

Embedded Systems Reverse Engineering

// WEEK 04

Variables in Embedded Systems:
Debugging and Hacking Variables
w/ GPIO Output Basics

George Mason University

RP2350 // ARM Cortex-M33

What is a Variable?

Labeled Boxes in Memory (SRAM)

Memory – A Row of Numbered Boxes



Anatomy of a Declaration

```
uint8_t age = 42;
```

`uint8_t`
`age`

Data type (1 byte)

Variable name (label)

`= 42`

`;`

Initial value

End of statement

Key Concepts

Declaration name + type

Definition allocates memory

Initialization assigns value

Important Rule

You MUST declare a variable BEFORE you use it!

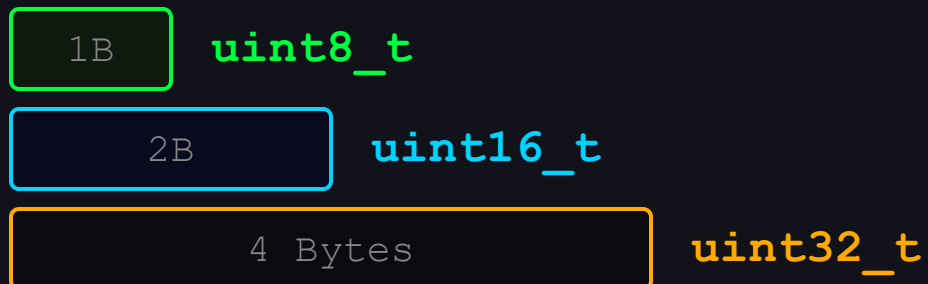
Compiler needs to know the type

Data Types & Sizes

How Much Memory Each Type Uses

Type	Size	Range	Description
uint8_t	1 byte	0 – 255	Unsigned 8-bit
int8_t	1 byte	-128 – 127	Signed 8-bit
uint16_t	2 bytes	0 – 65,535	Unsigned 16-bit
int16_t	2 bytes	-32,768 – 32,767	Signed 16-bit
uint32_t	4 bytes	0 – 4,294,967,295	Unsigned 32-bit
int32_t	4 bytes	-2.1B – 2.1B	Signed 32-bit

Size Comparison



Memory Sections

Where Variables Live After Compilation

.data

Flash -> copied to RAM at startup

Contains: Initialized global/static variables

```
int counter = 42;
```

Initial value stored in flash, copied to SRAM by data_cpy

.bss

RAM – zeroed at startup

Contains: Uninitialized global/static variables

```
int counter;
```

NOT stored in binary (saves space!) – memset to 0 at boot

.rodata

Flash – read only

Contains: Constants and string literals

```
const int MAX = 100;
```

Lives in flash permanently – cannot be modified at runtime

.data

RAM

Writable

Initialized globals

.bss

RAM

Writable

Uninitialized globals (zeroed)

.rodata

Flash

Read-only

Constants & strings

GPIO Basics

General Purpose Input/Output on RP2350

Pico 2 GPIO Pins

GPIO 16	————	Red LED
GPIO 17	————	Green LED
GPIO 18	————	Blue LED
GPIO 25	————	Onboard LED

Software-controlled switches

Pico SDK Functions

`gpio_init(pin)` Init pin

`gpio_set_dir(pin,d)` I/O dir

`gpio_put(pin,val)` Set H/L

`sleep_ms(ms)` Delay

Basic LED Blink Code

```
#define LED_PIN 16
int main(void) {
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    while (true) {
        gpio_put(LED_PIN, 1);    // ON
        sleep_ms(500);
        gpio_put(LED_PIN, 0);    // OFF
        sleep_ms(500);
    }
}
```

Ghidra Binary Analysis

Analyzing a Raw .bin Without Symbols

1. Import

File -> Import

Language:

ARM Cortex 32 LE

Block: **.text**

Base: **10000000**

XIP address for RP2350

2. Analyze

Auto-Analyze: Yes

Ghidra finds:

FUN_1000019a

FUN_10000210

FUN_10000234

Auto-generated names

3. Resolve

Edit Function Sig

Rename to:

data_cpy

frame_dummy

main

Fix signatures

Decompiled main() in Ghidra

Before Resolving:

```
void FUN_10000234(void) {
    FUN_10002f54();
    do {
        FUN_100030e4(
            DAT_10000244, 0x2b);
    } while(true);
}
```

After Resolving:

```
int main(void) {
    stdio_init_all();
    do {
        printf(
            "age: %d\r\n"
            , 0x2b);
    } while(true);
}
```

Compiler Optimization

Why Your Variable Disappeared

Source Code

```
int main(void) {  
    uint8_t age = 42;  
    age = 43;  
    stdio_init_all();  
    while (true)  
        printf("age: %d", age);  
}
```

Compiler Thinks...

age = 42 is NEVER read

Dead store -> REMOVED

age = 43 -> constant fold

Replaces variable with literal

Resulting Assembly

1000023a 2b 21 movs r1, #0x2b ; 0x2b = 43

No age=42 instruction — compiler removed it

Key Takeaway

Source Code

```
age = 42  
age = 43
```

->

Binary

```
movs r1, #0x2b
```

Compiler

Optimizes dead
stores away!

Binary Patching

Changing Values in the Binary

Before Patch

```
1000023a  2b 21  movs r1,#0x2b
```

0x2b = 43 decimal

Output: **age: 43**

Compiler-optimized constant

After Patch

```
1000023a  46 21  movs r1,#0x46
```

0x46 = 70 decimal

Output: **age: 70**

Changed program behavior!

How to Patch in Ghidra

1. Find Instr

->

2. Rt-Click

->

3. Patch Val

->

Done!

Patch Instruction: change operand

Export Patched Binary

File: Export

Format: Raw Bytes

Save as *-h.bin

Exported binary has your patches

GPIO Hacking

Patching GPIO 16 to GPIO 17

Original: GPIO 16

Red LED on pin 16

```
1000023a 10 20 movs r0,#0x10
```

```
10000244 10 23 movs r3,#0x10
```

```
10000252 10 24 movs r4,#0x10
```

0x10 = 16, three locations

Patched: GPIO 17

Green LED on pin 17

```
1000023a 11 20 movs r0,#0x11
```

```
10000244 11 23 movs r3,#0x11
```

```
10000252 11 24 movs r4,#0x11
```

0x11 = 17, all patched!

What Each Patch Controls

```
gpio_init r0
```

```
gpio_set_dir r3
```

```
gpio_put r4
```

ALL pin refs must be patched

Bonus: Change Print Value

```
00 21 movs r1,#0x0 age: 0
```

->

```
42 21 movs r1,#0x42 age: 66
```

Changed value: 0 to 66 (0x42)

GPIO Coprocessor

RP2350 Single-Cycle I/O via mcrr

mcrr Instruction Breakdown

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

mcrr Move to Coprocessor (2 regs)
p0 Coprocessor 0 (GPIO)

r4 GPIO pin number
r5 Value (0=LOW, 1=HIGH)

Output Value (c0)

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

r4 = pin number

r5 = 0 or 1

Controls GPIO output state

Output Enable (c4)

```
mcrr p0, #4, r4, r5, c4
```

r4 = pin number

r5 = 1 (enable output)

Sets pin direction to OUTPUT

gpio_init(16) Sequence

Step 1: Config Pad

```
addr 0x40038044
```

Step 2: Set Func

```
FUNCSEL = 5 (SIO)
```

Step 3: Enable Out

```
mcrr p0, #4, r4, r5, c4
```

Pad: clear OD, set IE, clear ISO

SIO = fast single-cycle GPIO access

Full Patching Pipeline

End-to-End Binary Hacking Workflow

1 Import .bin

Ghidra: Import

ARM Cortex 32 LE

Base: 0x10000000

2 Analyze

Auto-analyze

Rename functions

Fix signatures

3 Find Target

Listing window

Find `movs rN,#val`

Identify bytes to change

4 Patch

Right-click:

Patch Instruction

Change operand value

5 Export

File: Export

Format: Raw Bytes

Save as *-h.bin

6 Convert UF2

uf2conv.py

--family 0xe48bfff59

RP2350 family ID

UF2 Command

```
python uf2conv.py file.bin --base 0x10000000 -o hacked.uf2
```

Flash to Pico 2

1. Hold BOOTSEL + USB
 2. Drop hacked.uf2
 3. Pico reboots hacked
- RPI-RP2 drive in BOOTSEL

Key Sections

.text	Flash	Code
.rodata	Flash	Constants
.data	RAM	Init globals
.bss	RAM	Zeroed globals