Apple C1 Tales

41con 2025

lambda & turbocooler

An intro to Leda

- Apple C1/Initium/Sinope/Leda/c4000 is Apple’s new in-house baseband released with the iPhone 16e
  - Released ~3 months ago
- Leda is a mixture of ARCv2/ARMv7/ARM64e cores
- Our presentation will mostly focus on Leda’s CPS(Cellular Processor Stack)
  - Interesting for attackers since it seems to handle most GSM logic & ARI
- Runs OSCKit on the CPS core, RTKit on other non ARC cores

Architecture overview

CDP UL ARM64

CPS ARM64

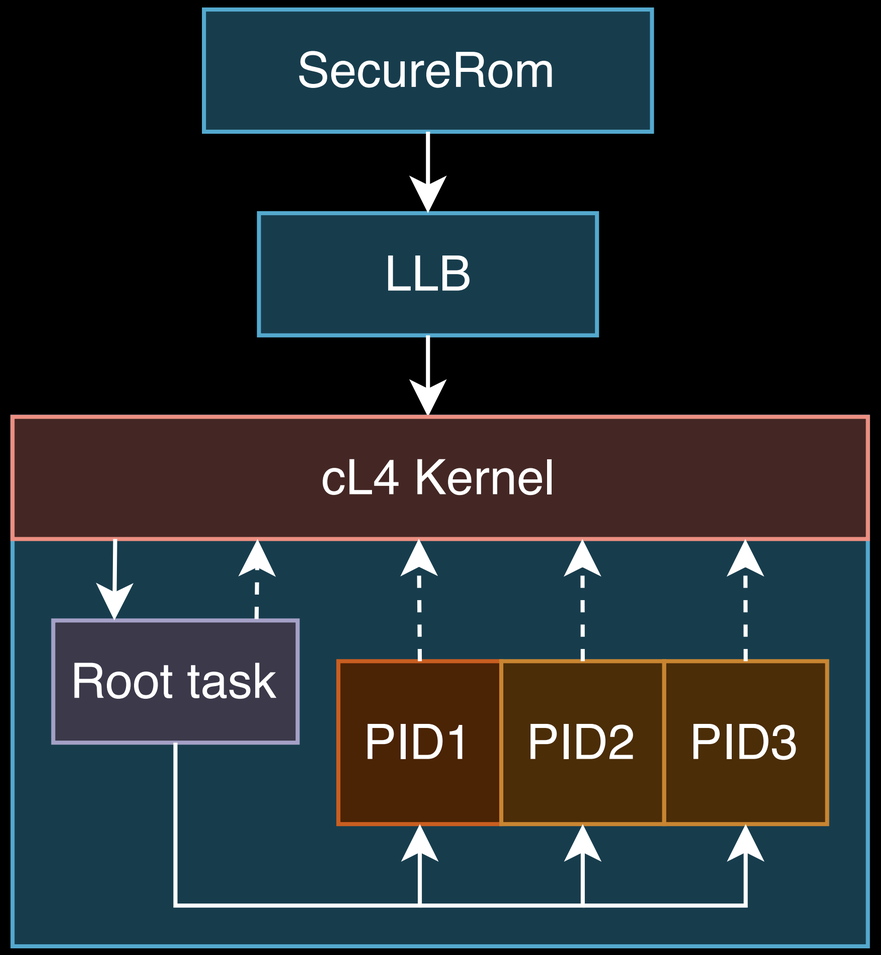
L1CS ARM64

CDP DL ARM64

and N ARC cores...

CDP Host ARM32

How CPS boots



- Basic hardware is initialised in LLB.
- Uses FTAB file format to separate components
- LLB then loads RKOS image on CPS which will then start the cL4 kernel
- cL4 kernel will start the OSCKit root task
- OSCKit root task will start 3 different processes

Peeking into the firmware

- Leda's firmware can be obtained from a 16e IPSW in path Firmware/c4000v59/Release/patched/ftab.bin
- 46 entries

```
struct FileEntry {
    char tag[4];
    u32 offset;
    u32 length;
    u32 pad;
};

struct FTAB_Header {
    u32 unk[4];
    u32 manifest_off;
    u32 manifest_sz;
    u32 unk2[2];
    char magic[8];
    u32 num_entries;
    u32 zero;
    FileEntry entries[num_entries];
};
```

Modified ImHex syntax based on @matteyeux's work

A few we understand:

- illb
- rkos
- rcpi
- l1cs
- cdpd
- cdph
- cdpu
- GNS1
- ARC?

Loading a fwsg file

- rkos image comes as a fwsg file.
- fwsg files contain an uuid header and a list of segments at the end
- LLB will load the fwsg segments into DRAM and pass cL4 a map of segment names and their vaddr/phys address alongside a handoff structure
- All RO segments will be loaded adjacent in DRAM so that cL4 can lock down the range with AMCC

```
typedef struct {
    uint64_t vaddr;
    uint64_t phys_addr;
    uint32_t seg_sz;
    char seg_name[8];
    uint8_t cluster_id;
    uint8_t padding[3];
} rtkit_segment;
```

```
typedef struct {
    char magic[4]; // uuid magic
    int version;
    int off1;
    int off2;
    char maybe_uuid[16];
    int unk4;
    int unk5;
    int patchbay_off;
    int patchbay_sz;
    int unk6;
    uint64_t physaddr; // will be used early in cl4 boot to set sp
} rtkit_uuid_header;
```

An RKOS example

- Flag == 4 means the segment is read only
- The following patch bays are present in it:

GMdb	Segment map DB	**BtAr**	CPS Boot Args
GMsz	Segment map SZ	**RSCD**	CDPD ASLR Slide
SLID	CPS ASLR Slide	**RSCU**	CPDU ASLR Slide
RSL1	L1C ASLR Slide	**DEVf**	Dev Fuse

```
     virt addr          file off     file sz    mem sz  flags     name
 0: ffffffff00000000 00000000 0000e19f 0000e19f 00000004 __TEXT
 1: ffffffff00010000 00010000 00009928 00011f20 00000000 __DATA
 2: 000000007fffc000 0001c000 000127e4 000127e4 00000004 r.__TEXT
 3: 0000000080010000 00030000 00002ae0 00029588 00000000 r.__DATA
 4: 0000000002000000 00034000 01b5a658 01b5a658 00000004 m.__TEXT
 5: 0000000003b5c000 01b90000 0014e8e8 001c5728 00000000 m.__DATA
 6: 0000000003d24000 01ce0000 000cc8d0 000cc8d0 00000000 m.__DATA
 7: 0000000003df4000 01db0000 0000d478 0000d478 00000000 m.APPCON
 8: 0000000003e04000 01dc0000 00002aec 00002aec 00000000 m.OSCSH2
 9: 0000000003e08000 01dc4000 00000031 00000031 00000000 m.OSCSHR
10: 0000000003e0c000 01dc8000 0048b498 0276d409 00000000 m.OSC_01
11: 000000000657c000 02254000 008c80c0 01837db0 00000000 m.OSC_02
12: 0000000007db4000 02b20000 00020110 00025528 00000000 m.OSC_03
13: 0000000007ddc000 02b44000 001b4188 0071b03c 00000000 m.OSC_04
14: 000000000084f8000 02cfc000 00002b70 00006708 00000000 m.OSC_05
15: 0000000008500000 02d00000 0000b578 00019728 00000000 m.OSC_06
16: 000000000851c000 02d0c000 00004000 00004000 00000000 m.__PW_P
17: 0000000008520000 02d10000 00006200 00006200 00000000 m.__MHOS
18: 0000000008528000 00000000 00000000 00400000 00000000 m.__RF_I
19: 0000000008928000 02d18000 00002000 00002000 00000000 m.__IOSM
20: 000000000892c000 02d1c000 00004000 00004000 00000000 m.__GTI_
21: 0000000008930000 02d20000 000004c0 000004c0 00000000 m.__CPS_
22: 0000000008934000 02d24000 00004000 00004000 00000000 m.__CPS_
23: 0000000008938000 02d28000 000028b8 000028b8 00000000 m.__EXPR
24: 000000000893c000 02d2c000 00004000 00004000 00000000 m.__RT_S
25: 0000000008940000 02d30000 00004000 00004000 00000000 m.__SENS
26: 0000000008944000 02d34000 00004000 00004000 00000000 m.__SAH_
27: 0000000008948000 02d38000 00100000 00100000 00000000 m.__UE_C
28: 0000000008a48000 02e38000 0005d944 0005d944 00000004 m.__RO_D
29: 0000000008aa8000 02e98000 0005a300 0005a300 00000000 m.__DRAM
30: 0000000008b04000 02ef4000 0000bf28 0000bf28 00000000 m.__DRAM
31: 0000000008b10000 02f00000 00006640 00006640 00000000 m.__CPS_
32: 0000000008b18000 00000000 00000000 007293f8 00000000 m.__SHMS
33: 0000000009244000 00000000 00000000 0008dc74 00000000 m.__SHMC
34: 000000000092d4000 00000000 00000000 00a40000 00000000 m.__PBM_
35: 0000000009d14000 00000000 00000000 000e0000 00000000 m.__MSG_
36: 0000000009df4000 00000000 00000000 00060000 00000000 m.__MSG2
37: 0000000009e54000 02f08000 00448000 00448000 00000000 m.__PS_F
38: 000000000a29c000 00000000 00000000 02100010 00000000 m.__L1ME
39: 000000000c3a0000 00000000 00000000 02f00000 00000000 m.__TAGG
40: 000000000f2a0000 03350000 00020000 00020000 00000000 m.__MC_S
41: 000000000f2c0000 00000000 00000000 00640000 00000000 m.__MW_M
42: 000000000f900000 03370000 000099d0 0009fe80 00000000 m.__2G_D
43: 000000000f9a0000 0337c000 001d5cc2 001d5cc2 00000004 m.__2G_C
44: 000000000fb78000 00000000 00000000 01800000 00000000 m.__CMSG
45: 0000000011378000 03554000 008d5448 008d5448 00000000 m.__RO_N
46: 0000000011c50000 03e2c000 000b4db4 000b4db4 00000000 m.__L2DP
47: 0000000011d08000 03ee4000 00573620 00573620 00000000 m.__NC_L
48: 000000001227c000 04458000 00000870 00000870 00000000 m.__RW_C
49: 0000000012280000 0445c000 00000001 00000001 00000000 m.__SRAM
50: 0000000012284000 04460000 00002840 00002840 00000000 m.__CPSI
51: 0000000012288000 04464000 00007a40 00007a40 00000000 m.__SRAM
```

An intro to OSCKit

- Has the same name as an open sound control swift library
- Mixture of cL4 and rtkit (aka rtkcl4)
- After LLB loads the image execution will continue onto cL4
- After cL4 setup is done it will start the root task at the start of section r.__TEXT

cL4 on Leda

- Leda’s cL4 build enables Async notification capabilities
  - This allows TCBs to receive asynchronous notification from cL4. It also specifies critical sections of code which must be run entirely before another async event is delivered
  - In RKOS this is done by calling UtaOsCriticalSectionCreate
- OSCKit RPC is based on cL4 notifications

OSCKit Root task

- Initialises processes as described by the APPCON section
  - Each process runs in its own address space
  - Initialises IPC table via L4_Cap_Mint_Notify
  - Creates mappings for MMIO registers they need to access
  - Loads segments into each process based on APPCON rules
- Implements a pagetracker on top of cL4 mmu apis the state of which gets handed off to pid 1 via L4_Cap_Move
- ASLR slides for processes derived from a PAC’d ptr

APPCON Segment

```
→  fwsg-helper python3 appcon-parser.py ../rkos
Dumping APPCON @ 1db0000 sz: d478
using image base: 2004000
main trampoline @ 0x200d554
thread trampoline @ 0x200d594
irq trampoline @ 0x200d574

Parsing pid 1 with flags 2000030
IRQs:
0x10000
...
0x10193
MMIOs:
PID 1 inherits root task mappings
entry point: 2004000

Parsing pid 2 with flags 2000000
IRQs:
0x1012e
...
0x10194
MMIOs:
base: 407fc000 sz: 2000
base: 40c80000 sz: 41000
base: 43880000 sz: 40008
base: dff00000 sz: 100000
base: 4a8a0000 sz: 20000
base: 4a900000 sz: 40000
base: 4a800000 sz: b000
base: 4ab40000 sz: 8000
heap def: 6580048
entry point: 20c5254

Parsing pid 3 with flags 2000000
IRQs:
0x1012c
...
0x10195
MMIOs:
base: 407fc000 sz: 2000
base: 40c80000 sz: 41000
base: 43880000 sz: 40008
base: dff00000 sz: 100000
heap def: 7de0000
entry point: 20c5330
```

```
client_pid: 2 server_pid: 1 channel_id: 2 flags:0 shm_sz:10000
client_pid: 2 server_pid: 1 channel_id: 3 flags:0 shm_sz:10000
client_pid: 3 server_pid: 1 channel_id: 1 flags:0 shm_sz:10000
client_pid: 3 server_pid: 1 channel_id: 2 flags:0 shm_sz:10000
client_pid: 3 server_pid: 1 channel_id: 3 flags:0 shm_sz:10000
client_pid: 1 server_pid: 2 channel_id: 1 flags:0 shm_sz:0
client_pid: 1 server_pid: 2 channel_id: 2 flags:0 shm_sz:0
client_pid: 1 server_pid: 2 channel_id: 3 flags:0 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 1 flags:0 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 2 flags:0 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 3 flags:0 shm_sz:0
client_pid: 1 server_pid: 3 channel_id: 1 flags:0 shm_sz:0
client_pid: 1 server_pid: 3 channel_id: 2 flags:0 shm_sz:0
client_pid: 1 server_pid: 3 channel_id: 3 flags:0 shm_sz:0
client_pid: 2 server_pid: 3 channel_id: 1 flags:0 shm_sz:0
client_pid: 2 server_pid: 3 channel_id: 2 flags:0 shm_sz:0
client_pid: 2 server_pid: 3 channel_id: 3 flags:0 shm_sz:0
client_pid: 2 server_pid: 1 channel_id: 4 flags:1 shm_sz:10000
client_pid: 2 server_pid: 1 channel_id: 5 flags:1 shm_sz:10000
client_pid: 2 server_pid: 1 channel_id: 6 flags:1 shm_sz:10000
client_pid: 3 server_pid: 1 channel_id: 4 flags:1 shm_sz:0
client_pid: 3 server_pid: 1 channel_id: 5 flags:1 shm_sz:0
client_pid: 3 server_pid: 1 channel_id: 6 flags:1 shm_sz:0
client_pid: 1 server_pid: 2 channel_id: 4 flags:1 shm_sz:8000
client_pid: 1 server_pid: 2 channel_id: 5 flags:1 shm_sz:0
client_pid: 1 server_pid: 2 channel_id: 6 flags:1 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 4 flags:1 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 5 flags:1 shm_sz:0
client_pid: 3 server_pid: 2 channel_id: 6 flags:1 shm_sz:0
client_pid: 1 server_pid: 3 channel_id: 4 flags:1 shm_sz:8000
client_pid: 1 server_pid: 3 channel_id: 5 flags:1 shm_sz:0
client_pid: 1 server_pid: 3 channel_id: 6 flags:1 shm_sz:0
client_pid: 2 server_pid: 3 channel_id: 4 flags:1 shm_sz:10000
client_pid: 2 server_pid: 3 channel_id: 5 flags:1 shm_sz:10000
client_pid: 2 server_pid: 3 channel_id: 6 flags:1 shm_sz:10000
client_pid: 3 server_pid: 2 channel_id: 14 flags:1 shm_sz:2c000
client_pid: 3 server_pid: 2 channel_id: 15 flags:1 shm_sz:8000
client_pid: 3 server_pid: 2 channel_id: 16 flags:1 shm_sz:8000
client_pid: 2 server_pid: 3 channel_id: 14 flags:1 shm_sz:2c000
client_pid: 2 server_pid: 3 channel_id: 15 flags:1 shm_sz:8000
client_pid: 2 server_pid: 3 channel_id: 16 flags:1 shm_sz:8000
client_pid: 1 server_pid: 2 channel_id: 8 flags:1 shm_sz:0
client_pid: 2 server_pid: 1 channel_id: 8 flags:1 shm_sz:0
client_pid: 1 server_pid: 2 channel_id: 7 flags:1 shm_sz:0
```

```
[ROOTTASK] create_process, idx 1
[ROOTTASK] PAC key for pid 2: lo=1585a1cf28441c9d hi=8b17ea269785ca17
[ROOTTASK] pid 2: allocating 196000 bytes of state, 352512 bytes of caps, 622592 bytes of tables
[ROOTTASK] mapping 360448 bytes into roottask took up 0 tables, 22 frameish, 22 cap slots
[ROOTTASK] mapping 196608 bytes into roottask took up 0 tables, 12 frameish, 12 cap slots
[ROOTTASK] map_segment:   __TEXT va=       0x2000000 size=0x1b5c000    shared r-x
[ROOTTASK]        pa=0xc0028000 .. 0xc1b84000
[ROOTTASK] map_segment:   __DATA va=       0x3b5c000 size=0x1c8000 nonshared rw- needs-copy
[ROOTTASK]        pa=0xd0ec4000 .. 0xd108c000
[ROOTTASK] map_segment:   __DATA va=       0x3d24000 size= 0xd0000 nonshared rw- needs-copy
[ROOTTASK]        pa=0xd108c000 .. 0xd115c000
[ROOTTASK] map_segment:   APPCON va=       0x3df4000 size= 0x10000 nonshared rw- needs-copy
[ROOTTASK]        pa=0xd115c000 .. 0xd116c000
[ROOTTASK] map_segment:   OSCSH2 va=       0x3e04000 size=  0x4000    shared rw-
[ROOTTASK]        pa=0xc20a4000 .. 0xc20a8000
[ROOTTASK] map_segment:   OSCSHR va=       0x3e08000 size=  0x4000    shared rw-
[ROOTTASK]        pa=0xc20a8000 .. 0xc20ac000
[ROOTTASK] map_segment: skip OSC_01
[ROOTTASK] map_segment:   OSC_02 va=       0x657c000 size=0x1838000 nonshared rw-
[ROOTTASK]        pa=0xc481c000 .. 0xc6054000
[ROOTTASK] map_segment:   OSC_03 va=       0x7db4000 size= 0x28000 nonshared rw- needs-copy
[ROOTTASK]        pa=0xd116c000 .. 0xd1194000
[ROOTTASK] map_segment: skip OSC_04
[ROOTTASK] map_segment: skip OSC_05
[ROOTTASK] map_segment:   OSC_06 va=       0x8500000 size= 0x1c000 nonshared rw-
[ROOTTASK]        pa=0xc67a0000 .. 0xc67bc000
[ROOTTASK] map_segment: skip __PW_P
[ROOTTASK] map_segment: skip __MHOS
[ROOTTASK] map_segment: skip __RF_I
[ROOTTASK] map_segment: skip __IOSM
[ROOTTASK] map_segment: skip __GTI_
[ROOTTASK] map_segment: skip __CPS_
[ROOTTASK] map_segment: skip __CPS_
[ROOTTASK] map_segment: skip __EXPR
[ROOTTASK] map_segment: skip __RT_S
[ROOTTASK] map_segment: skip __SENS
[ROOTTASK] map_segment: skip __SAH_
[ROOTTASK] map_segment: skip __UE_C
[ROOTTASK] map_segment: skip __RO_D
[ROOTTASK] map_segment: skip __DRAM
[ROOTTASK] map_segment: skip __DRAM
[ROOTTASK] map_segment: skip __CPS_
[ROOTTASK] map_segment: skip __SHMS
[ROOTTASK] map_segment: skip __SHMC
[ROOTTASK] map_segment: skip __PBM_
[ROOTTASK] map_segment: skip __MSG_
[ROOTTASK] map_segment: skip __MSG2
[ROOTTASK] map_segment: skip __PS_F
[ROOTTASK] map_segment: skip __L1ME
[ROOTTASK] map_segment: skip __TAGG
[ROOTTASK] map_segment: skip __MC_S
[ROOTTASK] map_segment: skip __MW_M
[ROOTTASK] map_segment: skip __2G_D
[ROOTTASK] map_segment: skip __2G_C
[ROOTTASK] map_segment: skip __CMSG
[ROOTTASK] map_segment: skip __RO_N
[ROOTTASK] map_segment: skip __L2DP
[ROOTTASK] map_segment: skip __NC_L
[ROOTTASK] map_segment: skip __RW_C
[ROOTTASK] map_segment: skip __SRAM
[ROOTTASK] map_segment: skip __CPSI
[ROOTTASK] map_segment: skip __SRAM
[ROOTTASK] Fixing up config var: 0x0050000001e01450 -> 0x0000000003e05450
[ROOTTASK] Fixing up config var: 0x0260000000009758 -> 0x000000000200d758
[ROOTTASK] Fixing up config var: 0x8010000000009554 -> 0x000000000200d554
[ROOTTASK] set TCB 0x80041160 prio=48 control=80
[ROOTTASK] Fixing up config var: 0x8020000000009594 -> 0x000000000200d594
[ROOTTASK] set TCB 0x840305a0 prio=c0 control=c0
[ROOTTASK] Fixing up config var: 0x8020000000009594 -> 0x000000000200d594
[ROOTTASK] set TCB 0x840305e0 prio=c0 control=c0
[ROOTTASK] Fixing up config var: 0x8020000000009594 -> 0x000000000200d594
[ROOTTASK] set TCB 0x84030620 prio=c0 control=c0
[ROOTTASK] creating IRQ tcb
[ROOTTASK] Fixing up config var: 0x8010000000009574 -> 0x000000000200d574
[ROOTTASK] set TCB 0x80041720 prio=80 control=80
[ROOTTASK] creating L2 IRQ proxy
[ROOTTASK] copying timer 0x8003c420
[ROOTTASK] copying cache 0x80040240 for pid 2
[ROOTTASK] create_irq_table idx 0 vec 0x1012e cap 0x1fe360c7c0
[ROOTTASK] create_irq_table idx 1 vec 0x1012f cap 0x1fe360c7e0
[ROOTTASK] create_irq_table idx 2 vec 0x10130 cap 0x1fe360c800
```

OSCKit proc initialisation

- All processes jump to main trampoline when starting
- First apply fixups to reslide and sign pointers
- Parse APPCON to get process information
- Call ctors if specified in the PID flags and jump to the PID’s entry points when pid != 1?
- pid 1 will continue to system thread

OSCKit System Thread

- OSC_1 contains a table with a series of MMIO read/writes that initialise hardware
- After HW is initialised an FTAB will be loaded
- Validates boot nonce, IM4M and RCPI
- PCIe & SPMI HAL accessible via UtaIoMgrRegisterEx
- …
- Init table

Non system threads

- Both PID 2 and 3 have similar initialisation sequences
  - After APPCON parsing we will jump to the entry point
    - Setup heap and some global offset to required sections
    - Find RPC channel 1 of pid 1 and await checkin
    - Map available MMIO regions
    - Init table
    - Start RPC servers

Initialisers table

```
__int64 osc_run_initializers()
{
  __int64 pid; // x0
  unsigned int idx; // w19
  int pid_mask; // w22
  osc_init_entry *v4; // x23

  pid = getpid();
  idx = 0;
  pid_mask = 1 << pid;
  do
  {
    if ( osc_init_table_start < &osc_init_table_end )
    {
      v4 = osc_init_table_start;
      do
      {
        if ( idx == v4->fields_buf[0] && (v4->pid_mask & pid_mask) != 0 )
          pid = (v4->initializer)();
        ++v4;
      }
      while ( v4 < &osc_init_table_end );
    }
  }
  while ( idx++ < 0xC );
  return pid;
}
```

```
osc_init_entry <aOscRpcRegister_169, \ ; "osc_rpc_register_client_OSC_RPC_ED_ASYN"...
                osc_rpc_register_client_OSC_RPC_ED_ASYNC_SERVICE_OTA_MIDDLEWARE_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 4>
osc_init_entry <aOscRpcRegister_170, \ ; "osc_rpc_register_server_OSC_RPC_ED_ASYN"...
                osc_rpc_register_server_OSC_RPC_ED_ASYNC_SERVICE_MIDDLEWARE_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 4>
osc_init_entry <aOscRpcRegister_171, \ ; "osc_rpc_register_client_OSC_RPC_ED_ASYN"...
                osc_rpc_register_client_OSC_RPC_ED_ASYNC_SERVICE_MIDDLEWARE_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 8>
osc_init_entry <aIcuInit, icu_init_2, <2, 0, 0, 0>, 2> ; "icu_init"
osc_init_entry <aIcuInit, icu_init_2, <2, 0, 0, 0>, 4> ; "icu_init"
osc_init_entry <aIcuInit, icu_init_2, <2, 0, 0, 0>, 8> ; "icu_init"
osc_init_entry <aKicInit, kic_init_2, <0, 0, 0, 0>, 2> ; "kic_init"
osc_init_entry <aKicInit, kic_init_2, <0, 0, 0, 0>, 4> ; "kic_init"
osc_init_entry <aKicInit, kic_init_2, <0, 0, 0, 0>, 8> ; "kic_init"
osc_init_entry <aOscRpcRegister_100, \ ; "osc_rpc_register_server_OSC_RPC_MFWK_MI"...
                osc_rpc_register_server_OSC_RPC_MFWK_MIDDLEWARE_PLATFORM_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 2>
osc_init_entry <aOscRpcRegister_172, \ ; "osc_rpc_register_client_OSC_RPC_MFWK_MI"...
                osc_rpc_register_client_OSC_RPC_MFWK_MIDDLEWARE_PLATFORM_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 8>
osc_init_entry <aOscRpcRegister_101, \ ; "osc_rpc_register_server_OSC_RPC_MFWK_OT"...
                osc_rpc_register_server_OSC_RPC_MFWK_OTA_PLATFORM_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 2>
osc_init_entry <aOscRpcRegister_102, \ ; "osc_rpc_register_client_OSC_RPC_MFWK_OT"...
                osc_rpc_register_client_OSC_RPC_MFWK_OTA_PLATFORM_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 4>
osc_init_entry <aOscRpcRegister_103, \ ; "osc_rpc_register_server_OSC_RPC_MFWK_PL"...
                osc_rpc_register_server_OSC_RPC_MFWK_PLATFORM_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 4>
osc_init_entry <aOscRpcRegister_104, \ ; "osc_rpc_register_client_OSC_RPC_MFWK_PL"...
                osc_rpc_register_client_OSC_RPC_MFWK_PLATFORM_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 2>
osc_init_entry <aOscRpcRegister_105, \ ; "osc_rpc_register_server_OSC_RPC_MFWK_MI"...
                osc_rpc_register_server_OSC_RPC_MFWK_MIDDLEWARE_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 4>
osc_init_entry <aOscRpcRegister_106, \ ; "osc_rpc_register_client_OSC_RPC_MFWK_MI"...
                osc_rpc_register_client_OSC_RPC_MFWK_MIDDLEWARE_OTA_CL4_CHANNEL_ASYNC22_RPC,\
                <1, 0, 0, 0>, 8>
```

OSC RPC

- Interface for doing RPC across processes
- Built on top of APPCON defined IPC channels
- Per process table where channel to handler map is stored
- Sync/async handlers need to be registered via osc_rpc_register

```
osc_register_async_handler(
  39,
  1,
  4,
  osc_rpc_trace_notify_service_handler,
  &unk_65800C0,
  "OSC_RPC_TRACE_NOTIFY_SERVICE",
  a7,
  a8);
```

```
sync_or_async = a1->sync_or_async;
if ( sync_or_async != 2 )
{
  if ( sync_or_async == 1 )
  {
    ipc_chan_curr_pid = osc_appc_find_channel_curr_pid(a1->info.service_no, a1->info.req_sz);
    sz = osc_rpc_msg_get_sz(ipc_chan_curr_pid);
    next_rpc_entry = osc_rpc_get_next_rpc_entry(ipc_chan_curr_pid);
    osc_rpc_await_l4_notify_sem(ipc_chan_curr_pid);
    *&rpc_entry.service_no = 0;
    rpc_entry.req_sz = 0;
    v6 = sz - 16;
    if ( sz < 0x10 )
      osc_logerr_f("ipc_read: message too short\n");
    for ( ; !memcpy_s(&rpc_entry, 0x10u, next_rpc_entry, 0x10u); rpc_entry.req_sz = 0 )
    {
      if ( (rpc_entry.service_no - 1) >= 0x37 )
        osc_logerr_f("RPC server %s: received invalid service %u\n");
      if ( rpc_entry.req_sz > v6 )
        osc_logerr_f("RPC server %s: invalid request size %zu\n", v7, v8);
      rpc_handler = a1->handlers[6 * rpc_entry.service_no + 9];
      if ( !rpc_handler )
        osc_logerr_f("RPC server %s: no handler for service %u\n");
      response_sz = 0;
      do_some_mmio_maybe(word_3BA1A70);       // ACK the msg?
      if ( rpc_handler(
             a1->handlers[6 * rpc_entry.service_no + 8],
             rpc_entry.req_sz,
             &next_rpc_entry->thread_name,
             sz - 16,
             &response_sz,
             &next_rpc_entry->thread_name) )
      {
        osc_logerr_f("RPC server %s: handler for service %s (%u) failed\n", v10, v11, v12);
      }
      if ( response_sz > v6 )
        osc_logerr_f("RPC server %s: invalid response size %zu\n");
      resp.pid = 0;
      resp.req_sz = response_sz;
      resp.service_no = rpc_entry.service_no;
      do_some_mmio_maybe(word_3BA1A7C);       // Send reponse signal?
      if ( memcpy_s(next_rpc_entry, sz, &resp, 0x10u) )
        osc_logerr_f("ipc_write copy failed\n");
      osc_send_sync_reply(ipc_chan_curr_pid, v13);
      osc_rpc_await_l4_notify_sem(ipc_chan_curr_pid);
      *&rpc_entry.service_no = 0;
    }
    osc_logerr_f("ipc_read copy failed\n");
  }
  osc_logerr_f("osc_rpc_server_main: invalid rpc_kind\n");
}
osc_rpc_start_async_srv(&a1->info.service_no);
```

OSC Tasks

An incomplete list…

- TCP/IP
  - lwip01
  - lwip02
  - lwip03
  - lwip04
- ICC
  - DMA_OSC_SERVER
  - RSM_PROXY
  - l1c_glue_icc_rx
  - cdp_glue_icc_rx
  - icc_low_pri
  - icc_lms
  - iosm
  - spmi_heb
  - usif
- Audio
  - IMAS
  - IMAS_PCIE
  - ICE_AUD_TASK
- ARI
  - Various ARI capsules
- BOOTMAN
  - BMM_HP
  - CSI_BMM
- SIM card
  - UICC_DRV
  - SIM_ARB_TASK
  - VINYL_TASK
- Analytics
  - AWD
  - ED_TASK
  - RAP_TASK
- MXS
  - asm_1
  - cdps_1
  - cgdci_1
  - cmr_1
  - dll_1
  - emm_1
  - errcl_1
  - errcs_1
  - errcv_1
  - gmm_1
  - gps_1
  - l2c_1
  - l2c3g_1
  - l2dlh3g_1
  - l2dll3g_1
  - l2ula3g_1
  - l2uls3g_1
  - l3c_1
  - llc_1
  - lpm_1
  - lqm_1
  - mac_1
  - mmc_1
  - mme_1
  - nmm_1
  - nrrcl_1
  - nrrcm_1
  - nrrcs_1
  - oms_1
  - ps_hist_1
  - ptmd_1
  - ptmu_1
  - rlc_1
  - rrc_1
  - grr_1
  - rrl_1
  - sdm_1
  - sic_1
  - smr_1
  - snp_1

MXS

Message Exchange Signals?

- Each service runs in its own thread and maintains a state and two queues of unprocessed signals
- Thread entry will await an UtaOsEventGroupRetrieve and dispatch the signal to the correct handler based on msg_id
- _glue_icc_rx threads use the inter-core communication interface to let L1CS and CDP raise MXS signals
- osc_mxs_receive_remote lets other pids generate MXS signals

[REDACTED]

MXS Tables

```
reciver = mxs_get_reciver(0x3Cu, 0);
v15 = sub_274FBB4(reciver);
mxsSignalSend(v15, v9, 0x290020, a1, mxs
```

```
DCQ mxs_mnc_msgid_290019
DCQ 0x290019
DCQ mxs_mnc_msgid_29001a
DCQ 0x29001A
DCQ mxs_mnc_msgid_29001b
DCQ 0x29001B
DCQ mxs_mnc_msgid_29001c
DCQ 0x29001C
DCQ mxs_mnc_msgid_29001d
DCQ 0x29001D
DCQ mxs_mnc_msgid_29001e
DCQ 0x29001E
DCQ mxs_mnc_msgid_29001f
DCQ 0x29001F
DCQ mxs_mnc_msgid_290020
DCQ 0x290020
DCQ mxs_mnc_msgid_290021
DCQ 0x290021
DCQ mxs_mnc_msgid_290022
DCQ 0x290022
DCQ mxs_mnc_msgid_290023
DCQ 0x290023
DCQ mxs_mnc_msgid_290024
DCQ 0x290024
DCQ mxs_mnc_msgid_290025
DCQ 0x290025
DCQ mxs_mnc_msgid_290026
DCQ 0x290026
DCQ mxs_mnc_msgid_290028
DCQ 0x290028
DCQ mxs_mnc_msgid_290029
```

```
__int64 __fastcall mxs_mnc_msgid_290020(mxs_base *a1, __int64 a2)
{
  int v3; // w20
  __int64 result; // x0

  if ( (a1->state_flags & 0x7F) != 6 )
    return sub_32CB964(a1);
```

```
DCQ aMxsMncStart        ; DATA XREF: m.OSC_02:0000000
                        ; "MXS_MNC_START"
DCQ aMxsMncNull         ; "MXS_MNC_NULL"
DCQ aMxsMncLocked       ; "MXS_MNC_LOCKED"
DCQ aMxsMncCheck        ; "MXS_MNC_CHECK"
DCQ aMxsMncMocAlloc     ; "MXS_MNC_MOC_ALLOC"
DCQ aMxsMncMocSetup     ; "MXS_MNC_MOC_SETUP"
DCQ aMxsMncMtcSetup     ; "MXS_MNC_MTC_SETUP"
```

Emulation in Qemu

Our approach

- Emulation based on fork of QemuAppleSilicon (forked from qemu-t8030).
- Greybox emulation based on tracing MMIO accesses and seeing how it works.
- Some hardware is happy with zeroes being returned or RAM regions.
- More complex hardware needed hacks or proper emulation.
- Patching out code that accesses non-relevant devices.
- Start with LLB (iBoot) and move on to booting rkos.
- Inject custom cellular frames.

Emulated devices

- SPMI Master
- Thebe PMU
- UART
- PMGR
- DCS
- KIC (aka ICU)
- PCI-E, SPMI RCS and more will follow

Reversing MMIO devices

- Most of devices available in rkos live in structures with a PA.
- Not all of them have it available statically.

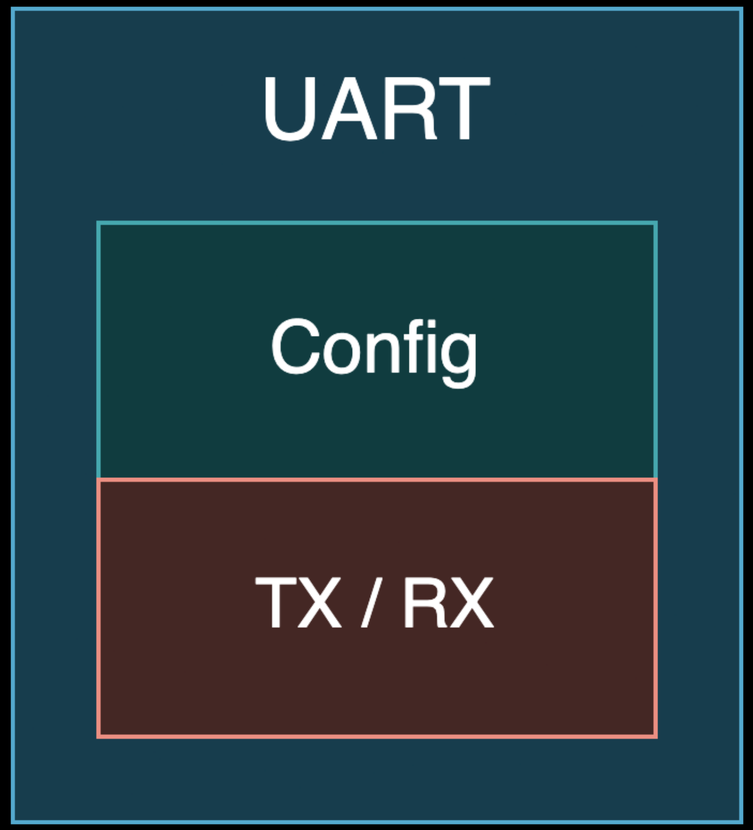
```
spmi_rcs_device <<g_dev_spmi_rcs_funcs, 0x21A00000, 0x40000, 0>, 0, \
                        ; DATA XREF: sub_2133E0C+30↑o
                        ; sub_2133E0C+4C↑o ...
                 0x100, 0xC000, 0x2000, 0x3000, 0x4000, 0x8000, 1, \
                 0x10095, 0, 0>
```

```
                DCB    0
RTK_DEV_ICU     rtk_device_mapping <off_3DF07B8, 0, 0, 0>
                                   ; DATA XREF: sub_37ED150+C4↑o
                                   ; sub_37ED150+D0↑o ...
                DCB    0
```

UART

- Most likely a variation of the good old Apple UART.
- Splits TX / RX and config into separate pages.

```
  if ( (byte_460CD8F0[0] & 1) != 0 )
    hw_uart_putc('\r');
  if ( ((v12 ^ (2 * v12)) & 0x4000000000000000LL) != 0 )
    __break(0xC471u);
  sub_4608380C('\r');
  v8 = (unsigned __int8)*i;
ABEL_20:
  v10 = v8;
  if ( (byte_460CD8F0[0] & 1) != 0 )
    hw_uart_putc((unsigned __int8)v8);
```



Interrupt Controller

Basic overview

- Likely named ICU or KIC
- A simpler version of AIC without advanced features.
- Uses Apple’s new approach with multiple physical pages for a device.

Interrupt Controller

Reverse engineering

Enable IRQ:

```
if ( (_DWORD)result != 510 )
{
  v1 = result;
  result = sub_46044ACC(result);
  v2 = 3LL * v1;
  v4 = __CFADD__(v2 * 8, g_interrupt_list);
  v3 = &g_interrupt_list[v2];
  if ( v4 )
    panic_unaligned_ptr(result);
  v4 = v3 < g_interrupt_list || v3 >= qword_460C2AA0;
  if ( v4 )
    ptr_oob();
  if ( (*(_BYTE *)v3 & 1) == 0 )
    *(_DWORD *)(((v1 >> 3) & 0x1FFFFFFC) + 0x40400A80LL) = 1 << v1;
}
```

Disable IRQ:

```
result = sub_46044A9C(v4, (unsigned __int64)g_interrupt_list, (unsig
if ( v2 != 510 )
  *(_DWORD *)(((a1 >> 3) & 0x1FFFFC) + 0x40400A00LL) = 1 << a1;
return result;
```

Interrupt Controller

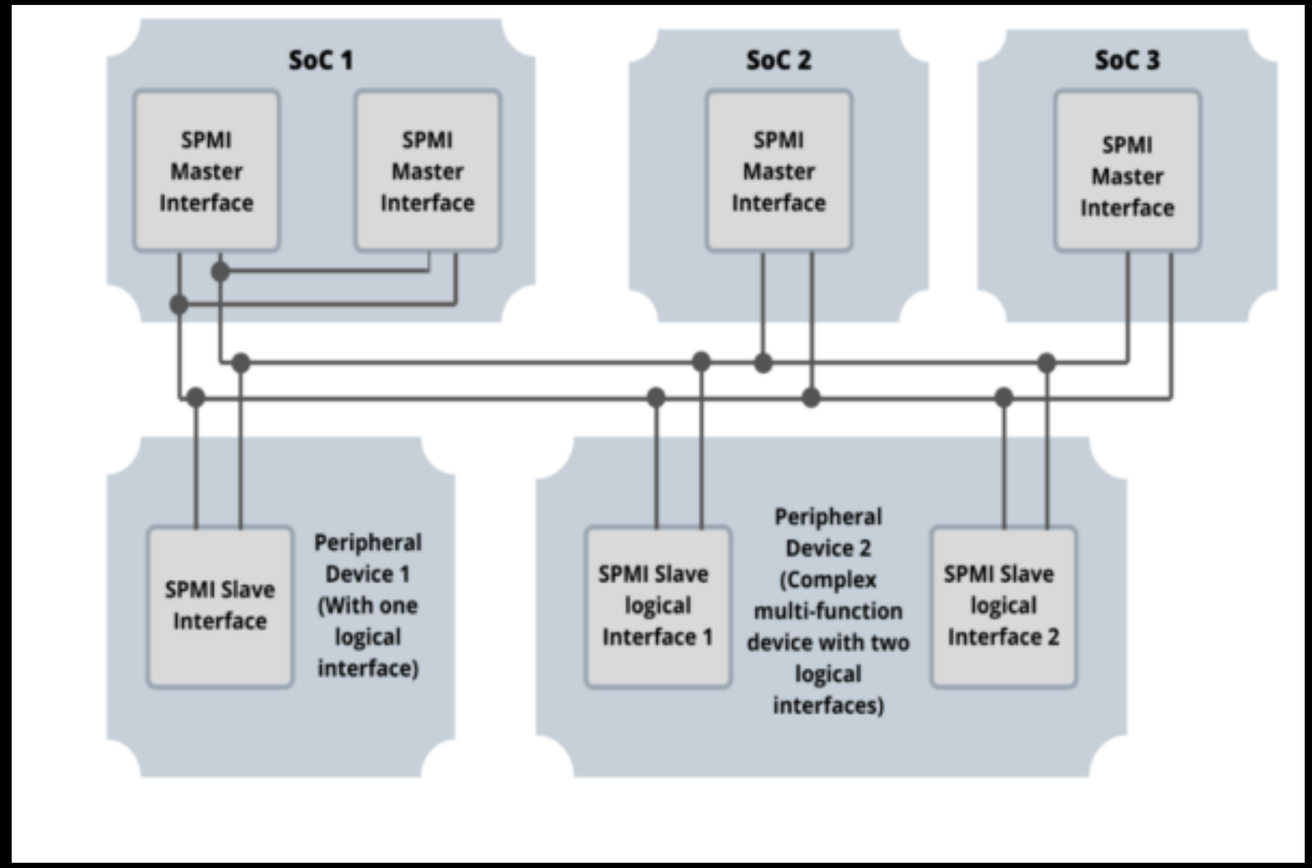
Reverse engineering

```
case REG_AIC_EIR_MASK_SET(0)... REG_AIC_EIR_MASK_SET(kAIC_NUM_EIRS): {
    uint32_t eir = (addr - REG_AIC_EIR_MASK_SET(0)) / 4;
    if (unlikely(eir >= s->numEIR)) {
        break;
    }
    s->eir_mask[eir] |= val;
    break;
}

case REG_AIC_EIR_MASK_CLR(0)... REG_AIC_EIR_MASK_CLR(kAIC_NUM_EIRS): {
    uint32_t eir = (addr - REG_AIC_EIR_MASK_CLR(0)) / 4;

    if (unlikely(eir >= s->numEIR)) {
        break;
    }

    s->eir_mask[eir] &= ~val;
```

SPMI Master

What is SPMI?

- I2C-like bus interface.
- Clearly defines allowed commands.
- Designed for Power Management
- Up to 4 masters, 16 slaves.
- Allows both physical and virtual connections.



SPMI Master

Basic overview

- Same Apple SPMI Master device found in M-series SoCs.
- Has multiple generations.
- Since generation 2 (first is 0) has multi-page design.
- Has multiple queues that in theory can be shared between processes by mapping corresponding pages.
- Has multiple types of queues: HW, SW and Backlight (which is also HW).

SPMI Master

Reverse engineering

```
t __fastcall spmi_recv_nofail(spmi_object *a1, unsigned __int64 a2, unsi

__int64 current_queue_idx; // x20

if ( (unsigned __int64)a1 < a2 || (unsigned __int64)a1 >= a3 )
  ptr_oob();
current_queue_idx = (unsigned __int8)a1->current_queue_idx;
if ( spmi_check_resp_queue_empty(a1, a2, a3) )
  panic_breadcrumbs(0x34B4DB11417AA37LL, 84);
return *(_DWORD *)(a1->sw_queues_addr + (current_queue_idx << 14) + 8);
```

```
if ( (v24 & 1) != 0 )
{
  a1 = (unsigned __int64)v31;
  v52 = v32;
  v53 = v33;
  v54 = v34;
  a2 = (__int64)"Response Queue Empty";
  v48 = "Response Queue Empty";
  v49 = "Header mismatch";
  v50 = &off_46098D60;
  spmi_panic(&a1, &a2);
}
return 0;
```

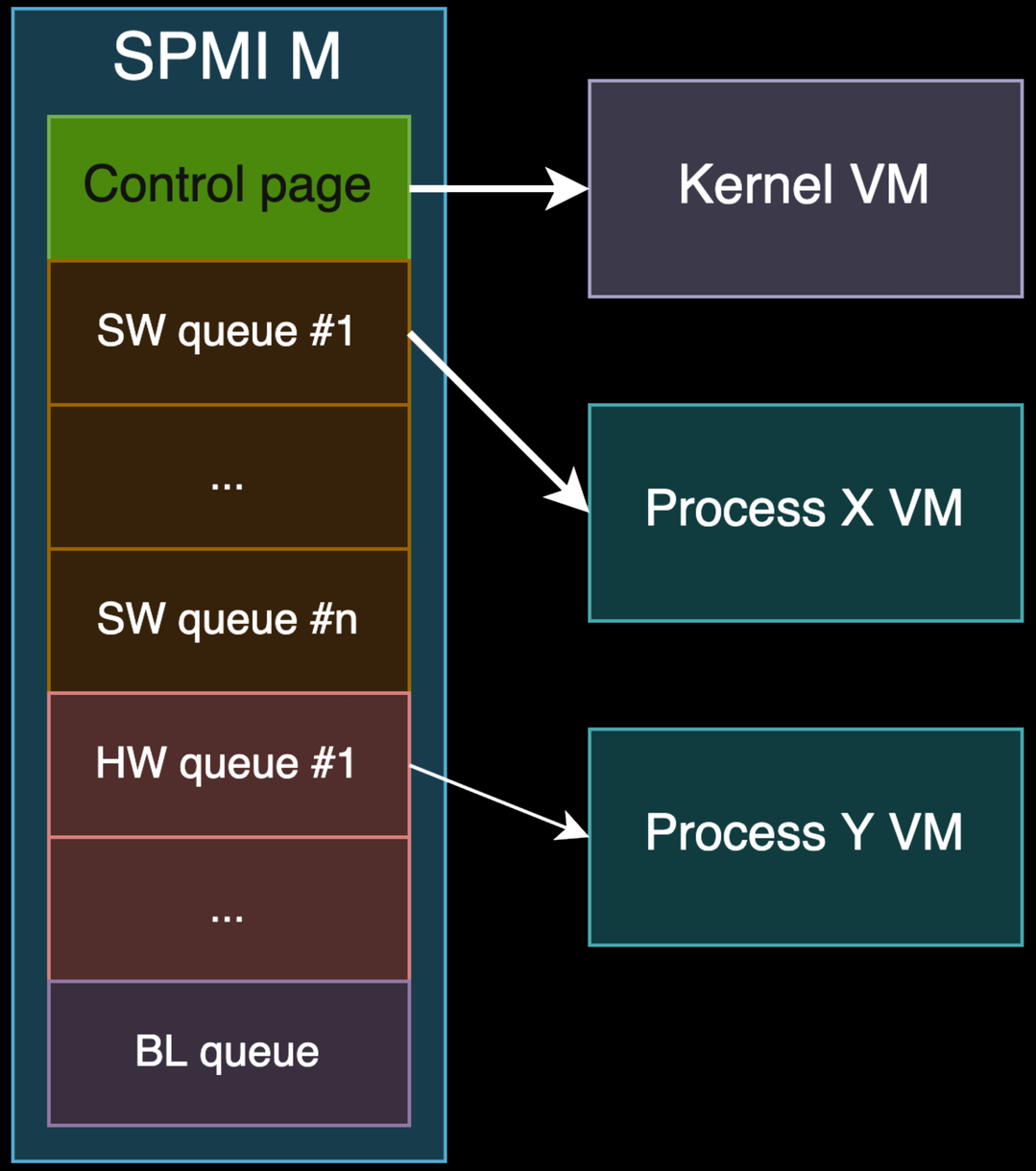
```
v28[2] = v9;
v28[3] = v8;
snprintf(v29, 200, "SPMI panic,
v10 = sub_46080844(v30, v30, v31
va_copy((va_list)v29, va);
sub_4607E8C4((__int64)&v30[v10],
printf("%llx:%d\n", 0x34B4DB1141
```

```
switch (addr) {
case SPMI_RSP_QUEUE_POP:
    if (fifo32_is_empty(&q->resp_fifo)) {
        qemu_log_mask(LOG_GUEST_ERROR, "%s: rsp queue em
                      DEVICE(s)->id);
        value = 0;
    } else {
        value = fifo32_pop(&q->resp_fifo);
    }
    qflg = true;
    iflg = true;
    break;
```

SPMI Master

Physical MMIO layout

- That is pure speculation, but...
- You can map individual queues to different processes.
- Allows to avoid context switches for MMIO device access.
- (each rectangle is a page)



Less known devices

- We don't know what some devices are.
- You don't really need it to emulate the parts you want.
- Some of them have names known from RKOS.

```
static uint64_t c1_21340000_read(void *opa
    CPUState *cpu = current_cpu;
    uint64_t pc = cpu != NULL ? cpu->cc->g
    //info_report("21340000 MMIO read @ %p

    if (addr == 0xC) return 0x80000000;
    else return 0;
}
```

Stupid bugs

- Defining most MSRs works normally
- This has one exception however, even after defining PMCR3 we still get an undefined instruction exception when accessing it, unlike PMCR4 and 2
- Solved in a very hacky way by having TCG not emit code when handling that specific MSR
- 😷

```
C1_CPREG_DEF(PMC9, 3, 2, 15, 10, 0, PL1_RW, 0),
C1_CPREG_DEF(PMCR0, 3, 1, 15, 0, 0, PL1_RW, 0),
C1_CPREG_DEF(PMCR1, 3, 1, 15, 1, 0, PL1_RW, 0),
C1_CPREG_DEF(PMCR2, 3, 1, 15, 2, 0, PL1_RW, 0),
C1_CPREG_DEF(PMCR3, 3, 1, 15, 3, 0, PL1_RW, 0),
C1_CPREG_DEF(PMCR4, 3, 1, 15, 4, 0, PL1_RW, 0),
C1_CPREG_DEF(PMESR0,3, 1, 15, 5, 0, PL1_RW, 0),
C1_CPREG_DEF(S3_7_C15_C4_6, 3, 7, 15, 4, 6, PL1_RW, 0),
C1_CPREG_DEF(PRE_LLCFLUSH_TMR, 3, 5, 15, 7, 0, PL1_RW, 0),
C1_CPREG_DEF(E_LSU_ERR_STS_EL1, 3, 3, 15, 2, 0, PL1_RW, 0),
C1_CPREG_DEF(E_FED_ERR_STS_EL1, 3, 4, 15, 0, 2, PL1_RW, 0),
C1_CPREG_DEF(L2C_ERR_STS_EL1, 3, 3, 15, 8, 0, PL1_RW, 0),
```

```
if(op0 == 3 && op1 == 1 && crn == 15 && crm == 3) {
    qemu_log_mask(LOG_UNIMP, "%s access to hacky MSR"
        "system register op0:%d op1:%d crn:%d crm:%d op2:%d\n",
        isread ? "read" : "write", op0, op1, crn, crm, op2);
    return;
}
```

It’s alive!

- After a lot of qemu tracing and reversing we got the following:

https://youtu.be/qCzoipRrk54

Conclusions

- Apple clearly put a lot of thought into the security of Leda
- More powerful bugs and possibly a pac forgery required for RCE
- Likely to ship in more future Apple products (17e & 17 Air?)
- Cellular SEP?
- Relatively easy to RE

Future plans

- Continue implementing HW and patch rkos enough to reach MXS initialisation
- End goal is to able to inject some code into RKOS that will send an MXS signal with controlled data and test the GSM stack
  - Similar approach to FirmWire
  - A long road ahead…

References

- SEPOS A Guided Tour
- FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware
- Breaking Band
- Unburdened By What Has Been: Exploiting New Attack Surfaces in Radio Layer 2 for Baseband RCE on Samsung Exynos
- Seemoo lab’s work on ARI, U1 and Stewie
- *OS Internals Volume 2

Thank you!

- qemu-c4000 eta son
- Any questions?