



## APPLICATION NOTE 4350

# Reading Temperature Using the MAXQ2010 Evaluation Kit

*Abstract: This application note demonstrates the ease of application development for MAXQ® microcontrollers with the availability of IAR Embedded Workbench® integrated-development environment and MAXQ Evaluation (EV) Kit. The easiest way to demonstrate the simple development process is with an example application. This application uses the UART, timer, and ADC on the MAXQ2010 RISC microcontroller to monitor and report temperature.*

## Application Overview

The example application described here uses the [MAXQ2010](#) processor and [MAXQ2010 Evaluation \(EV\) Kit](#). This example showcases the uses of one of the microcontroller's UARTs, the integrated ADC controller, and one of the device's timers. The timer is used to generate interrupts every 500ms. The MAXQ2010 port interfaces with the ADC for data acquisition. Temperature readings are then taken by connecting a thermistor to the MAXQ2010's ADC. When an interrupt occurs, the MAXQ2010 takes a temperature reading and outputs the results to one of its serial ports.

## Reading ADC Data

In the example application, the ADC controller reads an analog input that changes with temperature. Before any input is read, the application must set the ADC's clock, conversion mode (single or continuous), reference voltage source (internal, external, or  $AV_{DD}$ ), and input channel for source of interest. A few registers need to be initialized to enable ADC conversion.

```
void initADC ( )
{
    ADCN = 0x00; // Single conversion, external AVDD reference, divide-by-1 clock frequency.
    ADST = 0x10; //Select configuration register - read/write access on ADDATA.
    ADDATA = 0x00; //Channel 0 for AN0.
    ADDATA = 0x01; //Channel 0 for AN1.
    ADDATA = 0x02; //Channel 0 for AN2.
    ADDATA = 0x03; //Channel 0 for AN3.
    ADDATA = 0x04; //Channel 0 for AN4.
    ADDATA = 0x05; //Channel 0 for AN5.
    ADDATA = 0x06; //Channel 0 for AN6.
    ADDATA = 0x07; //Channel 0 for AN7, thermistor is attached on this channel.
    ADADDR = 0x07; //Selecting 0x07 as the last conversion configuration register.
}
```

After initialization, setting the ADCONV bit of the ADST register initiates a conversion. The hardware indicates that the conversion is complete by clearing that same bit. To read the results, set the ADIDX[3:0] bits of the ADST register to the input channel of interest. The ADDATA register will then hold the digital data of the corresponding channel. In this sample application, AN7 is assigned to ADC data buffer 7. Therefore to read thermistor data, set ADST = 0x07 and read ADDATA.

```
unsigned int getADCReading()
{
    unsigned short data = 0;

    ADST_bit.ADCONV = 1; //Enable conversion.
    while( ADST_bit.ADCONV == 0x01 ); //ADST.ADCONV bit indicates conversion in progress.
```

```

ADST = 0x07; //We are interested in reading only the AN7 input, that is thermistor data.
data = ADDATA; //Read AN7 data.
return data;
}

```

The following steps are performed on the digital value read to arrive at the temperature:

1. Calculate the analogInput at AN7 in volts corresponding to the digital value read.
2. Calculate the thermistor resistance corresponding to the analogInput at AN7.
3. Calculate the temperature in degree Celsius.

## Writing to a Serial Port

In the example application, one of the MAXQ2010's serial ports is used to output the current temperature reading. Before any data can be written to the port, the application must set the baud rate and the serial-port mode. Just a few registers need to be initialized to enable serial-port communications.

```

void initSerial()
{
    SCON0_bit.SM1 = 1; // Set to Mode 1.
    SCON0_bit.REN = 1; // Enable receives.
    SMD0_bit.SMOD = 1; // Set baud rate to 16 times the baud clock.
    PR0 = 0x75F7; // Set phase for 115200 with an 8MHz crystal.
    SCON0_bit.TI = 0; // Clear the transmit flag.
    SBUF0 = 0x0D; // Send carriage return to start communication.
}

```

In the UART, a single register sends and receives serial data. Writing to the SBUF0 register initiates a transfer. When data becomes available on the serial port, reading the SBUF0 register retrieves the input. The example program uses the following function to output data to the serial port.

```

int putchar(int ch)
{
    while(SCON0_bit.TI == 0); // Wait until we can send.
    SCON0_bit.TI = 0; // Clear the sent flag.
    SBUF0 = ch; // Send the char.
    return ch;
}

```

## Generating Periodic Interrupts with a Timer

The last component used in this example application is one of the 16-bit timers. The timer generates interrupts that trigger temperature readings twice a second. To configure the timer for this example, the programmer must set the reload value, specify the clock source, and start the timer. The following code shows the steps required for initializing timer B.

```

TBOV = 0x00000; // Set current timer value.
TBOR = 0x0F42; // Set reload value.
TBOCN = 0x0506; // Set Timer Clock = SysClk/1024, Reload Timer Mode, Interrupt Enabled.

```

Using this timer as an interrupt source for this example requires a few more steps. Interrupts for the MAXQ architecture must be enabled on three levels: globally, for each module, and locally. Using the IAR™ compiler, enable global interrupts by calling the `__enable_interrupt()` function. This function effectively sets the Interrupt Global Enable (IGE) bit of the Interrupt and Control (IC) register. Since timer B0 is located in module 4, set bit 4 of the Interrupt Mask Register (IMR) to enable interrupts for the module. Enable the local interrupt by setting the Enable Timer B Interrupts (ETB) bit in Timer B Control Register (TBOCN). These steps are shown below.

```
__enable_interrupt()  
TBOCN = 0x506; // Enable interrupts along with setting Timer Clock and run Timer.  
IMR |= 0x10; // Enable the interrupts for module 4.
```

Finally, using an interrupt requires initializing the interrupt vector. IAR's compiler allows a different interrupt handling function for each module. Setting the interrupt handler for a particular module requires using the `#pragma vector` directive. The interrupt-handling function declaration should also be preceded by the `__interrupt` keyword. The example application declares an interrupt handler for module 4 in the following way.

```
#pragma vector = 4  
__interrupt void timerInterrupt()  
{  
    // Add interrupt handler here.  
}
```

## MAXQ2010 EV Kit and IAR IDE Settings

The following setup is required to run the example application.

1. MAXQ2010 EV kit should have:
  - a. JU14 and JU22 shorted to enable thermistor reading on channel AN7.
  - b. JU9 pin 2 and 3 shorted to enable UART0.
  - c. JU8 open to let 8MHz crystal source the controller.
  
2. IAR IDE project options should have:
  - a. `lnkmaxq2010.xcl` as the Linker command file (select Options from the Project menu, then select Linker from the Category list, then select the Config tab).
  - b. `maxq2010.ddf` as the Device description file (select Options from the Project menu, then select Debugger from the Category list, then select the Setup tab).
  - c. JTAG as the Driver (select Options from the Project menu, then select Debugger from the Category list, then select the Setup tab).
  - d. COMx port setting to communicate with JTAG (select Options from the Project menu, then select JTAG from the Category list).
  
3. PC HyperTerminal setting should have:
  - a. PC COMx port, to which the UART0 of MAXQ2010 is connected.
  - b. Configuration properties as 115200 baud rate, 8 data bits, 1-STOP bit, and no parity and flow control.

This sample application was tested with the [MAXQ IAR Embedded Workbench V2.12A](#). **Figure 1** shows the expected output.



[for EE-Mail™](#).

---

### **Related Parts**

MAXQ2010: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN4350, AN 4350, APP4350, Appnote4350, Appnote 4350

Copyright © by Maxim Integrated Products

Additional legal notices: [www.maxim-ic.com/legal](http://www.maxim-ic.com/legal)