Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Supporting VHDL Design for Air-Conditioning Controller Using Evolutionary Computation

Kazuyuki Kojima * **Keiichi Watanuki** **

* *Saitama University, 255 Shimo-okubo, Sakura-ku, Saitama-shi, Saitama, JAPAN (Tel: 81-48-858-9576; e-mail: kojima@mech.saitama-u.ac.jp).*
** *Saitama University, 255 Shimo-okubo, Sakura-ku, Saitama-shi, Saitama, JAPAN (Tel: 81-48-858-3433; e-mail: watanuki@mech.saitama-u.ac.jp).*

Abstract: In recent years, as part of the remarkable development of electronic techniques, electronic control has been applied to various systems. Many sensors and actuators have been implemented into those systems, and energy efficiency and performance have been greatly improved. However, these systems have been complicated, and much time has been required to develop system controllers. In this paper, a method of automatic controller design for those systems is described. In order to automate the design of an electronic controller, an evolutionary hardware is applied. First, the framework for applying the genetic algorithm to the automation of controller design is described. In particular, the coding of a chromosome is shown in detail. Then, how to make a fitness function is represented, with an air conditioner as an example, and the controller of the air conditioner is developed automatically using our proposed framework. Finally, an evolutionary simulation is performed to confirm our framework.

1. INTRODUCTION

Advances in electronics have enabled highly efficient electronic parts to be built into control systems controlled more efficiently and intricately than ever. Many such systems convert analog signals from sensors to digital signals, which are processed by micro processing units (MPUs) or digital signal processors (DSPs), which determine controlled variables and/or control sequences. Digital signals are reconverted after processing into analog input to a drive circuit in order to drive actuators.

When designing such a controller, a designer organically combines resources such as sensors, actuators, and an MPU that constitute a system. This can be done comparatively easily when the system is on a small scale and the signal from a sensor is used directly without any complex calculation or any estimation. For example, positioning controls and speed controls are such cases. Of course, since there is a problem of stability or response even with a small-scale system, tuning is required, and design is not easy, although compared with a large-scale system, the design of a controller is also not so difficult. The gain-regulating proportional-integral-derivative (PID) controller is an example of parameter tuning. However, considerable development work is required in a large-scale system, where complex operation must be implemented. Control of a robot's walking with obstacle avoidance is such an example. In this case, in order to develop a function, the controller designer will pay great attention to using the full resources that constitute a robot and will build the structure of a control program or a logical circuit. Systems in recent years have many components similar to that of a robot, therefore the flexibility of controller design is increasing, but the difficulty of controller design is also increasing. In home electronics and car components, many of such control systems exist.

In this paper, in order to support the design of such a controller, a complex programmable logic device (CPLD) is used for the data-processing part of a controller, and very-high-speed integrated circuit hardware description language (VHDL) which describes the logical circuit in the CPLD is optimized using evolutionary computation.

2. COMPUTER-AIDED CONTROLLER DESIGN USING EVOLUTIONARY COMPUTATION

2.1 FPGA/CPLD/ASIC and VHDL

In this paper, a programmable LSI is used for the implementation of evolutionary computation. CPLDs and FPGAs are both a sort of programmable LSI. The FPGA is SRAM-based where the scale of a logical block is comparatively small, and the CPLD is EEPROM-based where the scale of a logical block is large. The internal logic of both can be designed using HDL. The ASIC is one example of a device that can be designed using HDL in the same way as programmable LSIs. CPLDs and FPGAs can be immediately evaluated on the system for the designed logical circuit. In addition, they are flexible for the rearrangement of a specification. These merits make them suitable for the intended use in the case of a rapid prototyping. For this reason, a CPLD is used as a controller in this paper. However, the proposed framework is applicable to all devices that can be designed by HDL.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity HALF_ADDER is
  port(
    A,B : in std_logic;
    S,CO : out std_logic);
end HALF_ADDER;

architecture DATAFLOW of HALF_ADDER is
  signal C, D : std_logic;
begin
  C  <= A or B;
  D  <= A nand B;
  CO <= not D;
  S  <= C and D;
end DATAFLOW;
```

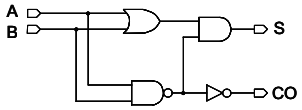

Fig. 1. VHDL for a simple logical circuit

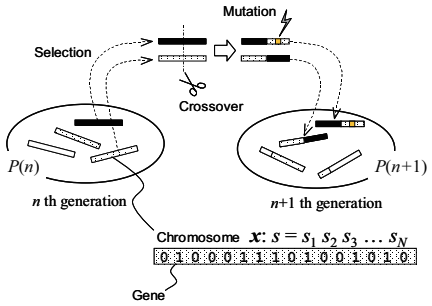

Fig. 2. Outline of the genetic algorithm

VHDL is one of the most popular HDLs, and is therefore used in this paper.

The logic described by VHDL is verified and synthesized using a simulator or a logic synthesis tool so that it can be written into a device. When CPLD or FPGA serves as target devices, the programming code which determines the function of the target device can be, through a download cable, written into it in order to obtain the target LSI easily. The VHDL for a simple logical circuit is shown in Fig. 1.

2.2 Genetic algorithm

The genetic algorithm used as a basis of our framework is outlined in Fig. 2. The decision-variable vector $\boldsymbol{x}$ of an optimization problem is expressed with the sequence of N notations $s_j (j=1,\cdots, N)$ as follows:

$$\boldsymbol{x} : s = s_1 s_2 s_3 \cdots s_N \tag{1}$$

It is assumed that the symbol string s is a chromosome consisting of N loci. s_j is a gene in the jth locus and value s_j is an allelomorph. The value is assumed to be a real number, a mere notation, and so on of a group of integers or a certain range of observations as an allelomorph. The population consists of K individuals expressed with Eq. (1). The population $p(n)$ in generation n changes to the population $p(n+1)$ in the next generation $n+1$ through the reproduction of a gene. If reproduction in a generation is repeated, and if the individual who expresses solution $\boldsymbol{x}$ nearer to an optimum value is chosen with high probability, then the value increases and an optimum solution is obtained as detailed in references [1, 2].

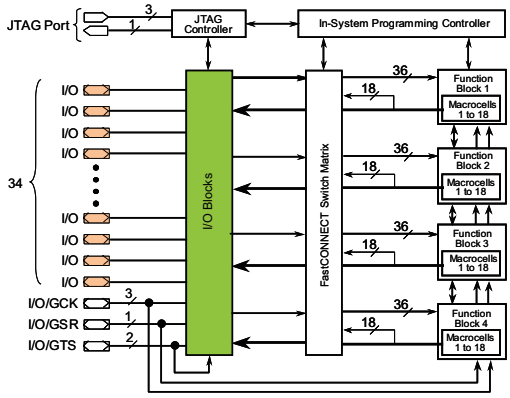

Fig. 3. XC9572 architecture

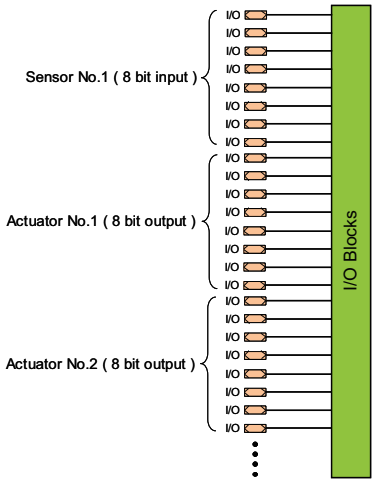

Fig. 4. Example of CPLD application

2.3 Framework of controller design using evolutionary computation

The study that optimizes a rewritable logical-circuit IC like the CPLD using a genetic algorithm has been applied in recent years. The framework which changes the internal configuration of logical-circuit IC so as to achieve its intended purpose in an evolutionary fashion is called evolvable hardware (EHW). Using this framework, a designer has only to define the criteria which evaluates a controller. In this paper, the framework of a controller design using EHW is explained with XC9572[3] as a test device. Internal blocks of XC9572 are shown in Fig. 3. XC9572 is a small CPLD that has 44 pins (34 user input-outputs), 72 macro cells, and the 1600 usable gates. The designer chooses input and output signals from 34 user I/Os, and defines the pin assignments. Each signal is configured to each I/O. In the case that a CPLD is used in a control system, sensors and actuators can be associated to the I/O pins of the CPLD. An example of association is shown in Fig. 4. In this example, I/O pins are associated to one sensor and two actuators. The sensor value is inputted into the CPLD as an 8-bit digital signal, and two 8-bit digital signals are outputted as reference signals to two actuators.

The VHDL, which describes the internal logic of the CPLD, is encoded on a chromosome. An example of generated VHDL is shown in Fig. 5. This example serves as a description corresponding to Fig. 4. This VHDL consists of three declaration parts: (a) entity declaration part, (b) signal declaration part, and (c) architecture declaration part. The I/O signals of the CPLD are defined on part (a). The internal signals of the CPLD are defined on part (b). As for a description of signals in VHDL, the `std_logic` type and the `std_logic_vector` type are mainly used. The `std_logic` type can be used when dealing with a signal alone, and the `std_logic_vector` type can be used when dealing with some signals collectively. It is better to use the `std_logic` type and the `std_logic_vector` type considering maintenance and readability. However, when applying to our framework, the `std_logic` type is better to use. If two or more types are used to describe signals, the VHDL decode process from the chromosome is complicated and searching space becomes wider. A VHDL description which uses a `std_logic_vector` type can be replaced by a VHDL description which uses two or more `std_logic` types. The description can be restored if all input, output and internal signals are used as the same `std_logic` type and only the number will be encoded on the chromosome. Then, the number of input signals, the number of output signals, and the number of internal signals are encoded on the head of the chromosome as shown in Fig. 6. In the case of Fig. 4, an input signal is set up with 8 bits and an output signal with 16 bits.

A chromosome which represents the VHDL statement of substitution indicated by Fig. 5(d) is shown in Fig. 7. The chromosome structure corresponding to a process statement is shown in Fig. 8. The value currently described in the figure is equivalent to the process statement in which "`S000`" and "`DI002`" are enumerated at the sensitivity list (Fig. 5(e)). A description of this VHDL has an if-statement in the inside of a process statement, and the description has two nesting levels. The hierarchy of the list structure is deep compared with the assignment statement indicated by line (d) in Fig. 5. As the gene of a multi-list structure is prepared, it would be possible to represent various VHDL expressions.

2.4 Variable length chromosome and genetic operations

The structure of the chromosome changes according to the design specification of the control system. The number of internal signals can be set up arbitrarily, and various descriptions in VHDL are expressed with different length of locus. The length of the chromosome is determined by the line count of VHDL. In addition, the length is determined by the number of internal signals enumerated on the sensitivity list or the length of the right-hand side of an assignment statement. When dealing with such a variable length chromosome, the problem is that the genetic operations will generate conflict on the chromosome. In order to avoid this problem, the following restrictions are observed.

(1) With a top layer, the length of the chromosome is equal to the number of internal signals plus the number of output signals plus one.
(2) All the signals are encoded on the chromosome using a reference number.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity GA_VHDL is
    port(
        DI000 : in std_logic;
        DI001 : in std_logic;
        DI002 : in std_logic;
        DI003 : in std_logic;
        DI004 : in std_logic;
        DI005 : in std_logic;
        DI006 : in std_logic;
        DI007 : in std_logic;
        DO000 : out std_logic;
        DO001 : out std_logic;
        DO002 : out std_logic;
        DO003 : out std_logic;
        DO004 : out std_logic;
        DO005 : out std_logic;
        DO006 : out std_logic;
        DO007 : out std_logic;
        DO008 : out std_logic;
        DO009 : out std_logic;
        DO010 : out std_logic;
        DO011 : out std_logic;
        DO012 : out std_logic;
        DO013 : out std_logic;
        DO014 : out std_logic;
        DO015 : out std_logic
    );
end GA_VHDL;

architecture Behavioral of GA_VHDL is

    signal S000 : std_logic;
    signal S001 : std_logic;

begin

    S000 <= (((((not DI007 nand DI004) nor not DI005) or DI003) and not DI007) nand not DI003);
    S001 <= ((((not DI007 nor not DI007) nor DI004) or not DI002) or DI002);

    process(S000, DI002) begin
        if(S000'event and S000='1')then
            DO000 <= (DI002 nand not S000);
        end if;
    end process;

    DO001 <= not DI000;

    process(S001) begin
        DO002 <= S001;
    end process;

    DO003 <= (((((not DI006 and not DI002) nand not DI001) nand not DI001) or not DI006) or DI005);
    DO004 <= ((((DI001 and not DI003) nand DI006) and DI002) nor not DI000);

    process(DI001, S000) begin
        DO005 <= (S000 nand DI001);
    end process;

    DO006 <= (((not DI000 and not DI002) nor DI003) and DI006);

    process(S000, DI001) begin
        if(S000'event and S000='1')then
            DO007<=DI001;
        end if;
    end process;

    DO008 <= (((((((S001 or DI000) and not DI006) or not DI001) nand not DI000) and not DI003)
                nor DI003) nand not DI002) or DI001);

    DO009 <= (((((not DI001 nand not DI001) nor DI006) nand DI001) nor not DI001) and S000);

    process(DI004) begin
        DO010 <= not DI004;
    end process;


    process(S001) begin
        if(S001'event and S001='0')then
            DO011<=S001;
        end if;
    end process;


    process(DI003, DI002) begin
        if(DI003='1')then
            DO012<=(DI003 nand not DI002);
        end if;
    end process;


    process(DI005) begin
        DO013 <= DI005;
    end process;

    DO014 <= (((((((not S000 nand not DI003) nand not DI006) or DI005) nand S001) nand S001)
                or not S001) or DI005) nand DI007);

    DO015 <= (((((((not DI001 nor not DI003) or not DI007) nor not DI003) nand not DI000)
                or DI005) and not DI007) nor not DI007) nand not DI001);

end Behavioral;
```

(a) Entity declaration
(b) Signal declaration
(c) Architecture body
(d) Substitution
(e) Process statement

Fig. 5. Automatically generated VHDL

(3) The signal with a large reference number is described by only the signal whose reference number is smaller than the signal.
(4) The top layer of the chromosome describes the entity declaration part using all internal signals and output signals in order with a low reference number. Each signal can be used only once.
(5) The crossover is operated on the top layer of the chromosome.

These restrictions help avoid the conflict produced by genetic operations.

Here, an example is given. Fig. 9 shows chromosomes of two different lengths. The length of the chromosome is determined at the initialization and it is changed by genetic operations. The length of a chromosome equals one plus the number of internal signals plus the number of output signals. The number of inputs and the number of outputs are determined by the specification of hardware. In Fig. 9, the number of inputs is determined as 8 and the number of outputs is determined as 16. They are determined by the designer. They do not change in evolutionary calculation. The number of internal signals is determined randomly at the initialization. In Fig. 9, they are determined to be 5 and 2. A reference table is made

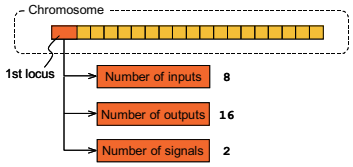

Fig. 6. Signal definition on the first locus

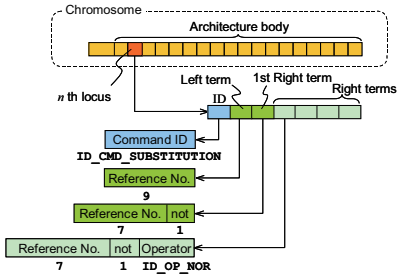

Fig. 7. Substitution

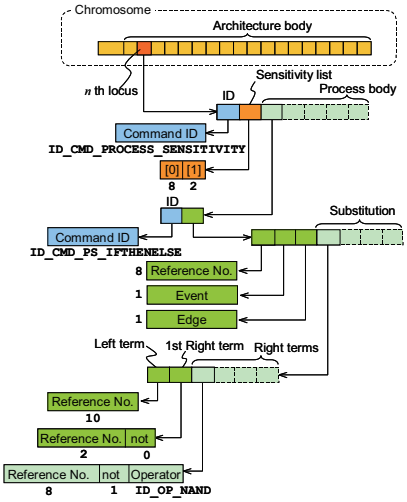

Fig. 8. If-then-else statement in process body

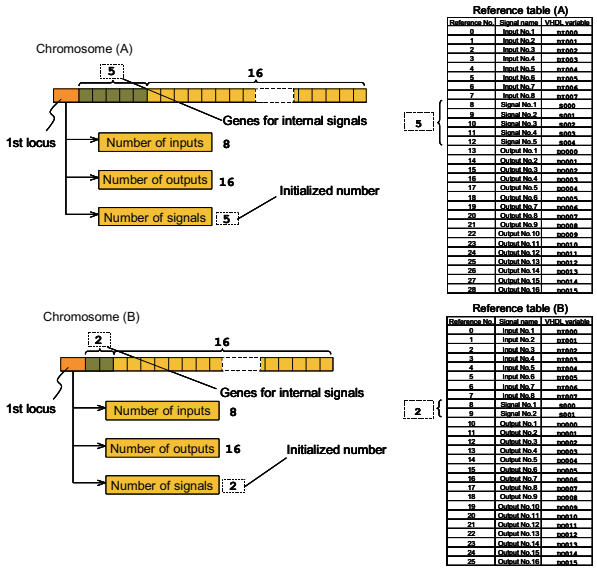

Fig. 9. Two different length chromosomes

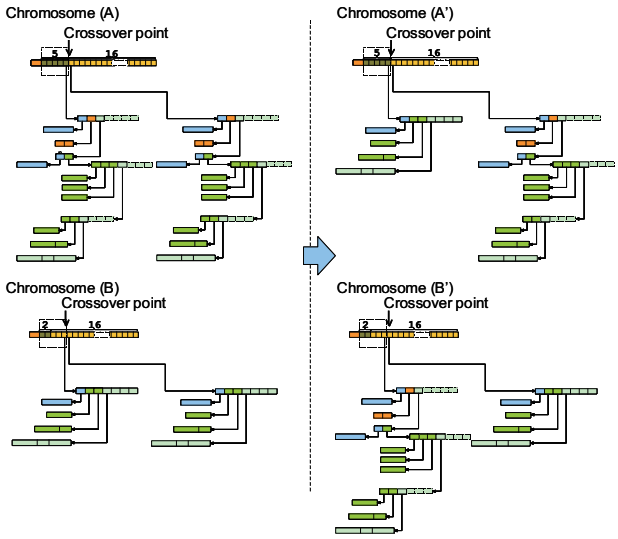

Fig. 10. Crossover

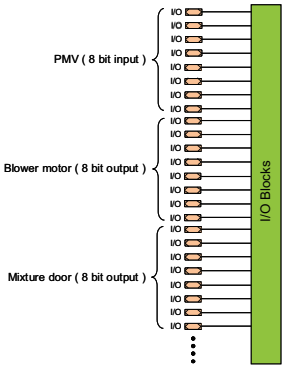

Fig. 11. CPLD application to air conditioning

according to the number of signals. If the number of signals differs, the size of the table also differs.

An example of crossover is shown in Fig. 10. The back of the 6th gene is chosen in this example. Chromosome (A) and chromosome (B) cross and change to chromosome (A') and chromosome (B'). Only the gene before and behind the crossover point of each chromosome shows the gene of a lower layer. In the figure, chromosome (A) has two sensitivity lists and chromosome (B) has two assignment statements. The structure of a chromosome changes by replacing the gene from the back of a top gene to before a crossover point. Both chromosome (A') and chromosome (B) came to have an assignment statement and a sensitivity list.

3. APPLICATION TO AIR CONDITIONING

3.1 Air-conditioning controller using evolutionary computation

In the sample application of Fig. 4, if the 8-bit input is set to Predicted Mean Vote (PMV)[4] in a cabin, and two 8-bit outputs are set to two actuators of the air conditioner, the controller can be used as an air-conditioning controller (Fig. 11). In this paper,

evolutionary computation application to air conditioning is shown.

An air-conditioning controller will be employed to keep the inside of a cabin comfortable with the internal and external thermal state. However, when developing the controller of such an air-conditioning system, the designer has to consider many sensors and actuators which constitute an air-conditioning system. The room temperature, water temperature, outdoor temperature, and solar radiation must be measured, and a controller has to control many actuators, such as a blower, an air mix door, and a mode change door. In such a case, since many parameters must be taken into consideration, *trial and error* of the system developments must be repeated many times. Great effort is required. So in this paper, evolutionary computation is applied to designing the controller in order to reduce such *trial and error*.

The schematic diagrams of an air-conditioning system are shown in Fig. 12. Air is taken from the inlet by a blower. All the air that flows in is cooled by 5 degrees. They are dehumidified at this time. Then, a part of the air is warmed by a heater to 80 degrees. The opening of a mix door is changed to adjust the mixing ratio of the warm air and the cold air. The input to a controller is PMV as follows.

3.2 Predicted mean vote (PMV)

PMV is the predicted mean vote of a large population of people exposed to a certain environment. PMV represents the thermal comfort condition on a scale from -3 to 3, derived from the physics of heat transfer combined with an empirical fit to sensation. Thermal sensation is matched as follows: "+3" is "hot." "+2" is "warm." "+1" is "slightly warm." "0" is "neutral." "-1" is "slightly cool." "-2" is "cool." "-3" is "cold." Fanger derived his comfort equation from an extensive survey of the literature on experiments on thermal comfort[4]. This equation contains terms that relate to clothing insulation I_{cl}[clo], metabolic heat production M[W/m^2], external work W[W/m^2], air temperature T_a[℃], mean radiant temperature T_r[℃], relative air speed v[m/s], and vapor pressure of water vapor P[hPa].

$$\begin{aligned}\mathrm{PMV} = \{0.33\exp(-0.036M)+0.028\}\Big[&(M-W)\\ &-3.05\{5.73-0.007(M-W)-P\}\\ &-0.42\{(M-W)-58.1\}\\ &-0.0173M(5.87-P)\\ &-3.96\times10^{-8}f_{cl}\{(T_{cl}+273.15)^4\\ &-(T_{mrt}+273.15)^4\}\\ &-f_{cl}h_c(T_{cl}-T_a)\Big] \end{aligned} \quad (2)$$

f_{cl} is the ratio of clothed and nude surface areas given by:

$$\begin{aligned} f_{cl} &= 1.0+0.2I_{cl}(I_{cl}\le 0.5)\\ f_{cl} &= 1.05+0.1I_{cl}(I_{cl}>0.5) \end{aligned} \quad (3)$$

where T_{cl} is the clothing surface temperature given by repeated calculation of:

$$T_{cl} = 35.7-0.028(M-W)$$

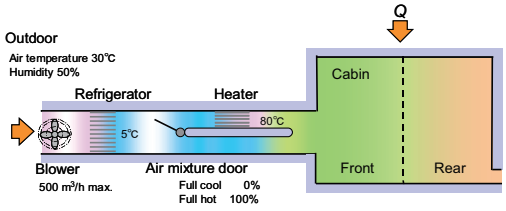

Fig. 12. Air-conditioning system

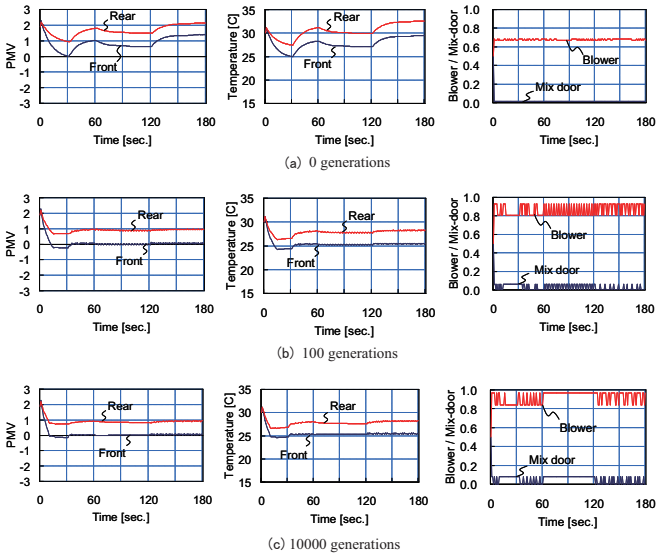

Fig. 13. Simulation results

$$\begin{aligned} -0.155I_{cl}\Big[&3.96\times10^{-8}f_{cl}\{(T_{cl}+273.15)^4\\ &-(T_{mrt}+273.15)^4\}+f_{cl}h_c(T_{cl}-T_a)\Big] \end{aligned} \quad (4)$$

where h_c is the heat transfer coefficient,

$$h_c = \max\{2.38(T_{cl}-T_a)^{0.25}, 0.0121\sqrt{v}\} \quad (5)$$

and T_{mrt} is mean radiant temperature. PMV is detailed in [4].

3.3 Fitness function

The fitness function is as follows, where ΔE is the difference between target PMV and estimated PMV, t is time, T_{end} is the end of calculation time.

$$\mathrm{fitness} = -\int_0^{T_{end}} \Delta E dt \quad (6)$$

The value will become high if it is a minimum of the integrated value. In the simulation, a variation of a heat load is given as a disturbance. The load is given randomly between 0 W and 1100 W. The load is changed two times in one fitness calculation; the timing of load switching is also given randomly. The fitness values of chromosomes are different from each other, even if the chromosomes are exactly the same. However, the logic generated in this way has high robustness to a disturbance.

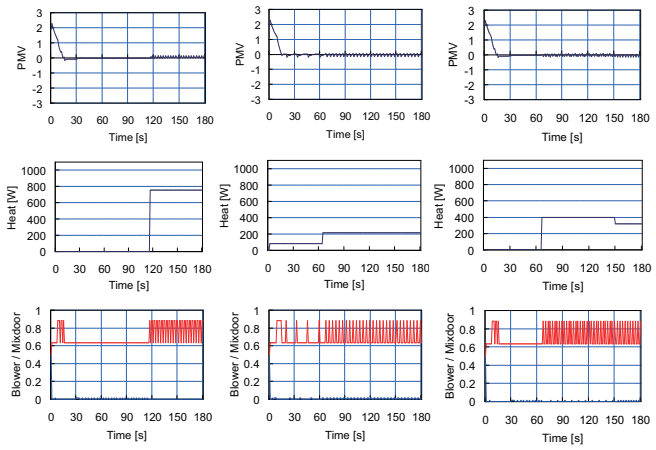

Fig. 14. Simulation results

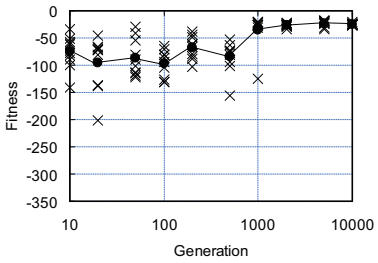

Fig. 15. Ten elite values in each generation

3.4 Simulation results

Figures 13, 14, and 15 shows the simulation results. In this case, PMV in a front cabin is fed back to the controller. In this calculation, the population size is 50, crossover rate is 1.0, mutation rate is 0.5, selection method is tournament, and tournament size is 10. In all graphs, a tendency changes every 60 seconds and changes at 30 seconds. These variations are based on the load change.

At the time of zero generation shown in Fig. 13(a), with a change of a heat load, the temperature rises or descends. PMV also changes simultaneously. This means that the optimization of a controller is inadequate. After 100 generations of calculation, the difference between the target value and the estimated value decreases (Fig. 13(b)). At the 10000th generation as shown in Fig. 13(c), the tolerance decreases further. These results show that the hardware corresponding to the purpose can be obtained automatically by using this framework.

Figure 14 shows the air-conditioning control under different heat loads. In order to correspond to an alternation of a disturbance and to minimize PMV, the rotation of the blower and the opening of the air mixture door are controlled.

Figure 15 shows ten fitness values in each generation. These values are the results of calculating the fitness value of each generation's elite 10 times. Since the fitness value has given the thermal load at random as mentioned above, a value which is different whenever calculated is shown, but variation becomes small as a generation progresses. However, a possibility that the optimal solution from which evolution calculation is obtained is a partial solution is pointed out. Also in the framework which we propose, the obtained optimal solution may be a partial solution. This is considered as a future subject.

Here, the calculation result up to 10000 generations was shown. However, in the case of this example simulation, about 2000 generations are enough. The slope of afitness value and the error of the maximum fitness values are consulted when deciding the end generation of calculation.

4. CONCLUSION

In this paper, in order to support the design of a controller of a mechatronics system, a CPLD used for the data-processing section of the controller and VHDL which describes the logical circuit were optimized using evolutionary computation. First, the framework of how to apply the evolutionary computation to the automation of controller design was described. In particular, the coding of a chromosome was shown in detail. Then, how to constuct a fitness function was illuminated with an air conditioner as an example case, and the controller of the air conditioner was developed automatically using our proposed framework. Finally, an evolutionary simulation was performed to confirm our framework.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
[2] N. Sannomiya et al., *Genetic Algorithm and Optimization*, Asakura Publishing Co., Ltd., pp. 13, 1998. (in Japanese)
[3] Xilinx, *XC9572 In-System Programmable CPLD Product Specification*, Xilinx, pp. 2, 1998.
[4] P. O. Fanger, *Thermal Comfort*, McGraw-Hill, 1970.
[5] J. Koza, *Genetic Programming*, MIT Press, 1994.
[6] H. Iba, *Genetic Programming*, Tokyo Denki University Press, 1996. (in Japanese)
[7] H. Iba, *Introduction to Genetic Programming*, University of Tokyo Press, pp. 177, 2001. (in Japanese)
[8] T. Higuchi et al., *Evolvable Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine*, Proceedings of the 2nd International Conference on the Simulation of Adaptive Behavior, MIT Press, pp. 417, 1992. (in Japanese)
[9] H. Hemmi, J. Mizoguchi and K. Shimohara, *AdAM: A Hardware Evolutionary System*, Proc. 1997 IEEE Conf. Evolutionary Computat.(ICEC'97), pp. 193-196, 1997.
[10] I. Kajitani and T. Higuchi, *Developments of Myoelectric Controllers for Hand Prostheses*, Proc. of the Myoelectric Controls Symposium 2005, pp. 107-111, 2005.
[11] H. Sakanash, M. Iwata and T. Higuchi, *Evolvable Hardware for Lossless Compression of Very High Resolution Bi-level Images*, IEEE Proceedings-Computers and Digital Techniques, Vol.151, No.4, pp.277-286, 2004.