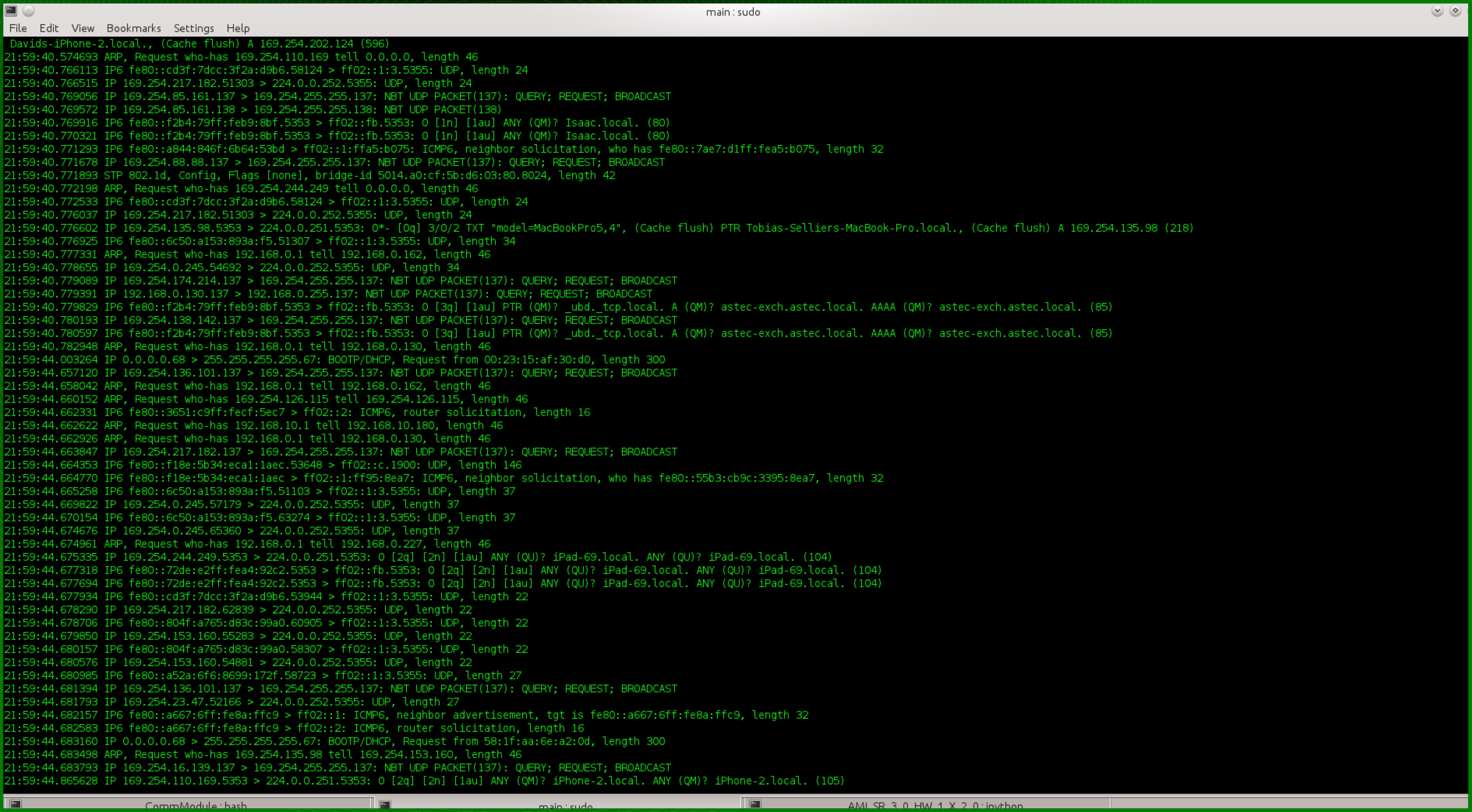
# 0x0000




"We are not as strong as we think we are"

- Rich Mullins

# <GHz or bust!

leveraging the power of the
chipcon 1111
(and RFCAT)

# 0x1000 – intro to <GHz

- FCC Rules(title 47) parts 15 and 18 allocate and govern parts of the RF spectrum for unlicensed ISM in the US (US adaptation of the ITU-R 5.138, 5.150, and 5.280 rules)
  - Industrial – power grid stuff and more!
  - Science -
  - Medical – insulin pumps and the like
- US ISM bands:
  - 300 :
  - 433 : 433.050 – 434.790 MHz
  - 915 : 902.000 – 928.000 MHz
- Popular European ISM band:
  - 868 : 863.000 – 870.000 MHz
- Other ISM includes 2.4 GHz and 5.8 GHz
  - cc2531.... hmmm... maybe another toy/talk?

# 0x1010 – what is <GHz? what plays there?

- Industry, Science, Medical bands, US and EU
- Cell phones
- Cordless Phones

- Personal Two-Way Radios

- Car Remotes

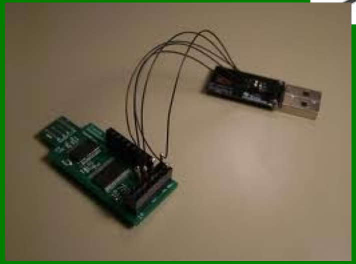
- Pink IM-ME Girl Toys!
- TI Chronos Watches


- Medical Devices (particularly 401-402MHz, 402-405MHz, 405-406MHz)
- Power Meters
- custom-made devices
- Old TV Broadcast
- much, much more...
  - <INSERT PICTURES OF STUFF>


# 0x1020 – how do we play with it?

- cc1110/cc1111 do 300-348MHz, 391-464MHz, 782-928MHz
  - and more...
- RFCAT uses the CC111x on some common dongles


  - Chronos dongle (sold with every TI Chonos watch)
  - “Don's Dongles”, aka TI CC1111EMK
  - IMME (currently limited to sniffer/detection firmware)
- but there are some catches
  - rf comms configuration?
  - channel hopping sequence?
  - bluetooth and DSSS? (not hap'nin)

# 0x1030 – why do i care!?

- the inner rf geek in all of us
- your security research may require that you consider comms with a wireless device
- your organization may **have** 900MHz devices that should be protected!

# 0x2000 - intro to the cc1111 radio (the cc1100 core)

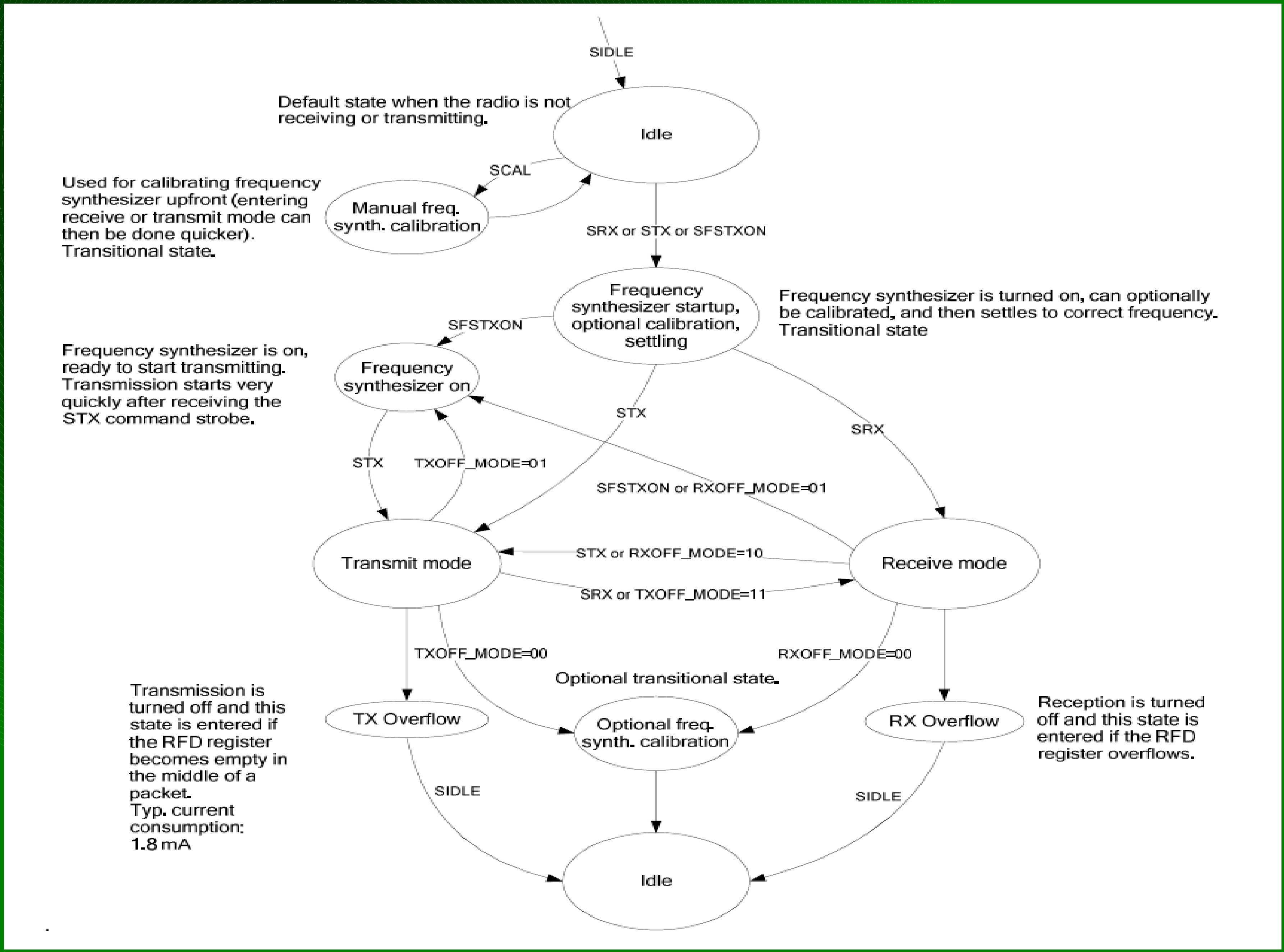
- mcu
- radio state engine
- radio configuration
- usb
- timers
- dma

# 0x2010 – cc1111 mcu

- modified 8051 core
  - 8-bit mcu
  - single-tick instructions
  - 256 bytes of iram
  - 4kb of xram
  - XDATA includes all code, iram, xram
  - execution happens anywhere :)
- register access to radio, dma, crypto, usb, timers, adc
- registers are simply memory locations is the XDATA address space

# 0x2020 – cc1111 radio state engine

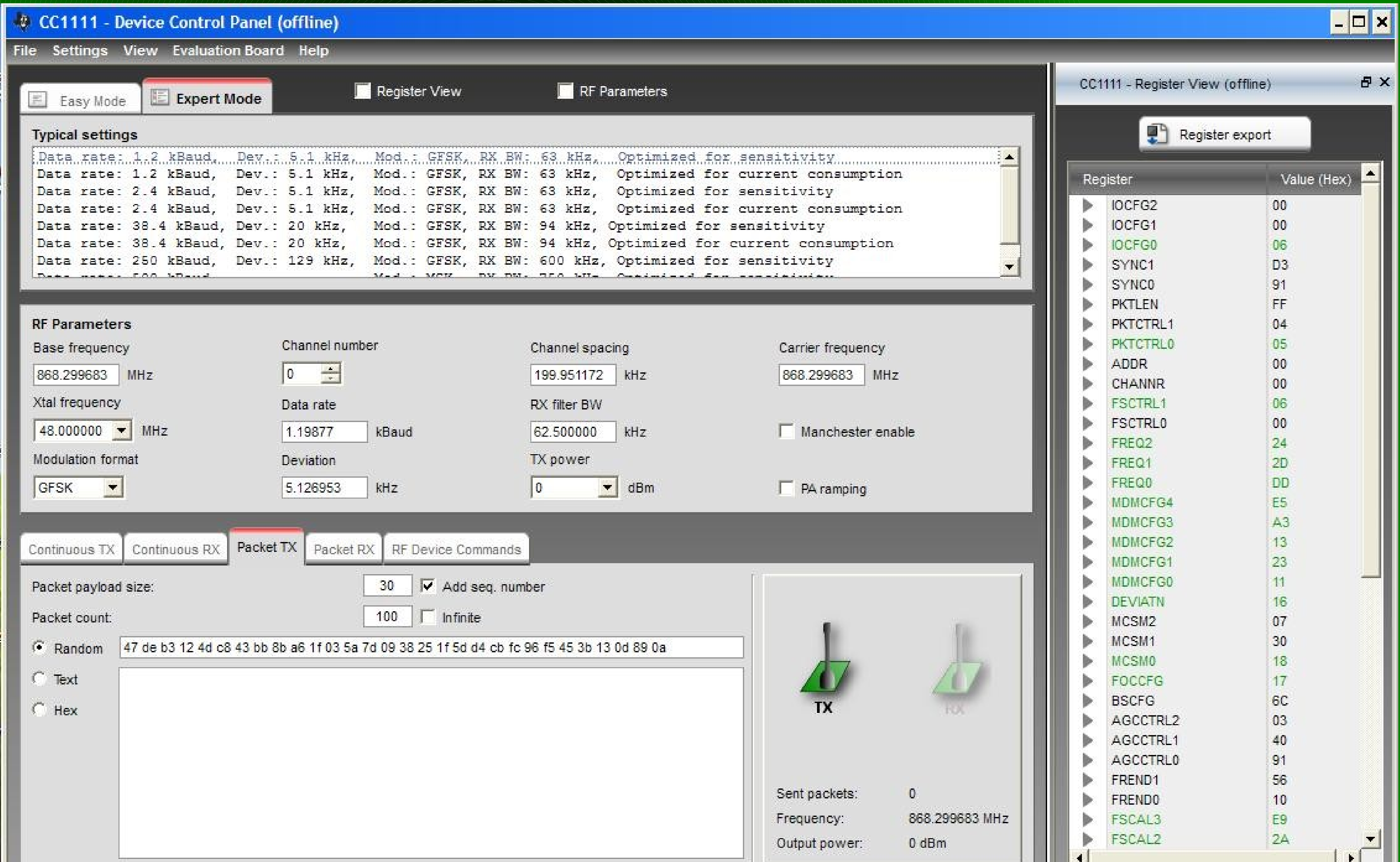
- •IDLE
- •CAL
- •FSTXON
- •RX
- •TX



# 0x2030 – cc1111 radio configuration

- configuring the radio is done through updating a set of 1-byte registers in varying bit-size fields
  - MDMCFG4 – MDMCFG0 – modem control
  - PKTCTRL1, PKTCTRL0 – packet control
  - FSCTRL1, FSCTRL0 – frequency synth control
  - FREND1, FREND0 – front end control
  - FREQ2, FREQ1, FREQ0 – base frequency
  - MCSM1, MCSM0 – radio state machine
  - SYNC1, SYNC0 – SYNC word, or the SFD
  - CHANNR, ADDR – channel and address
  - AGCCTRL2, AGCCTRL1, AGCCTRL0

# 0x2040 - Smart RF Studio (ftw)



# 0x2050 – cc1111 radio notes

- Data Rate, Bandwidth, and Intermediate Frequency and Freq-Deviation depend on each other

- put the radio in IDLE state before configuring
- put the radio in IDLE state before configuring
- put the radio in IDLE state before configuring

- STROBE  (SIDLE, STX, SRX, SCAL...)
  - then wait for the MARCSTATE == MARC_STATE_whatever
- CCA impacts entering TX state from RX
  - but not from IDLE state

# 0x2100 – usb

- usb is a world unto itself, with a massive standard and substandards (gg: usb-in-a-nutshell)
- cc1111's usb controller is accessed using:
  - registers for config/control of usb
  - registers indicating usb events that occur
  - endpoint-specific FIFO buffers
    - messages go there before sending to host
    - messages arrive there from host
  - usb “descriptors” as necessary by spec
    - host uses these to query the device
- our firmware provides all this and more

# 0x2110 – usb for devs

- application.c provides the template for new apps
  - copy it and make
- cc1111usb.c provides usb descriptors and framework
- txdata(buffer, length) to send data IN to host
- registerCbEP5OUT() to register a callback function to handle data OUT from host
  - data is in ep5iobuf[]
- follow the example, luke!

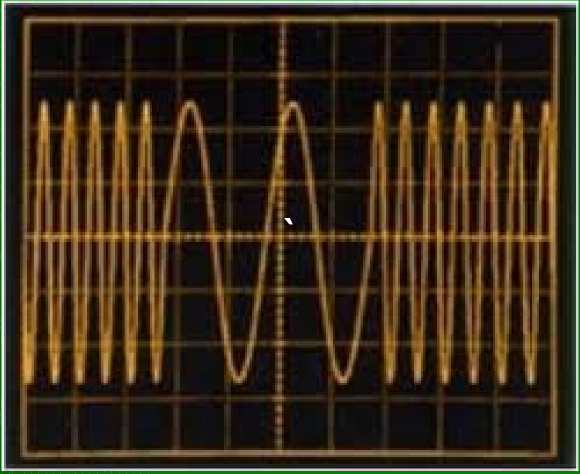
# 0x3000 – what radio things do we want to know!?

- frequencies
- modulation (2FSK/GFSK, MSK, ASK/OOK, other)
- intermediate frequency (IF)
- baud rate
- channel width/spacing/hopping?
- bandwidth filter
- sync words / bit-sync
- variable length/fixed length packets
- crc
- data whitening?
- encoding (manchester, fec, enc, etc...)

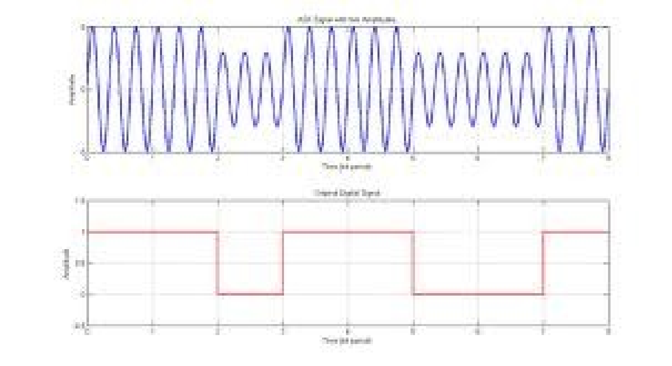
# 0x3010 – frequencies

- 315MHz – car fobs
- 433MHz – medical devices, EU loves this range
- 868MHz – EU loves this range too
- 915MHz – NA stuff of all sorts (power meters, insulin pumps, industrial plant equipment, industrial backhauls)
- 2.4GHz – 802.11/wifi, 802.15.4/zigbee/6lowpan, bluetooth
- 5.8GHz – cordless phones
- FREQ2, FREQ1, FREQ0

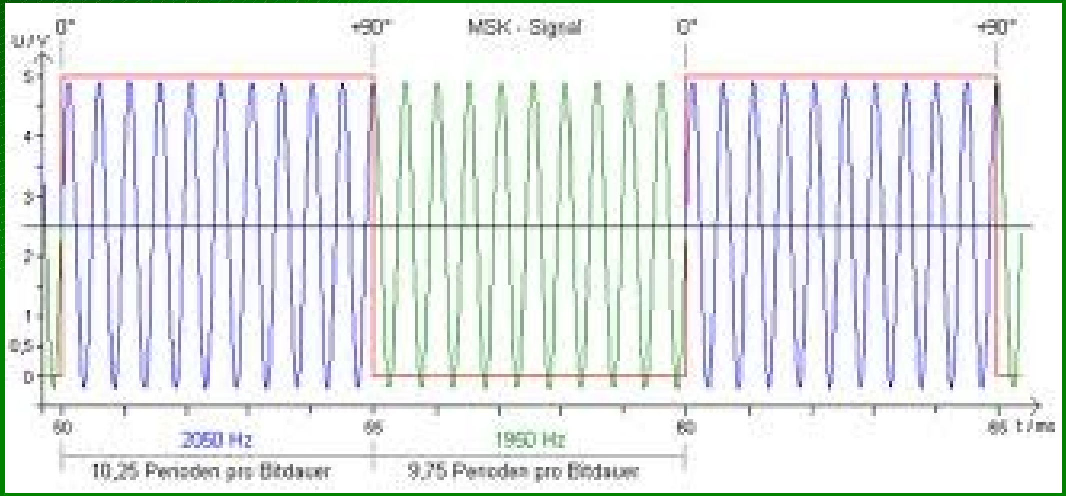
0x3020 – modulations



- 2FSK/GFSK – Frequency Shift Key
  - (digital FM)
  - cordless phones (DECT/CT2)
- ASK/OOK – Amplitude Shift Key
  - (digital AM)
  - morse-code, car-remotes, etc...



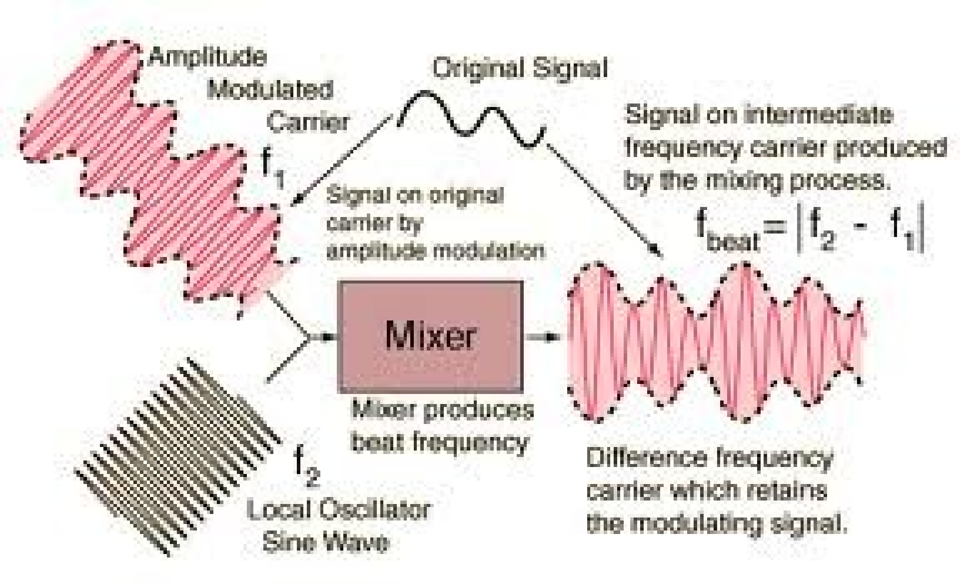
- MSK – Minimal Shift Key (a type of quadrature shift modulation like QPSK)
  - GSM




- MDMCFG2, DEVIATN

# 0x3030 – intermediate frequency

- mix the RF and LO frequencies to create an IF
  - improves signal selectivity
  - tune different frequencies to an IF that can be manipulated easily
  - cheaper/simpler components
- cc1111 supports a wide range of 31 different IF options:
  - 23437 hz apart, from 0 – 726.5 khz
- Smart RF Studio recommends:
  - 140 khz up to 38.4 kbaud
  - 187.5 khz at 38.4 kbaud
  - 281 khz at 250 kbaud
  - 351.5khz at 500 kbaud
- FSCTRL1



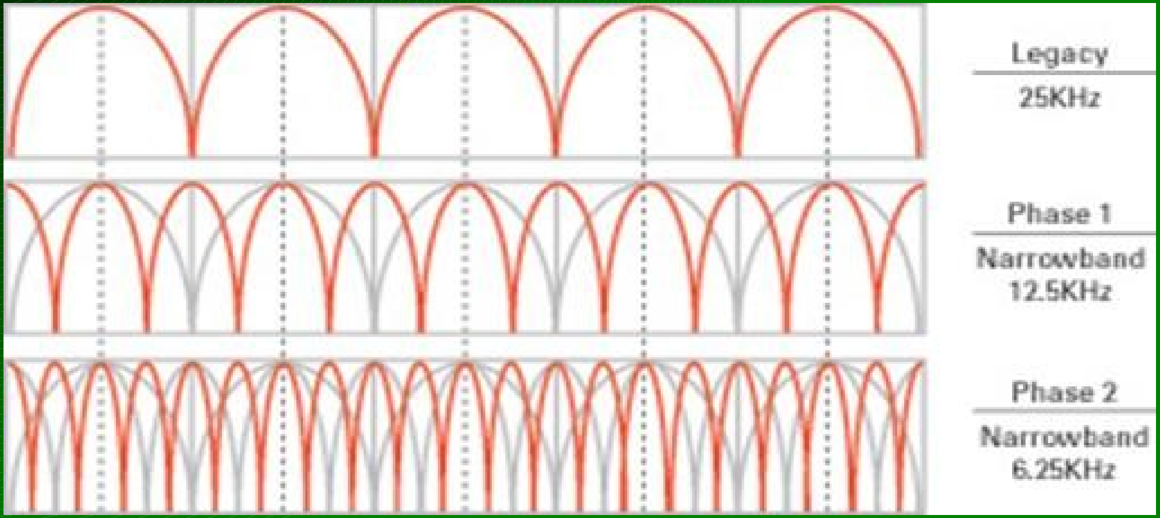
# 0x3040 – data rate (baud)

- much like your modems or old
- the frequency of bits
  - some can overlap and get garbage!
    - garbage can be good...
- baud has significant impact on IF, Deviation and Channel BW
- seeing use of 2400, 19200, 38400, 250000
- MDMCFG3 / 4

0x3050 – channel width / spacing

- simplifying frequency hopping / channelized systems
- real freq = base freq + (CHANNR * width)

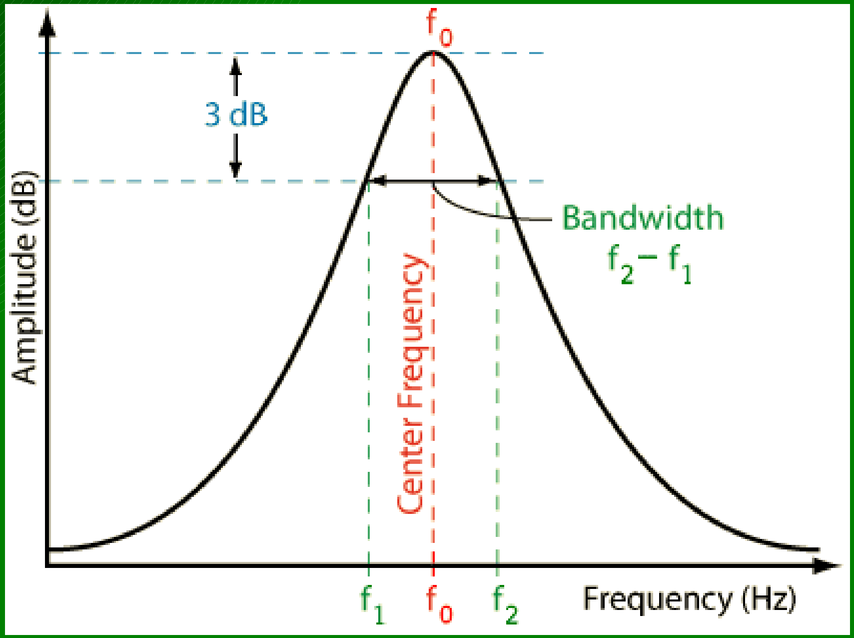
- MDMCFG0 / 1



# 0x3060 – bandwidth filter

- programmable receive filter
- provides for flexible channel sizing/spacing

- total signal bw = signal bandwidth + (2*variance)
- total signal bw **wants** to be less than 80% bw filter!
- MDMCFG4



# 0x3070 – preamble / sync words

- identify when real messages are being received!
- starts out with a preamble (1 0 1 0 1 0 1 0...)
- then a sync word (programmable bytes)
  - marking the end of the preamble
  - SFD – start of frame delimiter
- configurable to:
  - nothing (just send received crap)
  - carrier detect (if the RSSI value indicates a message)
  - 15 or 16 bits of the SYNC WORD identified
  - 30 out of 32 bits of double-SYNC WORD
- SYNC1, SYNC0, MDMCFG2

# 0x3080 – variable / fixed-length packets

- packets can be fixed length or variable length
- variable length assumes first  bytes is the length byte
- both modes use the PKTLEN register:
  - Fixed: the length
  - Variable: MAX length
- PKTCTRL0

# 0x3090 – CRC – duh, but not

- crc16 check on both TX and RX
- uses the internal CRC (part of the RNG) seeded by 0xffff
- DATA_ERROR triggers when CRC is enabled and fails

- PKTCTRL0

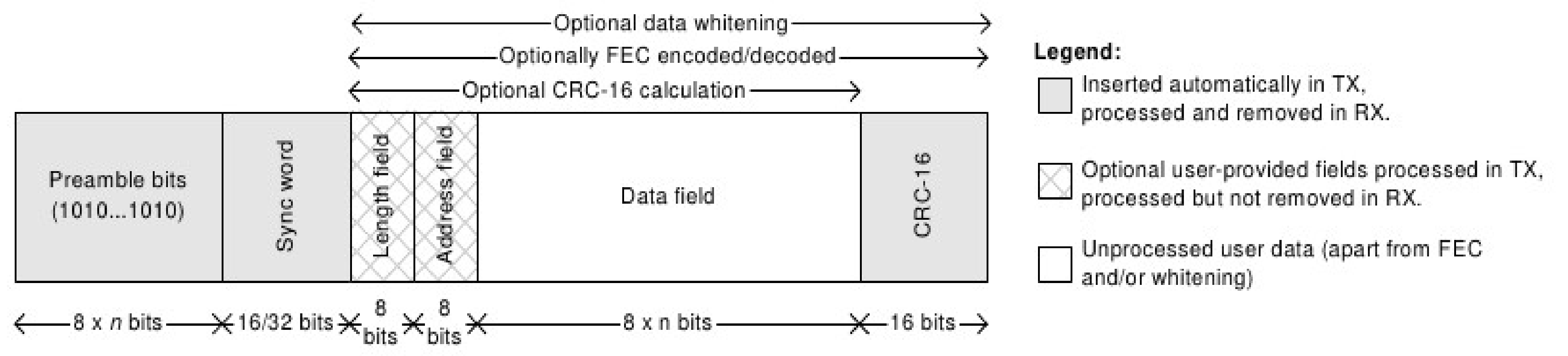



Figure 51: Packet Format

# 0x30a0 – data whitening – 9 bits of pain

- ideal radio data looks like random data
- real world data can contain long sequences of 0 or 1
- data to be transmitted is first XOR'd with a 9-bit sequence
    - sequence repeated as many times as necessary to match the data
- PKTCTRL0

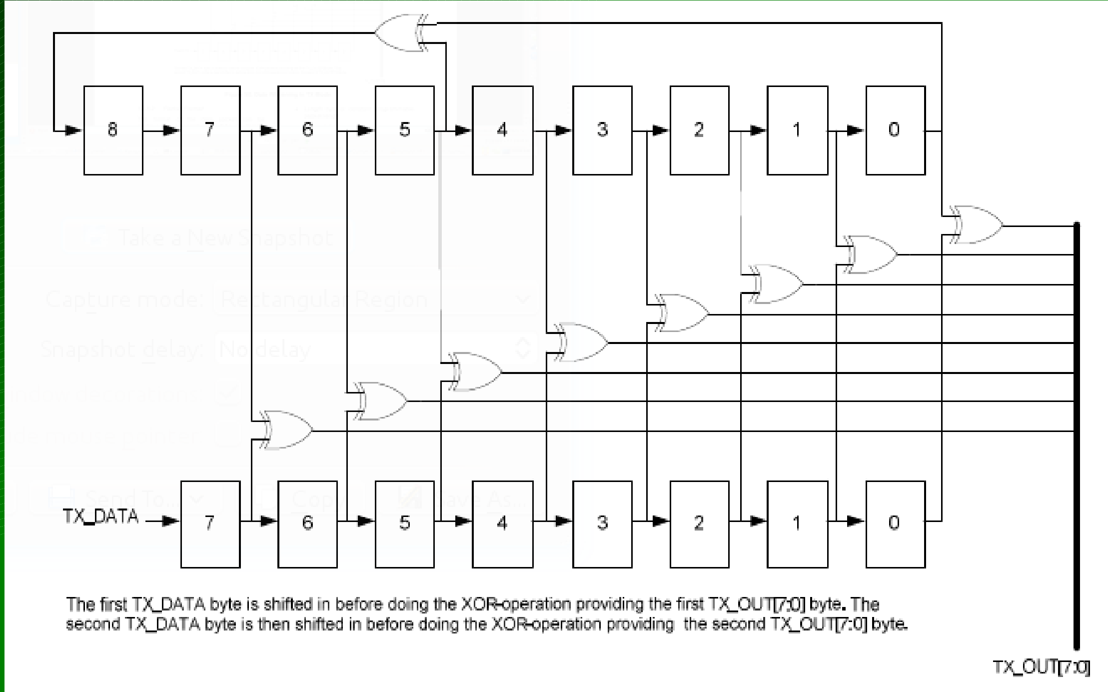
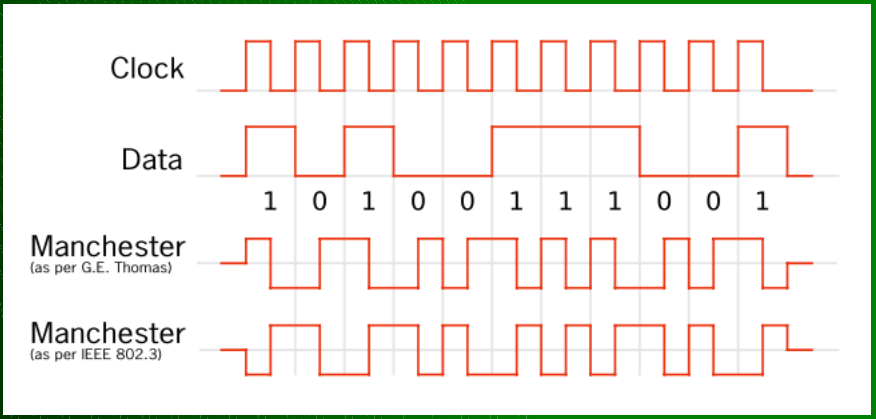
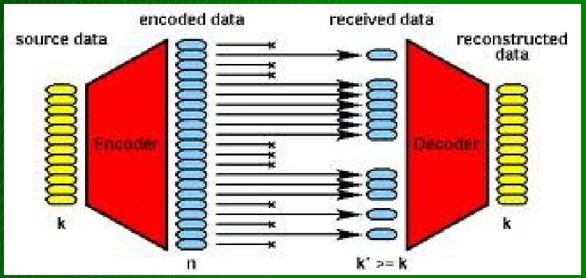


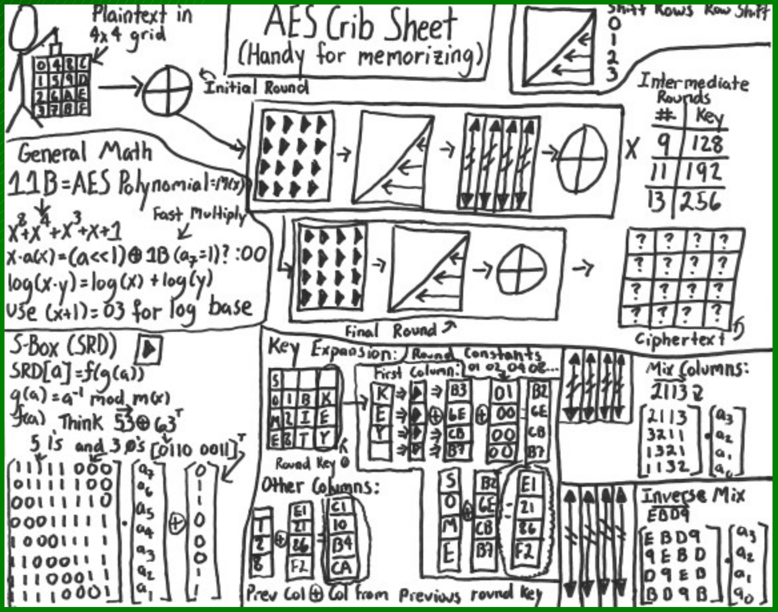

**Figure 50: Data Whitening in TX Mode**

# 0x30b0 – encoding

- manchester
  - MDMCFG2
- forward error correction
  - convolutional
    - MDMCFG1
  - reed-solomon (not supported)
- encryption - AES in chip



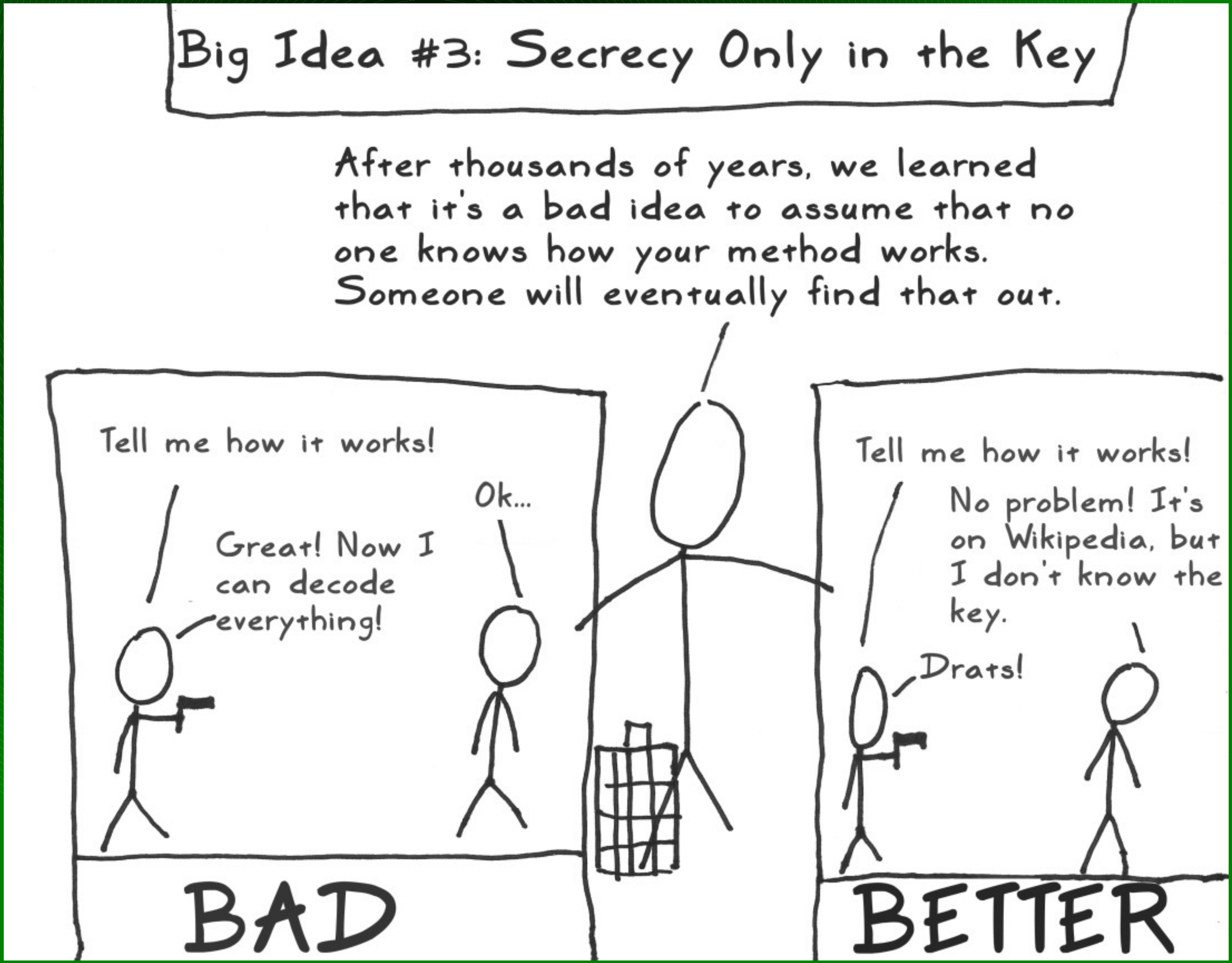



# 0x30c0 – MDMCFG2 register

**0xDF0E: MDMCFG2 - Modem Configuration**

| Bit | Field Name | Reset | R/W | Description |
|---|---|---|---|---|
| 7 | DEM_DCFILT_OFF | 0 | R/W | Disable digital DC blocking filter before demodulator. The recommended IF frequency changes when the DC blocking is disabled. Please use SmartRF® Studio [9] to calculate correct register setting. |
| | | | | 0 – Enable – Better Sensitivity |
| | | | | 1 – Disable – Current optimized. Only for data rates ≤ 100 kBaud |
| 6:4 | MOD_FORMAT[2:0] | 000 | R/W | The modulation format of the radio signal |
| | | | | 000 – 2-FSK |
| | | | | 001 – GFSK |
| | | | | 010 – Reserved |
| | | | | 011 – ASK/OOK |
| | | | | 100 – Reserved |
| | | | | 101 – Reserved |
| | | | | 110 – Reserved |
| | | | | 111 – MSK |
| | | | | Note that MSK is only supported for data rates above 26 kBaud and GFSK, ASK , and OOK is only supported for data rate up until 250 kBaud. MSK cannot be used if Manchester encoding/decoding is enabled. |
| 3 | MANCHESTER_EN | 0 | R/W | Manchester encoding/decoding enable |
| | | | | 0 – Disable |
| | | | | 1 – Enable |
| | | | | Note that Manchester encoding/decoding cannot be used at the same time as using the FEC/Interleaver option or when using MSK modulation. |
| 2:0 | SYNC_MODE[2:0] | 010 | R/W | Sync-word qualifier mode. |
| | | | | The values 000 and 100 disables preamble and sync word transmission in TX and preamble and sync word detection in RX. |
| | | | | The values 001, 010, 101 and 110 enables 16-bit sync word transmission in TX and 16-bits sync word detection in RX. Only 15 of 16 bits need to match in RX when using setting 001 or 101. The values 011 and 111 enables repeated sync word transmission in TX and 32-bits sync word detection in RX (only 30 of 32 bits need to match). |
| | | | | 000 – No preamble/sync |
| | | | | 001 – 15/16 sync word bits detected |
| | | | | 010 – 16/16 sync word bits detected |
| | | | | 011 – 30/32 sync word bits detected |
| | | | | 100 – No preamble/sync, carrier-sense above threshold |
| | | | | 101 – 15/16 + carrier-sense above threshold |
| | | | | 110 – 16/16 + carrier-sense above threshold |
| | | | | 111 – 30/32 + carrier-sense above threshold |

# sorry, couldn't resist



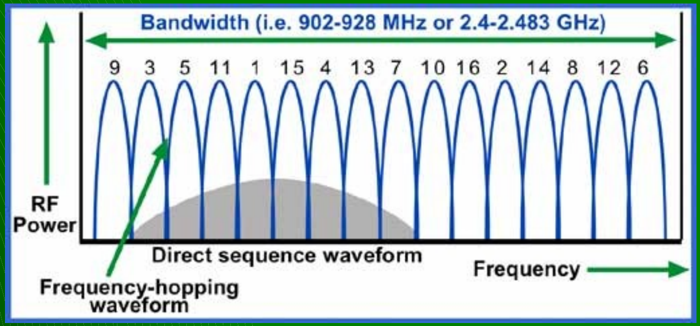
# 0x3100 – how can we figure it out!?

- open / public documentation
  - insulin pump published frequency
- open source implementation / source code
- “public” but harder to find (google fail!)
  - fcc.gov – search for first part of FCC ID
    - http://transition.fcc.gov/oet/ea/fccid/ -bookmark it
  - patents – amazing what people will patent!
    - french patent describing the whole MAC/PHY
- reversing hw
  - tapping bus lines – logic analyzer
    - grab config data
    - grab tx/rx data
  - pulling and analyzing firmware

# 0x3101 – how can we figure it out!? - part 2

- hopping pattern analysis
  - arrays of dongles – space them out and record results
  - hedyattack, or something similar
  - spectrum analyzer
  - USRP2
- trial and error – rf parameters
- MAC layer? - takes true reversing

# 0x2000 – intro to FHSS




- FHSS is common for devices in the ISM bands
  - provides natural protection against unintentional jamming /interferance
  - US Title 47 CFR 15.247 affords special power considerations to FHSS devices
    - >25khz between channels
    - pseudorandom pattern
    - each channel used equally (avg) by each transmitter
    - if 20db of hopping channel < 250khz:
      - must have at least 50 channels
      - average <0.4sec per 20 seconds on one channel
    - if 20dB of hopping channel >250khz:
      - must have at least 25 channels
      - average <0.4sec per 10 seconds on one channel

# 0x2010 – FHSS, the one and only - NOT!

- different technologies:
  - DSSS – Direct Sequence Spread Spectrum
    - hops happen more often than bytes (ugh)
    - typically requires special PHY layer
  - “FHSS”
    - hops occur after a few symbols are transmitted
- different topologies: (allow for different synch methods)
  - point-to-point (only two endpoints)
  - multiple access systems
- different customers:
  - military has used frequency hopping since Hedy and George submitted the patent in 1941.
  - commercial folks (WiFi, Bluetooth, proprietary stuff like power meters)

# 0x2020 – FHSS intricacies

- what's so hard about FHSS?
  - must know or be able to come up with the hopping pattern
    - can be anywhere from 50 to a million distinct channel hops before the pattern repeats (or more)
  - must be able to synchronize with an existing cell or partner
    - or become your own master!
  - must know channel spacing
  - must know channel dwell time (time to sit on each channel)
  - likely need to reverse engineer your target
  - DSSS requires that you have special hardware
- military application will be very hard to crack, as it typically will have hops based on a synchronized PRNG to select channels

# 0x2030 – FHSS, the saving graces

- any adhoc FHSS multi-node network: (power meters / sensor-nets)
  - node sync in a reasonable timeframe
    - limited channels in the repeated pattern
  - each node knows how to talk to a cell
    - let one figure it out, then tap the SPI bus to see what the pattern is...
- two keys to determining hopping pattern:
  - hop pattern generation algorithm
    - typically based on the CELL ID
    - one pattern gets you the whole cell :)
  - some sync information the cell gives away for free
    - gotta tell the n00bs how to sync up, right?
    - for single-pass repeating sequences, it's just the channel

# 0x2040 – FHSS summary

- FHSS comes in different forms for different uses and different users
- FHSS is naturally tolerant to interference, and allows a device to transmit higher power than nonFHSS comms
- getting the FHSS pattern, timing, and appropriate sync method for proprietary comms can be a reversing challenge
- getting a NIC to do something with the knowledge gained above has – to date – been very difficult

# 0x3000 – intro to the RFCAT project

- rfcat: RF Chipcon-based Attack Toolset
- background...
- goals...
- plans...
- where we're at so far...

# 0x3010 – rfcat background

- the power grid
  - power meters and the folks who love them (yo cutaway, q, travis and josh!)
  - no availability of good attack tools for RF
- vendor at Distributech 2008:

  “Our Frequency Hopping Spread Spectrum is too fast for hackers to attack.”

  - OMFW!  really?

# 0x3020 – rfcat goals

- RE tools - “how does this work?”
- security analysis tools - “your FHSS and Crypto is weak!”
- satiate my general love of RF

- a little of Nevil Maskelyne
- “I will not demonstrate to any man who throws doubt upon the system” - Guglielmo Marconi, 1903

# 0x3030 – this is not HedyAttack

- but leveraged the knowledge from HA...
- this is the base code which HedyAttack started...
- less "researchy" (this project won't find hopping patterns)
- more utilitarian - give us comms parameters and a hopping pattern, and we'll be a NIC, sniffer, and interact with RF gadgets

# 0x3040 – rfcat's interface

- rfcat is many things, but I like to think of it as an interactive python access to the <GHz spectrum!
  - <insert pic>
- rfcat_nic.py
  - FHSS-capable NIC (some assembly may be required for FHSS to arbitrary devices)
  - toolset for discovering what is currently using a given band/channel
- rfcat_nic_server.py
  - access the <GHz band over an IP network or locally and configure on the fly

# 0x3050 – rfcat_nic.py

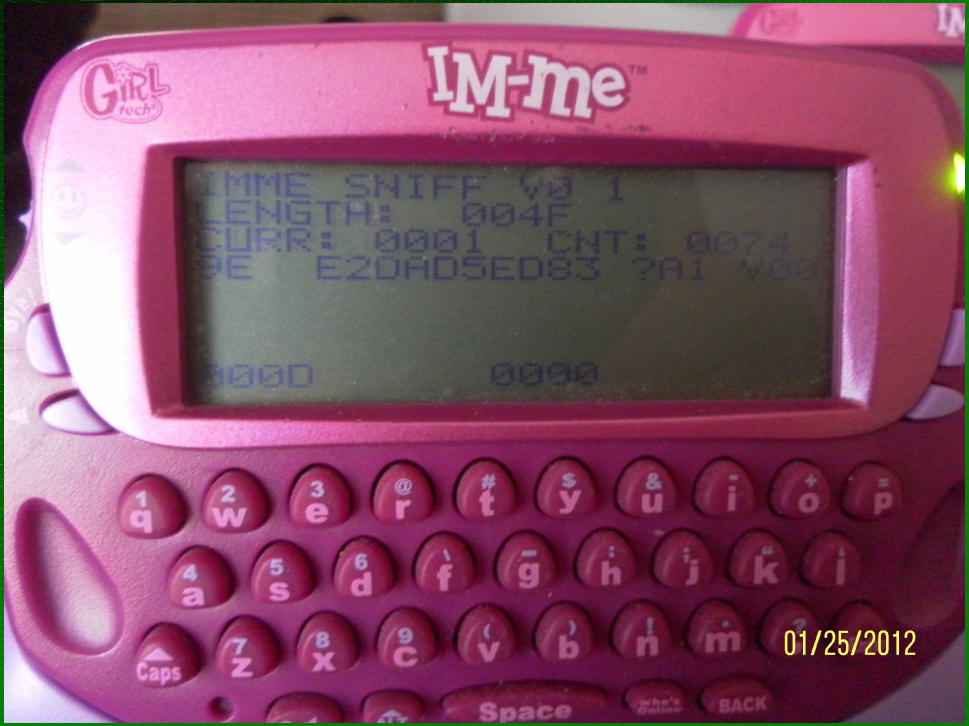
- customizable NIC-access to the ISM bands
- ipython for best enjoyment
- <insert pic>
- lame spoiler:  you get a global object called “d” to talk to the dongle
- d.RFxmit('blah')
- data = d.RFrecv()
- d.discover(lowball=1)

# 0x3060 – rfcat_nic_server.py

- bringing <GHz over the IP network!
- connect on one TCP port to access the wireless network
- connect on a second TCP port to access the wireless configuration
- created to allow non-python clients to play too
  - stdin is not the way you want to interact with embedded wireless protocols
  - <insert pic>

# 0x3070 – rfsniff  (pink version too!)

- focused primarily on capturing data from the wireless network
- IMME used to provide a nice simple interface
- recently added RF config adjustment using keyboard!



# 0x3100 – one dongle to rule them all

- example RF attack lab setup:
  - dongle “Gina” running hedyattack spec-an code
  - dongle “Anna” running rfcat_nic.py
  - IMME running rfsniff
  - (possibly an IMME's running SpecAn)
  - saleae logic analyzer for hacking of the wired variety

# rf attack form

- base freq:
- modulation:
- baud/bandwidth:
- deviation:
- channel hopping?
  - how many channels: channel spacing:
  - pattern and effective sync method?
  - dwell period (ms):
- fixed-/variable-length packets: len/maxlen:
- "address":
- sync word (if applicable):
- crc16 (y/n): does chip do correct style?
- fec (y/n): type (convolutional/reed-soloman/other):
- manchester encoding (y/n):
- data whitening? and 9bit pattern:
-

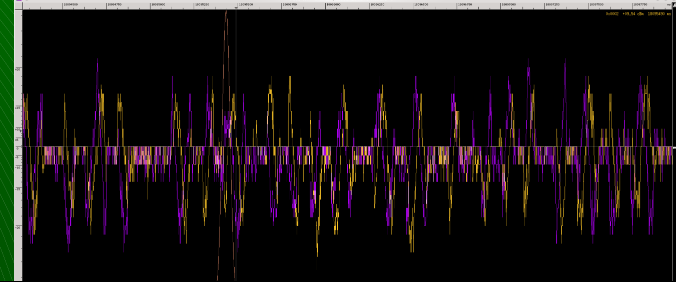
# 0x4000 – playing with a medical device

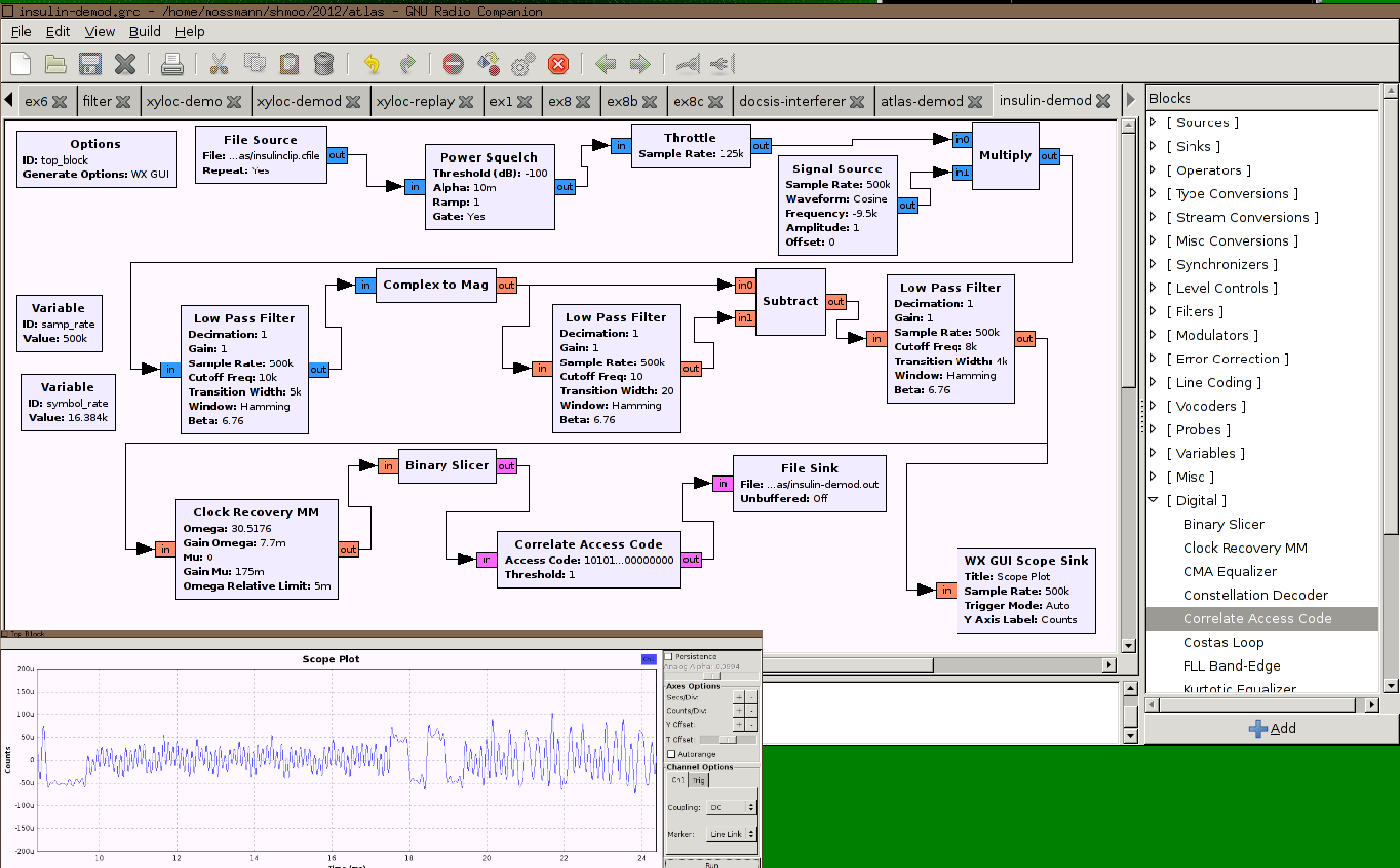
- CAUTION: MUCKING WITH THESE CAN KILL PEOPLE.
  - THIS FIRMWARE AND CLIENT NOT PROVIDED
- found frequency in the pdf manual from the Internet
  - what random diabetic cares what frequency his pump communicates with!? ok, who cares!
- modulation guessed based on spectrum analysis and trial/error
  - the wave form just **looks** like <blah> modulation!
- other characteristics discovered using a USRP and baudline (and some custom tools, thanks Mike Ossman!)

# 0x4010 – the discovery process

- glucometer was first captured using Spectrum Analyzer (IMME/hedyattack) to validate frequency range from the lay-documentation
- next a logic analyzer (saleae) used to tap debugging lines
- next, the transmission was captured using a USRP (thank you Mike Ossman for sending me your spare!)
- next, the “packet capture” was loaded into Baudline, and analysis performed to identify baudrate and modulation scheme, and get an idea of bits
- next, Mike Ossman did amazing-sauce, running the capture through GnuRadio Companion (the big picture on next slide)
- RF parameters confirmed through RF analysis, and real-life capture.

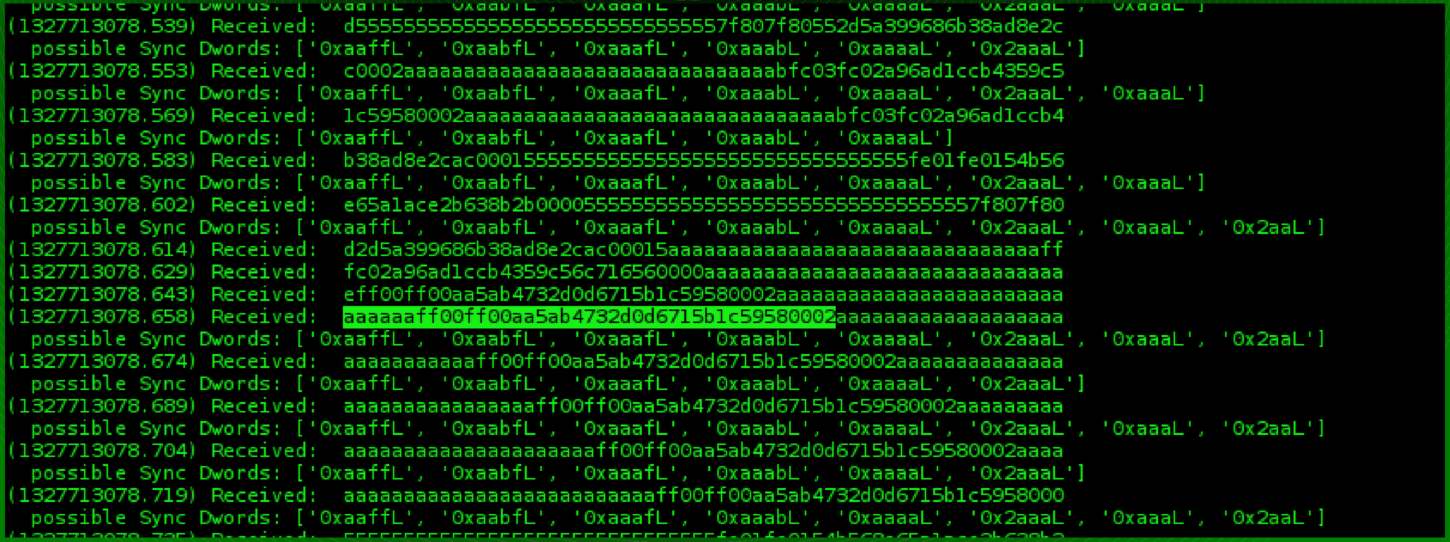
# 0x4011 – discovery reloaded



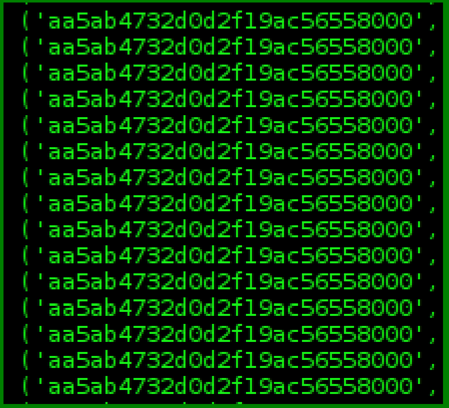


# 0x4020 –the immaculate reception

- punched in the RF parameters into a RFCAT dongle
  - created subclass of RFNIC (in python) for new RF config
- dropped into “discover” mode to ensure I had the modem right




- returned to normal NIC mode to receive real packets
- now need the pump to reverse the bi-dir protocol



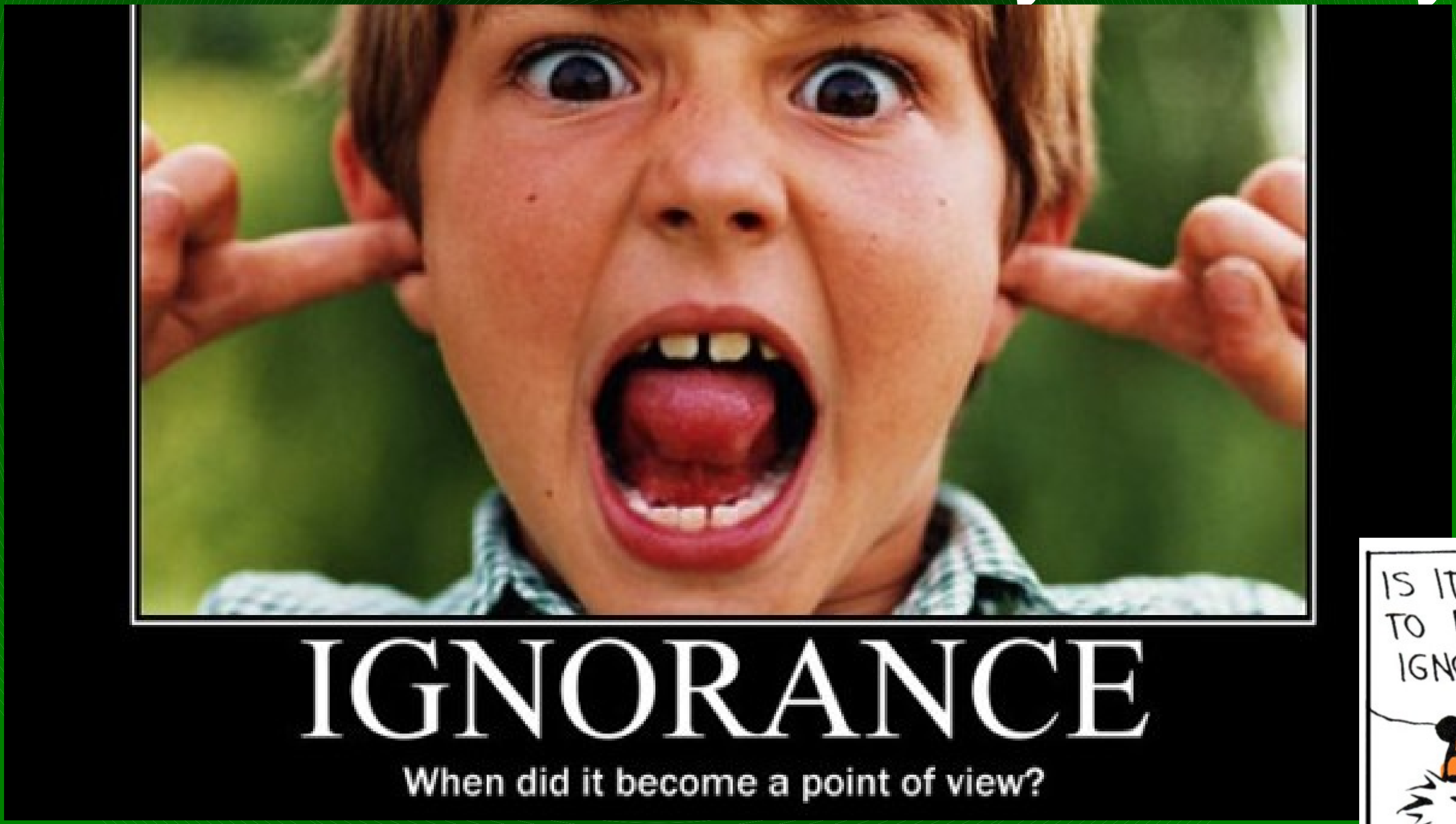
# 0x4100 – playing with a power meter

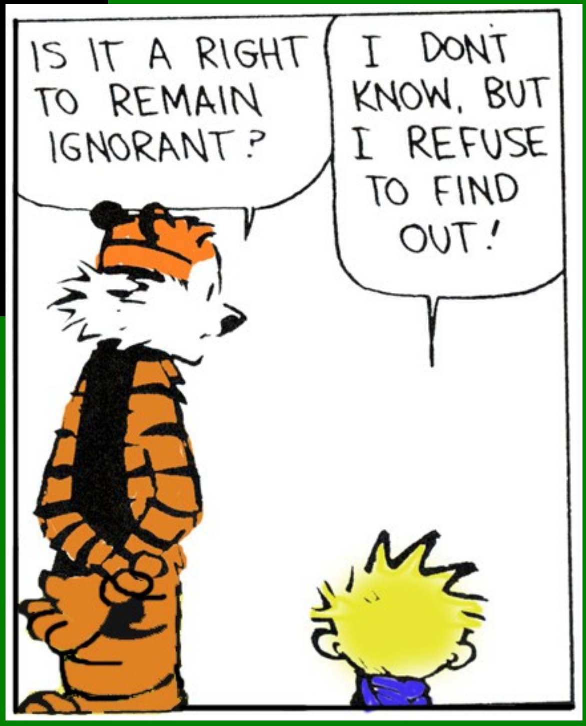

- CAUTION: MUCKING WITH POWER SYSTEMS WITHOUT APPROPRIATE AUTHORIZATION IS ILLEGAL, EVEN IF IT IS ON THE SIDE OF YOUR HOUSE!
- most power meters use their own proprietary “Neighborhood Area Network” (NAN), typically in the 900MHz range and sometimes 2.4GHz or licensed spectrum.
- to get the best reception over distance and gain tolerance to interference, all implement FHSS to take advantage of the Title 47: Part 15 power allowances
- many of the existing meters use the same cc1111 or cc1110 chips, or the cc1101 radio core
- this is the reason I'm here today

# 0x4110 – as sands through the hourglass

- power meters have long been “unavailable” for most security researchers
- **some vendors understand the benefits of security rigor by outside researchers**
  - others, however, do not.
- the gear used in my presentation was given to me by one such company
  - for various reasons, they have asked to remain anonymous, however, their security team has a well founded approach to finding out “how their baby is ugly” I would like to give them credit for their commitment to the improved security of their products.

# atlas, tell us what you really think



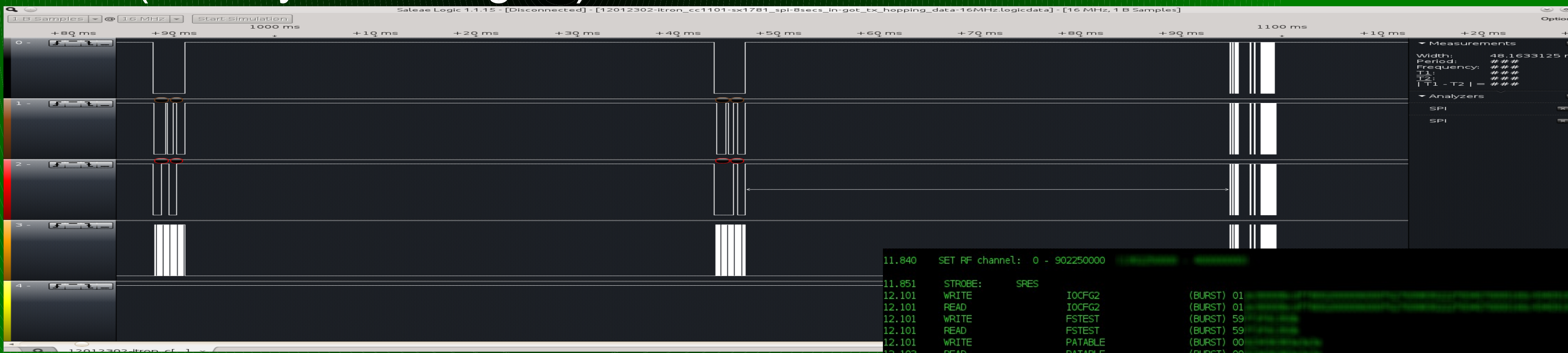



# 0x4120 – smart meter – the complication

- power meters are not so simple as glucometers
  - proprietary FHSS in a multiple-access topology
  - have to endure the RF abuse of the large metropolis
- complex mac sync/net-registration
- not easy to show with a single meter without a Master node.
- initial analysis was performed via my saleae LA:
- SpecAn code on IMME's and hedyattack dongles
  - good for identifying periods of scanning
- although the dongle can hop along with the meter, we won't be demoing synching with the meter today

# 0x4130 – the approach

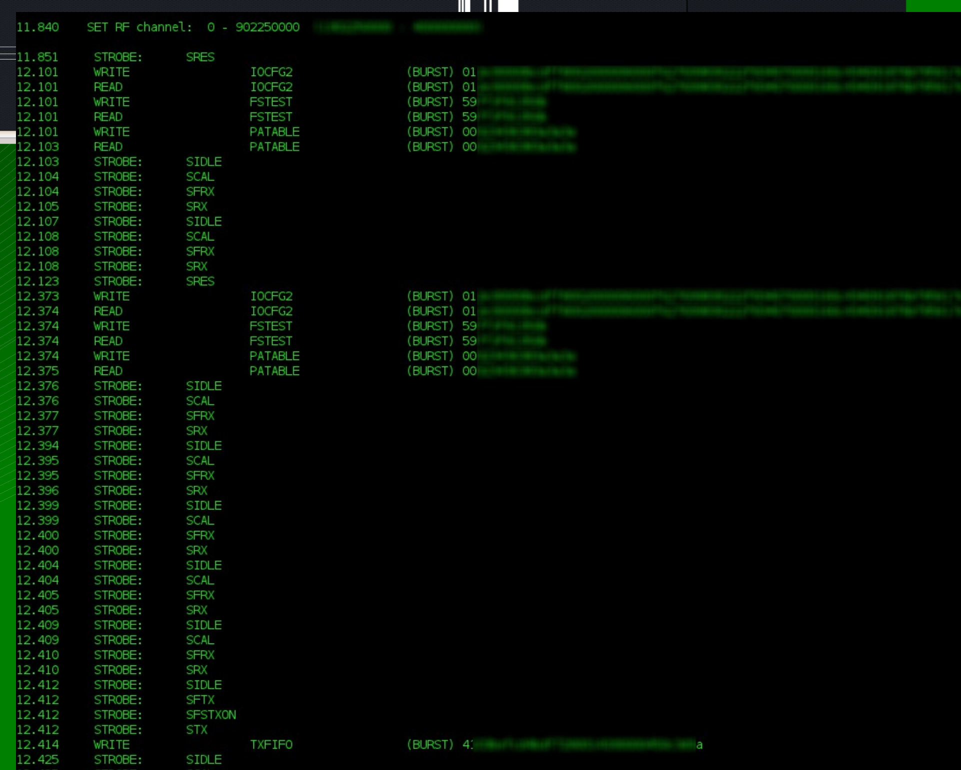
- determine the rf config and hopping pattern through SPI Bus sniffing (and my saleae again)



- decoding:

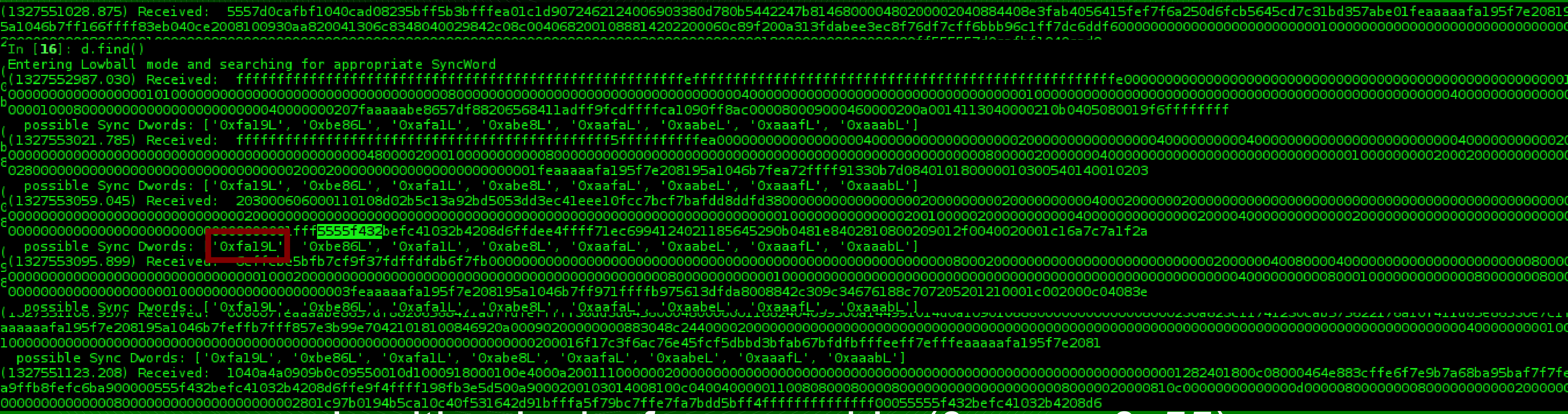

custom parser for the

target radio--->>>



# 0x4140 – the approach (2)

- discover mode:
  - disables sync-word so radio sends unaligned bits




  - algorithm looks for preamble (0xaa or 0x55)
    - then determines possible dwords
- ummm... but that's not any bit-derivation of the sync word(s) I expect.  wut? I am confident those are coming from the meter

# 0x4150 – the result

“Abuse is no argument”

- Nevil Maskelyne

# conclusions

- rfcat discover mode roxors
- rfcat is a **foundation** for your attack tool
  - way more than just a tool in itself
-

# References


- http://rfcat.googlecode.com
- http://en.wikipedia.org/wiki/Part_15_(FCC_rules)
- http://en.wikipedia.org/wiki/ISM_band
- http://www.ti.com/lit/ds/swrs033g/swrs033g.pdf - “the” manual
- http://edge.rit.edu/content/P11207/public/CC1111_USB_HW_User_s_Guide.pdf
- http://www.ti.com/litv/pdf/swru082b
- http://www.ti.com/product/cc1111f32#technicaldocuments
- http://www.ti.com/lit/an/swra077/swra077.pdf
- http://www.newscientist.com/article/mg21228440.700-dotdashdiss-the-gentleman-hackers-1903-lulz.html
- http://saleae.com/
- http://zone.ni.com/devzone/cda/epd/p/id/5150 - FSK details (worthwhile!)
- http://www.radagast.org/~dplatt/hamradio/FARS_presentation_on_modulation.pdf
  - very good detailed discussion on deviation/modulation
- http://en.wikipedia.org/wiki/Frequency_modulation

# 0xgreetz

- power hardware folk who play nice with security researchers
- cutaway and q (awesome hedyattackers)
- gerard van den bosch
- travis and mossman
- sk0d0 and the four J's
- invisigoth and kenshoto
- Jewel, bug, ringwraith, diva
- Jesus Christ