

Embedded Systems Reverse Engineering

// WEEK 01

Introduction and Overview of
Embedded Reverse Engineering:
Ethics, Scoping, and Basic Concepts

George Mason University

RP2350 // ARM Cortex-M33

ARM Cortex-M33 Regs

ARM Architecture & Registers

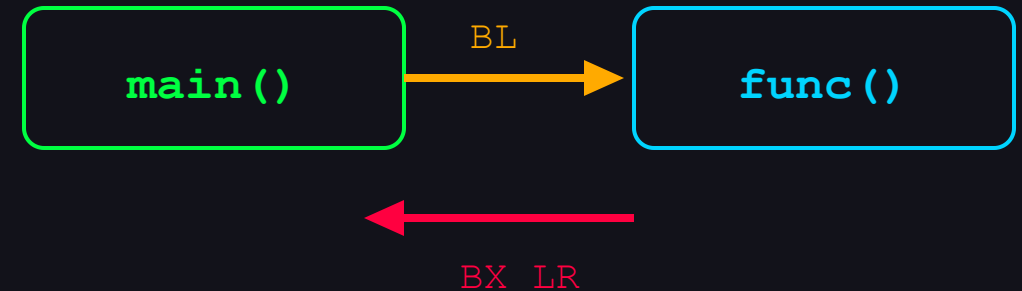
Key Registers

r0	Arg 1 / Return
r1	Arg 2
r2	Arg 3
r3	Arg 4

r0-r3 Caller-saved
r4-r11 Callee-saved

SP (r13)	Stack Ptr
LR (r14)	Return Addr
PC (r15)	Next Instr
xPSR	Status Flags

Function Call Flow



How It Works

r0 = first argument
`puts(r0)` passes the
string address in `r0`

LR saves return addr
BL: PC+4 stored in LR
BX LR: jump back

All registers: 32 bits wide

Stack Growth Direction

ARM Stack Mechanics

Before PUSH

0x20082000

SP here

(empty)

(empty)

(empty)

(empty)

0x20080000

Grows DOWN



After PUSH

0x20082000

(empty)

(empty)

LR value

r3 value

SP here now

SP moved down
by 8 bytes

0x20080000

Key Concepts

Full Descending

SP points to the
last pushed item

PUSH: SP -= 4

Each 32-bit val
drops SP by 4
Two vals = -8

POP: SP += 4

Restores values
SP moves back up

Initial SP

Set by vector
table at 0x00
StackTop=0x20082000

RP2350 Memory Map

RP2350 Address Space

Address Space

ROM Boot	0x0000_0000
XIP Flash 16MB max	0x1000_0000
SRAM 520KB total	0x2000_0000
APB Periph	0x4000_0000
AHB Periph	0x5000_0000
SIO	0xD000_0000
PPB Cortex	0xE000_0000

addr+

Key Details

XIP Flash

Code runs directly
from flash via cache
Execute-In-Place

SRAM Banks

SRAM0-7: 8x64KB
SRAM8-9: 2x4KB
Stack + Heap here

Peripherals

GPIO, UART, SPI
I2C, PWM, ADC
Memory-mapped I/O

SIO + PPB

Single-cycle I/O
Debug + NVIC

Stack vs Heap in RAM

Memory Allocation

SRAM Layout

0x2008_2000 (top)

STACK

Grows DOWN
SP decrements



FREE SPACE



HEAP

Grows UP
malloc expands

.data + .bss

0x2000_0000 (base)

Stack vs Heap

Stack

Local variables
Function args (r0-r3)
Return addresses (LR)
LIFO: last in first out
Auto cleanup on return

Heap

Dynamic allocation
malloc / free in C
Persists until freed
Risk: memory leaks

Collision Risk

If stack grows into
heap = crash / corrupt
No MMU on Cortex-M33
520KB shared space

Link Register & Return

ARM Function Calls

BL Call Flow

Step 1: caller

main: BL add

```
graph TD; S1[Step 1: caller] --> S2[Step 2: hardware saves LR]; S2 --> S3[Step 3: run function]; S3 --> S4[Step 4: return to caller];
```

Step 2: hardware saves LR

LR = return addr

Step 3: run function

add: ADD r0, r1

Step 4: return to caller

BX LR (jump back)

Key Concepts

BL instruction

Branch with Link
Saves return addr
in LR (r14)

BX LR

Branch to addr
stored in LR
Returns to caller

Nested Calls

Must PUSH LR first
PUSH {r3, lr}
POP {r3, pc}
POP into PC = return

r14 = LR

Always check LR in GDB

Program Counter Flow

Instruction Execution

PC Execution

ADDR	INSTRUCTION
0x1000	MOV r0, #5 PC
0x1002	MOV r1, #3
0x1004	ADD r0, r1
0x1006	BL func
0x2000	func: PUSH {lr}
0x2002	SUB r0, #1
0x2004	POP {pc}

PC jumps on BL
then returns via
POP {pc}

Key Concepts

r15 = PC

Points to current
instruction + 4

Prefetch pipeline

Sequential

PC += 2 (Thumb)

PC += 4 (ARM)

Cortex-M33 = Thumb

Branch

B = unconditional

BL = save LR, jump

BX = branch reg

BEQ = branch if Z=1

GDB tip

stepi = step 1 instr

Little-Endian Bytes

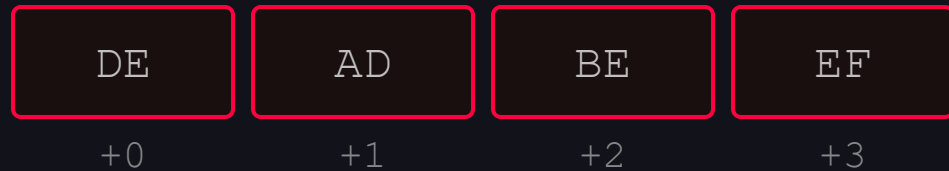
Byte Ordering

Byte Ordering

0xDEADBEEF

Big-Endian

MSB first



vs

Little-Endian

LSB first



Bytes are reversed!

Lowest address holds
least significant byte

ARM uses little-endian

Key Concepts

ARM = Little-Endian

Cortex-M33 uses LE
by default

Also: x86, RISC-V

Why It Matters

Memory dumps show
raw byte order
Must mentally flip
to get true value

GDB Example

x/4xb 0x2000

EF BE AD DE

= 0xDEADBEEF

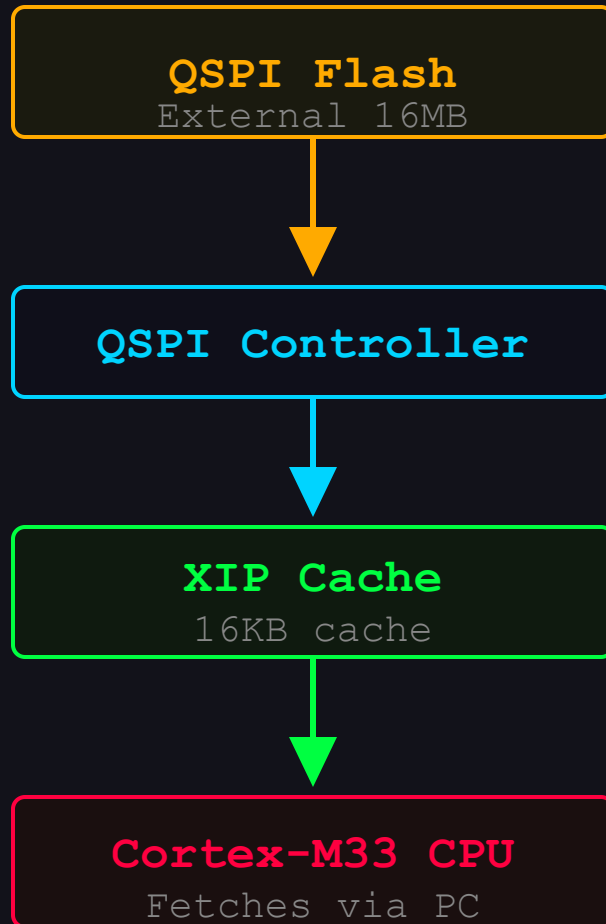
x/xw = word view

GDB auto-corrects

XIP Flash Model

Execute-in-Place

Execute-In-Place



Mapped at
0x1000_0000

Key Details

What is XIP?

Code stays in flash
CPU reads it as if
it were normal memory
No copy to SRAM needed

QSPI Interface

4 data lines
Fast serial flash
CLK, CS, IO0-IO3

Cache Behavior

Cache hit = fast
Cache miss = slow
Flash read latency

RE Insight

Dump flash via SWD

ELF File Structure

Binary Format

ELF Layout

ELF Header

Magic: 7f 45 4c 46

Program Headers

.text

Machine code

.rodata

.data

.bss

.symtab

Section Headers

Section Details

ELF Header

Arch: ARM 32-bit

Entry point addr

Type: executable

.text = code

All instructions

Maps to XIP flash

Disassemble this!

.data / .rodata

.data = initialized

.rodata = constants

.bss = zeroed vars

.symtab = symbols

Function names

Stripped = no names

GDB-OpenOCD Chain

Debug Toolchain

Debug Toolchain



GDB Commands

target remote :3333

Connect to OpenOCD

monitor reset halt

Reset + stop at entry

break main

Set breakpoint

info registers

Dump all regs

SWD Protocol

2 wires only

SWCLK = clock

SWDIO = data

Capabilities

Read/write memory

Read/write regs

Set breakpoints

Full chip control