ISSN 1726-5479

SENSORS & TRANSDUCERS

1 /09 vol. 100

vol. 100

Sensor Instrumentation, DAQ and Virtual Instruments

International Frequency Sensor Association Publishing

IFSA

Sensors & Transducers

Volume 100
January 2009

www.sensorsportal.com

ISSN 1726-5479

Sensors & Transducers Journal (ISSN 1726-5479) is a peer review international journal published monthly online by International Frequency Sensor Association (IFSA).
Available in electronic and CD-ROM.

Sensors & Transducers Journal

Contents

Volume 100
Issue 1
January 2009

www.sensorsportal.com

ISSN 1726-5479

Editorial

International Frequency Sensor Association (IFSA) Celebrates the 10th Anniversary I
Sergey Y. Yurish

Research Articles

A Log Amplifier Based Linearization Scheme for Thermocouples
Nikhil Mondal, A. Abudhahir, Sourav Kanti Jana,Sugata Munshi and D. P. Bhattacharya 1

Uncertainty Analysis of Thermocouple Circuits
B. Vasuki, M. Umapathy, S. K. Velumani 11

Calibration System for Thermocouple Testing
Dragan R. Milivojevic, Visa Tasic, Marijana Pavlov, Zoran Andjelkovic 16

Embedded Processor Based Automatic Temperature Control of VLSI Chips
Narasimha Murthy Yayavaram, Saritha Chappidi, Sukanya Velamakuri 27

Field of Temperature Measurement by Virtual Instrumentation
Libor Hargaš, Dušan Koniar, Miroslav Hrianka, Jozef Čuntala 45

Analyzing Electroencephalogram Signal Using EEG Lab
Mukesh Bhardwaj and Avtar. K. Nadir 51

New Aspects in Respiratory Epithelium Diagnostics Using Virtual Instrumentation
Dušan Koniar, Libor Hargaš, Miroslav Hrianka, Peter Bánovčin 58

A PC-based Technique to Measure the Thermal Conductivity of Solid Materials
Alety Sridevireddy, K. Raghavendra Rao. 65

A New Wide Frequency Band Capacitance Transducer with Application to Measuring Metal Fill Time
Wael Deabes, Mohamed Abdelrahman, and Periasamy K. Rajan 72

A Novel Hall Effect Sensor Using Elaborate Offset Cancellation Method
Vlassis N. Petoussis, Panos D. Dimitropoulos and George Stamoulis 85

A Review of Material Properties Estimation Using Eddy Current Testing and Capacitor Imaging
Mohd. Amri Yunus, S. C. Mukhopadhyay and G. Sen Gupta 92

Surface Plasmon Resonance Based Fiber Optic Sensor with Symmetric and Asymmetric Metallic Coatings: a Comparative Study
Smita Singh, Rajneesh K. Verma and B. D. Gupta 116

Increasing of Excursion Range of Absolute Optical Sensors Intended for Positioners
Igor Friedland, Ioseph Gurwich, Amit Brandes.. 125

Field-Effect-Transistor Behavior of a Multiwall Carbon Nano Fiber Directly Grown on Nickel Electrodes
L. W. Chang, P. S. Wu, J. T. Lue and Z. P. Chen.. 137

Classification of Fiber-Optic Pressure Sensors with Amplitude Modulation of Optical Signal
Vladyslav Kondratov, Vitalii Redko.. 146

New e-Book

Laboratories of Instrumentation for Measurement
Maria Teresa Restivo, Fernando Gomes de Almeida, Maria de Fátima Chouzal, Joaquim Gabriel Mendes, António Mendes Lopes.. 161

Authors are encouraged to submit article in MS Word (doc) and Acrobat (pdf) formats by e-mail: editor@sensorsportal.com
Please visit journal's webpage with preparation instructions: http://www.sensorsportal.com/HTML/DIGEST/Submition.htm

Sensors & Transducers Journal, Vol. 100, Issue 1, January 2009, pp. 27-44

Sensors & Transducers

ISSN 1726-5479

http://www.sensorsportal.com

Embedded Processor Based Automatic Temperature Control of VLSI Chips

Narasimha Murthy Yayavaram, Saritha Chappidi, Sukanya Velamakuri
Department of Electronics, S.S.B.N. College (Autonomous), Anantapur, A.P. India
E-mail: yayavaram@yahoo.com

Received: 7 October 2008 /Accepted: 19 January 2009 /Published: 26 January 2009

Abstract: This paper presents embedded processor based automatic temperature control of VLSI chips, using temperature sensor LM35 and ARM processor LPC2378. Due to the very high packing density, VLSI chips get heated very soon and if not cooled properly, the performance is very much affected. In the present work, the sensor which is kept very near proximity to the IC will sense the temperature and the speed of the fan arranged near to the IC is controlled based on the PWM signal generated by the ARM processor. A buzzer is also provided with the hardware, to indicate either the failure of the fan or overheating of the IC. The entire process is achieved by developing a suitable embedded C program. *Copyright © 2009 IFSA.*

Keywords: Temperature sensor, ARM processor, VLSI chips, Brushless DC motor

1. Introduction

With the phenomenal developments in VLSI technology, the ambitious IC designers are trying to put more transistors in to smaller packages. So, the ICs run at higher speeds and produce large amount of heat which creates the problem of thermal management. For example, nowadays the CPU chips are becoming smaller and smaller with almost no room for the heat to escape. The total power dissipation levels now reside on the order of 100 W with a peak power density of 400-500 W/Cm^2, and are still steadily climbing [1]. As the chip temperature increases its performance is very much degraded by parameters shift, decrease in operating frequencies and out-of specification of timings. So the high speed chips must be cooled to maintain good performance for the longest possible operating time and over the widest possible range of environmental conditions. The maximum allowable temperature for a high speed chip to meet its parametric specifications depends on the process and how the chip is designed.

Among the various cooling techniques, heat sinks, heat pipes, fans and clock throttling are usually employed. Among these techniques, fans can dramatically reduce the temperature of a high speed chip, but they also generate a great deal of acoustic noise. This noise can be reduced significantly by varying the fans speed based on temperature i.e. the fan can turn slowly when the temperature is low and can speed up as the temperature increases.

The other prominent method is clock throttling i.e. reducing the clock speed to reduce power dissipation. But it also reduces the system performance and the systems functionality is lost [2].

So, the objective of the present work is, to design a hardware system consisting of a brushless DC motor fan whose speed is controlled based on the temperature of the chip, sensed by the sensor LM35. The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in °Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1⁄4°C at room temperature and ±3⁄4°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package [3]. To monitor the voltage at the terminals of the DC motor fan, the PWM signal is generated by the ARM7TDMI processor. This PWM signal is changed in accordance to the output of the LM35 temperature sensor. So the important component of this entire project is the temperature sensor.

2. Description

In ARM processor based automatic temperature control system, the output of the temperature sensor is fed to the on chip ADC and the output of the ADC is given to the L293D driver IC which in turn is fed to DC motor fan as shown in the block diagram in Fig. 1. A graphic LCD (128x64 pixels) is interfaced to the ARM LPC 2378 processor to display the temperature of the IC and the speed of the fan. A buzzer is also connected to the processor which gives an indication, in case of the failure of the fan or overheating of the chip beyond some level. The entire circuit diagram is shown in Fig. 2.

3. Software Description

The present work is implemented using ARM IAR Workbench IDE and the necessary embedded C program is developed and dumped into the embedded processor using Flash magic ISP Utility. The ARM IAR Workbench IDE is a very powerful Integrated Development Environment (IDE) that allows you to develop and manage complete embedded application projects [4]. In-System Programming is programming or reprogramming the on-chip flash memory, using the boot-loader software and a serial port. The LPC2387 microcontroller is based on a 16-bit/32-bit ARM7TDMI-S CPU with real-time emulation that combines the microcontroller with 512 kB of embedded high-speed flash memory.

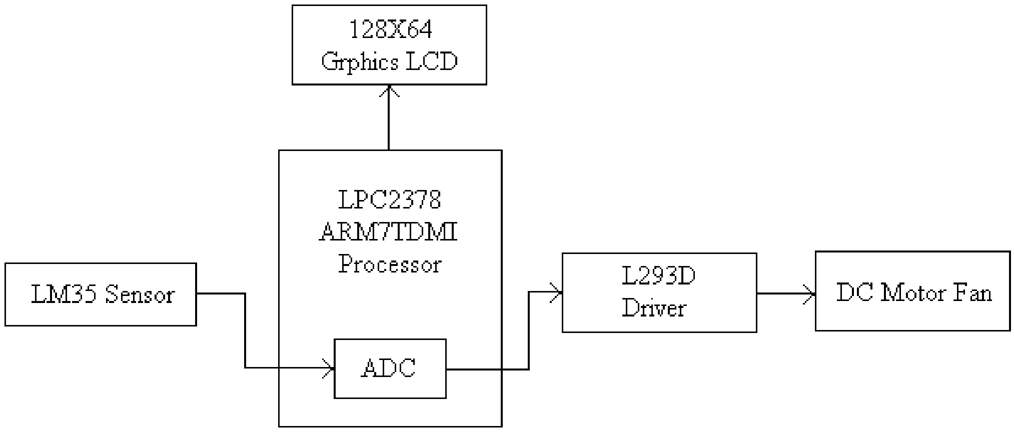

Fig. 1. Block diagram.

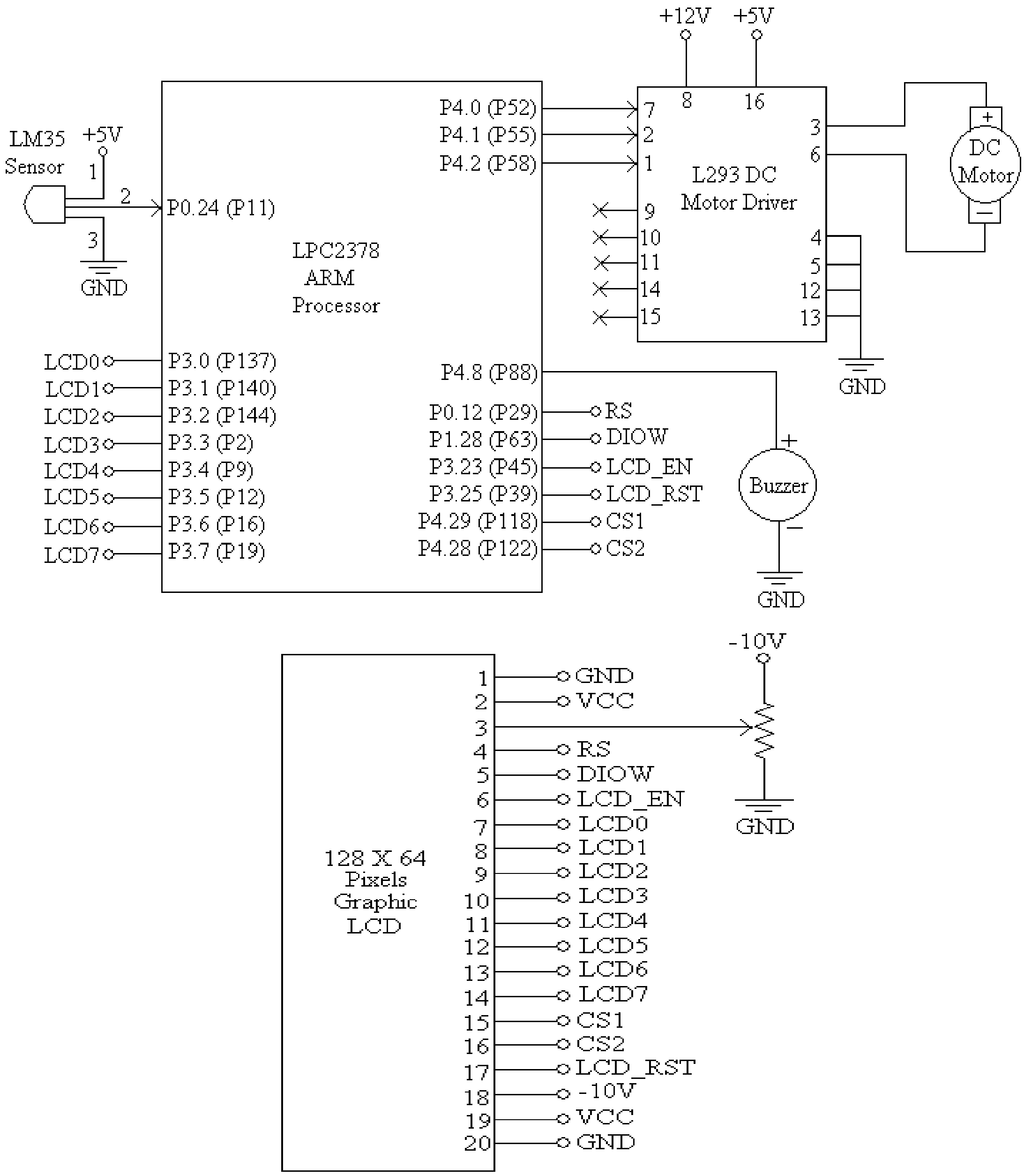

Fig. 2. Circuit Diagram.

A 128-bit wide memory interface and unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical performance in interrupt service routines and DSP algorithms, this increases performance up to 30 % over Thumb mode. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty. The LPC2387 is ideal for multi-purpose serial communication applications. It incorporates a 10/100 Ethernet Media Access Controller (MAC), USB full speed device with 4 kB of endpoint RAM, four UARTs, two CAN channels, an SPI interface, two Synchronous Serial Ports (SSP), three I2C interfaces, and an I^2S interface. This blend of serial communications interfaces combined with an on-chip 4 MHz internal oscillator, 64 kB SRAM, 16 kB SRAM for Ethernet, 16 kB SRAM for USB and general purpose use, together with 2 kB battery powered SRAM makes this device very well suited for communication gateways and protocol converters. Various 32-bit timers, an improved 10-bit ADC, 10-bit DAC, one PWM unit, a CAN control unit, and up to 70 fast GPIO lines with up to 12 edge or level sensitive external interrupt pins make this microcontroller particularly suitable for industrial control and medical systems.

The LPC2378 Microcontroller provides on-chip boot-loader software that allows programming of the internal flash memory over the serial channel [5]. Philips provides a utility program for In-System programming called Flash magic Software [6].

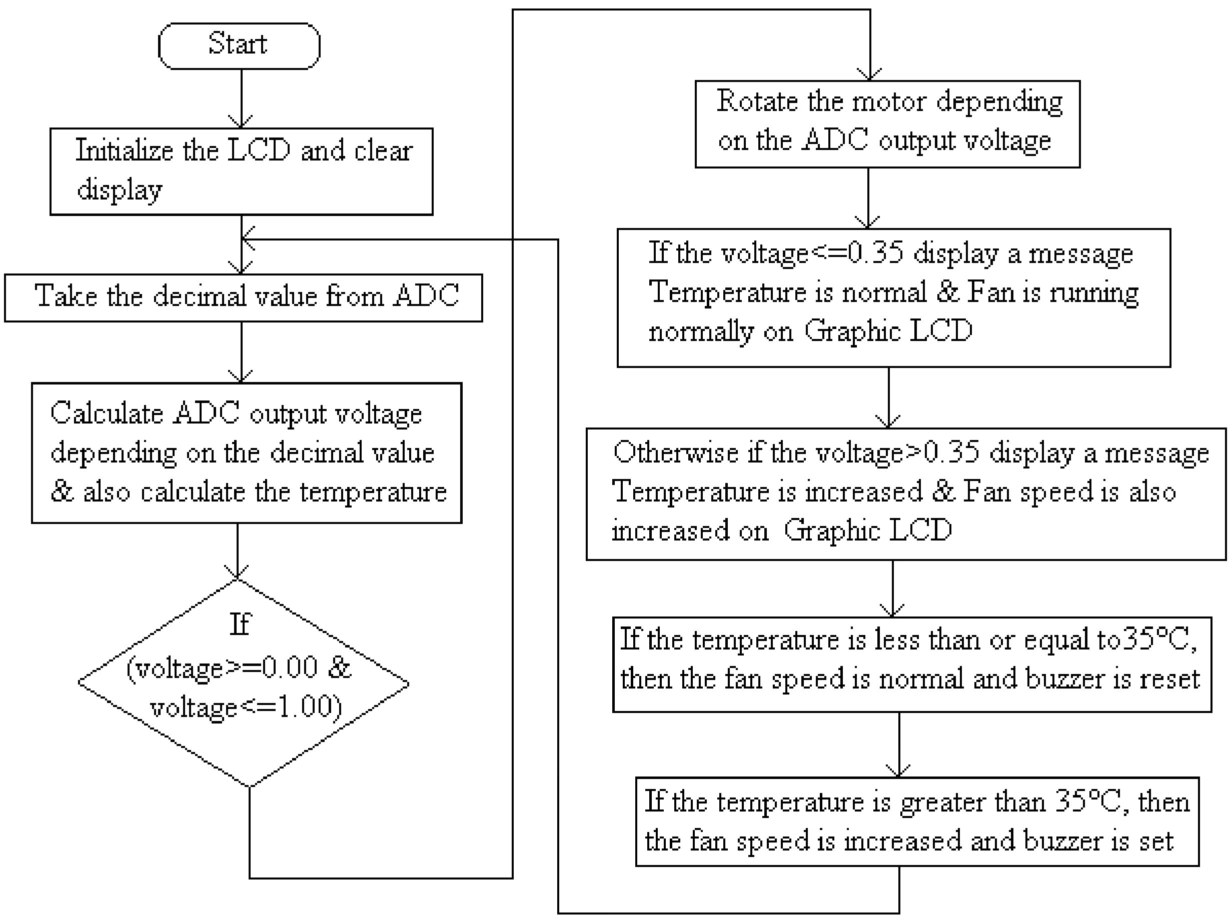

Fig. 3. Flow chart for the program.

4. Results and Conclusions

Embedded ARM processor based automatic speed control DC motor fan is designed and implemented .To test the validity of the design, the temperature sensor is kept inside a small oven and its temperature is increased beyond the room temperature. Now the fan is operated to run with full speed and the temperature is found to comeback to normal temperature. This is repeated with various VLSI chips like Pentium processor, FPGA chips etc. Now the temperature sensor is kept very near to the Pentium processor of the computer and it is observed that, as the time lapses the speed of the fan is automatically increased and the temperature of the chip is found to be controlled. These results are displayed on LCD panel. Though the present system is working well in the given environment, still it is worthwhile to highlight the following conclusions.

Normally, controlling fan speed or clock throttling based on temperature requires that the temperature of the high speed chip should be first measured. This is done by placing a temperature sensor close to the target chip either directly next to it or in some cases, under it or on the heat sink. The temperature measured in this way corresponds to that of the high speed chip, but can be significantly lower and the difference between measured temperature and the actual die temperature increases as the power dissipation increases. So, the temperature of the circuit board or heat sink must be correlated to the die temperature of the high speed chip [7]. Of course a better alternative is possible with a number of high speed chips. Many CPUs, FPGAs and other high speed ICs include a thermal diode which is actually a diode connected bipolar transistor, on the die. Using a remote diode temperature sensor connected to this thermal diode, the temperature of the high speed IC's die can be measured directly with an excellent accuracy. This not only eliminates the large temperature gradients involved in measuring temperature outside the target IC's package, but it also eliminates the long thermal time constants, from several seconds to minutes, that cause delays in responding to die temperature changes.

There is also a drawback in fan speed control. Normally the fan speed is controlled by adjusting the power supply voltage of the fan. This is done by a low-frequency PWM signal, usually in the range of about 50 Hz, whose duty cycle is varied to adjust the fan's speed. This is inexpensive and also efficient. But the disadvantage of this method is that it makes the fan somewhat nosier because of the pulsed nature of the power supply. The PWM waveforms fast edges cause the fans mechanical structure to move, which is easily audible.

In some systems, it is also important to limit the rate of change of the fan speed. This is critical when the system is in close proximity to users. Simply switching a fan on and off or changing speed immediately as temperature changes is acceptable in some environments. But when users are in nearby, the sudden changes in fans noise are highly annoying. So to avoid these effects the fan's drive signal must be limited to an acceptable level.

5. Future Scope of the Work

In the present work temperature is sensed using the temperature sensor LM35 and the speed of the motor is controlled by varying the width of PWM generated by the processor. But the temperature sensed by the IC LM35 is not very accurate even though we keep the IC very near to the processor or VLSI chip. So, we can use a remote diode temperature sensor connected to the thermal diode which measures the temperature of the high speed ICs directly with excellent accuracy. Another important aspect is a variety of remote temperature sensors with up to five sensing channels is available that can detect the die temperature of the high speed chip and transmit temperature data to a microcontroller. Fan speed regulators with multiple channels of fan tachometer monitoring can provide reliable control of fan RPM or supply voltage based on commands from an external microcontroller. For this simple ICs are provided by MAXIM MAX6660 and MAX6653 [8]. The first IC can sense the remote temperature and controls the fan speed based on that temperature. It produces a DC supply voltage for the fan through an internal power transistor. The second IC also performs a similar function but drives the fan with a PWM waveform through an external pass transistor. Both include complete thermal fault monitoring with over temperature outputs, which can be used to shut down the system if the high speed chips get too hot. So, the present work can be improved further by using the above mentioned techniques.

Acknowledgements

The authors wish to acknowledge Sri. PL. N. Reddy, Correspondent, SSBN College (Autonomous), Anantapur, India, for his support & encouragement in promoting the research activities in the Institution.

References

[1]. Ma Kunquan and Liu Jing, Liquid Metal Cooling in Thermal Management of Computer Chips, *Frontiers Energy Power Engineering China,* 2007, Vol.1, No. 4, 2007, pp. 384-402.
[2]. MAXIM Application note 707, Fan speed Control is Cool, Feb. 02, 2001.
[3]. National Semiconductor's Temperature Sensor Hand Book.
[4]. ARM Architecture Reference Manual, *Addison-Wesley*, 2001.
[5]. Philips Semiconductors – LPC2378 Manual.
[6]. Philips ISP Software, LPC 2000 Flash magic, http://www.esacademy.com
[7]. MAXIM Application Note 3173, Automatic Fan Control Techniques: Trends in Cooling High – Speed Chips, July 08, 2004.
[8]. http://maxim-ic.com

Appendix

Note: The readers who are interested to obtain the software should mail to the author by e-mail: yayavaram@yahoo.com

Embedded C Program

main.c

```
#include <iolpc2378.h>
#include "..\header\irq.h"
#include "..\header\config.h"
#include "..\header\all.h"
void main (void)
 {
    unsigned int Fdiv;
    TargetResetInit();
    PCONP |= (1 << 12);
    PINSEL1 = 0X01454000;       //AD0.0 as Analog input
    PINSEL0 = 0x00000050;       // RxD0 and TxD0

    FIO4DIR = 0XFF000FFF;
    FIO3DIR = 0X008000FF;
    IO0DIR  = 0X0021F0FF;
    IO1DIR  = 0XFF000000;
```

```
    AD0CR = ( 0x01 << 0 ) |                                    // SEL=1,select
channel 0~7 on ADC0
          ( ( 1000000 / ADC_CLK - 1 ) << 8 ) |        // CLKDIV = Fpclk / 1000000
- 1
          ( 1 << 16 ) |              //BURST = 0, no BURST, software controlled
          ( 0 << 17 ) |              // CLKS = 0, 11 clocks/10 bits
          ( 1 << 21 ) |              // PDN = 1, normal operation
          ( 0 << 22 ) |              // TEST1:0 = 00
          ( 1 << 24 ) |              // START = 0 A/D conversion stops
          ( 0 << 27 );              // EDGE = 0 (CAP/MAT singal falling,trigger
A/D conversion)

    U0LCR = 0x83;                               // 8 bits, no Parity, 1 Stop bit
    Fdiv = ( 1000000/ 16 ) / 19200 ;    // baud rate
    U0DLM = Fdiv / 256;
    U0DLL = Fdiv % 256;
    U0LCR = 0x03;                                 /* DLAB = 0 */
    U0FCR = 0x07;                                 /* Enable and reset TX and RX
FIFO. */
        I2C0CONCLR=0x6C;
        I2C0CONSET=0x40;                  //Enable I2C.
        I2C0SCLH=110;
        I2C0SCLL=90;
        intlcd();
        clear_display();

        while(1)
         {
                serial_tx();
         }
  }
```

adc.c

```
#include<iolpc2378.h>
#include "..\header\all.h"
#include<stdio.h>
#define ADC_CLK     1000000                   /* set to 1Mhz */

unsigned char ch1;
unsigned long int num;
unsigned int serial_rx();
void serial_tx();
void delay_ms(unsigned int);
unsigned int ADC_VALUE=0;

void Timer_init()
{
  T0TC  = 0;
  T0MCR = 3;
  T0PR  = 59;   // 60/(59+1) = 1 Mhz
  T0MR0 = 1000000;
  T0TCR = 1;
}
unsigned int serial_rx()
  {
  while(U0LSR&0X01)
              {
                  ch1=U0RBR;
                  putchar(ch1);
              }
  return(ch1);
  }
void send_serial_data(unsigned char serial)
```

```
 {
    while((U0LSR & 0x20)==0);
    U0THR = serial;
 }
int putchar(int ch)
  {
    if (ch == '\n')
    {
    while (!(U0LSR & 0x20));
    U0THR = 0x0d;
    }
    while (!(U0LSR & 0x20));
    return (U0THR = ch);
  }
void breaks()
   {
   FIO3PIN = 0X00000000;
   }
void delay_ms(unsigned int i)
  {
    unsigned int j;
      while(i-->0)
       {
       for(j=0;j<4500;j++);
       }
  }
void adc_lcd()
{
  float adc_voltage;
  unsigned int calc;
  clear_display();
start:
   while(1)
    {
       while((AD0GDR & 0X80000000)!=0X80000000);
       num = (AD0GDR>>6)& 0x3ff ;         // convesion data holds AD0DR0[6] to
                                                                    AD0DR0[15]
       adc_voltage=(num*3.3)/1023;
       printf("\n Voltage = %f\t", adc_voltage);
       printf(" Temperature = %f  Degrees", adc_voltage *100);
     if(adc_voltage >=0.00 & adc_voltage <=1.00)
      {
        calc= adc_voltage *100000;

      Timer_init();
      FIO3PIN = 0X00000005;
      while(T0TC!=calc);
      T0TCR = 0;
      Timer_init();
      FIO3PIN = 0X00000000;
      while(T0TC!=100000);
      T0TCR = 0;
       if(adc_voltage <=0.35)
          {
          clear_display();
          dis_border();
          display_string_s("TEMP IS NORMAL",3,22);
          display_string_s("FAN IS RUNNING ",5,24);
          display_string_s("  NORMALLY  ",7,26);

          printf("\nTemperature is Normal...");
          printf("\t Fan is running Normally");
          }
```

```
          else if (adc_voltage >0.35)
          {
           clear_display();
          dis_border();
          display_string_s("TEMP IS INCREASED",3,16);
          display_string_s("FAN SPEED ALSO ",5,24);
          display_string_s("  INCREASED ",7,30);
          printf("\nTemperature is Increased...");
          printf("\t Fan speed also Increased");
          }
     serial_rx();
     if(ch1=='C' | ch1=='c')
      {
        goto start;
      }
     else
     if(ch1=='E' | ch1=='e')
       {
          while(1)
            {
              breaks();
              clear_display();
              dis_border();
              display_string_s("FAN IS STOPPED",3,20);
              printf("\nFan is stopped  ...");
              delay_ms(500);
              serial_rx();
            if(ch1=='C' | ch1=='c')
              {
                goto start;
              }
           }  //end of while
       }  //end of if
     }  //end of if
    }  //end of while
  }  //end of main

void serial_tx()
  {
      unsigned long int Fdiv;
      U0LCR = 0x83;               // 8 bits, no Parity, 1 Stop bit
      Fdiv = ( 12000000 / 16 ) / 19200 ;  //baud rate
      U0DLM = Fdiv / 256;
      U0DLL = Fdiv % 256;
      U0LCR = 0x03;               // DLAB = 0
      U0FCR = 0x07;               // Enable and reset TX and RX FIFO.
      while(1)
      {
       adc_lcd();
      }
   }
```

display.c

```
#include <iolpc2378.h>
#include "..\header\all.h"
#include <string.h>
#define X_ADRESS  0xB8
#define Y_ADRESS  0x40

int va,i;
void busy_check();
extern  char array1[10];
extern  unsigned int vol1,vol2;
```

```
unsigned int col[] ={0x00, 0x00, 0X00, 0x00, 0x0c, 0x0c, 0x00, 0x00, 0x00, 0x00};
unsigned int col1[] ={0x00, 0x00, 0X00, 0x00, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00};

unsigned int Arr[]= { 0X00,0x7e, 0x42, 0x42, 0x42, 0x3c,
                      0X00,0x3c, 0x42, 0x42, 0x42, 0x24,
                      0X00,0x00, 0x28, 0x28, 0x28, 0x00,
                      0X00,0x01, 0x01, 0x7f, 0x01, 0x01 };

char border_arr[1024]={
   0,254,  2,250, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,250,  2,254,  0,
   0,255,  0,255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,  0,255,  0,
   0,255,  0,255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,  0,255,  0,
   0,255,  0,255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,  0,255,  0,
   0,255,  0,255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,  0,255,  0,
   0,255,  0,255,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
```

```
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,  0,255,  0,
   0,127, 64, 95, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
  80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 95, 64,127,  0
};

unsigned char alpha[]= {  0X00,0x7c, 0x12, 0x12, 0x12, 0x7c,  //A
                          0X00,65,   127,  73,   73,   54,                //B
                          0X00,0x3c, 0x42, 0x42, 0x42, 0x24,       //C
                          0X00,0x7e, 0x42, 0x42, 0x42, 0x3c,       //D
                          0X00,0x7e, 0x4a, 0x4a, 0x4a, 0x42,       //E
                          0X00,126,  0x0a, 0x0a, 0x0a, 0x02,        //F
                          0X00,0x3C, 0x42, 0x42, 0x52, 0x70,     //G
                          0X00,0x7e, 0x10, 0x10, 0x10, 0x7E,     //H
                          0X00,0x00, 0x42, 0x7E, 0x42, 0x00,     //I
                          0X00,0x7e, 0x18, 0x24, 0x42, 0x00,      //J
                          0X00,0x7e, 0x18, 0x24, 0x42, 0x00,      //K
                          0X00,0x7e, 0x40, 0x40, 0x40, 0x60,      //L
                          0X00,0x7e, 0x04, 0x18, 0x04, 0x7e,      //M
                          0X00,0x7e, 0x04, 0x08, 0x10, 0x7e,      //N
                          0X00,0x3c, 0x42, 0x42, 0x42, 0x3C,     //O
                          0X00,0x7e, 0x12, 0x12, 0x12, 0x0c,      //P
                          0X00,0x7E, 0x12, 0x12, 0x32, 0x4c,     //Q
                          0X00,0x7E, 0x12, 0x12, 0x32, 0x4c,     //R
                          0X00,0x24, 0x4a, 0x4a, 0x4a, 0x32,      //S
                          0X00,0x02, 0x02, 0x7e, 0x02, 0x02,     //T
                          0X00,0x3E, 0x40, 0x40, 0x40, 0x3E,    //U
                          0X00,0x1E, 0x20, 0x40, 0x20, 0x1E,    //V
                          0x00,126,  64,   120,  64,   126,            //W
                          0X00,195,  36,   24,   36,   195,             //X
                          0X00,0x06, 0x08, 112,  0x08, 0x06,      //Y
                          0X00,99,   81,   73,   69,   99                //Z
   };
unsigned char space[]={0X00,0x00, 0x00, 0x00, 0x00, 0x00} ;

void display_string_s(unsigned char str[],unsigned char row,unsigned char col)
 {
    unsigned int start_pos=0,row_addr,col_addr;
    unsigned int loop,first_half=0,l,c,index=0;

    start_pos  = (col+63) - 64;   // To Identify the Starting Location
    row_addr   = 0xb8 + (row-1);
    col_addr   = col+(64-1);

    l=strlen(str);
    writecmd(row_addr);     //Row Address
    writecmd(col_addr);     //Col Address

   for(loop=0;loop<l;loop++)
    {
         writecmd(row_addr);  //Row Address
         for(first_half = 0; first_half<6;first_half++)
         {
             writecmd(row_addr);  //Row Address
             writecmd(col_addr);  //Col Address

             if(start_pos<64)
```

```
                    IO0CLR  = 0X00010000;   // CS2 DISABLED
                if((start_pos)==64)
                {
                   IO0SET = 0X00010000;
                   writecmd(row_addr);          //Row Address
                   writecmd(0x40);              //Col Address
                   IO0CLR = 0X00008000;
                }
                if(start_pos>64)
                IO0CLR  = 0X00008000;     // CS2 DISABLED
                c = str[loop];
                if(c==0x20)
                {
                   index=(c-0x20)*6;
                   writedata(space[first_half+index]);
                   start_pos = start_pos + 1;
                   col_addr = col_addr+1;
                }
                if(c>64)
                {
                  index = (c-65)*6;
                  writedata(alpha[first_half+index]);
                  start_pos = start_pos + 1;
                  col_addr = col_addr+1;
                }

              }
        }
}
void busy_check()
{
  for(int i=0;i<0x30;i++);
}

void dis_border()
{
    unsigned int page,dat;
       for(page =0;page<8;page++)
      {
        dat = page * 128 ;
              writecmd(0xB8+page);
              writecmd(0x40);
              write_main4(0x10000000,dat);

              writecmd(0xB8+page);
              writecmd(0x40);
              write_main5(0x80000000,dat + 64);
      }
}
void write_main4(unsigned long int cs,unsigned int col)
{
        int i;
        IO0SET = CS1;                        // PAGE 1  IS SELECGTED.
        IO0CLR = CS2;
        for(i=col;i<(64+col);i++)
        writedata(border_arr[i]);
}
void write_main5(unsigned long int cs,unsigned int col)
{
        int i;
        IO0SET = CS2;
        IO0CLR = CS1;
        for(i=col;i<(64 +col);i++)
```

```
        writedata(border_arr[i]);
}

void clearpage()
{
      IO0SET = CS2;
            IO0SET = CS1;
      for(int j=0xbb;j<0xc0;j++)
          {
                  writecmd(j);
                  writecmd(0x40);
                  for (int y=0;y<64;y++)
                  writedata(0x0);
          }
}

void writecmd(unsigned  int ch)
  {
        IO0SET  = 0X00018000;   // CS1 AND CS2 ENABLED
        busy_check();
        IO0CLR  = 0X00200000;   // RS = O
        IO1CLR  = 0X10000000;   // DIOW
        FIO3PIN = ch ;
        FIO3SET = 0X00800000;
        FIO3CLR = 0X00800000;
  }

void writedata(unsigned int ch1)
   {
        busy_check();
      IO0SET  = 0X00200000;   // RS = 0
            IO1CLR  = 0X10000000;   // DIOW
      FIO3PIN = ch1;
      FIO3SET = 0X00800000;
            FIO3CLR = 0X00800000;
    }
void pageclear()
     {
        int i,y;
      IO0SET  = 0X00018000;   // CS1 AND CS2 WNABLED
      for(i=0xb8;i<0xc0;i++)
            {
                  writecmd(i);
                          writecmd(0x40);
                  for (y=0;y<64;y++)
                    writedata(0x00);
            }
     }
void clear_display()
{
      pageclear();
}
void intlcd()
{

      writecmd(0x3e);
      writecmd(0x3f);
      writecmd(0xc0);
      writecmd(0xb8);
  }
```

Header files:

all.h

```
#define ADC_CLK     1000000   /* set to 1Mhz */
#define LCD_EN     0X00800000
#define RS     0X00200000
#define DIOW         0X10000000
#define CS1        0X00008000
#define CS2        0x00010000

void dis_border();
int putchar(int ch);
void serial_tx(void);
void write_main4(unsigned long int cs,unsigned int col);
void write_main5(unsigned long int cs,unsigned int col);
void display_string_s(unsigned char str[],unsigned char row,unsigned char col);
void pageclear();
void clearpage();
void clear_display();
void adc_serial_tx(unsigned int ch);
void adc_lcd();
void send_serial_data(unsigned char serial);
void writecmd(unsigned  int ch);
void busy_check();
void writedata(unsigned int ch1);
void intlcd();
```

irq.h

```
/*************************************************************************
 *   irq.h:  Interrupt related Header file for NXP LPC2378 Family
*************************************************************************/
#ifndef __IRQ_H
#define __IRQ_H

#define I_Bit     0x80
#define F_Bit     0x40

#define SYS32Mode   0x1F
#define IRQ32Mode   0x12
#define FIQ32Mode   0x11

#define HIGHEST_PRIORITY  0x01
#define LOWEST_PRIORITY   0x0F

#define WDT_INT     0
#define SWI_INT     1
#define ARM_CORE0_INT   2
#define ARM_CORE1_INT   3
#define TIMER0_INT    4
#define TIMER1_INT    5
#define UART0_INT   6
#define UART1_INT   7
#define PWM0_1_INT    8
#define I2C0_INT    9
#define SPI0_INT    10      /* SPI and SSP0 share VIC slot */
#define SSP0_INT    10
#define SSP1_INT    11
#define PLL_INT     12
#define RTC_INT     13
#define EINT0_INT   14
#define EINT1_INT   15
#define EINT2_INT   16
#define EINT3_INT   17
```

```
#define ADC0_INT    18
#define I2C1_INT    19
#define BOD_INT     20
#define EMAC_INT    21
#define USB_INT     22
#define CAN_INT     23
#define MCI_INT     24
#define GPDMA_INT   25
#define TIMER2_INT    26
#define TIMER3_INT    27
#define UART2_INT   28
#define UART3_INT   29
#define I2C2_INT    30
#define I2S_INT     31

#define VIC_SIZE    32

#define VIC_BASE_ADDR ((unsigned int)&VICIRQSTATUS)

#define VECT_ADDR_INDEX 0x100
#define VECT_CNTL_INDEX 0x200

void init_VIC( void );
long install_irq( long IntNumber, void *HandlerAddr, long Priority );

#endif /* end __IRQ_H */

/************************************************************************
                                                End Of  File
************************************************************************/
```

config.h

```
#if USE_USB   /* 1 is USB, 0 is non-USB related */
/* Fcck = 57.6Mhz, Fosc = 288Mhz, and USB 48Mhz */
#define PLL_MValue      11
#define PLL_NValue      0
#define CCLKDivValue    4
#define USBCLKDivValue    5

/* System configuration: Fosc, Fcclk, Fcco, Fpclk must be defined */
/* PLL input Crystal frequence range 4KHz~20MHz. */
#define Fosc  12000000
/* System frequence,should be less than 80MHz. */
#define Fcclk 57600000
#define Fcco  288000000
#else

/* Fcck = 50Mhz, Fosc = 300Mhz, and USB 48Mhz */
#define PLL_MValue      24
#define PLL_NValue      1
#define CCLKDivValue    5
#define USBCLKDivValue    6

/* System configuration: Fosc, Fcclk, Fcco, Fpclk must be defined */
/* PLL input Crystal frequence range 4KHz~20MHz. */
#define Fosc  12000000
/* System frequence,should be less than 80MHz. */
#define Fcclk 50000000
#define Fcco  300000000

#endif
```

```
/* APB clock frequence , must be 1/2/4 multiples of ( Fcclk/4 ). */
/* If USB is enabled, the minimum APB must be greater than 16Mhz */
#if USE_USB
#define Fpclk (Fcclk / 2)
#else
#define FPCLK (Fcclk / 4)
#endif

void init_VIC(void)
{
long i = 0;
long *vect_addr, *vect_cntl;

  /* initialize VIC*/
  VICINTENCLEAR = 0xffffffff;
  VICADDRESS = 0;
  VICINTSELECT = 0;

  /* set all the vector and vector control register to 0 */
  for ( i = 0; i < VIC_SIZE; i++ )
  {
    vect_addr = (long *)(VIC_BASE_ADDR + VECT_ADDR_INDEX + i*4);
    vect_cntl = (long *)(VIC_BASE_ADDR + VECT_CNTL_INDEX + i*4);
    *vect_addr = 0x0;
    *vect_cntl = 0xF;
  }
  return;
}
void GPIOResetInit( void )
{
  /* Reset all GPIO pins to default: primary function */
  PINSEL0 = 0x00000000;
  PINSEL1 = 0x00000000;
  PINSEL2 = 0x00000000;
  PINSEL3 = 0x00000000;
  PINSEL4 = 0x00000000;
  PINSEL5 = 0x00000000;
  PINSEL6 = 0x00000000;
  PINSEL7 = 0x00000000;
  PINSEL8 = 0x00000000;
  PINSEL9 = 0x00000000;
  PINSEL10 = 0x00000000;

  IO0DIR = 0x00000000;
  IO1DIR = 0x00000000;

  FIO0DIR = 0x00000000;
  FIO1DIR = 0x00000000;
  FIO2DIR = 0x00000000;
  FIO3DIR = 0x00000000;
  FIO4DIR = 0x00000000;

  FIO0MASK = 0x00000000;
  FIO1MASK = 0x00000000;
  FIO2MASK = 0x00000000;
  FIO3MASK = 0x00000000;
  FIO4MASK = 0x00000000;

  return;
}
void ConfigurePLL ( void )
{
long  MValue, NValue;
```

```
  if ( PLLSTAT & (1 << 25) )
  {
    PLLCON = 1;     /* Enable PLL, disconnected */
    PLLFEED = 0xaa;
    PLLFEED = 0x55;
  }

  PLLCON = 0;       /* Disable PLL, disconnected */
  PLLFEED = 0xaa;
  PLLFEED = 0x55;

  SCS |= 0x20;      /* Enable main OSC */
  while( !(SCS & 0x40) ); /* Wait until main OSC is usable */

  CLKSRCSEL = 0x1;    /* select main OSC, 12MHz, as the PLL clock source */

  PLLCFG = PLL_MValue | (PLL_NValue << 16);
  PLLFEED = 0xaa;
  PLLFEED = 0x55;

  PLLCON = 1;       /* Enable PLL, disconnected */
  PLLFEED = 0xaa;
  PLLFEED = 0x55;

  CCLKCFG = CCLKDivValue; /* Set clock divider */
  #if USE_USB
    USBCLKCFG = USBCLKDivValue;   /* usbclk = 288 MHz/6 = 48 MHz */
  #endif

  while ( ((PLLSTAT & (1 << 26)) == 0) ); /* Check lock bit status */

  MValue = PLLSTAT & 0x00007FFF;
  NValue = (PLLSTAT & 0x00FF0000) >> 16;
  while ((MValue != PLL_MValue) && ( NValue != PLL_NValue) );

  PLLCON = 3;       /* enable and connect */
  PLLFEED = 0xaa;
  PLLFEED = 0x55;
  while ( ((PLLSTAT & (1 << 25)) == 0) ); /* Check connect bit status */
  return;
}
void TargetResetInit(void)
{
  /* This if-else statement detects if interrupt vectors located by the linker
  command file are at memory location 0 or not. */
  #pragma segment = "INTVEC"
  if (( void * )0x00000000UL == __segment_begin( "INTVEC" ))
  {
    MEMMAP = 1;  // normal flash mode
  }
  else
  {
    MEMMAP = 2 ; // user ram mode - Map lowest 64 bytes of the address space to
                 // bottom of internal RAM, moving exception vectors into place
  }

#if USE_USB
  PCONP |= 0x80000000;    /* Turn On USB PCLK */
#endif
  /* Configure PLL, switch from IRC to Main OSC */
  ConfigurePLL();

  /* Set system timers for each component */
```

```
#if (Fpclk / (Fcclk / 4)) == 1
  PCLKSEL0 = 0x00000000;  /* PCLK is 1/4 CCLK */
  PCLKSEL1 = 0x00000000;
#endif
#if (Fpclk / (Fcclk / 4)) == 2
  PCLKSEL0 = 0xAAAAAAAA;  /* PCLK is 1/2 CCLK */
  PCLKSEL1 = 0xAAAAAAAA;
#endif
#if (Fpclk / (Fcclk / 4)) == 4
  PCLKSEL0 = 0x55555555;  /* PCLK is the same as CCLK */
  PCLKSEL1 = 0x55555555;
#endif

  /* Set memory accelerater module*/
  MAMCR = 0;
#if Fcclk < 20000000
  MAMTIM = 1;
#else
#if Fcclk < 40000000
  MAMTIM = 2;
#else
  MAMTIM = 3;
#endif
#endif
  MAMCR = 2;

  GPIOResetInit();

  init_VIC();

  return;
}
```

___

Guide for Contributors

Aims and Scope

Sensors & Transducers Journal (ISSN 1726-5479) provides an advanced forum for the science and technology of physical, chemical sensors and biosensors. It publishes state-of-the-art reviews, regular research and application specific papers, short notes, letters to Editor and sensors related books reviews as well as academic, practical and commercial information of interest to its readership. Because it is an open access, peer review international journal, papers rapidly published in *Sensors & Transducers Journal* will receive a very high publicity. The journal is published monthly as twelve issues per annual by International Frequency Association (IFSA). In additional, some special sponsored and conference issues published annually.

Topics Covered

Contributions are invited on all aspects of research, development and application of the science and technology of sensors, transducers and sensor instrumentations. Topics include, but are not restricted to:

- Physical, chemical and biosensors;
- Digital, frequency, period, duty-cycle, time interval, PWM, pulse number output sensors and transducers;
- Theory, principles, effects, design, standardization and modeling;
- Smart sensors and systems;
- Sensor instrumentation;
- Virtual instruments;
- Sensors interfaces, buses and networks;
- Signal processing;
- Frequency (period, duty-cycle)-to-digital converters, ADC;
- Technologies and materials;
- Nanosensors;
- Microsystems;
- Applications.

Submission of papers

Articles should be written in English. Authors are invited to submit by e-mail editor@sensorsportal.com 6-14 pages article (including abstract, illustrations (color or grayscale), photos and references) in both: MS Word (doc) and Acrobat (pdf) formats. Detailed preparation instructions, paper example and template of manuscript are available from the journal's webpage: http://www.sensorsportal.com/HTML/DIGEST/Submition.htm Authors must follow the instructions strictly when submitting their manuscripts.

Advertising Information

Advertising orders and enquires may be sent to sales@sensorsportal.com Please download also our media kit: http://www.sensorsportal.com/DOWNLOADS/Media_Kit_2008.pdf