"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 – microwave ovens?
  - Medical – insulin pumps and the like

- US ISM bands:
  - 300 : 300
  - 433 : 433.050 – 434.790 MHz
  - 915 : 902.000 – 928.000 MHz
  - cc1111 does 300-348, 372-460, 779-928... but we've seen more.

- 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?
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...
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!
• 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 :)

• Full Speed USB

• RfCat hides most of these details by default!
**0x2010 – cc1111 radio state engine**

- **IDLE**
- **CAL**
- **FSTXON**
- **RX**
- **TX**

**IDLE**

Default state when the radio is not receiving or transmitting.

**CAL**

Used for calibrating frequency synthesizer upfront (entering receive or transmit mode can then be done quicker). Transitional state.

**FSTXON**

Frequency synthesizer is on, ready to start transmitting. Transmission starts very quickly after receiving the STX command strobe.

**RX**

Reception is turned off and this state is entered if the RFD register overflows.

**TX**

Transmission is turned off and this state is entered if the RFD register becomes empty in the middle of a packet. Typ. current consumption: 1.8 mA
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
- FRENDF1, FRENDF0 – 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 – gain control

RfCat hides most of these details by default!
**Data Rate, Bandwidth, and Intermediate Frequency and Freq-Deviation depend on each other**

### Typical settings

<table>
<thead>
<tr>
<th>Data rate</th>
<th>Dev.</th>
<th>Mod.</th>
<th>RX BW</th>
<th>Optimized for</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 kBaud</td>
<td>5.1 kHz</td>
<td>GFSK</td>
<td>69 kHz</td>
<td>Optimised for current consumption</td>
<td></td>
</tr>
<tr>
<td>2.4 kBaud</td>
<td>5.1 kHz</td>
<td>GFSK</td>
<td>63 kHz</td>
<td>Optimised for sensitivity</td>
<td></td>
</tr>
<tr>
<td>2.4 kBaud</td>
<td>5.1 kHz</td>
<td>GFSK</td>
<td>63 kHz</td>
<td>Optimised for current consumption</td>
<td></td>
</tr>
<tr>
<td>9.6 kBaud</td>
<td>20 kHz</td>
<td>GFSK</td>
<td>94 kHz</td>
<td>Optimised for sensitivity</td>
<td></td>
</tr>
<tr>
<td>250 kBaud</td>
<td>129 kHz</td>
<td>GFSK</td>
<td>600 kHz</td>
<td>Optimised for sensitivity</td>
<td></td>
</tr>
</tbody>
</table>

### RF Parameters

- **Base frequency**: 868.299683 MHz
- **Channel number**: 0
- **Channel spacing**: 199.951172 kHz
- **Carrier frequency**: 868.299683 MHz
- **Data rate**: 1.159877 kBaud
- **RX filter BW**: 62.500000 kHz
- **Deviation**: 5.126953 kHz
- **TX power**: 0 dBm

### Packet TX

- **Packet payload size**: 30, 100
- **Packet count**: Random, Text, Hex
- **Sent packets**: 0
- **Frequency**: 868.299683 MHz
- **Output power**: 0 dBm
0x2100 — RfCat for devs

- **cc1111usb.c** provides usb descriptors and framework
  - shouldn't need much tinkering
- **cc1111rf.c** provides the core of the radio firmware
  - shouldn't need much tinkering
- **application.c** provides the template for new apps
  - copy it and make your amazing toy
- **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[]
- **transmit(*buf, length)** allows you to send on the RF pipeline
- **appMainLoop()** — modify this for handling RF packets, etc...
- follow the examples, luke!
  - RfCat's “application” source is **appFHSSNIC.c**
0x3000 – radio info 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?
- any encoding (manchester, fec, enc, etc...)
0x3010 – interesting frequencies

- 315MHz – car fobs
- 433MHz – medical devices, garage door openers
- 868MHz – EU loves this range
- 915MHz – NA stuff of all sorts (power meters, insulin pumps, industrial plant equipment, industrial backhaul)
- 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 (heterodyne)
  - 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
  - aka 'SFD' – start of frame delimiter
- configurable to:
  - nothing (just dump 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 byte is the length byte
- both modes use the PKTLEN register:
  - Fixed: the length
  - Variable: MAX length
- PKTCTRL0, PKTLEN
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 flag triggers when CRC is enabled and fails
- some systems do this in firmware instead

- PKTCTRL0

![Packet Format Diagram](image-url)
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
0x30b0 — encoding

- manchester
  - MDMCFG2
- forward error correction
  - convolutional
    - MDMCFG1
      - reed-solomon (not supported)
- encryption - AES in chip
0x30c0 — example: MDMCFG2 register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Name</th>
<th>Reset</th>
<th>R/W</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>DEM_DCFILT_OFF</td>
<td>0</td>
<td>R/W</td>
<td>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.</td>
</tr>
<tr>
<td>6:4</td>
<td>MOD_FORMAT[2:0]</td>
<td>000</td>
<td>R/W</td>
<td>The modulation format of the radio signal</td>
</tr>
<tr>
<td></td>
<td>000: 2-FSK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001: GFSK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010: Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>011: ASK/OOK</td>
<td>100</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>101: Reserved</td>
<td>110</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>111: MSK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Name</th>
<th>Reset</th>
<th>R/W</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>MANCHESTER_EN</td>
<td>0</td>
<td>R/W</td>
<td>Manchester encoding/decoding enable</td>
</tr>
<tr>
<td></td>
<td>0: Disable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Enable</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that Manchester encoding/decoding cannot be used at the same time as using the FEC/Interleaver option or when using MSK modulation.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Name</th>
<th>Reset</th>
<th>R/W</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>SYNC_MODE[2:0]</td>
<td>010</td>
<td>R/W</td>
<td>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 010, 010, 101 and 110 enables 16-bit sync word transmission in TX and 16-bit 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-bit sync word detection in RX (only 32 of 32 bits need to match).</td>
</tr>
<tr>
<td></td>
<td>000: No preamble/sync</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001: 15/16 sync word bits detected</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010: 16/16 sync word bits detected</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>011: 30/32 sync word bits detected</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100: No preamble/sync, carrier-sense above threshold</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>101: 15/16 + carrier-sense above threshold</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>110: 16/16 + carrier-sense above threshold</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111: 30/32 + carrier-sense above threshold</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Big Idea #3: Secrecy Only in the Key

After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.

Tell me how it works!

Great! Now I can decode everything!

Ok...

Tell me how it works!

No problem! It's on Wikipedia, but I don't know the key.

Drats!

BAD

BEFTER
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!
    - http://freepatentsonline.com
    - french patent describing the whole MAC/PHY of one meter
    - and another:
0x3101 — how can we figure it out!? -part2

- reversing hw
  - tapping bus lines – logic analyzer
    - grab config data
    - grab tx/rx data
  - pulling and analyzing firmware

- hopping pattern analysis
  - arrays of dongles – space them out and record results
  - hedyattack, or something similar
  - spectrum analyzer
  - USRP2 or latest gadget from Michael Ossman

- trial and error – rf parameters

- MAC layer? - takes true reversing.. unless you find a patent :)}
0x4000 — intro 2 FHSS — SPDY!

- FHSS is common for devices in the ISM bands
  - provides natural protection against unintentional jamming /interference
  - 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
0x4010 – 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 (couple different options)
    - each cell has their own hopping pattern
    - each node has own hopping pattern

- 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)
0x4020 — 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
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
  - often based on the CELL ID
    - one pattern gets you the whole cell :)
  - others generate a unique pattern per node
- 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
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 non-FHSS 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
0x5000 — intro to RfCat

- RfCat: RF Chipcon-based Attack Toolset
- RfCat is many things, but I like to think of it as an interactive python access to the <GHz spectrum!
0x5010 – 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?
0x5020 – 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
  - Iulwut?
0x5030 — rfcat's interface

- **rfcat**
  - FHSS-capable NIC
    - some assembly may be required for FHSS to arbitrary devices
    - toolset for discovering/interfacing with RF devices
- **rfcat_server**
  - access the <GHz band over an IP network or locally and configure on the fly
  - connect to tcp port 1900 for raw data channel
  - connect also to tcp port 1899 for configuration
0x5040 – rfcat

- customizable NIC-access to the ISM bands
- ipython for best enjoyment
- lame spoiler: you get a global object called “d” to talk to the dongle
  - d.RFxmit('blah')
  - data = d.RFrecv()
  - d.discover(lowball=1)
  - d.RFlisten()
  - help(d)
0x5050 - rfcat_server

- bringing <GHz over the IP network!
- connect on TCP port 1900 to access the wireless network
- connect on TCP port 1899 to access the wireless configuration
- created to allow non-python clients to play too
  - stdin is not always the way you want to interact with embedded wireless protocols
0x5060 — rfsniff  (pink version too!)

- focused primarily on capturing data from the wireless network
- IMME used to provide a nice simple interface
- RF config adjustment using keyboard!
0x5065 – rfsniff – key bindings

q, a - inc/dec highest sync word nibble
w, s - inc/dec high-middle sync word nibble
e, d - inc/dec low-middle sync word nibble
r, f - inc/dec lowest sync word nibble
z - NO sync word matching

menu - inc Modulation type
bye! - dec Modulation type

up - inc recv bandwidth
down - dec recv bandwidth

right - inc baudrate
left - dec baudrate

p, Enter - inc/dec frequency
o, ',' - faster inc/dec frequency
i, m - even faster inc/dec frequency
l - set freq to 915mhz
k - set freq to 868mhz
j - set freq to 433mhz
h - set freq to 315mhz
t, v - inc/dec channels
g - set channel = 0

SPACE - switch screens
SPKR - toggle CARRIER TX mode (good for showing up on a SpecAn, or, umm, jamming?)
• d._debug = 1 – dump debug messages as things happen
• d.debug() - print state infoz once a second
• d.discover() - listen for specific SYNCWORDS
• d.lowball() - disable most “filters” to see more packets
• d.lowballRestore() - restore the config before calling lowball()
• d.RFlisten() - simply dump data to screen
• d.RFcapture() - dump data to screen, return list of packets
• d.scan() - scan a configurable frequency range for “stuff”
• print d.reprRadioConfig() - print pretty config infoz
• d.setMdm*() d.setPkt*() d.make*()
0x5100 — example lab setup

- example RF attack lab setup:
  - dongle “Gina” running hedyattack spec-an code
  - dongle “Paul” running rfcat
  - IMME running rfsniff
  - (possibly an IMME's running SpecAn)
  - saleae logic analyzer for hacking of the wired variety
  - FunCube Dongle and quisk/qthid or other SDR
rf attack form

- base freq:
- modulation:
- baud/bandwidth:
- deviation:
- channel hopping?
  - how many channels:
  - pattern and effective sync method?
- channel spacing:
- 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:
- more complete information:
0x6000 — playing with medical devices

• 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!)
0x6010 – the discovery process

- The glucometer was first captured using Spectrum Analyzer (IMME/hedyattack) to validate frequency range from the lay-documentation.

- Next, a logic analyzer (saleae) was used to tap debugging lines.

- Next, the transmission was captured using a USRP (thank you Mike Ossman for sending me your spare!) - alt: FunCube.

- 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.
0x6011 – discovery reloaded
0x6020 – 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
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
0x6110 — as sands through the hourglass

- power meter RF comms 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 who understands
  - 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
0x6120 — 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
0x6130 — the approach

- determine the rf config and hopping pattern through SPI Bus sniffing (and my saleae again)
### 0x6135 – Logic Analyzer

- decoding:
- custom parser for the target radio

<table>
<thead>
<tr>
<th>Address</th>
<th>Action</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.840</td>
<td>SET RF channel</td>
<td>0 - 902250000</td>
<td></td>
</tr>
<tr>
<td>11.851</td>
<td>WRITE</td>
<td>SPES</td>
<td></td>
</tr>
<tr>
<td>11.101</td>
<td>WRITE</td>
<td>IOCFG2</td>
<td>(BURST) 01</td>
</tr>
<tr>
<td>11.101</td>
<td>READ</td>
<td>IOCFG2</td>
<td>(BURST) 01</td>
</tr>
<tr>
<td>11.101</td>
<td>WRITE</td>
<td>FSSTEST</td>
<td>(BURST) 59</td>
</tr>
<tr>
<td>11.101</td>
<td>READ</td>
<td>FSSTEST</td>
<td>(BURST) 59</td>
</tr>
<tr>
<td>11.101</td>
<td>WRITE</td>
<td>PATABLE</td>
<td>(BURST) 00</td>
</tr>
<tr>
<td>11.103</td>
<td>READ</td>
<td>PATABLE</td>
<td>(BURST) 00</td>
</tr>
<tr>
<td>11.103</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>11.104</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>11.105</td>
<td>STROBE</td>
<td>SPRX</td>
<td></td>
</tr>
<tr>
<td>11.107</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>11.108</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>11.108</td>
<td>STROBE</td>
<td>SFRX</td>
<td></td>
</tr>
<tr>
<td>11.123</td>
<td>STROBE</td>
<td>SPX</td>
<td></td>
</tr>
<tr>
<td>12.373</td>
<td>WRITE</td>
<td>IOCFG2</td>
<td>(BURST) 01</td>
</tr>
<tr>
<td>12.374</td>
<td>READ</td>
<td>IOCFG2</td>
<td>(BURST) 01</td>
</tr>
<tr>
<td>12.374</td>
<td>WRITE</td>
<td>FSSTEST</td>
<td>(BURST) 59</td>
</tr>
<tr>
<td>12.374</td>
<td>READ</td>
<td>FSSTEST</td>
<td>(BURST) 59</td>
</tr>
<tr>
<td>12.374</td>
<td>WRITE</td>
<td>PATABLE</td>
<td>(BURST) 00</td>
</tr>
<tr>
<td>12.375</td>
<td>READ</td>
<td>PATABLE</td>
<td>(BURST) 00</td>
</tr>
<tr>
<td>12.376</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>12.376</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>12.377</td>
<td>STROBE</td>
<td>SFRX</td>
<td></td>
</tr>
<tr>
<td>12.394</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>12.394</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>12.395</td>
<td>STROBE</td>
<td>SFRX</td>
<td></td>
</tr>
<tr>
<td>12.396</td>
<td>STROBE</td>
<td>SPX</td>
<td></td>
</tr>
<tr>
<td>12.399</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>12.399</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>12.400</td>
<td>STROBE</td>
<td>SFRX</td>
<td></td>
</tr>
<tr>
<td>12.400</td>
<td>STROBE</td>
<td>SPX</td>
<td></td>
</tr>
<tr>
<td>12.404</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>12.404</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
<tr>
<td>12.405</td>
<td>STROBE</td>
<td>SFRX</td>
<td></td>
</tr>
<tr>
<td>12.405</td>
<td>STROBE</td>
<td>SPX</td>
<td></td>
</tr>
<tr>
<td>12.409</td>
<td>STROBE</td>
<td>SIDLE</td>
<td></td>
</tr>
<tr>
<td>12.409</td>
<td>STROBE</td>
<td>SCAL</td>
<td></td>
</tr>
</tbody>
</table>
**0x6140 — 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

- intro: Bit Inversion (see highlighted hex)
0x6145 — new developments

- vendors have filed numerous patents with hopping pattern calculations, comms parameters, etc...
  - WIN!
  - plenty of work to be done! jump right in!

- http://www.patentstorm.us/applications/20080204272/fulltext.html
- http://www.patentstorm.us/applications/20080238716/fulltext.html
“Abuse is no argument”
- Nevil Maskelyne
rfcat discover mode roxors

rfcat is a *foundation* for your attack tool
  – way more than just a tool in itself

**we** are responsible for ensuring our devices use appropriate security. **do not** simply expect someone else to do it. the first med-device death could be your best friend.
References

- http://rfcat.googlecode.com
- http://www.ti.com/lit/ds/swsr033g/swrs033g.pdf - “the” manual
- http://edge.rit.edu/content/P11207/public/CC1111_USB_HW_User_s_Guide.pdf
- http://www.ti.com/product/cc1111f32#technicaldocuments
- http://saleae.com/
- http://zone.ni.com/devzone/cda/epd/p/id/5150 - FSK details (worthwhile!)
  - very good detailed discussion on deviation/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