of the distributed GA.

Salami investigated an implementation of simple GA on FPGAs (Salami, 1996). This work was similar to Scott's hardware-based GA. The GA processor (GAP) was designed using VHDL, and therefore synthesised on Xilinx XC4013 FPGAs. The GAP was tested with De Jong test suites and adaptive IIR filter.

Shackleford proposed a high-performance genetic algorithm machine (Shackleford, 1997). The survival-based GA was proposed because it was more suitable for pipelining. The design was exactly matched to a benchmark problem. A powerful pipeline and an evaluator calculating the fitness in one clock delivered an extreme speedup compared to software. At 1 MHz, the hardware produced an evaluated individual every clock cycle. In other words, one million solutions were explored in one second. Testing with the set coverage problem indicated a 2,200× speedup over software on a 100 MHz workstation.

Kajitani proposed an evolvable hardware chip running steady-state GA (Kajitani, 1998). The steady-state GA, which was similar to survival-based GA, was preferable because it was more suitable for pipelining than generational GA. The chip consisted of 16-bit CPU (NEC V30) executing GA and two programmable logic devices (PLDs) used for fitness evaluation. This paper pointed out that the compactness was a key to realise the on-line EHW. The EHW chip was applied to the myoelectric artificial hand used for disable person. The 62× speedup compared to software running on Ultra Sparc 2 (200 MHz) was reported.

Yoshida introduced a VLSI implementation of steady-state GA, called genetic algorithm processor (GAP) (Yoshida, 1999). The experiment was set into three configurations: mono-GAP with single fitness evaluation processor (FEP), GAP with dual FEPs, and multi-GAP with dual FEP's. The design was realised using hardware description language (HDL). The HDL simulator was used to optimise the Royal Road function. The design is now fabricating in FGPA technology.

Most of the cited works achieved their speedup because of pipelining and dedicated function customised to the problems. The initial works were solely to speedup the GA. The tremendous resources were used to attain the high performance, for example, (Graham, 1995; Sitkoff, 1995). A group of researches used hardware description language which can be changed and scalable, for example, (Scott, 1995; Salami, 1996; Yoshida, 1999). However, the designs were not practical because the number of transistors increased with the problem size. The tradeoff between cost and performance should be

trimmed. In the later works, the goal was to realise the EHW in practice. The compactness was an essential whereas the speed was less significant. A modest microprocessor combined with a fitness evaluator was favourably effective to perform GA function, for example, (Kajitani, 1998).

Our work is similar to (Kajitani, 1998) using the microprocessor and the fitness evaluator. The design techniques are acquired from the previous works, for example, the random number generator. The difference is our custom microprocessor of which the instruction set is customised to an execution of GA. With this aim, we hope to achieve a design that is simple and effective to demonstrate the mimetic EHW.