รายการอ้างอิง

1. สมพงศ์ ฉัตรแสงอุทัย. การพัฒนาตัวควบคุมแบบลำดับที่โปรแกรมได้ขนาดเล็ก. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2532.
2. กฤษดา วิศวธีรานนท์. PC ตัวควบคุมซีเคว้นซ์ หลักการทำงานและการประยุกต์. พิมพ์ครั้งที่ 9. พิมพ์ที่โรงพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย: สำนักพิมพ์บริษัท บุญชัยวิศวกรรม จำกัด, 2541.
3. KEYENCE (THAILAND) CO.,LTD. ทำความรู้จักกับ PLC. INDUSTRIAL ปีที่ 8 ฉบับที่ 95 (เมษายน 2545) : 89.
4. กฤษดา วิศวธีรานนท์, สมบูรณ์ จงชัยกิจ, สิทธิพร ประวัติรุ่งเรือง. เครื่องควบคุมที่โปรแกรมได้ชนิดที่มีความสามารถด้านจัดการข้อมูล. รายงานผลการวิจัยและพัฒนาในโครงการหลักวงจรอิเลคทรอสิกส์เพื่ออุตสาหกรรม ห้องวิจัยวัดคุมทางอุตสาหกรรม ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2533.
5. Phil Melore. Your Personal PLC Tutor Site [Online]. Available from: http://www.plcs.net [2002, June 8].
6. วิริยะ กองรัตน์, วิทยา ทิพย์สุวรรณพร, สุพรรณ กุลพาณิชย์. เครื่องควบคุมแบบตรรกที่โปรแกรมได้. การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18, 2538.
7. Astron Logic Research & Development. เปิดโลก FPGA กับ WIZARD PLD-A01., 2545.
8. Jim Hamblen, Henry Owen, Sudhakar Yalamanchili. Introduction of Rapid Systems Prototyping into Undergraduate Computer Engineering Curricula. School of Electrical and Computer Engineering Georgia Institute of Technology Atlanta Georiga. IEEE, 1995.
9. Marek Perkowski. FPGA Computer Architectures. Department of Electrical Engineering Portland State University Oregon. IEEE.
10. ALTERA The programmable solutions company [Online]. Available from: http://www.altera.com [2002, July 5].
11. Charles H., Roth Jr. Digital System Design Using VHDL. The University of Texas at Austin, 1997.

12. อัษฎางค์ แทนสถิตย์. การพัฒนาทีวีไมโครคอนโทรลเลอร์ที่สามารถถอดรหัสคำบรรยายภาพไทย-อังกฤษแบบซ่อนได้. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2541.

13. บรรจง ปิยธำรง, วิชราภร หนูทอง. การออกแบบไมโครคอนโทรลเลอร์โดยใช้ FPGA. การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 21, 2541.

14. Miyazawa I., Nagao T., Fukagawa M., Itoh Y., Mizuya T., Sekiguchi T.. Implementation of Ladder Diagram for Programmable Controller Using FPGA. Faculty of Engineering Yokohama National University Japan, IEEE 1999.

15. OMRON [Online]. Available from: http://www.fa.omron.co.jp [2002, September 20].

16. FUJI Electric [Online]. Available form: http://www.fujielectric.com [2002, September 20].

17. MITSUBISHI ELECTRIC. Available from: http://www.meau.com/eprise/main/web_site_pages/public/products/product_selection_guide/plc-fx-family/p-plc-fx-family-fx1s [2002, September 20].

18. ประภาส จงสถิตย์วัฒนา. Computer Architecture: A Synthesis. เอกสารคำสอน ภาควิชาวิศวกรรม คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2544.

19. Gilles Michel. Programmable Logic Controllers Architecture and Application. Printed in Great Britain : Biddles Ltd, 1990.

20. International Electrotechnique International. CEI IEC 1131-3 Part 3 Programming languages International Standard. First edition 1993-03, 1993.

21. กฤษดา ใจเย็น, อรรถพล บุญยะโภคา, ชัยวัฒน์ ลิ้มพรจิตรวิไล. เรียนรู้และปฏิบัติการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ตอนุกรม. บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด, 2544.

22. OMRON. SYSMAC PROGRAMMABLE CONTROLLER OPERATION MANUAL, 1988.

23. FANUC. Programmable Controller Series 90-10A User's Manual. Fanuc GE Automation Asia, Ltd., 1990.

ภาคผนวก

ภาคผนวก ก
แสดงละเอียดของภาษา VHDL ที่ใช้ออกแบบ PLC คอนโทรลเลอร์

```
---                    Main Program for PLC Controller                    ---
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
LIBRARY lpm;
USE lpm.lpm_components.all;
------------------------------------------------------------
--- INPUT/OUTPUT PORT  CONNECT TO PIN ----
------------------------------------------------------------
ENTITY nplc1 IS
PORT(   clock,reset         : IN    std_logic;
        modeswitch          : IN    std_logic;                      -- Select Mode Run/Stop
        rx                  : IN    std_logic;                      -- Receive Serial Data From PC
        input               : IN    std_logic_vector(15 downto 0);  -- Input Form User
        watchdogout         : OUT   std_logic;                      -- Reset Watchdog
        led_run             : OUT   std_logic;                      -- Light on PLC in Rum Mode
        led_stop            : OUT   std_logic;                      -- Light on PLC in Stop Mode
        tx                  : OUT   std_logic;                      -- Send Serial Data to PC
        ce,oe,we,cbuffer    : OUT   std_logic;                      -- Control Signal EEprom
        addressout          : OUT   std_logic_vector(12 downto 0);  -- Address EEprom
        output              : OUT   std_logic_vector(15 downto 0);  -- Output to User
        dataout             : INOUT std_logic_vector(7 downto 0) );  -- Data IN/OUT from EEprom
END nplc1;
------------------------------------------------------------
ARCHITECTURE a OF nplc1 IS
------------------------------------------------------------
---        Write or Read Data EEprom        ---
------------------------------------------------------------
COMPONENT wreerom
PORT(   clock               : IN    std_logic;
        reset               : IN    std_logic;
        writeread_flag      : IN    std_logic;      -- Tell wreerom part 1 : write ... 0 : Read...1
        wr_start_flag       : IN    std_logic;      -- Tell wreerom part to start process Write or Read
        addressrom          : IN    STD_LOGIC_VECTOR(12 downto 0);  -- Send Address to wreerom
        datawrite           : IN    std_logic_vector( 7 downto 0);  -- Main Send data to Write
        ce                  : OUT   STD_LOGIC := '1';               -- Control Signal EEprom
        oe                  : OUT   std_logic   := '1';             -- Control Signal EEprom
        we                  : OUT   std_logic   := '1';             -- Control Signal EEprom
        cbuffer             : OUT   std_logic;                      -- Assign Buffer Direction
        wr_cpl_flag         : OUT   std_logic   := '0';             -- wreerom tell Write or Read Complete
```

```
        addressout      : OUT    STD_LOGIC_VECTOR(12 downto 0);
        dataread        : OUT    std_logic_vector( 7 downto 0);  -- wreerom send read Data back to Main
        dataout         : INOUT  std_logic_vector( 7 downto 0) );
-------------------------------------------------------------
---        Transmition Part PLC to PC        ---
-------------------------------------------------------------
COMPONENT transmit
PORT(   clock           : IN  std_logic;
        reset           : IN  std_logic;
        tdr_serial      : IN  std_logic_vector(7 downto 0);
        requestsend_flag : IN  std_logic;
        tx              : OUT std_logic;
        sendready_flag  : OUT std_logic );
END COMPONENT;
-------------------------------------------------------------
---        Receive Part Form PC to PLC       ---
-------------------------------------------------------------
COMPONENT receive
PORT(   clock           : IN  std_logic;
        reset           : IN  std_logic;
        getcomplete_flag : IN  std_logic;
        rx              : IN  std_logic;
        rdr_serial      : OUT std_logic_vector(7 downto 0);
        receive_flag    : OUT std_logic );
END COMPONENT;
-------------------------------------------------------------
---            Watch Dog Timer               ---
-------------------------------------------------------------
COMPONENT watchdog
port(   clock           : IN    std_logic;
        reset           : IN    std_logic;
        rswatchdog      : IN    std_logic;
        watchdogout     : OUT   STD_LOGIC := '1' );
END COMPONENT;
-------------------------------------------------------------
TYPE STATE_TYPE IS (reset_pc,modeselect,fetch,decode,decode2,execute_out,updateinput,updateoutput,
                    updateoutput2, saveoutput,saveoutput2, execute_ld,execute_and,execute_setrst,
                    execute_tim,execute_cnt,execute_cnt2,execute_cnt3, execute_data,execute_data2,
                    execute_data3,execute_mov,execute_add,execute_sub,execute_cmp,saveprogram1,
                    saveprogram2,saveprogram3,saveprogram4,saveprogram5, savetomem1,savetomem2,
                    savetomem3,readprogram1,readprogram2,readprogram3,monitor,monitor2 );
```

```
SIGNAL state: STATE_TYPE;
SIGNAL memory_data_register                 : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL register_AC                          : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL addressrom                           : STD_LOGIC_VECTOR(12 DOWNTO 0);
SIGNAL rdr_serial,tdr_serial                : STD_LOGIC_VECTOR( 7 DOWNTO 0);
SIGNAL program_counter                      : STD_LOGIC_VECTOR( 8 DOWNTO 0);
SIGNAL memory_address_register              : STD_LOGIC_VECTOR( 8 DOWNTO 0);
SIGNAL datawrite,dataread                   : STD_LOGIC_VECTOR( 7 DOWNTO 0);
SIGNAL timebase                             : STD_LOGIC_VECTOR( 2 DOWNTO 0);
SIGNAL requestsend_flag, sendready_flag     : STD_LOGIC;
SIGNAL getcomplete_flag,receive_flag        : STD_LOGIC;
SIGNAL wr_start_flag,writeread_flag,wr_cpl_flag : STD_LOGIC := '0';
SIGNAL dclock                               : STD_LOGIC;
SIGNAL memory_write                         : STD_LOGIC;
SIGNAL rswatchdog                           : STD_LOGIC;
BEGIN
memory : lpm_ram_dq          -- Internal Memory 16 * 256 --
        GENERIC MAP(lpm_widthad => 9,
                                lpm_outdata => "UNREGISTERED",
                                lpm_indata  => "REGISTERED",
                                lpm_address_control => "UNREGISTERED",
                                lpm_file    => "end.mif",
                                lpm_width=> 16)
        -- Control Register/Signal Internal Memory --
        PORT MAP(data => Register_AC,address=>memory_address_register,
                        we=>memory_write,inclock=>dclock,q=>memory_data_register);
stage0 : transmit    PORT MAP( clock ,reset ,tdr_serial ,requestsend_flag, tx ,sendready_flag);
stage1 : receive     PORT MAP( clock ,reset ,getcomplete_flag ,rx ,rdr_serial ,receive_flag);
stage2 : wreerom     PORT MAP( clock ,reset ,writeread_flag ,wr_start_flag ,addressrom ,datawrite,
                        ce,oe ,we ,cbuffer ,wr_cpl_flag ,addressout ,dataread ,dataout);
stage3 : watchdog    PORT MAP( clock ,reset ,rswatchdog ,watchdogout );
-- Start Main process PLC CPU --
PROCESS (DCLOCK,RESET)
        Variable  jmp_flag                      : std_logic;
        Variable  mc_flag                       : std_logic;
        Variable  r_register                    : std_logic;
        Variable  s_register                    : std_logic_vector( 7 downto 0);
        Variable   data_address                 : std_logic_vector( 7 downto 0);
        Variable   data1_address,data2_address  : std_logic_vector( 7 downto 0);
        Variable   monitoraddress               : STD_LOGIC_VECTOR( 7 DOWNTO 0);
        Variable   data,data2,Correctdata       : STD_LOGIC_VECTOR( 7 DOWNTO 0);
```

```
        Variable  numlinedata                    : STD_LOGIC_VECTOR( 9 DOWNTO 0);
        Variable  instruction_register           : std_logic_vector(11 downto 0);
        Variable  register_counter               : std_logic_vector(13 downto 0);
        Variable  data1_data,data2_data          : std_logic_vector(15 downto 0);
        Variable  highlowbyte                    : INTEGER range 0 to 1;
        Variable  bitno                          : INTEGER range 0 to 15;
        Variable  multiplex                      : INTEGER range 0 to 5;
        Variable  timebaseselect                 : INTEGER range 0 to 2;
        Variable  run                            : INTEGER range 0 to 5000;
BEGIN
IF reset = '0' then
        state <= reset_pc;          -- When Press Reset Buttom
ELSIF (dclock'EVENT) AND (dclock = '1') THEN
        case state is
        when reset_pc =>
                --- Reset PLC Program Ladder Part ---
                program_counter          <= "000000000";
                memory_address_register  <= "000000000";
                register_ac              <= "0000000000000000";
                s_register               := "00000000";
                r_register               := '0';
                mc_flag                  := '1';
                memory_write             <= '0';
                --- When Start PLC read Ladder Program from Eeprom Save to Internal Memmory ---
                run                      := 0;
                highlowbyte              := 0;
                wr_start_flag            <='0';
                addressrom               <= "0000000000000";
                ------------------------------------------
                state                    <= readprogram1;
        when modeselect  =>
                mc_flag                  := '1';
                rswatchdog               <= '1';
                if modeswitch = '1' then
                        --- Run Mode Start at here ---
                        led_run          <= '0';
                        led_stop         <= '1';
                        state            <= updateinput;
                        --------------------------------
                else
```

```
                --- Stop Mode start at here ---
                led_run             <= '1';
                led_stop            <= '0';
                -- Wait for "00000001" form PC --
                if (receive_flag = '1') and (rdr_serial = "00000001" ) then
                        getcomplete_flag <= '1';
                        -- Send Data Back to Tell PLC program --
                        tdr_serial          <= "01100100";
                        requestsend_flag    <= '1';
                        state       <= saveprogram1;
                else
                        requestsend_flag <= '0';
                        getcomplete_flag <= '1';
                        state       <= modeselect;
                end if;
        end if;
-- Get Data from PC save to EEprom Part    ---
when saveprogram1 =>
        -- Wait for Send Data to PC complete --
        if sendready_flag = '1' then
                requestsend_flag    <= '0';
                state               <= saveprogram2;
        else
                state               <= saveprogram1;
        end if;
when saveprogram2 =>
        -- Wait for Number of program line --
        if (receive_flag = '1') then
                numlinedata         := rdr_serial & "00";
                getcomplete_flag    <= '1';
                addressrom          <= "0000000000000";
                state               <= saveprogram3;
        else
                state               <= saveprogram2;
        end if;
when saveprogram3 =>
        if ( addressrom(9 downto 0) < numlinedata ) then
                if (receive_flag = '1') then
                        -- recevie already --
                        datawrite  <= rdr_serial;
                        getcomplete_flag    <= '1';
```

```
                        -- start write to eeprom  Address be at //Addressrom// --
                        writeread_flag      <= '1';
                        wr_start_flag       <= '1';
                        state       <= saveprogram4;
                else
                        led_run             <= '1';
                        led_stop            <= '1';
                        state               <= saveprogram3;
                end if;
        elsif numlinedata = 0 then
                state       <= modeselect;
        else
                -- Initial Variable prepare for read data --
                state <= saveprogram5;
        end if;
when saveprogram4 =>
        -- Wait for write each data complete --
        if (wr_cpl_flag = '1') then
                wr_start_flag <= '0';
                addressrom  <= addressrom + 1;
                state           <= saveprogram3;
        else
                state  <= saveprogram4;
        end if;
when saveprogram5 =>
        -- Wait for changing to Run Mode --
        if modeswitch = '1' then
                state       <= reset_pc;
        else
                led_run    <= '1';
                led_stop   <= '0';
                state      <= saveprogram5;
        end if;
------------------------------------------------------------
--      Read Data From EEprom Part          ---
------------------------------------------------------------
when readprogram1 =>
        -- Delay for read data form EEprom --
        if run <= 4000 then
                run         := run + 1;
                state       <= readprogram1;
```

```
            else
                        run    := 0;
                        state      <= readprogram2;
            end if;
when readprogram2 =>
            -- Read Data until End of program --
            if ( addressrom(9 downto 0)  < "1100000000" ) then
                        writeread_flag      <= '0';
                        wr_start_flag       <= '1';
                        state  <= readprogram3;
            elsif addressrom(9 downto 0) < "1111111100" then
                        addressrom          <= addressrom + 2;
                        register_ac         <= "0000000000000000";
                        state  <= savetomem1;
            else
                        -- Reach the End of Ladder Program --
                        state  <= modeselect;
            end if;
when readprogram3=>
            if (wr_cpl_flag = '1') then
                        wr_start_flag       <= '0';
                        addressrom          <= addressrom + 1;
                        if highlowbyte = 0 then
                                    -- First data lay at low byte --
                                    register_ac(7 downto 0) <= dataread;
                                    highlowbyte   := 1;
                                    state   <= readprogram1;
                        else
                         -- Secound data lay at high byte complete  16 bit Register_AC prepare for write --
                                    register_ac(15 downto 8)  <= dataread;
                                    highlowbyte         := 0;
                                    state  <= savetomem1;
                        end if;
            else
                        led_run     <= '0';
                        led_stop  <= '0';
                        state      <= readprogram3;
            end if;
when savetomem1 =>
            memory_write        <= '1';
            state   <= savetomem2;
```

```
when savetomem2 =>
        memory_write        <= '0';
        state  <= savetomem3;
when savetomem3 =>
        memory_address_register      <= memory_address_register + 1;
        state   <= readprogram1;
------------------------------------------------------------
--          Start PLC Part Program          --
------------------------------------------------------------
when updateinput =>
        memory_address_register  <= "110000000";        -- Input Address 380H --
        program_counter     <= "000000000";
        register_ac         <= input;
        memory_write        <= '0';
        rswatchdog          <= '0';
        state               <= saveoutput;
when fetch =>       -- Fetch load program counter --
        memory_address_register <= program_counter;
        state   <= decode;
when decode =>      -- Get Information from Instruction --
        program_counter     <= program_counter + 1;
        memory_address_register <= memory_data_register(11 downto 4) + "110000000";
        instruction_register  := memory_data_register(15 downto 4);
        data_address          := memory_data_register( 7 downto 0);
        bitno                 := conv_integer(memory_data_register(3 downto 0));
        state                 <= decode2;
when decode2 =>   -- Decode Instruction from Opcode --
        if (jmp_flag = '1') and (instruction_register/= "111111110110") then
                state <= fetch;        - if jmp flag = 1 Don't Operate any command
        else
        ---------------------
        CASE instruction_register (11 downto 8) IS
        when "0000" =>      -- LD command --
                s_register(7) := s_register(6);
                s_register(6) := s_register(5);
                s_register(5) := s_register(4);
                s_register(4) := s_register(3);
                s_register(3) := s_register(2);
                s_register(2) := s_register(1);
                s_register(1) := s_register(0);
                s_register(0) := r_register;
```

```
            r_register     := memory_data_register(bitno) and mc_flag;
            state          <= fetch;
when "0001" =>      -- LDI command --
            s_register(7)          := s_register(6);
            s_register(6)          := s_register(5);
            s_register(5)          := s_register(4);
            s_register(4)          := s_register(3);
            s_register(3)          := s_register(2);
            s_register(2)          := s_register(1);
            s_register(1)          := s_register(0);
            s_register(0)          := r_register;
            r_register             := ( not memory_data_register(bitno) ) and mc_flag;
            state                  <= fetch;
when "0010" =>      -- AND command --
            r_register             := r_register and memory_data_register(bitno);
            state  <= fetch;
when "0011" =>      - ANDI command --
            r_register             := r_register and ( not memory_data_register(bitno) );
            state  <= fetch;
when "0100" =>      -- OR command --
            r_register             := (r_register or memory_data_register(bitno)) and mc_flag;
            state  <= fetch;
when "0101" =>      -- ORI command --
            r_register     := (r_register or  ( not memory_data_register(bitno)) ) and mc_flag;
            state  <= fetch;
when "0110" =>      -- Out command --
            register_AC            <= memory_data_register;
            state  <= execute_out;
when "0111" =>      -- Set Command --
            register_AC            <= memory_data_register;
            state  <= execute_setrst;
when "1000" =>      -- Reset Command --
            register_AC            <= memory_data_register;
            state  <= execute_setrst;
when "1001" =>      -- Timer Command --
            if r_register = '0' then           -- Reset buttom press clear counter value --
                        register_ac            <= "0000000000000000";
                        program_counter    <= program_counter + 1;
                        state  <= saveoutput;
            else        -- Execute Time Delay Command --
                        timebaseselect         := bitno;
```

```
                register_AC          <= memory_data_register;
                state  <= execute_tim;
        end if;
when "1010" =>      -- Counter Command --
        if r_register = '1' then          -- Reset buttom press reset counter value --
                register_ac          <= "0000000000000000";
                program_counter    <= program_counter + 1;
                state  <= saveoutput;
        else        -- Execute Count Command --
                register_AC          <= memory_data_register;
                state  <= execute_cnt;
        end if;
when "1111" =>      -- Expand 1 PLC Command --
        CASE instruction_register (7 downto 4) IS
        when "0000" =>      -- Compare Command --
                if r_register = '1' then
                        memory_address_register <= program_counter;
                        state  <= execute_data;
                else
                        program_counter <= program_counter + 1;
                        state  <= fetch;
                end if;
        when "0001" =>      -- ADD Command --
                if r_register = '1' then
                        memory_address_register <= program_counter;
                        state  <= execute_data;
                else
                        program_counter <= program_counter + 1;
                        state  <= fetch;
                end if;
        when "0010" =>      -- SUB Command --
                if r_register = '1' then
                        memory_address_register <= program_counter;
                        state  <= execute_data;
                else
                        program_counter <= program_counter + 1;
                        state  <= fetch;
                end if;
        when "0011" =>      -- Mov Command --
                if r_register = '1' then
                        memory_address_register <= program_counter;
```

```
                state   <= execute_mov;
        else
                program_counter <= program_counter + 1;
                state  <= fetch;
        end if;
when "1111" =>      -- Expand 2 PLC Command --
        CASE instruction_register (3 downto 0) IS
        when "0000" =>      -- And Block Command --
                r_register          := r_register and s_register(0);
                s_register(0)       := s_register(1);
                s_register(1)       := s_register(2);
                s_register(2)       := s_register(3);
                s_register(3)       := s_register(4);
                s_register(4)       := s_register(5);
                s_register(5)       := s_register(6);
                s_register(6)       := s_register(7);
                s_register(7)       := '0';
                state   <= fetch;
        when "0001" =>      -- Or Block Command --
                r_register          := r_register or s_register(0);
                s_register(0)       := s_register(1);
                s_register(1)       := s_register(2);
                s_register(2)       := s_register(3);
                s_register(3)       := s_register(4);
                s_register(4)       := s_register(5);
                s_register(5)       := s_register(6);
                s_register(6)       := s_register(7);
                s_register(7)       := '0';
                state   <= fetch;
        when "0010" =>      -- End Command
                -- Execute end command start program counter at 0000 ---
                program_counter    <= "000000000";
                state   <= updateoutput;
        when "0011" =>      -- MC Command --
                mc_flag   := r_register;
                state   <= fetch;
        when "0100" =>      -- MCC Command --
                mc_flag   := '1';
                state   <= fetch;
        when "0101" =>
                -- JMP Command --
```

```
                                jmp_flag  := not r_register;
                                state  <= fetch;
                        when "0110" =>      -- JME Command --
                                jmp_flag  := '0';
                                state  <= fetch;
                        when OTHERS =>
                                state  <= fetch;      -- NOP Command --
                        End case;
                when OTHERS =>
                        state  <= modeselect;
                End case;
        when OTHERS =>
                state  <= modeselect;
        End case;
        ----------
        end if;
-----------------------------------------
-- Start Execute PLC Command --
-----------------------------------------
when execute_out =>
        register_AC(bitno) <= r_register;          -- Put Result register to Output at Specific bitnot --
        memory_write       <= '0';
        state  <= saveoutput;
when execute_setrst =>          -- 0111 Set 1000 RST command --
        if instruction_register (11 downto 8) = "0111" then
                register_AC(bitno)   <= '1';
        else
                register_AC(bitno)   <= '0';
        end if;
        state  <= saveoutput;
when execute_tim =>             -- Execute Counter Command --
        if register_ac(15) = '1' then     -- If counter value is full stop count --
                program_counter    <= program_counter + 1;
                state  <= fetch;
        elsif  (register_ac(14) ='0') and (timebase(timebaseselect) = '1') then
                -- Pulse changed count value increase --
                memory_address_register      <= program_counter;
                register_ac(14)      <= '1';
                register_ac(13 downto 0)       <= register_ac(13 downto 0) + 1;
                state  <= execute_cnt2;
        elsif (register_ac(14) ='1')  and (timebase(timebaseselect) = '0') then
```

```
                -- Pulse change from High to Low --
                program_counter     <= program_counter + 1;
                register_ac(14)     <= '0';
                state  <= saveoutput;
        else    -- The same pluse come LS-high come high LS-Low come Low --
                program_counter     <= program_counter + 1;
                state   <= fetch;
        end if;
when execute_cnt =>             -- Execute Counter Command --
        if register_ac(15) = '1' then     -- If counter value is full stop count --
                program_counter     <= program_counter + 1;
                state   <= fetch;
        elsif  (register_ac(14) ='0') and (s_register(0) = '1') then
                -- Pulse changed count value increase --
                memory_address_register     <= program_counter;
                register_ac(14)     <= '1';
                register_ac(13 downto 0)     <= register_ac(13 downto 0) + 1;
                state   <= execute_cnt2;
        elsif (register_ac(14) ='1') and (s_register(0) = '0') then
                -- Pulse change from High to Low --
                program_counter     <= program_counter + 1;
                register_ac(14)     <= '0';
                state  <= saveoutput;
        else    -- The same pluse come LS-high come high LS-Low come Low --
                program_counter     <= program_counter + 1;
                state   <= fetch;
        end if;
when execute_cnt2 =>            -- Load Timer/Counter Set Value --
        program_counter     <= program_counter + 1;
        register_counter    := memory_data_register(13 downto 0);
        state   <= execute_cnt3;
when execute_cnt3 =>            -- Compare Timer/Counter Set Value --
        if register_ac(13 downto 0) >= register_counter then
                register_ac(15)     <= '1';
        end if;
        memory_address_register     <= instruction_register (7 downto 0) + "110000000";
        state   <= saveoutput;
when execute_data =>            -- Execute Data handling Command --
        program_counter  <= program_counter + 1;
        memory_address_register  <= memory_data_register(15 downto 8) + "110000000";
        data2_address       := memory_data_register(7 downto 0);
```

```
            state   <= execute_data2;
when execute_data2 =>
            data1_data              := memory_data_register and  "0011111111111111";
            memory_address_register     <= data2_address + "110000000";
            state   <= execute_data3;
when execute_data3 =>
            data2_data              := memory_data_register and "0011111111111111";
            CASE opcode_register(7 downto 4) IS
            when "0001" =>
                        state       <= execute_cmp;
            when "0010" =>
                        state       <= execute_add;
            when "0011" =>
                        state       <= execute_sub;
            when OTHERS =>
                        state       <= modeselect;
            End case;
when execute_add =>             -- Execute Add Command --
            register_AC             <= data1_data + data2_data;
            memory_address_register     <= data_address + "110000000";
            state   <= saveoutput;
when execute_sub =>             -- Execute Sub Command --
            if data1_data > data2_data then
                        register_AC <= data1_data - data2_data;
            else
                        register_AC <= "0000000000000000";
            end if;
            memory_address_register     <= data_address + "110000000";
            state  <= saveoutput;
when execute_cmp =>             -- Execute Cmp Command --
            if data1_data < data2_data then
                        register_AC <= "0000000000000100";
            elsif data1_data = data2_data then
                        register_AC <= "0000000000000010";
            else
                        register_AC <= "0000000000000001";
            end if;
            memory_address_register <= "110010000";
            state       <= saveoutput;
when execute_mov =>             -- Execute Mov Command --
            program_counter     <= program_counter + 1;
```

```
            register_AC          <= memory_data_register;
            memory_address_register      <= data_address + "110000000";
            state  <= saveoutput;
when saveoutput  =>
            memory_write         <= '1';
            state  <= saveoutput2;
when saveoutput2 =>          -- Write to memory wait for Zero to low --
            memory_write         <= '0';
            state  <= fetch;
when updateoutput =>         -- Read data from memory --
            memory_address_register <= "110000100";          -- Output 900 --
            state  <= updateoutput2;
when updateoutput2 =>        -- Output to output --
            output     <= memory_data_register;
            state  <= monitor;
-------------------------------------------------------------------------------
--    Send Data In Internal Memory from PLC to Computer    --
when monitor =>
            if sendready_flag = '0' then
                        tdr_serial           <= data;
                        requestsend_flag     <= '1';
                        state  <= modeselect;
            else
                        requestsend_flag     <= '0';
                        memory_address_register      <= monitoraddress + "110000000";
                        state  <= monitor2;
            end if;
when monitor2 =>
            case multiplex is
            when 0 =>                -- Send Start 1 --
                        data         := "01100011";
                        multiplex  := 1;
            when 1 =>                -- Send Start 2 --
                        data         := "01100100";
                        correctdata := "10000000";
                        multiplex := 2;
            when 2 =>                -- Send Address --
                        if monitoraddress = "00000000" then
                                    data    := "00000001";
                                    correctdata(6) := '1';
                        else
```

```
                        data := monitoraddress;
                    end if;
                    multiplex := 3;
                when 3 =>           -- Send Data Low byte  --
                    if memory_data_register( 7 downto 0) = "00000000" then
                        data := "00000001";
                        correctdata(5) := '1';
                    else
                        data := memory_data_register( 7 downto 0);
                    end if;
                    data2  := memory_data_register(15 downto 8);
                    multiplex := 4;
                when 4 =>           -- Send Data High byte  --
                    if data2 = "00000000" then
                        data   := "00000001";
                        correctdata(4) := '1';
                    else
                        data := data2;
                    end if;
                    monitoraddress      := monitoraddress + 1;
                    multiplex := 5;
                when 5 =>           -- Send Correction Data  --
                    if monitoraddress = "10000000" then
                        monitoraddress := "00000000";
                    end if;
                    data       := correctdata;
                    multiplex  := 0;
                when OTHERS =>
                    multiplex  := 0;
                END case;
                state      <= modeselect;
            when others =>
                state      <= reset_pc;
            end case;
end if;
end process;
-------------------------------------------------------
--   Divide Clock 25 MHz to 12.5 Mhz     --
process (clock)
begin
        if (clock'EVENT AND clock = '1') then
```

```
                dclock <= not dclock;
        end if;
end process;
-------------------------------------------------------
--  Clock Divider Time base  Generate   --
-------------------------------------------------------
process (clock)
Variable   countclock  : INTEGER range 0 to 25175 :=  0;
Variable   countx100,countx10000 : INTEGER range 0 to 100 := 100;
begin
        if (clock'EVENT AND clock = '1') then
                CASE countclock IS
                when 25175 =>
                        countclock  := 0;
                        CASE countx100 IS
                        when 100 =>
                                countx100 := 0;
                                CASE countx10000 IS
                                when 100 =>
                                        countx10000 := 0;
                                when 0 to 50 =>
                                        timebase(2) <= '1';             -- Time base 2 Period 10 Sec
                                when others =>
                                        timebase(2) <= '0';
                                end case;
                                countx10000 := countx10000 + 1;
                        when 0 to 50 =>
                                timebase(1) <= '1';                     -- Time base 1 Period 0.1 Sec
                        when others =>
                                timebase(1) <= '0';
                        end case;
                        countx100 := countx100 + 1;
                when 0 to 12580 =>
                        timebase(0) <= '1';                             -- Time base 1 Period 0.001 Sec
                when others =>
                        timebase(0) <= '0';
                end case;
                countclock := countclock + 1;
        end if;
end process;
end a;
```

--- Subcircuit for Read/write program to Eeprom (User Memory) ---

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY wreerom IS
        port(
                clock           : IN    std_logic;
                reset           : IN    std_logic;
                writeread_flag  : IN    std_logic;
                wr_start_flag   : IN    std_logic;
                addressrom      : IN    STD_LOGIC_VECTOR(12 downto 0);
                datawrite       : IN    std_logic_vector( 7 downto 0);
                ce              : OUT   STD_LOGIC := '1';
                oe              : OUT   std_logic := '1';
                we              : OUT   std_logic := '1';
                cbuffer         : OUT   std_logic;
                wr_cpl_flag     : OUT   std_logic := '0';
                addressout      : OUT   STD_LOGIC_VECTOR(12 downto 0);
                dataread        : OUT   std_logic_vector( 7 downto 0);
                dataout         : INOUT std_logic_vector( 7 downto 0) );
END wreerom;
ARCHITECTURE a OF wreerom IS
TYPE STATE_TYPE IS (wait_start,write1,write2,write3,write4,read1,read2,read3,send_cpl,wait_cpl);
SIGNAL state: STATE_TYPE;
SIGNAL dclock                           : STD_LOGIC;
BEGIN
process(dclock,reset)
Variable  run       : INTEGER range 0 to 1000;
begin
        IF reset = '0' then
                wr_cpl_flag <= '0';
                ce      <= '1';
                oe      <= '1';
                we      <= '1';
                run     := 0;
                state   <= wait_start;
        ELSIF dclock'EVENT AND dclock = '1' THEN
                case state is
                        when wait_start =>          -- writeread_flag = 1 is write / = 0 is read --
                                addressout      <= addressrom;
```

```
                wr_cpl_flag         <= '0';
                ce          <= '1';
                oe          <= '1';
                we          <= '1';
                if (writeread_flag = '1') and (wr_start_flag = '1') then
                        dataout     <= datawrite;
                        cbuffer     <= '0';
                        state       <= write1;
                elsif (writeread_flag = '0') and (wr_start_flag = '1') then
                        cbuffer     <= '1';
                        state       <=  read1;
                else
                        state       <=  wait_start;
                end if;
        when write1 =>
                ce          <= '0';
                state       <= write2;
        when write2 =>
                we          <= '0';
                state       <= write3;
        when write3 =>
                state       <= write4;
        when write4 =>
                we          <= '1';
                state       <= send_cpl;
        when read1 =>
                ce          <= '0';
                run         := 0;
                state       <= read2;
        when read2 =>
                oe          <= '0';
                if run <= 999 then
                        run  :=  run + 1;
                        state <= read2;
                else
                        run    := 0;
                        state  <= read3;
                end if;
        when read3 =>
                dataread <= dataout;
                oe          <= '1';
```

```
                    state       <= send_cpl;
                when send_cpl =>
                    ce          <= '1';
                    wr_cpl_flag <= '1';
                    state       <= wait_cpl;
                when wait_cpl =>
                    if wr_start_flag = '0' then
                        wr_cpl_flag <= '0';
                        state  <= wait_start;
                    else
                        state  <= wait_cpl;
                    end if;
                when others =>
                    state   <= wait_start;
            end case;
        END IF;
end process;
-------------------------------------------------------
--   Divide Clock 25 MHz to 12.5 Mhz      --
-------------------------------------------------------
process (clock)
begin
-------------------------------------------------------
        if (clock'EVENT AND clock = '1') then
            dclock <= not dclock;
        end if;
-------------------------------------------------------
end process;
-------------------------------------------------------
END a;
-------------------------------------------------------------------------------------------------------------------------
```

--- Subcircuit for Transmit Data to Computer ---

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY transmit IS
PORT(   clock              : IN  std_logic;
        reset              : IN  std_logic;
         tdr_serial        : IN  std_logic_vector(7 downto 0);
         requestsend_flag  : IN  std_logic;
         tx                : OUT std_logic := '1';
         sendready_flag    : OUT std_logic := '0');
END transmit;
ARCHITECTURE a OF transmit IS
TYPE STATE_TYPE IS (wait_send,sendtx,sendready);
SIGNAL state: STATE_TYPE;
SIGNAL tsr_serial          : STD_LOGIC_VECTOR( 7 DOWNTO 0);
SIGNAL dclock              : STD_LOGIC;
BEGIN
-----------------------------------
process (dclock,reset)
Variable  sertxclock       : INTEGER range 0 to 1311 :=  0;
Variable  counttxbitser    : INTEGER range 0 to   11 := 11;
Variable  bitserial        : INTEGER range 0 to    7;
begin
IF reset = '0' then
        state <= wait_send;
ELSIF (dclock'EVENT) AND (dclock = '1') THEN
        case state is
        when wait_send =>
                if  (requestsend_flag = '1') then
                        -- Send Start bit when tdre become 1 --
                        tsr_serial  <= tdr_serial;
                        tx          <= '0';    -- Start Bit start signal to Zero --
                        sendready_flag      <= '0';
                        counttxbitser       := 0;
                        sertxclock          := 0;
                        state       <= sendtx;
                else
                        tx          <= '1';
                        sendready_flag      <= '0';
```

```
                    state   <= wait_send;
                end if;
            when sendtx         =>
                    if sertxclock = 1311 then
                        CASE counttxbitser IS
                            WHEN 0 to 7 =>
                                bitserial   := counttxbitser;
                                tx          <= tsr_serial(bitserial);
                                state       <= sendtx;
                            WHEN 8 to 9 =>
                                tx          <= '1';
                                state       <= sendtx;
                            WHEN 10 =>
                                sendready_flag <= '1';
                                state       <= sendready;
                            WHEN OTHERS =>
                                state       <= sendready;
                        END CASE;
                        ------
                        sertxclock     := 0;
                        counttxbitser  := counttxbitser + 1;
                    else
                        sertxclock     := sertxclock + 1;
                        state   <= sendtx;
                    end if;
            when sendready =>
                if (requestsend_flag = '1') then
                    state <= sendready;
                else
                    sendready_flag <= '0';
                    state <= wait_send;
                end if;
            when others =>
                state <= wait_send;
            end case;
end if;
end process;
```

```
-------------------------------------------------------
--   Divide Clock 25 MHz to 12.5 Mhz     --
-------------------------------------------------------
process (clock)
begin
-------------------------------------------------------
        if (clock'EVENT AND clock = '1') then
                dclock <= not dclock;
        end if;
-------------------------------------------------------
end process;
-------------------------------------------------------
END a;
```

--- Sub Circuit for Receive Data from Computer ---

```
LIBRARY IEEE;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY receive IS
PORT(   clock               : IN  std_logic;
        reset               : IN  std_logic;
        getcomplete_flag    : IN  std_logic;
        rx                  : IN  std_logic;
        rdr_serial          : OUT std_logic_vector(7 downto 0);
        receive_flag        : OUT std_logic := '0' );
END receive;
ARCHITECTURE a OF receive IS
TYPE STATE_TYPE IS (wait_receive,receive_startbit,receive_data,receive_complete);
SIGNAL state: STATE_TYPE;
SIGNAL rsr_serial           : STD_LOGIC_VECTOR( 7 DOWNTO 0);
SIGNAL dclock               : STD_LOGIC;
BEGIN
process (Dclock,reset)
Variable  serrxclock        : INTEGER range 0 to  1311 :=  0;
Variable  countrxbitser     : INTEGER range 0 to   10 := 10;
Variable  bitserial         : INTEGER range 0 to    7;
begin
```

```
IF reset = '0' then
        state <= wait_receive;
ELSIF (dclock'EVENT) AND (dclock = '1') THEN
        case state is
        when wait_receive =>
                if (rx = '0') then
                        -- Wait for start Signal from RS 232 --
                        countrxbitser       := 0;
                        receive_flag        <= '0';
                        state  <= receive_startbit;
                else
                        state  <= wait_receive;
                end if;
        when receive_startbit =>
                if (serrxclock = 655) then
                        serrxclock   := 0;
                        -- Check sure after rx = 0 at center pulse must be still 0 --
                        if (rx = '0') then
                                countrxbitser := 1;
                                state  <= receive_data;
                        else
                                countrxbitser := 10;
                                state   <= wait_receive;
                        end if;
                else
                        -- Add clock for next time --
                        serrxclock := serrxclock + 1;
                        state   <= receive_startbit;
                end if;
        when receive_data =>
                -- in the next bit try to collect data --
                if (serrxclock = 1311 ) then
                        if (countrxbitser <= 8) then
                                bitserial   := countrxbitser - 1;
                                rsr_serial(bitserial) <= rx;
                                state   <= receive_data;
                        else
                                -- Copy data to rdr_derial --
                                rdr_serial   <= rsr_serial;
                                receive_flag <= '1';
                                state   <= receive_complete;
```

```
                    end if;
                    serrxclock := 0;
                    countrxbitser := countrxbitser + 1;
                else
                    -- Add clock for next time --
                    serrxclock := serrxclock + 1;
                    state <= receive_data;
                end if;
        WHEN receive_complete =>
                if getcomplete_flag = '0' then
                    state  <= receive_complete;
                else
                    receive_flag <= '0';
                    rdr_serial    <= "00000000";
                    state <= wait_receive;
                end if;
        WHEN OTHERS =>
                state  <= wait_receive;
        END CASE;
        ------
end if;
end process;
-------------------------------------------------------------
--   Divide Clock 25 MHz to 12.5 Mhz      --
-------------------------------------------------------------
process (clock)
begin
        if (clock'EVENT AND clock = '1') then
                dclock <= not dclock;
        end if;
end process;
-------------------------------------------------------------
END a;
```

--- Sub Circuit for internal Watchdog ---

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY watchdog IS
        port(   clock                   : IN    std_logic;
                reset                   : IN    std_logic;
                rswatchdog              : IN    std_logic;
                watchdogout             : OUT   STD_LOGIC := '1' );
        END watchdog;
ARCHITECTURE a OF watchdog IS
BEGIN
process(clock,reset)
Variable  run       : INTEGER range 0 to 125875;
begin
        IF reset = '0' then
                run := 0;
                watchdogout <= '1';
        ELSIF clock'EVENT AND clock = '1' THEN
                if rswatchdog = '1' then
                        run := 0;
                        watchdogout <= '1';
                elsif run = 125875 then
                        watchdogout <= '0';
                else
                        run := run + 1;
                end if;
        END IF;
end process;
END a;
```

ภาคผนวก ข
รายงานการใช้ทรัพยากรของชิป FPAG

Project Information e:\yut\vhdlplc\nplc1.rpt
MAX+plus II Compiler Report File
Version 10.1 06/12/2001
Compiled: 09/04/2002 14:20:32
***** Project compilation was successful
NPLC1
** DEVICE SUMMARY **

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	Memory Bits	Memory %Utilized	LCs LCs	LCs %Utilized
nplc1	EPF10K70RC240-2	20	37	8	8192	44 %	2666	71 %
User Pins:		20	37	8				

Project Information
Device-Specific Information:
nplc1
** RESOURCE USAGE **

Embedded Array Block	Embedded Cells	Column Interconnect Driven	Row Interconnect Driven	Clocks	Read/ Write	External Interconnect
A53	8/8 (100%)	4/8 (50%)	0/8 (0%)	1/2	2/2	15/26(57%)
D53	8/8 (100%)	5/8 (62%)	1/8 (12%)	1/2	2/2	15/26(57%)
E53	8/8 (100%)	4/8 (50%)	4/8 (50%)	1/2	2/2	15/26(57%)
F53	8/8 (100%)	4/8 (50%)	4/8 (50%)	1/2	2/2	15/26(57%)

Total dedicated input pins used:	6/6	(100%)
Total I/O pins used:	59/183	(32%)
Total logic cells used:	2666/3744	(71%)
Total embedded cells used:	32/72	(44%)
Total EABs used:	4/9	(44%)
Average fan-in:	3.46/4	(86%)
Total fan-in:	9242/14976	(61%)
Total input pins required:	20	
Total input I/O cell registers required:	0	
Total output pins required:	37	
Total output I/O cell registers required:	0	
Total buried I/O cell registers required:	0	
Total bidirectional pins required:	8	
Total reserved pins required	0	
Total logic cells required:	2666	
Total flipflops required:	425	
Total packed registers required:	0	
Total logic cells in carry chains:	0	
Total number of carry chains:	0	
Total logic cells in cascade chains:	0	
Total number of cascade chains:	0	
Total single-pin Clock Enables required:	0	
Total single-pin Output Enables required:	0	

Logic cells inserted for fitting: 69
Synthesized logic cells: 1303/3744 (34%)
Device-Specific Information: e:\yut\vhdlplc\nplc1.rpt
nplc1
** INPUTS **

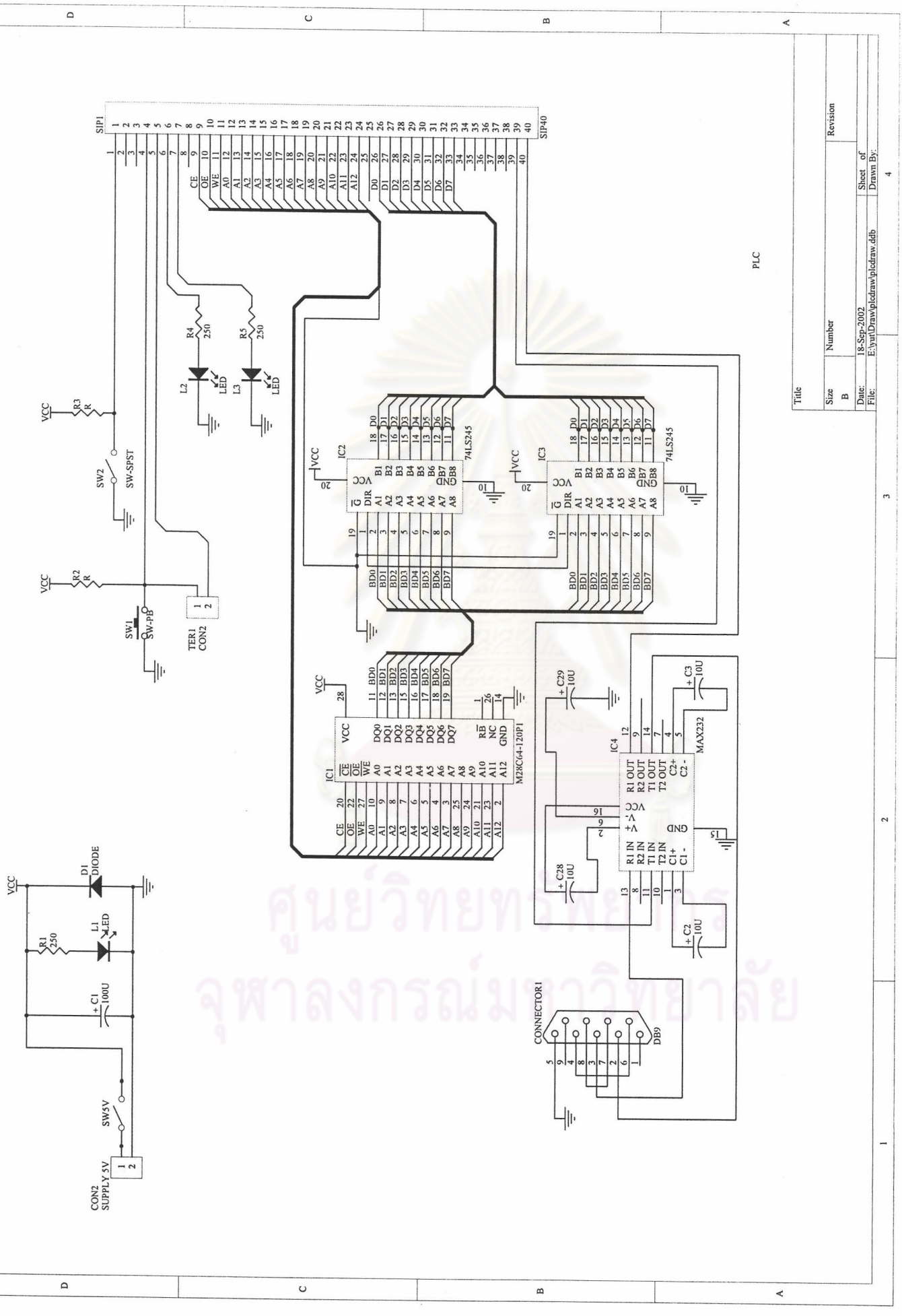
Pin	LC	EC	Row	Col	Primitive	Code	Fan-In INP	Fan-In FBK	Fan-Out OUT	Fan-Out FBK	Name
91	-	-	-	--	INPUT	G	0	0	0	0	clock
110	-	-	-	10	BIDIR		0	1	0	2	dataout0
113	-	-	-	08	BIDIR		0	1	0	2	dataout1
115	-	-	-	06	BIDIR		0	1	0	2	dataout2
117	-	-	-	03	BIDIR		0	1	0	2	dataout3
119	-	-	-	02	BIDIR		0	1	0	2	dataout4
126	-	-	I	--	BIDIR		0	1	0	2	dataout5
128	-	-	I	--	BIDIR		0	1	0	2	dataout6
131	-	-	H	--	BIDIR		0	1	0	2	dataout7
133	-	-	H	--	INPUT		0	0	0	1	input0
136	-	-	H	--	INPUT		0	0	0	1	input1
138	-	-	G	--	INPUT		0	0	0	1	input2
141	-	-	G	--	INPUT		0	0	0	1	input3
143	-	-	G	--	INPUT		0	0	0	1	input4
146	-	-	F	--	INPUT		0	0	0	1	input5
148	-	-	F	--	INPUT		0	0	0	1	input6
151	-	-	E	--	INPUT		0	0	0	1	input7
90	-	-	-	--	INPUT		0	0	0	1	input8
92	-	-	-	--	INPUT		0	0	0	1	input9
210	-	-	-	--	INPUT		0	0	0	1	input10
212	-	-	-	--	INPUT		0	0	0	1	input11
211	-	-	-	--	INPUT		0	0	0	1	input12
51	-	-	I	--	INPUT		0	0	0	1	input13
39	-	-	F	--	INPUT		0	0	0	1	input14
15	-	-	B	--	INPUT		0	0	0	1	input15
154	-	-	E	--	INPUT		0	0	0	9	modeswitch
29	-	-	D	--	INPUT		0	0	0	10	reset
156	-	-	D	--	INPUT		0	0	0	12	rx

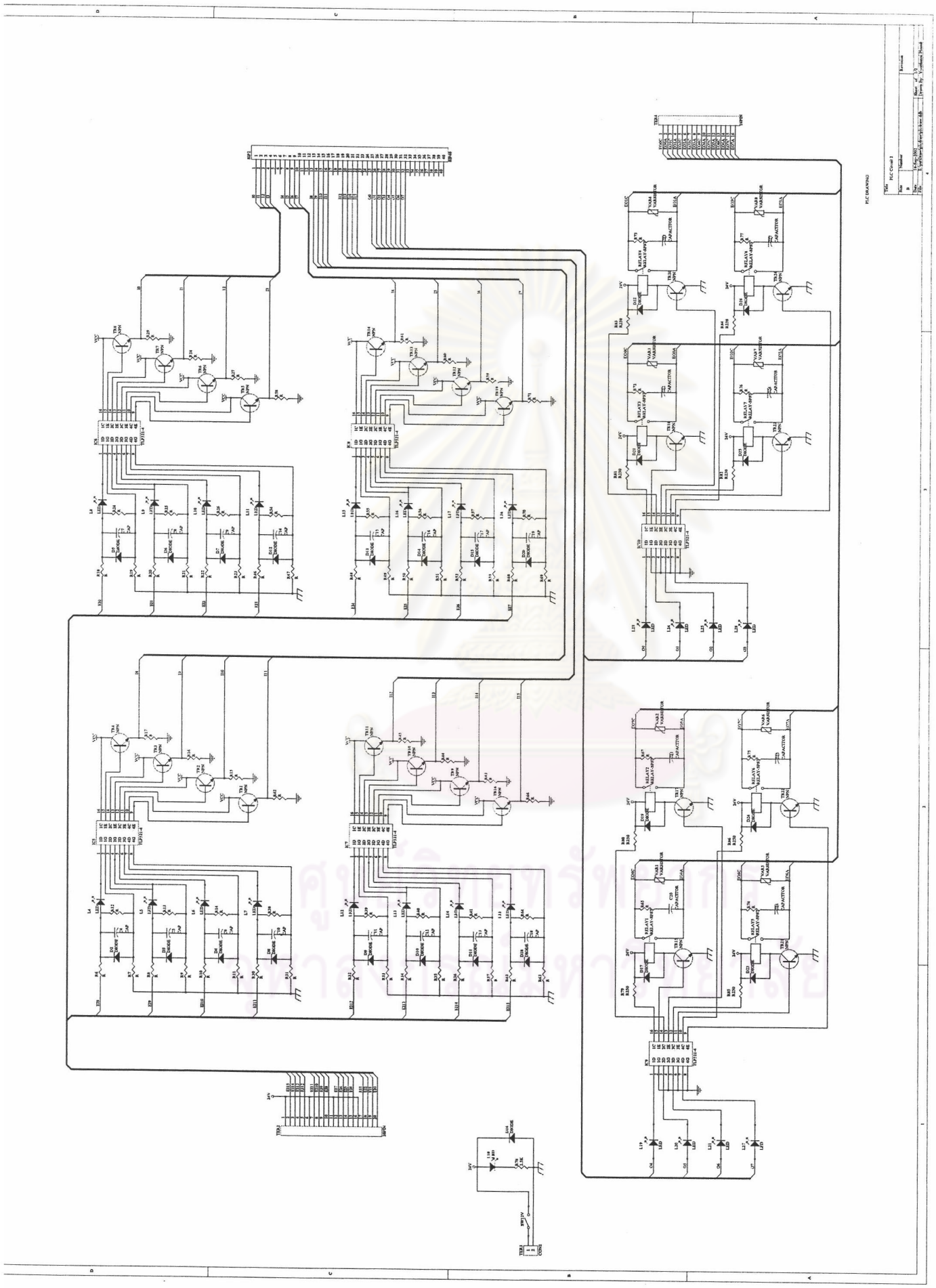
Device-Specific Information: e:\yut\vhdlplc\nplc1.rpt
.* OUTPUTS **

Pin	Fed By LC	Fed By EC	Row	Col	Primitive	Code	Fan-In INP	Fan-In FBK	Fan-Out OUT	Fan-Out FBK	Name
116	-	-	-	5	OUTPUT		0	1	0	0	addressout0
118	-	-	-	02	OUTPUT		0	1	0	0	addressout1
120	-	-	-	01	OUTPUT		0	1	0	0	addressout2
127	-	-	I	--	OUTPUT		0	1	0	0	addressout3
129	-	-	I	--	OUTPUT		0	1	0	0	addressout4
132	-	-	H	--	OUTPUT		0	1	0	0	addressout5

134	-	-	H	--	OUTPUT	0	1	0	0	addressout6
137	-	-	H	--	OUTPUT	0	1	0	0	addressout7
139	-	-	G	--	OUTPUT	0	1	0	0	addressout8
142	-	-	G	--	OUTPUT	0	1	0	0	addressout9
144	-	-	F	--	OUTPUT	0	1	0	0	addressout10
147	-	-	F	--	OUTPUT	0	1	0	0	addressout11
149	-	-	E	--	OUTPUT	0	1	0	0	addressout12
152	-	-	E	--	OUTPUT	0	1	0	0	cbuffer
109	-	-	-	11	OUTPUT	0	1	0	0	ce
110	-	-	-	10	TRI	0	1	0	2	dataout0
113	-	-	-	08	TRI	0	1	0	2	dataout1
115	-	-	-	06	TRI	0	1	0	2	dataout2
117	-	-	-	03	TRI	0	1	0	2	dataout3
119	-	-	-	02	TRI	0	1	0	2	dataout4
126	-	-	I	--	TRI	0	1	0	2	dataout5
128	-	-	I	--	TRI	0	1	0	2	dataout6
131	-	-	H	--	TRI	0	1	0	2	dataout7
157	-	-	D	--	OUTPUT	0	1	0	0	led_run
159	-	-	D	--	OUTPUT	0	1	0	0	led_stop
114	-	-	-	07	OUTPUT	0	1	0	0	oe
162	-	-	C	--	OUTPUT	0	1	0	0	output0
158	-	-	D	--	OUTPUT	0	1	0	0	output1
161	-	-	C	--	OUTPUT	0	1	0	0	output2
163	-	-	C	--	OUTPUT	0	1	0	0	output3
55	-	-	I	--	OUTPUT	0	1	0	0	output4
48	-	-	H	--	OUTPUT	0	1	0	0	output5
50	-	-	H	--	OUTPUT	0	1	0	0	output6
53	-	-	I	--	OUTPUT	0	1	0	0	output7
240	-	-	-	52	OUTPUT	0	1	0	0	output8
62	-	-	-	51	OUTPUT	0	1	0	0	output9
19	-	-	C	--	OUTPUT	0	1	0	0	output10
20	-	-	C	--	OUTPUT	0	1	0	0	output11
21	-	-	C	--	OUTPUT	0	1	0	0	output12
24	-	-	C	--	OUTPUT	0	1	0	0	output13
23	-	-	C	--	OUTPUT	0	1	0	0	output14
56	-	-	I	--	OUTPUT	0	1	0	0	output15
230	-	-	-	43	OUTPUT	0	1	0	0	rswatchdog
153	-	-	E	--	OUTPU	0	1	0	0	tx
111	-	-	-	09	OUTPUT	0	1	0	0	we

ภาคผนวก ค
รายละเอียดแผนภาพวงจรบอร์ดต้นแบบ





ภาคผนวก ง
คู่มือการใช้เครื่องและซอฟต์แวร์พัฒนาโปรแกรมขั้นบันได

การใช้เครื่อง

เมื่อเริ่มจ่ายไฟหลอด LED Power on จะติด การทำงานของ PLC ก็จะขึ้นอยู่กับว่าสวิตช์เลือกโหมดอยู่ในตำแหน่งใด ถ้าอยู่ในตำแหน่งโหมดทำงาน PLC ก็จะอ่านโปรแกรมจาก EEPROM มาเก็บใน RAM ภายในแล้วทำการดำเนินการตามโปรแกรมหลอด LED RUN MODE จะติด แต่ถ้าสวิตช์เลือกโหมดอยู่ในตำแหน่งโหมดหยุด PLC จะไม่ทำงาน หลอด LED STOP MODE จะติด ซึ่งผู้ใช้สามารถจะโหลดโปรแกรมใหม่ได้

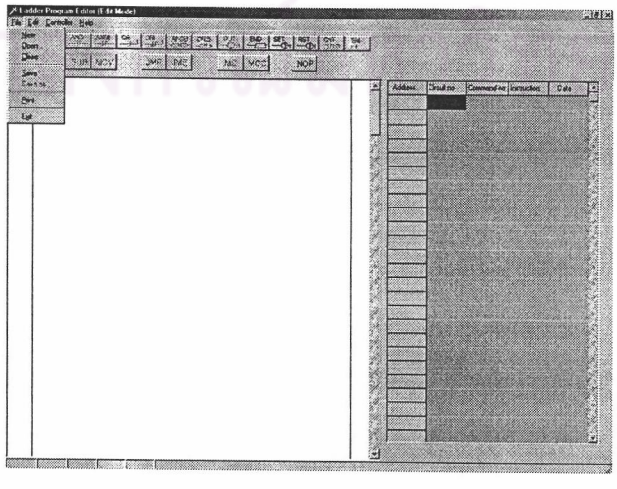
การเชื่อมต่อระหว่าง PLC และคอมพิวเตอร์ใช้สายแบบอนุกรมหัวต่อ DB9 ซึ่งเมื่อเชื่อมต่อสายแล้วจะทำให้ผู้ใช้สามารถโหลดโปรแกรมขั้นบันไดใหม่ได้เมื่อ PLC อยู่ในโหมดหยุด หรือสามารถดูสถานะรีเลย์ภายในของ PLC ได้ เมื่อ PLC อยู่ในโหมดหยุด แต่การทำงานปกติของ PLC ไม่จำเป็นต้องมีการเชื่อมต่อสายใดๆ กับคอมพิวเตอร์

คู่มือการใช้โปรแกรมขั้นบันได

โปรแกรมขั้นบันไดทำหน้าที่ในการสร้างหรือแก้ไขโปรแกรมขั้นบันได เปลี่ยนโปรแกรมขั้นบันไดเป็นภาษาเครื่องที่ PLC เข้าใจ สามารถเชื่อมต่อกับ PLC เพื่อทำการโหลดโปรแกรมจากเครื่องคอมพิวเตอร์ส่วนบุคคลลงไปในเครื่อง PLC และยังสามารถรับข้อมูลสถานะรีเลย์ภายในจาก PLC เพื่อแสดงผล ประกอบด้วย 2 โหมด คือ โหมดแก้ไขโปรแกรม (Editor Mode) และโหมดเชื่อมต่อกับ PLC (Connection Mode)

โหมดแก้ไขโปรแกรม (Editor Mode)

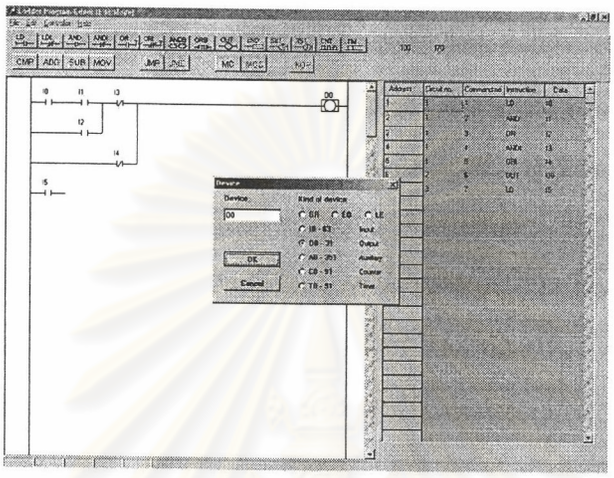
เมื่อแรกเข้าโปรแกรมผู้ใช้จะเข้ามาสู่โหมดนี้ก่อน และสามารถเลือกที่จะสร้างโปรแกรมใหม่ (New) หรือเปิดโปรแกรมเก่า (Open)



เมนู File

NEW (**สร้างโปรแกรมขั้นบันไดใหม่**)

จะเห็นปุ่มคำสั่งที่สามารถเลือกใช้งานได้ คือคำสั่ง LD, LDI, END, SET, RST, NOP จะเป็นคำสั่งคำสั่งที่สามารถเริ่มต้นโปรแกรมขั้นบันไดได้ โดยรายละเอียดของแต่ละคำสั่งมีดังต่อไปนี้



LD, LDI ใช้เริ่มต้นวงจร เมื่อคลิก LD หรือ LDI เมนู Input Device จะแสดงขึ้น สามารถป้อนค่า I0 – I31 สำหรับอ่านค่า Input O0 – O31 สำหรับอ่านค่า Output A0 – A159 สำหรับอ่านค่า Auxiliary Relay T/C0 – T/C59 หรือ GR, EQ ,LE สำหรับอ่านค่าผลของคำสั่ง CMP (Compare) หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจร LD หรือ LDI เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี LD หรือ LDI เพิ่มขึ้น

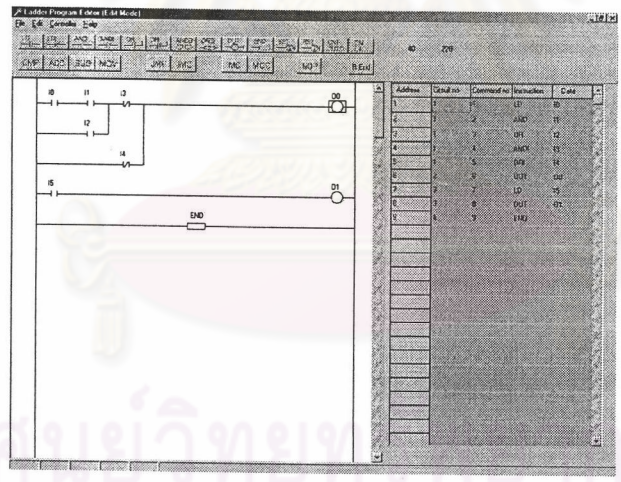
AND, ANDI สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว เมื่อคลิก AND หรือ ANDI เมนู Input Device จะแสดงขึ้น สามารถป้อนค่า I0 – I31 สำหรับอ่านค่า Input O0 – O31 สำหรับอ่านค่า Output A0 – A159 สำหรับอ่านค่า Auxiliary Relay T/C0 – T/C59 หรือ GR, EQ ,LE สำหรับอ่านค่าผลของคำสั่ง CMP (Compare) หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก เพื่อมาทำการ AND หรือ ANDI เลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรเกิดจากการ AND หรือ ANDI เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี AND หรือ ANDI เพิ่มขึ้น

OR, ORI เมื่อคลิก OR หรือ ORI เมนู Input Device จะแสดงขึ้น สามารถป้อนค่า I0 – I31 สำหรับอ่านค่า Input O0 – O31 สำหรับอ่านค่า Output A0 – A159 สำหรับอ่านค่า Auxiliary Relay T/C0 – T/C59 หรือ GR, EQ ,LE สำหรับอ่านค่าผลของคำสั่ง CMP (Compare) หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก เพื่อมาทำการ OR

หรือ ORI เลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรเกิดจากการ OR หรือ ORI เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี OR หรือ ORI เพิ่มขึ้น

ANDB, ORB เมื่อคลิก ANDB หรือ ORB ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรเกิดจากการ ANDB หรือ ORB เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี ANDB หรือ ORB เพิ่มขึ้น ก่อนจะใช้คำสั่ง ANDB หรือ ORB ต้องมีการกำหนดบล็อกมากกว่า 2 บล็อกไว้ก่อนแล้ว แต่สูงสุดไม่เกิน 8 บล็อก คำสั่ง ANDB หรือ ORB สามารถใช้ต่อกันสูงสุดไม่เกิน 8 คำสั่งเช่นเดียวกัน

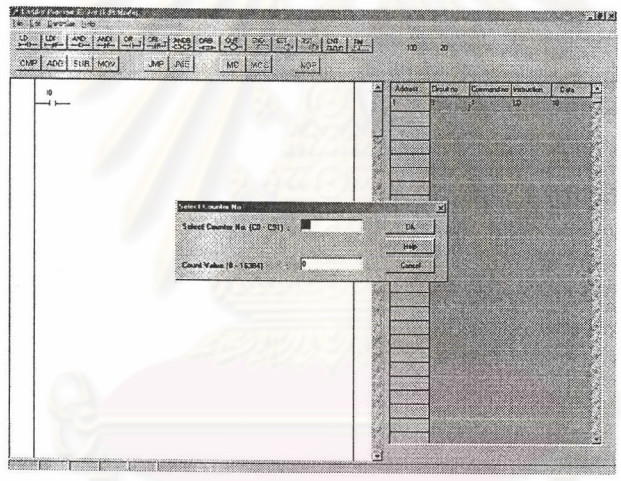
OUT ใช้ปิดท้ายวงจร สำหรับนำผลลัพธ์ออกไปยังรีเลย์ที่กำหนด เมื่อคลิก OUTเมนู Input Device จะแสดงขึ้น สามารถป้อนค่า O0 – O31 สำหรับ Output A0 – A159 สำหรับ Auxiliary Relay หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจร OUT เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี OUT เพิ่มขึ้น



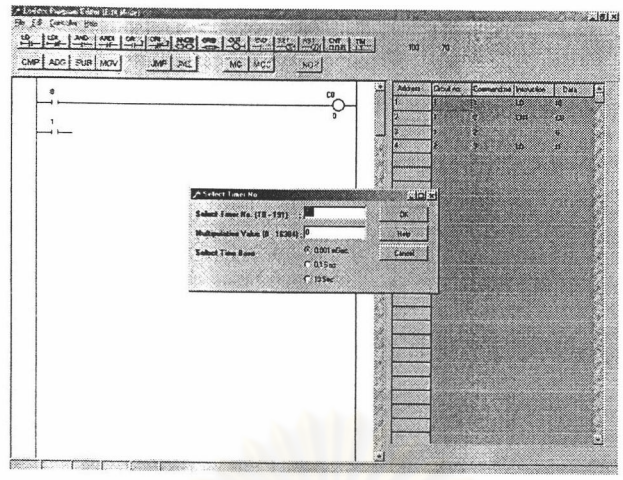
END ใช้เมื่อเขียนโปรแกรมขั้นบันไดจนถึงคำสั่งสุดท้าย ใช้เป็นตัวปิดท้ายของทุกคำสั่ง เมื่อคลิก END จะมีเมนูเพื่อทำการยืนยัน (Confirm) เมื่อทำการยืนยัน ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจร END เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี END เพิ่มขึ้น เมื่อใส่คำสั่ง END เข้าไปแล้ว จะไม่สามารถใส่คำสั่งอื่นเข้าไปเพิ่มได้อีก ผู้ใช้สามารถจะยกเลิก END โดยกดปุ่ม REND

SET ใช้ป้อนค่า ON ไปยังรีเลย์ที่กำหนด เมื่อคลิก SET เมนู Input Device จะแสดงขึ้น O0 – O31 สำหรับค่า Output A0 – A159 สำหรับค่า Auxiliary Relay หรือ T/C0 – T/C59 สำหรับ Counter Timer หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรเกิดจากการ SET เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี SET เพิ่มขึ้น

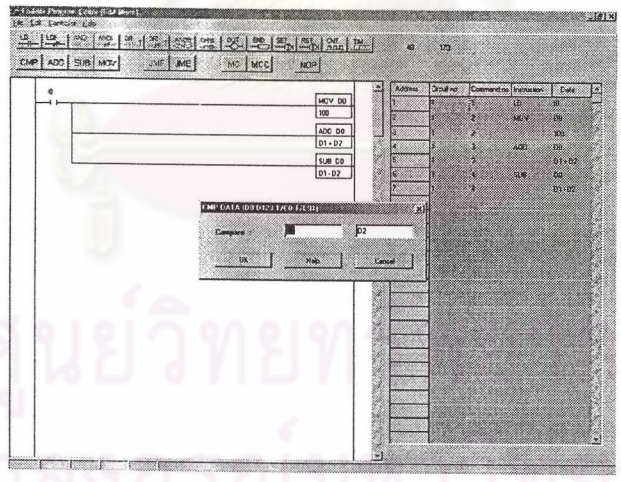
RST ใช้ป้อนค่า OFF ไปยังรีเลย์ที่กำหนด เมื่อคลิก RST เมนู Input Device จะแสดงขึ้น O0 – O31 สำหรับค่า Output A0 – A159 สำหรับค่า Auxiliary Relay หรือ T/C0 – T/C59 สำหรับ Counter Timer หรือผู้ใช้สามารถคลิกที่ปุ่ม Reference เพื่อช่วยในการเลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรเกิดจากการ RST เพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี RST เพิ่มขึ้น



CNT เป็นคำสั่ง Counter สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว ก่อนใช้คำสั่งนี้จะต้องมีสองบล็อก บล็อกแรกเป็นสัญญาณที่ใช้นับ บล็อกที่สองเป็นสัญญาณให้การ Reset เมื่อคลิก CNT เมนู Counter จะแสดงขึ้น ให้ใส่ Input Counter C0 – C59 ห้ามใส่หมายเลขที่ซ้ำกับส่วน TIMER และค่าการนับที่ Set Value สามารถใส่ค่าได้ 0 – 16383 ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ CNT และเลขการนับเพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี CNT และเลขการนับเพิ่มขึ้นเช่นกัน



TIM เป็นคำสั่ง Timer สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว เมื่อคลิก TIM เมนู TIMER จะแสดงขึ้น ให้ใส่ Input Counter T0 – T59 ห้ามใส่หมายเลขที่ซ้ำกับส่วน COUTER และค่าการนับที่ Set Value สามารถใส่ค่าได้ 0 – 16383 ต้องเลือก Time base ซึ่งมีด้วยกัน 3 ช่วงเวลาด้วยกันคือ 0.001 วินาที 0.1 วินาที และ 10 วินาที ค่า Delay Time on สามารถคำนวณได้จากผลคูณของค่าการนับกับ Time base ที่เลือก ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ TIM + Time base และเลขการนับเพิ่มขึ้น เช่นเดียวกับ Statement List ที่จะมี TIM + Time base และเลขการนับเพิ่มขึ้นเช่นกัน



CMP เป็นคำสั่ง Compare สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว ค่าระหว่างข้อมูลสองชุดซึ่งอาจจะเป็น T/C0 – T/C59 สำหรับ Counter Timer หรือส่วน Data Memory D0 – D49 เมื่อคลิกที่ CMP เมนู Compare จะแสดงขึ้น ซึ่งให้ป้อนตำแหน่งข้อมูลตัวหน้าก่อนแล้วตามด้วยตัวหลัง โดยตัวแรกเป็นตัวตั้งในการเปรียบเทียบ ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ CMP และข้อมูลทั้งสอง เช่นเดียวกับ Statement List ที่จะมี CMP และข้อมูลทั้งสอง

ADD เป็นคำสั่ง Addition สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว ค่าผลลัพธ์และข้อมูลสองชุดที่บวกกัน ซึ่งอาจจะเป็น T/C0 – T/C59 สำหรับ Counter Timer หรือส่วน Data Memory D0 – D49 เมื่อคลิกที่ ADD เมนู Addtion จะแสดงขึ้น ซึ่งให้ป้อนตำแหน่งข้อมูลผลลัพธ์ก่อน แล้วตามด้วยตำแหน่งข้อมูลสองชุดที่บวกกัน ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ ADD + ตำแหน่งของผลลัพธ์และตำแหน่งข้อมูลสองชุดที่บวกกัน เช่นเดียวกับ Statement List ที่จะมี ADD + ตำแหน่งของผลลัพธ์และตำแหน่งข้อมูลสองชุดที่บวกกัน จะเพิ่มขึ้นเช่นกัน

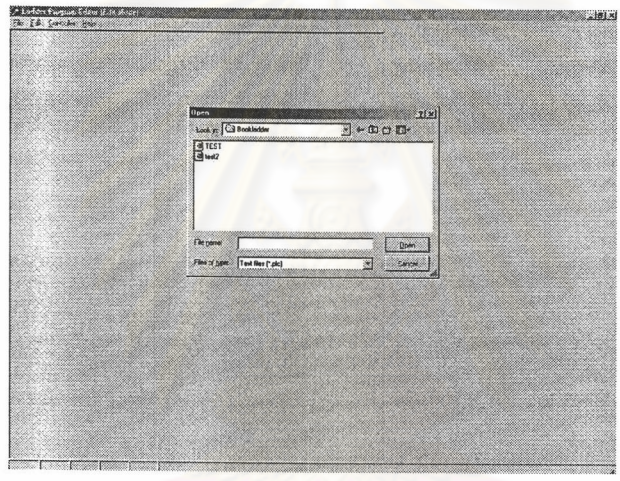
SUB เป็นคำสั่ง Subtraction สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว ค่าผลลัพธ์และข้อมูลสองชุดที่ลบกัน ซึ่งอาจจะเป็น T/C0 – T/C59 สำหรับ Counter Timer หรือส่วน Data Memory D0 – D49 เมื่อคลิกที่ SUB เมนู Subtraction จะแสดงขึ้น ซึ่งให้ป้อนตำแหน่งข้อมูลผลลัพธ์ก่อน แล้วตามด้วยตำแหน่งข้อมูลชุดแรกที่จะเป็นตัวตั้ง และตัวที่สองที่จะเป็นตัวลบ ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ SUB + ตำแหน่งของผลลัพธ์และตำแหน่งข้อมูลสองชุดที่ลบกัน เช่นเดียวกับ Statement List ที่จะมี SUB + ตำแหน่งของผลลัพธ์และตำแหน่งข้อมูลสองชุดที่ลบกัน จะเพิ่มขึ้นเช่นกัน

MOV เป็นคำสั่ง MOVE สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว โดยนำค่าตัวเลขสูงสุด 14 บิท 0 – 16383 ใส่ใน /C0 – T/C59 สำหรับ Counter Timer หรือส่วน Data Memory D0 – D49 เมื่อคลิกที่ MOV เมนู MOVE จะแสดงขึ้น ซึ่งให้ป้อนตำแหน่งข้อมูลที่จะนำค่าตัวเลขไปเก็บ แล้วตามด้วยค่าตัวเลขนั้นๆ ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ MOV + ตำแหน่งของผลลัพธ์และค่าตัวเลข เช่นเดียวกับ Statement List ที่จะมี MOV + ตำแหน่งของผลลัพธ์และค่าตัวเลขเพิ่มขึ้นเช่นกัน

JMP, JME สำหรับ JMP สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว โดยตอนเริ่มต้นมีเพียงคำสั่ง JMP ที่ Enable อยู่สามารถคลิกได้ หลังจากถูกคลิกแล้ว JMP ก็จะ Disable ส่วน JME ต้องรอให้มีการกด JMP ก่อนแล้วและต้องเป็นช่วงเริ่มวงจรใหม่เท่านั้นถึงจะ Enable ให้คลิกได้ หลังจากคลิก JMP จึงจะถูก Enable อีกครั้ง ช่วงที่ JMP ถูกคลิก คำสั่ง MC ก็จะถูก Disable ไม่ให้ใช้งาน

MC, MCC สำหรับ MC สามารถใช้คำสั่งนี้เมื่อวงจรได้ถูกเริ่มต้นแล้ว โดยตอนเริ่มต้นมีเพียงคำสั่ง MC ที่ Enable อยู่สามารถคลิกได้ หลังจากถูกคลิกแล้ว MC ก็จะ Disable ส่วน MCC ต้องรอให้มีการกด MC ก่อนแล้วและต้องเป็นช่วงเริ่มวงจรใหม่เท่านั้นถึงจะ Enable ให้คลิกได้ หลังจากคลิก MCC จึงจะถูก Enable อีกครั้ง ช่วงที่ MC ถูกคลิก คำสั่ง JMP ก็จะถูก Disable ไม่ให้ใช้งาน

NOP เป็นคำสั่งที่ไม่ได้สั่งให้ PLC ทำงานอะไรทั้งสิ้น สามารถคำสั่งนี้เมื่อวงจรเมื่อเริ่มวงจร นั้นๆ ส่วนแสดงผลโปรแกรมขั้นบันไดจะแสดงวงจรของ NOP เช่นเดียวกับ Statement List ที่จะมี NOP เพิ่มขึ้นเช่นกัน



OPEN (**เปิดโปรแกรมขั้นบันไดที่ได้มีบันทึกไว้แล้ว**)

โปรแกรมขั้นบันไดจะถูกจัดเก็บในรูป Text File ของ Statement List โดยใช้นามสกุล .PLC ผู้ใช้สามารถเลือกเปิด File นามสกุล .PLC เพื่อมาทำการเปลี่ยนแปลงแก้ไข จัดเก็บใหม่ หรือทำการเปลี่ยนให้เป็นภาษาเครื่องเพื่อทำการโหลดลง PLC

CLOSE (**ปิดโปรแกรมขั้นบันไดที่ทำงานอยู่**)

ผู้ใช้ต้องตรวจสอบว่าโปรแกรมขั้นบันไดได้มีการบันทึกเรียบร้อยหรือยังก่อนใช้คำสั่งนี้

SAVE (**บันทึกโปรแกรมขั้นบันได**)

ใช้บันทึกโปรแกรมขั้นบันไดที่กำลังใช้งานอยู่ นามสกุล .PLC

SAVE AS (**บันทึกโปรแกรมขั้นบันไดเป็น File อื่น**)

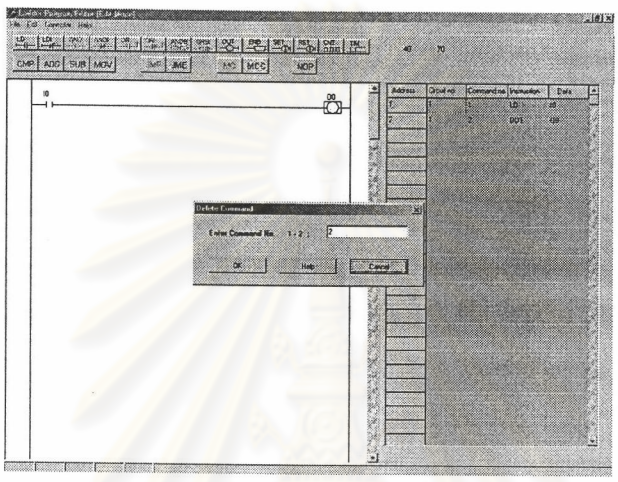
ใช้บันทึกโปรแกรมขั้นบันไดที่กำลังใช้งานอยู่ ให้เป็นชื่อไฟล์อื่น นามสกุล .PLC

PRINT (**พิมพ์โปรแกรมขั้นบันได**)

ผู้ใช้สามารถพิมพ์โปรแกรมขั้นบันไดในรูปของโปรแกรม Ladder หรือในรูป Statement List

EXIT (**ออกจากโปรแกรม**)

ออกจากโปรแกรม ตรวจสอบการบันทึกโปรแกรมก่อนออกจากโปรแกรม



เมนู EDIT

UNDO (**เลิกทำ**)

ยกเลิกการทำคำสั่งสุดท้าย

INSERT (**แทรกคำสั่ง**)

แทรกคำสั่งขั้นบันไดเข้าไปในโปรแกรม โดยใส่หมายเลข Command no. ที่ต้องการจะแทรกคำสั่งนั้นเข้าไป

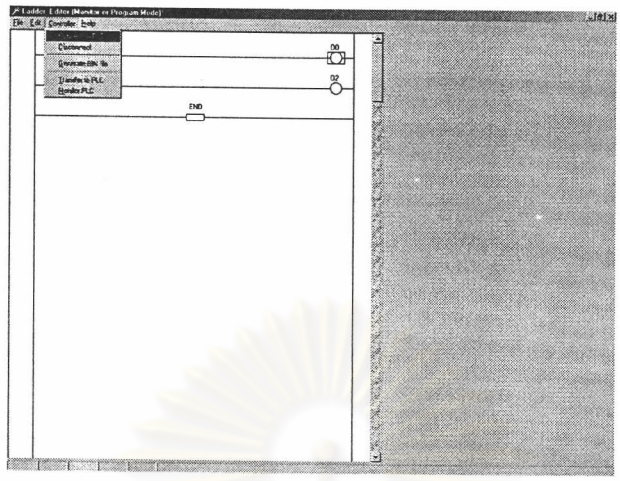
DELETE (**ลบคำสั่ง**)

ลบคำสั่งขั้นบันไดที่อยู่ในโปรแกรมออก โดยใส่หมายเลข Command no. ที่ต้องการจะลบคำสั่งนั้นเข้าไป

Clear All (**ล้างคำสั่งขั้นบันไดทั้งหมด**)

ล้างคำสั่งขั้นบันไดทั้งหมดที่เขียนอยู่ มีไว้สำหรับจะเริ่มเขียนโปรแกรมขั้นบันไดใหม่

เมนู Controller

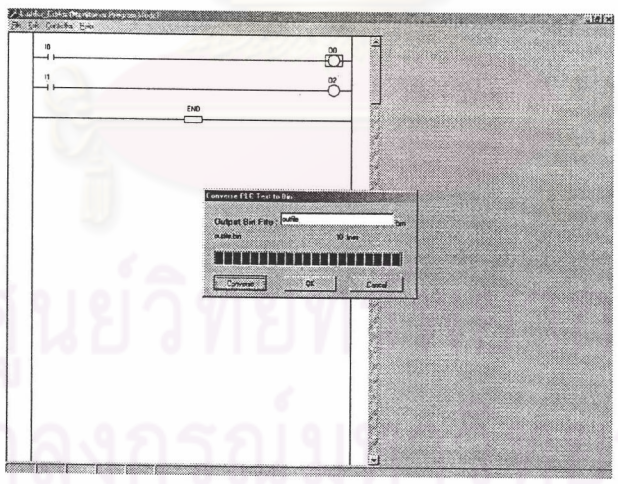


Connect to PLC (**ต่อกับ** PLC)

เปิดพอร์ตอนุกรมเพื่อเตรียมสื่อสารกับ PLC จะสามารถเปิดพอร์ตอนุกรมนี้ได้ก็ต่อเมื่อโปรแกรมขั้นบันไดที่เปิดอยู่มีคำสั่ง END ปิดท้ายโปรแกรมแล้ว

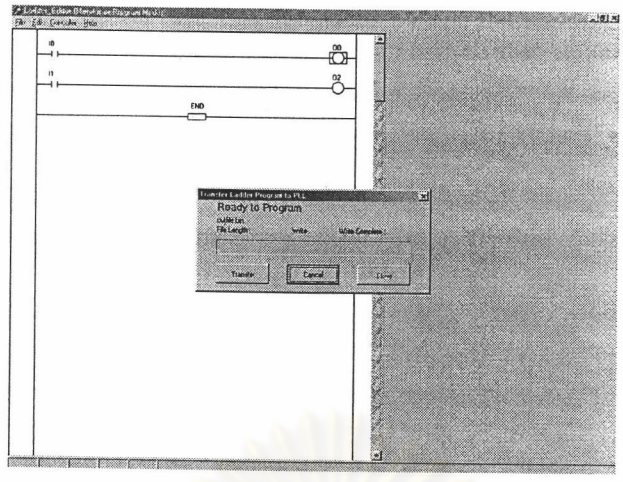
Disconnect (**ตัดจาก** PLC)

ปิดพอร์ตอนุกรมตัดการสื่อสารกับ PLC



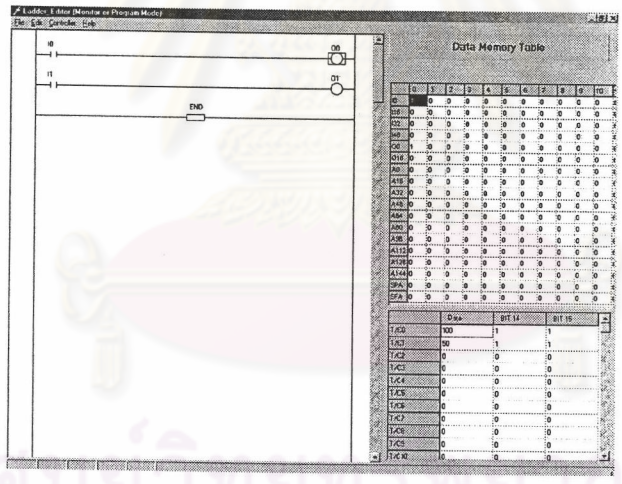
Generate BIN file (**แปลงให้เป็นภาษาเครื่อง** BIN file)

คลิกเลือกเมื่อต้องการเปลี่ยนโปรแกรมภาษาขั้นบันไดให้เป็นภาษาเครื่อง จะได้ไฟล์ผลลัพธ์นามสกุล .BIN



Transfer to PLC (**โหลดโปรแกรมลง** PLC)

คลิกเลือกเมื่อต้องการโหลดโปรแกรมลง PLC โดยโปรแกรมจะตรวจสอบสถานะของ PLC ว่าอยู่ในโหมดหยุดหรือยัง ถ้าอยู่ในโหมดหยุดจะมีข้อความ Ready to Transfer แสดงขึ้นมา เมื่อผู้ใช้คลิกเลือก Transfer ก็จะส่งข้อมูลโปรแกรมมายัง PLC จนครบ จากนั้นเมื่อผู้ใช้เลือกไปยังโหมดทำงาน PLC ก็เริ่มทำการควบคุมตามโปรแกรมที่โหลดใหม่



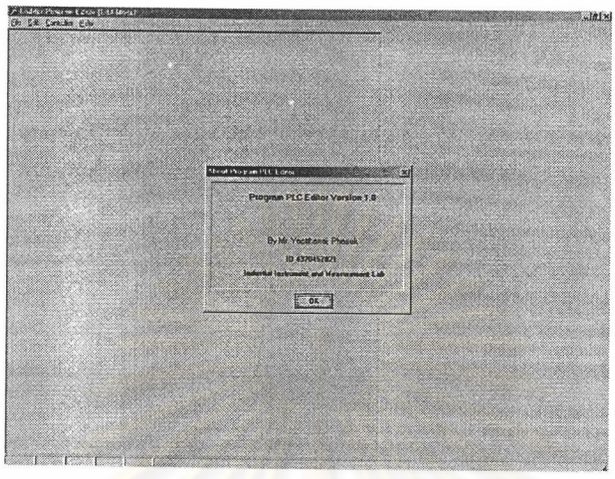
Monitor PLC (**แสดงสถานะภายใน** PLC)

เมื่อ PLC อยู่ในโหมดทำงาน ผู้ใช้สามารถดูสถานะของรีเลย์ภายใน PLC โดยเลือกไปที่เมนู Monitor PLC

เมนู Help

Content (**วิธีใช้**)

แสดงวิธีของโปรแกรม



About (**เกี่ยวกับโปรแกรม**)

แสดงรายละเอียดเกี่ยวกับโปรแกรม

ประวัติผู้เขียนวิทยานิพนธ์

นายยุทธนัย ผาสุข เกิดวันที่ 23 ตุลาคม พ.ศ. 2517 ที่จังหวัดอุบลราชธานี สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า จากมหาวิทยาลัยเกษตรศาสตร์ เมื่อปี พ.ศ. 2539 เคยทำงานในตำแหน่งวิศวกรไฟฟ้าที่บริษัทแอ๊ดวานซ์ อโกร จำกัด เป็นเวลา 4 ปี ก่อนออกมาศึกษาในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้าที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ 2543