18

VHDL Design Automation Using Evolutionary Computation

Kazuyuki Kojima
Saitama University
Japan

1. Introduction

This chapter describes the automatic generation method of VHDL which lays out the control logic of a control system. This framework releases a designer from the work of describing the VHDL directly. Instead, the designer inputs the equation of motion of a system and target operation.

In this chapter, first, FPGA, CPLD, VHDL and evolutionary computation are outlined. This is basic knowledge required for an understanding of this chapter. Next, the framework of automatic generation of FPGA using evolutionary computation is described. VHDL description is expressed by several kinds of data structures called a chromosome. VHDL expressed in a chromosome is changed using evolutionary computation and changes to a more suitable code for controller purposes. Finally two example applications are shown. The first one is the controller for a simple inverted pendulum. After that, the framework is applied to a more complicated system, an air-conditioning system. The simulation results show that the controller automatically generated using this framework can control the system appropriately.

2. Computer-aided controller design using evolutionary computation

2.1 FPGA/CPLD/ASIC and VHDL

CPLDs and FPGAs are both sorts of programmable LSIs. 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. However, the proposed framework is applicable to all devices that can be designed by HDL. 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.

```
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;
```

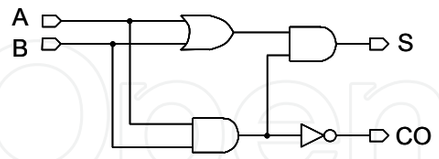


Fig. 1. VHDL for simple logical circuit

2.2 Genetic algorithm

The genetic algorithm used as a basis of this framework is outlined below (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 symbol string is a chromosome consisting of N loci. s_j is a gene in the j-th locus and value s_j is an allelomorph. The value is assumed to be a real number, a mere notation, and so on of the group of a certain integer or a certain range of observations as an allelomorph. The population consists of K individuals expressed with Eq. (1). Individual population $p(n)$ in generation n changes to individual population $p(n+1)$ in 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 (Goldberg, 1989; Koza, 1994).

2.3 Evolvable hardware

Higuchi et al. proposed evolvable hardware that regards the architecture bit of CPLD as a chromosome of a genetic algorithm to find a better hardware structure by genetic algorithm (Higuchi et al., 1992). They applied this to myoelectric controllers for electrical prosthetic hands, image data compression and so on (Kajitani et al., 2005; Sakanasi et al., 2004). This approach features coding of an architecture bit directly for a chromosome. The designer must determine which CPLD is used beforehand, implicitly determining the meaning of the architecture bit. If hardware is changed for the problem of circuit scale or structure, it is necessary to recalculate evolutionary computation.

Henmi et al. evolved a hardware description language (HDL) called structured function description language (SFL) and applied it to robot walking acquisition (Hemmi et al., 1997).

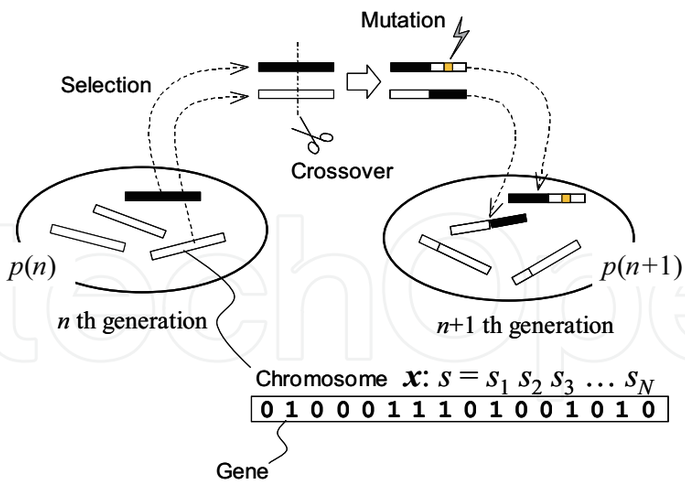


Fig. 2. Genetic algorithm overview

The basic motion, called an "action primitive", must be designed in a binary string. In our approach, the designer needs only to define I/O pins for CPLD. VHDL rather than an architecture bit is coded directly onto the chromosome, so the chromosome structure does not depend on CPLD scale or type, and after VHDL is generated automatically, CPLD is selected appropriately to the VHDL scale. Pin assignment is set after VHDL generation.

2.4 Controller design framework using evolutionary computation

The study of optimizing rewritable logical-circuit ICs, such as CPLD, using a genetic algorithm has increased. The framework that changes an internal logic circuit IC configuration to attain evolution is called evolvable hardware (EHW). With this framework, the designer needs only to define the criteria used to evaluate a controller. We explain the controller design framework using evolutionary computation with XC95144 (Xilinx, 1998) as the test device. Internal blocks of XC95144 are shown in Fig. 3. XC95144 is a small CPLD that has 144 pins (117 user input-outputs), 144 macro cells and 3200 usable gates. A designer chooses input and output signals from 117 user I/Os and assigns these pins. Each signal is defined for each I/O. If CPLD is used in control, sensors and actuators can be associated with CPLD I/O pins (Fig. 4). Here, I/O pins are associated with two sensors and one actuator. The sensor values are inputs to CPLD as two 16-bit digital signals and a 10-bit digital signal is output as a reference signal to the actuator.

VHDL, which describes internal CPLD logic, is encoded on a chromosome. An example of the VHDL generated is shown in Fig. 5, corresponding to Fig. 4. This VHDL consists of three declarations — (a) entity declaration, (b) signal declaration, and (c) architecture body. CPLD I/O signals are defined in (a) and internal CPLD signals are in (b). Signals mainly used in VHDL are a `std_logic` type and a `std_logic_vector` type. The `std_logic` type is used when dealing with a signal alone and the `std_logic_vector` type is used when dealing with signals collectively. The `std_logic` and the `std_logic_vector` types should be combined to optimize maintenance and readability. A description of VHDL can be restored if all of the input output signals and internal signals are used as the same `std_logic` type and only the number is encoded on a chromosome. The number of input signals, output

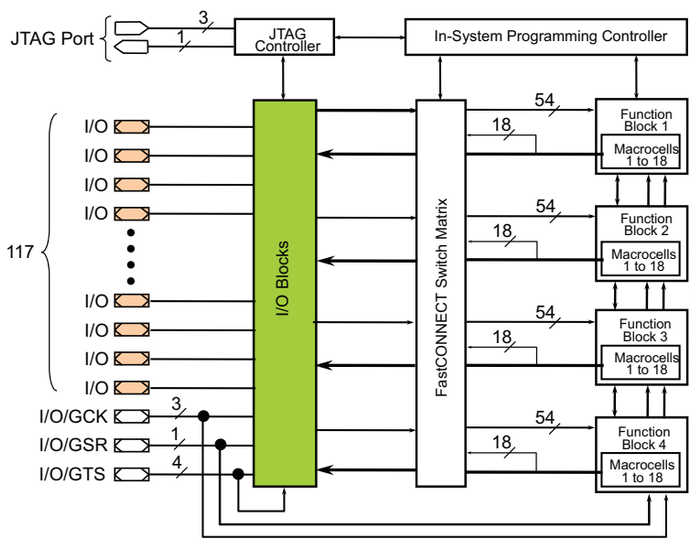


Fig. 3. XC95144 architecture

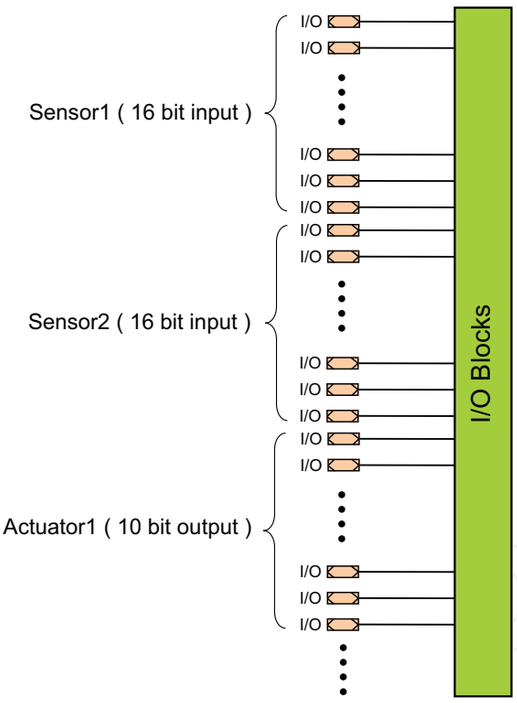


Fig. 4. Example of CPLD application

signals and internal signals are encoded on the head of a chromosome (Fig. 6). In Fig. 4, two input signals are set up with 16 bits and an output signal with 10 bits.

A chromosome that represents a VHDL assignment statement is shown in Fig. 7. A chromosome structure corresponding to a process statement is shown in Fig. 8. The value currently described is equivalent to the process statement in which "`DI009`" and "`DI014`" are enumerated in the sensitivity list.

```
000: --------------------------------------------------------------------------------
001: -- VHDL for I/P controller generated by _hiGA_AC with _hiGA.dll
002: --     (c) 2011, Saitama Univ., Human Interface Lab., programmed by K.Kojima
003: --     Chromo#:E/50 Length:13 Inputs:32 Outputs:10 Signals:2
004: --     Simulation: Initial condition: 180.0, dt=0.0100
005: --------------------------------------------------------------------------------
006: library IEEE;
007: use IEEE.std_logic_1164.all;
008: use IEEE.std_logic_arith.all;
009: use IEEE.std_logic_unsigned.all;
010:
011: entity GA_VHDL is
012:     port(
013:         DI000 : in std_logic;
014:         DI001 : in std_logic;
015:         DI002 : in std_logic;
016:         DI003 : in std_logic;
017:         DI004 : in std_logic;
018:         DI005 : in std_logic;
019:         DI006 : in std_logic;
020:         DI007 : in std_logic;
021:         DI008 : in std_logic;
022:         DI009 : in std_logic;
023:         DI010 : in std_logic;
024:         DI011 : in std_logic;
025:         DI012 : in std_logic;
026:         DI013 : in std_logic;
027:         DI014 : in std_logic;
028:         DI015 : in std_logic;
029:         DI016 : in std_logic;
030:         DI017 : in std_logic;
031:         DI018 : in std_logic;
032:         DI019 : in std_logic;
033:         DI020 : in std_logic;
034:         DI021 : in std_logic;
035:         DI022 : in std_logic;
036:         DI023 : in std_logic;
037:         DI024 : in std_logic;
038:         DI025 : in std_logic;
039:         DI026 : in std_logic;
040:         DI027 : in std_logic;
041:         DI028 : in std_logic;
042:         DI029 : in std_logic;
043:         DI030 : in std_logic;
044:         DI031 : in std_logic;
045:         DO000 : out std_logic;
046:         DO001 : out std_logic;
047:         DO002 : out std_logic;
048:         DO003 : out std_logic;
049:         DO004 : out std_logic;
050:         DO005 : out std_logic;
051:         DO006 : out std_logic;
052:         DO007 : out std_logic;
053:         DO008 : out std_logic;
054:         DO009 : out std_logic
055:     );
056: end GA_VHDL;
057:
```

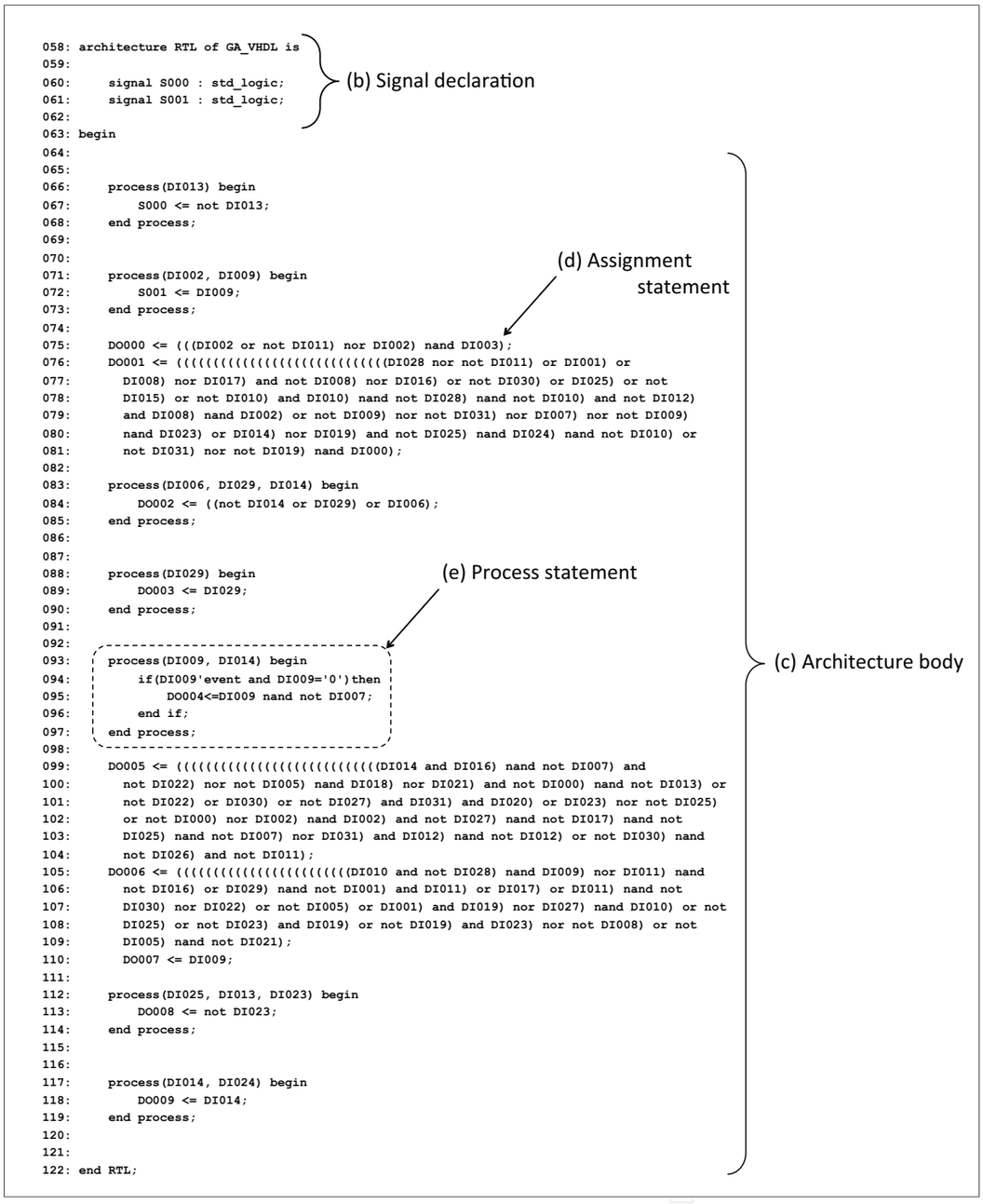
(a) Entity declaration

Fig. 5. (a) Automatically generated VHDL (List-1)

The VHDL description has an if-statement inside of a process statement and the description has two nesting levels. The hierarchy of the list structure is deep compared to the substitution statement. When the gene of such a multilist structure is prepared, it is possible to represent various VHDL expressions.

2.5 Variable length chromosome and genetic operations

The structure of a chromosome changes with the control design specification. The number of internal signals is set arbitrarily and different descriptions in VHDL are expressed with different locus lengths. The length of a chromosome is determined by the VHDL line count. The length determines the number of internal signals enumerated on the sensitivity list or




```
058: architecture RTL of GA_VHDL is
059:
060:     signal S000 : std_logic;
061:     signal S001 : std_logic;
062:
063: begin
064:
065:
066:     process(DI013) begin
067:         S000 <= not DI013;
068:     end process;
069:
070:
071:     process(DI002, DI009) begin
072:         S001 <= DI009;
073:     end process;
074:
075:     DO000 <= (((DI002 or not DI011) nor DI002) nand DI003);
076:     DO001 <= ((((((((((((((((((((((((((((((DI028 nor not DI011) or DI001) or
077:       DI008) nor DI017) and not DI008) nor DI016) or not DI030) or DI025) or not
078:       DI015) or not DI010) and DI010) nand not DI028) nand not DI010) and not DI012)
079:       and DI008) nand DI002) or not DI009) nor not DI031) nor DI007) nor not DI009)
080:       nand DI023) or DI014) nor DI019) and not DI025) nand DI024) nand not DI010) or
081:       not DI031) nor not DI019) nand DI000);
082:
083:     process(DI006, DI029, DI014) begin
084:         DO002 <= ((not DI014 or DI029) or DI006);
085:     end process;
086:
087:
088:     process(DI029) begin
089:         DO003 <= DI029;
090:     end process;
091:
092:
093:     process(DI009, DI014) begin
094:         if(DI009'event and DI009='0')then
095:             DO004<=DI009 nand not DI007;
096:         end if;
097:     end process;
098:
099:     DO005 <= ((((((((((((((((((((((((((((((DI014 and DI016) nand not DI007) and
100:       not DI022) nor not DI005) nand DI018) nor DI021) and not DI000) nand not DI013) or
101:       not DI022) or DI030) or not DI027) and DI031) and DI020) or DI023) nor not DI025)
102:       or not DI000) nor DI002) nand DI002) and not DI027) nand not DI017) nand not
103:       DI025) nand not DI007) nor DI031) and DI012) nand not DI012) or not DI030) nand
104:       not DI026) and not DI011);
105:     DO006 <= ((((((((((((((((((((((((((((DI010 and not DI028) nand DI009) nor DI011) nand
106:       not DI016) or DI029) nand not DI001) and DI011) or DI017) or DI011) nand not
107:       DI030) nor DI022) or not DI005) or DI001) and DI019) nor DI027) nand DI010) or not
108:       DI025) or not DI023) and DI019) or not DI019) and DI023) nor not DI008) or not
109:       DI005) nand not DI021);
110:       DO007 <= DI009;
111:
112:     process(DI025, DI013, DI023) begin
113:         DO008 <= not DI023;
114:     end process;
115:
116:
117:     process(DI014, DI024) begin
118:         DO009 <= DI014;
119:     end process;
120:
121:
122: end RTL;
```

Fig. 5. (b) Automatically generated VHDL (List-2)

the length of the right-hand side of an assignment statement. When dealing with such a variable length chromosome, the problem is that genetic operations will generate conflict on a chromosome. To avoid this problem, we prepared the following restrictions:

1. With a top layer, the length of a chromosome is equal to the number that added one to the summary of the number of internal signals and the number of output signals.
2. All signals are encoded on a chromosome using a reference number.

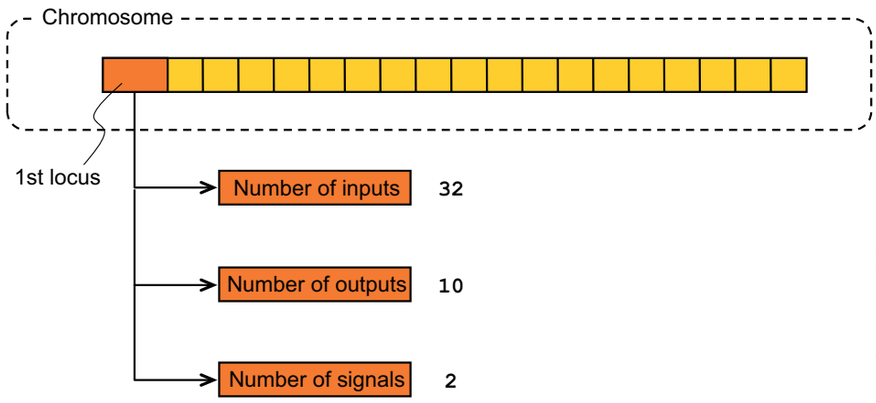


Fig. 6. Signal definition on the first locus

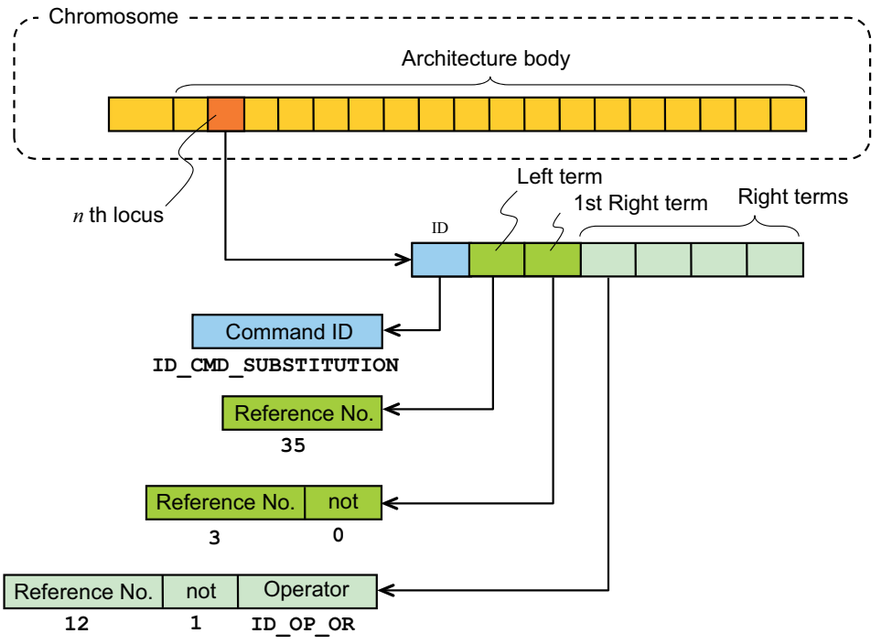


Fig. 7. Gene structure for assignment statement

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 a chromosome describes the entity declaration using all internal signals and output signals in order with a low reference number. Each signal can be used only once.
5. Crossover operates on the top layer of a chromosome.

These restrictions avoid the conflict caused by genetic operations.

Figure 9 shows an example of crossover operation. 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

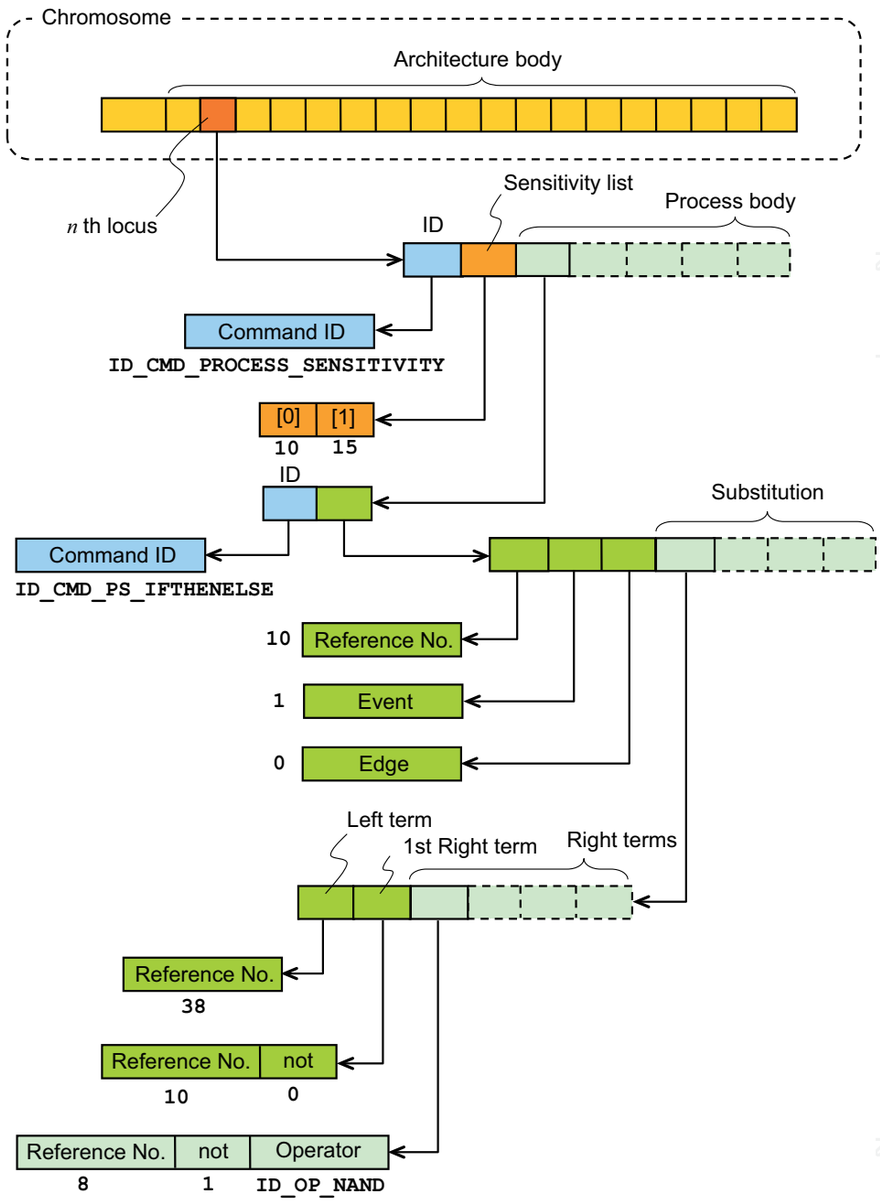


Fig. 8. Gene structure for if-then-else statement in process body

show 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 HDL-based controller of inverted pendulum

In this section an application to an HDL-based controller for the inverted pendulum is described (Kojima, 2011).

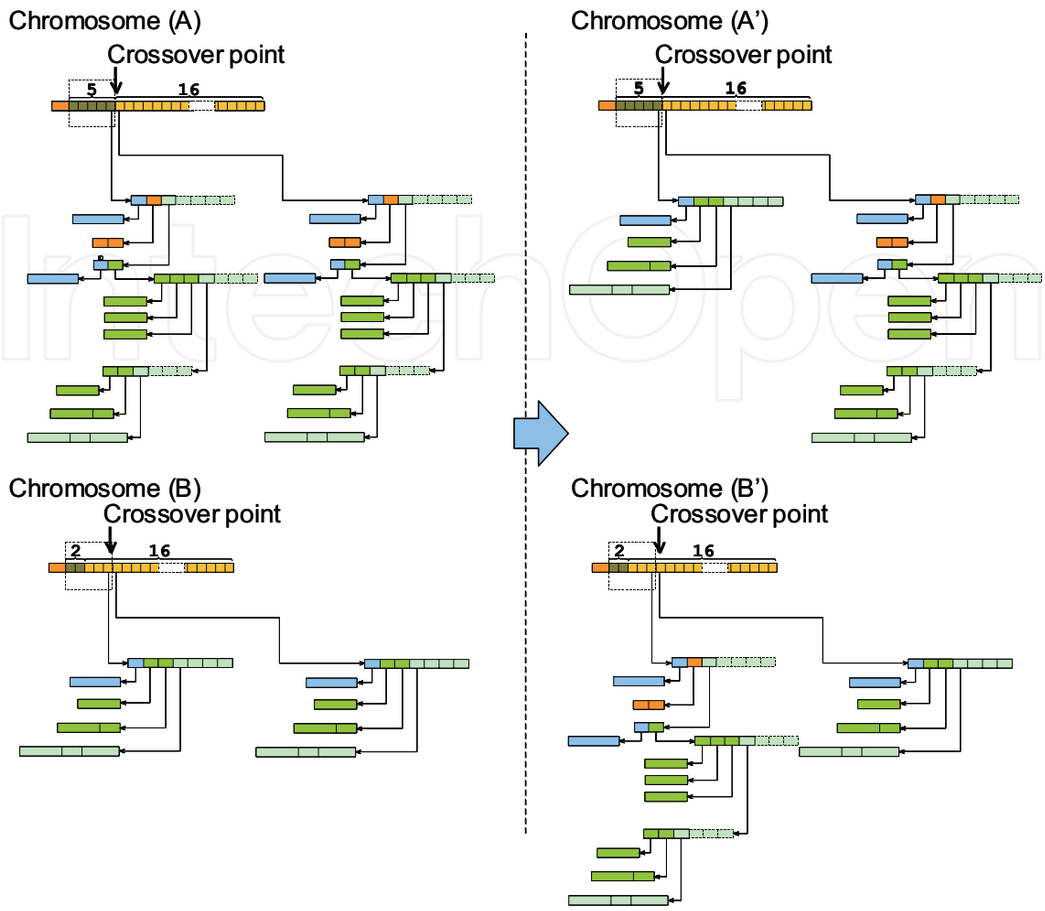


Fig. 9. Crossover operation

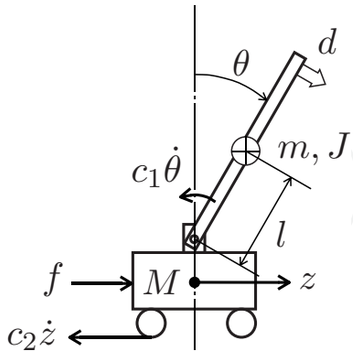


Fig. 10. Inverted pendulum

3.1 Equations of motion

Figure 10 shows the model of the inverted pendulum. The equations of motion are given by

$$(ml^2 + J)\ddot{\theta} = mgl\sin\theta - ml\ddot{z}\cos\theta - c_1\dot{\theta} + d \tag{2}$$

$$(M + m)\ddot{z} = f - c_2\dot{z} \tag{3}$$

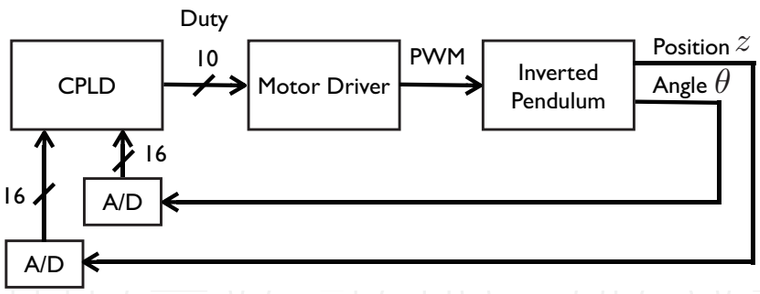


Fig. 11. Control system block diagram

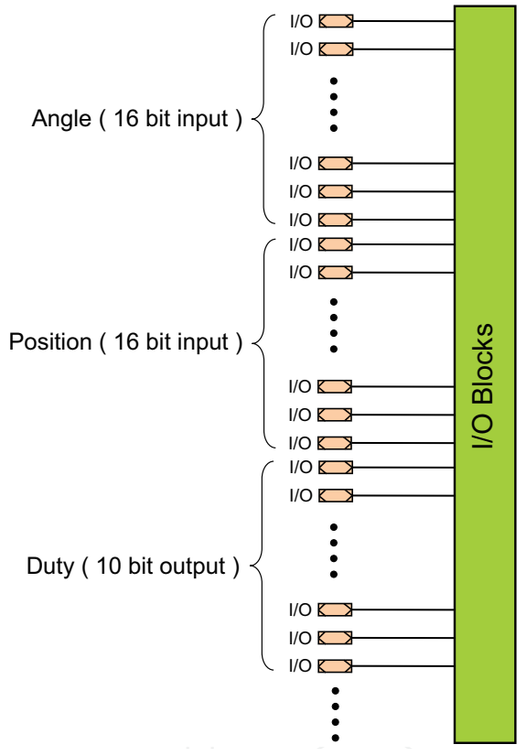


Fig. 12. CPLD pin assignment

$$f = Kv \tag{4}$$

where θ [rad] is the angle of the pendulum, m [kg] is the mass of the pendulum, M [kg] is the mass of the cart, J [kgm^2] is the inertia of the rod, l [m] is the length between the gravity point and fulcrum, z [m] is the position of the cart, d [Nm] is the disturbance torque, c_1 [Ns/rad] is the viscous resistance of the pendulum, c_2 [Ns/m] is the viscous resistance of the cart, f [N] is the controlling force, v is the parameter for the control input and K is the gain. $\bar{J}$, a and b are defined as:

$$\bar{J} = J + ml^2 \tag{5}$$

$$a = \frac{c_2}{M+m} \tag{6}$$

$$b = \frac{K}{M+m} \tag{7}$$

then, equations (2) and (3) become:

$$\ddot{\theta} = \frac{mgl}{\bar{J}} \sin\theta - \frac{ml\ddot{z}}{\bar{J}} \cos\theta - \frac{c_1}{\bar{J}}\dot{\theta} + \frac{d}{\bar{J}} \tag{8}$$

$$\ddot{z} = -a\dot{z} + bv. \tag{9}$$

Each parameter can be determined by measurement and experiment as $l = 0.15$, $m = 46.53 \times 10^{-3}$, $\bar{J} = 1.58 \times 10^{-3}$, $c_1 = 2.05 \times 10^{-2}$, $a = 4.44$ and $b = 2.46 \times 10^{-1}$. Using these parameters, evolutionary simulations are conducted.

3.2 Control system and CPLD pin assignment

Figure 11 shows the control system block diagram. The CPLD is used as the controller of the inverted pendulum. The position of the cart and the angle of the pendulum are converted 16 bit digital signals respectively and input to the 32 CPLD pins. The control logic in the CPLD, which is formed using the framework previously mentioned, determines the 10 bit control signal driving the motor of the inverted pendulum. Figure 12 shows the CPLD pin assignment for this application. In this case, 32 bit parallel inputs and 10 bit parallel outputs are adopted. Instead of using them, we can use serial I/Os connected with A/D converters and D/A converters. In this case, serial to parallel logics should be formed in the CPLD and even though serial inputs are used, automatically generated VHDL can be used as a VHDL component.

3.3 Fitness

The fitness value is calculated as a penalty to the differences in the rod angle and the cart position.

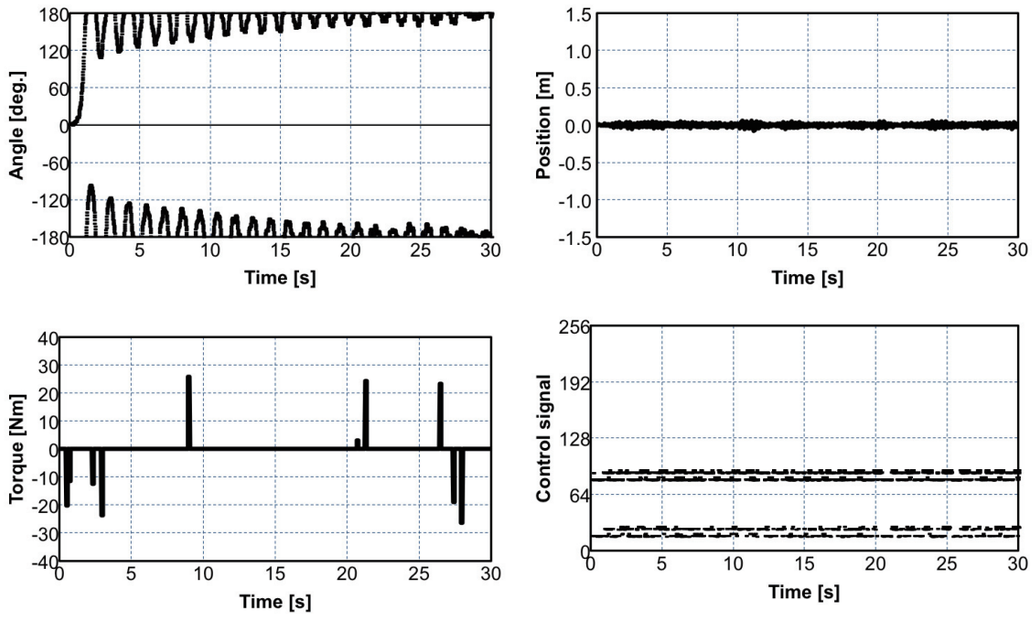
$$\text{fitness} = -\frac{1}{\pi}\int_0^{30} |\theta| dt - \int_0^{30} |z| dt \tag{10}$$

Disturbances are given as a random torque during the control simulation. When calculating fitness value, disturbance torque is always initialized. Therefore, all individuals are given different disturbance at each evaluation. This kind of disturbance makes the controller robust to various disturbances.

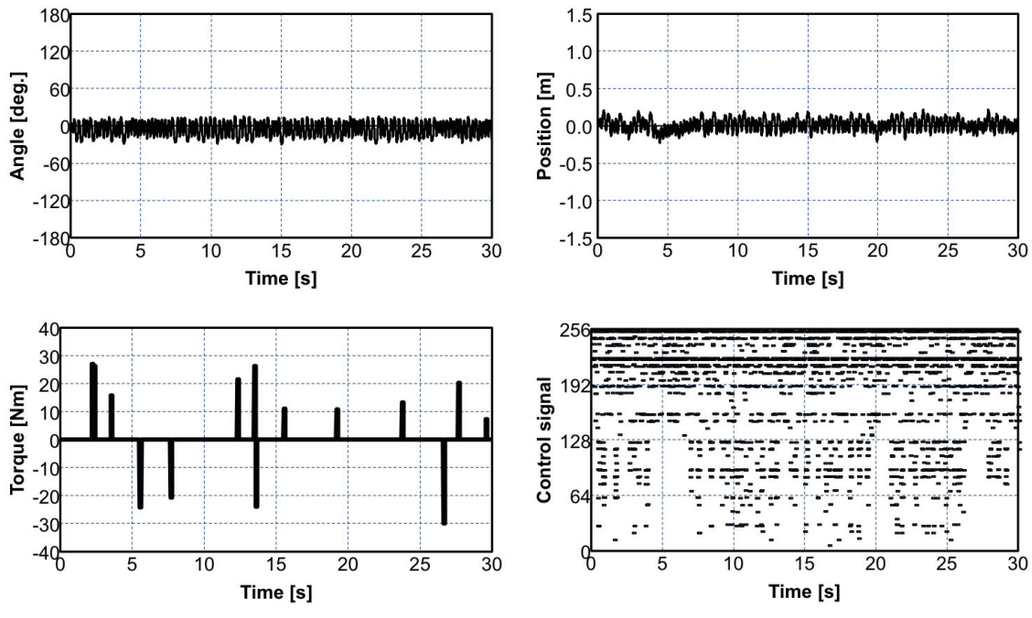
3.4 Simulation

Simulations are conducted under two conditions – (1) $\theta_0 = 1^\circ, z_0 = 0$, (2) $\theta_0 = 180^\circ, z_0 = 0$. Population size is 50, mutation rate is 0.5, crossover rate is 1.0, tournament strategy, tournament size is 10 and the elite strategy is adopted.

Figures 13 and 14 show the simulation results. Figure 13 shows the result of condition (1), Figure 14 shows the result of condition (2). (a) the result at 0 generation and (b) the result at 1000 generation are represented respectively. The angle of the rod, the position of the cart, disturbance and the control signal are shown at each generation. At zero generation, in both conditions (1) and (2), the obtained controller cannot control the inverted pendulum adequately. The rod moves in a vibrating manner at around 180°. At 1000 generation, the controller controls the inverted pendulum successfully in both conditions. Further, in condition (2), swing up motion can be observed (Fig.14(b)). The control signal at 1000 generation has more various patterns than the signal at 0 generation.

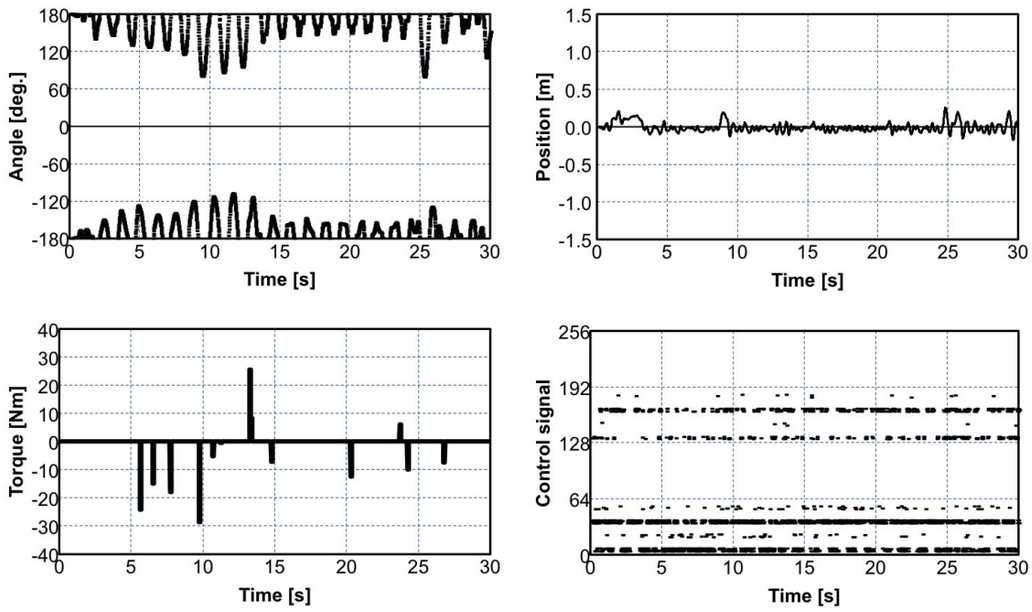


(a) 0th generation

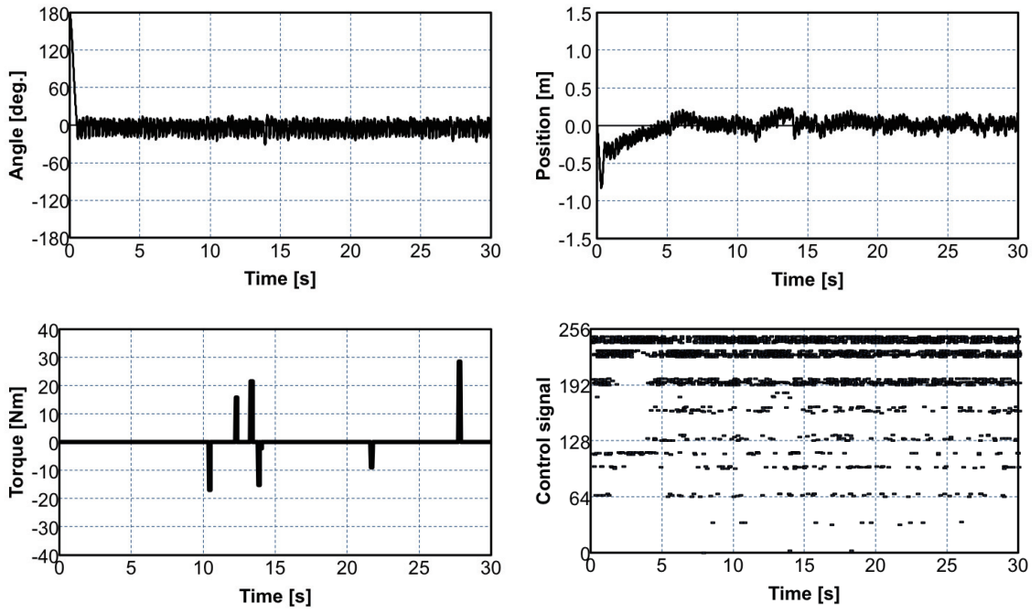


(b) 1000th generation

Fig. 13. Simulation results ($\theta_0 = 1^\circ$, $z_0 = 0$)



(a) 0th generation



(b) 1000th generation

Fig. 14. Simulation results ($\theta_0 = 180°$, $z_0 = 0$)

4. Application to an air-conditioning controller

Next, the framework is applied to an air-conditioning controller. Simulation model, task definition and fitness function are described in this section. Simulation results will be shown at the end of the section (Kojima et al., 2007; Kojima, 2009).

4.1 Simulation model

In the targeted air-conditioning system(Fig. 15), the outside air imported is cooled or warmed and sent to the console to adjust temperature. Air entering from the inlet (a) is cooled by the refrigerator (c). Part of the cooled air is warmed with a heater (e), then mixed with cool air to adjust the temperature. The angle of the mixture door (d) controls the mixing ratio of warm and cold air. Mixed air is sent to console (f), changing the indoor temperature. The system controller controls blower motor rotation and the angle of the air mixture door.

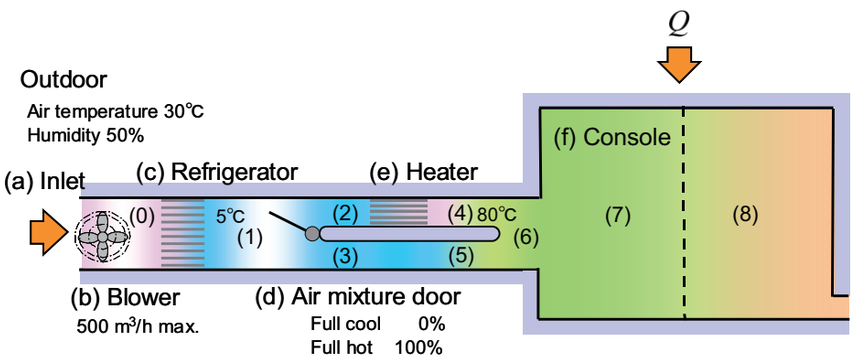


Fig. 15. Air-conditioning system

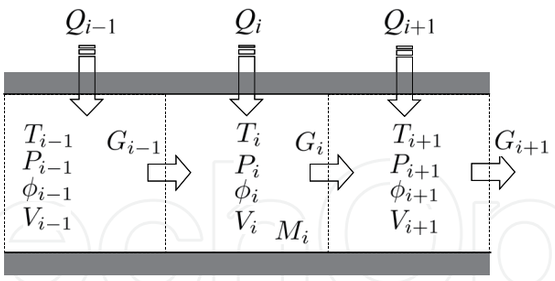


Fig. 16. Control volumes

To evaluate the controller performance, an air-conditioning simulation model is required in evolutionary computation. We consider an air-conditioning model combining the console and duct using nine control volumes (Fig. 16). Temperature and humidity, but not compressibility, are considered in this model. To calculate the predicted mean vote (PMV), which indicates thermal comfort, we require globe temperature, metabolization, flow rate, insulation of clothes and external work.

Figure 16 shows three control volumes in this system. We assume that air flows toward the control volume indexed with $(i+1)$ from the control volume indexed with (i). Air temperature T_{i+1}[K] and relative humidity ϕ_{i+1} vary with temperature T_i[K], relative

humidity ϕ_i, mass flow G_i[kg/s] and heat transfer Q[kJ/s]. In air-conditioning simulation, mass flow is proportional to the opening of blower motor α.

$$G_0 = G_i = \alpha G_{max} \ (0 \leq \alpha \leq 1) \tag{11}$$

where G_{max} is maximum flow at full blower opening. Flow rate G_0 is the sum of air flow G_{a0} and steam flow G_{w0}.

$$G_0 = G_{a0} + G_{w0} \tag{12}$$

Water vapour pressure P_{w0} is as follows:

$$P_{w0} = \phi_0 \cdot P_{s0} \tag{13}$$

where P_{s0} saturation water vapour pressure from Tetens' formula (Tetens, 1930):

$$P_{s0} = 610.78 \times e^{\frac{17.2694(T_0 - 273.15)}{(T_0 - 273.15) + 238.3}} \tag{14}$$

Specific humidity x_0 is given by water vapour pressure P_{w0}.

$$x_0 = 0.622 \frac{P_{w0}}{P_0 - P_{w0}} \tag{15}$$

Air mass flow G_{a0} is calculated from the following gas equation:

$$G_{a0} = P_{a0} \cdot \frac{V}{R_a T_0} \tag{16}$$

where R_a is a gas constant. Water vapour mass flow G_{w0} is given by:

$$G_{w0} = x_0 \cdot G_{a0} \tag{17}$$

Air flow rate G_a is constant. Steam flow G_w is constant except during dehumidification. Considering the air flow into control volume (i) from control volume $(i-1)$ in unit time dt[s], temperature T_i', humidity x_i' and mass of control volume M_i' after dt is given as follows:

$$T_i' = \frac{G_{i-1} C_{i-1} T_{i-1} dt + (M_i - G_i dt) C_i T_i}{G_{i-1} C_{i-1} dt + (M_i - G_i dt) C_i} + \frac{Q_i dt}{M_i C_i} \tag{18}$$

$$x_i' = \frac{M_i x_i + (1 + x_i)(G_{wi-1} - G_{wi}) dt}{M_i + (1 + x_i)(G_{ai-1} - G_{ai}) dt} \tag{19}$$

$$M_i' = M_i + (G_{i-1} - G_i) dt \tag{20}$$

where specific heat C_i[kJ/kg $\cdot$ K] is given by

$$C_i = \frac{1.005 + x_i \{(2501.6/T_i) + 1.859\}}{1 + x_i} \tag{21}$$

At the mixture door, air is divided into two flows. The ratio of the mass of divided air depends on the mixture door angle. Here G_1 is the mass flow at location (1) in Fig. 15, G_2 that at location (2) and G_3 that at location (3),

$$G_2 = \beta G_1 \tag{22}$$

$$G_3 = (1-\beta)G_1 \tag{23}$$

where β $(0 \leq \beta \leq 1)$ is the opening ratio of the mixture door. Adding two mass flow rates enables us to calculate the downstream mass at a juncture. Here G_4 is the mass flow at location (4) in Fig.15, G_5 that at location (5) and G_6 that at location (6),

$$G_6 = G_4 + G_5 \tag{24}$$

Temperature and humidity at a juncture are given in the same way as for when three control volumes are considered.

4.2 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 (Fanger, 1970). 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[°C], mean radiant temperature T_r[°C], relative air speed v[m/s] and vapour pressure of water vapour P[hPa].

$$\begin{aligned}
\mathrm{PMV} = {} & \{0.33\exp(-0.036M) + 0.028\}\Big[(M-W) \\
& -3.05\{5.73 - 0.007(M-W) - P\} \\
& -0.42\{(M-W) - 58.1\} \\
& -0.0173M(5.87 - P) \\
& -3.96\times 10^{-8} f_{cl}\{(T_{cl}+273.15)^4 \\
& -(T_{mrt}+273.15)^4\} \\
& -f_{cl}h_c(T_{cl}-T_a)\Big]
\end{aligned} \tag{25}$$

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

$$\begin{aligned}
f_{cl} &= 1.0 + 0.2I_{cl}(I_{cl} \leq 0.5) \\
f_{cl} &= 1.05 + 0.1I_{cl}(I_{cl} > 0.5)
\end{aligned} \tag{26}$$

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

$$\begin{aligned}
T_{cl} = {} & 35.7 - 0.028(M-W) \\
& -0.155I_{cl}\Big[3.96\times 10^{-8} f_{cl}\{(T_{cl}+273.15)^4 \\
& -(T_{mrt}+273.15)^4\} + f_{cl}h_c(T_{cl}-T_a)\Big]
\end{aligned} \tag{27}$$

where h_c is the heat transfer coefficient,

$$h_c = \max\{2.38(T_{cl} - T_a)^{0.25}, 0.0121\sqrt{v}\} \tag{28}$$

and T_{mrt} is mean radiant temperature. PMV is detailed in (Fanger, 1970).

4.3 Task definition

The task is to adjust PMV in the console despite heat transfer from outside changes. The air-conditioning controller controls the opening ratio of blower n and the opening ration of mixture door m appropriately.

4.4 Control system and CPLD pin assignment

Figure 17 shows the control system block diagram. PMV is input to the CPLD as a 8 bit signal. The control logic in the CPLD determines the 8 bit blower opening control signal and the 8 bit mixture door control signal. Figure 18 shows the CPLD pin assignment for this application.

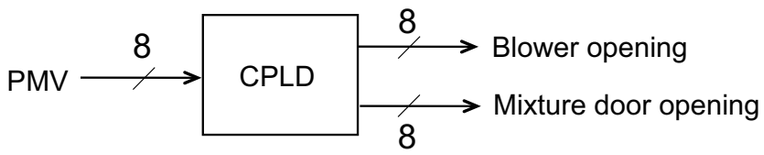


Fig. 17. Control system block diagram

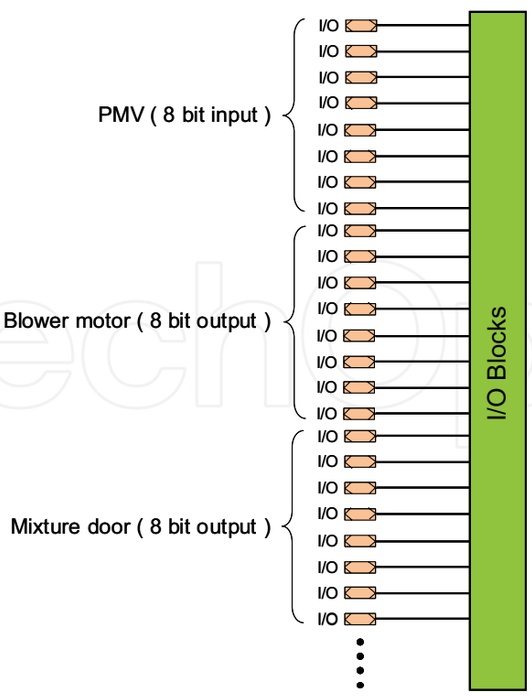


Fig. 18. Control system block diagram

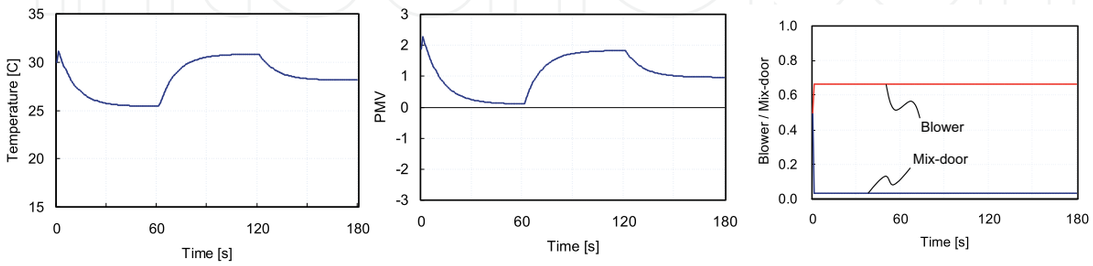
4.5 Fitness

The fitness function is as follows:

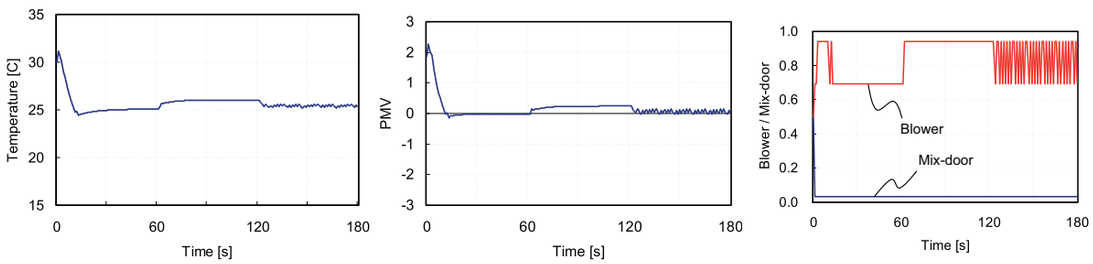
$$\text{fitness} = -\int_0^{t_{end}} |\text{PMV}_{\text{ctrl}} - \text{PMV}_{\text{target}}| dt \tag{29}$$

where t_{end} is the end of simulation time. The difference between target and controlled PMV is integrated as a penalty in the controller simulation.

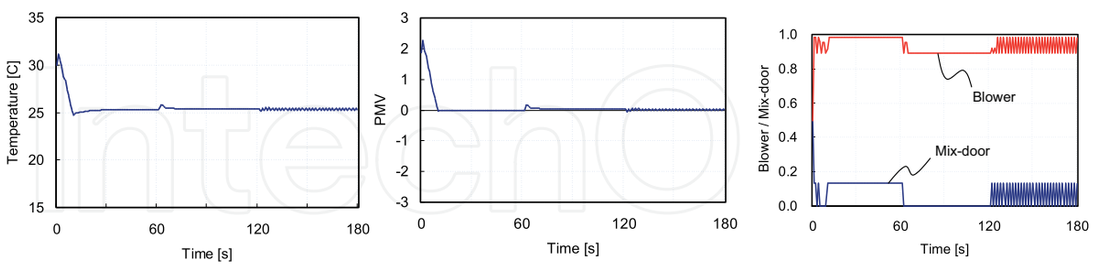
4.6 Simulation results



(a) 0th generation



(b) 100th generation



(c) 10000th generation

Fig. 19. Simulation results

Figure 19 shows the simulation results. PMV in a console is fed back to the controller. In the graphs, trends change every 60 seconds. Variations are based on the load effect change each 60 seconds. At zero generation (Fig. 19 (a)), temperature rises or falls with the change in heat load. PMV also changes simultaneously. This means that simply optimizing a controller is not enough. After 100 generations of calculation, the difference between the target and the

estimated value decreases (Fig. 19 (b)). In the 10,000th generation (Fig. 19 (c)), tolerance decreases further. These results show that hardware corresponding to the purpose is obtained automatically using this framework.

Figure 20 shows simulation results under other conditions. Nine graphs result for three thermal loads. Three graphs — temperature, PMV and control — are shown for each thermal condition. The controller used under three conditions was obtained after 10,000 generations of calculation. Although thermal load and load change timing were random, the blower and mix door were controlled so that PMV is set to zero.

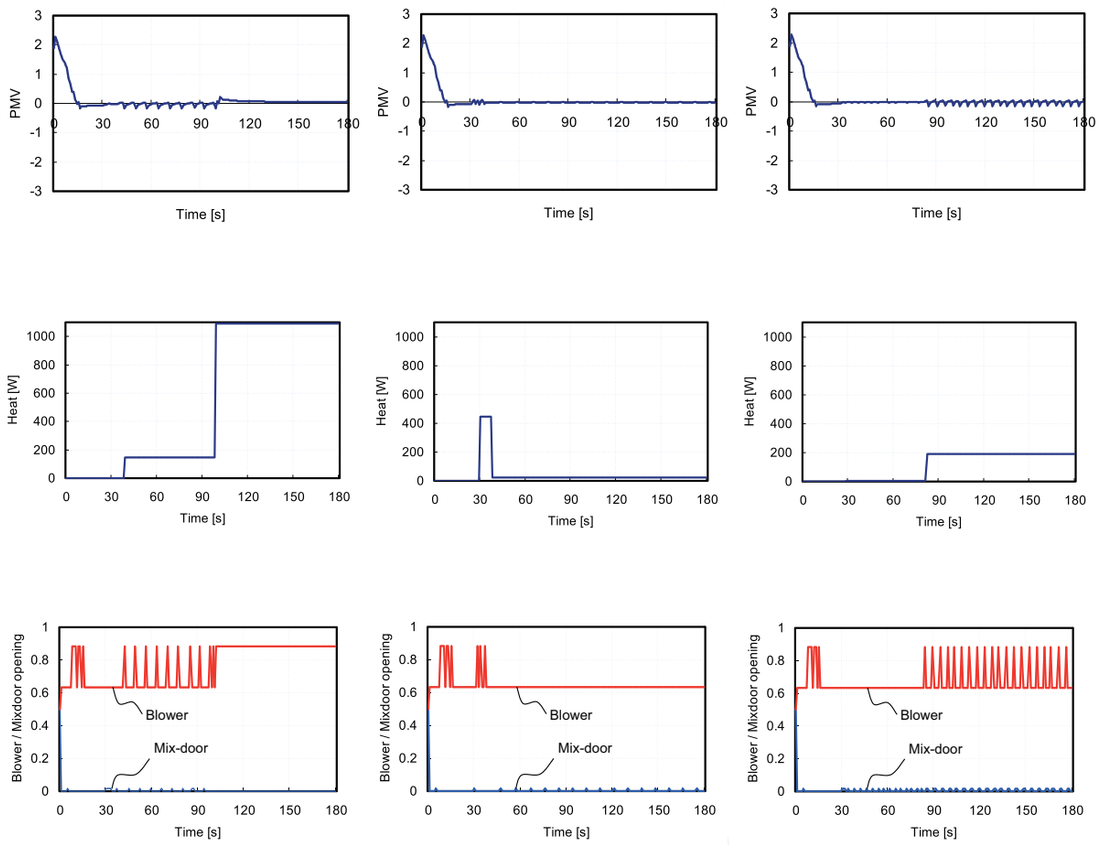


Fig. 20. Simulation results

5. Conclusion

In this chapter, to automate controller design, CPLD was used for controller data-processing and VHDL to describe the logical circuit was optimized using the genetic algorithm. Two example cases (an inverted pendulum and an air-conditioner) were shown and we confirmed this framework was able to be applied to both systems. Since this framework is a generalized framework, so in these kinds of systems which process data from some sensors and drive some actuators, this framework will work functionally.

6. References

Balderdash, G., Nolfi, S., & Parisi, D. (2003). Evolution of Collective Behavior in a Team of Physically Linked Robots, *Proceedings of the Applications of Evolutionary Computing, EvoWorkshops 2003*, pp.581-592, Ti£¡bingen, April 2009.

Barate, R. & Manzanera, A. (2008). Evolving Vision Controllers with a Two-Phase Genetic Programming System Using Imitation, *Proceedings of the 10th international conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp.73-82, ISBN: 978-3-540-69133-4, Osaka, July 2008.

Fanger, P. O. (1970). *Thermal Comfort*, McGraw-Hill.

Goldberg, D. E. (1989). *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley.

Hemmi, H., et al., (1997). AdAM: A Hardware Evolutionary System, *Proc. 1997 IEEE Conf. Evolutionary Computat.(ICEC'97)*, pp.193-196.

Higuchi, T. et al. (1992). 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.417-424.

Kajitani,I. & Higuchi, T. (2005). Developments of Myoelectric Controllers for Hand Prostheses, *Proceedings of the Myoelectric Controls Symposium 2005*, pp.107-111, Fredericton, August 2005.

Kojima, K. et al. (2007). Automatic Generation of VHDL for Control Logic of Air-Conditioning Using Evolutionary Computation, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol.11, No.7, pp.1-8.

Kojima, K. (2009). VHDL Design Automation Using Evolutionary Computation, *Proceedings of 2009 International Symposium on Industrial Electronics (IEEE ISIE 2009)*, pp.353-358, Seoul, July 2009.

Kojima, K. (2011). Emergent Functions of HDL-based Controller of Inverted Pendulum in Consideration for Disturbance, *Proceedings of 2011 IEEE/SICE International Symposium on System Integration*, Kyoto, December 2011.

Koza, J. (1994). *Genetic Programming*, A Bradford Book, ISBN0262111705.

Sakanashi, H. et al., (2004). 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, Ibaraki, August 2004.

Tetens, O. (1930). *Uber einige meteorologische Begriffe, Zeitschrift fur Geophysik*, Vol.6, pp.297-309. (in German)

wyns10] Wyns, B., et al., (2010). Evolving Robust Controllers for Corridor Following Using Genetic Programming, *Proceedings of the International Conference on Agents and Artificial Intelligence*, volume 1 : artificial intelligence; pp.443-446, Valencia, January 2010.

Xilinx. (1998). *DS056(v.2.0) XC95144 High Performance CPLD Product Specification*, Xilinx; 2.

IntechOpen