

Embedded Systems Reverse Engineering

// WEEK 05

Integers and Floats in Embedded Systems:
Debugging and Hacking Integers and Floats
w/ Intermediate GPIO Output Analysis

George Mason University

RP2350 // ARM Cortex-M33

Integer Data Types

Fixed-Size Types for Embedded Systems

uint8_t

Unsigned 8-bit 1 byte

Range: 0 to 255

Ages, counts, always positive

int8_t

Signed 8-bit 1 byte

Range: -128 to 127

Temperature, can be negative

uint16_t

Unsigned 16-bit 2 bytes

Range: 0 to 65,535

Sensor readings, medium values

uint32_t

Unsigned 32-bit 4 bytes

Range: 0 to ~4 billion

Addresses, timestamps

Code Example

```
uint8_t age = 43;           // unsigned, 0-255
int8_t range = -42;         // signed, -128 to 127
```

Key Insight

The **u** prefix means **unsigned** (no negatives)

Without **u** = signed (allows negatives)

Choose the smallest type that fits your data

Two's Complement

How Negative Numbers are Stored

Encoding -42 as int8_t

Step 1: Start

42 = 0x2A

Step 2: Flip

$\sim 0x2A = 0xD5$

Step 3: Add 1

$0xD5 + 1 = 0xD6$

Binary: 00101010 -> 11010101 -> 11010110

Result: -42 stored as 0xD6 in memory

Top bit = 1 means negative

Signed vs Unsigned: Same Bits!

Hex	Binary	uint8_t	int8_t
0x2A	00101010	42	42
0xD6	11010110	214	-42

Same byte 0xD6 = 214 unsigned, -42 signed

GDB Verification

```
(gdb) x/1xb &range
0x200003e7:    0xd6            // -42 in memory
```

Inline Assembly GPIO

Direct Hardware Control via ASM

GPIO Init Loop (pins 16-19)

1. Config Pad

```
PADS_BANK0  
Clear OD+ISO, set IE  
0x40038000
```

2. Set Function

```
IO_BANK0  
FUNCSEL = 5 (SIO)  
0x40028004
```

3. Enable Out

```
GPIO Coprocessor  
mccr p0,#4,r4,r5,c4  
Output Enable
```

Loop: r0 = 16 to 19 Red, Green, Blue, Yellow LEDs

Each pin: pad config + function select + OE

Blink Loop

```
mccr p0,#4,r4,r5,c0
```

```
r4 = pin, r5 = value  
c0 = output value register  
r5=1 ON, r5=0 OFF
```

Pin Cycling

```
pin++;  
if (pin > 18) pin=16;
```

Cycles: 16 -> 17 -> 18 -> 16
Red -> Green -> Blue -> repeat

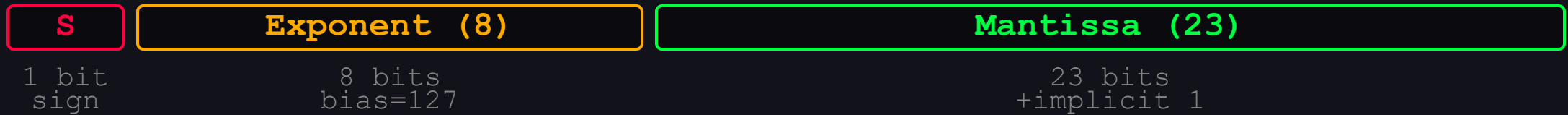
Why Inline Assembly?

gpio_put(16,1) calls **mccr** underneath
Inline ASM shows what the SDK does at hardware level

IEEE 754 Float

32-bit Single Precision Encoding

Float Bit Layout (32 bits)



Decode Formula

Value = $(-1)^{\text{sign}} \times 2^{(\text{exp}-127)} \times (1 + \text{mantissa})$

Sign determines +/- Exponent scales with bias 127 Mantissa adds precision
Special cases: exp=0 or 255 (denorm, inf, NaN)

Decode Example: 42.5

Sign Bit: 0 Positive number
Exponent: 10000100 = 132 decimal
Bias subtraction: $132 - 127 = 5$
Mantissa: 01010100...0 = 1.010101
Combine: $1.010101 \times 2^5 = 42.5$
Hex: 0x422A0000
Binary: 01000010001010100000000000000000

32-bit Storage Comparison

Integer:	Exact values	Range: -2.1B to +2.1B	No decimals
Float:	Approximate values	Range: $\pm 10^{38}$	~7 sig digits

Float in Ghidra

Analyzing 42.5 in the Binary

Decompiled main()

```
int main(void) {  
    stdio_init_all();  
    uVar1 = DAT_1000024c;  
    do {  
        printf(fmt,0,uVar1);  
    } while(true);  
}
```

Key Discovery

printf with %f always
receives a **double** (64-bit)

C promotes float to double
for variadic functions!

42.5 sent as double

Register Pair r2:r3

r2 (low): 0x00000000

r3 (high): 0x40454000

Combined: 0x4045400000000000

= 42.5

Assembly View

```
1000023a    00 24    movs r4, #0x0        // r2 = 0  
1000023c    03 4d    ldr r5,[DAT...]    // r3 = 0x40454000
```

Patching Float

Changing 42.5 to 99.0 in Ghidra

Calculate 99.0 as Double

99 decimal = **1100011** binary
Normalize: **1.100011** $\times 2^6$
Sign: 0 **Exp:** 6+1023 = 1029
Mantissa: 100011000...0
Full double: 0x4058C00000000000

Before (42.5)

r2: **0x00000000**

r3: **0x40454000**

Output: fav_num: 42.500000

After (99.0)

r2: **0x00000000** same!

r3: **0x4058C000** **changed**

Output: fav_num: 99.000000

Patch in Ghidra

1. Window: Bytes

Open byte editor

2. Find 00404540

High word of 42.5

3. Patch 00C05840

High word of 99.0

Only one word to patch (low word is 0)

Double Precision

IEEE 754 64-bit Floating Point

64-Bit Layout

Sign

1 bit

Exponent

11 bits (bias 1023)

Mantissa (Fraction)

52 bits

Formula: $(-1)^S \times 1.\text{Mantissa} \times 2^{(\text{Exp}-1023)}$

Encoding 42.52525

Sign: 0 (positive)

Exponent: 5 + 1023 = 1028 = 0x404 (hex)

Mantissa: 0101010000110011...

Full 64-bit hex: **0x4045433B645A1CAC**

Float (32-bit)

Size: 4 bytes

Exp: 8 bits

Mantissa: 23 bits

Precision: ~7 digits

Bias: 127

1 register (ARM)

Double (64-bit)

Size: 8 bytes

Exp: 11 bits

Mantissa: 52 bits

Precision: ~15 digits

Bias: 1023

2 registers (r2:r3)

Double in Ghidra

Analyzing 42.52525 in memory

Decompiled main()

```
int main(void) {  
    double fav_num  
    = 42.52525;  
    stdio_init_all();  
}
```

Key Difference

Float (42.5):

r2 = 0x00000000 (zero!)

r3 = 0x40454000

Double (42.52525):

r2 = 0x645A1CAC (non-zero!)

Register Pair r2:r3

r3 (HIGH): 0x4045433B

r2 (LOW): 0x645A1CAC

Combined: 0x4045433B645A1CAC = 42.52525

Assembly (ldrd)

```
ldrd r2,r3,[PC,#0x10]  
; Loads BOTH words at once  
; r2 gets low word  
; r3 gets high word
```

ldrd = Load
Register
Doubleword

ARM Cortex-M33

Patching Double

Changing 42.52525 to 99.99

99.99 as IEEE 754 Double

Sign: 0 **Exp:** 6 + 1023 = 1029

Result: **0x4058FF5C28F5C28F**

r3 (HIGH) = 0x4058FF5C

r2 (LOW) = 0x28F5C28F

Before (42.52525)

r3: **0x4045433B**

r2: **0x645A1CAC**

printf: 42.525250

Both words non-zero

After (99.99)

r3: **0x4058FF5C** **changed**

r2: **0x28F5C28F** **changed**

printf: 99.990000

BOTH words changed!

Float Patch

Words changed: **1**

r2 (low): stays 0x0

r3 (high): **patched**

Easier to patch

Double Patch

Words changed: **2**

r2 (low): **patched**

r3 (high): **patched**

Both words must change

IEEE 754 & Data Types

Data Types and IEEE 754 Reference

IEEE 754 Hex Values

Value	Float (32b)	Double (64b)
42.0	0x42280000	0x4045000000000000
42.5	0x422A0000	0x4045400000000000
99.0	0x42C60000	0x4058C00000000000
99.99	0x42C7F5C3	0x4058FF5C28F5C28F
3.14	0x4048F5C3	0x40091EB851EB851F
100.0	0x42C80000	0x4059000000000000

Tip: float low word often 0x0; double low word usually non-zero

Patching Workflow

1. Identify
float / double

2. Check r2
zero = float

3. Calculate
new hex value

4. Patch
in byte editor

5. Export
UF2 + test

Integer Types

uint8_t: 0-255

int8_t: -128 to 127

Two's complement for signed

Key Insight

Float: patch 1 word

Double: patch 2 words

Check r2 to detect type