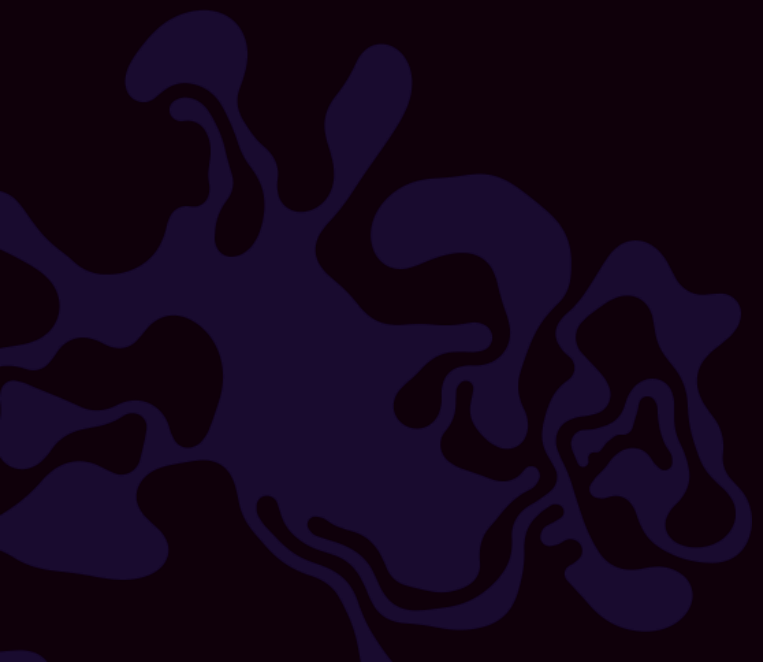
Beyond BLE: Cracking Open the Black-Box of RF Microcontrollers

Adam Batori & Robert Pafford



whoami

Adam Batori (hcadam)

- HW Security Research
- Side Channels & Fault Injection
- Reverse Engineer
- Silicon 0day Enjoyer

- DECT: 70569

Robert Pafford

- Forward & Reverse Engineer
- Experienced with Embedded MCU Development
- Decent at staring at undocumented registers to figure out what they do

- GitHub: `rjp5th`

Background

- Market is flooded with low-cost RF MCUs
  - ESP32, nRF, CC26xx, etc.
- RF hardware with enough DSP power for at least 1Mbps Bluetooth modulation, potentially more
- No info on how the actual RF peripherals work for any of these chips
  - That's not cool
  - We wanted to change that
- *I paid for the whole chip, I want to program the whole chip!*

TI SimpleLink

- Family of 2.4GHz and Sub-1GHz RF Transceivers and MCUs
- CC13xx line for Sub-1GHz + MCU, CC26xx line for 2.4GHz + MCU
  - Some special PNs can do **both** Sub-1GHz and 2.4GHz in the same chip
- 3 Main Generations
  - CC13x0/CC26x0 "Chameleon" (~2015)
    - ARM Cortex-M3 @ 48MHz
  - CC13x2/CC26x2 "Agama" (~2018)
    - ARM Cortex-M4F @ 48MHz
  - CC13x4/CC26x4 "Thor" (~2022)
    - ARM Cortex-M33 @ 48 MHz

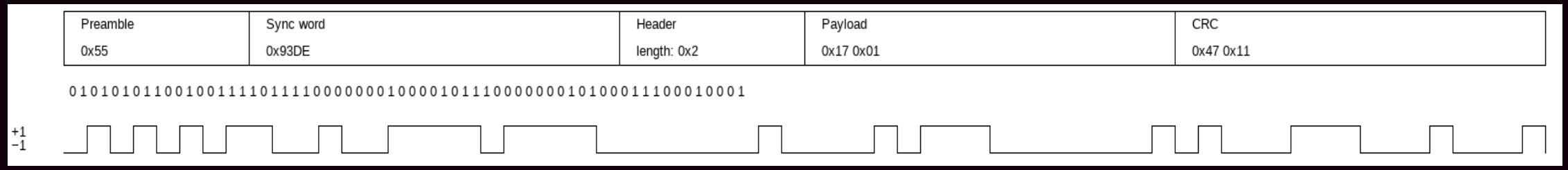
The Target

TI SimpleLink CC1352R Dual-Band Wireless MCU

- Wanted a fully-featured chip for analysis
  - Bluetooth LE 5
  - ZigBee
  - Other IEEE 802.15.4 mode
  - Sub-GHz Proprietary mode (backwards compatible with CC1101)
    - Flipper Zero transceiver
  - Also supports 2.4 GHz Proprietary modes
- Investigate how modes are "locked out" on lower models
- Multi-band IC increases likelihood of flexible RF tuning

A Closer Look At Proprietary Radio Format

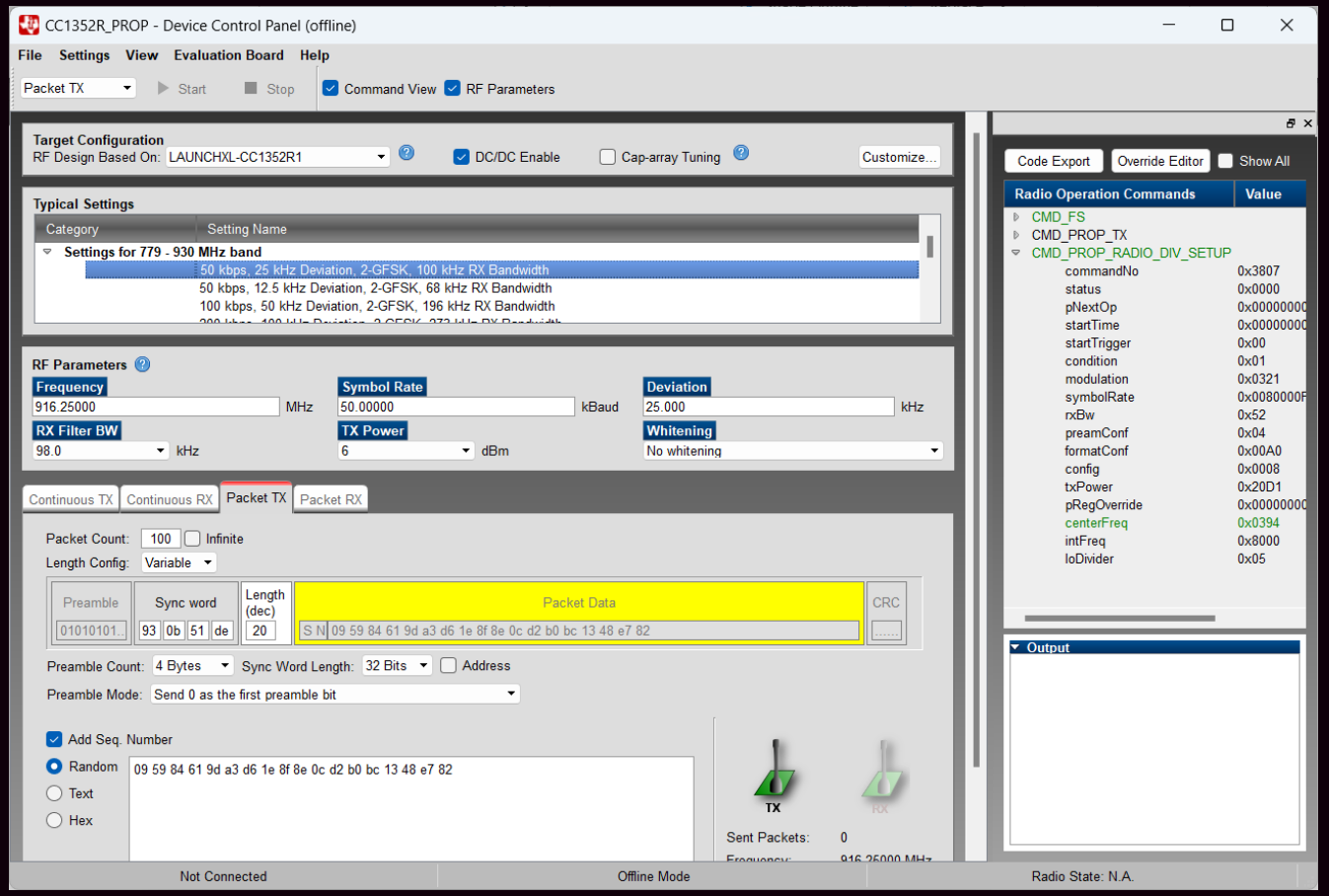
- Simple TX/RX Command Format:



- Supports Various Modulation Schemes:
  - 2-GFSK
  - OOK
  - Long Range (DSSS)
- Everything is abstracted away into a simple TX/RX of a data packet
  - Higher-level features up to the implementer (auto-retry, segmentation, etc.)
- This is already better low-level access than most other RF chips
  - Still far from being able to fully control PHY

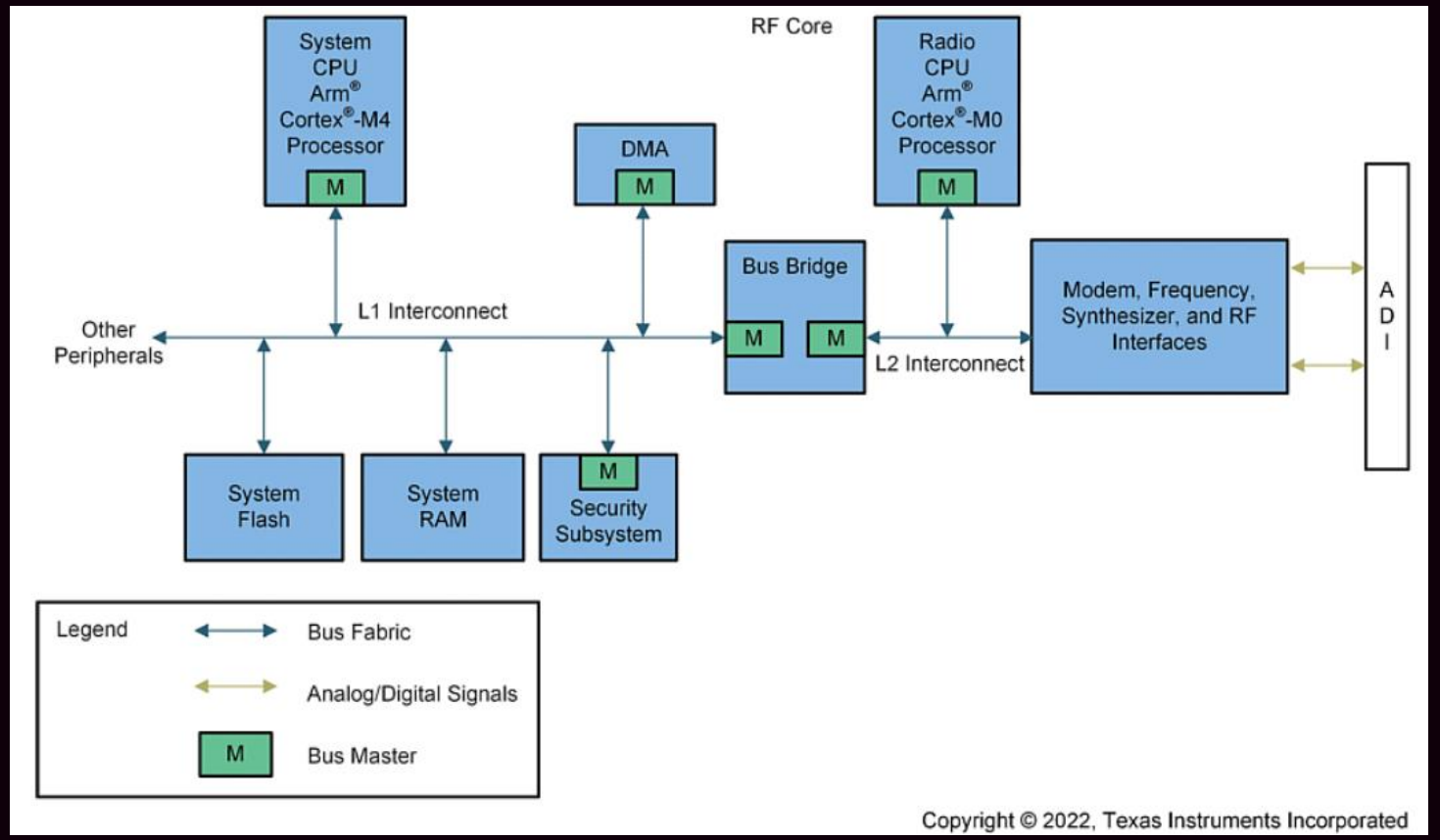
TI SmartRF Studio

- TI GUI tool to interact directly with radio
- Generates C headers that call radio APIs to be integrated in user code
- Proprietary RF modes are suspiciously flexible compared to your standard Bluetooth MCU…



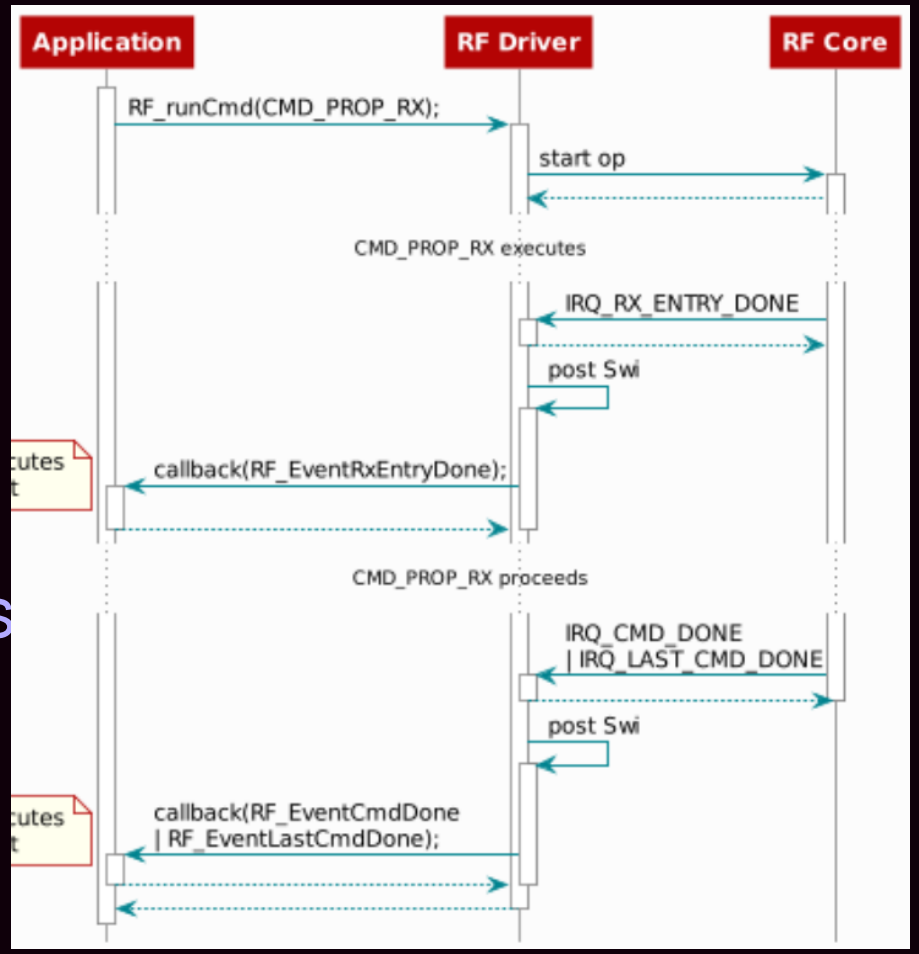
Versatile but Undocumented

- All of this configurability requires an incredibly versatile RF subsystem
- No documentation exists for the hardware
- “Only to be used through TI provided API”



How “TI Provided APIs” Interact w/ RF

- All driver code must exist in the SDK
  - Unlike other peripherals, RF uses a higher-level API system instead of direct MMIO driver
- API only passes messages between the RF Command Packet Engine (CPE)
- Uses a mailbox system, sending predefined commands to request various RF operations
  - Initialize Protocol, Set Frequency, Transmit, Receive, etc.

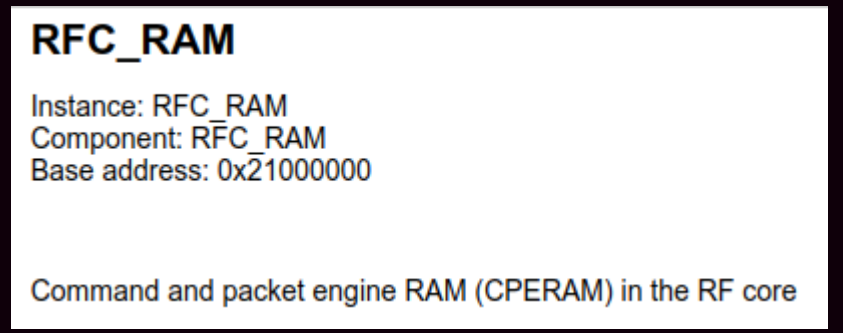


The Command Packet Engine (CPE)

- A Bonus CPU Core*!*
  - Can’t™ be programmed by the user
- Gateway to the rest of the RF Subsystem
- Designed as a dedicated processor for real-time management of the RF hardware and protocol stack without main CPU intervention
- CPU is an ARM Cortex-M0
  - Runs from its own private ROM
  - Has access to primary system bus
  - Also gets extra privileges in bus fabric for accessing RF specific MMIO ranges
- Wanted code execution on CPE to dump ROM for further analysis

Gaining Access to CPE

- CPE ROM is not mapped in CM4 address space
- CPE SRAM is mapped at 0x2100_0000 ☺
  - 4K of mostly random/garbage data
- Need to do some blind exploitation
- SimpleLink SDK contains "patches" which can be loaded into CPERAM to fix bugs or enable new features
  - Likely contains mechanism to load executable code



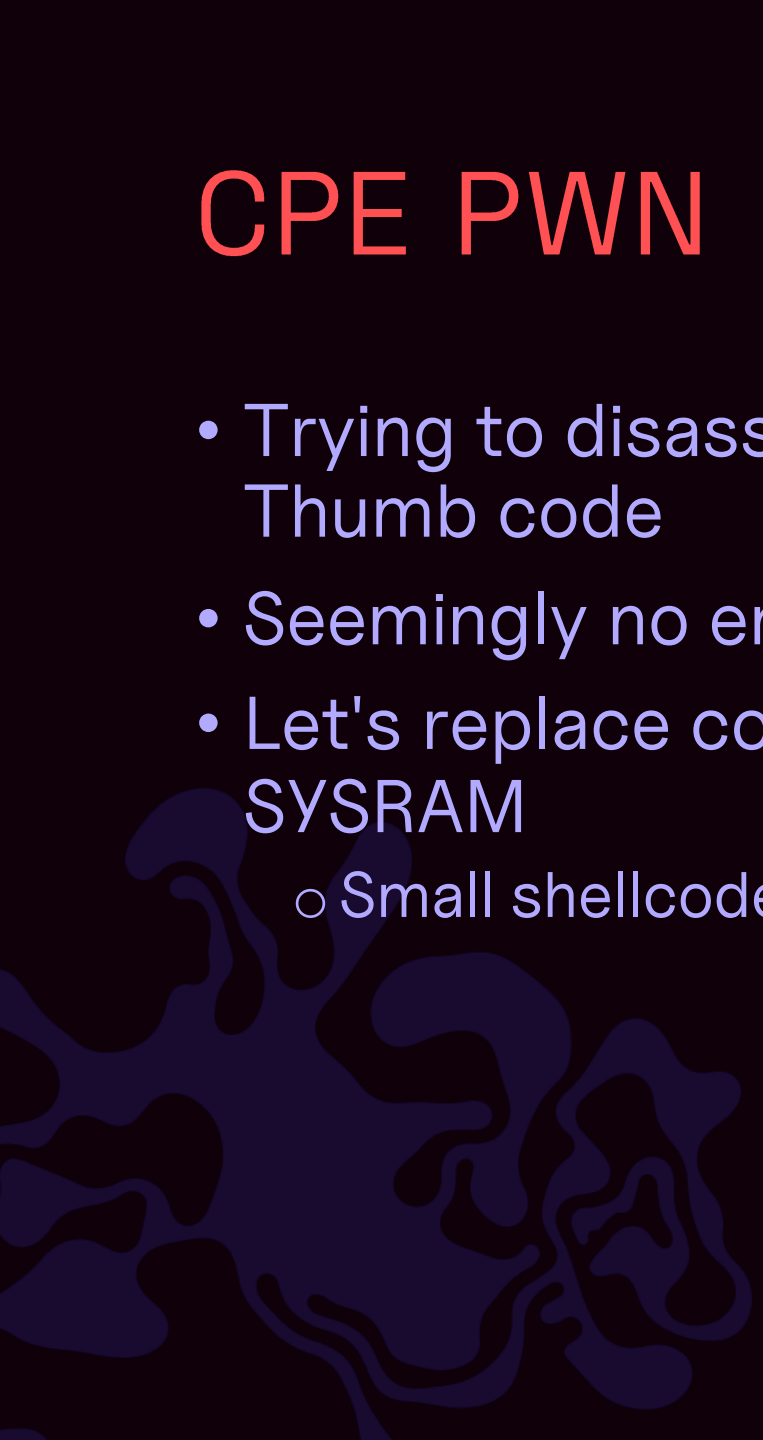
RF Patches

- Opaque binary blobs released by TI to add support for additional features
  - Bluetooth Co-existence
  - DSSS
  - Proprietary Mode OOK Modulation
- Loaded into CPE private SRAM after boot
- **These are powerful enough to allow TI to release new protocols for existing chips, without changing the ROM**
- Understanding the patches is key to allow us to do the same thing

```
73  CPE_PATCH_TYPE patchImageGenook[] =
74  {
75      0x21000569,
76      0x2100045d,
77      0x21000491,
78      0x21000495,
79      0x210004bd,
80      0x2100064d,
81      0x210006fd,
82      0x21000725,
83      0x2100052b,
84      0x210004f1,
85      0x21000767,
86      0x21000789,
87      0x4710b5f8,
88      0x460eb5f8,
89      0x25012100,
90      0x473004ad,
91      0x7803480a,
92      0xf80ff000,
93      0xd00b079b,
94      0x78204c12,
95      0xd00728ff,
96      0x702121ff,
97      0x240f490e,
98      0x43200224,
99      0x82c83160,
100     0xb5f8bdf8
```

CPE PWN

- Trying to disassemble patch blob yielded some regions of valid Thumb code
- Seemingly no encryption/signatures
- Let's replace code with a bunch of NOPs + jump to main SYSRAM
  - Small shellcode to copy chunks of CPE ROM to SYSRAM



CPE ROM

	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00000000	00 10 00 21 0D 01 00 00 DF 32 00 00 DF 32 00 00	. . . ! . . . . . 2 . . . 2 .
00000010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
00000020	00 00 00 00 00 00 00 00 00 00 00 00 DF 32 00 00	. . . . . 2 .
00000030	00 00 00 00 00 00 00 00 DF 32 00 00 DF 32 00 00	. . . . . 2 . . . 2 .
00000040	2B 0B 00 00 81 04 00 21 89 04 00 21 91 04 00 21	+ . . . . . ! . . . ! . . .
00000050	99 04 00 21 11 75 00 00 C3 32 00 00 A1 04 00 21	. . . ! . u . . . 2 . . . . .
00000060	A9 04 00 21 B1 04 00 21 D1 04 00 21 D9 04 00 21	. . . ! . . . ! . . . ! . . .
00000070	B9 04 00 21 C1 04 00 21 C9 04 00 21 C9 04 00 21	. . . ! . . . ! . . . ! . . .
00000080	C9 04 00 21 AF 7D 00 00 AF 7D 00 00 AF 7D 00 00	. . . ! . } . . . } . . . } .
00000090	00 F0 F8 F8 00 F0 3A F8 10 3A 02 D3 78 C8 78 C1	. . . . . : . . : . . x . x
000000A0	FA D8 52 07 01 D3 30 C8 30 C1 01 D5 04 68 0C 60	. . R . . . 0 . 0 . . . . h .
000000B0	70 47 1F B5 C0 46 C0 46 1F BD 10 B5 10 BD 70 47	p G . . . F . F . . . . . p
000000C0	00 00 00 00 01 01 03 05 08 0D 15 1F 2C 3C 4E 62	. . . . . , < N
000000D0	76 8A 9C AB B8 C1 C8 CB 00 00 00 00 01 01 02 03	v . . . . .
000000E0	05 08 0D 14 1C 26 32 3F 4C 58 64 6D 76 7C 80 82	. . . . . & 2 ? L X d m v | .
000000F0	00 23 00 24 00 25 00 26 10 3A 01 D3 78 C1 FB D8	. # . $. % . & . : . . x . .
00000100	52 07 00 D3 30 C1 00 D5 0B 60 70 47 00 F0 DE F8	R . . . 0 . . . . ` p G . . .

Code Exec As a Service (CEAaS)

- It would be nice to have an easy and reliable way to run arbitrary code on CPE without needing to load patches
- Found some undocumented mailbox API commands in the CPE ROM
- Including one `0x0811` which just calls a function pointer without any checks
- Thank you TI!

```
    break;
  case CMD_RUN_USER_FUNCTION:            // Undocumented!
    LOWORD(some_cmd_hdlr_set_flag) = 0;
    cmd_callback_fptr = (int (*)())rfcmd_com_run_user;
    break;
  case CMD_COUNTER_BRANCH:
```

```
usr_retcode = ((int (__fastcall *)(int))(*(_DWORD *)(pCpeCmdCopy + 16) | 1))(pCpeCmdCopy + 20);
```

CPE Patches

```
1  int startRx()
2  {
3    int result; // r0
4
5    if ( patchInd_startRx != 0xFF )
6      return (*(int (**)(void))(PATCH_VEC_ADDR_OFFSET + 4 * patchInd_startRx))();
7    if ( synth_is_powered != RX_MODE )
```

- Structure of CPE patches can now be understood by RE'ing ROM
- Majority of ROM functions check if patch is enabled at entry
  - Allows execution to be detoured to address written to patch table
  - Even functions like IRQ/NMI handlers are patchable
- Patches are powerful enough to effectively replace CPE ROM entirely
- Want to make our own patches now*!*
  - Created a custom 'toolchain' to compile CPE patches
  - Can reuse existing RFC mailbox system to talk to CPE custom firmware
- With access to CPE, we can reverse engineer the rest of the RF core

Part 2: Gaining Knowledge of the Chip

TER *over 9 years ago in reply to David Oswald* — TI_Guru**** 317180 points

Yes and no.

We support custom firmware for the radio MCU, more features will be added to the chip by providing patches to the existing code but we will not open this for customers. To write a patch requires knowledge of the chip far outside what we are going to include in the documentation.

0True

The RFE and MCE

```
#ifndef RFC_MCERAM_BASE
    #define RFC_MCERAM_BASE 0x21008000
#endif
```

- SDK contains 2 other types of patches
  - Referred to as "RFE" and "MCE" patches
  - These do not disassemble to ARM code, and look vastly different from CPE patches
  - Also get loaded into separate RFERAM/MCERAM addresses
    - Not listed in the TRM memory map
- Very little is documented about what these are
  - Acronym only occurs once in TRM seemingly by accident
  - The TRM doesn’t even give the full meaning of the acronym

MCE/RFE Override Entry

Table 25-20. Format of an MCE/RFE Override Mode Entry

Bit Index	Bit Field Name	Description
0–1	entryType	11: Firmware-defined parameter
2–3	entrySubType	01: MCE/RFE override mode
4	bMceCopyRam	If 1, copy the contents of the MDM ROM bank given by mceRomBank to RAM after MCE has completed setup.
5	bRfeCopyRam	If 1, copy the contents of the RFE ROM bank given by rfeRomBank to RAM after MCE has completed setup.
6	bMceUseRam	0: Run MCE from ROM 1: Run MCE from RAM
7–10	mceRomBank	MCE ROM bank to run from
11	bRfeUseRam	0: Run RFE from ROM 1: Run RFE from RAM
12–15	rfeRomBank	RFE ROM bank to run from
16–23	mceMode	Mode to send to MCE
24–31	rfeMode	Mode to send to RFE

A Lucky Break

- Found one patch in one SDK version was not exported "properly"
- Accidentally includes full assembly listing for the MCE patch source code
  - Including all headers ☺
- Reveals new instruction set
  - Assembly to opcode matching (for most instructions)
  - Fuzz the remaining opcode map
  - IO-space map for MCE DSP peripherals
  - A lot of interesting comments providing insight to how the modem works

```
;  mce_ram_bank.asm:    676 IIR_K4:
;  mce_ram_bank.asm:    677         ;; make a simple IIR   y[n] = y[n-1]3/4 + x[n]/4
;  mce_ram_bank.asm:    678         ;; r5 = y
;  mce_ram_bank.asm:    679         ;; first calculate y[n-1]*3/4
;  mce_ram_bank.asm:    680         mov  r5, r6            ; y[n-1] into r6
;  mce_ram_bank.asm:    681         sl0  2, r6             ; multiply by 4
;  mce_ram_bank.asm:    682         sub  r5,r6             ; sub x1 to get multiply by 3
;  mce_ram_bank.asm:    683         add  r2, r6            ; add new sample
;  mce_ram_bank.asm:    684         srx  2, r6             ; scale back to normal again
;  mce_ram_bank.asm:    685         mov  r6, r5            ; copy to r5
;  mce_ram_bank.asm:    686         jmp  HARD_DECISION        ;
;  mce_ram_bank.asm:    687 IIR_K8:
;  mce_ram_bank.asm:    688         ;; make a simple IIR   y[n] = y[n-1]7/8 + x[n]/8
;  mce_ram_bank.asm:    689         ;; r5 = y
;  mce_ram_bank.asm:    690         ;; first calculate y[n-1]*7/8
;  mce_ram_bank.asm:    691         mov  r5, r6            ; y[n-1] into r6
;  mce_ram_bank.asm:    692         sl0  3, r6             ; multiply by 8
;  mce_ram_bank.asm:    693         sub  r5,r6             ; sub x1 to get multiply by 7
;  mce_ram_bank.asm:    694         add  r2, r6            ; add new sample
;  mce_ram_bank.asm:    695         srx  3, r6             ; scale back to normal again
;  mce_ram_bank.asm:    696         mov  r6, r5            ; copy to r5
;  mce_ram_bank.asm:    697         jmp  HARD_DECISION        ;
;  mce_ram_bank.asm:    698 NO_IIR_FILTER:
;  mce_ram_bank.asm:    699         mov     r2, r6         ;
;  mce_ram_bank.asm:    700
```

The TopSM Architecture

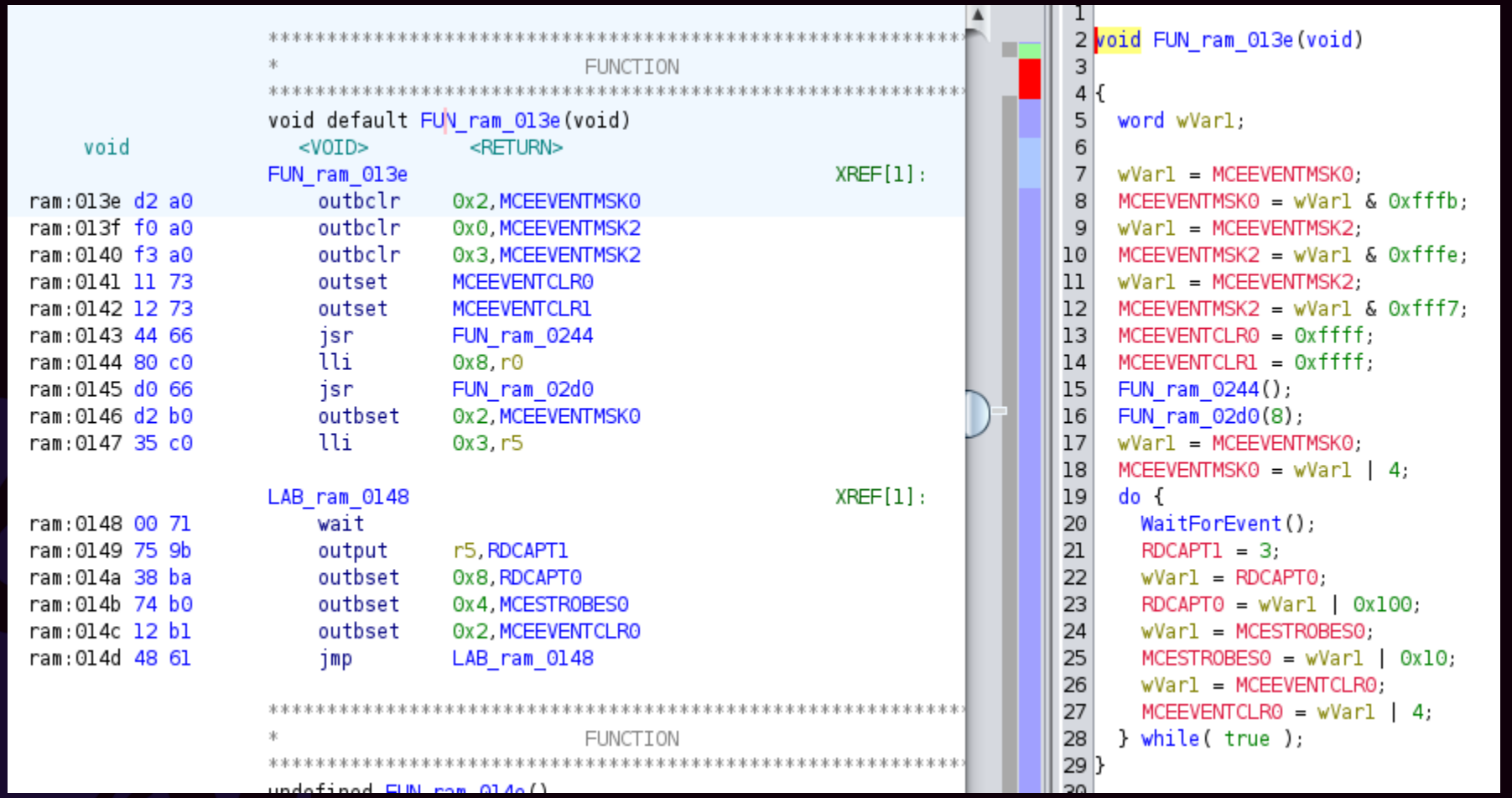
- TI calls this architecture TopSM
- Simple RISC CPU Instruction Set
  - 16-bit word-addressable architecture
  - 16 registers
  - Small internal hardware stack solely for subroutine support
  - Read-only data/instruction bus (10-bit address)
    - Point to 1 of 8 ROM banks, selected by CPE
    - Can also boot from dedicated 2KiB RAM bank (used by patches)
    - Seemingly no instruction to write to RAM ☹
      - Alternatively, bus locks up if writing to RAM while running from RAM
  - I/O bus (8-bit address)
    - Allows control of a subset of the RF hardware in the chip
- Interacts with RF analog blocks and DSP accelerator peripherals

RFE/MCE Reverse Engineering

- Want to analyze the RFE and MCE ROMs
  - Seemingly no way to read out the ROMs from the TopSM, as running code from RAM disconnects the ROM from the bus
- Luckily, TI built ROM-dumping functionality right into the hardware*!*
  - There's also another undocumented CPE command for that

```
// CMD_TOPSM_COPY: Radio Copy TOPsm ROM-to-RAM Command
PACKED_ALIGNED_TYPEDEF_STRUCT
{
  rfOpCmd_t   rfOpCmd;              // radio command common structure
  int8        mceBank;              // W: ROM bank number for the MCE (0-5). Negative: Do not copy MCE ROM.
  int8        rfeBank;              // W: ROM bank number for the RFE (0-5). Negative: Do not copy RFE ROM.
  uint16      mceStopAddr;          // W: Last 16-bit address top copy for MCE ROM. 0: Copy entire ROM
  uint16      rfeStopAddr;          // W: Last 16-bit address top copy for RFE ROM. 0: Copy entire ROM
} rfOpCmd_TopsmCopy_t;
```

TopSM Ghidra Plugin



```
                        ****************************************************************
                        *                          FUNCTION                            *
                        ****************************************************************
                        void default FUN_ram_013e(void)
     void                 <VOID>          <RETURN>
                        FUN_ram_013e                                    XREF[1]:
ram:013e d2 a0              outbclr     0x2,MCEEVENTMSK0
ram:013f f0 a0              outbclr     0x0,MCEEVENTMSK2
ram:0140 f3 a0              outbclr     0x3,MCEEVENTMSK2
ram:0141 11 73              outset      MCEEVENTCLR0
ram:0142 12 73              outset      MCEEVENTCLR1
ram:0143 44 66              jsr         FUN_ram_0244
ram:0144 80 c0              lli         0x8,r0
ram:0145 d0 66              jsr         FUN_ram_02d0
ram:0146 d2 b0              outbset     0x2,MCEEVENTMSK0
ram:0147 35 c0              lli         0x3,r5

                        LAB_ram_0148                                    XREF[1]:
ram:0148 00 71              wait
ram:0149 75 9b              output      r5,RDCAPT1
ram:014a 38 ba              outbset     0x8,RDCAPT0
ram:014b 74 b0              outbset     0x4,MCESTROBES0
ram:014c 12 b1              outbset     0x2,MCEEVENTCLR0
ram:014d 48 61              jmp         LAB_ram_0148

                        ****************************************************************
                        *                          FUNCTION                            *
                        ****************************************************************
                        undefined FUN_ram_014e()
```

```
1
2  void FUN_ram_013e(void)
3
4  {
5    word wVar1;
6
7    wVar1 = MCEEVENTMSK0;
8    MCEEVENTMSK0 = wVar1 & 0xfffb;
9    wVar1 = MCEEVENTMSK2;
10   MCEEVENTMSK2 = wVar1 & 0xfffe;
11   wVar1 = MCEEVENTMSK2;
12   MCEEVENTMSK2 = wVar1 & 0xfff7;
13   MCEEVENTCLR0 = 0xffff;
14   MCEEVENTCLR1 = 0xffff;
15   FUN_ram_0244();
16   FUN_ram_02d0(8);
17   wVar1 = MCEEVENTMSK0;
18   MCEEVENTMSK0 = wVar1 | 4;
19   do {
20     WaitForEvent();
21     RDCAPT1 = 3;
22     wVar1 = RDCAPT0;
23     RDCAPT0 = wVar1 | 0x100;
24     wVar1 = MCESTROBES0;
25     MCESTROBES0 = wVar1 | 0x10;
26     wVar1 = MCEEVENTCLR0;
27     MCEEVENTCLR0 = wVar1 | 4;
28   } while( true );
29 }
30
```

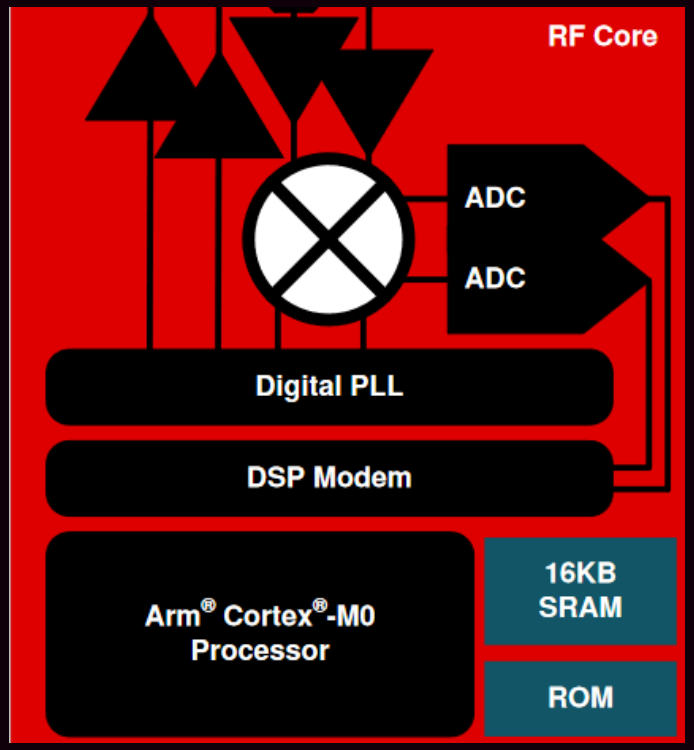
So what actually is the RFE/MCE?

- Now that we have all these tools and ROM dumps, we can actually answer the question: *What is the RFE and MCE?*
- First of all: What do they actually stand for?
  - RFE: RF Engine
  - MCE: Modem Command Engine
- RFE is responsible for real-time control of the RF Front End
  - Configuration
  - RSSI Estimation
  - Automatic Gain Control
- MCE is responsible for real-time control of the Modem block
  - Handles some configuration of modem settings
  - On-the-fly configuration of the various DSP accelerator blocks
  - Sequencing the various stages of packet transmission/reception

A Quick Review

- Full code execution on the CPE with ROM to map various patch locations to code to enable full access to RF MMIO registers
- Instruction map of MCE/RFE CPUs & ROMs to understand the default configuration of the RF hardware
- Register map for MCE
  - Very useful as this is where the interesting DSP blocks are that can be used to implement new protocols/features
- Good guesses for the rest of the RF subsystem based on CPE/RFE reverse engineering and analyzing patents

Part 3: The RF Subsystem



RF ”Analog” Section

- Based on All-Digital "ADPLL" architecture
  - Digital LO
  - Digital Mixer + IQ ADC chain
  - Digital Transmitter modulator
- Tries to move as much of the RF subsystem into digital as possible
  - Reduced cost, complexity, and variability vs. traditional analog architecture
- Most of the processing occurs in DSP hardware accelerators, coordinated by a real-time processor (the MCE)
- Quite complex, but luckily lots of information available in patents*!*
  - US 9,473,155 B2
  - US 8,045,670 B2

RF "Analog" Section

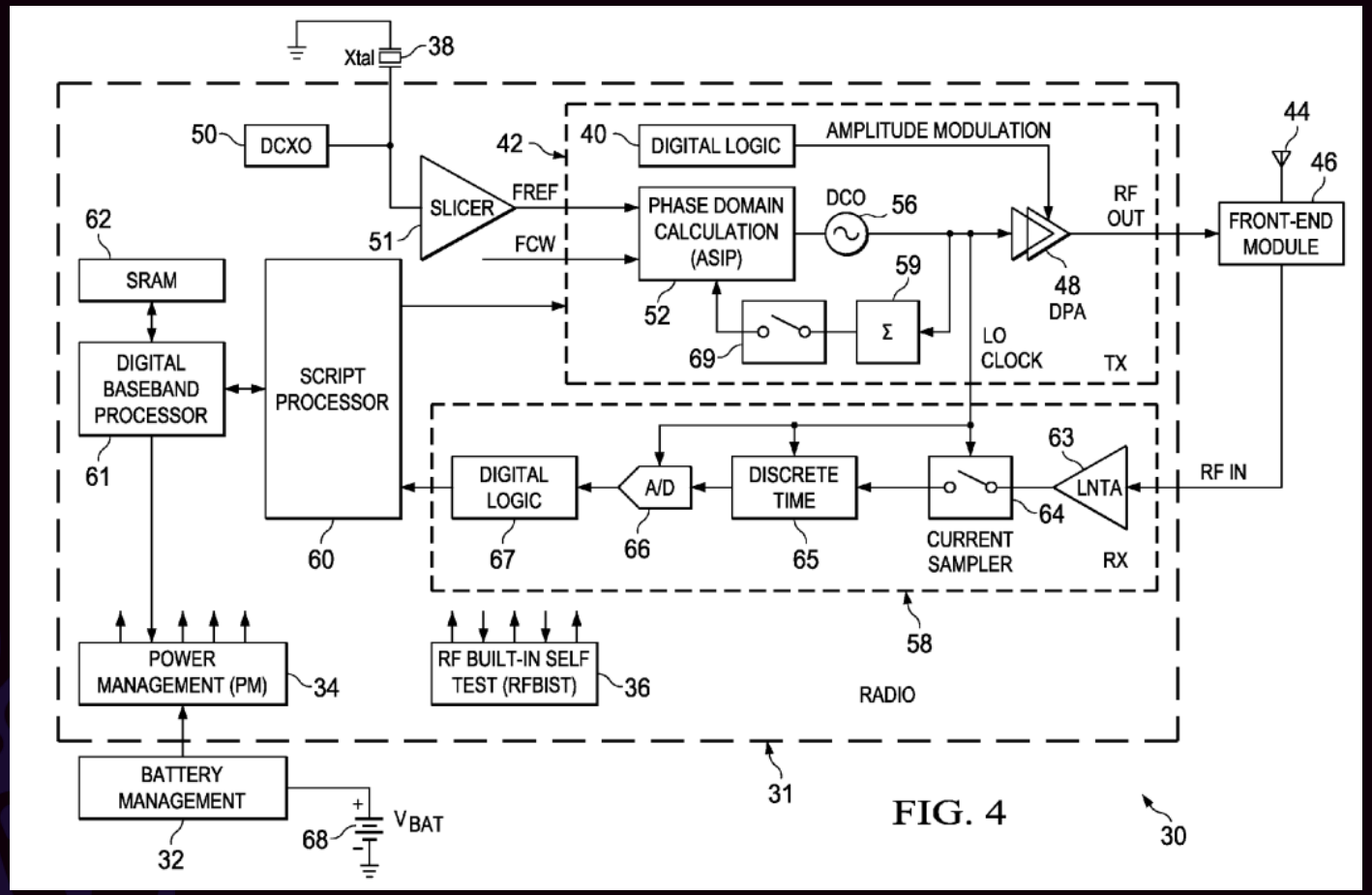


FIG. 4

RF "Analog" Section

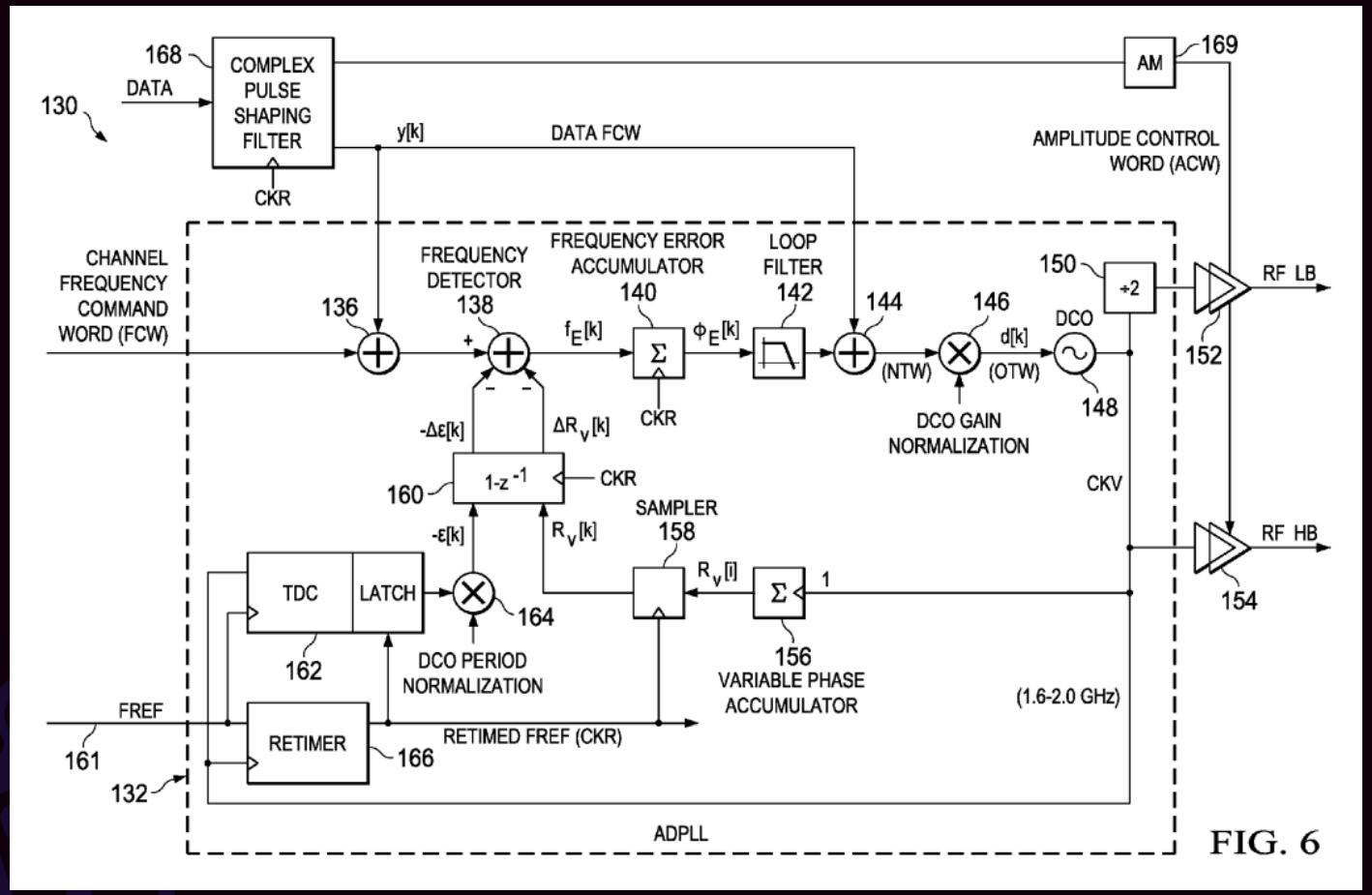
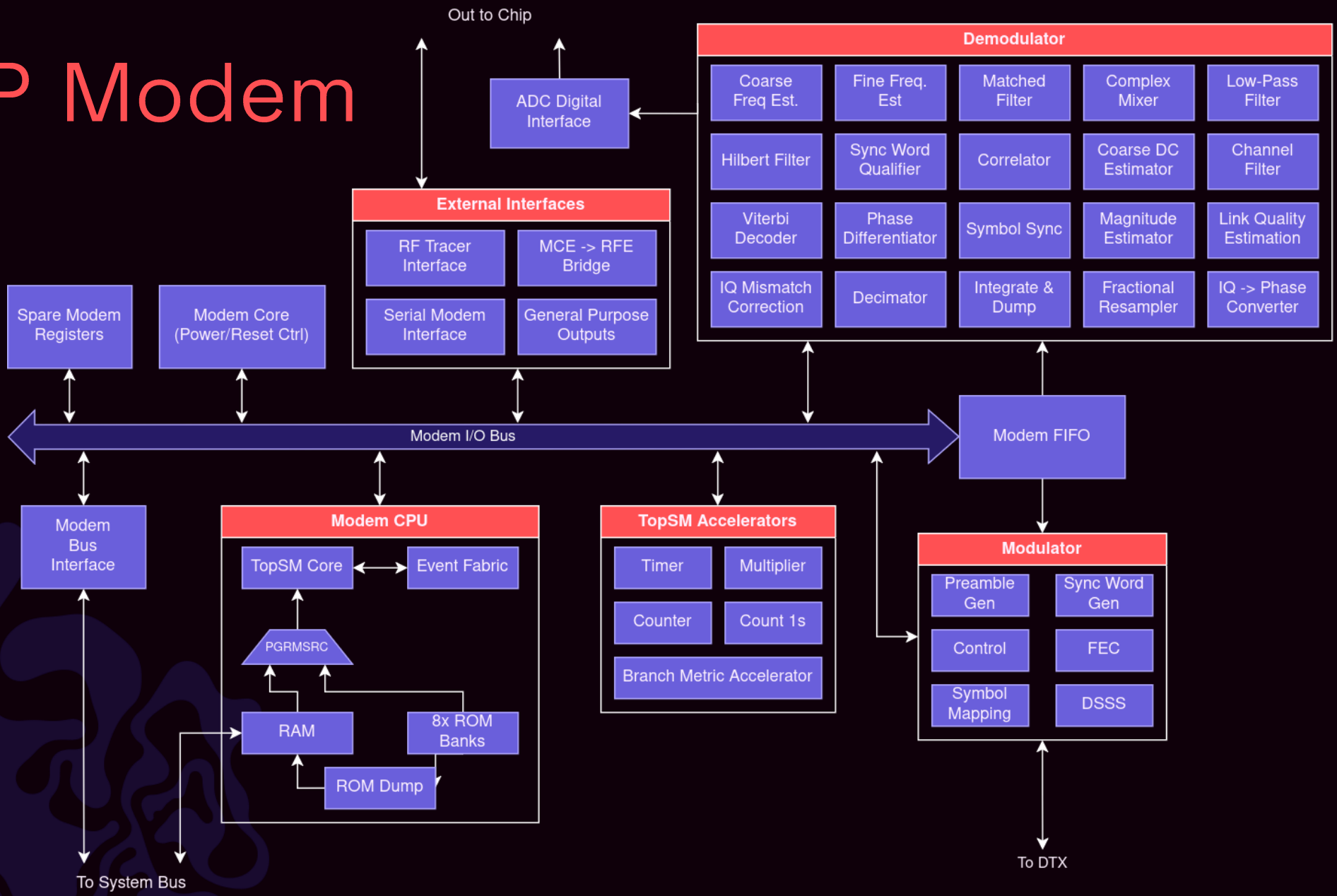


FIG. 6

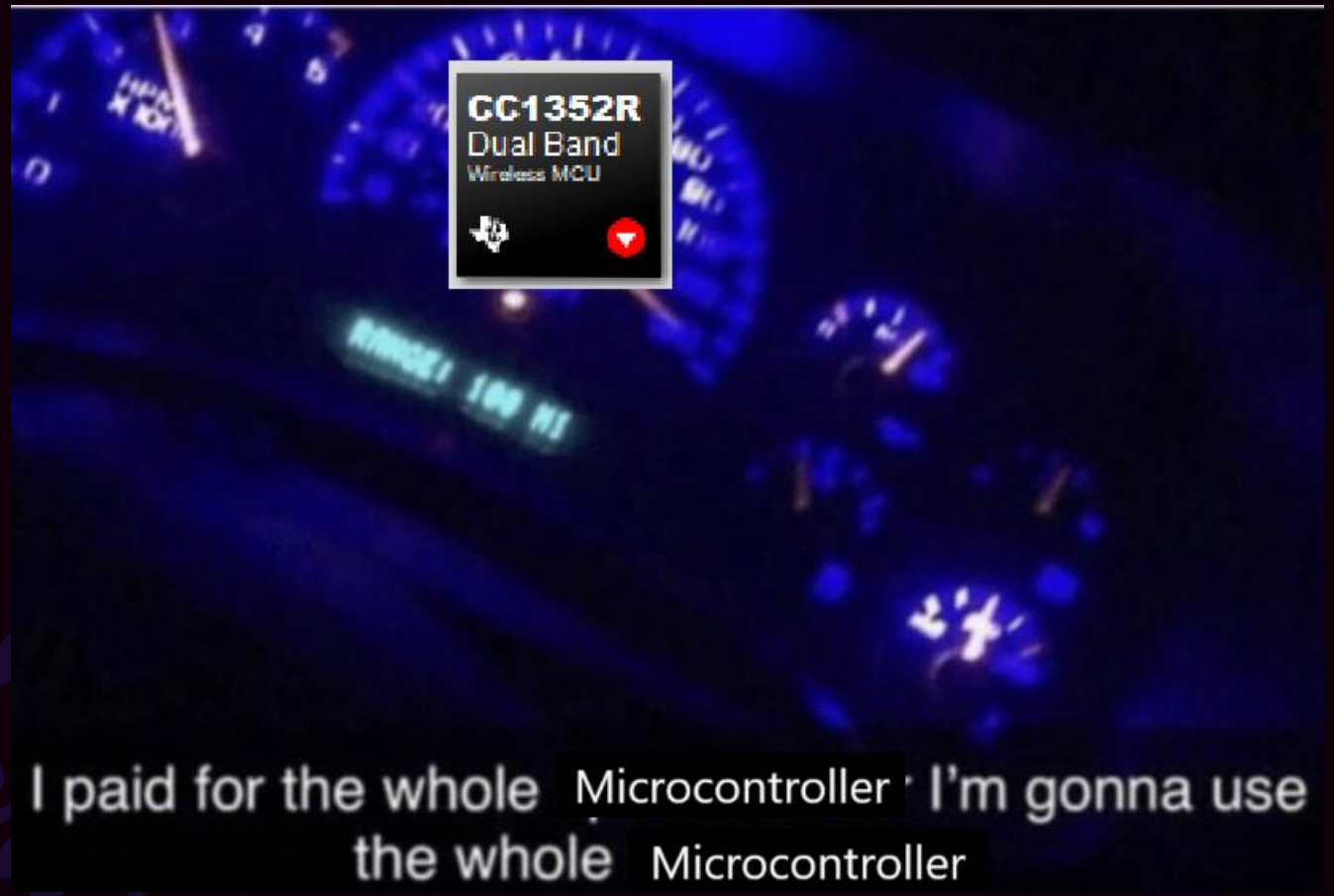
DSP Modem


Out to Chip
ADC Digital Interface
Demodulator
Coarse Freq Est.
Fine Freq. Est
Matched Filter
Complex Mixer
Low-Pass Filter
Hilbert Filter
Sync Word Qualifier
Correlator
Coarse DC Estimator
Channel Filter
Viterbi Decoder
Phase Differentiator
Symbol Sync
Magnitude Estimator
Link Quality Estimation
IQ Mismatch Correction
Decimator
Integrate & Dump
Fractional Resampler
IQ -> Phase Converter
External Interfaces
RF Tracer Interface
MCE -> RFE Bridge
Serial Modem Interface
General Purpose Outputs
Spare Modem Registers
Modem Core (Power/Reset Ctrl)
Modem I/O Bus
Modem FIFO
Modem Bus Interface
Modem CPU
TopSM Core
Event Fabric
PGRMSRC
RAM
8x ROM Banks
ROM Dump
TopSM Accelerators
Timer
Multiplier
Counter
Count 1s
Branch Metric Accelerator
Modulator
Preamble Gen
Sync Word Gen
Control
FEC
Symbol Mapping
DSSS
To System Bus
To DTX

CPE ROM Masking

- There are several chips in the SimpleLink family, all with slightly different features enabled
  - Some support only Bluetooth, others enable IEEE 802.15.4 support, some support all of the available protocols
- Each variant is very likely the same die, just with features fused off
- We noticed some jumps in CPE ROM that go to pages of all 0xFF's
  - But on other variants, code exists there for a certain supported protocol
- CPE ROM checks "supported protocols" OTP value before jumping
  - Patches can override these checks
  - As a secondary protection measure, ROM pages for unsupported protocols are disabled in HW based off OTP bitmask

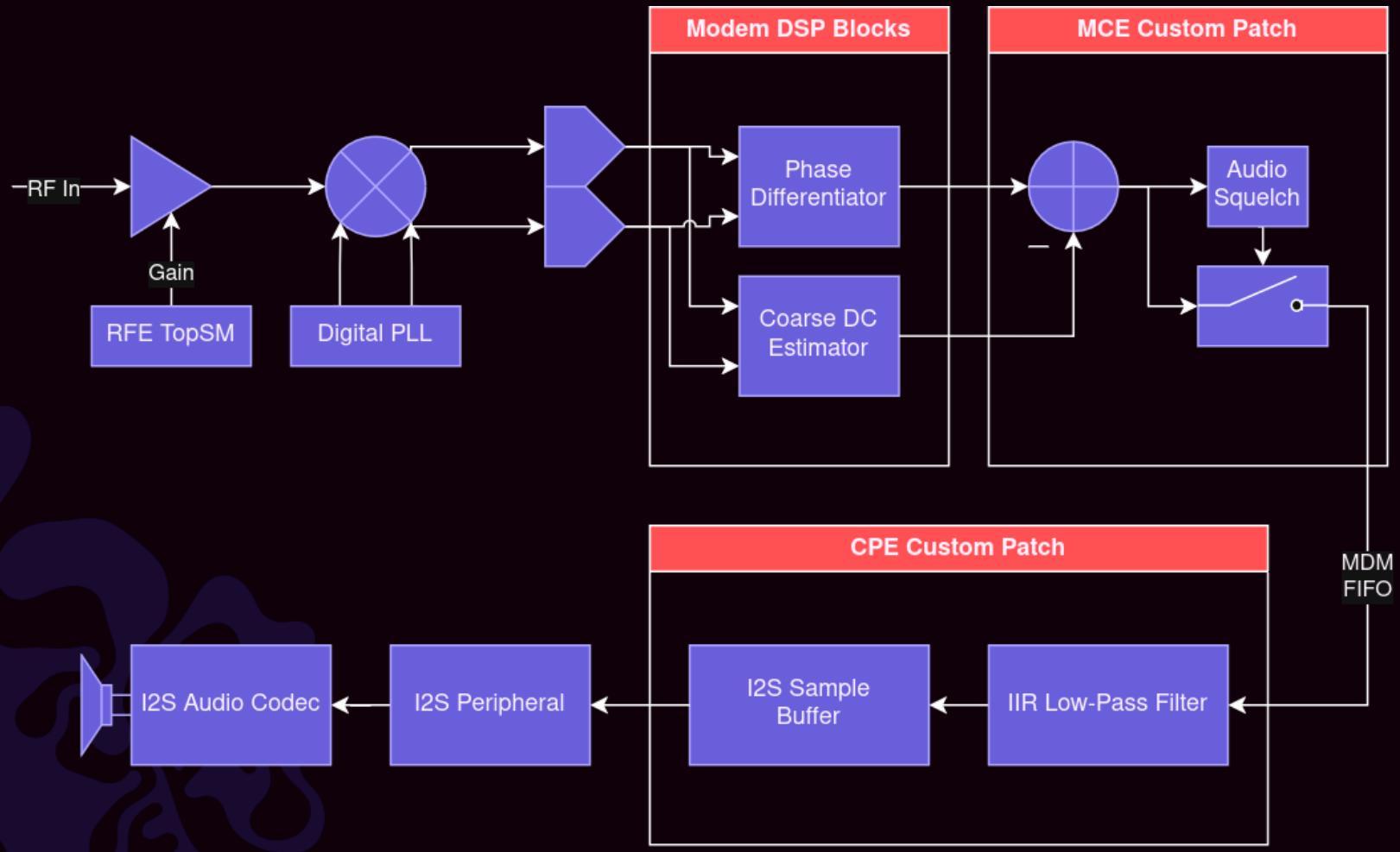
Part 4: Custom RF Firmware



Creating Custom RF Patches

- We can now implement custom protocols in MCE!
- For this example, we are going to implement a Narrowband FM Demodulator with the SimpleLink
  - This is an *analog* modulation scheme
  - The SimpleLink does not natively support any form of analog modulation
  - Leverage IQ receiver architecture to demodulate

Custom NBFM Patch



TopSM Toolchain

- MCE patches *could* be written by hand
  - Easier to work in native assembly
- Wrote a TopSM assembler toolchain for developing MCE patches
  - Even managed to catch a syntax error in TI's MCE patch that was missed

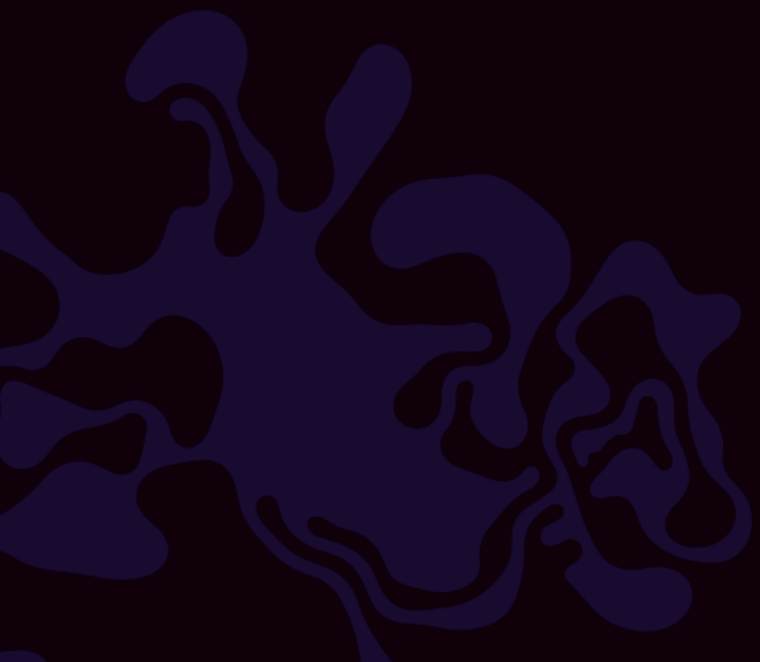
GitHub: https://github.com/rjp5th/beyond-ble-tools

Future Work

- Explore full capabilities of the device
  - Frequency tuning range, analog front-end bandwidth, etc.
- Continue RE of Modem DSP Blocks
  - Custom TX
- Path to ultra-low cost single-chip SDR
  - RF core can pass raw IQ samples to CM4 for more advanced processing
- Other chips in SimpleLink family
  - CC32xx Wi-Fi transceiver

Putting it All Together

- Demo of NBFM receiver*!*



Thank You!

Q&A