Technical Note

TN233

Interfacing the Rabbit 3000 with a Magnetic Stripe Card Reader

Most magnetic card reader hardware includes integrated electronics that convert the raw signals from a pickup coil into a clocked serial stream. However, these raw bits can represent a few different formats each with different bit lengths. Most card reader hardware does not perform any further translation of the card data and merely outputs these clocked bits. A library is available for the Rabbit 3000 that uses the input capture hardware to read this clocked bit stream. The library includes routines to decode the bits for the two most common formats.

Overview

The library `CARD_READER.LIB` is only for reading cards with hardware that outputs card data as clocked bits. This output format is sometimes referred to as 'F/2F.' The library does not support direct connection to magnetic pickup signals. The conversion from a raw pickup signal to clocked bits requires a mixture of analog and digital circuitry. MagTek Inc. (http://www.magtek.com) makes an interface IC that does this conversion.

Standard magstripe (magnetic stripe) cards have up to three separate tracks on them which are read simultaneously as the card is swiped through the reader. The library can be configured to read all three tracks and store them separately for decoding.

The standard formats which the library is able to decode are:

Table 1. Formats Supported by CARD_READER.LIB

Clocked Bit Stream Format	Description
ANSI/ISO BCD	• 5 bits per character • numbers only • odd parity bit on each character • LRC checksum at the end of each frame (track)
ANSI/ISO ALPHA	• 7 bits per character • numbers, uppercase letters, and some symbols • odd parity bit on each character • LRC checksum at the end of each frame (track)

Other proprietary formats exist, but the vast majority of magstripe cards use these two.

Hardware Connections

There are 4 input capture channels on the Rabbit 3000. The card reader software uses one for each track read, plus one channel for the ‘card present’ signal. The input capture hardware is used in its simplest mode: It interrupts on transitions of the strobe signal for each track. The input capture hardware can use pins on either parallel port D or G. The card reader library takes advantage of this by using port D by default, but allowing port G to be used instead if `CR_USEPORTG` is defined.

Here is the pinout:

Table 2. Parallel Port Pins used for Input Capture

Pin	Description
PD1 or PG1	/CARD PRESENT
PD2 or PG2	Track 1 DATA
PD3 or PG3	Track 1 STROBE
PD4 or PG4	Track 2 DATA
PD5 or PG5	Track 2 STROBE
PD6 or PG6	Track 3 DATA
PD7 or PG7	Track 3 STROBE

Most card reader hardware will only have pins for tracks 1 and 2, or even track 1 only. The card reader will also need power and ground signals connected. There is no real standard for card reader hardware, so you must consult the datasheet for your card reader model to make the correct connections. If your card reader device supports fewer than three tracks, you can leave the extra track signals on the rabbit unconnected. An application program must set the macro `READER_TRACKS` to however many tracks the card reader hardware has.

Library API

The card reader software is interrupt driven, so collection of card data is automatic. The routines described below initialize the input capture ISR, get completed bit streams, and translate bit streams into BCD or ASCII data. Because a set of card data is read automatically, a practical application should only need to check for a completed read 2-5 times per second, rather than constantly polling for incoming data.

CRinit

```
void CRinit( int tracks );
```

DESCRIPTION

Sets up the software to listen for `/CARD PRESENT` signal and read tracks.

PARAMETERS

`tracks`	Number of tracks used by reader hardware.

CRsetBuffer

```
void CRsetBuffer( int track, char *buffer, int len );
```

DESCRIPTION

Assigns buffer space for track data to be read into. Bits read from a card are packed into this byte-array with the LSB of the first byte being the first bit received.

PARAMETERS

`track`	Track using this buffer
`buffer`	Pointer to byte array for track data
`len`	Size of buffer (in bytes)

CRstartRead

```
void CRstartRead( void );
```

DESCRIPTION

Prepares to read the next card swipe that comes along. The track buffers should be considered ‘in use’ until `CRcheckRead()` below returns true.

CRcheckRead

```
int CRcheckRead( void );
```

DESCRIPTION

Indicates if a card read has completed. When this function returns true (1) `CRgetBuffer()` is called to determine how many bits were read from each track. Once a set of data has been read in, the buffers will hold it until the next call to `CRstartRead()`.

RETURN VALUE

`1`: Read operation has completed.
`0`: Still waiting for read operation to finish.

CRgetBuffer

```
int CRgetBuffer( int track, char **track_buffer );
```

DESCRIPTION

After a card is read, this function returns the number of bits (not bytes) read into the buffer for a track. The bits are arranged in the buffer with the first bit read as the least significant bit of the first byte.

PARAMETERS

`track`	The track to get data from.
`track_buffer`	Address of a char pointer variable. If this is a non-null address, the pointer will be set to the read buffer for the specified track.

RETURN VALUE

`0`: Empty buffer
`>0`: Number of bits read from the specified track.

CRalphaDecode

```
int CRalphaDecode( char *bits, int len, char *result_buf );
```

DESCRIPTION

Attempts to decode bits read from a card into ASCII data using the standard for ASCII data: 6 data + 1 parity bit per character. Looks for start and end sentinel characters, but does not include them in the result buffer. Checks parity and LRC.

PARAMETERS

`bits`	Array of raw bits from a card read. The read buffer for a track can be used 'as is' here.
`len`	Size, in bits, of the data read.
`result_buf`	A character array that will be filled with a null-terminated string containing the decoded ASCII string.

RETURN VALUE

1 - Decoding was successful. Decoded ASCII string loaded into `result_buf`.
2 - Decoding failed; the data had errors or is not in ASCII format.

CRbcdDecode

```
int CRbcdDecode( char *bits, int len, char *result_buf );
```

DESCRIPTION

Attempts to decode bits read from a card into BCD data, i.e. 4 data + 1 parity bit per digit. Looks for start and end sentinel characters, but does not include them in the result buffer. Field separators are included as their ASCII equivalent '='. Checks parity and LRC.

PARAMETERS

`bits`	Array of raw bits from a card read. The read buffer for a track can be used 'as is' here.
`len`	Size, in bits, of the data read.
`result_buf`	A character array that will be filled with a null-terminated string containing the decoded BCD string.

RETURN VALUE

`1`: Decoding is successful. Decoded BCD string loaded into `result_buf`.
`0`: Decoding failed, the data had errors or is not in BCD format.

Library Internals

The card reader library uses the 4 input capture channels built in to the Rabbit 3000. In this application, their pulse measurement capabilities are not used and they are set up as simple edge-triggered external interrupts. The function `CRinit()` sets this up to trigger an ISR written in assembly that reads bits into their designated buffers. The ISR also monitors the `/CARD PRESENT` signal to determine when a card has been completely read. The data in the buffers is kept until the next call to `CRstartRead()`. The buffered data must be processed and `CRstartRead()` must be called before another card can be read. This is generally not a burden, since cards will not be read more than a few times per second in most applications.

Sample Application

A very simple sample program (`CARD_READ_TEST.C`) is provided that reads each track, attempts to decode it into both ASCII and BCD formats, and prints the results if either decoding is successful. To use this sample you will need a card reader device that outputs card data in clocked format. Swiping a card through the reader should cause the card data to be printed to STDIO if any of the tracks have BCD or ALPHA data on them.

```
// Priority for interrupts on card reader lines
#define CR_IPLEVEL 1
//#define CR_USEPORTG

#use "card_reader.lib"

// Set to number of tracks on card reader hardware
#define READER_TRACKS 2

// Need a place to store bits as they are read in
#define TRACK_BUF_SIZE 128

char track_buf[READER_TRACKS][TRACK_BUF_SIZE];

main()
{
   int i;
   int bit_count;
   char track_str[150];

   CRinit(READER_TRACKS);

   // Assign buffers to each track to be read.
   for(i = 0;i < READER_TRACKS;i++)
      CRsetBuffer(i, track_buf[i], TRACK_BUF_SIZE);

   while(1){
      CRstartRead();
      printf("Waiting for card swipe.\n");
      while(!CRcheckRead());              //just wait

      for(i = 0;i < READER_TRACKS;i++){
         bit_count = CRgetBuffer(i, NULL);
         printf("Track %d(%d bits)\n", i+1, bit_count);

         if(CRalphaDecode(track_buf[i], bit_count, track_str))
            printf("\nAppears to be ALPHA data\n%s\n", track_str);

         else if(CRbcdDecode(track_buf[i], bit_count, track_str))
            printf("\nAppears to be BCD data\n%s\n", track_str);

         else
            printf("Could not decode bits\n");

         printf("\n");
      }
      printf("\n");
   }
}
```

The steps in this sample are:

1. Initialize the card reader ISR by calling `CRinit()`
2. Set up the bit storage buffers for each track with calls to `CRsetBuffer()`
3. Call `CRstartRead()` to listen for data from the reader
4. Wait for `CRcheckRead()` to return true, indicating a card has been completely read.
5. For each track, get the number of bits read from `CRgetBuffer()` and attempt to decode it by calling `CRalphaDecode()`, and then `CRbcdDecode()`.
6. Print results and loop back to step 3.

This is the bare minimum of a card reading application. The important point is that the card data must be unloaded or processed from the buffers before another card can be read. The ISR will not overwrite these buffers unless `CRstartRead()` is called, even if another card swipe occurs.

Summary

A magnetic card reader is a widely used device that is simple to interface to a Rabbit 3000 based controller system. General information on magnetic stripe cards is a little harder to come by. This is due to the notion the information can be used for criminal purposes. One good source on the web is:

`http://www.repairfaq.org/filipg/LINK/F_Phrack_Mag.html`

A major manufacturer of mag-stripe card equipment is MagTek (http://www.magtek.com). They were the source of card reader hardware used during development of the card reader library. They also manufacture the interface chips described at the beginning of this document.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792

www.zworld.com

Rabbit Semiconductor

2932 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-8400
Fax: (530) 757-8402

www.rabbitsemiconductor.com