Application Note

Reading Temperature Using the F082A Series MCUs

AN019102-0708

Abstract

This Application Note discusses a method of using the on-chip temperature sensor of Zilog's F082A Series microcontrollers. It describes how to read the temperature data and display it using the RS-232 serial cable and the on-chip Universal Asynchronous Receiver/Transmitter (UART). Using the on-chip timer, temperature data is stored at periodic intervals on a Non-Volatile Data Storage (NVDS) area. The data is retrieved from the NVDS and is displayed on a HyperTerminal at half-hour intervals.

The demo application allows setting of the upper and lower limits for the temperature. When the ambient temperature surpasses the set limits, LEDs are switched ON to indicate the limit break.

Note: *The source code file associated with this application note (`AN0191-SC01.zip`) is available for download at www.zilog.com.*

F082A Series Flash Microcontrollers

Zilog's Z8 Encore! XP® products are based on the new eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8® MCU.

Z8 Encore! XP MCUs combine a 20 MHz core with Flash memory, a linear-register SRAM, and an extensive array of on-chip peripherals.

The F082A Series of devices support up to 4 KB of Flash Program Memory and 1 KB register RAM. An on-chip temperature sensor allows temperature measurement over a range of –40 ºC to +105 ºC. These devices include two enhanced 16-bit timer blocks featuring PWM and Capture and Compare capabilities.

An on-chip Internal Precision Oscillator (5 MHz/32 kHz) can be used as a trimmable clock source requiring no external components. The F082A Series of devices include 128 Bytes of NVDS memory, where individual bytes can be written or read. The full-duplex UART, besides providing serial communications and IrDA encoding and decoding capability, also supports multidrop address processing in hardware.

The rich set of on-chip peripherals make the F082A Series MCUs suitable for various applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

This section provides a brief discussion on the following F082A Series MCUs on-chip peripherals, used to develop the temperature sensor application:

- UART
- Temperature Sensor
- Analog-to-Digital Converter (ADC)
- NVDS Memory
- Timer
- GPIO Ports

UART

The F082A Series UART0 is used to communicate with the PC's HyperTerminal. It is initialized to the required baud rate by writing the appropriate value to the UART baud rate registers.

Temperature Sensor

The on-chip temperature sensor uses a proportional-to-absolute temperature (PTAT) topology, with a provision for zero-point calibration. The temperature sensor is enabled or disabled by setting or resetting a corresponding bit in the Power Control Register 0. The on-chip Analog-to-Digital Converter (ADC) directly reads the temperature sensor to determine the absolute value of its output.

Analog-to-Digital Converter (ADC)

The F082A Series MCUs feature an on-chip Sigma-Delta ADC that converts an analog input signal to its digital representation. This ADC can be configured for DIFFERENTIAL mode or SINGLE-ENDED mode.

The ADC features 11-bit resolution in DIFFERENTIAL mode and 10-bit resolution in SINGLE-ENDED mode

The F082A Series ADC can take up to 8 single-ended analog input sources that are multiplexed with general-purpose I/O ports. The 9th analog input is directly obtained from the on-chip temperature sensor. The ADCs can be configured for either a single-shot conversion or for continuous conversion.

NVDS Memory

The F082A Series of devices contain a NVDS area with a size of up to 128 Bytes. This memory features an endurance of 100,000 write cycles. The NVDS is implemented by special-purpose Zilog® software routines stored in areas of Program Memory not accessible to the user. These routines use the Flash memory to store the data.

Timer

The F082A Series of devices contain up to two 16-bit reloadable timers that can be used for timing, event counting, or generation of pulse-width modulated (PWM) signals.

The timer features include:

- 16-bit reload counter.
- Programmable prescaler with prescale values from 1 to 128.
- PWM output generation.
- Capture and compare capability.
- External input pin for timer input, clock gating, or capture signal. External input pin signal frequency is limited to a maximum of one-fourth the system clock frequency.
- Timer output pin.
- Timer interrupt generation.

GPIO Ports

The F082A Series products support a maximum of 25 pins (Ports A-D) for the GPIO operations. A control and data register is associated with each port. The GPIO control registers are used to determine data direction, open-drain, output drive current, programmable pull-ups, STOP Mode Recovery functionality, and alternate pin functions. Each port pin is individually programmable. In addition, the Port C pins are capable of direct LED drive at programmable drive strengths.

Note: *For more details on the above on-chip peripherals, refer to the Z8 Encore! XP® F082A Series Product Specification (PS0228) available for download at www.zilog.com.*

Developing the Temperature Sensor Application

The temperature sensor application for the F082A Series MCUs is developed to provide the following functionalities:

- To select the on-chip temperature sensor and connect to the input of the ADC.
- To read the temperature every minute and display the temperature data on the HyperTerminal.
- To store the temperature data at 5-minute intervals on the NVDS memory area and display the stored data at half-hour intervals.

Software Implementation

The software implementation of the temperature sensor application is sectioned into the following tasks:

- Initializing the peripherals—the GPIO, UART, timer, and the ADC peripherals are initialized.
- Writing to the NVDS memory—the NVDS area where the temperature data is stored.
- Displaying the temperature data on the HyperTerminal—this involves reading from the NVDS area, converting the raw data to the actual temperature value and displaying the actual value on the HyperTerminal console.

Each task is described below.

Initializing the GPIO Pins

In the temperature sensor application, the GPIO pins are configured as output pins to switch ON/OFF the LEDs—D2, D3, and D4 on the F082A Series Development Platform. The LEDs are connected to the development platform as shown in Table 1.

Table 1. LED Connections

LED	Port Pin on Development Platform
D2	Port A pin 6
D3	Port A pin 7
D4	Port C pin 3

The function `void init_gpio(void)`, defined in the `gpio.c` file, initializes the GPIO ports in accordance with the above requirements. Port A pins, PA6 and PA7, as output to control the LEDs D2, D3 respectively, and Port C pin, PC3, to control LED D4 on the Development Platform.

When the ambient temperature exceeds the set upper limit (30 °C), LED D4 is switched ON. When the ambient temperature falls below the set lower limit (20 °C), LED D2 is switched ON. When the ambient temperature is within the range of 20 °C to 30 °C, LED D3 is switched ON, indicating normal temperature.

To change the temperature limit settings, redefine the `#define HIGH_SET_TEMP` and `#define LOW_SET_TEMP`, in the `main.h` file, with new values.

Initializing the UART Peripheral

The UART0 peripheral is used to transmit temperature data through the RS-232 port to the HyperTerminal—a serial communication application. Port A pins 4 and 5 are set for alternate functions for receiving data (Rx) and transmitting data (Tx), respectively.

The function `void init_uart0(void)`, defined in the `uart.c` file, initializes the UART0 for 38400 bps, 8-bit data, No parity, No flow control, and one stop bit.

To change the baud rate settings, redefine the `#define BAUD0` in `uart.h` file with the required baud rate values. For example, to change the baud rate to 9600 bps redefine as follows:

```
#define   BAUD0  9600
```

The actual values are calculated and written automatically to the Baud Rate HIGH and LOW registers.

Initializing the Timer

The timer, timer0, is initialized to generate an interrupt at one-second intervals. The function `void init_timer0(void)`, defined in the `timer.c` file, initializes timer0.
The `init_timer0()` routine loads the appropriate value to the T0H, T0L, TORH, and TORL registers. The `init_timer0()` routine also loads the prescale value. The timer-operating mode is selected in this routine.

To change the time-out period, redefine the following in the `timer.h` file:

```
#define PRESCALE,
#define TIMEOUT,
#define TIMER_START_VALUE,
#define SYSTEMCLOCK
```

The compiler calculates the timer reload value during compilation. To modify the timer operating mode, the TOCTL register must be loaded with appropriate values. These values are defined in the `init_timer0()` function.

The timer interrupt service routine, `void isr_timer0(void)`, defined in the `timer.c` file, keeps a track of the timing aspects. Two counters—a minutes counter and a seconds counter—are used. When the seconds counter reaches count 60 (indicating one minute), it sets the `print_every_minute` flag. This flag is checked in the `main()` routine; if the flag is set, temperature data is printed on the HyperTerminal console. The seconds counter is reset to zero and the minutes counter is incremented.

When the minutes counter reaches a value of 5 (indicating five minutes), the flag `store_data` is set. This flag is checked in the `main` routine, if the flag is set, the temperature data is stored in the NVDS memory area at the appropriate address. After 30 minutes, the data stored in the NVDS is read and printed on the HyperTerminal console. For the timer ISR flowchart, see Figure 5 on page 12.

To change the time interval for printing the temperature data on the HyperTerminal or storing it in the NVDS area, redefine the following in the `main.h` file:

```
#define SECONDS_COUNT,
#define MINUTES_COUNT,
#define HALF_HOUR_COUNT
```

Initializing the ADC and Enabling the Temperature Sensor

The ADC peripheral is initialized by writing appropriate values to the ADC Control registers 0 and 1.

For example,

Writing `ADC_TEMP_CHANNEL = 0x0e` to the ADC Control register 0 (ADCCTL0) accomplishes the following tasks:

- Temperature sensor output is selected as an input to ADC.
- The default internal reference voltage of 2.0 V is selected.
- The ADC is set for Single shot conversion.

Writing `ADC_BUF_INPUT_X1 = 0x01` to the ADC Control register 1 (ADCCTL1) sets the ADC to single-ended mode with buffered input and unity gain.

Writing to the Interrupt Request Enable registers, IRQ0E1 and IRQ0E0, sets the ADC interrupts as high priority interrupts.

The ADC conversion is enabled in the `main()` routine. When the conversion is complete, the ADC generates an interrupt. The ADC values are read and stored in a global variable in the ADC interrupt service routine, `void isr_ADC(void)`.

The `adc_data_available` flag is set in the ADC ISR indicating that new data is available. Analog to Digital conversion is enabled and the conversion of the input data (from the temperature sensor) starts again.

Writing to the NVDS Area

The `NVDS_Write()` and `NVDS_Read()` routines are used to access the NVDS area. These routines are accessed with a CALL instruction to a predefined address outside of the user-accessible Program Memory.

The NVDS address and data are single-byte values. As these routines disturb the working register set, the user code must ensure that any required working register values are preserved by pushing them onto the stack or by changing the working register pointer just prior to the NVDS execution.

Note: *Interrupts are not disabled during the read and write operations to the NVDS. Any interrupts that occur during the NVDS execution must take care not to disturb the working register and existing stack contents. Disabling interrupts before executing NVDS operations is recommended. Thus before calling the* `NVDS_Write` *or* `NVDS_Read` *routines, the interrupts are disabled and then enabled after the routines are executed.*

As the timer is set to generate interrupts at one-second intervals, the seconds counter increments every second. The minutes counter is incremented after the seconds counter reaches 60 counts.

The temperature data is refreshed on the HyperTerminal at one-minute intervals. After an interval of 5 minutes, the temperature data at that instant is stored in the NVDS area. After 30 minutes (when six readings are accumulated in the NVDS area) all of the data stored in the NVDS area is read and displayed on the HyperTerminal.

Raw data from the ADC Data registers is stored in the NVDS area between address locations `0x00` and `0x0B`.

The `NVDS_Write()` routine writes data to the NVDS area. The address of the location and the data to be written are selected carefully. Refer to the `main.c` file for the complete write sequence. The NVDS write routine is invoked by calling a special program in the `0x10B3` location, which is not exposed to the user.

```
char NVDS_Write(void)
{
            asm ("\tLDX R2,
_NVDS_Address");
            asm ("\tPUSH R2");
            asm ("\tLDX R2,
_NVDS_Data");
            asm ("\tPUSH R2");
            asm ("\tCALL %10B3");
            asm ("\tPOP R2");
            asm ("\tPOP R2");
}
```

To read data from the NVDS memory, the `NVDS_Read()` routine is used.

```
char NVDS_Read()
{
            asm ("\tLDX R2,
_NVDS_Address");
            asm ("\tPUSH R2");
            asm ("\tCALL %1000");
```

```
        asm ("\tPOP R2");
}
```

For example, if `high_byte = NVDS_Read();` is used to read data, the data is read from the required location, returned through the R0 register and assigned to the `high_byte` variable.

For successful NVDS operation, the Flash Frequency registers must be programmed based on the system clock frequency. The Flash Frequency registers are initialized by the `init_flash(void)` function.

The programming frequency is calculated by the following formula:

$$\textbf{FFREQ[15:0]} = \frac{\textbf{System Clock Frequency (Hz)}}{\textbf{1000}}$$

The System clock frequency in the Temperature sensor application is 8 MHz and the values to be loaded for the Flash frequency registers are indicated below.

```
void init_flash(void)
{
   FPFH = 0x1F;// Program Flash
   frequency registers to
   // 8MHz
   FPFL = 0x40;
}
```

Selecting the System Clock

The F082A Series of devices use five possible clocking schemes, each of which is user-selectable. The different clocking schemes are listed below:

- On-chip precision trimmed RC oscillator
- On-chip oscillator using off-chip crystal or resonator
- On-chip oscillator using external RC network
- External clock drive
- On-chip low precision Watchdog Timer oscillator

You must select any one of the above oscillators. The Oscillator Control register (OSCCTL) enables/disables the various oscillator circuits, enables/disables the failure detection/recovery circuitry and selects the primary oscillator, which becomes the system clock. The Oscillator Control register must be unlocked before writing. Writing the two-step sequence `0xE7` followed by `0x18` to the Oscillator Control register unlocks it. The register becomes locked on successful completion of a register write operation to the OSCCTL.

The function `void init_external_systemclock()` defined in the `main.c` file selects the type of the oscillator used with the Z8F04A2 XP MCU. The development kit uses 8 MHz off-chip ceramic resonator.

The sequence to write to the Oscillator Control register is provided below:

```
OSCCTL = 0xE7; // Unlock sequence for
                  OSCTL write
OSCCTL = 0x18;
OSCCTL = 0xE2;
```

The above sequence accomplishes the following tasks:

- Crystal oscillator is enabled.
- Internal precision oscillator is enabled.
- WDT oscillator is enabled.
- Crystal oscillator or the external RC oscillator functions as system clock.

Displaying the Temperature Readings

The ADC generates 10-bit data for a corresponding input voltage. The data is available in the ADC Data HIGH Byte register (ADCDH) and ADC Data LOW Byte register (ADCDL). The data in HIGH (excluding the sign bit) and LOW byte registers (bits 5, 6 and 7) are concatenated to form the actual data for a given input voltage. The Transfer function table for the temperature sensor is available in the *Z8 Encore! XP® F082A Series Product Specification (PS0228)*.

The Temperature sensor application program displays the temperature directly, in degrees Centigrade (°C), on the HyperTerminal console. The temperature measurement is accurate for temperatures between 20 °C to 30 °C.

To calculate the actual temperature, the following method is used:

1. The routine `unsigned int actual_temp()` calculates the actual temperature based on the ADC Data register values.
2. The contents of the ADC Data registers are manipulated and stored in an unsigned integer value and stored in the `valueH_new` variable.
3. `0x18B` is subtracted from the `valueH_new` and the result is divided by 5 to obtain the actual temperature in degree centigrade.
4. The `display_data(unsigned int measured_temp)` function displays the temperature data on the HyperTerminal. The floating-point values are discarded.

Demonstration

This section provides information on how to setup the demonstration and run the Temperature sensor application using the F082A Series Development Kit.

Setup

Figure 1 displays the basic setup for the Temperature sensor demonstration using the F082A Series Development Kit and a PC with a HyperTerminal application.

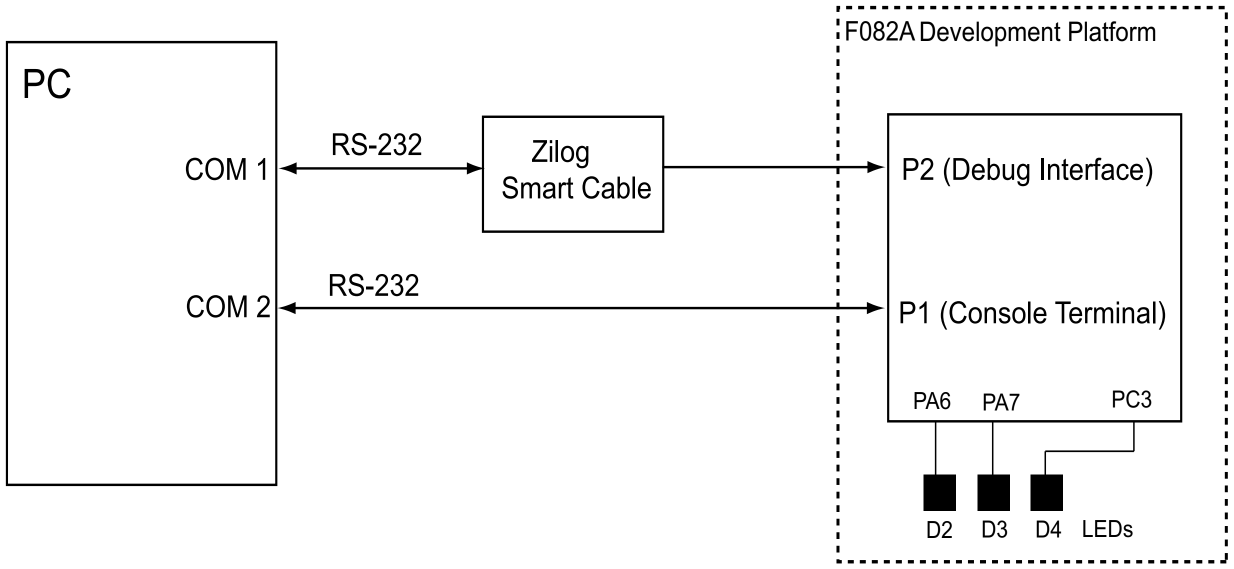

Figure 1. Basic Setup to Run the Temperature Sensor Demo

Equipment Used

The equipment used to run the Temperature sensor demo include:

- F082A Series Development Kit, which includes:
  - Development Platform with the Z8F042A module
  - Smart Cable for PC to F082A Series Development Board
  - Universal 5 V DC Power Supply
  - ZDS II-Z8 Encore!® IDE with ANSI C-Compiler
- PC with HyperTerminal application

Procedure

Follow the steps below to run the Temperature sensor demo:

1. Connect the F082A Series Development Platform to the PC as displayed in Figure 1.
2. Launch HyperTerminal application on the PC with the following settings:
   - 38400 bps
   - 8-N-1
   - No flow control
3. Launch ZDS II-Z8 Encore! v4.7.2. Compile and build the `xp_tempsensor.pro` project file. This project file is in the `AN0191-SC01.zip` file, available for download at www.zilog.com.
4. Download the `xp_tempsensor.lod` file to the F082A Series Development Platform using ZDS II.
5. Execute the code.
6. Observe that the current temperature is displayed on the HyperTerminal. When the ambient temperature exceeds the set limits, the LED D4 on the F082A Series Development Platform switches ON.

   When the temperature falls back to normal, LED D3 switches ON and D4 switches OFF. When the temperature falls below the normal range, LED D2 switches ON and the other LEDs are switched OFF.

Summary

This Application Note discusses a method of using the F082A Series on-chip temperature sensor to display ambient temperature readings. The presence of an on-chip temperature sensor in F082A Series microcontrollers eliminates the requirement of a separate temperature sensor. It also eliminates the efforts required to interface an external temperature sensor.

This Application Note also demonstrates the use of NVDS area, where required data can be stored on-the-fly to be retrieved when required. Presence of 128 Bytes of NVDS area on the F082A Series MCUs eliminates interfacing to an external EEPROM to write/read a byte of data, when additional storage memory is required.

References

Further details about the F082A Series of devices can be found in the references listed below:

- Z8 Encore! XP® F082A Series Product Specification (PS0228)
- eZ8™ CPU User Manual (UM0128)
- Zilog Developer Studio II-Z8 Encore!® User Manual (UM0130)

Appendix A—Flowcharts

This appendix contains the flowcharts for the temperature sensor application described in this Application Note.

Figure 2 through Figure 4 display the flowchart for the main routine to initialize the on-chip peripherals and to control the LEDs.

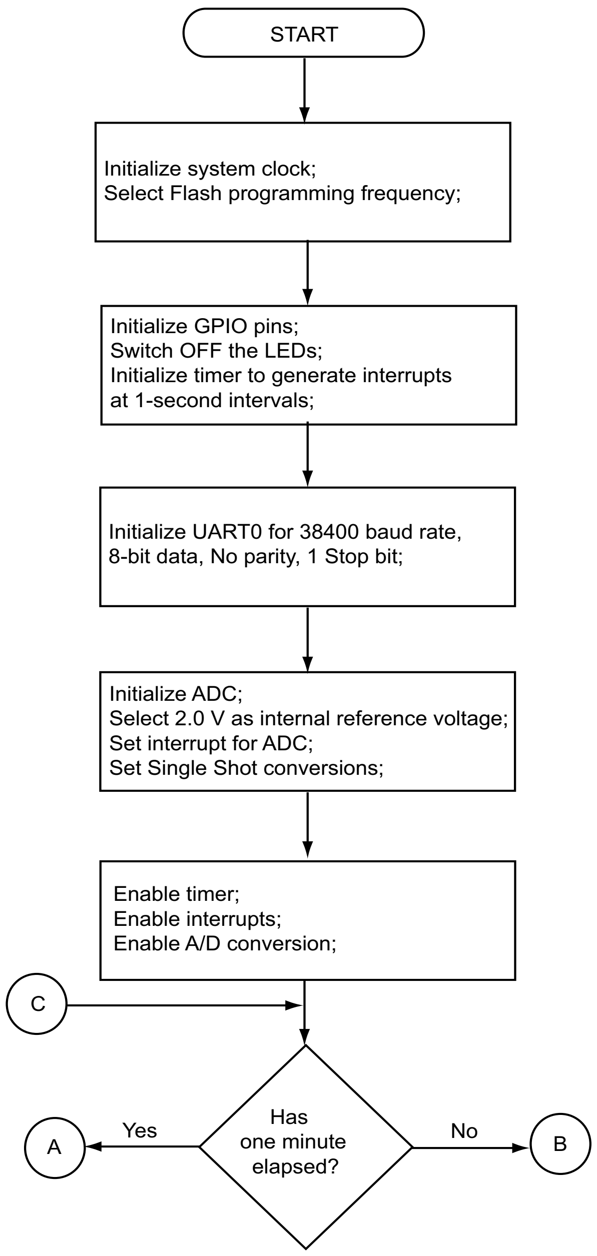
START
Initialize system clock;
Select Flash programming frequency;
Initialize GPIO pins;
Switch OFF the LEDs;
Initialize timer to generate interrupts at 1-second intervals;
Initialize UART0 for 38400 baud rate, 8-bit data, No parity, 1 Stop bit;
Initialize ADC;
Select 2.0 V as internal reference voltage;
Set interrupt for ADC;
Set Single Shot conversions;
Enable timer;
Enable interrupts;
Enable A/D conversion;
C
Has one minute elapsed?
Yes
A
No
B

Figure 2. Main Routine to Initialize Peripherals and Control LEDS

Figure 3 is a continuation of the main routine flowchart.

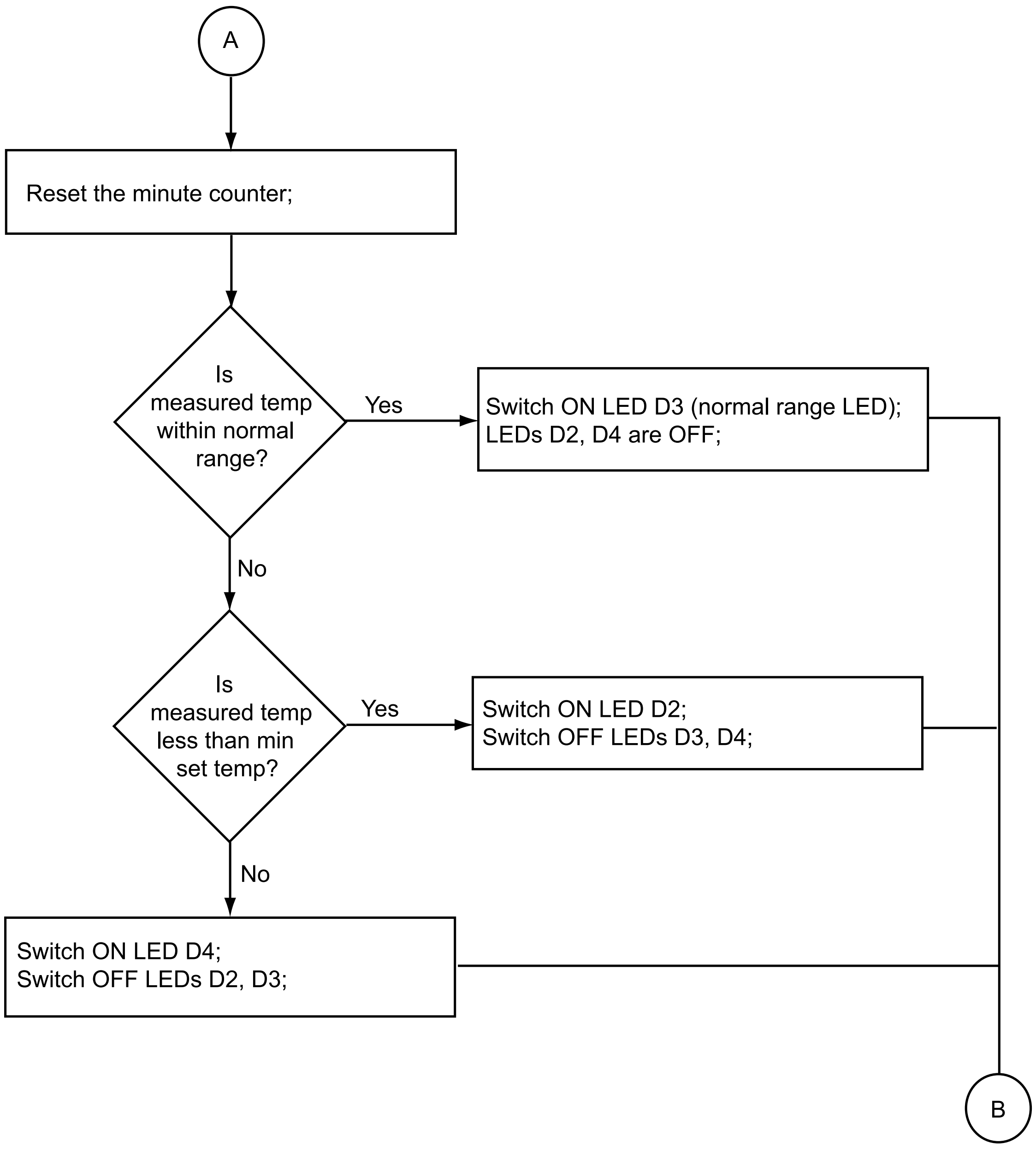
A
Reset the minute counter;
Is measured temp within normal range?
Yes
Switch ON LED D3 (normal range LED); LEDs D2, D4 are OFF;
No
Is measured temp less than min set temp?
Yes
Switch ON LED D2; Switch OFF LEDs D3, D4;
No
Switch ON LED D4; Switch OFF LEDs D2, D3;
B

Figure 3. Main Routine (Continued-1)

Figure 4 is a continuation of the main routine flowchart.

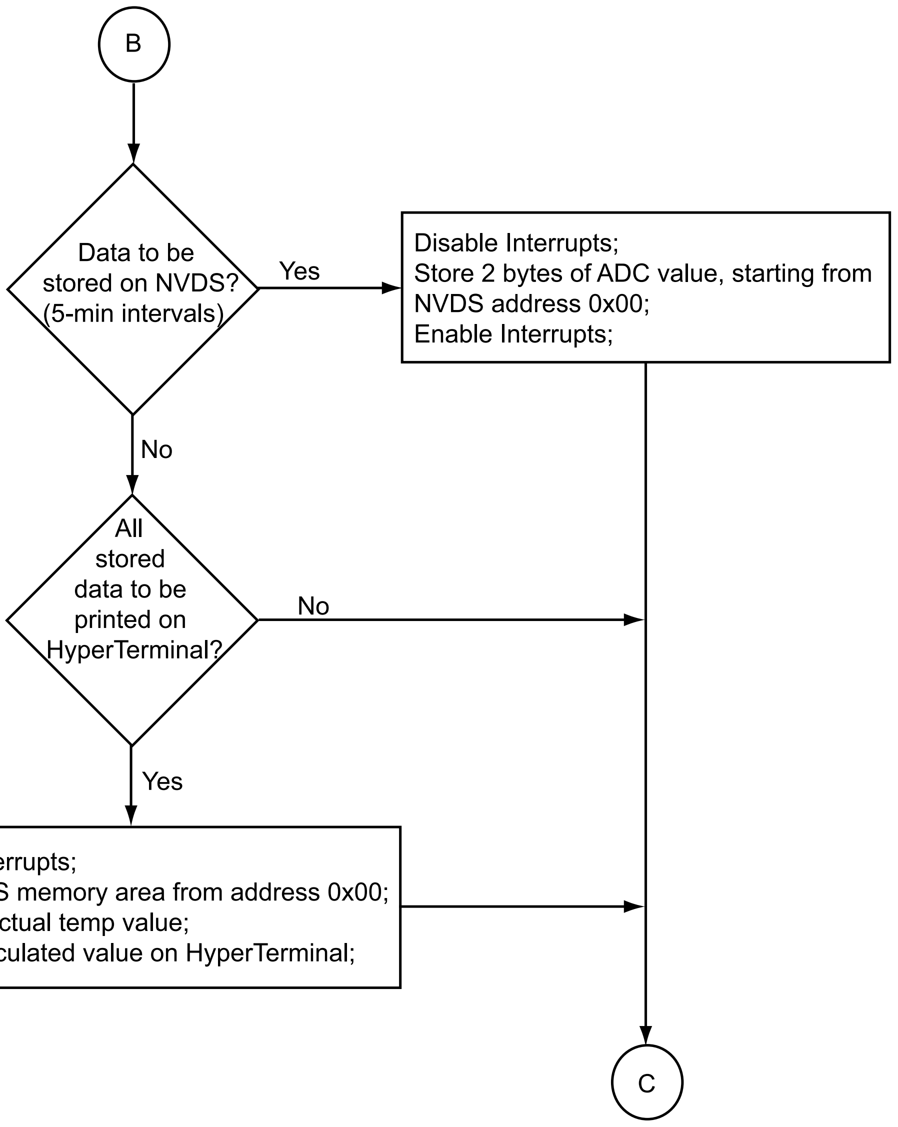
B
Data to be stored on NVDS? (5-min intervals)
Yes
Disable Interrupts;
Store 2 bytes of ADC value, starting from NVDS address 0x00;
Enable Interrupts;
No
All stored data to be printed on HyperTerminal?
No
Yes
Disable Interrupts;
Read NVDS memory area from address 0x00;
Calculate actual temp value;
Display calculated value on HyperTerminal;
C

Figure 4. Main Routine (Continued-2)

Figure 5 displays the flowchart for the timer Interrupt Service Routine.

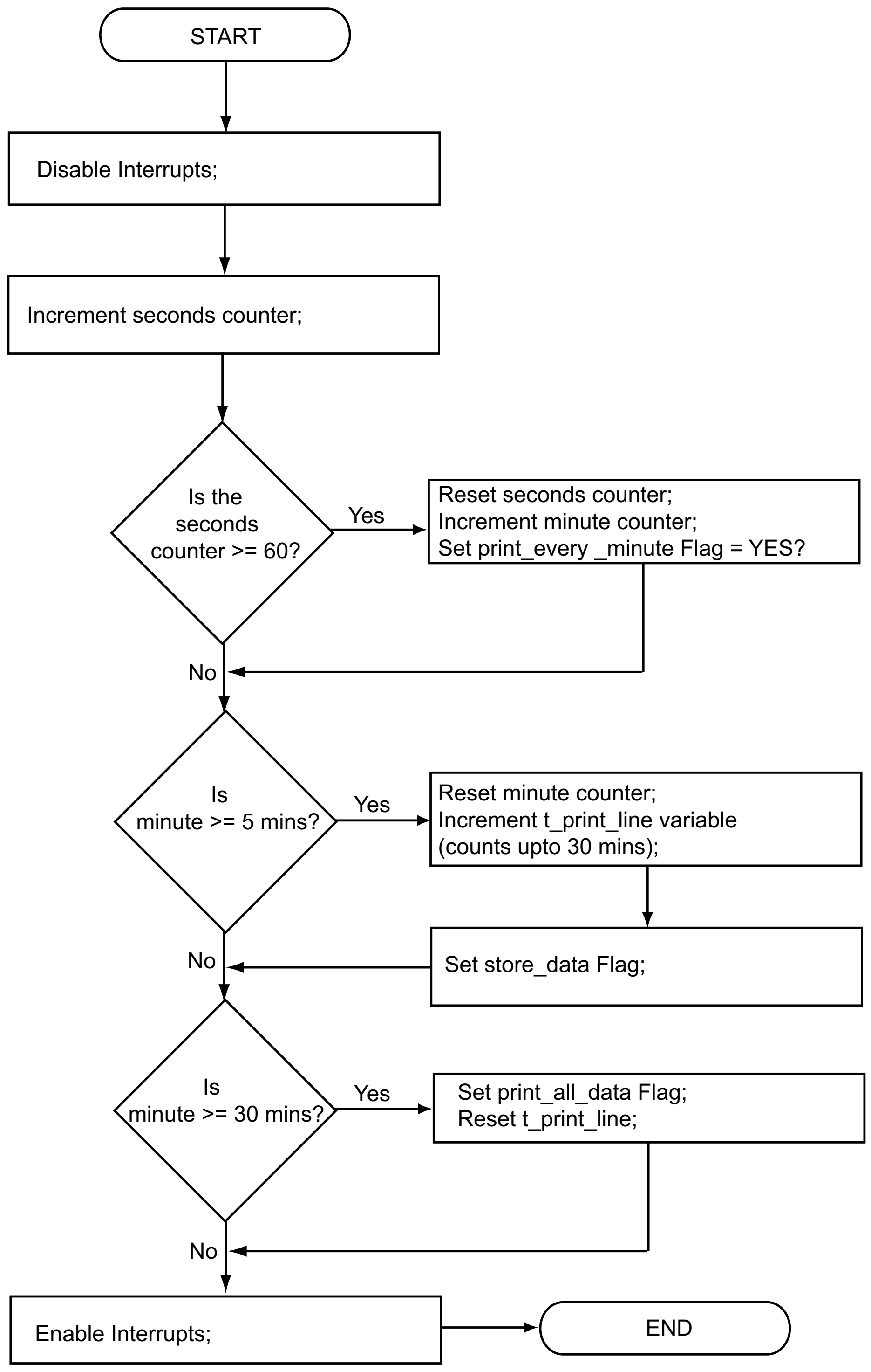

Figure 5. Timer Interrupt Service Routine

Figure 6 displays the flowchart for the ADC Interrupt Service Routine.

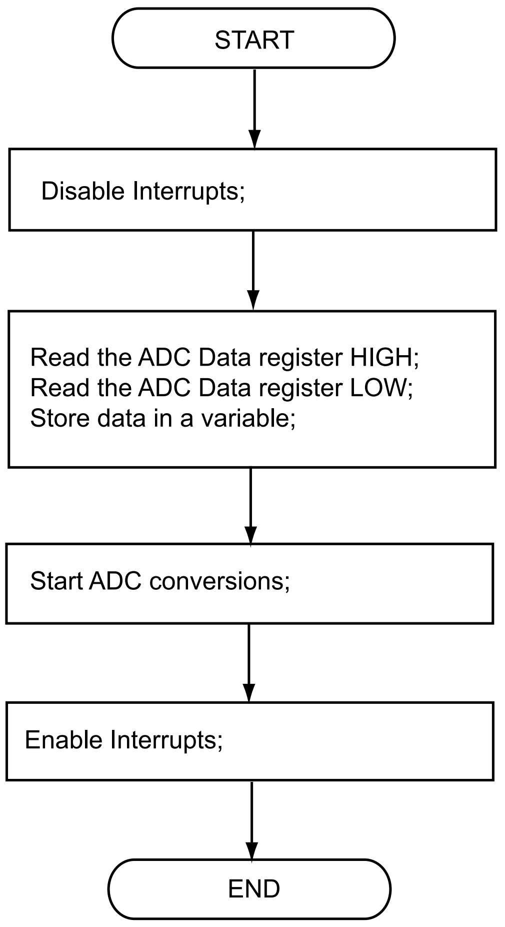
START
Disable Interrupts;
Read the ADC Data register HIGH;
Read the ADC Data register LOW;
Store data in a variable;
Start ADC conversions;
Enable Interrupts;
END

Figure 6. ADC Interrupt Service Routine

Warning: DO NOT USE IN LIFE SUPPORT