Swope Filter Tag Reader

Revisions

Rev	Date	Notes
1.0	2014-01-09	Initial release (au)
1.1	2014-01-21	Added ADM1490 RS-422 transceiver information & more links

Summary

The Filter Tag Reader reads data from near field communications (NFC) tags. For tags mounted on filter holders this information can identify filter wheel contents. It can also write to the NFC tag.

Software

Communications is through an RS-422 serial port on the Galil DMC 4080 controller on the Swope IMB. The setup is 19,200 baud, 8-bits, no parity, one stop bit. There is no flow control; the Galil handshaking lines (RTS and CTS) are tied.

Commands sent to the tag reader are one-byte ASCII characters as described below. The command character is echoed. If the command is not 'w' then the echoed command character is followed by "\r\n". If there is returned information (the tag data, status information, or an error code) that information is sent and followed by another "\r\n". Finally the prompt character '>' is sent.

Commands

Command	Name	Details
0 or **1**	Select wheel	These are ASCII characters 0 (numeral zero, not oh) and 1 (numeral one, not ell). Subsequent commands will reference the selected filter wheel, 0 or 1.
i	Tag ID	Prints the 32-bit MiFare tag ID in hex. This probably isn't a useful number to know but if a reply is returned you know it's a MiFare Classic tag.
r	Read tag	If a tag is present 32 bytes are returned. A brand-new blank tag will likely contain zeros (0x00, not the ASCII numeral). For prepared tags, an end-of-string byte (\0) is *not* necessarily included in the 32 bytes.
R	Reset	Reboots the microcontroller and resets the hardware. Total time should be less than one second and you'll see a "WD\r\n>" returned. The "WD" refers to a watchdog reset, which can happen only with this command (the watchdog timer is disabled except for this reset function). The input stream to the tag reader is cleared on reboot so commands sent immediately following the "R" and before the WD response will be lost.
s	status	Prints system status information. In order, the returned items are the program name ("FIlter_Reader"), this software version number (yyyymmdd), firmware version of the selected tag reader (if it's "X.X" and not "1.6" then no tag reader is present), and the contents of the program status byte. Bit 0 of the status byte shows the currently selected wheel number (0 or 1). The other seven bytes are program error codes. If they're not all zeros (that is, if the status byte is not 0x00 or 0x01), you should probably try a read command ('r') or do a reset ('R' command)..
w	Write tag	This command writes the following 32 bytes to the tag. A carriage return (\r) terminates data entry if you have fewer than 32 bytes to write. The \r will not be included in the 32 bytes. The remainder of the 32 bytes will be padded with spaces (' ' or 0x20). Example: **w**H-Beta Filter\r will write "H-beta Filter" padded with spaces into the tag.
\r (CR)	No-op	Sending a carriage return will return the command prompt string "\r\n>"

Errors

The tag reader may reply with an error code. The error code format is “\r\nENN\r\n>” where NN are ASCII numerals as described below. Note that the “\r\n>” sequence right after ENN distinguishes this from a read-back of valid tag data which always returns 32 bytes. Except for errors 10, 11, and 99, you’ll need to look at the firmware source code for why the problem occurred.

Code	Meaning
E99	Not a command (the byte sent was not recognized as a command)
E10	No tag or an unrecognized tag at wheel 0
E11	No tag or an unrecognized tag at wheel 1
E20	Problem writing to tag on wheel 0
E21	Problem writing to tag on wheel 1
E80	Initialization error on tag reader for wheel 0
E81	Initialization error on tag reader for wheel 1

Reboot transmissions

The tag reader will send these messages without being commanded. All of these happen after a reboot and all are followed by a “\r\n>” sequence.

Message	Meaning
BO	Reboot after power brown-out (supply voltage under 2.7VDC)
PB	Pushbutton reset (the button on the MCU hardware was pushed)
PU	Power up start
WD	Watchdog reset (only happens with the R command)

Firmware

Source Code

The firmware source code resides in a GitHub repository. The program is for an Atmel ATtiny4313, compiles under avr-gcc (Windows version), and uses avr-gcc specific macros.

Status Register

The contents of an 8-bit status register is the last item printed on a status command (letter s). Bit 0 of this byte contains the current selected filter wheel (from the 0 or 1 commands). The other seven bits contain error codes that we expect only rarely. These might be caused by a hardware failure, loose wire connection, or a bad tag. If any of bits 1-7 are not zero, try a Reset ('R') command to see if that clears the condition. These can be confusing in the sense that you cannot tell which tag reader (or both) caused the error.

Bit 0 - Indicates which filter wheel reader is connected.
Bit 1 - recvAck error. The reader did not send an ACK frame after a command.
Bit 2 - Firmware version error. The reader firmware wasn't read or is wrong.
Bit 3 - authenticateBlock error. The password for the block is not 0xFFFFFFFFFFFF.
Bit 4 - readBlock error. A problem reading one or both data blocks.
Bit 5 - writeBlock error. A problem writing a block.
Bit 6 - sendCmdAck error. A command transmission / reception error.
Bit 7 - init_pn531 error. Initialization (samconfig or rfconfig error).

Hardware

Tags

We use Mifare Classic 13.56 MHz tags. They have a few hundred bytes of usable storage memory although we're only using 32 bytes and come in a wide variety of shapes, sizes, and colors. Because our filter frames are made of aluminum, we need to use "anti-metal" tags, which have a ferrous layer isolating the tag from the underlying aluminum.

You should get unformatted tags (*not* preformatted NDEF tags).

These tags come from the factory with authentication key (password) of 0xFFFFFFFFFFFF and we do not change this.

Tag reader

The NFC tag reader controller is the NXP PN532 chip. We chose an Adafruit-branded "shield" version of the microBuilder board. This board has a built-in antenna and runs on 5 VDC.

The PN532 can communicate via SPI, I2C, or serial interfaces. We chose I2C (TWI). In retrospect, it might've been easier to use SPI for this project since the PN532 has only one fixed I2C address option, which makes using two of them more difficult.

MCU

The MCU is an Atmel ATtiny4313. This chip has a hardware UART for serial communications to the Galil controller. We're using a 14.7456 MHz crystal that allows us to choose common serial baud rates.

Bus switch

Since all PN532 boards have identical I2C addresses we will use the PCA9543AD I2C bus switch to select which of the two boards we want.

RS-422 conversion

To convert the RS-422 signals on the Galil controller to TTL serial levels, we'll use two ADM1490 transceivers. Each ADM1490 has one transmitter and one receiver. We need two because the Galil RS-422 interface requires hardware handshaking with the RTS and CTS signals as well as data receive and transmit.

Glossary

Item	Description
I2C	Inter-Integrated Circuit bus
IMB	Instrument Mounting Base (Carnegie Observatories notation)
MCU	MicroController Unit
NDEF	NFC Data Exchange Format
NFC	Near Field Communications
SPI	Serial Peripheral Interface
TTL	Transistor-Transistor Logic
TWI	Two-Wire Interface, similar to I2C, functionally equivalent
UART	Universal asynchronous receiver/transmitter