Semiconductors

AN10388

Application of P89LPC917 in five-fan control system

Rev. 02 — 07 December 2005

Application note

Document information

Info	**Content**
Keywords	Fan controller, MCU, D/C brushless motor, P89LPC917
Abstract	The article describes the design of the fan controller, which adopts P89LPC917 design to control five fans. The controller may independently or simultaneously control the operation of several fans according to the environmental temperature, promising to provide intelligent and flexible responsive control over the abnormality of multiple kinds of operation. The PWM and AD functions of P89LPC917 make it able to realize multiple control curves precisely hence applicable to multiple control environment.

PHILIPS

Revision history

Rev	Date	Description
02	20051207	Replaced SI4410DY MOSFET with PHK12NQ03LT.
01	20050808	Initial version.

Contact information

For additional information, please visit: http://www.semiconductors.philips.com

For sales office addresses, please send an email to: sales.addresses@www.semiconductors.philips.com

1. Introduction

In many electronic appliances, heat dissipation is always an important problem to solve. Fans that are used for cooling the systems usually use a constant full speed fan drive. There is a more effective method to control the speed of fans thus enabling the whole system to be more energy-efficient, more convenient to control heat and ensuring less noise yet more reliability.

This article talks about a reference design using the Philips P89LPC917 MCU for fan control. Applying pulse width modulation mode to control five brushless DC fan motors, following the ambient temperature of the system to control the speed of the DC brushless motors. The measurement of temperature only needs a thermal resister with an ordinary negative temperature coefficient. Simultaneously, the system can monitor the failure of fans thus to reinforce the protection of the system, elevate its reliability and lower the power consumption.

2. Characteristics of system functions

- The rotation speed of fans will change following the change of temperature to reduce noise and lengthen their service life.
- PWM drive of five PWM fans.
- Continual temperature monitoring ensures the reliable start-up at turn-on and the reliable recovery from over heat failure.
- Continual fan failure monitoring avoids the undesirable effect brought forth by damage, shortage and block of fans.
- LED indicators direct driven by 20 mA may indicate various conditions including faulty power supply, fan failure, overheat alarm and disconnection of fans.
- The 500 ms start delay interval with one-by-one start of fans diminishes the surge current and noise.
- For initial start, PWM signal-driving fan is adopted with 2.5 s period and 65 % duty ratio.
- Watchdog timer may achieve self-check and reset functions.
- On-chip oscillator may save the external oscillation circuit.
- On-chip power on reset may save external reset circuit.
- Supporting low power consumption of NTC thermal resistor.
- Hysteresis control between 10 kΩ (25 °C) and 15 kΩ (20 °C) (when THRESHOLD pins being grounded) to avoid unnecessary fan turning during low temperature, the fan is turned only when the resistance value is lower than 10 kΩ to lower the temperature.
- Multiple protections to monitor failures may prevent the system from being damaged due to heat.
- 2.7 V to 3.6 V operating voltage.
- P89LPC917: 16 pin TSSOP package.

3. Hardware design

3.1 Introduction of system

The functional block diagram of the system is indicated in Fig 1, including fan drive and monitoring circuit, temperature sampling circuit, status display circuit.

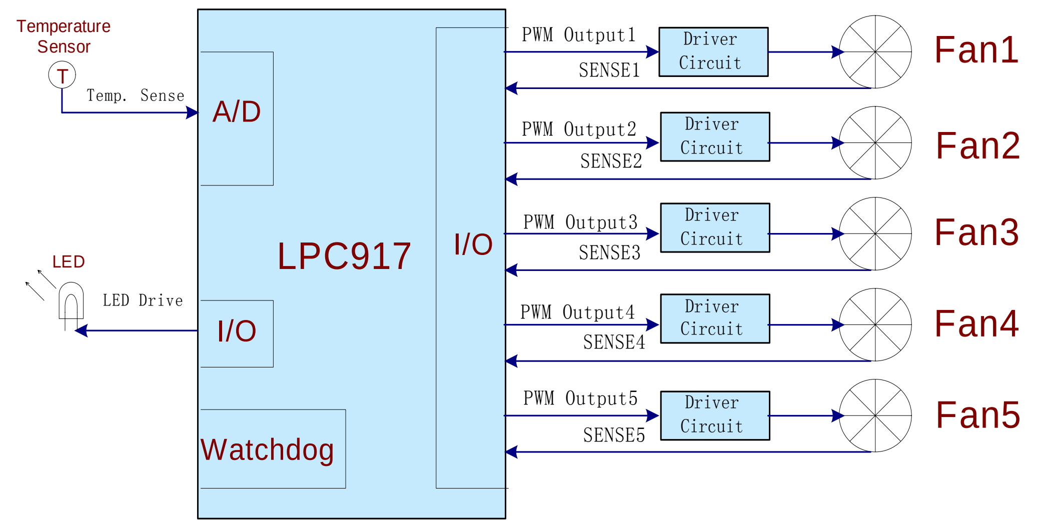

Fig 1. Functional block diagram

3.2 AD sampling circuit

The temperature measurement is taken by an on-chip 8-bit ADC conversion of the P89LPC917. The circuit is shown in Fig 2. U4 is a thermal resistor with a negative temperature coefficient; a variable resistor may be mounted on U3 to simulate the thermal resistor. P0.4 is the AD conversion input pin of P89LPC917; the resistance value of the thermal resistor will change following the change of ambient temperature.

The resistance value can be measured by the voltage of P0.4 ADC input pin, which will provide the ambient temperature. The output of the PWM signal’s duty cycle can be decided simply by the resistance value thus to control the fan speed. The duty cycle of PWM signal ranges from 30 % to 98 %, respectively corresponding to the resistance value of thermal resistor from 10 k (temperature 25 °C) going down to 1.5 k (temperature 76 °C) ohm (NTC thermal resistor prevails).

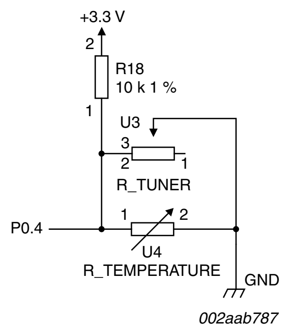

Fig 2. Temperature sensing circuit

3.3 PWM output driver circuit

As shown in Fig 3, the power supply of fan motors is DC 12 V, the PWM output pin connects to a 1st grade buffering circuit to drive a FET (PHK12NQ03LT), where the NPN transistor of buffering circuit is PDTC114TT.

3.4 Fan monitoring circuit

As shown in Fig 3, a small resistor is grounded at the drain of an N-channel FET (PHK12NQ03LT), when the current passing the fans flows via the resistor on which a slight voltage drop may occur. When the PWM output drives the brushless fan motor to work normally, on the resistor, a narrow pulse may happen which, after being enlarged passing the triode, outputs to the 1/0 pin of P89LPC917. When software of one chip computer fails to monitor these signals, which indicates that the system suffers faults. However, the resistance value of this small resistor can be only determined by the current passing the brushless D/C fan motor.

A group of recommended values are given as follows:

Table 1: Recommended resistor values

Current normally passing fans (mA)	Minimum resistance value (ohms)
100	5.1
200	2.5
450	2.2
800	1.0

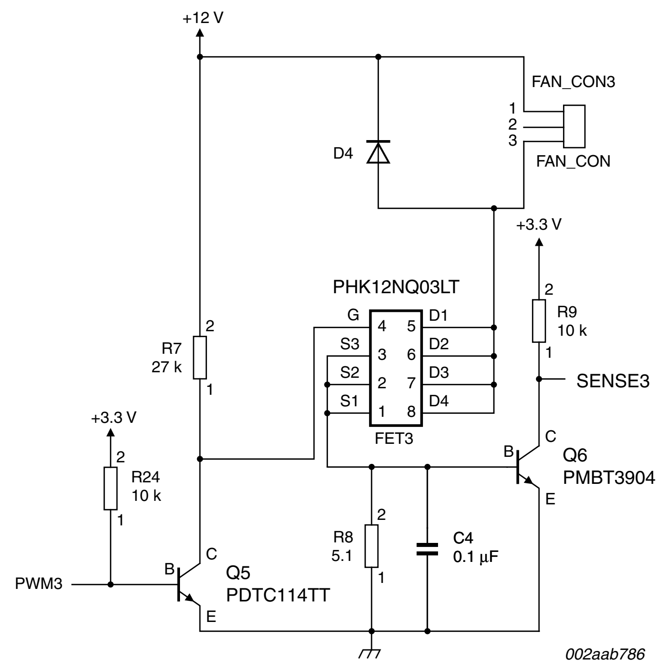

Fig 3. Schematic drawing showing connection of P89LPC917 with fans

3.5 Status display circuit

A LED is used to display the working state of fans. For detailed definitions, please refer to the software description.

3.6 Application of on-chip functions

In this design we use the on-chip RC oscillator, the on-chip power on reset and the on-chip watchdog functions of P89LPC917. Use of these on-chip functions greatly reduces system cost.

3.7 Reference schematic

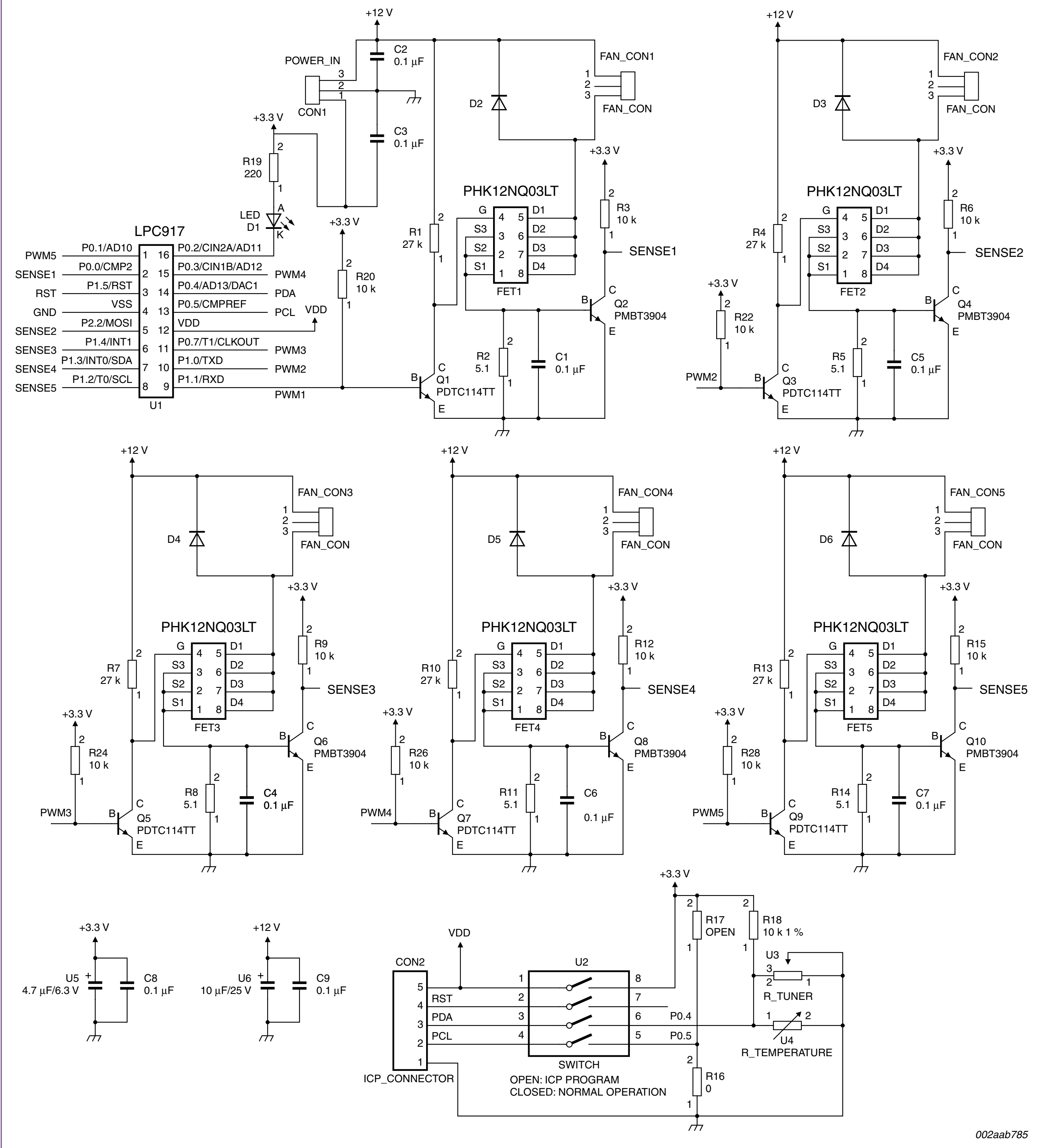

Fig 4. Schematic drawing of system

4. Software design

4.1 Software development environment

- Integrated development environment: Keil µVision2
- Emulation board: EPM900 Emulator-Programmer for Philips 89LPC9xx
- Programming tools: Flash Magic (Philips) and MCB900 Evaluation Board

4.2 Initial start up

The frequency adopted for this design is 30 Hz, PWM signal with duty cycle from 30 % to 98 % is used to drive fans, when the temperature exceeds 25 °C, (i.e., resistance value of NTC thermal resistor is equivalent or lower than 10 kΩ). The fans will be started one by one, the next fan will be started after 500 ms delay. This kind of design may effectively avoid excessive surge current during the start of fans with noise also reduced.

In the process of initial start, as some kinds of fans need high driving for the initial start, PWM drive signals with low duty cycle are probably unable to start fans normally. Therefore the PWM signals are driven with 2.5 s period and 65 % duty cycle to start driving the fans. You may choose your own duty cycle to match the fans you use for the initial start. After the initial start is completed, the output of PWM signals can be adjusted in accordingly with the ambient temperature.

4.3 Fans drive

The system follows the change of resistance value of thermal resistor to adjust the output of PWM signals, Fig 5 indicates the relation of the change between the duty cycle and resistance vale of thermal resistor for PWM drive signals.

- When P0.5 is set at “1” and the resistance value surpasses 10 kΩ, the fans stop running.
- When P0.5 is set at “0” and the resistance value surpasses 15 kΩ, the fans stop running.
- When P0.5 is set at “0” and resistance value is between 10 kΩ and 15 kΩ the fans are driven by PWM signals with duty cycle 30 %, which will reduce the times of start and stop of fans with low range temperature fluctuations.
- When the resistance value is 10 kΩ, the fans are driven by PWM signals with 30 % duty cycle,
- When the resistance value is 1.7 kΩ, the fans are driven by PWM signal with 95 % duty cycle.
- When the resistance value is between 1.7 kΩ and 10 kΩ, the duty cycle of PWM signals will change linearly following the change of the resistance value of thermal resister.

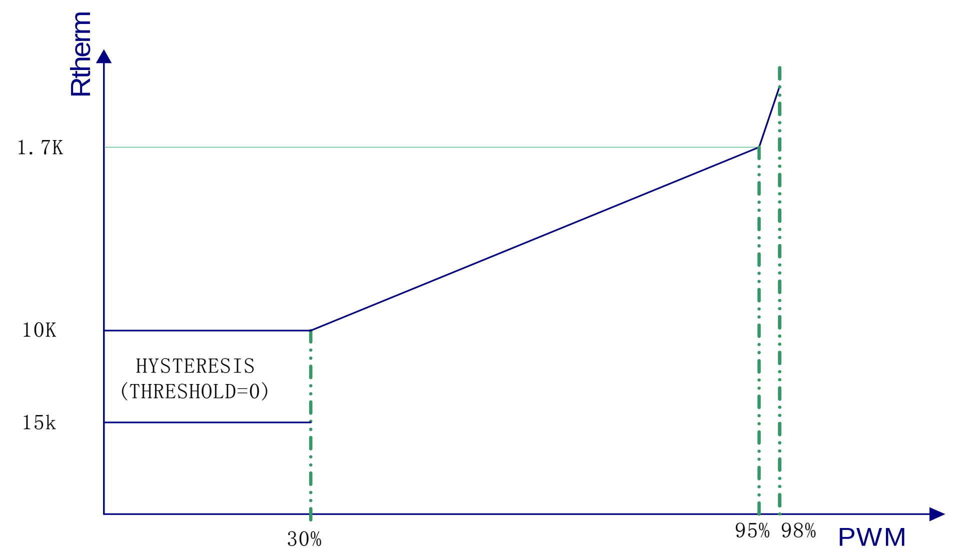

Fig 5. PWM-Rtherm chart

4.4 Monitoring and processing of failure

The failure of fans would lead to obvious rise in a short time for part temperature of electronic equipment, eventually thermally destroying the whole system. This design features functions of monitoring and processing the failures of motors so as to prevent the thermal damage of system.

During the operation of the fans, the V_{sense} pins connected with each fan can monitor the fan operation. When the output of PWM signals drives the motor, the V_{sense} pins maintain a high level, which indicates the blockage of motor or disconnection of the fans or failure of field effect transistor. When the LPC917 detects continual failure on the V_{sense} signals the PWM drive signals will stop, and a LED indicator will flash and alarm.

In addition, when the resistance value is lowered to below 1.5 kΩ, namely, the system environment exceeds 76 °C, the system will also send alarm signal, at the same time, PWM signals with maximal duty cycle (98 %) are used to drive all fans. When the temperature drops to below 76 °C, overheat indication will disappear automatically.

Table 2: Tactics for treatment of fan failure

Fault type	Condition	Temperature	Dealing method
Overheat warning		Temp > 76 °C	98 % PWM to all fans
Fan failure	1 fan failure	Temp ≤ 40 °C	No PWM drive to the failure fan and 50 % PWM to other fans
		40 °C < temp ≤ 50 °C	No PWM drive to the failure fan and 65 % PWM to other fans
		50 °C < temp ≤ 60 °C	No PWM drive to the failure fan and 80 % PWM to other fans
		Temp > 60 °C	No PWM drive to the failure fan and 98 % PWM to other fans
	2 fans failure	Temp ≤ 40 °C	No PWM drive to the failure fan and 50 % PWM to other fans
		40 °C < temp ≤ 50°C	No PWM drive to the failure fan and 65 % PWM to other fans
		Temp > 50 °C	No PWM drive to the failure fan and 98 % PWM to other fans
	3 or 4 fans failure	Temp ≤ 40 °C	No PWM drive to the failure fan and 50 % PWM to other fans
		Temp > 40 °C	No PWM drive to the failure fan and 98 % PWM to other fans
	5 fans failure		No PWM drive

4.5 AD conversion in temperature measurement

The temperature measurement is taken with the on-chip ADC, a software lookup table is implemented to get a linear relationship between ambient temperature and PWM duty cycle. The output of the PWM signals can be obtained through looking up this table according to the result of AD conversion.

The software lookup table is the key for the software design.

4.6 Definition of function of indicator

Constant lighting of red LED: system power on, normal operation.

Red LED flashing in low frequency (2 Hz): failure.

5. Appendix A - Reference software

```
//*****************************************************************************
//*                   Five Fan Controller
//*****************************************************************************

//*****************************************************************************
//*   Philips BU MMS China(Shanghai)
//*   Description:  P89LPC917 (internal RC clock, external reset disable, watchdog
enable)
//*                 500ms Delay Turn-On for the next fan
//*                 Turning on at 25C with a 30% duty cycle
//*                 Initial 65% PWM driving when turning on (Duration:2.5s)
//*                 Using 1 thermistor (1 8bit ADC)
//*                 Fault Management according to different fault types
//*                 Hysteresis (30% PWM dutycycle) at 10 k(25C) to 15k (20C)
//*                 LED flicker display error (2Hz)
//*                 PWM cycle 30HZ (30% to 98%)
//*
//* Modifier:       Date :
//*****************************************************************************
#include <reg917.h>
//*****************************************************************************
//* Functions
//*****************************************************************************
void init(void);               // initial program
void Get_Thermistor(void);     // ADC block
void PollFeedback(void);       // errors detect and output
void main(void);               // main loop program

//*****************************************************************************
//* Variables
//*****************************************************************************
#define MaxReading 10;            // Average ADC Reading Number
#define MaxTimerReload 80;           // PWM Control Number - according to 30Hz
#define FaultCounter 15;             // Fault count duration: 15/30=0.5s
unsigned char code ADCTable[93];     // AvgAD to PWM table

// PWM control variable
unsigned char PWMreload1, PWM1;
unsigned char counter;

// 500ms Turn-on Delay control variable
unsigned char Delaycounter;
unsigned char t1count;

// FanOn status variable
bit Fan1On = 0;                   // indicate fan? running
bit Fan2On = 0;
bit Fan3On = 0;
bit Fan4On = 0;
bit Fan5On = 0;

bit InitialOn = 0;               // indicate initial fan turn-on
bit GlobalOn = 0;             // indicate global fan running
bit GlobalOn1 = 0;
bit GlobalFan1On = 0;
bit GlobalFan2On = 0;
bit GlobalFan3On = 0;
bit GlobalFan4On = 0;
bit GlobalFan5On = 0;

// Feedback Test variable
bit TestEnable = 0;
bit Feedback1Low = 0;
bit Feedback1High = 0;
bit Feedback2Low = 0;
bit Feedback2High = 0;
bit Feedback3Low = 0;
bit Feedback3High = 0;
bit Feedback4Low = 0;
bit Feedback4High = 0;
bit Feedback5Low = 0;
bit Feedback5High = 0;

unsigned char Fault1Counter;     // Fault count duration: 15/30=0.5s
unsigned char Fault2Counter;
unsigned char Fault3Counter;
unsigned char Fault4Counter;
unsigned char Fault5Counter;

bit Fan1Jam = 0;                    // Fan Jam indication
bit Fan2Jam = 0;
bit Fan3Jam = 0;
bit Fan4Jam = 0;
bit Fan5Jam = 0;

// ADC module parameters
unsigned char ADCreading;
unsigned char AvgAD3;
unsigned int TempAD3 = 0;
bit ADCdone = 0;
unsigned char ADCTimer = 240;       // ADC running interval£¬240*416.67us= 100ms

unsigned char nomb;              // used for ADC Table

// Define LedFlicker Freq
bit LedFlicker = 0;              // Fan Jam indication
bit LedTempFlicker = 0;             // Overtemperature indication
unsigned int Delay250ms;            // 600*416.67=250ms

// Define the  Pinout
```

```
sbit Fan1Drive = P1^1;            // Fan PWM Drive Pin
sbit Fan2Drive = P1^0;
sbit Fan3Drive = P0^7;
sbit Fan4Drive = P0^3;
sbit Fan5Drive = P0^1;

sbit Sensor1 = P0^0;                 // Fan Sensor Pin
sbit Sensor2 = P2^2;
sbit Sensor3 = P1^4;
sbit Sensor4 = P1^3;
sbit Sensor5 = P1^2;

sbit TempTest = P0^4;                // temperature detect pin
sbit StateLed = P0^2;                // status LED pin
sbit FunDefine = P0^5;            // Function define pin

//*****************************************************************************
//* init()
//* Input(s) : none.
//* Returns : none.
//* Description : initialization of P89LPC917
//*****************************************************************************
void init(void)
{
// CPU clock=7.373MHz,Pclk=7.373/2=3.6865MHz
  Fan1Drive = 1;            // Fan1 off, initial state
  Fan2Drive = 1;            // Fan2 off, initial state
  Fan3Drive = 1;            // Fan3 off, initial state
  Fan4Drive = 1;            // Fan4 off, initial state
  Fan5Drive = 1;            // Fan5 off, initial state

  Fan1On = 0;
  Fan2On = 0;
  Fan3On = 0;
  Fan4On = 0;
  Fan5On = 0;
  InitialOn = 0;
  GlobalOn = 0;
  GlobalOn1 = 0;
  GlobalFan1On = 0;
  GlobalFan2On = 0;
  GlobalFan3On = 0;
  GlobalFan4On = 0;
  GlobalFan5On = 0;
  Fan1Jam = 0;
  Fan2Jam = 0;
  Fan3Jam = 0;
  Fan4Jam = 0;
  Fan5Jam = 0;

  StateLed = 0;               // LED on, initial state, indicating work normally
  LedFlicker = 0;
  LedTempFlicker = 0;

// define P0,P1,P2 input&output mode
// ports  00 =Quasi-Bi, 01= Push-pull, 10 =Input only, 11 =Open drain
  P0M1 = 0x71;
  P0M2 = 0x8E;
  P1M1 = 0xFC;
  P1M2 = 0x03;
// P2M1 = 0xFF;
// P2M2 = 0x00;

// WDT init
  WDL = 0xFF;                   // watchdog load
  WDCON = 0x25;                 // watchdog control
  WFEED1 = 0xA5;                // watchdog feed1
  WFEED2 = 0x5A;                // watchdog feed2

// power control
  PCONA = 0xA2;                 // power control register A

// initialize parameters
  counter = 0;
  Delaycounter = 0;
  t1count = 0;
  ADCreading = MaxReading;
  Delay250ms = 600;
  AvgAD3 = 180;

//AD convert init
  ADCON1 = 0x04;                // enable ADC1
  ADMODA = 0x10;                // mode: fix channel single conversion
  ADMODB = 0x40;                // divide by 3
  ADINS = 0x80;                 // channel 3 for ADC1
  ADCON1 |= 0x01;            // immiedate start ADC

//timers init (pclk=3.6865MHz,this timer is upward timer.)
  TH0 = 0xF9;
  TL0 = 0xFF;                   // 63999d
  TMOD = 0x11;               // timer0 16-bit mode
  TR0 = 1;                      // turn on timer0
  TR1 = 0;

//interrupt init
  EA = 1;                       // enable interrupts
  ET0 = 1;                      // enable timer0 interrupt
  ET1 = 0;                      // disable timer1 interrupt
}

//****************************************************************************
//* main()
//* Input(s) : none.
//* Returns : none.
//* Description : main loop
//******************************************************************************
void main(void)
{
  init();                         // Initialize part

  while(1)                        // loop forever
  {
   EA = 0;                      // Watchdog
   WDL = 0xFF;
   WDCON = 0x25;
    WFEED1 = 0xA5;
    WFEED2 = 0x5A;
   EA = 1;

    if(!(Fan1Jam && Fan2Jam && Fan3Jam && Fan4Jam && Fan5Jam) && ADCdone)    // if ADC
conversion over
    {
      Get_Thermistor();                                              // Get thermistor
and control fans
    }
   if((Fan1On || Fan2On || Fan3On || Fan4On || Fan5On) && TestEnable)       // Check
fan status
    {
      PollFeedback();
    }
  }
}

//******************************************************************************
//* PollFeedback()
//* Input(s) : none.
//* Returns : none.
//* Description : Check fan status
//******************************************************************************
void PollFeedback(void)
{
  while (1)
  {
   if (Fan1On)
    {
     if (Sensor1)                      // Check Fan Run Status
        {
          Feedback1High = 1;
        }
     else
        {
          Feedback1Low = 1;
        }
   if (Feedback1High && Feedback1Low)
       {
        Fault1Counter = FaultCounter;
      }
  }
else
 {
      Fault1Counter = FaultCounter;
 }

if (Fan2On)
 {
    if (Sensor2)
      {
         Feedback2High = 1;
       }
    else
      {
         Feedback2Low = 1;
      }
    if(Feedback2High && Feedback2Low)
      {
       Fault2Counter = FaultCounter;
      }
 }
else
 {
      Fault2Counter = FaultCounter;
 }

if (Fan3On)
 {
      if (Sensor3)
      {
      Feedback3High = 1;
      }
      else
      {
       Feedback3Low = 1;
      }
      if(Feedback3High && Feedback3Low)
      {
       Fault3Counter = FaultCounter;
      }
 }
else
 {
    Fault3Counter = FaultCounter;
 }

if (Fan4On)
```

```
  {
     if (Sensor4)
     {
      Feedback4High = 1;
      }
     else
     {
      Feedback4Low = 1;
     }
     if(Feedback4High && Feedback4Low)
     {
     Fault4Counter = FaultCounter;
     }
 }
 else
 {
    Fault4Counter = FaultCounter;
 }

if (Fan5On)
 {
     if (Sensor5)
      {
      Feedback5High = 1;
      }
     else
     {
      Feedback5Low = 1;
     }
     if(Feedback5High && Feedback5Low)
      {
      Fault5Counter = FaultCounter;
     }
 }
else
 {
    Fault5Counter = FaultCounter;
 }

if (!TestEnable)
{
   Fault1Counter--;
   Fault2Counter--;
   Fault3Counter--;
   Fault4Counter--;
   Fault5Counter--;

    if (!Fault1Counter)           // FaultCounter has recorded >15 failures
   {
     Fan1On = 0;
     LedFlicker = 1;
```

```
             Fan1Jam=1;
           }
          if (!Fault2Counter)
          {
             Fan2On = 0;
             LedFlicker = 1;
             Fan2Jam=1;
            }
           if (!Fault3Counter)
           {
             Fan3On = 0;
             LedFlicker = 1;
             Fan3Jam=1;
           }
          if (!Fault4Counter)
           {
             Fan4On = 0;
             LedFlicker = 1;
             Fan4Jam=1;
           }
          if (!Fault5Counter)
          {
             Fan5On = 0;
             LedFlicker = 1;
             Fan5Jam=1;
           }
           break;
         }
    }

    Feedback1High = 0;
    Feedback1Low = 0;
    Feedback2High = 0;
    Feedback2Low = 0;
    Feedback3High = 0;
    Feedback3Low = 0;
    Feedback4High = 0;
    Feedback4Low = 0;
    Feedback5High = 0;
    Feedback5Low = 0;
}

//****************************************************************************
//* Get_Thermistor()
//* Input(s) : none.
//* Returns : none.
//* Description : according to the ADC result to get PWM value.
//*               100ms AD Time Cycle, 1s AvgAD Time Cycle.
//****************************************************************************
void Get_Thermistor(void)
{
       // Get Average AD data
       TempAD3 += AD1DAT3;
       ADCreading--;

       if (!ADCreading)
       {
          TempAD3 = TempAD3/10;
          ADCreading = MaxReading;
          AvgAD3 = TempAD3;
          TempAD3 = 0;
          // AvgAD3 = 126;   // for test
        }

       GlobalOn1 = GlobalOn;                        // Last status

       if ((AvgAD3 < 127)&&(GlobalOn == 0))         // GlobalOn indicate the running
status of global fans
       {
          GlobalOn = 1;
       }

       if ((!GlobalOn1) && GlobalOn)                // InitialOn indicate Turn-On of
fans
       {
          InitialOn = 1;
       }

       if (AvgAD3 < 34)                             // Overheating >75C
       {
          PWMreload1 = 2;                           // 98% PWM

          if((!Fan1Jam) && GlobalFan1On)
             Fan1On = 1;                            // Turn on Fan1
          if((!Fan2Jam) && GlobalFan2On)
             Fan2On = 1;
          if((!Fan3Jam) && GlobalFan3On)
             Fan3On = 1;
          if((!Fan4Jam) && GlobalFan4On)
             Fan4On = 1;
          if((!Fan5Jam) && GlobalFan5On)
             Fan5On = 1;

          LedTempFlicker = 1;                       // Overheating indication
       }
       else if ((AvgAD3 >= 127)&&(FunDefine))       // Temp < 25C, FunDefine=1
       {
          if(!Fan1Jam)
             Fan1On = 0;                            // Turn off Fan1
          if(!Fan2Jam)
             Fan2On = 0;
          if(!Fan3Jam)
```

```
            Fan3On = 0;
         if(!Fan4Jam)
            Fan4On = 0;
         if(!Fan5Jam)
            Fan5On = 0;

         GlobalOn = 0;
         GlobalFan1On = 0;
         GlobalFan2On = 0;
         GlobalFan3On = 0;
         GlobalFan4On = 0;
         GlobalFan5On = 0;
         LedTempFlicker = 0;
      }
      else if ((AvgAD3 >= 127)&&(AvgAD3 <= 153)&&(!FunDefine)&&(GlobalOn))
   //Hysterisis, hold PWM at 30%
      {
         PWMreload1 = 56;
         if((!Fan1Jam) && GlobalFan1On)
            Fan1On = 1;
         if((!Fan2Jam) && GlobalFan2On)
            Fan2On = 1;
         if((!Fan3Jam) && GlobalFan3On)
            Fan3On = 1;
         if((!Fan4Jam) && GlobalFan4On)
            Fan4On = 1;
         if((!Fan5Jam) && GlobalFan5On)
            Fan5On = 1;

         LedTempFlicker = 0;
      }
      else if ((AvgAD3 >= 127)&&(AvgAD3 <= 153)&&(!FunDefine)&&(!GlobalOn))
   //20C<Temp< 25C,Initial turn on
      {
         if(!Fan1Jam)
            Fan1On = 0;
         if(!Fan2Jam)
            Fan2On = 0;
         if(!Fan3Jam)
            Fan3On = 0;
         if(!Fan4Jam)
            Fan4On = 0;
         if(!Fan5Jam)
            Fan5On = 0;

         GlobalFan1On = 0;
         GlobalFan2On = 0;
         GlobalFan3On = 0;
         GlobalFan4On = 0;
         GlobalFan5On = 0;
         LedTempFlicker = 0;
```

```
}
else if ((AvgAD3 > 153)&&(!FunDefine))               // Temp < 20C, FunDefine=0
{
if(!Fan1Jam)
   Fan1On = 0;
if(!Fan2Jam)
   Fan2On = 0;
if(!Fan3Jam)
   Fan3On = 0;
if(!Fan4Jam)
   Fan4On = 0;
if(!Fan5Jam)
   Fan5On = 0;

   GlobalOn = 0;
   GlobalFan1On = 0;
   GlobalFan2On = 0;
   GlobalFan3On = 0;
   GlobalFan4On = 0;
   GlobalFan5On = 0;
   LedTempFlicker = 0;
}
else if ((AvgAD3 < 127)&&(AvgAD3 >= 34))           // 25C < Temp < 75C
{
   nomb = 0x7E;
    nomb -= AvgAD3;
    PWMreload1 = ADCTable[nomb];
   if((!Fan1Jam) && GlobalFan1On)
      Fan1On = 1;
   if((!Fan2Jam) && GlobalFan2On)
      Fan2On = 1;
   if((!Fan3Jam) && GlobalFan3On)
      Fan3On = 1;
   if((!Fan4Jam) && GlobalFan4On)
      Fan4On = 1;
   if((!Fan5Jam) && GlobalFan5On)
      Fan5On = 1;

   LedTempFlicker = 0;
}
else                               // error state
{
if(!Fan1Jam)
   Fan1On = 0;
if(!Fan2Jam)
   Fan2On = 0;
if(!Fan3Jam)
   Fan3On = 0;
if(!Fan4Jam)
   Fan4On = 0;
if(!Fan5Jam)
```

```
        Fan5On = 0;

        GlobalOn = 0;
        GlobalFan1On = 0;
        GlobalFan2On = 0;
        GlobalFan3On = 0;
        GlobalFan4On = 0;
        GlobalFan5On = 0;
        LedTempFlicker = 1;
     }
     ADCdone = 0;                      // ADC Reading Complete
    // Turn-on Ordinal 500ms Delay control
     if (InitialOn)
     {
         GlobalFan1On = 1;
         t1count++;
         if (Delaycounter == 1)
             GlobalFan2On = 1;
         else if(Delaycounter == 2)
             GlobalFan3On = 1;
         else if(Delaycounter == 3)
             GlobalFan4On = 1;
         else if(Delaycounter == 4)
             GlobalFan5On = 1;

         if (t1count == 5)
         {
             Delaycounter++;
             t1count = 0;
         }

          if (Delaycounter == 5)
         {
             InitialOn = 0;
             Delaycounter = 0;
         }
      }
}

//******************************************************************************
//* timer0Int()
//* Input(s) : none.
//* Returns : none.
//* Description : CPU clock=7.373MHz,Pclk=7.373/2=3.6865MHz
//*                timer0 interrupt(416.67us)
//*                output the PWM waveform to fans according to the Status
//******************************************************************************
void timer0Int(void) interrupt 1
{
  unsigned char tempa;
  unsigned char fancounter;
```

```
    TH0 = 0xF9;                        // reload values for 30Hz     63999d
    TL0 = 0xFF;

   if (Fan1On || Fan2On || Fan3On || Fan4On || Fan5On)
    {
       if(!counter)                    // reload counter value on underflow
       {
         counter = MaxTimerReload;
        fancounter = 0;

      // Loading PWM1 value according to various Fan On and PWMreload1 Value
        if(Fan1On)
           fancounter = fancounter++;
        if(Fan2On)
           fancounter = fancounter++;
        if(Fan3On)
           fancounter = fancounter++;
        if(Fan4On)
           fancounter = fancounter++;
        if(Fan5On)
           fancounter = fancounter++;

        if(InitialOn)
        {
          PWM1 = 28;                        // Initial Big 65% PWM turn on drive!
Duration:500ms*5=2.5s
        }
        else
        {
           if(fancounter == 5)               // Normal Operation
              PWM1 = PWMreload1;
           else if(fancounter == 4)          // One fan failure Operation
           {
              if(PWMreload1 >= 40)              // temperature <= 40c, 50% PWM, 40c ->
PWMreload1 40
                 PWM1 = 40;
              else if(29 <= PWMreload1 < 40)  // 40c <= temperature <= 50c, 65% PWM, 50c
-> PWMreload1 29
                 PWM1 = 28;
              else if(18 <= PWMreload1 < 29)  // 50c <= temperature <= 60c, 80% PWM, 60c
-> PWMreload1 18
                 PWM1 = 16;
              else if(PWMreload1 < 18)        // temperature > 60c, 98% PWM
                 PWM1 = 2;
            }
           else if(fancounter == 3)          // Two fans failure Operation
           {
              if(PWMreload1 >= 40)              // temperature <= 40c, 50% PWM, 40c ->
PWMreload1 40
                 PWM1 = 40;
              else if(29 <= PWMreload1 < 40) // 40c <= temperature <= 50c, 65% PWM, 50c -
> PWMreload1 29
                 PWM1 = 28;
              else if(PWMreload1 < 29)       // temperature > 50c, 98% PWM
                 PWM1 = 2;
            }
           else                              // Three or Four fans failure Operation
           {
              if(PWMreload1 >= 40)              // temperature <= 40c, 50% PWM, 40c ->
PWMreload1 40
              PWM1 = 40;
              else if(PWMreload1 < 40)       // temperature > 40c, 98% PWM
              PWM1 = 2;
            }
         }
      }

       if(counter <= PWM1)
       {
           Fan1Drive = 1;                    // close fan
           Fan2Drive = 1;
           Fan3Drive = 1;
           Fan4Drive = 1;
           Fan5Drive = 1;
       }
       else
       {
         if(Fan1On)
           Fan1Drive = 0;              // turn on fan
        else
           Fan1Drive = 1;
        if(Fan2On)
           Fan2Drive = 0;              // turn on fan
        else
           Fan2Drive = 1;
        if(Fan3On)
           Fan3Drive = 0;              // turn on fan
        else
           Fan3Drive = 1;
        if(Fan4On)
           Fan4Drive = 0;              // turn on fan
        else
           Fan4Drive = 1;
        if(Fan5On)
           Fan5Drive = 0;              // turn on fan
        else
           Fan5Drive = 1;
       }

       if ((counter <= 78)&&(counter > (PWM1+2)))    // range of fail detection
       {
        TestEnable = 1;
       }
       else
       {
        TestEnable = 0;
       }
       counter--;
    }
    else
    {
        Fan1Drive = 1;                          // close fan
       Fan2Drive = 1;
       Fan3Drive = 1;
       Fan4Drive = 1;
       Fan5Drive = 1;

       TestEnable = 0;
       counter = 0;
    }

    if (LedFlicker || LedTempFlicker)           // LED flicker
    {
        if (!Delay250ms)
        {
          StateLed =~ StateLed;
          Delay250ms = 600;
        }
       Delay250ms--;
    }
    else
    {
       StateLed = 0;                            // open LED
    }
    tempa =ADCON1;
    if (tempa & 0x08)                        // ADC convertion complete
    {
     ADCON1 &= ~0x08;                           // clear convertion complete bit
     ADCdone = 1;
    }
    if (!ADCTimer)
    {
      ADCON1 |= 0x01;                        // immediately start ADC!
     ADCTimer = 240;
    }
    ADCTimer--;
  }

  //*****************************************************************************
  //*ADC lookup table
  //*20c<temperature <=75C,1.5K<resister<10K, sequence 126->34
  //*****************************************************************************
```

```
unsigned char code ADCTable[93]={
55,54,53,52,51,50,49,48,47,46,
46,45,44,43,42,42,41,40,39,39,
38,37,36,36,35,34,34,33,32,32,
31,30,30,29,29,28,27,27,26,26,
25,24,24,23,23,22,22,21,21,20,
20,19,19,18,18,17,17,16,16,15,
15,15,14,14,13,13,12,12,12,11,
11,10,10,10,9,9,8,8,8,7,
7,6,6,6,5,5,5,4,4,3,
3,2,2
};
```

6. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

7. Trademarks

Notice — All referenced brands, product names, service names and trademarks are the property of their respective owners.

8. Contents

Date of release:07 December 2005
Document number: AN10388_2

Published in The Netherlands