Remote Control Program 9. May. 2016

by Mark DiVecchio
markd@silogic.com

This is part of the Remote Train Control Manual.

This version of this document covers using the RTC program with a radio connection between the PC and TIU. There is another document that covers the wired connection between the PC and TIU.

Background

I've been working on understanding the interface between the Remote and TIU for several years. I started with the work done by Mike Hewett in about 2011. He investigated the wired connection (using a phone handset cord) between the two. He figured out a lot of the interface. It was RS-232 (using 0v and 3.3v levels). It was 9600 baud, 8 data bits, no parity, 1 stop bit. He could see command packets going between the Remote the TIU. He wrote software to capture those packets, store them on a PC and send them to the TIU on request.

I took the work that Mike did and expanded on it. I developed a better understanding of the command packet from the Remote and the response packet from the TIU. I put all that I learned into a program called Remote Train Control.

I was still using a wired connection between the PC the TIU. I thought that I could figure out how the radio connection worked.

Searches on Google developed a few leads:

1. Frequency requests to the FCC for Hand Held RF module and for Base RF module:

https://fccid.io/OT8RFMR#axzz3RPE7qRYL
https://fccid.io/OT8RFMT#axzz3RPE7qRYL

This showed that the Remote transmitted on 916.5 MHz and the TIU transmitted on 905.8 MHz

2. This I found this post on an SDR web site:
SDR-noob needs help. Want to decode 9600bps OOK signal at 905.8MHz.

submitted 17 Nov 2014 by playaspec

Hi all. I've been trying to reverse engineer the protocol used by a toy train with little luck. Through a great deal of digging through Google, I've discovered that the remote and the base communicate at 905.8MHz, using OOK at 9600bps. I would like to snoop this protocol to bring the train under computer control. Can anyone lend a hand?

The radio chip in the receiver (and I assume the transmitter) is a TI TRF6901

He never got any responses, except from me, and could not add to his comment at that time. I don't know if he ever got to a solution. But at least, if he was right, I could search for OOK and the TRF6901.

3. I needed to find out if anyone sold a radio board using the TRF6901. Searching the Internet turned up nothing.

4. On the TI web page, the device was listed as "NRND" - TI speak for "Not Recommended for New Designs". I downloaded the data sheet and saw that the chip supported FSK (Frequency Shift Keying) and OOK. So the Remote and TIU probably spoke either FSK or OOK. I found this article: "I'm OOK, You're OOK?" from Maxim. Another big hint was the comment on the TI TRF6901 web page that stated: "Replaced by CC1101".

5. Since the TRF6901 was obsolete, maybe the CC1101 could be used. The data sheet said that it could do OOK. It was worth investigating.

6. Now did anyone make a board using the CC1101? I found Erwan's Blog which talks about connecting a CC1101 to an Arduino. He used a board from Elechouse which contains the CC1101.

7. Then I found the panStamp web site and product called the "panStamp AVR 1". Exactly what I needed, an *Atmega328p* MCU and CC1101 on a small circuit board along with a carrier board containing a USB port for a PC. I've used the *Atmega328p* MCU (its the basis of the Arduino) in other projects so I was familiar with it. Easily covers the two frequencies I needed to cover. Its a very low power draw solution. I ordered one. Took about 2 months to get it from Spain - I didn't get the expedited shipping. Mental note: next time, pay for the faster shipping.

8. The *Atmega328p* MCU is programmed using the standard Arduino IDE and board files from panStamp. Their web page shows you everything you need to know about installing the IDE with panStamp support. I learned that panStamp supported the CC1101 using only a protocol called "SWAP". A very complex high level protocol. For most users it is the best choice - as long as the radio on the other end of your connection is speaking SWAP. I needed to be speaking OOK.

9. I learned how to program the CC1101 from scratch. I searched the Internet and got little bits of help.

Months of digging through the CC1101 data sheet and test code for the *Atmega328p* MCU finally led me to a working receiver. Then a working transmitter. I wrote a program based on the example program "modem.ino". I called my program "RTCModem". I could communicate between the PC and the TIU with my Remote Train Control program. (note: this paragraph took about 5 months actual work.)

The developers of the panStamp had written a fairly complex and large set of routines to use the CC1101 in “SWAP” mode. My first problem was understanding that code so that I could start to modify it.

I could see that there were 48 registers in the CC1101 that controlled its operation. The CC1101 is very complex chip but as I read the specifications, I could see that in OOK mode, most of the chip's high level protocol would not be used.

I found a program called SmartRF Studio 7 from TI which helps set the values needed for the CC1101 configuration registers. An app note, DN022, helps with a few registers that the SmarRF program gets wrong.

Here are the configuration registers and their values that I use (as a snippet of C code). There are other registers which are not changed from their default values and are not listed here.

```
static const registerSetting_t C916_5[]=
{
  {CC1101_IOCFG2,0x2E}, // High-Impedence - GDO2 is not connected on the panStamp
  {CC1101_IOCFG1,0x2E}, // High-Impedence
                        // GDO1 is also used as the SPI output from the CC1101
                        /// to the AtMega328
  {CC1101_IOCFG0,0x0D}, // GDO0 is hardwired as Serial Data input to CC1101 in
                        // asynchronous TX mode and this setting selects it as Serial
                        // data output from the CC1101 in asynchronous RX mode
  {CC1101_FIFOTHR,0x47}, // OK per DN022, when RX filter bandwidth <= 325 kHz
  {CC1101_PKTCTRL0,0x32}, // OOK
  //{CC1101_CHANNR, 0x00}, // 100KHz channels - #0 gives 905.799438 MHz  -  Remote RX
  {CC1101_CHANNR, 0x6B}, // 100KHz channels - #107 gives 916.496826 MHz  -  Remote TX
  {CC1101_FSCTRL1, 0x06}, // IF Frequency   152 KHz
  {CC1101_FREQ2, 0x22}, // 905.8 MHz - base frequency
  {CC1101_FREQ1, 0xD6},
  {CC1101_FREQ0, 0xA5},
  {CC1101_MDMCFG4,     0x88}, // Bandwdith
      // 0xF8 = 58 kHz - would not work with with RTCEngineE (too narrow)
      // 0xE8 = 68kHz - Worked with RTCEngineE.
      // 0xD8 = 81 kHz - Worked with RTCEngineE.
      // 0xC8 = 101 kHz - Worked with RTCEngineE, would not work with my Rev I3A TIU.
      // 0xB8     - Worked with RTCEngineE, would not work with my Rev I3A TIU.
      // 0xA8     - Worked with RTCEngineE. Borderline operation with my Rev I3A TIU.
      // 0x98 = 162.5 kHz - one TIU would not work at 101Khz, it worked at 203 kHz
      //          and at 162.5 kHz.
      //          Worked with RTCEngineE. Worked with Rev I3A and Rev L TIU.
      //          Would not work with Mike's Rev I TIU
      // 0x88 = 203.125 kHz  Worked with RTCEngineE. Worked with Rev I3A TIU.
      //          Seems like solid operation without any noise issues.
      // 0x48 = 406.2 KHz - would not work with Mike's Rev I TIU.
      // 0x28 = 541 KHz - worked with Mike's Rev I TIU - but furthur testing
      //          showed the RX was much more suseptable to noise (seen when
      //          testing using RTCEngineE).
```

```
  {CC1101_MDMCFG3, 0x83}, // Data Rate  9.596 baud
  {CC1101_MDMCFG2, 0x30}, // Modulation format ASK/OOK
  {CC1101_MDMCFG1, 0x21}, // Channel Spacing w/MDMCFG0
  {CC1101_DEVIATN, 0x14}, // Deviation - not used in ASK/OOK
  {CC1101_MCSM1, 0x00}, // Always select clear channel, was 0x30 unless
                        // currently receiving a packet
  {CC1101_MCSM0, 0x18}, // FS_AUTOCAL  when going from IDLE to RX or TX
  {CC1101_FOCCFG, 0x16},      // Per CC1101 doc, FOC_LIMIT[1:0] s/b 0 for OOK.
                        // That means this should be 0x14
  {CC1101_AGCCTRL2, 0x07},    // SmartRF gives 0x43  -
                        //per DN022 it should be between 0x03 and 0x07,
                        // 0x06 and 0x07 seem to work well
                        // 0x92 with 0x03, 0x04, 0x05, 0x06 generates spurious bits
                        // 0x91 with 0x03, 0x04, 0x05 generates spurious bits,
                        // 0x06 and 0x07 seem to work
  {CC1101_AGCCTRL1, 0x00 | 0x30 | 0x08}, // setup for CS detection relative,
                                         // Absolute disabled.
  {CC1101_AGCCTRL0, 0x91},    // per DN022 between 0x91 and 0x92, overall,
                              //0x91 seems to work well
  {CC1101_WORCTRL, 0xFB},
  {CC1101_FREND1, 0x56}, // OK per DN022, when RX filter bandwidth <= 101 kHz
                        // NOTE: I have since widened the bandwidth
                        // to 162.5 kHz but I didn't change this value.
                        // Seems to work fine.
  {CC1101_FREND0, 0x11}, // PA Power Setting index 1 - select PATABLE[1] when
                              // transmitting a one
  {CC1101_FSCAL3, 0xE9},
  {CC1101_FSCAL2, 0x2A},
  {CC1101_FSCAL1, 0x00},
  {CC1101_FSCAL0, 0x1F},
  {CC1101_TEST2, 0x81}, // OK per DN022, when RX filter bandwidth <= 325 kHz
  {CC1101_TEST1, 0x35}, // OK per DN022, when RX filter bandwidth <= 325 kHz
  {CC1101_TEST0, 0x09},
};
```

CC1101 Frequency Control

I set the base frequency for the CC1101 to 905.799438 MHz (nominally 905.8 MHz), I set channel spacing to 99.975586 KHz (nominally 100 KHz). I use two different channel numbers: channel 0 for Remote RX and TIU TX and channel 107 for Remote TX and TIU RX.

Channel 0 (0 * channel spacing) gives a frequency of 905.799438 MHz.

Channel 107 (107 * channel spacing) gives a frequency of 916.496826 MHz.

I set the baud rate to 9.596 Kbaud which is as close as the CC1101 can come to 9600 baud.

I didn't know what to use for the receiver bandwidth. I originally tried 58 kHz, then101 kHz but that was apparently too narrow. I ended up using 162.5 kHz

I don't understand many of the other configuration register setting and I'm just following what SmartRF says.

CC1101 Power Levels

There are 8 power level registers in the CC1101. OOK uses two of these. Register 0 is the power level for the transmitter off state – I used a value of 0x00. Register 1 is the power level for the transmitter on state – I used a value of 0xC3. This value set the transmitter output level to 10 dbm. Power register 1 is selected by the FREND0 configuration register.

10. To program the CC1101, I took the code written by the panStamp developers that supports SWAP, deleted all of the SWAP routines and added routines to implement OOK. For the most part, that was deleting the routines that handled the data bytes sent and received. In addition, it was changing the parts of the code that loaded the configuration registers to, instead, load up the values that I showed above.

The code in my RTC Modem program treats the CC1101 as an RS-232 serial connection. I used the SoftwareSerial library connected to the CC1101 GDO0 pin. This pin is used for both serial TX and RX since the interface is half-duplex.

I use the HardwareSerial to communicate with the PC. Both serial interfaces run at 9600 baud.

So the RTC Modem program basically listens on each serial port and if a character comes in, the program merely sends it out the other port. There is some debug code left but it is deactivated.

11. The protocol over the radio is just like the RS-232 wired connection. You might think of it as inverted, though. When the RS-232 signal is a '1' or "Marking", the transmitter is off. In RS-232-land, this is the idle state. With a radio, we want the transmitter to be off when the connection is idle. This lets other remotes transmit and control the TIU. It also saves our battery power. When the RS-232 signal is a '0' or "Spacing", the transmitter is on. (On our wired RS-232 connection, a '1' is indicated by 3.3v on the wire and a '0' is indicated by zero volts.)

12. I found that both the Remote and the TIU require what I call a "wakeup" signal be transmitted to them before data is transmitted. I'm guessing that the receiver circuit needs a carrier signal for some amount of time in order for it to lock onto the incoming signal. I found by experimentation that a carrier on signal of about 2000 microseconds followed by a carrier off signal of about 600 microseconds worked. Following this wakeup pulse, the next carrier on is the START bit of the first character of the packet of data. If you look at the code, you can see where I do this.

Feedback appreciated.

Look at my web page for updates or changes:

http://www.silogic.com/trains/OOK_Radio_Support.html