AND9213/D

無電源ワイヤレス・センサの読み取り

ON Semiconductor®

www.onsemi.jp

APPLICATION NOTE

概要

Magnus®−S ICを実装した、オン・セミコンダクターの無電源ワイヤレス・センサは、RF入力で観測したインピーダンスを表すセンサ・コード、センサ・タグがリーダから受信している電力を表すオンチップRSSIコード、ダイの正確な温度読取値に変換可能な温度コードのうち最大3つの独自情報部分をリーダに渡します。本書ではこれらのコードを読み取る手順について説明します。

センサ・タグ・モデル番号の決定

センサ・コード、オンチップRSSIコード、温度コードは、センサ・タグ・メモリ内で3つの異なるワード・アドレスに格納されます。このアドレスは使用中のMagnus−S ICのタグ・モデル番号によって決まります。タグ・モデル番号が事前にわからない場合は、標準Class−1 Generation−2 UHF Readコマンドを使用して、TIDメモリ・バンク(バンク2h)内のワード1h (16進数)を読み取って決定できます。

タグ・モデル番号は4桁の16進数になります(センサ・タグ・メモリ内では、どのワードも2バイト長です)。最上位3桁がセンサ・データのアドレスを決定します。例えば、$402E_h$という値が取得された場合、センサ・データは下表で左側の列が402_hになっている行で見つかります。

センサ・コードの読み取り

センサ・コードは、標準Class−1 Generation−2 UHF Readコマンドを使用してタグから読み取ることができます。前述のように、メモリ位置はタグ・モデル番号によって異なりTable 1に示されています。

Table 1. LOCATION OF SENSOR CODE VALUE

Tag Model Number Starts with	Memory Bank	Word Address
401_h	USER (Bank 3_h)	B_h
402_h	RESERVED (Bank 0_h)	B_h
403_h	RESERVED (Bank 0_h)	C_h

センサ・コードは、5ビット値または9ビット値であり、タグ・モデル番号によって異なります。ワード内の残りのビットは0です(Table 2)。

Table 2. NUMBER OF BITS USED IN SENSOR CODE

Tag Model Number Starts with	Number of Bits Used in Sensor Code
401_h, 402_h	5
403_h	9

オンチップRSSIコードの読み取り

オンチップRSSIコードは、5ビット値(10進数で0〜31の範囲)であり、センサ・タグが受信した電力量を示します。値が大きくなるほど、受信した電力レベルが大きいことを示します。オンチップRSSIコードは2ステップのプロセスで読み取ることができます。

1. 標準Class−1 Generation−2 UHFのSelectコマンドを送信し、オンチップRSSIコードが規定スレッショルドより大きい、小さい、または等しいすべてのセンサ・タグに通知します。
2. 標準Class−1 Generation−2 UHFのReadコマンドを送信し、電力スレッショルド基準を満たす特定のセンサ・タグに対応するオンチップRSSIコードを取得します。

このプロセスを通じて、受信している電力量に応じてセンサ・タグをフィルタして、電力レベルが、インベントリ・ラウンドのリマインダに対する特定スレッショルドより大きいまたは小さいタグを応答させないようにすることができます。受信した電力量に関係なくセンサ・タグが応答するように設定す

ることも可能です。この方法については、以下の例で説明します。

センサ・タグがTable 3に記載するパラメータが指定されたSelectコマンドを受信すると、オンチップRSSIコードが生成されます。このコード値がスレッショルドより大きいか小さいかに関係なく、コード値はセンサ・タグの揮発性メモリに格納され、リーダ信号がオフになるまで維持されます。ただし、以降のReadコマンドに応答するには、Maskパラメータ(Table 4)がオンチップRSSIコード値に一致している必要があります。

Table 3. ON-CHIP RSSI SELECT COMMAND PARAMETERS

Tag Model Number Starts with	Memory Bank	Pointer Bit Address	Mask Length	Mask (Table 4)
401_h, 402_h	USER (Bank 3_h)	$A0_h$	8_h	M[7:0]
403_h	USER (Bank 3_h)	$D0_h$	8_h	M[7:0]

Table 4. BIT MASK FOR ON-CHIP RSSI SELECT COMMAND

Mask Bit	M7	M6	M5	M4	M3	M2	M1	M0
Bit Value	0	0	0: Match if Code is ≤ Threshold 1: Match if Code is > Threshold	5-bit Threshold Most Significant Bit First				

例えば、センサ・タグの受信電力量に関係なく、マスクがセンサ・タグに確実に一致するようにするには、マスクのビット5を0に設定し、スレッショルドを最大値11111に設定すると、00011111 ($1F_h$)という8ビット・マスクが得られます。センサ・タグがSelectを満たす場合は、Table 5に示すアドレスを指定したReadコマンドを使用してオンチップRSSIを取得できます。ワードの最下位5ビットがコードで、ワード内の他のビットは0になります。

Table 5. LOCATION OF ON-CHIP RSSI CODE VALUE

Tag Model Number Starts with	Memory Bank	Word Address
401_h	USER (Bank 3_h)	9_h
402_h, 403_h	RESERVED (Bank 0_h)	D_h

温度コードの読み取り

Magnus−S3 ICは高精度温度センサと組み合わせて使用できます。センサは温度コードを生成します。温度コードは摂氏温度値に変換できます。この機能が利用できるのは、タグ・モデル番号が403_hで始まるMagnus−S ICのみです。

オンチップRSSIコードの場合と同様、温度コードの読み取りは標準UHF SelectコマンドおよびReadコマンドを必要とする2ステップ・プロセスです。

1. Table 6に示すパラメータを指定して、標準Class−1 Generation−2 UHF Selectコマンドを送信し、温度センサを初期化するとともに温度コードを計算します。
2. 標準Class−1 Generation−2 UHF Readコマンドを送信して、Table 7に示す位置にあるセンサ・タグ・メモリから温度コードを取得します。

Table 6. TEMPERATURE CODE SELECT COMMAND PARAMETERS

Tag Model Number Starts with	Memory Bank	Pointer Bit Address	Mask Length	Mask
403_h	USER (Bank 3_h)	$E0_h$	0_h	Empty

センサ・タグがSelectコマンドを受信した後、次のTable 7に示すメモリ位置で温度コードが読み取れるようになります。温度コードはワードの最下位12ビットを占有しています。他のビットは0です。

Table 7. LOCATION OF TEMPERATURE CODE VALUE

Tag Model Number Starts with	Memory Bank	Word Address
403_h	RESERVED (Bank 0_h)	E_h

高精度の温度コードを取得するには、Selectコマンドの後に2 msの連続波を送信してから、リーダがそれ以降のコマンドを送信します。この休止により温度センサ回路が動作する時間を確保できます。どんな時にもSelectコマンドとReadコマンドの間でリーダの電源を切らないでください。

校正済み温度の測定

温度コードは、各センサ・タグ独自の校正データに従ってスケーリングすると、高精度の温度測定値に変換できます。温度測定対応のMagnus−S3 ICは、ユーザ・メモリ・バンクの最後の4ワード(8_h、9_h、A_h、B_hの各ワード)に校正データが事前にロードされています。これら4ワードつまり64ビットは、2点の直線校正を記述する6個のデータ・フィールドに編成されています。Table 8にこれらのフィールドを要約し、メモリ内での位置をTable 9にマップの形で示します。ここでは、いくつかのフィールドがワード境界にまたがっていることがわかります。

Table 8. ORGANIZATION OF TEMPERATURE CALIBRATION DATA

Field Name	Starting Bit Number (MSB)	Number of Bits	Description
CRC	80_h	16	CRC−16 Applied to the Remaining 48 Calibration Data Bits
CODE1	90_h	12	Temperature Code Measured at the First Calibration Temperature
TEMP1	$9C_h$	11	(First Calibration Temperature ni Decimal Degrees C) × 10 + 800
CODE2	$A7_h$	12	Temperature Code at the Second Calibration Temperature
TEMP2	$B3_h$	11	(Second Calibration Temperature ni Decimal Degrees C) × 10 + 800
VER	BE_h	2	Calibration Format Version Number (Set to 0_h)

Table 9. LOCATION OF TEMPERATURE CALIBRATION DATA IN USER BANK

Word Address	Label	Bit Description															
8_h	Bit Address (Hex)	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	Filed Description	CRC[15:0]															
9_h	Bit Address (Hex)	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
	Filed Description	CODE1[11:0]												TEMP1[10:7]			
A_h	Bit Address (Hex)	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	Filed Description	TEMP1[6:0]							CODE2[11:3]								
B_h	Bit Address (Hex)	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
	Filed Description	CODE2[2:0]			TEMP2[10:0]											VER[1:0]	

CODE1およびTEMP1フィールドは、最初のキャリブレーション時点で測定された温度コードと実際の温度を表しています。CODE2およびTEMP2フィールドは2番目の点における温度コードと温度を表しています。これらの2点は温度センサの直線応答を規定し、測定された他のすべての温度コードにおける校正済み温度を計算する目的で使用されます。

VERフィールドは2ビットのバージョン・コードを格納するもので0hに設定されています。CRCフィールドには校正データ内にある他の48ビットを対象として計算された16ビットCRCがあり、EPC ™ Generation−2 UHF RFID Specification, Annex F.2で定義されるCRC−16と同じ仕様に従います。

TEMP1およびTEMP2フィールドは長さ11ビット符号なし整数です。10進数で表記したTEMP1またはTEMP2フィールドに対して、次式を適用するとこれらの値を温度(摂氏単位)に変換できます。

$$\text{Calibration Temperature in}^{\circ}\text{C} = \frac{\text{TEMPx} - 800}{10} \quad \text{(eq. 1)}$$

ある温度コードCを校正済み温度(摂氏単位)に変換するには次式を適用します。ここで、値はすべて10進数です。

$$\text{Temperature in}^{\circ}\text{C} = \frac{1}{10} \cdot \left[\left(\frac{\text{TEMP2} - \text{TEMP1}}{\text{CODE2} - \text{CODE1}} \cdot (\text{C} - \text{CODE1}) \right) + \text{TEMP1} - 800 \right] \quad \text{(eq. 2)}$$

温度測定例

ユーザ・メモリ・バンク内の8_h、9_h、A_h、B_hの各ワードがそれぞれ$BD9F_h$、$88A7_h$、$E147_h$、および7900_hで読み取られたとします。オプションですが、CRCワードが実際のデータと一致していることを確認するのは良い考えです。$88A7E1477900_h$をCRC−16アルゴリズムに与えると、生成されるCRCの結果は$BD9F_h$になります。これは校正データに格納されているCRCワードに一致するため、校正情報は有効です。CRCワードを生成するためのサンプルC#コードを、本書の「サンプル・コード」セクションに示します。

残りの3ワードを分解すると、CODE1 = $88A_h$、TEMP1 = $3F0_h$、CODE2 = $A3B_h$、TEMP2 = 640_h、VER = 0_h (Figure 1)になります。10進数では、これらの値はCODE1 = 2186、TEMP1 = 1008、CODE2 = 2619、TEMP2 = 1600、VER = 0になります。

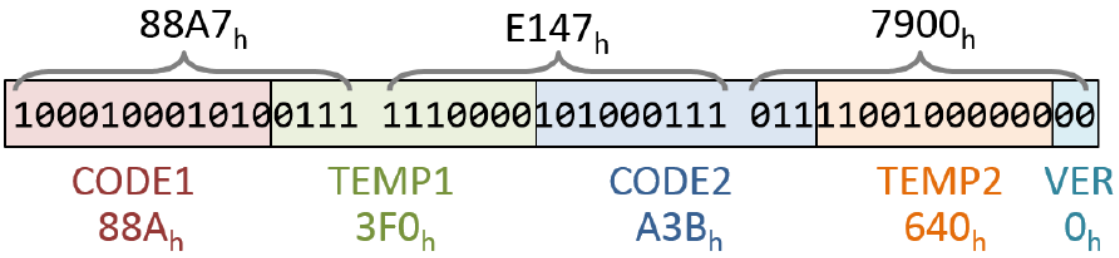

Figure 1. Unpacking Calibration Words into Fields

温度測定を行ったところ、温度コードが10進数の2315と通知されたと仮定しましょう。この値を、他の該当する校正フィールド値とともに上式に代入すると、次のように温度が得られます。

$$\text{Temperature in } ^\circ\text{C} = \frac{1}{10} \cdot \left[\left(\frac{1600 - 1008}{2619 - 2186} \cdot (2315 - 2186) \right) + 1008 - 800 \right] = 38.44^\circ\text{C} \quad \text{(eq. 3)}$$

10進数のTEMP1値から800を減算して10で除算することにより、最初の校正が実施された時点の温度を求めることもできます。

$$\frac{1008 - 800}{10} = 20.8^\circ\text{C} \quad \text{(eq. 4)}$$

サンプル・コード

次のサンプルC#コードは、ThingMagic RFIDリーダとThingMagic Mercury APIを使用してセンサ・コードとオンチップRSSIコードを読み取る方法を示します。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ThingMagic;

namespace ThingMagicSample
{
    class Program
    {
        static void Main(string[] args)
        {
            // Connect to the reader and adjust its settings ==========================================================
            Reader r = Reader.Create("tmr:///COM5");                    // Create Reader object
            r.Connect();                                                // Connect to the reader on emulated serial port COM5
            r.ParamSet("/reader/region/id", Reader.Region.NA);          // Set the regulatory region to North America
            r.ParamSet("/reader/radio/readPower", 2000);                // Set the reader power to 20 dBm

            // Read the Sensor Code ===================================================================================
            // Define a tag read operation which reads from a Magnus-S2;
            // Sensor Code is in the RESERVED bank, in memory location hex 8
            TagOp sensorCodeRead = new Gen2.ReadData(Gen2.Bank.RESERVED, 0x8, 1);
            // Define a read plan for antenna 1 using Gen2 protocol which uses our Sensor Code read operation
            SimpleReadPlan readPlan = new SimpleReadPlan(new int[] { 1 }, TagProtocol.GEN2, null, sensorCodeRead, true, 100);
            r.ParamSet("/reader/read/plan", readPlan);                  // Load the read plan we've defined into the reader
            TagReadData[] sensorReadResults = r.Read(75);               // Read for 75 ms, then stop
            foreach (TagReadData result in sensorReadResults)           // Loop through all tags found and print the EPC,
            {                                                           // frequency, and Sensor Code for each
                string EPC = ByteFormat.ToHex(result.Epc, "", "");
                string frequency = result.Frequency.ToString();
                string sensorCode = ByteFormat.ToHex(result.Data, "", "");
                Console.WriteLine("EPC: " + EPC + " Frequency (kHz): " + frequency + " Sensor Code:  " + sensorCode);
            }

            // Read the On-Chip RSSI Code =============================================================================
            // Define a Select command which applies to a Magnus-S2;
            // We want all tags to respond, regardless of their On-Chip RSSI Code value,
            // so our select mask should be a hex value of 1F
            byte[] mask = { Convert.ToByte("1F", 16) };
            // Select USER memory bank and pointer value bit address of hex A0.
            Gen2.Select select = new Gen2.Select(false, Gen2.Bank.USER, 0xA0, 8, mask);
            // The On-Chip RSSI Code is stored in the RESERVED bank, word location hex D
            TagOp onChipRSSIRead = new Gen2.ReadData(Gen2.Bank.RESERVED, 0xD, 1);
            // Define a read plan which uses our Select command and On-Chip RSSI Code read operation
            readPlan = new SimpleReadPlan(new int[] { 1 }, TagProtocol.GEN2, select, onChipRSSIRead, true, 100);
            r.ParamSet("/reader/read/plan", readPlan);                  // Load the read plan we've defined into the reader
            TagReadData[] onChipRSSIReadResults = r.Read(75);           // Read for 75 ms, then stop
            foreach (TagReadData result in onChipRSSIReadResults)       // Loop through all tags found and print the EPC,
            {                                                           // frequency, and On-Chip RSSI Code for each
                string EPC = ByteFormat.ToHex(result.Epc, "", "");
                string frequency = result.Frequency.ToString();
                string onChipRSSICode = ByteFormat.ToHex(result.Data, "", "");
                Console.WriteLine("EPC: " + EPC + " Frequency (kHz): " + frequency + " On-Chip RSSI: " + onChipRSSICode);
            }
        }
        // Example program output:
        // EPC: 002C000000000000000001234 Frequency (kHz): 923250 Sensor Code:  000A
        // EPC: 002C000000000000000001234 Frequency (kHz): 913250 On-Chip RSSI: 0014
    }
}
```

次のサンプルC#コードは、Impinj SpeedwayリーダとOctane APIを使用してセンサ・コードを読み取る方法を示します。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Impinj.OctaneSdk;

namespace OctaneSample
{
    class Program
    {
        static ImpinjReader reader = new ImpinjReader();            // Create a new Reader object

        static void Main(string[] args)
        {
            reader.Connect("172.16.1.17");                          // Connect to the reader
            reader.TagOpComplete += OnTagOpComplete;                // Assign an event handler which runs when a tag is read
            reader.DeleteAllOpSequences();                          // Reset the reader
            Settings settings = reader.QueryDefaultSettings();      // Get the default settings
            settings.Antennas.GetAntenna(1).TxPowerInDbm = 20;      // Set antenna 1 to an output power to 20 dBm
            TagReadOp readSensorCodeOp = new TagReadOp();           // Define a read operation for a Magnus-S2 chip
            readSensorCodeOp.MemoryBank = MemoryBank.Reserved;      // Sensor Code is in the Reserved Bank...
            readSensorCodeOp.WordPointer = 0x5;                     // ... at word address hex 5
            readSensorCodeOp.WordCount = 1;
            settings.Report.OptimizedReadOps.Add(readSensorCodeOp); // Load the read operation to the settings object
            settings.Report.IncludeChannel = true;                  // Request channel frequency information
            settings.Report.Mode = ReportMode.BatchAfterStop;
            reader.ApplySettings(settings);                         // Load the settings into the reader
            reader.Start();                                         // Read for 2 seconds, then stop
            System.Threading.Thread.Sleep(2000);
            reader.Stop();
            reader.Disconnect();
        }

        // This is the event handler which prints the results to the console window when the reader reads a tag.
        static void OnTagOpComplete(ImpinjReader reader, TagOpReport report)
        {
            foreach (TagOpResult result in report)
            {
                if (result is TagReadOpResult)
                {
                    TagReadOpResult readResult = result as TagReadOpResult;
                    string EPC = readResult.Tag.Epc.ToString();
                    string frequency = readResult.Tag.ChannelInMhz.ToString();
                    string sensorCode = readResult.Data.ToString();
                    Console.WriteLine("EPC: " + EPC + " Frequency (MHz): " + frequency + " Sensor Code: " + sensorCode);
                }
            }
        }
    }
}
// Example program output:
// EPC: 002C 0000 0000 0000 0000 1234 Frequency (MHz): 924.25 Sensor Code: 0006
// EPC: 002C 0000 0000 0000 0000 1234 Frequency (MHz): 917.25 Sensor Code: 0007
// EPC: 002C 0000 0000 0000 0000 1234 Frequency (MHz): 909.25 Sensor Code: 0000
```

次のサンプルC#コードは、EPC Generation−2 UHF RFID Specificationに従って、CRC−16ワードを計算する方法を示します。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace CRC_16_Example
{
    class Program
    {
        static string CRC16(string dataHexString)
        {
            int numBytes = dataHexString.Length / 2;
            byte[] dataByteArray = new byte[numBytes];    // Convert hex data string into an array of bytes,
            for (int b = 0; b < numBytes; b++)            // with byte 0 corresponding to the least-significant
            {                                             // two hex digits in the string
                dataByteArray[numBytes - 1 - b] = Convert.ToByte(dataHexString.Substring(2 * b, 2), 16);
            }
            BitArray data = new BitArray(dataByteArray);  // Convert the byte array to a BitArray type
            BitArray CRC = new BitArray(16);              // Create the CRC register
            CRC.SetAll(true);                             // Initialize all bits in the CRC register to 1
            for (int j = data.Length - 1; j >= 0; j--)    // This loop simulates the CRC logic described in
            {                                             // Annex F.2 of the EPC Generation-2 UHF RFID Spec
                bool newBit = CRC[15] ^ data[j];          // Data bits are fed into the register MSB first
                for (int i = 15; i >= 1; i--)
                {
                    if (i == 12 || i == 5)
                    {
                        CRC[i] = CRC[i - 1] ^ newBit;
                    }
                    else
                    {
                        CRC[i] = CRC[i - 1];
                    }
                }
                CRC[0] = newBit;
            }
            CRC.Not();
            byte[] CRCbytes = new byte[2];
            CRC.CopyTo(CRCbytes, 0);                      // Convert the CRC BitArray back to a byte array
            string CRCword = Convert.ToString(CRCbytes[1], 16) + Convert.ToString(CRCbytes[0], 16);
            return CRCword;
        }

        static void Main(string[] args)
        {
            string data = "88A7E1477900";
            string CRC = CRC16(data);
            Console.WriteLine(CRC);
        }
    }
}
// This example demonstrates calculating the CRC-16 word when the calibration words in User memory locations
// 0x9, 0xA, and 0xB are 0x88A7, 0xE147, and 0x7900.  The result is 0x8D9F
```


PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81−3−5817−1050

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative