

Embedded Systems Reverse Engineering

// WEEK 11

Structures and Functions in
Embedded Systems: Debugging and Hacking
w/ IR Remote Control & NEC Protocol

George Mason University

RP2350 // ARM Cortex-M33

C Structures (Structs)

Grouping Related Data Together

What is a Struct?

A user-defined type that groups related variables under one name

Like a form with multiple fields
-- each field holds different data

Struct Definition

```
typedef struct {  
    uint8_t led1_pin;  
    uint8_t led2_pin;  
    uint8_t led3_pin;  
    bool led1_state;  
    bool led2_state;  
    bool led3_state;  
} simple_led_ctrl_t;
```

Why Use Structs?

1. Organization
Related data stays together
2. Readability
Code easier to understand
3. Scalability
Easy to add more features
4. Pass to Functions

simple_led_ctrl_t leds

pin1: 16

pin2: 17

pin3: 18

state1: 0

state2: 0

state3: 0

Struct Memory Layout

How Structs Are Stored and Accessed

Memory Layout (6 bytes total)

Address	Member	Size	Value
0x20000000	led1_pin	1 byte	16 (0x10)
0x20000001	led2_pin	1 byte	17 (0x11)
0x20000002	led3_pin	1 byte	18 (0x12)
0x20000003	led1_state	1 byte	0 (false)
0x20000004	led2_state	1 byte	0 (false)
0x20000005	led3_state	1 byte	0 (false)

Dot Operator (.)

Use with struct variable

```
leds.led1_pin = 16;  
leds.led1_state = true;
```

Arrow Operator (->)

Use with pointer to struct

```
ptr->led1_pin = 16;  
// same as (*ptr).led1_pin
```

Designated Initializers

```
simple_led_ctrl_t leds = {  
    .led1_pin = 16, .led2_pin = 17, .led3_pin = 18  
};
```

Clear which value goes
to which member
Order doesn't matter

NEC IR Protocol

Infrared Remote Control Communication

How IR Works

Remote sends invisible light pulses
IR receiver on GPIO 5 reads signal

IR LED flashes in specific patterns
Each pattern = different button

NEC Protocol Frame (32 bits)

Leader

Address

Addr Inv

Command

Cmd Inv

Stop

9ms+4.5ms

8-bit

8-bit check

8-bit

8-bit check

Leader says "attention!", address identifies device, command is the button pressed

NEC Command Codes

Button	NEC Code	LED
1	0x0C	Red (GP16)
2	0x18	Green (GP17)
3	0x5E	Yellow (GP18)

Hardware Wiring

GPIO 5	IR Receiver (VS1838B)
GPIO 16	Red LED + 220 ohm
GPIO 17	Green LED + 220 ohm
GPIO 18	Yellow LED + 220 ohm

Projects: 0x0023_structures and 0x0026_functions

Functions in C

Reusable Blocks of Code

Anatomy of a Function

```
int ir_to_led_number(int ir_command) {
```

```
^^^  ^^^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^
```

```
ret  function name      parameter
```

```
if (ir_command == 0x0C) return 1;    // body + return value
```

Function Types

Type	Example
No params, no ret	leds_all_off()
Params, no return	blink_led(..)
No params, return	ir_getkey()
Params + return	ir_to_led_num()
Struct pointer	get_led_pin()

Key Functions

```
ir_to_led_number(cmd)
```

Maps NEC code to LED 1/2/3

```
get_led_pin(leds, num)
```

Returns GPIO pin for LED

```
blink_led(pin, cnt, ms)
```

Blinks LED cnt times

Function Call Chain

```
main()    --> process_ir_led_command()
```

1. leds_all_off() Turn all LEDs off

2. ir_to_led_number() Map NEC to LED

3. get_led_pin() Get GPIO pin

4. blink_led() Blink + stay on

Struct Pointers in Functions

Passing Data Efficiently

Why Pass by Pointer?

Efficient	4 bytes (address) not 6
Modifiable	Function can change original
Standard	Embedded systems practice

Arrow Operator

`leds->led1_pin`
Same as `(*leds).led1_pin`
Use `->` when `leds` is a pointer

leds_all_off()

```
void leds_all_off(
    simple_led_ctrl_t *leds) {
    gpio_put(leds->led1_pin, 0);
    gpio_put(leds->led2_pin, 0);
}
```

blink_led()

```
void blink_led(uint8_t pin,
               uint8_t count, uint32_t ms){
    gpio_put(pin, true);
    sleep_ms(ms);
}
```

process_ir_led_command() -- Main Command Processor

```
int process_ir_led_command(int cmd,
    simple_led_ctrl_t *leds, uint8_t blink_count) {
    leds_all_off(leds);                // turn all off first
    int num = ir_to_led_number(cmd);    // map NEC to LED
    blink_led(get_led_pin(leds, num),    // blink then stay on
}
```

Structures Source Code

0x0023_structures.c

Full Source

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "ir.h"

typedef struct {
    uint8_t led1_pin, led2_pin, led3_pin;
    bool led1_state, led2_state, led3_state;
} simple_led_ctrl_t;

int main(void) {
    stdio_init_all();
    simple_led_ctrl_t leds = {
        .led1_pin=16, .led2_pin=17, .led3_pin=18
    };
    gpio_init(leds.led1_pin);    // init 16, 17, 18
    ir_init(5);                  // IR on GPIO 5
    while (true) {               // main loop
```

Main Loop Flow

```
ir_getkey()  --> check NEC code --> set state --> gpio_put()
0x0C=LED1 (red)  0x18=LED2 (green)  0x5E=LED3 (yellow)
```

Struct Flattening

How Compilers Transform Structs

C Code (High Level)

```
gpio_init(leds.led1_pin);  
// leds.led1_pin = 16  
gpio_init(leds.led2_pin);  
// leds.led2_pin = 17
```

Assembly (Flattened)

```
movs r0, #0x10    // 16  
bl gpio_init  
movs r0, #0x11    // 17  
bl gpio_init
```

The Key Insight

Struct abstraction DISAPPEARS at assembly level
You see individual values (16, 17, 18) not struct names

Struct Member Mapping

Assembly	Struct Member	Physical	NEC Code
0x10 (16)	led1_pin	Red LED	0x0C
0x11 (17)	led2_pin	Green LED	0x18
0x12 (18)	led3_pin	Yellow LED	0x5E

Sequential values (16,17,18) reveal the struct pattern
Recognize patterns to reconstruct original structs in Ghidra

Hacking Structures

Swapping GPIO Pin Assignments

Swap LED 1 and LED 2

Find gpio_init values:

0x10 (16) --> **0x11 (17)**

0x11 (17) --> **0x10 (16)**

Swap the two byte values

Result After Hack

Before:

Btn 1 --> GPIO 16 --> Red

Btn 2 --> GPIO 17 --> Green

After: SWAPPED!

Before (Normal)

Btn 1 (0x0C) --> GPIO 16 **Red**

Btn 2 (0x18) --> GPIO 17 **Green**

Log and LED match correctly

After (Hacked)

Btn 1 (0x0C) --> GPIO 17 **Green!**

Btn 2 (0x18) --> GPIO 16 **Red!**

Log says RED but GREEN lights

Log Desynchronization

Terminal Log: NEC command: 0x0C (expects Red)

Physical LED: **GREEN LED on** **MISMATCH!**

Operator sees correct logs but WRONG behavior

Stuxnet: False "normal" data to operators, equipment destroyed

.ELF vs .BIN Analysis

Ghidra Analysis of 0x0026_functions

.ELF vs .BIN Comparison

Feature	.BIN File	.ELF File
Symbols	None	Function names
Sections	Raw bytes only	.text .data .rodata
Debug info	None	May include debug
Use case	Flash to device	Analysis + debug

Importing .BIN

Manual setup required:
ARM Cortex 32 little endian
Block: .text
Base: 10000000
No function names

Importing .ELF

Auto-detected by Ghidra:
ARM format recognized
Sections auto-loaded
Symbol tree populated
Named functions visible

Important Rule

Analyze the .ELF for symbol information Patch the .BIN for flashing

Export Workflow

Patch .bin in Ghidra --> uf2conv.py --> flash to Pico 2

Structs & IR Protocol

Structs, Functions, IR, and Hacking

C Structures

Group related data together

Dot (.) for variables

Arrow (->) for pointers

Designated init: .pin = 16

NEC IR Protocol

32-bit frame: addr + cmd

0x0C Button 1

0x18 Button 2

0x5E Button 3

Functions

Reusable blocks, one job each

ir_to_led_number()

get_led_pin() / blink_led()

process_ir_led_command()

Assembly Flattening

Structs vanish in assembly

Only see values: 0x10 0x11 0x12

Pattern recognition is key

.ELF has symbols, .BIN doesn't

Key Values

0x10000234 main()

GPIO 5 IR receiver

GPIO 16/17/18 Red/Green/Yellow

Hacking Techniques

GPIO swap 0x10 <--> 0x11

Log desync Logs lie!

Stuxnet Same concept

Projects

0x0023_structures

0x0026_functions

Key Takeaway

Patch bytes, mislead logs

hardware does what YOU say