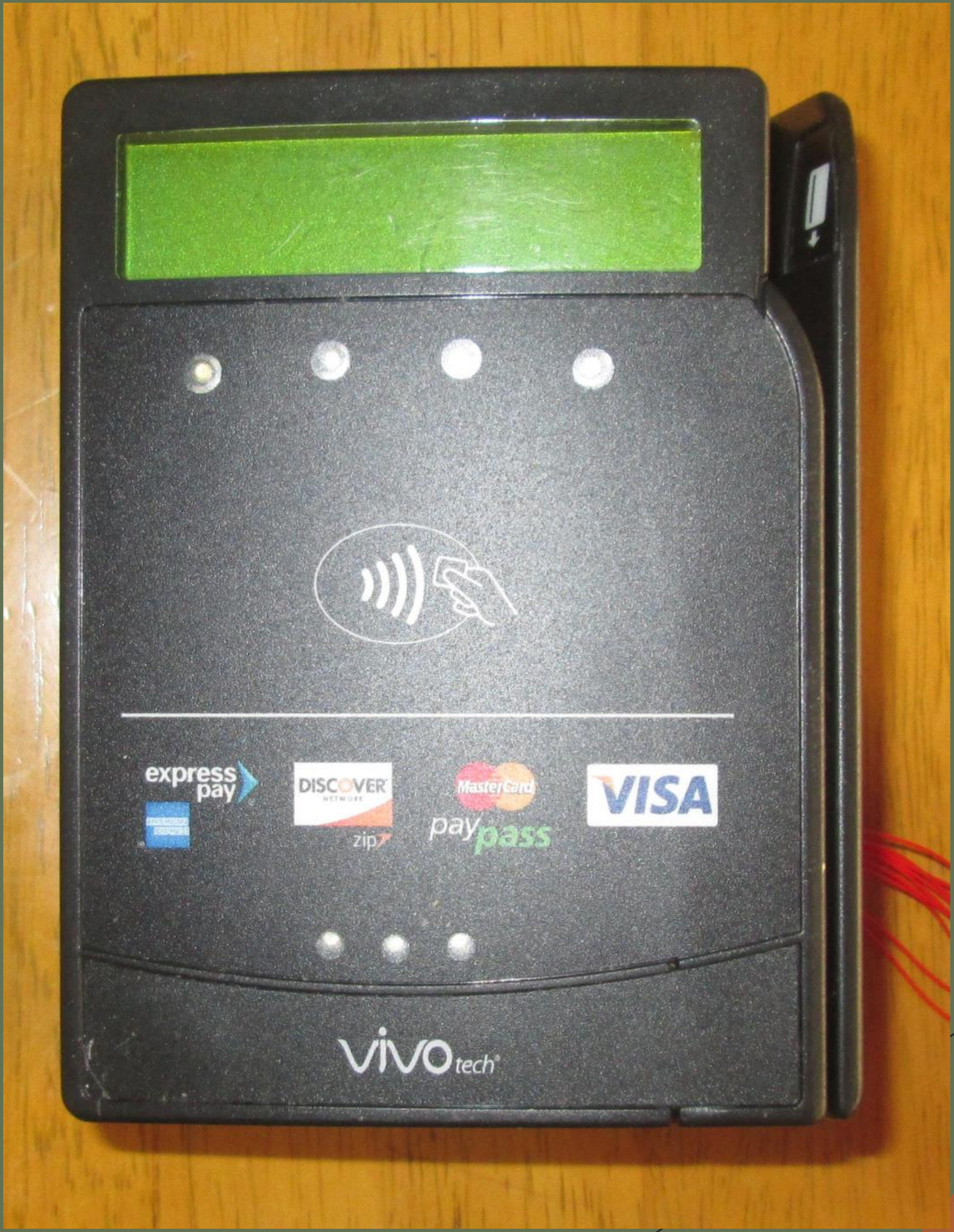
QEMU'ing up a storm

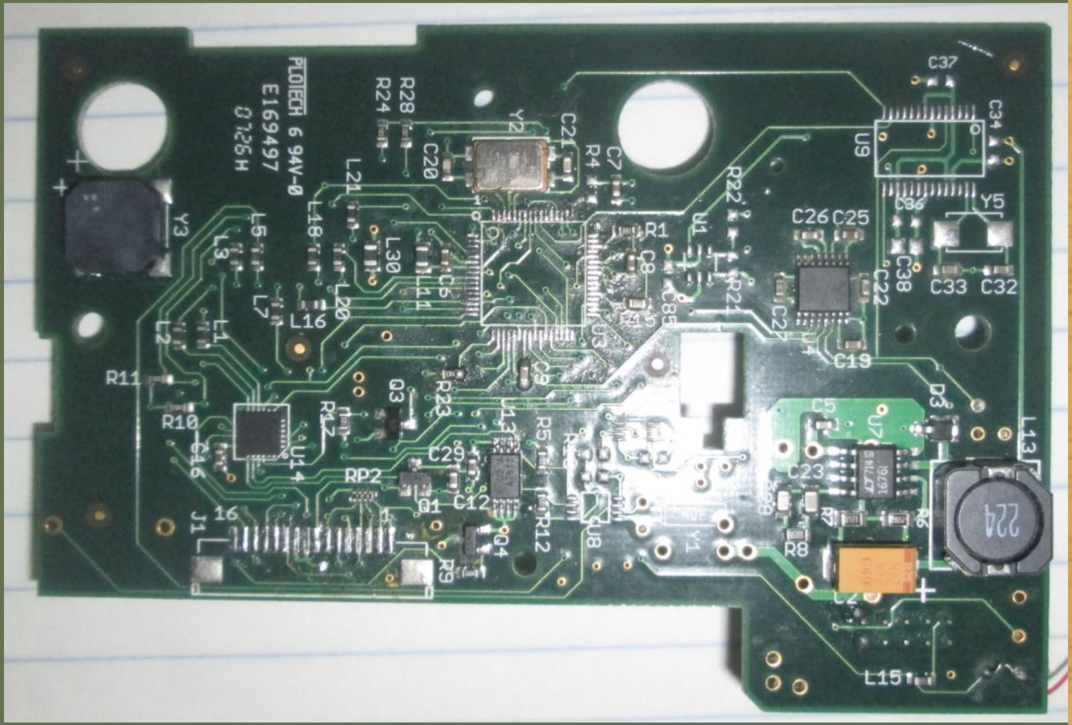
Why QEMU is pretty goddamn awesome!

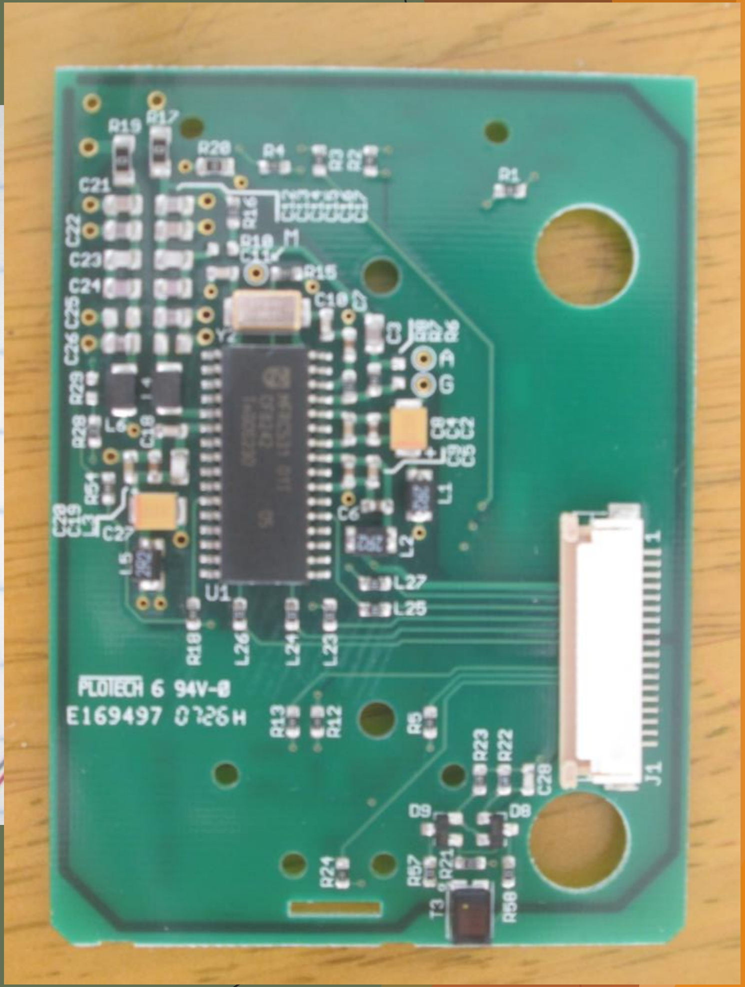
whoami

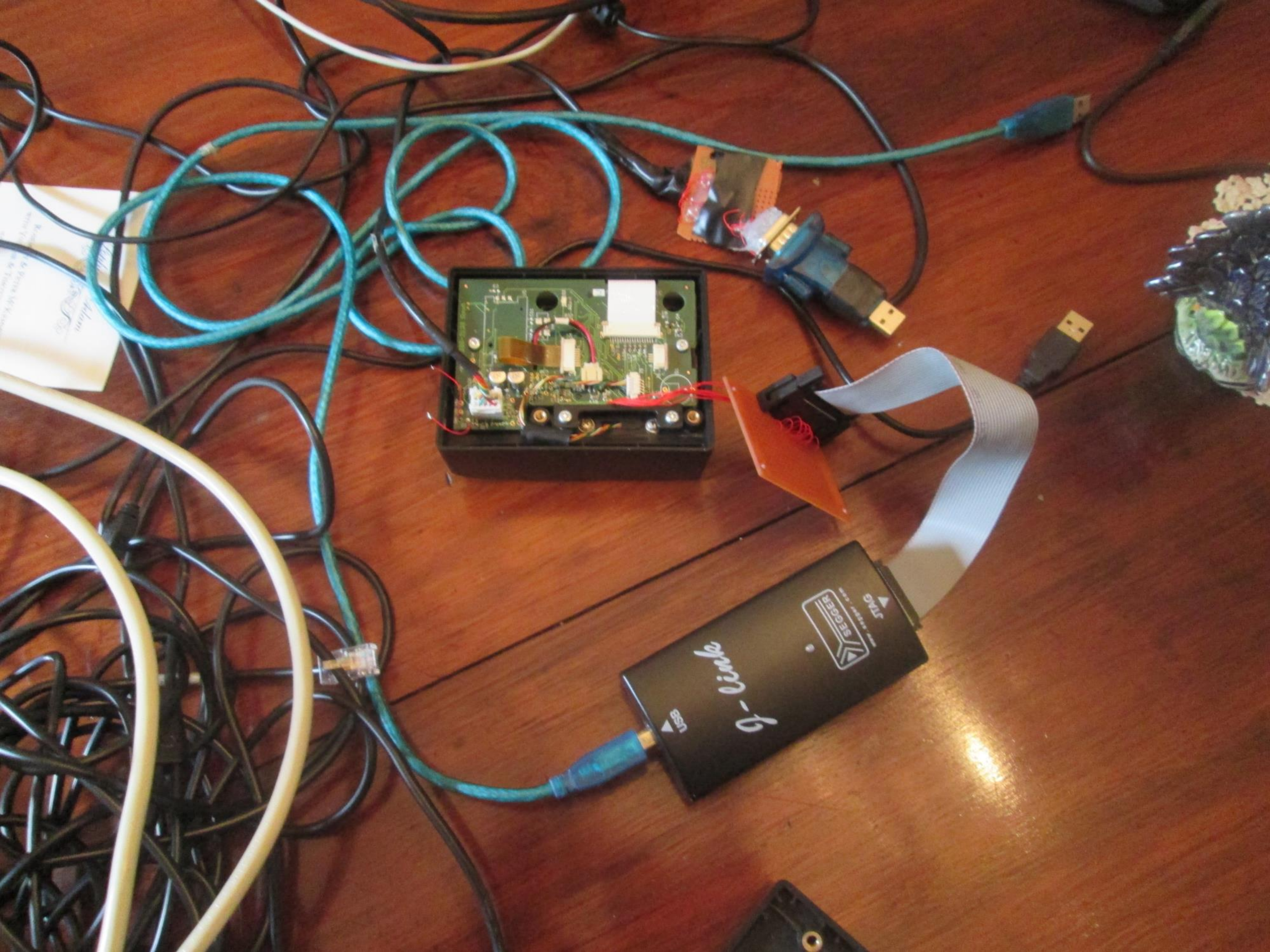
- Nominally do payment security stuff – i.e PCI…
- Sometimes do silly stuff
- Email: peter@peterfillmore.com
- Github: http://www.github.com/peterfillmore

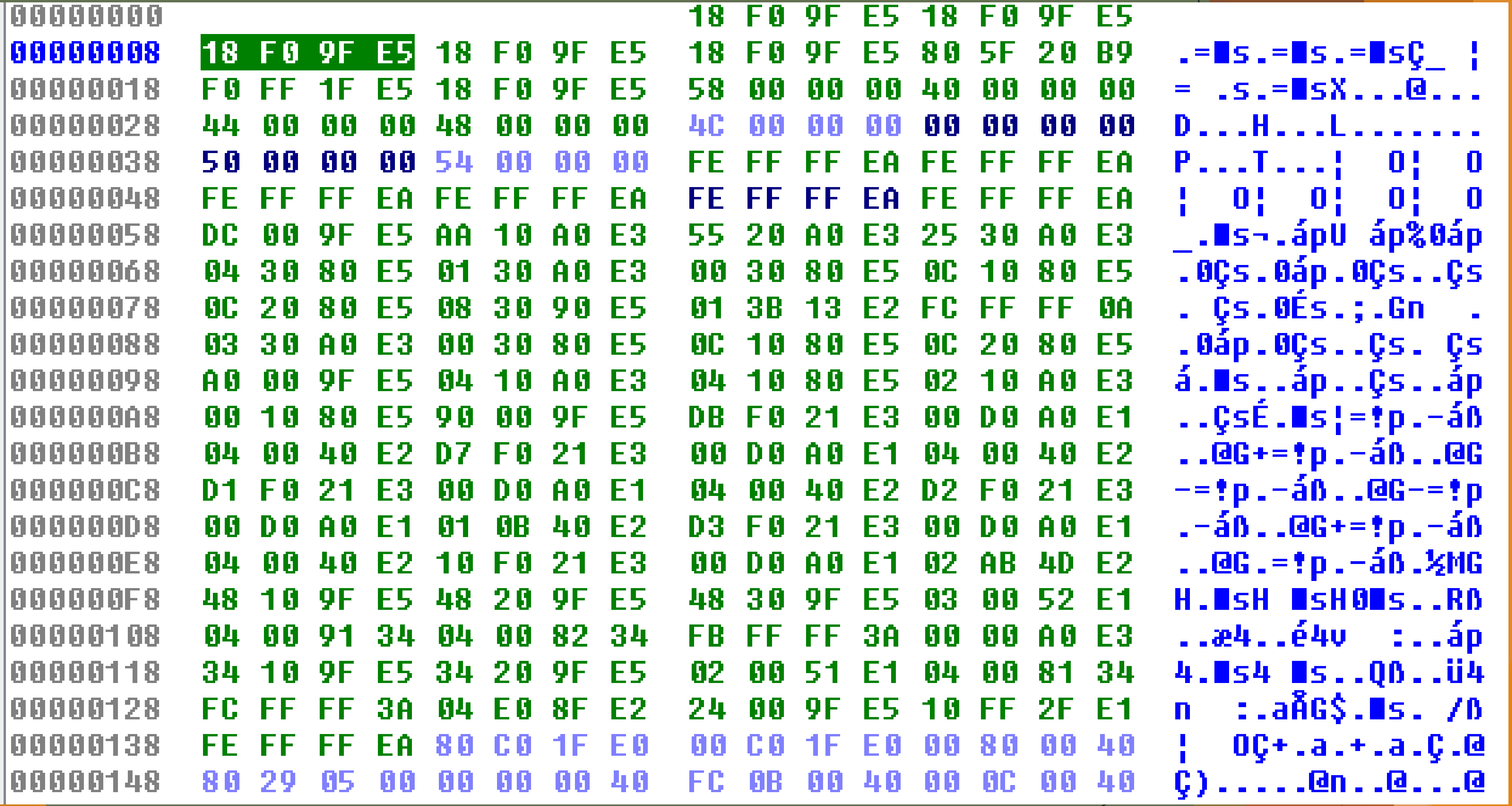
How I got to QEMU


express pay
DISCOVER NETWORK
zip
MasterCard
paypass
VISA
vivotech

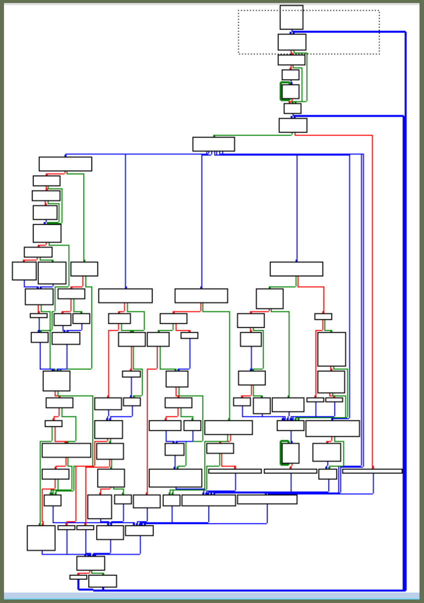
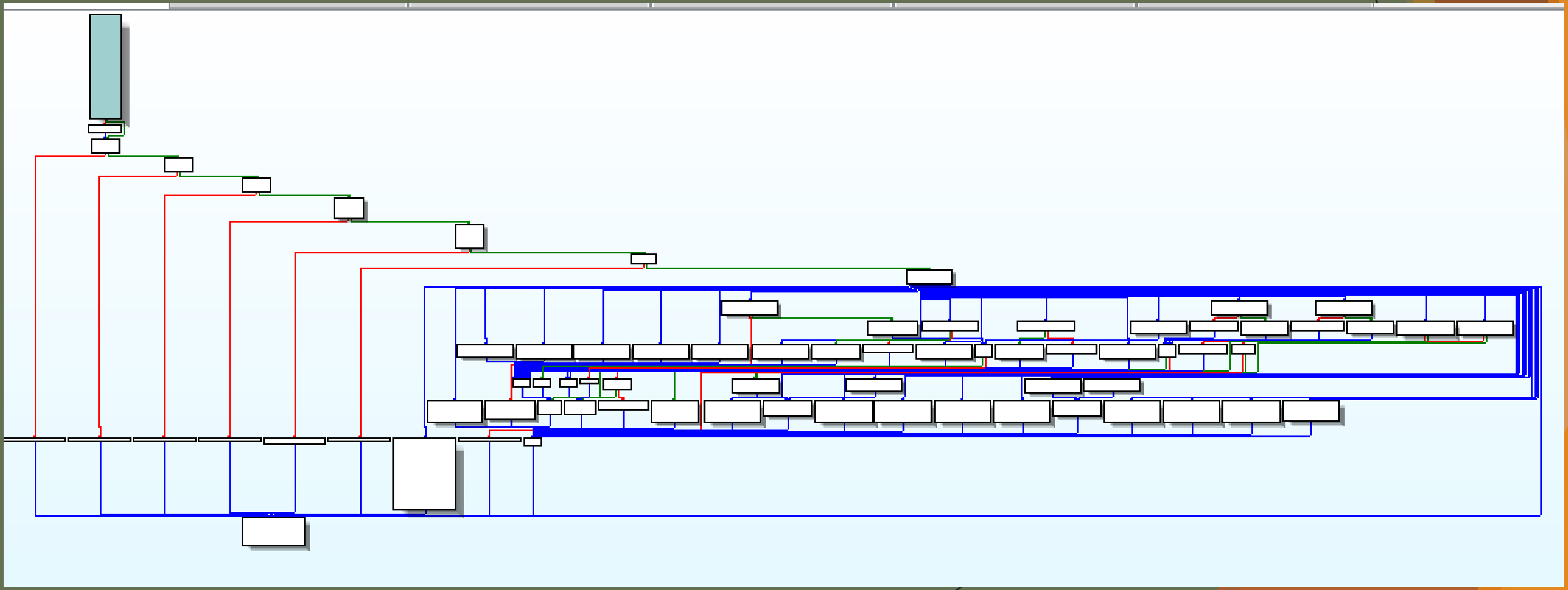

PLOTECH 6 94V-0
E169497
0726H


PLOTECH 6 94V-0
E169497 0726H


J-Link
SEGGER
USB
JTAG



```
00000000                              18 F0 9F E5 18 F0 9F E5
00000008  18 F0 9F E5 18 F0 9F E5     18 F0 9F E5 80 5F 20 B9    .=█s.=█s.=█sÇ_ ¦
00000018  F0 FF 1F E5 18 F0 9F E5     58 00 00 00 40 00 00 00    =  .s.=█sX...@...
00000028  44 00 00 00 48 00 00 00     4C 00 00 00 00 00 00 00    D...H...L.......
00000038  50 00 00 00 54 00 00 00     FE FF FF EA FE FF FF EA    P...T...¦  O¦  O
00000048  FE FF FF EA FE FF FF EA     FE FF FF EA FE FF FF EA    ¦  O¦  O¦  O¦  O
00000058  DC 00 9F E5 AA 10 A0 E3     55 20 A0 E3 25 30 A0 E3    _.█s¬.ápU áp%0áp
00000068  04 30 80 E5 01 30 A0 E3     00 30 80 E5 0C 10 80 E5    .0Çs.0áp.0Çs..Çs
00000078  0C 20 80 E5 08 30 90 E5     01 3B 13 E2 FC FF FF 0A    . Çs.0És.;.Gn   .
00000088  03 30 A0 E3 00 30 80 E5     0C 10 80 E5 0C 20 80 E5    .0áp.0Çs..Çs. Çs
00000098  A0 00 9F E5 04 10 A0 E3     04 10 80 E5 02 10 A0 E3    á.█s..áp..Çs..áp
000000A8  00 10 80 E5 90 00 9F E5     DB F0 21 E3 00 D0 A0 E1    ..ÇsÉ.█s¦=!p.-áß
000000B8  04 00 40 E2 D7 F0 21 E3     00 D0 A0 E1 04 00 40 E2    ..@G+=!p.-áß..@G
000000C8  D1 F0 21 E3 00 D0 A0 E1     04 00 40 E2 D2 F0 21 E3    -=!p.-áß..@G-=!p
000000D8  00 D0 A0 E1 01 0B 40 E2     D3 F0 21 E3 00 D0 A0 E1    .-áß..@G+=!p.-áß
000000E8  04 00 40 E2 10 F0 21 E3     00 D0 A0 E1 02 AB 4D E2    ..@G.=!p.-áß.½MG
000000F8  48 10 9F E5 48 20 9F E5     48 30 9F E5 03 00 52 E1    H.█sH █sH0█s..Rß
00000108  04 00 91 34 04 00 82 34     FB FF FF 3A 00 00 A0 E3    ..æ4..é4v   :..áp
00000118  34 10 9F E5 34 20 9F E5     02 00 51 E1 04 00 81 34    4.█s4 █s..Qß..ü4
00000128  FC FF FF 3A 04 E0 8F E2     24 00 9F E5 10 FF 2F E1    n   :.aÅG$.█s. /ß
00000138  FE FF FF EA 80 C0 1F E0     00 C0 1F E0 00 80 00 40    ¦   OÇ+.a.+.a.Ç.@
00000148  80 29 05 00 00 00 00 40     FC 0B 00 40 00 0C 00 40    Ç).....@n..@...@
```

QEMU

- “Quick Emulator”
- Originally written by Fabrice Bellard
- That dude is scary good, you’ve heard of:
  - FFMPEG
  - 4G LTE Base Station http://bellard.org/lte/
  - LZEXE !?! http://bellard.org/lzexe.html

http://bellard.org/

QEMU – Why so fast?

- Dynamic translation of target instructions to host.
- Completely written in C
- Uses TCG – Tiny Code Generator to generate RISC like instructions.
- These are then compiled dynamically for the host.
- So, speeeeeed.

What can you use it for?

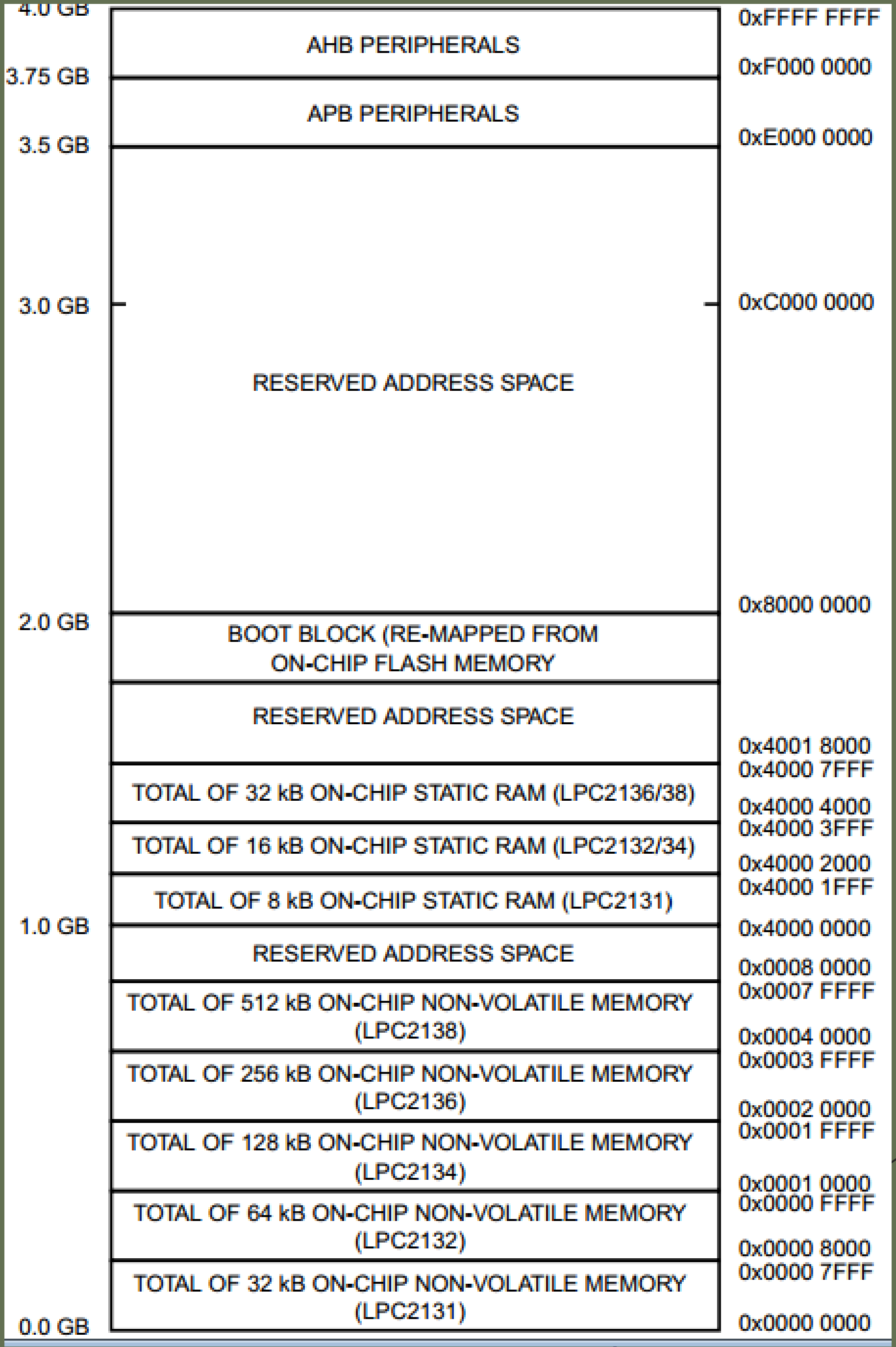
- User Mode Emulation – emulate a program compiled in another language directly on the host.
- System Emulation – emulate a complete system
- Kernel-based Virtual Machine stuff
- Xen Hosting stuff

What can I emulate?

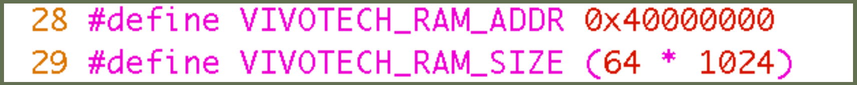
PC (x86 or x86_64 processor)	Luminary Micro LM3S811EVB (ARM Cortex-M3)
PREP (PowerPC processor)	Luminary Micro LM3S6965EVB (ARM Cortex-M3)
ISA PC (old style PC without PCI bus)	Freescale MCF5208EVB (ColdFire V2).
G3 Beige PowerMac (PowerPC processor)	Arnewsh MCF5206 evaluation board (ColdFire V2).
Mac99 PowerMac (PowerPC processor, in progress)	Palm Tungsten|E PDA (OMAP310 processor)
Sun4m/Sun4c/Sun4d (32-bit Sparc processor)	N800 and N810 tablets (OMAP2420 processor)
Sun4u/Sun4v (64-bit Sparc processor, in progress)	MusicPal (MV88W8618 ARM processor)
Malta board (32-bit and 64-bit MIPS processors)	Gumstix "Connex" and "Verdex" motherboards (PXA255/270).
MIPS Magnum (64-bit MIPS processor)	Siemens SX1 smartphone (OMAP310 processor)
ARM Integrator/CP (ARM)	AXIS-Devboard88 (CRISv32 ETRAX-FS).
ARM Versatile baseboard (ARM)	Spitz, Akita, Borzoi, Terrier and Tosa PDAs (PXA270 processor)
ARM RealView Emulation/Platform baseboard (ARM)	Petalogix Spartan 3aDSP1800 MMU ref design (MicroBlaze).

Why can’t I just QEMU out of the box?

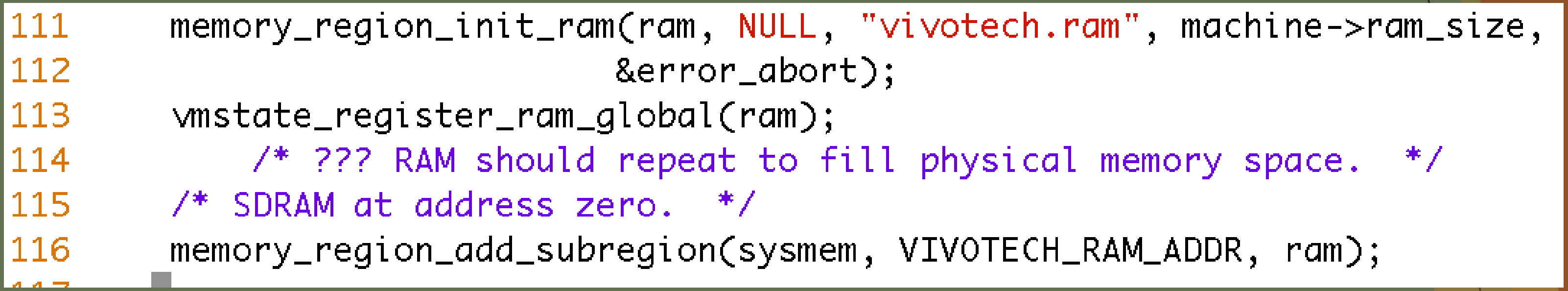
- QEMU is set out of the box for running OS’s
- However my image has no OS – it’s bare metal
- I know I have an ARM – so copy the “versitilePB.c” to “vivotech.c” – and add to make files
- Read QEMU code – get jealous as its pretty good – “self documenting”
- Make some stupid errors!


4.0 GB
0xFFFF FFFF
AHB PERIPHERALS
0xF000 0000
3.75 GB
APB PERIPHERALS
3.5 GB
0xE000 0000
3.0 GB
0xC000 0000
RESERVED ADDRESS SPACE
2.0 GB
0x8000 0000
BOOT BLOCK (RE-MAPPED FROM ON-CHIP FLASH MEMORY
RESERVED ADDRESS SPACE
0x4001 8000
0x4000 7FFF
TOTAL OF 32 kB ON-CHIP STATIC RAM (LPC2136/38)
0x4000 4000
0x4000 3FFF
TOTAL OF 16 kB ON-CHIP STATIC RAM (LPC2132/34)
0x4000 2000
0x4000 1FFF
TOTAL OF 8 kB ON-CHIP STATIC RAM (LPC2131)
1.0 GB
0x4000 0000
RESERVED ADDRESS SPACE
0x0008 0000
0x0007 FFFF
TOTAL OF 512 kB ON-CHIP NON-VOLATILE MEMORY (LPC2138)
0x0004 0000
0x0003 FFFF
TOTAL OF 256 kB ON-CHIP NON-VOLATILE MEMORY (LPC2136)
0x0002 0000
0x0001 FFFF
TOTAL OF 128 kB ON-CHIP NON-VOLATILE MEMORY (LPC2134)
0x0001 0000
0x0000 FFFF
TOTAL OF 64 kB ON-CHIP NON-VOLATILE MEMORY (LPC2132)
0x0000 8000
0x0000 7FFF
TOTAL OF 32 kB ON-CHIP NON-VOLATILE MEMORY (LPC2131)
0.0 GB
0x0000 0000

Initializing your RAM

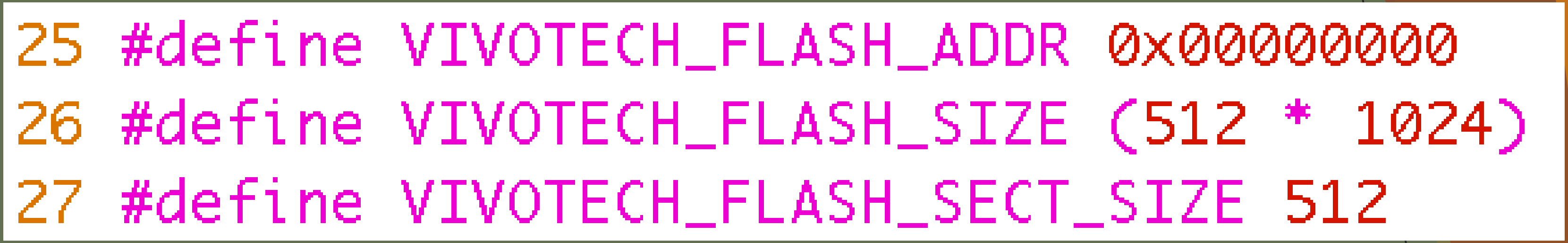


```
#define VIVOTECH_RAM_ADDR 0x40000000
#define VIVOTECH_RAM_SIZE (64 * 1024)
```

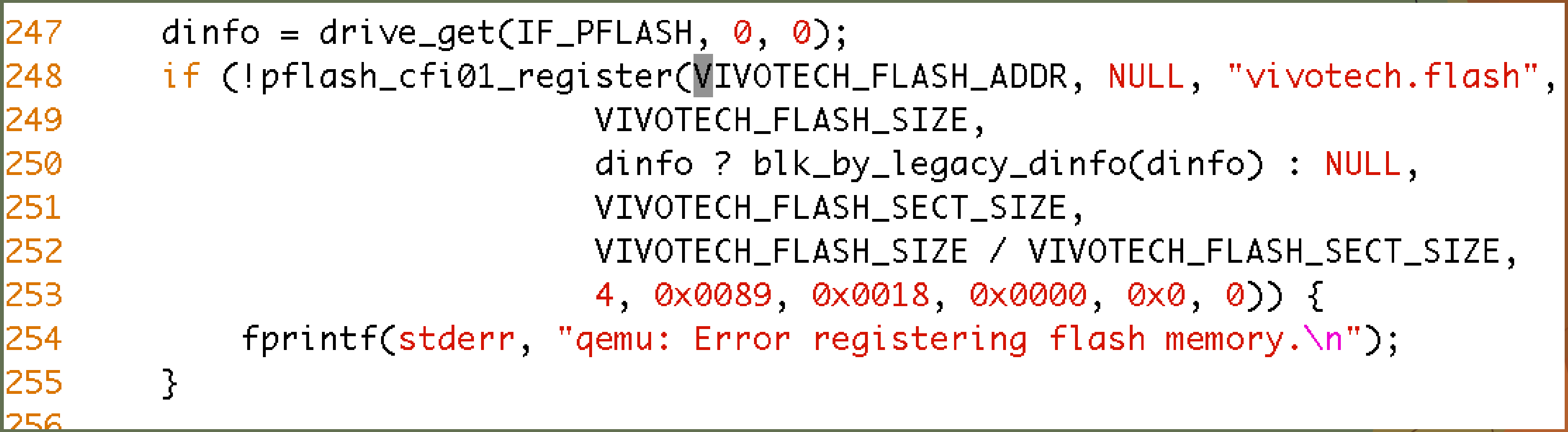


```
    memory_region_init_ram(ram, NULL, "vivotech.ram", machine->ram_size,
                           &error_abort);
    vmstate_register_ram_global(ram);
        /* ??? RAM should repeat to fill physical memory space.  */
    /* SDRAM at address zero.  */
    memory_region_add_subregion(sysmem, VIVOTECH_RAM_ADDR, ram);
```

Initializing your ROM

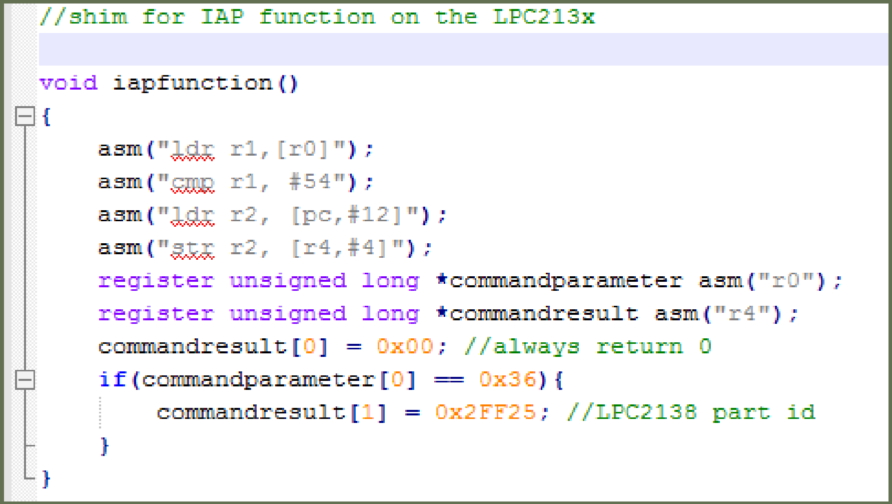


```
#define VIVOTECH_FLASH_ADDR 0x00000000
#define VIVOTECH_FLASH_SIZE (512 * 1024)
#define VIVOTECH_FLASH_SECT_SIZE 512
```



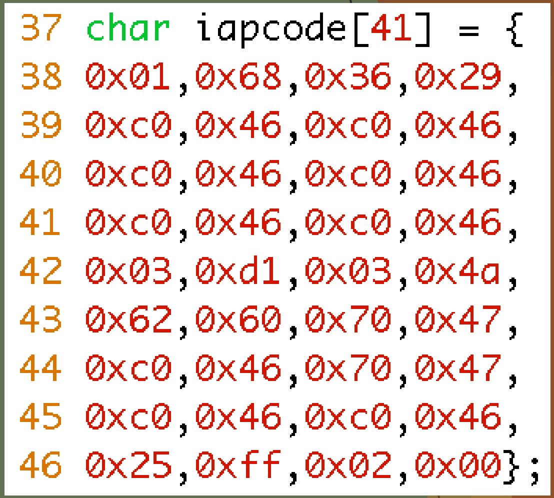
```
    dinfo = drive_get(IF_PFLASH, 0, 0);
    if (!pflash_cfi01_register(VIVOTECH_FLASH_ADDR, NULL, "vivotech.flash",
                          VIVOTECH_FLASH_SIZE,
                          dinfo ? blk_by_legacy_dinfo(dinfo) : NULL,
                          VIVOTECH_FLASH_SECT_SIZE,
                          VIVOTECH_FLASH_SIZE / VIVOTECH_FLASH_SECT_SIZE,
                          4, 0x0089, 0x0018, 0x0000, 0x0, 0)) {
        fprintf(stderr, "qemu: Error registering flash memory.\n");
    }
```

Shell Code for ROM functions

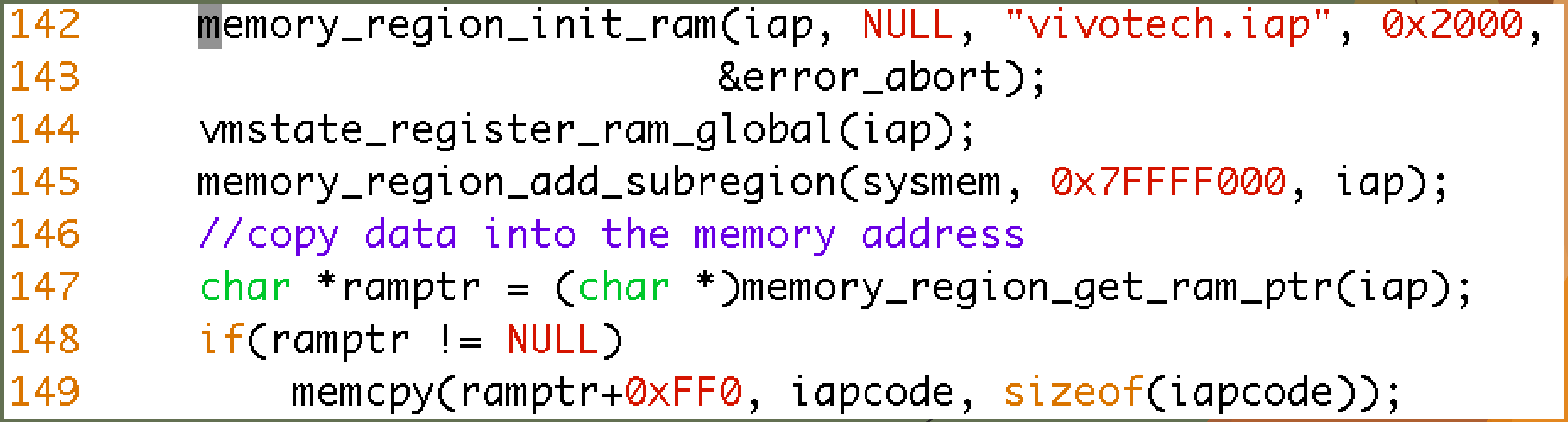


```
//shim for IAP function on the LPC213x

void iapfunction()
{
    asm("ldr r1,[r0]");
    asm("cmp r1, #54");
    asm("ldr r2, [pc,#12]");
    asm("str r2, [r4,#4]");
    register unsigned long *commandparameter asm("r0");
    register unsigned long *commandresult asm("r4");
    commandresult[0] = 0x00; //always return 0
    if(commandparameter[0] == 0x36){
        commandresult[1] = 0x2FF25; //LPC2138 part id
    }
}
```



```
char iapcode[41] = {
0x01,0x68,0x36,0x29,
0xc0,0x46,0xc0,0x46,
0xc0,0x46,0xc0,0x46,
0xc0,0x46,0xc0,0x46,
0x03,0xd1,0x03,0x4a,
0x62,0x60,0x70,0x47,
0xc0,0x46,0x70,0x47,
0xc0,0x46,0xc0,0x46,
0x25,0xff,0x02,0x00};
```



```
    memory_region_init_ram(iap, NULL, "vivotech.iap", 0x2000,
                           &error_abort);
    vmstate_register_ram_global(iap);
    memory_region_add_subregion(sysmem, 0x7FFFF000, iap);
    //copy data into the memory address
    char *ramptr = (char *)memory_region_get_ram_ptr(iap);
    if(ramptr != NULL)
        memcpy(ramptr+0xFF0, iapcode, sizeof(iapcode));
```

QEMU Command Line

```
qemu-system-arm -M vivotech -pflash flash.img -m 32k -nographic -S -s
```

- -M = machine – in this case its our vivotech platform
- -pflash = our binary flash image
- -m = guest ram (in our case 32k)
- -nographic = no display
- -S = freeze CPU at startup
- -s = start the GDB server

So why not just use hardware?

- Full control over execution – can single step, read memory, insert code anywhere.
- Hardware may not have debugging enabled.
- Hardware may be slow and resource constrained
- Can run multiple instances on the same machine

Future Stuff I want to do

- Convince SoC vendors to ship their own QEMU image platforms for testing.
- Use QEMU to set up fuzzing farms for embedded systems
- Slowly working on implementing basic crash handler for embedded arm systems