Hacking Nordic proprietary protocol with GNU Radio

H. Boeglen

XLIM, UMR CNRS 6172, Université de Poitiers, France

European GNURadio days 2019

Outline

1 – Nordic Semiconductor 2.4GHz Proprietary solutions

2 – Bastille Research gr-nordic OOT module

3 – Building a nrf24L01+ transceiver with GNU Radio

4 – Conclusion

Outline

1 – Nordic Semiconductor 2.4GHz Proprietary solutions

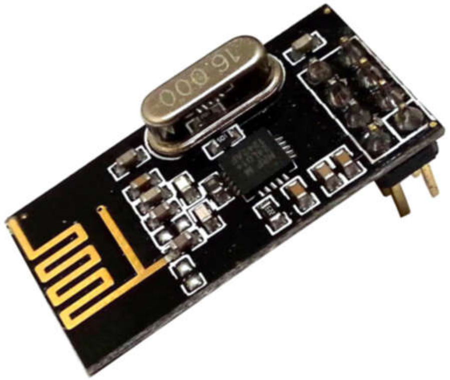
2 –

3 –

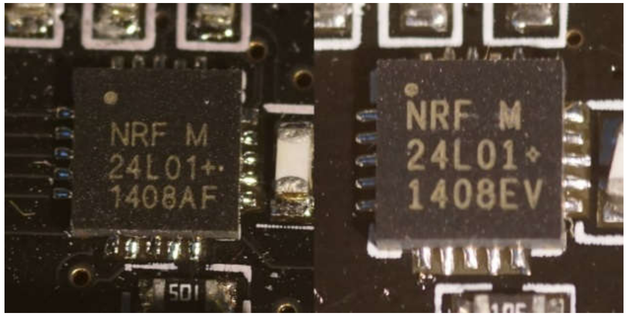
4 –

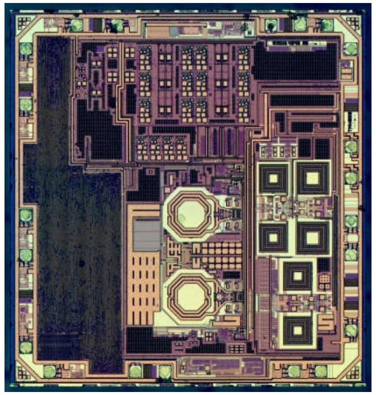
1. Nordic Semiconductor 2.4GHz Proprietary solutions

- **NRF24L01+**
  - **Well known by embedded systems guys**
  - **This module can be found on Chineese sites for less than 1€**

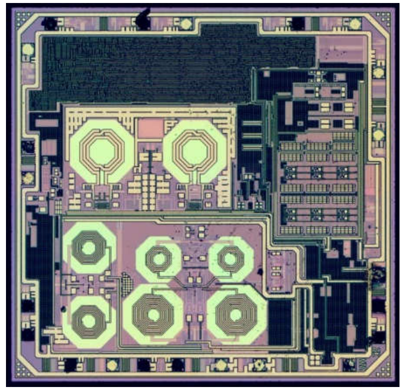


  - **For that price most of them are fake (chip copied)**





Original (250nm)



Fake (350nm)

- NRF24L01+
  - Inside NRF24L01+ transceiver

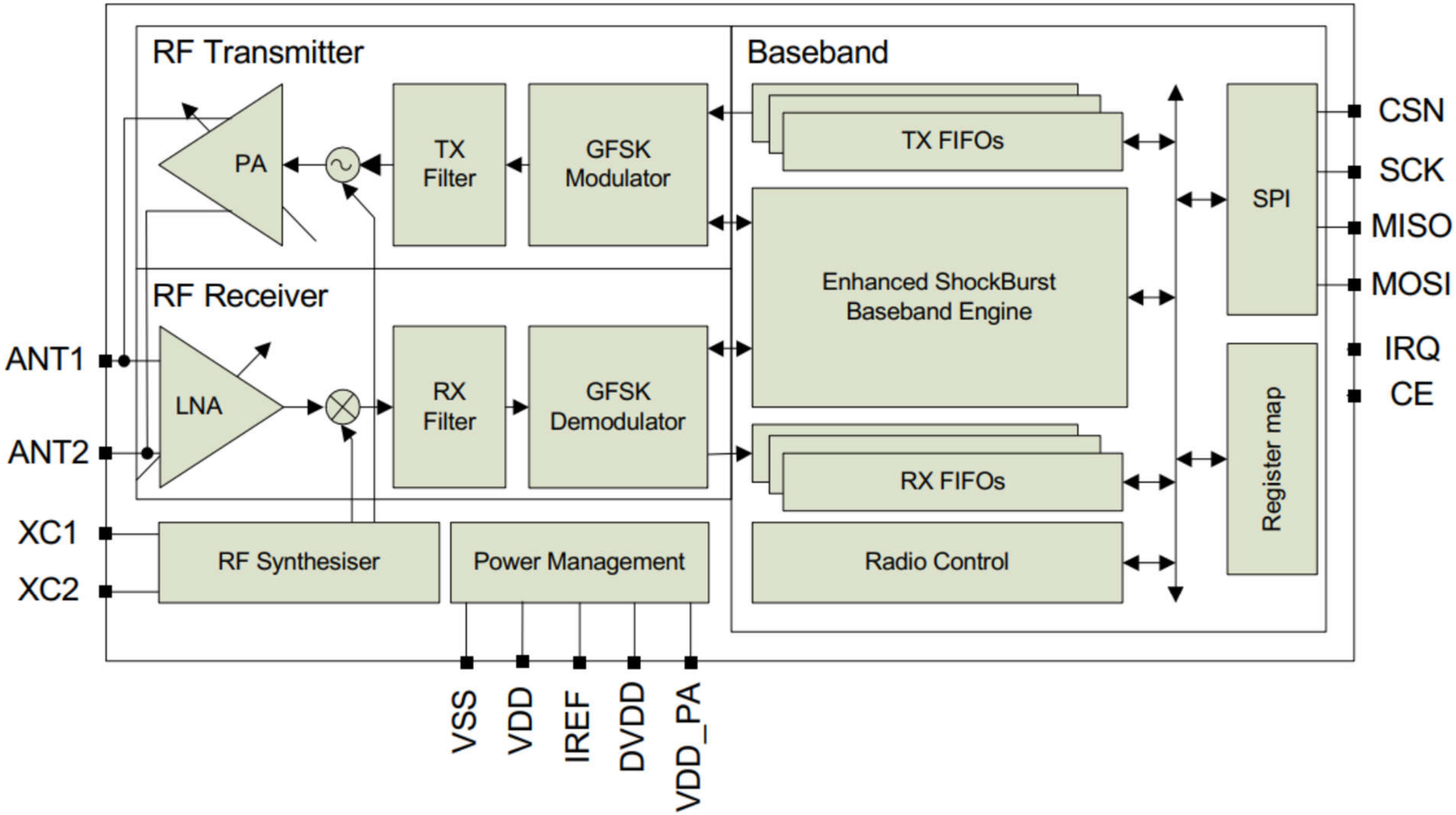
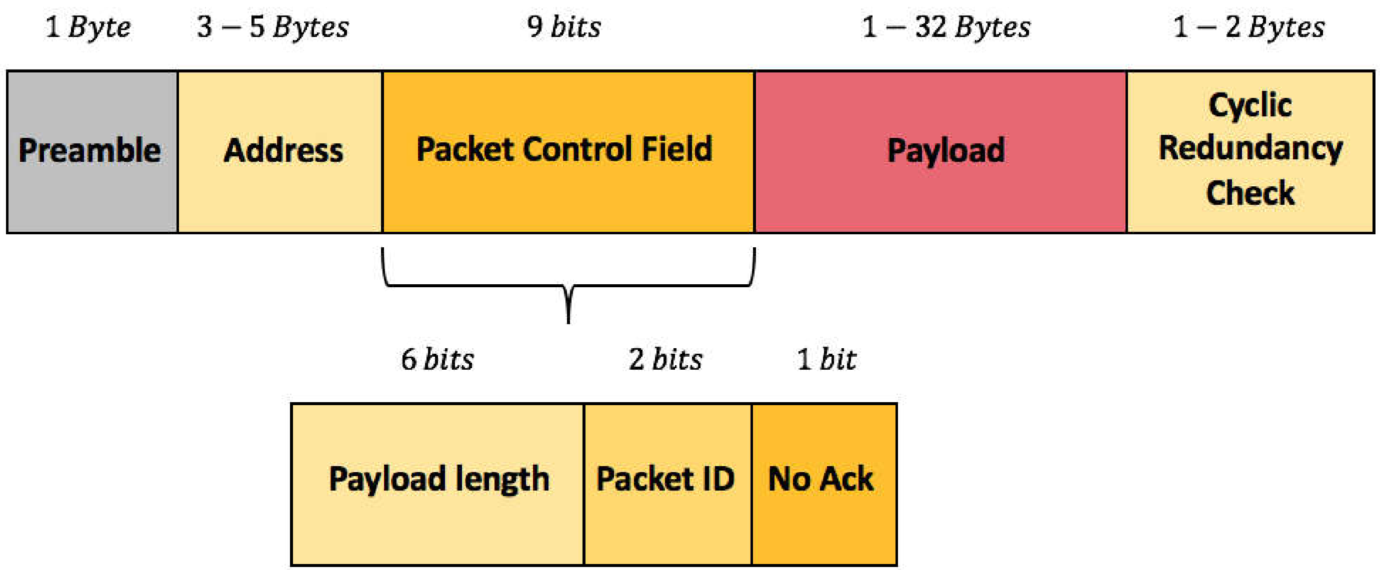


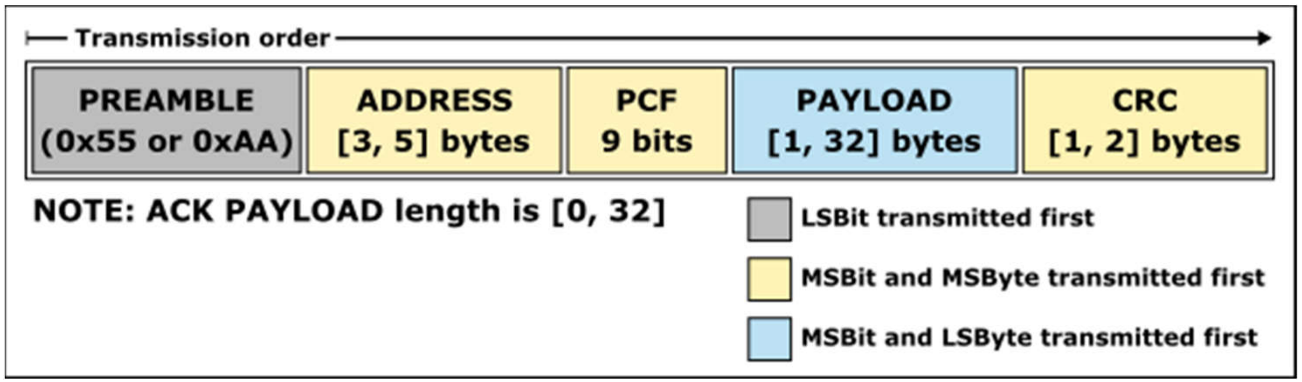
Figure 1. nRF24L01+ block diagram

- **NRF24L01+**
  - **Main features:**
    - **Worldwide 2.4GHz ISM band operation, 126 RF channels**
    - **GFSK modulation**
    - **250kbps, 1 and 2Mbps air data rate**
    - **Programmable output power: 0, -6, -12 or -18dBm**
    - **11.3mA at 0dBm output power**
    - **-94dBm sensitivity at 250kbps**
    - **1 to 32 bytes dynamic payload length with automatic packet handling**
    - **Host interface : 4-pin hardware SPI**
    - **1.9 to 3.6V supply range**

https://infocenter.nordicsemi.com/pdf/nRF24L01P_PS_v1.0.pdf?cp=8_4_0_0

- **NRF24L01+**
  - **Enhanced ShockBurst (ESB) protocol**

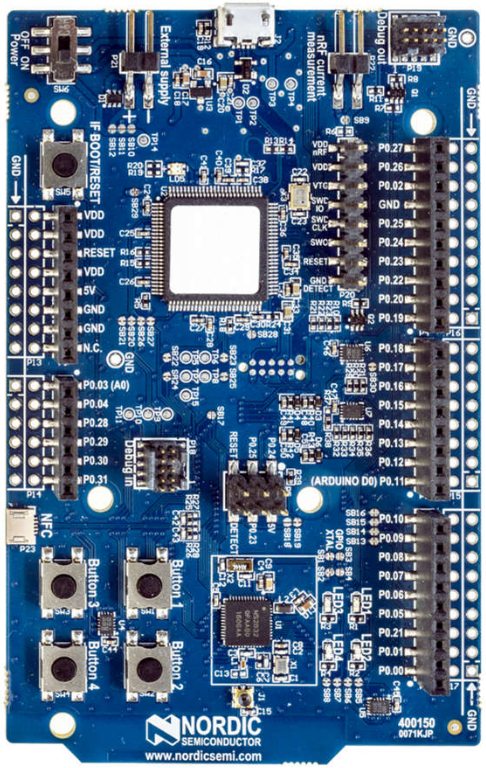




- Other NRF Chips
  - NRF24LU1+ = NRF24L01+ + USB FS + 8051. Found in Microsoft and Logitech keyboards and mice
  - NRF24L01+ and NRF24LU1+ are not recommended for new designs

  - More on this dongle later on

- ❑ **Other NRF Chips**
  - ❑ **NRF52 series (SoC = RF + ARM Cortex M4) are the replacement chips**



NRF52832 DK
40€

	nRF52810	nRF52811	nRF52832	nRF52840
LTE-M, NB-IoT, GPS				
Bluetooth 5.1 - Direction Finding		(✓)		
Bluetooth 5 - 2 Mbps	✓	✓	✓	✓
Bluetooth 5 - Long Range		✓		✓
Bluetooth mesh	(✓)	(✓)	✓	✓
ANT	✓	(✓)	✓	✓
NFC			✓	✓
Thread, Zigbee, 802.15.4		✓		✓
CPU	M4	M4	M4F	M4F
Flash (KB)	192	192	256/512	1024
RAM (KB)	24	24	32/64	256
SPI, TWI, UART, PWM	✓	✓	✓	✓
High Speed SPI				✓
ADC, Comparators	✓	✓	✓	✓
I2S, PDM	PDM	PDM	✓	✓
ARM® TrustZone® Cryptocell				✓
USB				✓
QSPI				✓
GPIO	Up to 32	Up to 32	Up to 32	Up to 48

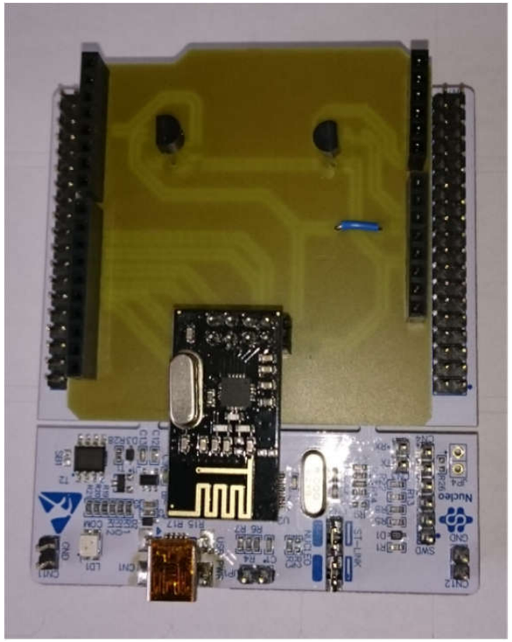
- ❑ **NRF52 series can still use Nordic proprietary protocol (ESB)**

- Where do we find NRF Chips
  - Microsoft and Logitech wireless keyboards and mice
  - Drone remote controllers
  - Sport watches
  - Bike equipment
  - Heart rate monitors…

- **Well known technology**
  - **Lots of resources about hacks (mice, drones etc,)**
  - **Uses packet communication: interesting to fiddle with it in GNU Radio**

➔ Goal of this tutorial: to build a Nordic ESB compliant transceiver with GNU Radio

- **Two STM32 boards fitted with NRF24L01+ breakout board transmitting the room temperature every 0.5s (one at 2485MHz and the other at 2500MHz)**
- **STM32 program + PCB shield (available for those interested)**



Outline

2. Bastille Research gr-nordic OOT module

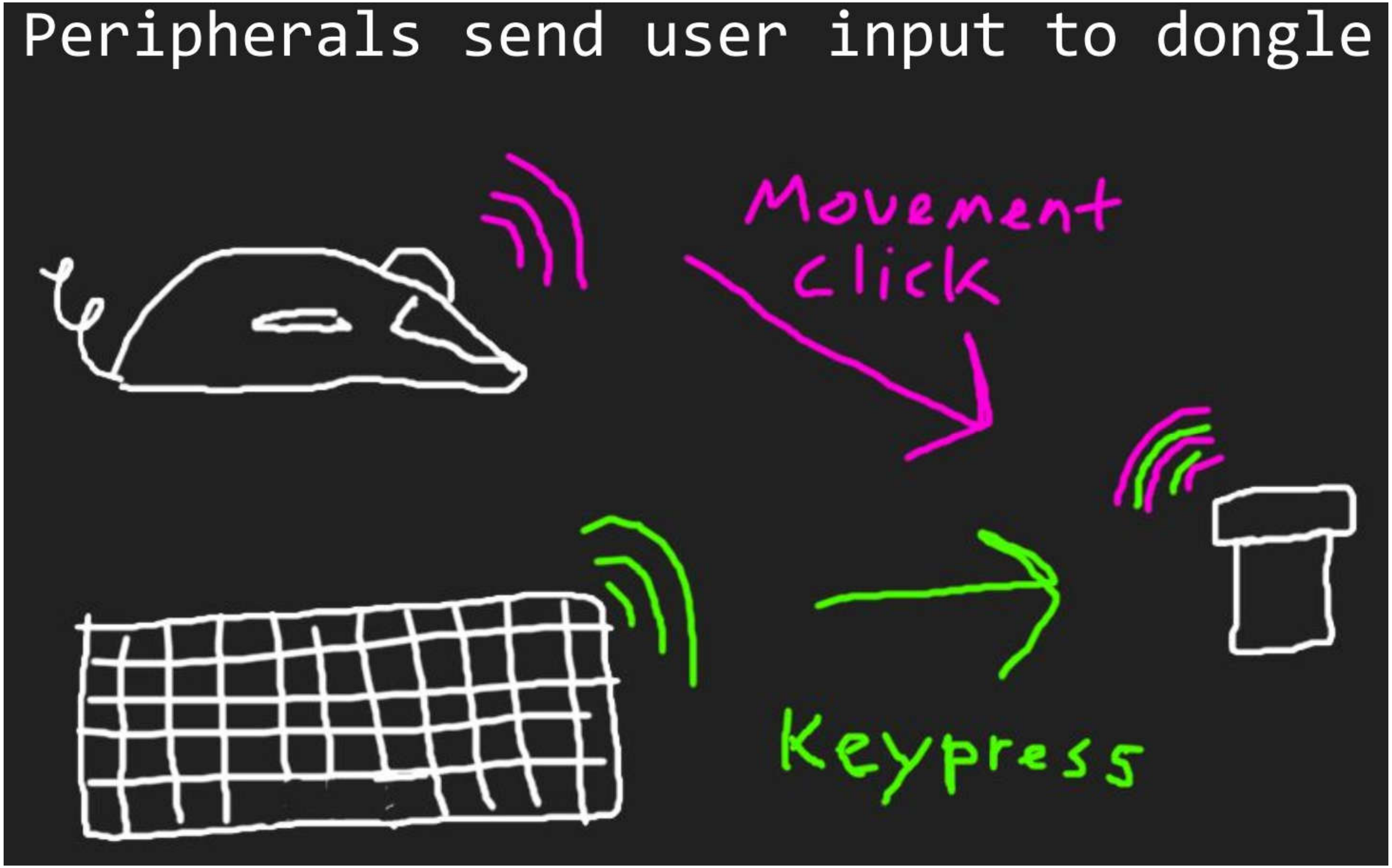
- Do we have to start from nothing?
- Not many up to date implementations
- Some of them use the old message queue way of packet transmission, e.g. : https://github.com/funoverip/gr-cc1111
- I came up with the gr-nordic OOT module from Bastille Research
- Uses GNU Radio message passing interface
- Has been designed to identify vulnerabilities of wireless keyboards and mice
- Presented at DEF CON 24 (2016) hacking convention

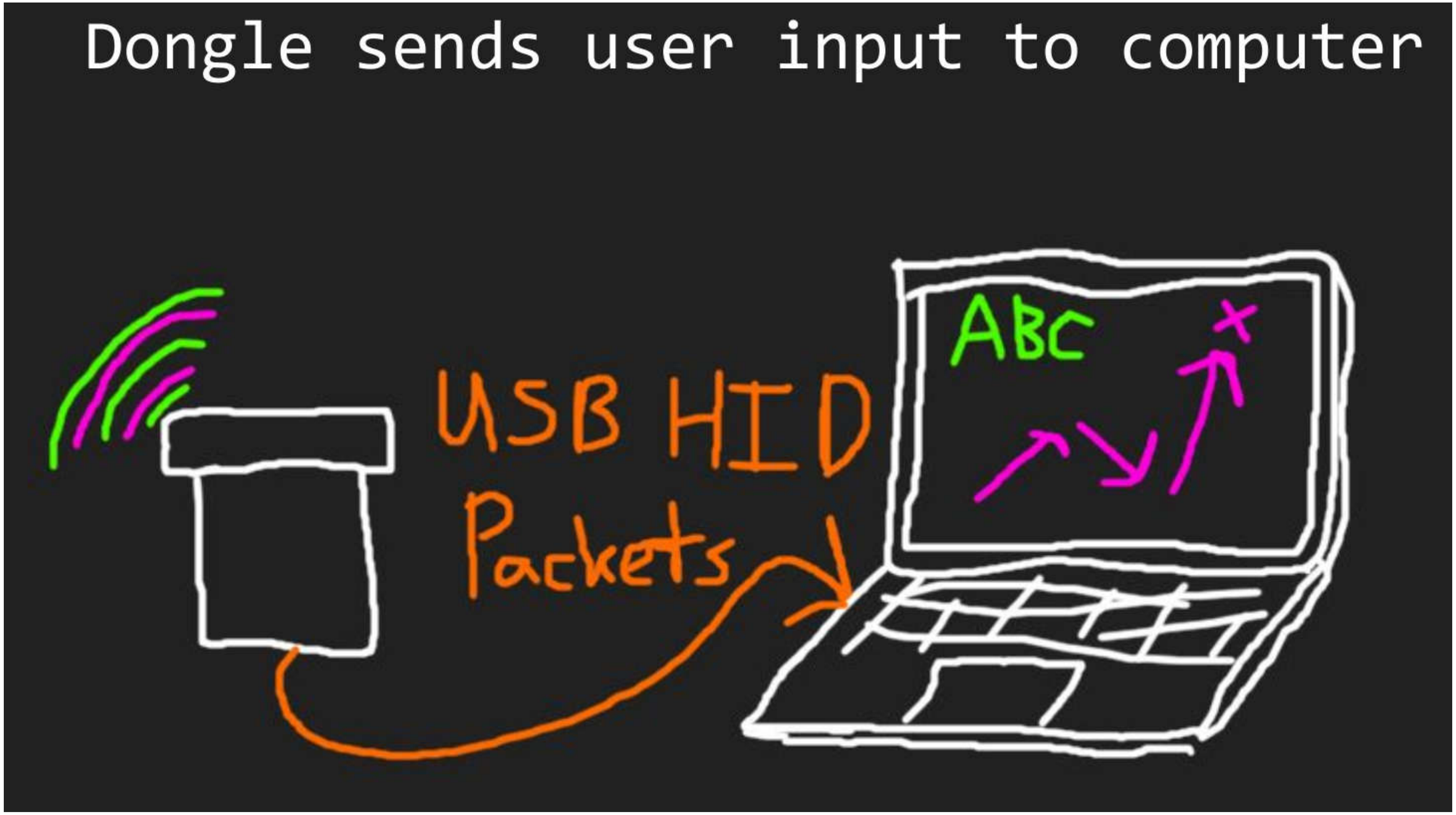
((Mouse|Key)Jack|KeySniffer)

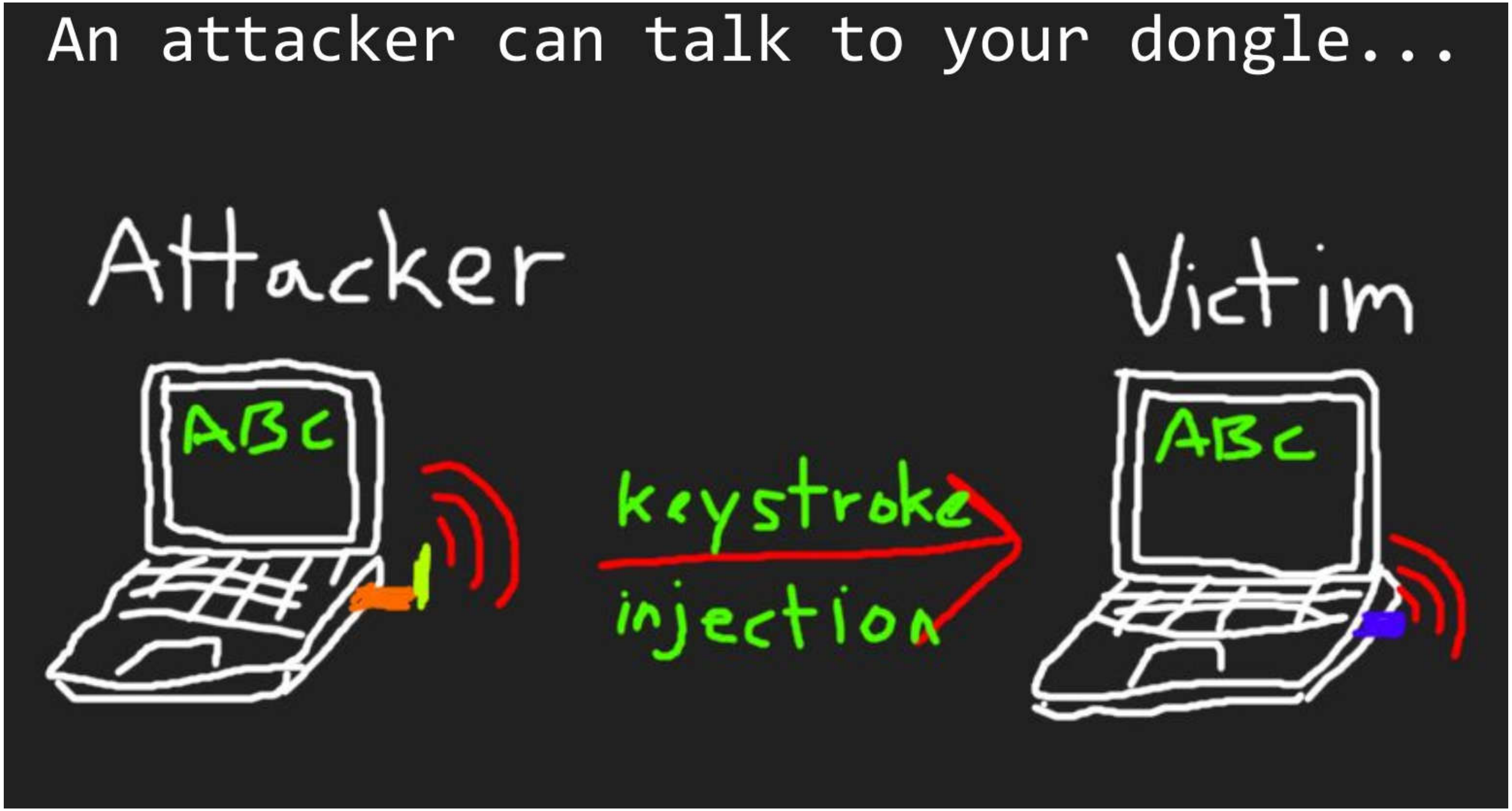
- Wireless mice and keyboards
  - 16 vendors
  - proprietary protocols (non-Bluetooth)
  - 4 families of transceivers
- 16 vulnerabilities
  - keystroke sniffing
  - keystroke injection
  - many are unpatchable

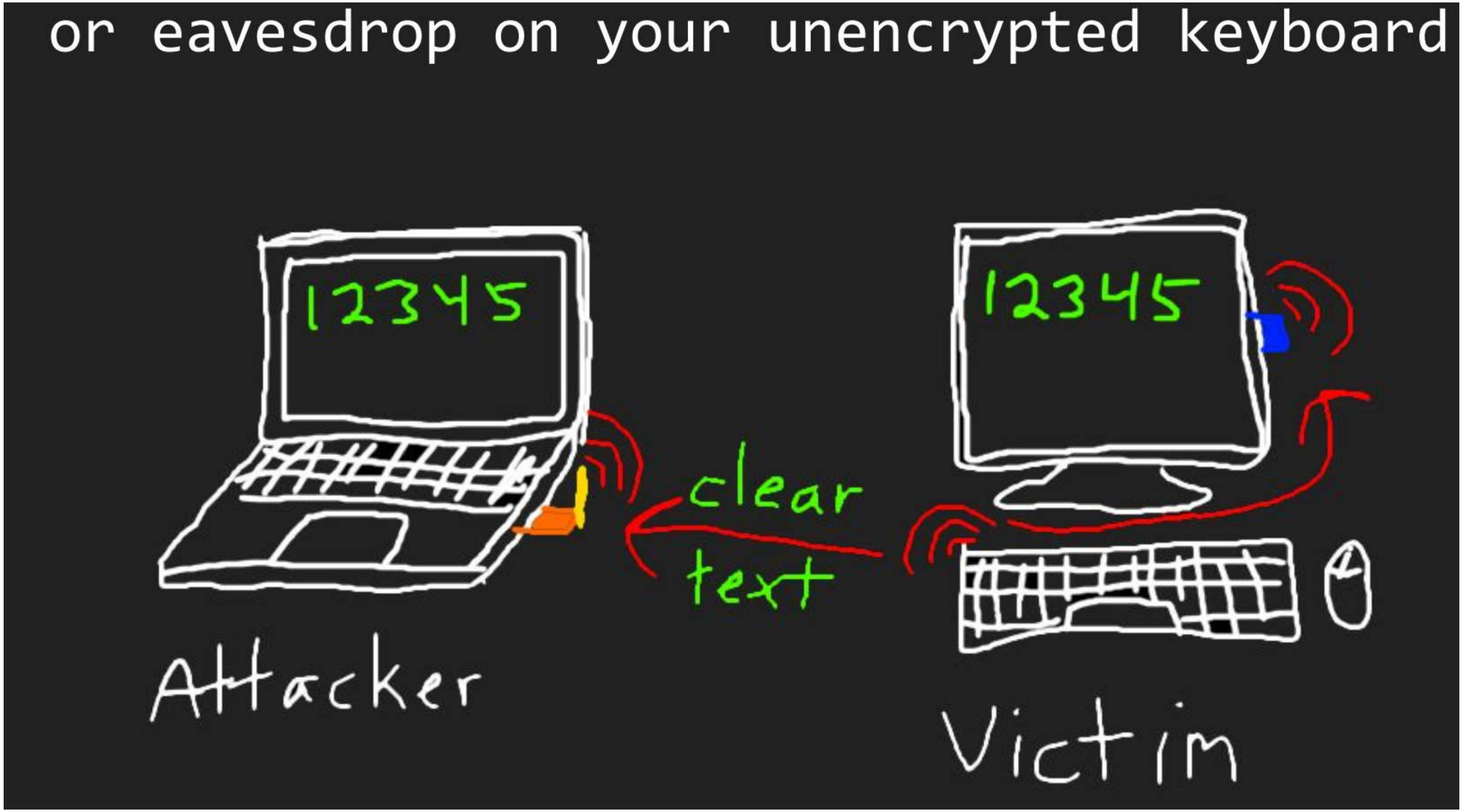
"Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted."

- Logitech (2009)


Peripherals send user input to dongle
Movement
click
Keypress


Dongle sends user input to computer
USB HID
Packets
ABC


An attacker can talk to your dongle...
Attacker
ABC
keystroke
injection
Victim
ABC



- For more details see:

https://github.com/BastilleResearch/mousejack/tree/master/doc/pdf

Outline

1 –

2 –

3 – Building a nrf24L01+ transceiver with GNU Radio

4 –

❑ Let's discover gr-nordic!

➢ Get and build gr-nordic:

```
>> git clone
https://github.com/BastilleResearch/gr-nordic.git
>> cd gr-nordic
>> mkdir build
>> cd build
>> cmake ../
>> make -j2
>> sudo make install
>> sudo ldconfig
```

❑ Two main blocks:
* nordic_tx : input = nordictap_in message output = stream of uint8_t

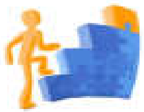
```
nordic_tx_impl::nordic_tx_impl(uint8_t channel_count)
  : gr::sync_block("nordic_tx",
          gr::io_signature::make(0, 0, 0),
          gr::io_signature::make(1, channel_count, sizeof(uint8_t))),
    m_channel_count(channel_count)
{
  // Register nordictap input, which accepts packets to transmit
  message_port_register_in(pmt::intern("nordictap_in"));
  set_msg_handler(pmt::intern("nordictap_in"), boost::bind(&nordic_tx_impl::nordictap_message_handler, this, _1));
}
```

* nordic_rx : input = stream of uint8_t output = nordictap_out message

```
nordic_rx_impl::nordic_rx_impl(uint8_t channel,
                               uint8_t address_length,
                               uint8_t crc_length,
                               uint8_t data_rate)
  : gr::sync_block("nordic_rx",
          gr::io_signature::make(1, 1, sizeof(uint8_t)),
          gr::io_signature::make(0, 0, 0)),
          m_decoded_bits_bytes(42*8 /* buffer sufficient for max ESB frame length */),
          m_crc_length(crc_length),
          m_address_length(address_length),
          m_channel(channel),
          m_data_rate(data_rate)
{
  message_port_register_out(pmt::mp("nordictap_out"));
}
```

➢ **We would like to use these blocks in GRC**

➔ **You can check that this has not been implemented (try it in GRC)**



➔**Modify `nordic_nordic_tx.xml` and `nordic_nordic_rx.xml` accordingly!**

➔**Verify that your implementation works in GRC**

❑ Result

V1

public constructor

```
<?xml version="1.0"?>
<block>
  <name>nordic_tx</name>
  <key>nordic_nordic_tx</key>
  <category>[Nordic]</category>
  <import>import nordic</import>
  <make>nordic.nordic_tx($channel_count)</make>

 <param>
   <name>Number of channels</name>
   <key>channel_count</key>
   <value>1</value>
   <type>int</type>
 </param>

 <sink>
    <name>nordictap_in</name>
    <type>message</type>
 </sink>

 <source>
    <name>out</name>
     <type>byte</type>
    <nports>$channel_count</nports>
 </source>
</block>
```

❑ Result

V1

```
<?xml version="1.0"?>
<block>
  <name>nordic_rx</name>
  <key>nordic_nordic_rx</key>
  <category>[Nordic]</category>
  <import>import nordic</import>
  <make>nordic.nordic_rx($channel, $address_length, $crc_length, $data_rate)</make>
  <param>
    <name>RF Channel</name>
    <key>channel</key>
    <value>85</value>
    <type>int</type>
  </param>
 <param>
    <name>Address Length</name>
    <key>address_length</key>
    <value>5</value>
    <type>int</type>
  </param>
  <param>
    <name>CRC Length</name>
    <key>crc_length</key>
    <value>2</value>
    <type>int</type>
  </param>
  <param>
    <name>Data Rate</name>
    <key>data_rate</key>
    <value>2</value>
    <type>int</type>
  </param>
  <sink>
    <name>in</name>
    <type>byte</type>
  </sink>
  <source>
    <name>nordictap_out</name>
    <type>message</type>
  </source>
</block>
```

- **Let's build a NRF24L01+ receiver with gr-nordic and GRC**

➔ **We have two STM32 boards fitted with NRF24L01+ modules as test transmitters**

- **They send an Enhanced ShockBurst (ESB) Packet every 0.5s**
- **Payload is the room temperature. E.g.: 20.0°C**
- **ESB packet:**

Address length	**5 bytes**
Address	**0x55 0x55 0x55 0x55 0x55**
Payload size	**8 bytes**
Payload	**0x20 0x20 0x20 0x20 + Temp (XX.X)**
CRC length	**2 bytes**
Data Rate	**2Mbits/s**
RF channel	**85 (2485MHz)**

- **Let us first have a try with the existing Python example nordic_receiver.py**

- **gr-nordic is said to be SDR agnostic. It uses gr-osmosdr which does not work with the ADALM-PLUTO.**

➔ modify nordic_receiver.py accordingly

➔ Study nordic_receiver.py and identify the relevant blocks and their connections

❑ Modifications and the blocks

```
from gnuradio import iio
```

```
self.pluto_source = iio.pluto_source('', int(self.freq), int(self.sample_rate),
                    int(2e6), 0x8000, True, True, True, "manual", 64.0, '', True)

# Receive chain
dr = 0
if args.data_rate == 1e6:
    dr = 1
elif args.data_rate == 2e6:
    dr = 2
self.rx = nordic.nordic_rx(
    args.channel, args.address_length, args.crc_length, dr)
self.gfsk_demod = digital.gfsk_demod(
    samples_per_symbol=args.samples_per_symbol)
self.lpf = filter.fir_filter_ccf(
    1, firdes.low_pass_2(1, self.sample_rate, self.symbol_rate / 2, 100e3, 50))
self.connect(self.pluto_source, self.lpf)
self.connect(self.lpf, self.gfsk_demod)
self.connect(self.gfsk_demod, self.rx)

# Handle incoming packets
self.nordictap_printer = nordictap_printer()
self.msg_connect(
    self.rx, "nordictap_out", self.nordictap_printer, "nordictap_in")
```

V2

Blocks: Pluto_source, lpf, gfsk_demod, nordic_rx and nordictap_printer

➢ Integration of the nordictap_printer block

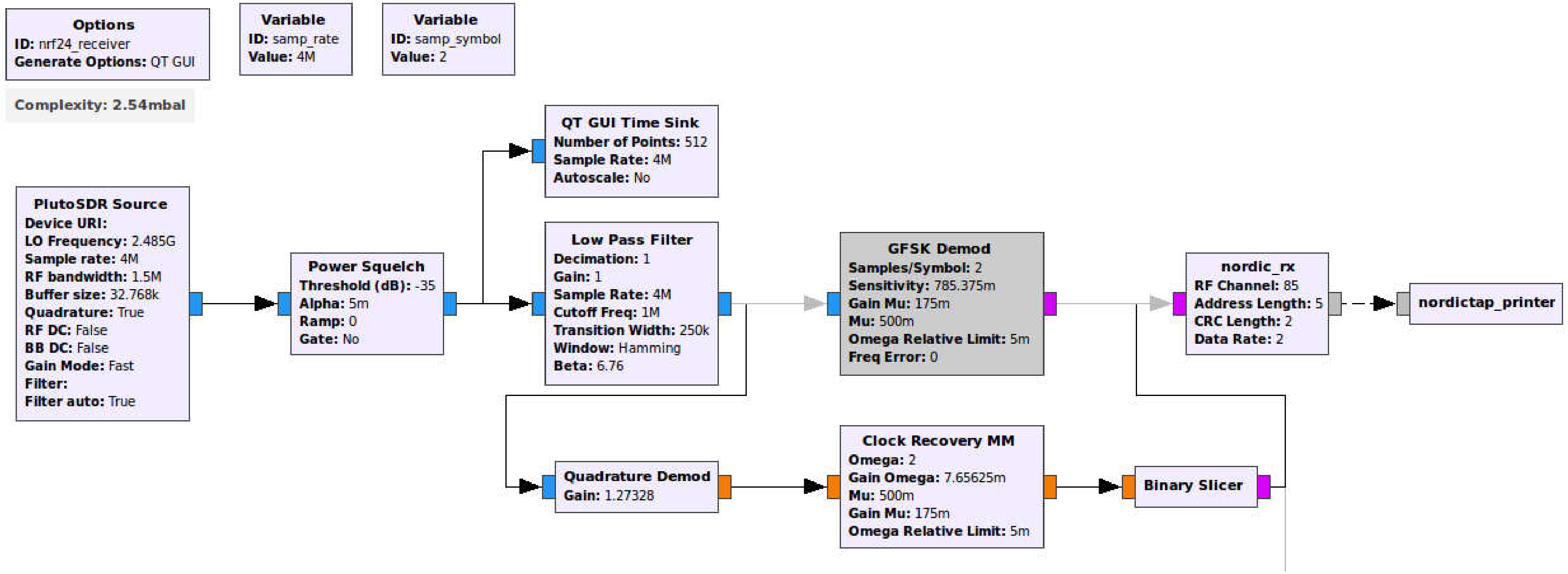
V3

- ➔Create a python file in gr-nordic/python called nordic_blocks.py, copy the code of nordictap_printer found in nordic_receiver.py and paste it into nordic_blocks.py
- ➔Add nordic_blocks.py to the module by modifying accordingly CMakeLists.py and __init__.py
- ➔Create a GRC xml file for the new nordictap_printer block
- ➔Rebuild the module

- **Create a GRC Nordic receiver flowgraph**

→ Using the new blocks and appropriate GNU Radio blocks design a GRC flowgraph receiver
→ Visualize the incoming packets. Add a power squelch block and synchronize the time sink block with the squelch_sob tag
→ Study gfsk.py located in /usr/lib/python2.7/dist-packages/gnuradio/digital and in particular the gfsk_mod hier block. Expose the blocks that compose gfsk_mod

❑ This is my receiver:



V4

- **GFSK modulation in a nutshell**

- **Is defined by:**

$$s(t) = \sqrt{\frac{2E}{T}} \cos(2\pi f_c t + \phi(t, \alpha))$$

- **E is the bit energy, T is the symbol duration and fc is the carrier frequency.**

- **Transmitted information is contained in the phase (angle) term:**

$$\phi(t, \alpha) = 2\pi h \sum_{i=-\infty}^{\infty} \alpha_i q(t - iT)$$

- **where α_i is the message signal (here +1/-1) and:**

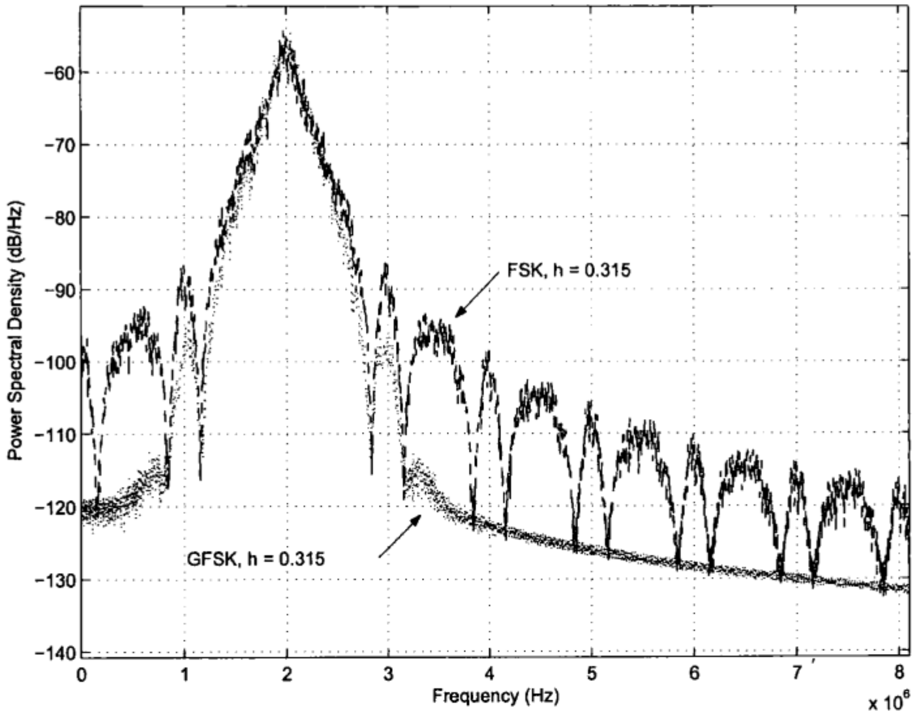
$$q(t) = \int_{-\infty}^{t} g(\tau) \mathrm{d}\tau$$

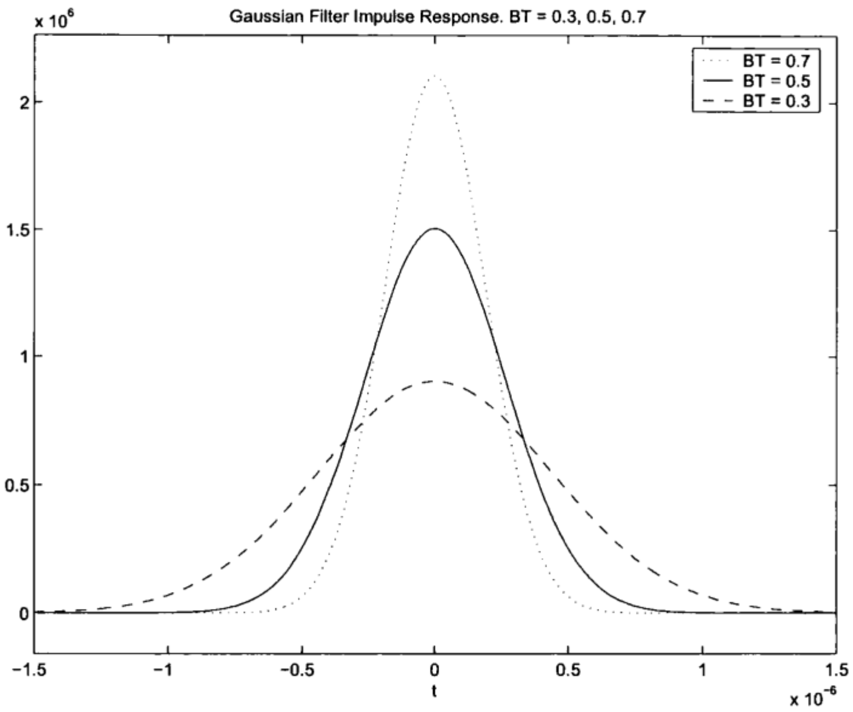
- **where q(t) describes the shape of the phase transitions whose magnitude is proportional to the modulation index h**

- **We have:**

$$h = (f_1 - f_0)T$$
$$= 2 f_d T \text{ where}$$
$$f_d = (f_1 - f_c) = (f_c - f_0)$$

- **Pulse shaping**

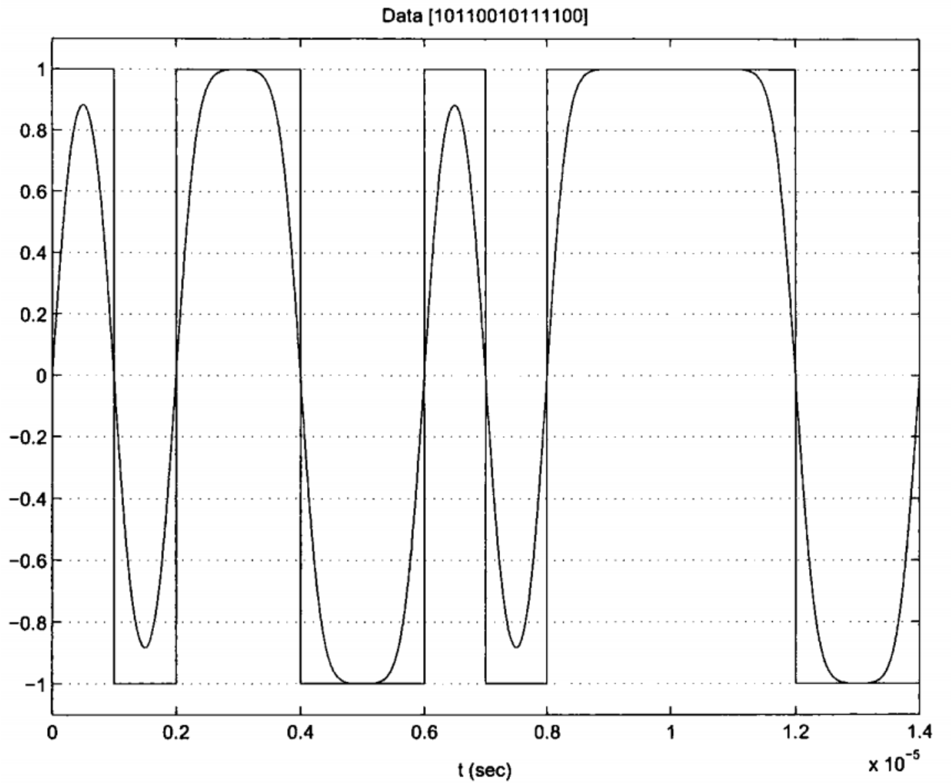




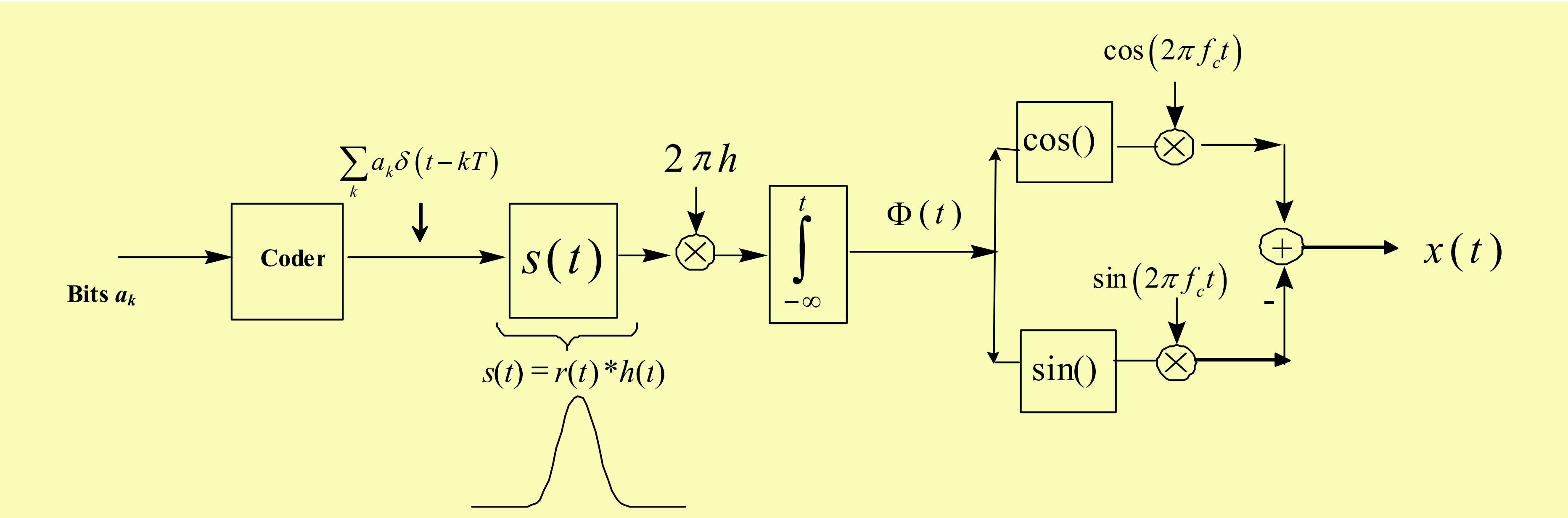
- Gaussian filters have an impulse response h(t) given by:

$$h(t) = B\sqrt{\frac{2\pi}{\ln 2}} \exp \frac{-2\pi^2 B^2 t}{\ln 2}$$

- B is the filter's -3dB bandwidth. Caracterised by BT product, e.g. BT = 0.5 for Bluetooth.

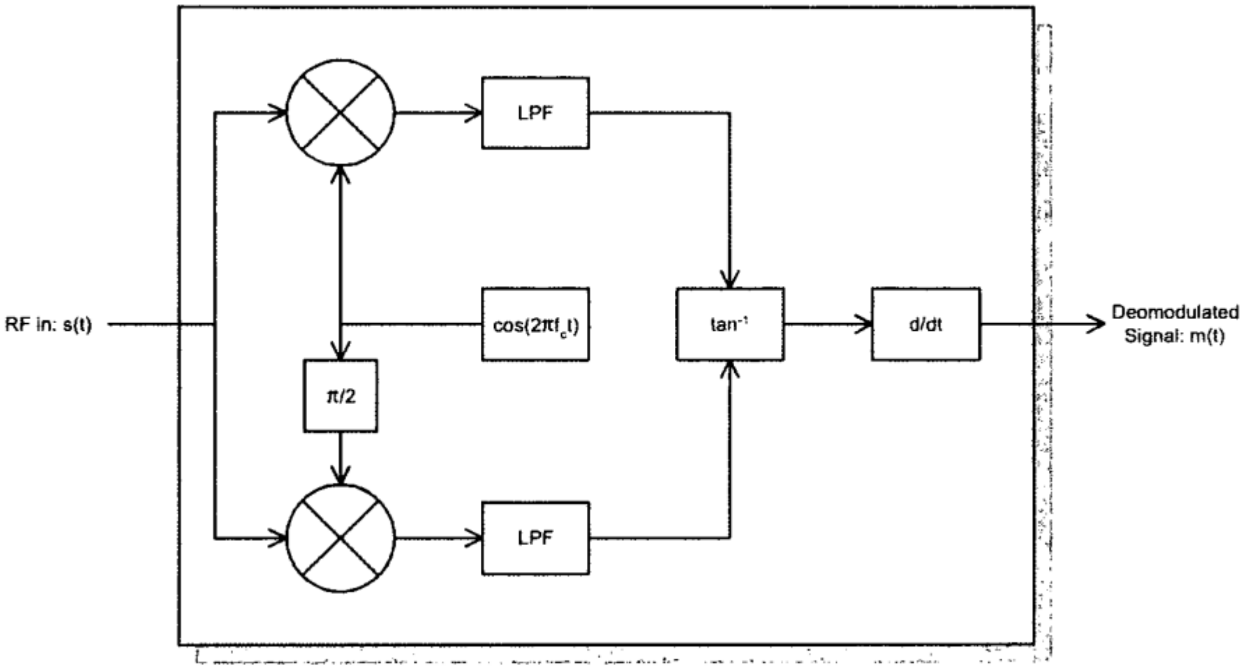


❑ GFSK modulator



➔ Have a look in gfsk.py to see how it is done in GNU Radio!

❑ GFSK demodulator (phase shift discriminator)



➔ **We have:**

$$s_I(t) = \frac{A(t)}{2}[\cos(4\pi f_c t + 2\pi h \int_{-\infty}^{t} m(\tau)\mathrm{d}\tau) - \cos(2\pi h \int_{-\infty}^{t} m(\tau)\mathrm{d}\tau)]$$

$$s_Q(t) = \frac{A(t)}{2}[\sin(4\pi f_c t + 2\pi h \int_{-\infty}^{t} m(\tau)\mathrm{d}\tau) - \sin(2\pi h \int_{-\infty}^{t} m(\tau)\mathrm{d}\tau)]$$

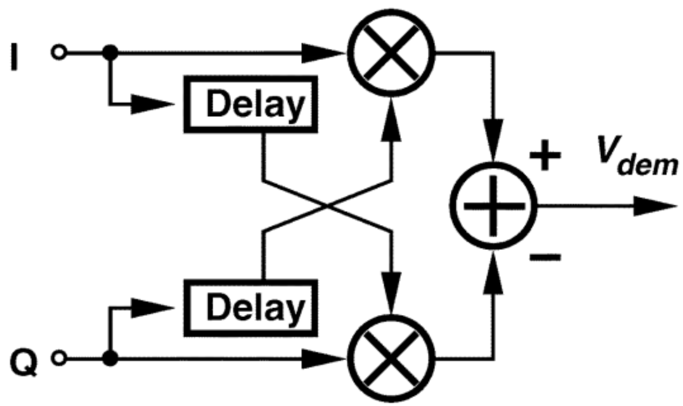
- **After low pass filtering and arctan operation we get:**

$$\phi(t) = h \int_{-\infty}^{t} m(\tau)\mathrm{d}\tau$$

- **The differentiation operation leads to:**

$$m(t) = \frac{\mathrm{d}(\phi(t))}{\mathrm{d}t}$$

➔ Have a look in gfsk.py to see how it is done in GNU Radio! Rather something like that?



❑ Something working behind the scene: Wireshark to decode Nordic ESB packets

>> sudo apt-get install wireshark
>> sudo groupadd wireshark
>> sudo usermod -a -G wireshark #yourusername#
>> sudo chgrp wireshark /usr/bin/dumpcap
>> sudo chmod 750 /usr/bin/dumpcap
>> sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/dumpcap
>> sudo getcap /usr/bin/dumpcap
>> cd ~/gr-nordic/
>> sudo wireshark -X lua_script:wireshark/nordic_dissector.lua -i lo -k -f udp

❑ GNU Radio message passing interface and PMTs

➔ In the case of packet transmission we are not interested in data streaming. We would like to transmit asynchronous messages.

➔ In GNU Radio there are two mecanisms to pass messages:

- ➔ Synchronously to a data stream, using stream tags
- ➔ Asynchronously, using the message passing interface

➔ From a programming perspective a message can be represented as a data type.

➔ In Python, this is not a problem, since it is weakly typed

➔ C++ on the other hand is strongly typed, and it is not possible to create a variable without knowing its type

➔ In GNU Radio we need to exchange the same data objects between Python and C++ ➔ Polymorphic Types or PMTs

- **Hence messages are PMTs and can contain anything**
  - **Commonly vectors of bytes for PDUs**
  - **Or a dictionary (Key: value pair)**

- **We need to understand a minimum how PMTs work. Let’s have a look here:**

https://wiki.gnuradio.org/index.php/Guided_Tutorial_Programming_Topics#5.1_Polymorphic_Types_.28PMT.29

➢ Let's build a transmitter

➔ The nordic_transmitter block requires a message input block
➔ This message has to be a PMT
➔ The PMT will be made up of these fields:

channel index = 0
channel = 85
header.datarate = 2
header.address_length = 5
header.payload_length = 8
header.sequence_number = 0
header.no_ack = 0
header.crc_length = 2
Address = 0x55 0x55 0x55 0x55 0x55
payload = 0x20 0x20 0x20 0x20 0x32 0x30 0x2E 0x30

➔ We would like to send this packet every 0.5s (same as the STM32 + NRF24L01+ breakout board)

❑ Let's build a transmitter

➔ To validate the transceiver chain, build a TX/RX loop model with GRC.

➔ Some tips:

- Use a Message Strobe block as input to the nordic_tx block
- Take care of the PMT (look at line 79 of nordic_tx_impl.cc)!

➔ Your model is complete but it does not work…

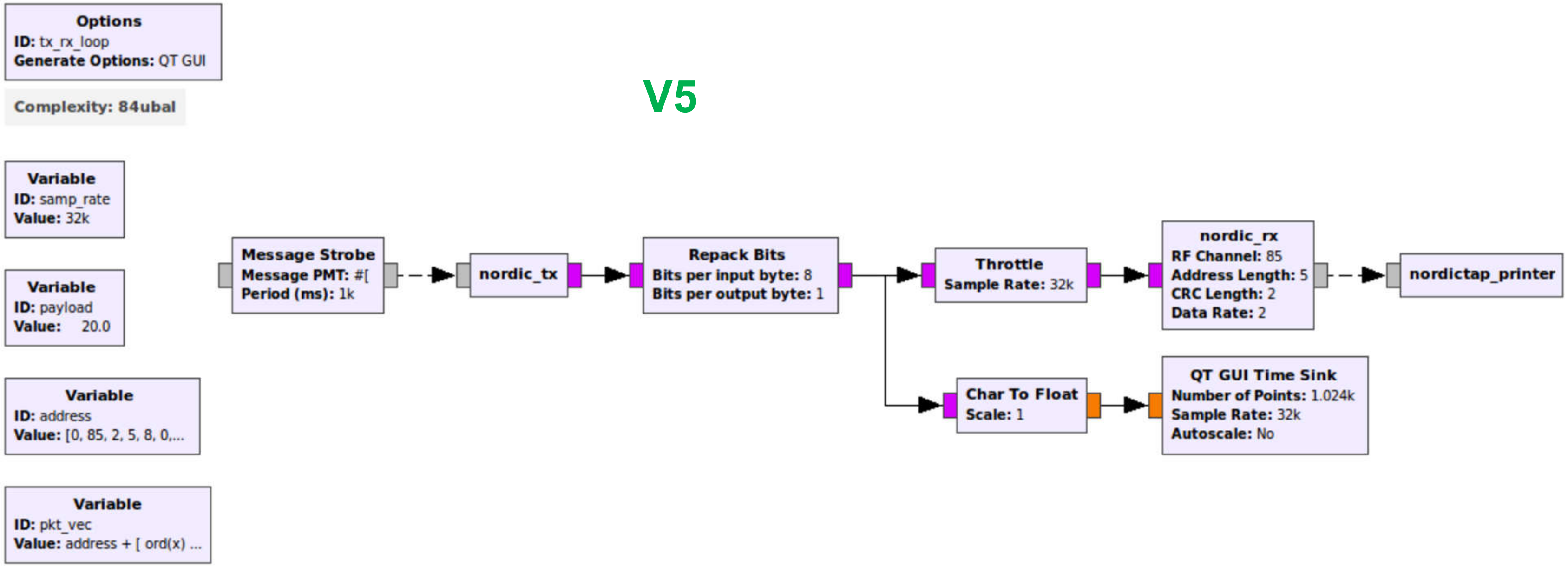
- Find the bug in nordic_tx_impl.cc!

➔ OK it's working now: let's build a transmitter model.

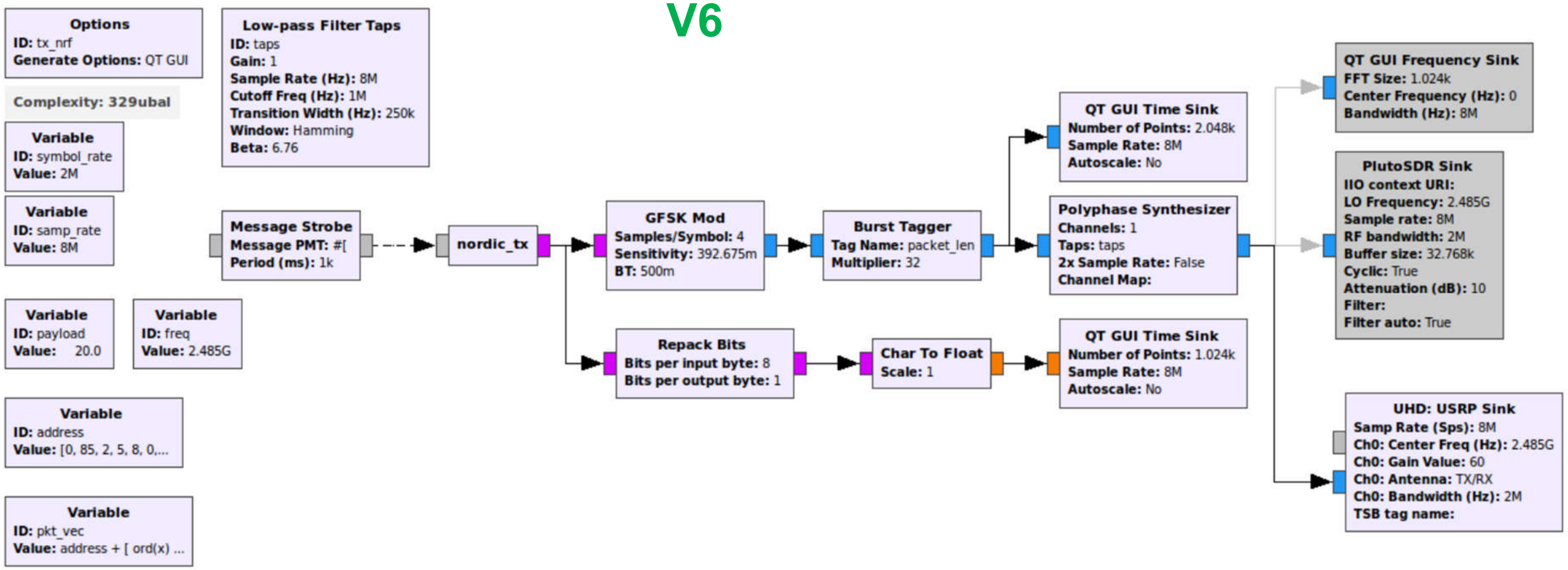
➔ Check your model with your neighbour as a receiver

- Here are my models:

V5



V6



- **Adding a length tag to the nordic_tx block**

➔ **This can be useful to synchronise vizualisation blocks**

➔ **Edit nordic_tx_impl.cc and add a tag named packet_len whose value is the length of the produced packet**

➔ **Tips:**

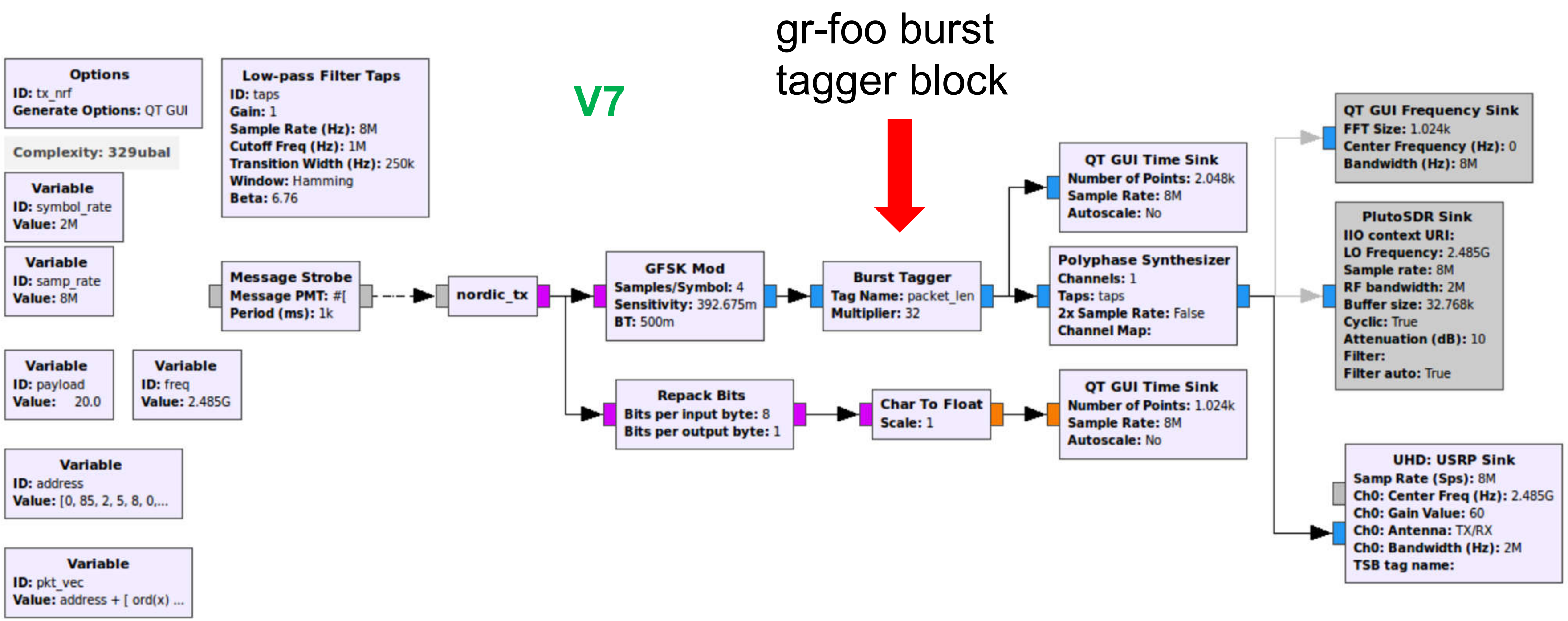
  ➔ **Have a look here:**

https://wiki.gnuradio.org/index.php/Guided_Tutorial_Programming_Topics#5.2.1_Adding_tags_to_the_QPSK_demodulator

➔ **Our packet_len should increase because of the byte to bit conversion and the upsampling rate. How can we do that?**

➔ **Get, build and install gr-foo (maint-3.7 branch). You will find a nice block called burst tagger!**

- **Here is my model:**



- **Creating our own nordictap_transmitter block**

➔ part of this block in nordic_channelized_transmitter.py

➔Does not meet our needs for GRC

➔We want our block to be triggered by a message strobe block

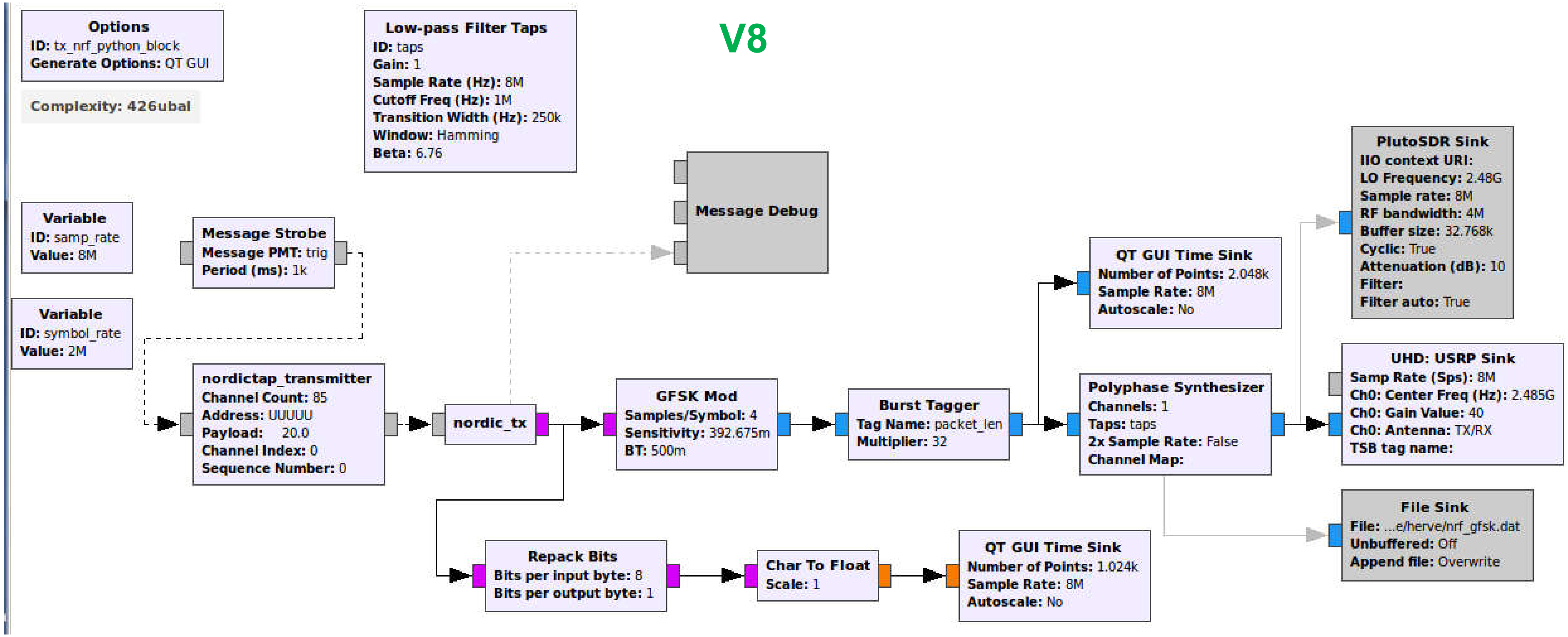
➔User can enter easily packet data from GRC block

➔Write this block and add it to nordic_blocks.py

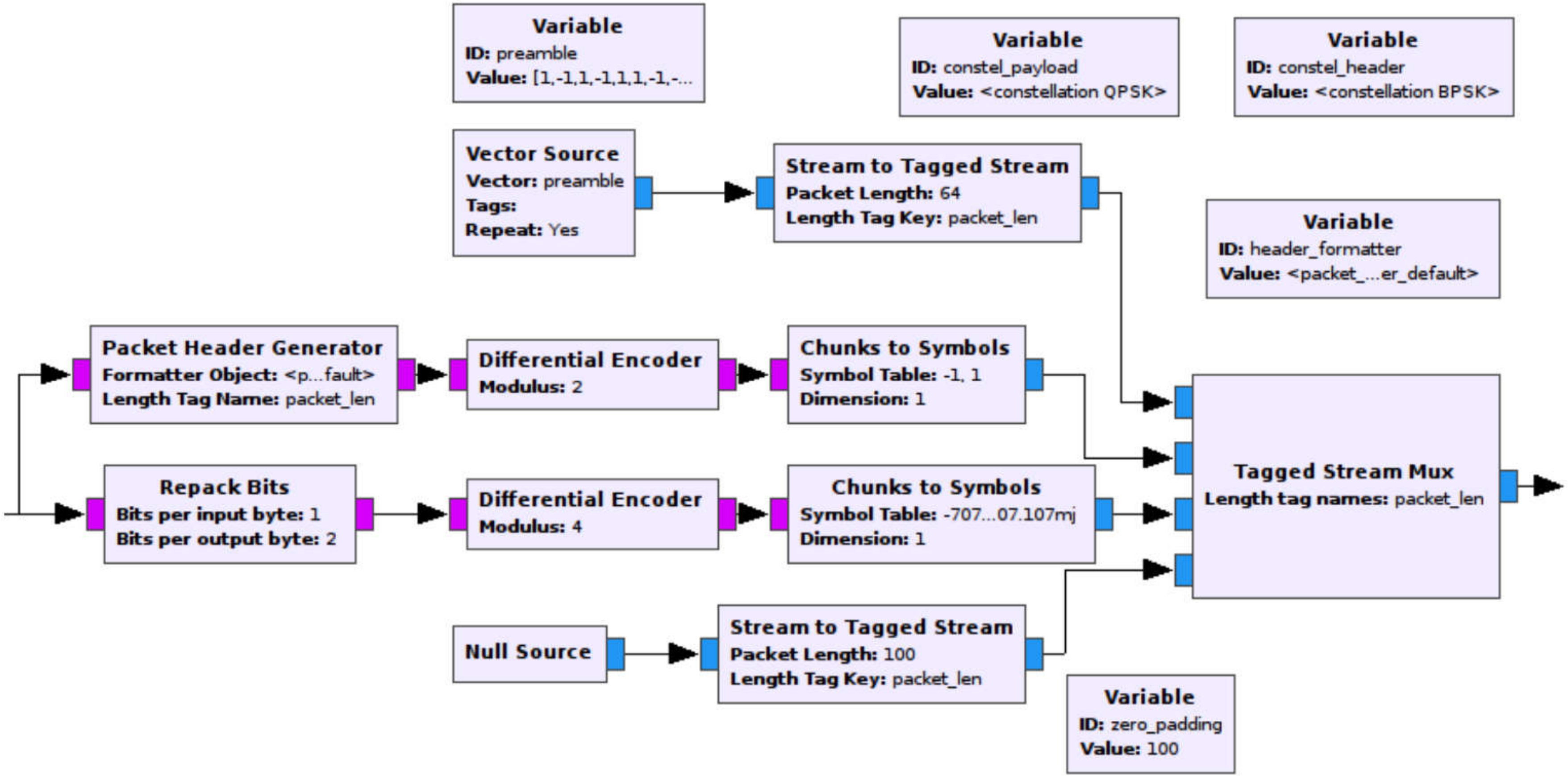
➔Create the associated GRC nordic_nordictap_transmitter.xml

➔Add your block to the GRC transmitter model and check that everything works

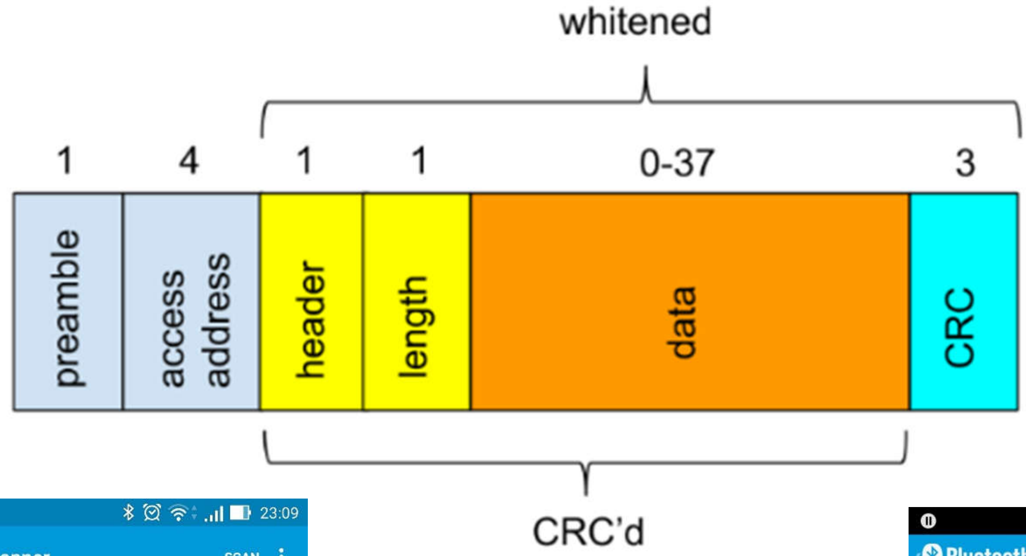
❑ Here is my model:

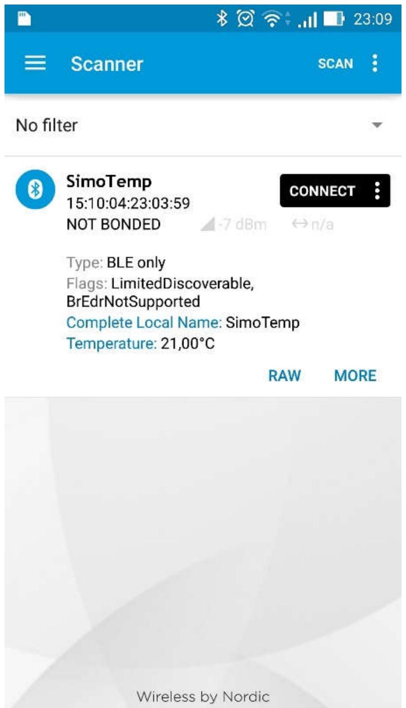


- ❑ **Can we build the TX with existing GNU Radio blocks?**



❑ Homework: build a BLE Advertiser with gr-nordic blocks





Outline

1 –

2 –

3 –

4 – Conclusion

- ❑ Logitech research firmware demo
- ❑ How to increase your knowledge in GNU Radio: the electronics engineer's point of view
- ❑ GNU Radio Packet Communications
- ❑ Several OOT implementations available:
  - gr-burst (Tim O'Shea)
  - gr-eventstream (Tim O'Shea) → example fsk mod/demod or psk mod/demod
  - Digital communication point of view
  - gr-packetizer (Thomas Verelst)

THANK YOU FOR YOUR ATTENTION!

References

[1] https://wiki.gnuradio.org/index.php/Tutorials