Keywords: DS5250, 8051, magnetic, stripe, magstripe, POS, point of sale, ATM, card

APPLICATION NOTE 4095

Interfacing the DS5250 with a Magnetic Stripe Card Reader

Sep 12, 2007

Abstract: This application note explains how magnetic stripe card reading and decoding can easily be implemented on the DS5250 secure microcontroller. The article also demonstrates how applications can be developed that combine magnetic stripe card reading functions with the higher level security and cryptographic functions of a secure microcontroller. This example uses the DS5250 evaluation (EV) kit to interface to a magnetic stripe card reader. Source code is provided, and can easily be ported to any of Maxim's other 8051-based microcontrollers.

Overview

Cards with a magnetic-encoded stripe running along the back side are typically referred to as magnetic stripe cards, and are used in a vast number of banking, stored-value, and other financial applications. Credit, ATM, and debit cards are all typically magnetic stripe cards, sometimes with a smart-card chip embedded as well. Many gift cards, security or room-key cards, and stored-value cards used for gaming, photocopier use, and public transit are other examples. With a maximum capacity of approximately 160 bytes when standard ISO formats are used, these magnetic stripe cards do not typically store a large amount of data. Compared to smart cards or portable flash memory drives, the storage capacity of magnetic stripe cards is sufficient for the financial and banking applications for which the cards are most commonly used. Magnetic stripe cards are, moreover, rugged, inexpensive, require no internal power source, and relatively straightforward to read.

This application note describes how to use the DS5250 secure microcontroller to interface to a magstripe card reader. The DS5250 was chosen because it provides the critical security and cryptographic features needed to construct a common device that reads magnetic stripe cards, i.e., a point-of-sale (POS) terminal.

The magnetic stripe card reader contains the circuitry to perform a low-level task: decoding the magnetic fluxes on the magnetic stripe into a stream of bits. The DS5250 manages the higher level tasks: translating these bits into characters, then reading and verifying data fields.

Hardware and Software Requirements

Hardware

This example application uses the DS5250 evaluation (EV) kit (rev. B), with the following modification:

- The pullup resistor pack (RN2) for Port 0 pins was removed.

The example application also requires the following hardware:

- Card reader—MagTek® Triple Track card swipe reader with 3V decoder ASIC; part number 21030001 rev. G or newer (www.magtek.com)
- Two pullup resistors (approximately 10kΩ)
- Linear regulator (capable of producing 3V to 3.6V from a 5V supply)
- Magnetic stripe cards (ATM cards, credit cards, etc.) for testing

For more information on these magnetic stripe cards, please use these links:

- MagTek for documentation about the Delta Triple Track 3V card reader ASIC. Go to: www.magtek.com/documentation/public/99875258-9.02.pdf
- MagTek for documentation on magnetic stripe card track locations and formats. Go to: www.magtek.com/documentation/public/99800004-1.03.pdf
- MagTek for a character conversion table for magnetic stripe card data. Go to: www.magtek.com/documentation/public/99875065-4.02.pdf

Software

The example application was written in C and compiled using Keil's µVision® IDE version 2.40a. (www.keil.com/). Maxim's Microcontroller Tool Kit (MTK) was used to load the compiled application into the DS5250.

Application Details

The sections below describe the implementation of the sample application. Full example C code is available for download from Maxim's FTP site.

Connecting the Card Reader

The MagTek card reader has five interface pins (including power and ground):

- Pin 1—STROBE
  This signal is always an input to the card reader. It is driven by the master microcontroller to clock out card data bits and to assert a reset sequence.
- Pin 2—DATA
  This signal is generally used as an output by the card reader. It carries the card data bits from Tracks A/B/C, which are clocked out by the STROBE signal. Under certain circumstances (e.g., when asserting a reset sequence), this signal can be driven by the master microcontroller.
- Pin 3—VDD
  Powers the card reader (2.7V to 3.6V).
- Pin 4—GND
- Pin 5—GND

Because the STROBE and DATA pins operate at 3V levels, the DS5250 cannot drive them directly with standard port pins. Consequently, pins from Port 0 (which operate in open-drain mode) are used instead, and resistors pull the signal levels up to 3.6V when the pin is tristated by the DS5250. The DS5250 reads the DATA signal directly from the DATA line, because the minimum V_{IH} level for the DS5250's port pins is low enough to be compatible with 3V I/O levels. (Refer to the DS5250 data sheet for more details.)

The card reader operates at 3V levels; it cannot run directly from the 5V supply used by the DS5250. Since the DS5250 EV kit does not provide a 3V supply, a separate linear regulator (such as the

MAX1658) is used to power the card reader. See **Figure 1**.

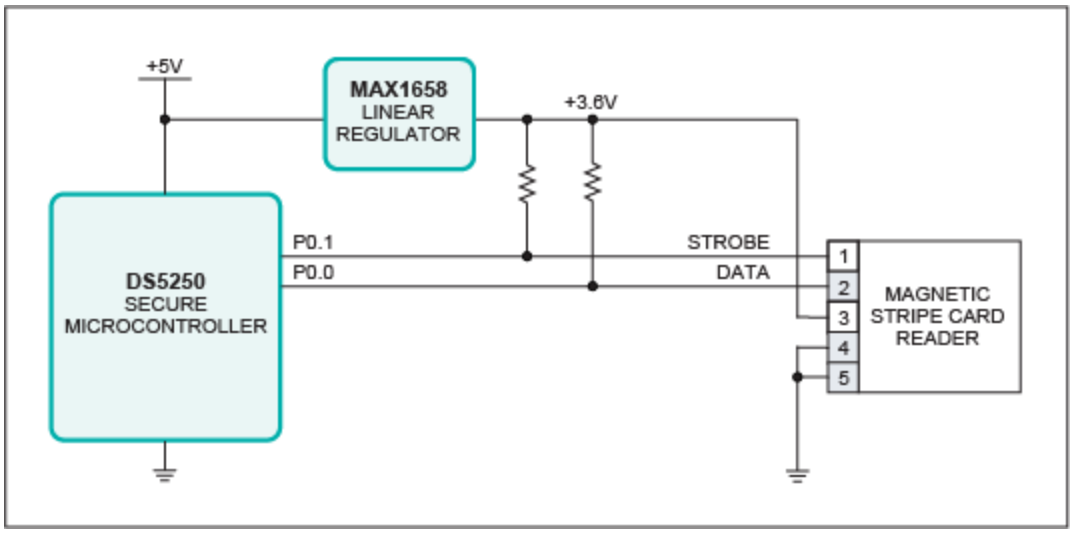

Figure 1. Connections for example application.

Performing the Reset Sequence

After first powering up, the master microcontroller must reset the card reader before it can accept a card swipe. This reset sequence must also be performed following each card swipe, both to clear the internal card-reader ASIC's memory and to prepare the card reader to accept a new card swipe.

To reset the card reader, the master microcontroller (i.e., the DS5250 in this application) must drive STROBE and DATA in the following sequence. (Refer to the MagTek documentation for exact voltage levels and timing parameters.)

1. Both DATA and STROBE begin in the high (idle) state.
2. Force DATA low.
3. With DATA held low, drive STROBE low and then high again.
4. Drive STROBE low again and then release DATA, allowing it to float high.
5. With DATA floating high, drive STROBE low and then high again. At this point, the card reader is reset and in a low-power wait state.
6. Drive STROBE low and then high again. This action "arms" the card reader and prepares it to accept a card swipe.

```
// Generate a long delay for card reset and read intervals.

void longDelay()
{
   int i, j;

   for (i = 1; i < 5; i++) {
      for (j = 1; j < 5000; j++) {
         ;
      }
   }
}

// Generate a shorter delay (used between STROBE/DATA transitions).

void delay()
```

```
{
   int i;

   for (i = 1; i < 1000; i++) {
      ;
   }
}

// Release the DATA line (P0.0) and allow it to float high.

void dataHigh()
{
   P0 |= 0x01;
   delay();
}

// Drive the DATA line (P0.0) low.

void dataLow()
{
   P0 &= 0xFE;
   delay();
}



// Release the STROBE line (P0.1) and allow it to float high.

void strobeHigh()
{
   P0 |= 0x02;
   delay();
}

// Drive the STROBE line (P0.1) low.

void strobeLow()
{
   P0 &= 0xFD;
   delay();
}

void resetCardReader()
{
   dataHigh();
   strobeHigh();
   longDelay();

   dataLow();                // Force DATA low.
   longDelay();
   strobeLow();              // Drive STROBE low, then high again.
   strobeHigh();
   strobeLow();              // Drive STROBE low, then release DATA.
   dataHigh();
   longDelay();

   strobeHigh();             // Drive STROBE low and high again two more
times
   strobeLow();              //    to complete the reset and leave the card
reader
   strobeHigh();             //    in the ready state, prepared to scan a
card.
   strobeLow();
}
```

Detecting a Card Swipe

Once the card reader has been reset and armed, it is ready to accept a card swipe. The MagTek card reader performs an entire read cycle of the card as it is swiped; no intervention is required from the master microcontroller. The entire contents of all three tracks, Track A, Track B, and Track C, on the card stripe are stored on the card-reader IC. This data can then be clocked out by the master microcontroller one bit at a time after the card-read cycle has completed.

The card reader goes through a handshaking cycle to inform the master when the card read begins and is complete.

1. The cycle begins with STROBE low and DATA floating high (idle state).
2. When the card reader detects a magnetic stripe moving across the reader head, it begins scanning the card. It drives the DATA line low to inform the master that a card swipe has begun.
3. The master responds by driving STROBE high and then low again.
4. The card reader drives DATA high again.
5. Once the card-read cycle is complete, the card reader drives DATA low again. This action tells the master that the card swipe is complete and that the card data is ready to be clocked out.

```
// Wait for the DATA line to be driven low by the card reader.

void waitForDataLow()
{
   int i = 0xFF;

   dataHigh();               // Make sure that DATA is floating high.

   while ((i & 1) == 1) {
      i = P0;
   }
}

....

   resetCardReader();
   printf("\r\n");
   printf("Waiting for card swipe...\r\n");
   printf("\r\n");

   waitForDataLow();      // DATA low indicates that card swipe has begun.
   strobeHigh();
   longDelay();
   strobeLow();
   longDelay();
   waitForDataLow();      // DATA low indicates that card swipe is complete.
```

Reading and Decoding Card Data

After the card swipe is complete and all card data decoded and stored by the card-reader ASIC, the card reader drives the DATA line low. As noted above, this action alerts the DS5250 that the card data is ready to be clocked out. At this point, the DS5250 can retrieve each bit in the card data sequentially by driving STROBE high and then low once for each bit to be clocked out. After STROBE is driven high and then low, the next bit is clocked out on the DATA line by the card-reader ASIC. The ASIC drives DATA high for a 0 bit and drives DATA low for a 1 bit.

```
// Clock a single bit value out of the card reader by driving STROBE high,
// then low, and reading the DATA line.

int readBit()
```

```
{
   int i;

   strobeHigh();              // Drive STROBE high.
   strobeLow();               // Drive STROBE low (DATA line now contains
bit).

   i = P0;
   if ((i & 1) == 0) {
      return 1;               // Low on DATA line indicates a 1 bit.
   } else {
      return 0;               // High on DATA line indicates a 0 bit.
   }
}
```

The first 16 bits clocked out of the card reader are a "preamble" which indicates the version of the card-reader ASIC. These bits can be discarded by the application, as they do not represent card data.

Translating Track A

After the 16-bit preamble, the next 704 bits clocked out on the DATA line contain the data read from Track A on the magnetic stripe card. When encoded using the standard ISO formats, Track A contains up to 76 characters, using a 7-bit character set which includes alphanumerics and an assortment of other symbols.

Each 7-bit character is clocked out least significant bit (LSB) first. The highest order bit (i.e., the seventh bit) is a parity bit which can optionally be used to verify the integrity of the card data. Disregarding the parity bit, the remaining six bits define one of 64 characters that can be encoded on Track A. For example, 000000b represents a space character and 000001b represents an exclamation point. The `char7bit[64]` character array is shown in the code below.

```
//          0123456789012345678901234567890123456789012345678901234567890 123
char char7bit[64] =
          " !'#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_";


// Clock out and decode a 7-bit character from the track memory, returning
the
// character value.  7-bit (alphanumeric) characters are found on Track A
only.

char read7BitChar()
{
   int i, c;

   // Each character is composed of 7 bits, which we clock out of the track
memory
   // beginning with the least significant bit.  Bit 7 is parity, which is
ignored.

   c = 0;
   for (i = 1; i < 128; i *= 2) {
      c |= (readBit() * i);
   }
   c &= 0x3F;

   return char7bit[c];       // Decode/return the character using the 7-bit
table.
}

....
```

```
   // Track A - 76 characters, 7 bits per alphanumberic character including
parity.

   printf("Track A > ");

   for (i = 0; i < 76; i++) {
      putchar(read7BitChar());
   }
   printf("\r\n\r\n");

   // At this point, we have read 532 bits of the 704-bit Track A memory on
the
   // card reader IC.  Flush the remaining 172 bits.

   for (i = 0; i < 172; i++) {
      readBit();
   }
```

The actual data held on Track A varies from one card type to another. Track A can, moreover, contain alphabetic characters. Thus Track A is often used to store a cardholder's name, possibly address, an account number, and other numeric information. As code above show, all 704 bits of the Track A memory must be read out (even though not all of the bits will contain encoded data) before Track B data can be clocked out.

Translating Track B

Track B is encoded similar to Track A, except that these characters are 5-bit (four bits, plus one parity bit) encoded instead of 7-bit. The Track B character set contains numeric characters and symbols only, as shown in the `char5bit[16]` character array below.

```
//                    0123456789012345
char char5bit[16] = "0123456789:;<=>?";

// Clock out and decode a 5-bit character from the track memory, returning
the
// character value.  5-bit (numeric+symbol) characters are found on Tracks B
and C.

char read5BitChar()
{
   int i, c;

   // Each character is composed of 5 bits, which we clock out of the track
memory
   // beginning with the least significant bit.  Bit 5 is parity, which is
ignored.

   c = 0;
   for (i = 1; i < 32; i *= 2) {
      c |= (readBit() * i);
   }
   c &= 0x0F;

   return char5bit[c];       // Decode/return the character using the 5-bit
table.
}

....

   // Track B - 40 characters, 5 bits per numeric/symbol character including
parity.

   printf("Track B > ");
```

```
for (i = 0; i < 40; i++) {
   putchar(read5BitChar());
}
printf("\r\n\r\n");
```

As at the end of Track A, all remaining bits from the Track B memory must be read before proceeding to read Track C (if, that is, Track C is desired). Since the code has already read 200 bits from the Track B memory (40 characters x five bits), an additional 504 bits must be clocked out before Track C data can be accessed.

Translating Track C

Track C is encoded in an identical manner and uses the same character set as Track B, with a maximum of 107 7-bit characters encoded. Track C was originally intended to contain a rewriteable data area to support offline financial transactions, but it is not commonly used. Most magnetic stripe cards do not contain encoded data on Track C.

Conclusion

Magnetic stripe cards are used in a wide variety of financial, access control, governmental, and stored-value applications. By adding a simple MagTek card-swipe reader and a small amount of supporting hardware to the DS5250 EV kit, applications can be developed that combine magnetic stripe card-reading functions with the higher level security and cryptographic functions of the secure micrcontroller. An application to demonstrate magnetic stripe card reading and decoding can be easily implemented on the DS5250 secure microcontroller by using Keil's µVision C compiler.

***Note**: Due to its secure nature, the DS5250 secure microcontroller is controlled by U.S. export laws. A non-disclosure agreement (NDA) is required for access to the full DS5250 documentation, including the data sheet and user's guide. This application note and its source code, however, are freely available and can easily be ported to any of Maxim's other 8051-based microcontrollers.*

Related Parts	
DS5250	High-Speed Secure Microcontroller

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 4095: http://www.maximintegrated.com/an4095
APPLICATION NOTE 4095, AN4095, AN 4095, APP4095, Appnote4095, Appnote 4095