

MAXQ2000評価キットを使った サンプルアプリケーション

標準ANSI Cツールや、そのツールを統合する開発環境があると、新しくて不慣れなプロセッサを使ったアプリケーション開発が大幅に楽になります。MAXQプロセッサで利用することができるツールとしては、IARのANSI CコンパイラとIAR Embedded Workbench統合開発環境があります。このプログラムに加え、MAXQ専用レジスタについての基本的な知識があれば、MAXQアーキテクチャのアプリケーションを短期間で簡単に作成を開始することが可能です。MAXQアーキテクチャを使った開発がいかにシンプルであるかは、サンプルアプリケーションの作成を見ればお分かりいただけるはずです。

標準ANSI Cツールや、そのツールを統合する開発環境があると、新しくて不慣れなプロセッサを使ったアプリケーション開発が大幅に楽になります。

ここでは、例として、MAXQ2000プロセッサとMAXQ2000評価キットを使ったアプリケーションを取り上げます。MAXQ2000は、以下のようにさまざまな周辺機器を内蔵しています。

- 132セグメントのLCDコントローラ
- マスタ/スレーブモードを持つ統合SPIポート
- 1-Wireバスマスタ
- 2本のシリアルUART
- ハードウェア乗算器
- 3個の16ビットタイマ/カウンタ
- ウォッチドッグタイマ
- 32ビットのリアルタイムクロック(サブセカンドアラームと時刻アラーム付き)
- インサーキットデバッグ機能をサポートするJTAGインタフェース

アプリケーションの概要

この例では、LCDコントローラ、マスタモードのSPIポート、UART 1本、ハードウェア乗算器、タイマ1個を使用します。タイマにより、定期的な割込を発生させます。割込が発生すると、MAXQ2000は温度を読み取り、結果をLCDとシリアルポートに出力します。SPIポートは、ADCを持つMAX1407データ収集システム(DAS)とのインタフェースとして使用します。温度は、MAX1407のADCに接続したサーミスタによって読み取られます。

LCDコントローラの使い方

LCDを使うためには、2つのコントロールレジスタの構成が必要です。レジスタの設定が完了し、どちらかのLCDデータレジスタのあるビットをセットすると、LCDセグメントがオンになります。以下のコードは、LCDコントローラを構成する例です。

```
void initLCD()
{
    LCRA_bit.FRM = 7;    // Set up frame frequency.
    LCRA_bit.LCCS = 1;   // Set clock source to HFC1k / 128.
    LCRA_bit.DUTY = 0;   // Set up static duty cycle.
    LCRA_bit.LRA = 0;    // Set R-adj to 0.
    LCRA_bit.LRIGC = 1;  // Select external LCD drive power.

    LCFG_bit.PCF = 0x0F; // Set up all segments as outputs.
    LCFG_bit.OPM = 1;    // Set to normal operation mode.
    LCFG_bit.DPE = 1;    // Enable display.
}
```

SPI経由の通信

SPIレジスタにデータを書き込むとSPIマスタとスレーブの間で双方向通信が開始されます。

MAXQ2000にはさまざまなSPIモードがあり、3つのレジスタによってモードをコントロールします。MAX1407との通信では、以下のコードによってSPIコンポーネントを初期化し、適切なモードにすることができます。

```
PD5 |= 0x070;           // Set CS, SCLK, and DOUT pins as output.
PD5 &= ~0x080;          // Set DIN pin as input.
SPICK = 0x10;           // Configure SPI for rising edge, sample input
SPICF = 0x00;           // on inactive edge, 8 bit, divide by 16.
SPICN_bit.MSTM = 1;     // Set Q2000 as the master.
SPICN_bit.SPIEN = 1;    // Enable SPI.
```

SPI構成レジスタの設定が完了したら、SPIレジスタを使ってデータの送受信を行います。SPIレジスタに書き込むとSPIマスタとスレーブの間で双方向通信が開始されます。データ伝送の完了は、SPICNレジスタのSTBYビットで確認します。SPIを使った送受信コードの例を示します。

```
unsigned int sendSPI(unsigned int spib)
{
    SPIB = spib;          // Load the data to send
    while(SPICN_bit.STBY); // Loop until the data has been sent.
    SPICN_bit.SPIC = 0;   // Clear the SPI transfer complete flag.
    return SPIB;
}
```

シリアルポートへの書き込み

サンプルアプリケーションでは、MAXQ2000のシリアルポートの一方から現在の温度を出力します。シリアルポートにデータを書き込むためには、まず、アプリケーションからボーレートとシリアルポートモードの設定を行う必要があります。この場合も、ほんのいくつかのレジスタを初期化するだけで、シリアルポート通信を行うことができます。

```
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 = 0x3AFB;           // Set phase for 115200 with a 16MHz crystal.
    SCON0_bit.TI = 0;       // Clear the transmit flag.
    SBUF0 = 0x0D;           // Send carriage return to start communication.
}
```

SPI通信ルーチンと同じように、1つのレジスタでシリアルデータの送受信を行うことができます。SBUF0レジスタに書き込むと、伝送が開始されます。データがシリアルポートに届いたとき、SBUF0レジスタを読み出すと入力値を取得することができます。次の関数は、サンプルアプリケーションでシリアルポートにデータを出力するプログラム例で使用されます。

```
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;
}
```

MAXQアーキテクチャでは、グローバルレベルと各モジュールレベル、ローカルレベルの3つのレベルで割込をイネーブルにする必要があります。

タイマによる定期的な割込の生成

サンプルアプリケーションで紹介する最後のコンポーネントは、16ビットタイマです。このタイマによって、1秒に2回の温度の読み取りをトリガする割込を発生させます。このようにタイマを構成するためには、リロード値を設定し、クロックソースを指定して、タイマを起動します。タイマ0の初期化に必要なコードの例を、以下に示します。

```
T2V0 = 0x00000;          // Set current timer value.
T2R0 = 0x00BDC;          // Set reload value.
T2CFG0_bit.T2DIV = 7;    // Set div 128 mode.
T2CNA0_bit.TR2 = 1;      // Start the timer.
```

このタイマを割込ソースとして使うためには、あと数ステップが必要です。MAXQアーキテクチャでは、グローバルレベルと各モジュールレベル、ローカルレベルの3つのレベルで割込をイネーブルにする必要があります。IARコンパイラでは、__enable_interrupt() 関数を呼び出すとグローバル割込がイネーブルになります。これによって、IC(Interrupt and Control)レジスタのIGE(Interrupt Global Enable)ビットがセットされます。タイマ0はモジュール3にあるため、IMR(Interrupt Mask Register)のビット3をセットすると、モジュールレベルの割込がイネーブルになります。ローカル割込のイネーブルは、T2CNA(Timer/Counter 2 Control Register A)のET2(Enable Timer Interrupts)ビットをセットします。このステップのコード例を、以下に示します。

```
__enable_interrupt()
T2CNA0_bit.ET2 = 1;      // Enable interrupts.
IMR |= 0x08;            // Enable the interrupts for module 3.
```

もう1点、割込を使用するためには、割込ベクトルの初期化が必要です。IARコンパイラでは、モジュールごとに異なる割込処理関数を使用することができます。モジュールの割込ハンドラを設定するためには、#pragmaベクトル指示を使用します。また、割込処理関数の宣言で、関数名の前に__interruptキーワードをつける必要があります。以下の例は、モジュール3の割込ハンドラを宣言するものです。

```
#pragma vector = 3
__interrupt void timerInterrupt()
{
    // Add interrupt handler here.
}
```

**周辺機器レジスタについて
若干のことを学ぶだけで、
MAXQプロセッサのアプリ
ケーションを開発すること
ができます。**

まとめ

コード例からわかるように、周辺機器レジスタについて若干のことを学ぶだけで、MAXQ2000プロセッサをはじめとする各種MAXQプロセッサのアプリケーションを開発することができます。IARのEmbedded Workbenchを使うと、ANSI準拠のC言語でコードを書くことが可能になり、開発をさらに加速することができます。

今回紹介したサンプルアプリケーションのソースコードは、すべて、japan.maxim-ic.com/MAXQ_codeからダウンロードすることができます。その他に必要な配線やセットアップについては、各コードの冒頭に記入してある説明やコメントを参照してください。IARのEmbedded Workbenchについての詳細は、2番目の記事、「MAXQ環境におけるプログラミング」を参照してください。