Faculté des Sciences & Techniques — Université de Limoges

Master 2nde année

TMC

Pico Boot

Étude du démarrage du RP2040

1 – Allez dans le « Chapter 02 » du dépôt :
`https://github.com/LifeWithDavid/RaspberryPiPico-BareMetalAdventures`

a. Que pouvez vous dire des rapports entre les fichiers « `assembly.s` » et « main.c » ?

> **Attention**
> Assurez vous de disposer du « pico-sdk ».

On va compiler le projet présent dans ce répertoire :

xterm
```
$ export PICO_SDK_PATH=~/pico-sdk
$ cp ~/pico-sdk/external/pico_sdk_import.cmake .
$ cmake -B build .
$ cd build
$ make
```

Vous deverez avoir un échec du « linker *» car il essaie de relier des objets qui ne sont pas tous en instructions sur 16bits «thumb».*
Vous modifierez le source du fichier « `assembly.s` » :

```
//bare metal assembly blinking routine
.global start
.thumb_func ⟸ à ajouter
start:

//releases the peripheral reset for iobank_0
    ldr r0, =rst_clr    // atomic register for clearing reset controller (0x4
000c000+0x3000)
```

Vous pourrez alors compiler le firmware avec succès :

xterm
```
$ make clean
$ make
```

Vous obtiendrez un elf : « `first_asm_demo1.elf` ».

Que se passe-t-il au boot ?

Step	Where	What it does
Reset	bootrom 0x00000000	The vector table there gives SP at 0x0 and the reset handler at 0x4.
Stage 1	bootrom	Reads the core ID from SIO CPUID (0xd0000000). Core 1 parks and waits on the FIFO. Core 0 checks watchdog scratch for a “vector boot”, then checks BOOTSEL (the QSPI CS pin). Otherwise it reads 256 bytes of flash over the SSI in slow mode (cmd 0x03) into 0x20041f00, the top 256 B of SRAM5.
Check	bootrom	Computes a CRC-32 over bytes 0..251 and compares it with the word at offset 252. If they match, it jumps to 0x20041f00 (Thumb). If not, it retries with other SPI modes and then falls back to USB/BOOTSEL.
Stage 2 (boot2)	SRAM 0x20041f00	Sets up the XIP SSI for fast QSPI (e.g. 0xEB quad, continuous read). Then it runs exit_from_boot2.S: VTOR ← 0x10000100, MSP ← [0x10000100], bx [0x10000104].
crt0 / runtime_init	flash (XIP)	clocks_init() moves from ROSC (6 MHz) to XOSC 12 MHz → PLL_SYS 125 MHz / PLL_USB 48 MHz. Then main() runs.

Analyse du boot de second stage

On va utiliser renode :

xterm
```
(monitor) include @boards/initialize_raspberry_pico.resc
(raspberry_pico) sysbus LoadELF @/BUILD/first_asm_demo1.elf
21:01:59.9664 [INFO] piocpu1: Setting PC value to 0x100001E9.
(raspberry_pico) sysbus LoadELF @bootroms/rp2040/b2.elf
21:02:05.4018 [INFO] sysbus: Loading block of 16384 bytes length at 0x0.
21:02:05.4019 [INFO] sysbus: Loading block of 3052 bytes length at 0x50100400.
21:02:05.4060 [INFO] piocpu0: Setting PC value to 0x0.
21:02:05.4060 [INFO] piocpu1: Setting PC value to 0x0.
(raspberry_pico) cpu0 PC 0x100001e9
(raspberry_pico) cpu1 PC 0x100001e9
(raspberry_pico) machine StartGdbServer 3333
(raspberry_pico) logLevel -1 sysbus.gpio.led
```

Puis avec `gdb` :

xterm
```
$ gdb-multiarch -ex "target extended-remote :3333" first_asm_demo1.elf
```

On va arrêter le fonctionnement du firmware au calcul du CRC :

xterm
```
(gdb) hbreak *0x20041f00 ⇐ pour mettre un «hardware break»
(gdb) x/wx 0x20041ffc
(gdb) x/wx 0x100000fc
(gdb) continue
(gdb) x/wx 0x20041ffc
(gdb) x/wx 0x100000fc
(gdb) dump binary memory boot2_live.bin 0x20041f00 0x20042000
```

On va extraire la partie « `boot2` » du firmware :

xterm
```
$ arm-none-eabi-objcopy -O binary -j .boot2 first_asm_demo1.elf boot2.bin
```

On va calculer le CRC32 sur son contenu avec le programme Python suivant :

```
import struct, sys
def crc32_mpeg2(data, crc=0xFFFFFFFF):
    for b in data:
        crc ^= b << 24
        for _ in range(8):
            crc = ((crc << 1) ^ 0x04C11DB7) if crc & 0x80000000 else (crc << 1)
            crc &= 0xFFFFFFFF
    return crc
d = open(sys.argv[1], "rb").read()
calc, stored = crc32_mpeg2(d[:252]), struct.unpack("<I", d[252:256])[0]
print(f"calc={calc:#010x} stored={stored:#010x} {'OK' if calc == stored else 'BAD'}")
```

b. Est-ce que les différentes valeurs de CRC sont correctes et identiques ?

c. Pouvez vous modifier le CRC pour faire échouer l'exécution du firmware ?

Liens C et Assembleur

2 – Retournez sur le dépôt `https://github.com/carlosftm/RPi-Pico-Baremetal.git`
Étudiez le firmware dans le répertoire : `01_c_blink` :

xterm
```
boot_stage2.ld
c-blink-asm.s
c-blink.list
c-blink-main.c
Makefile
```

a. Quels sont les liens entre C et assembleur ?

b. Vous compilerez le firmware en corrigeant dans le Makefile le chemin de « pad_checksum »

c. Comparez avec les firmware précédemment étudiés.

3 – Dans le répertoire « 02_Flash_2_SRAM_SDK », compilez et étudiez ce firmware.

a. Où est localisé le code du firmware ?

b. Comment est-ce possible ?

c. Quelles sont les tâches du boot de stage 2 ?