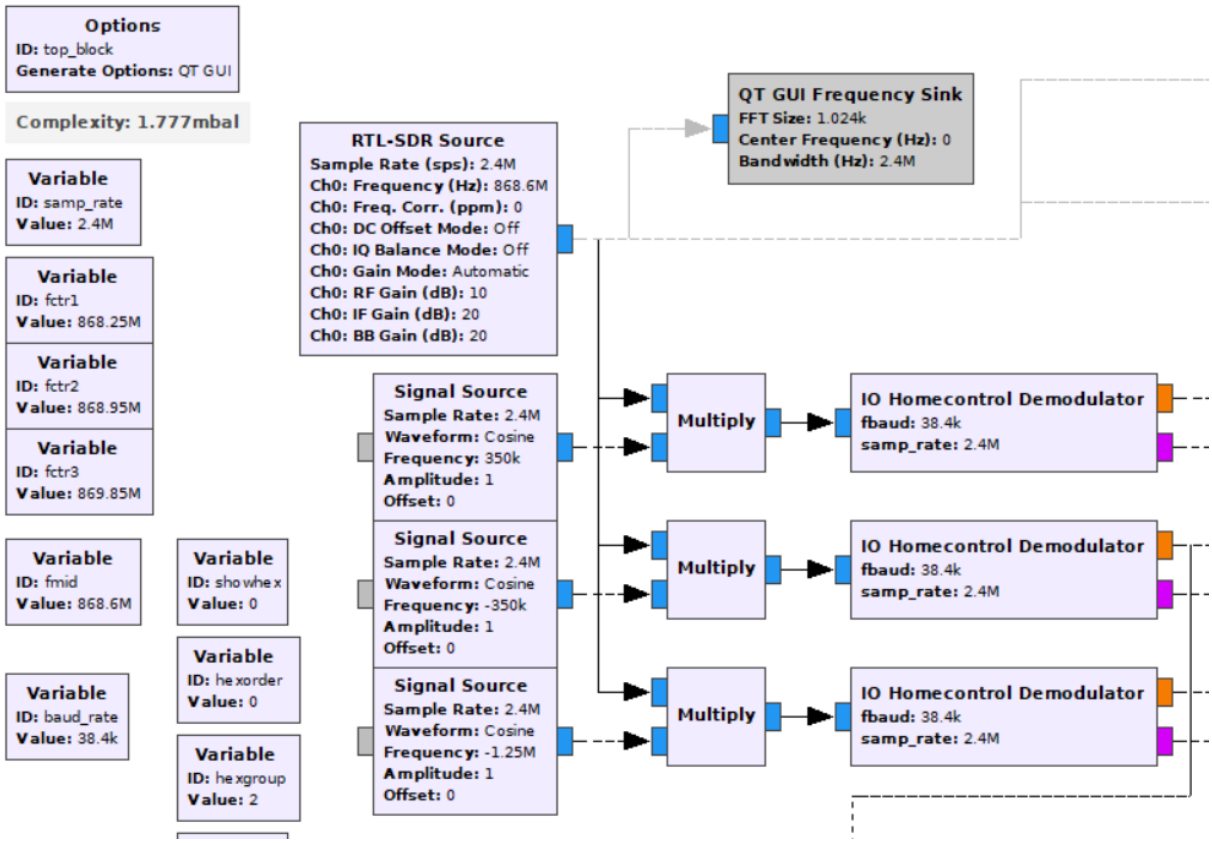
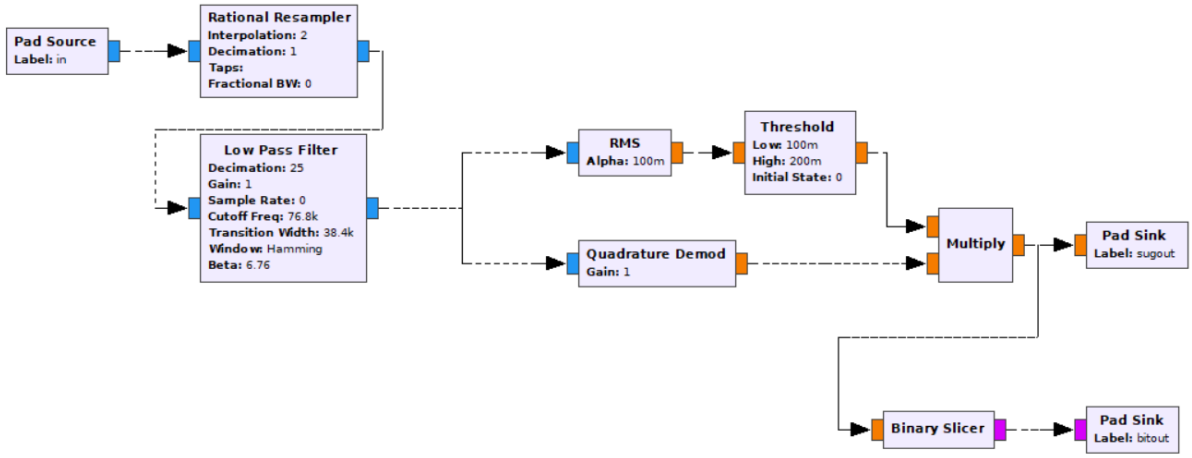
Reception and Demodulation

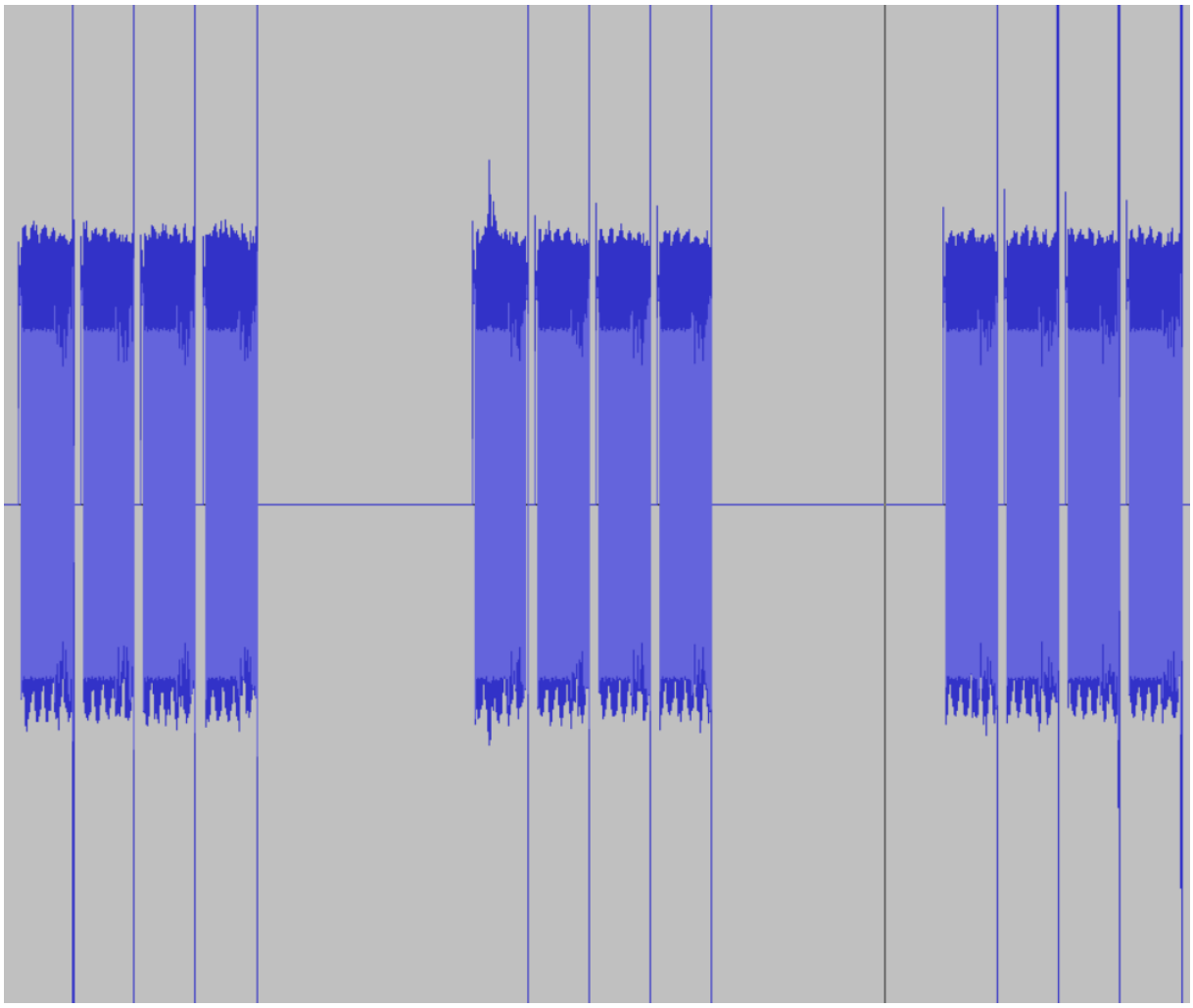


- Sample rate is 2.4 MHz
- Center Frequency was set to 868.6MHz, so that I can sample the three channels mentioned in the ADF7022 Datasheet (2-page version) at once.
- To demodulate, I first mix each channel down to the base band (the multipliers in the above graph) and then demodulate them using the following graph



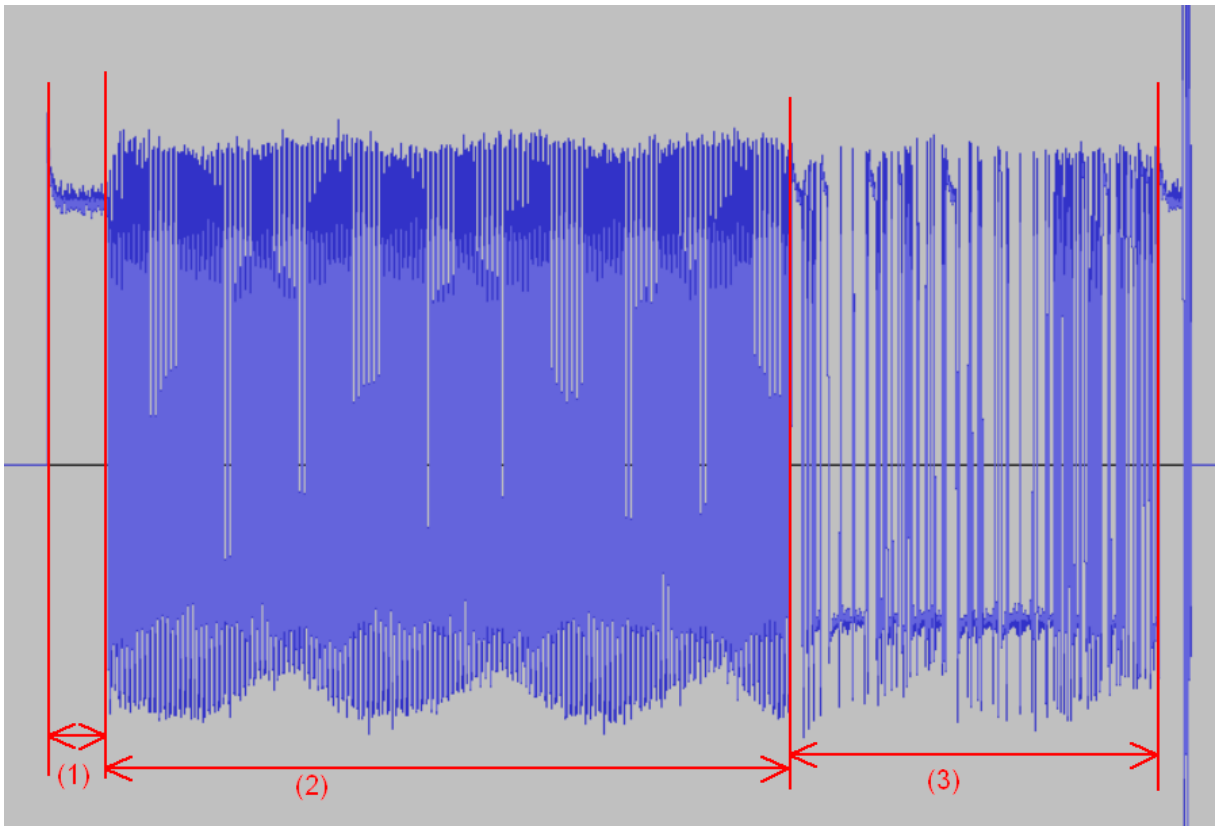
- First step: Interpolate to twice the sample rate (2.4 --> 4.8 MHz)
- Low pass filter with twice the baud rate and decimate down by 25.
- This results in a sample rate of 192 kHz
- Why? this is exactly 5x the symbol rate of 38.4kbaud
- The signal is split in two paths
  - The amplitude to actually recognize packets and blank the noise floor by a threshold.
  - A quadrature demodulated signal.
- Both signals are recombined by multiplication.
- One output carries the "analog" signal, the other one the bit sliced version (only 0 or 1)

The resulting "analog" signal can be stored in a wav file which, cropped to one transmission, looks like this:



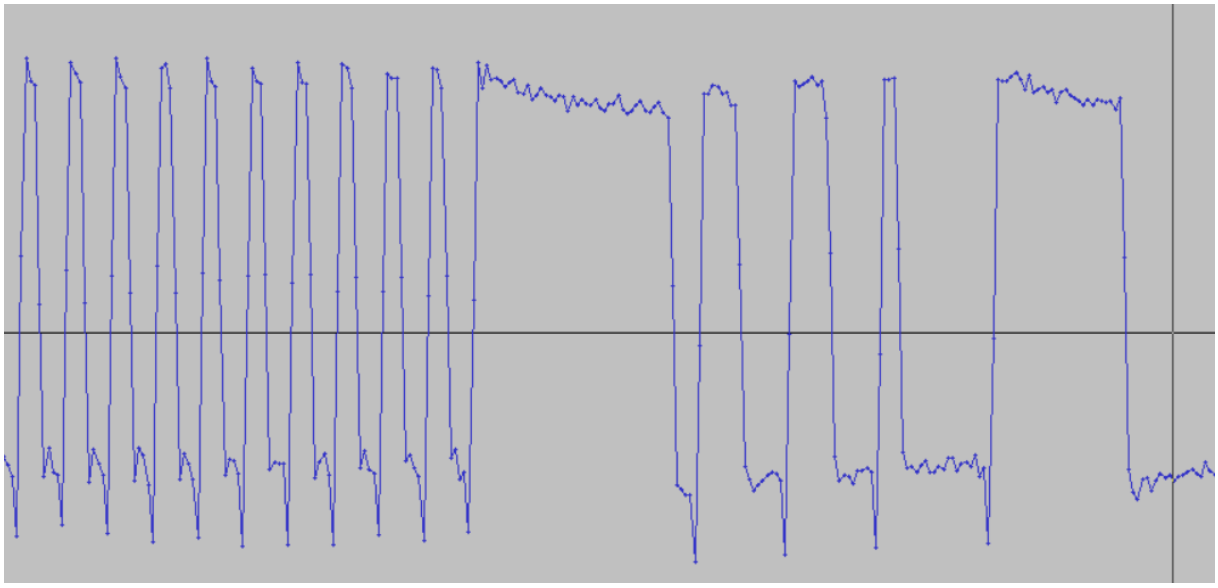
- There are three groups of four packets each
- The four packets in each of the groups are bitwise identical.

Zooming into the beginning of one of the packets shows their basic structure:



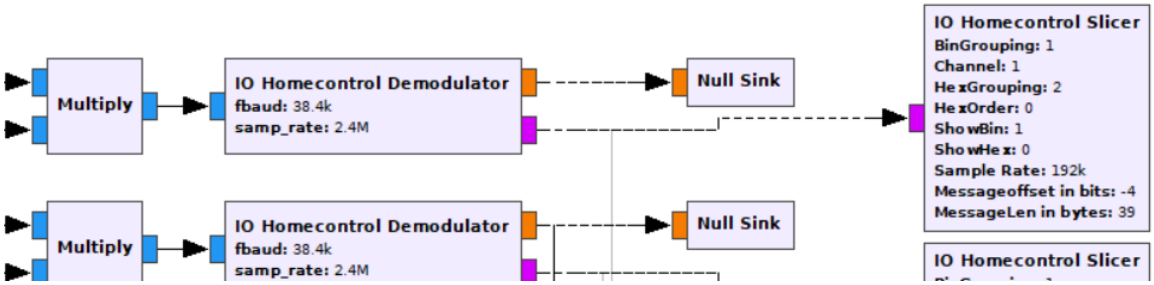
1. Packet start - RF is on, but not modulated
2. Synchronization phase - this is likely used to synchronize the receivers clock with the transmitter. It contains a 1010101... chain
3. Data phase - direct binary encoded

The end of the synchronization phase (2) and the beginning of data phase (3) looks as follows:



One can see the last ...0101010-bits and then a group of ones - then the actual bits seem to start with 01100110...

In GnuRadio, the binary sliced data goes to the next block which does the synchrinization and sampling of the bits:



This block encapsulates a python script which does this job by implementing a simple state machine.

The whole python code including the grc-Files can be found here: https://github.com/101010b/io-home

Feel free to add to the repository.

Some more details to the parameters to the final block:

- Sample rate - the actual sample rate of the data reaching this skript. As shown above, it is useful to be at a multiple of the symbol rate of 38.4kHz. In my code it is 192 kHz (5x38.4k)
- Channel - informal parameter only - it is output together with the data to track the channel it was received on
- ShowHex - Combine bits to bytes and show as hex values - see open questions below
- HexOrder - How to combine bits to bytes (LSB fist (1) or MSB first (0)) - see open questions below
- HexGroup - Group one or more bytes in the output - (0: No groupng at all)
- ShowBin - Show output as binary stream
- BinGroup - Group one or more bits in the output (0: No grouping at all)
- MsgLen - Length of the whole Message in bytes
- MsgOffset - Offset for starting the actual data - see open questions below

Open Basic Questions

- Which is actually the first bit? As shown above, the sync-sequence stops with a bunch of ones. Which is actually the first data bit? By default (MsgOffset = 0), the data output starts with the first zero after the set of ones. By changing MsgOffset, you can use part of the ones sequence as well or blank additional bits as they don't seem to differ in all my transmitters - but this may be coincidence.
- How to interpret the databits to bytes (MSB first or LSB first)?

Example Messages in different notations

- MsgOffset = -4 --> the four "1" at the beginning are likely part of the packet start header
  1 1 1 1 0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0
  1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0
  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0
  0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1
  1 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
- MsgOffset = 0 - The first 0 ist the first bit
  0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1
  0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0
  0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
  1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1
  0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
- Converted to Hex at MsgOffset=0, MSB first, grouped by 4 bytes to improve readability
  6643F004 01005F92 C4396840 14050D00 40100C37 50401204 CB124893 A5A362D1 57D495FF FFFFE5
  6643F004 01005F92 C4396840 14050D00 40100C37 50401204 CB124893 A5A362D1 57D495FF FFD3FF
  6643F004 01005F92 C4396840 14050D00 40100C37 50401204 CB124893 A5A362D1 57D495FF FFF5FF
  6643F004 01005F92 C4396840 14050D00 40100C37 50401204 CB124893 A5A362D1 57D495FF FFFFF8
  6643F004 01005F92 C4396840 9205FF40 50D004E1 00401205 CB7948F3 7C637942 356D61FF FFFFD3
  6643F004 01005F92 C4396840 9205FF40 50D004E1 00401205 CB7948F3 7C637942 356D61FF FFFFFD
  6643F004 01005F92 C4396840 9205FF40 50D004E1 00401205 CB7948F3 7C637942 356D61FF FFFFED
  6643F004 01005F92 C4396840 9205FF40 50D004E1 00401205 CB7948F3 7C637942 356D61FF FFC9FF
  6643F004 01005F92 C4396840 9205FF40 50D00541 7FC01204 2B71C5D5 A4FF7C59 96E483FF FFFFF2
  6643F004 01005F92 C4396840 9205FF40 50D00541 7FC01204 2B71C5D5 A4FF7C59 96E483FF FFE9FF
  6643F004 01005F92 C4396840 9205FF40 50D00541 7FC01204 2B71C5D5 A4FF7C59 96E483FF FFF3FF
  6643F004 01005F92 C4396840 9205FF40 50D00541 7FC01204 2B71C5D5 A4FF7C59 96E483FF FFFFFF
  The three groups of four repeated packets each can be clearly seen - the last bytes which change from transmission to transmission are likely not part of the packet as they look like random noise.They can be blanked by reducing the message length parameter. I guess the packet ends before the final FF.
  All my transmitters start the transmission with 664 - so this may also be part of the constant header.

I am thankful for any additional inputs on this.