Embedded Systems

Mobile Robotics

Anthony J. Clark

Today

- Our robot’s task
- Project workflow
- Hardware Systems
- Software Systems
- Arduino Ecosystem

Our Robot’s Task

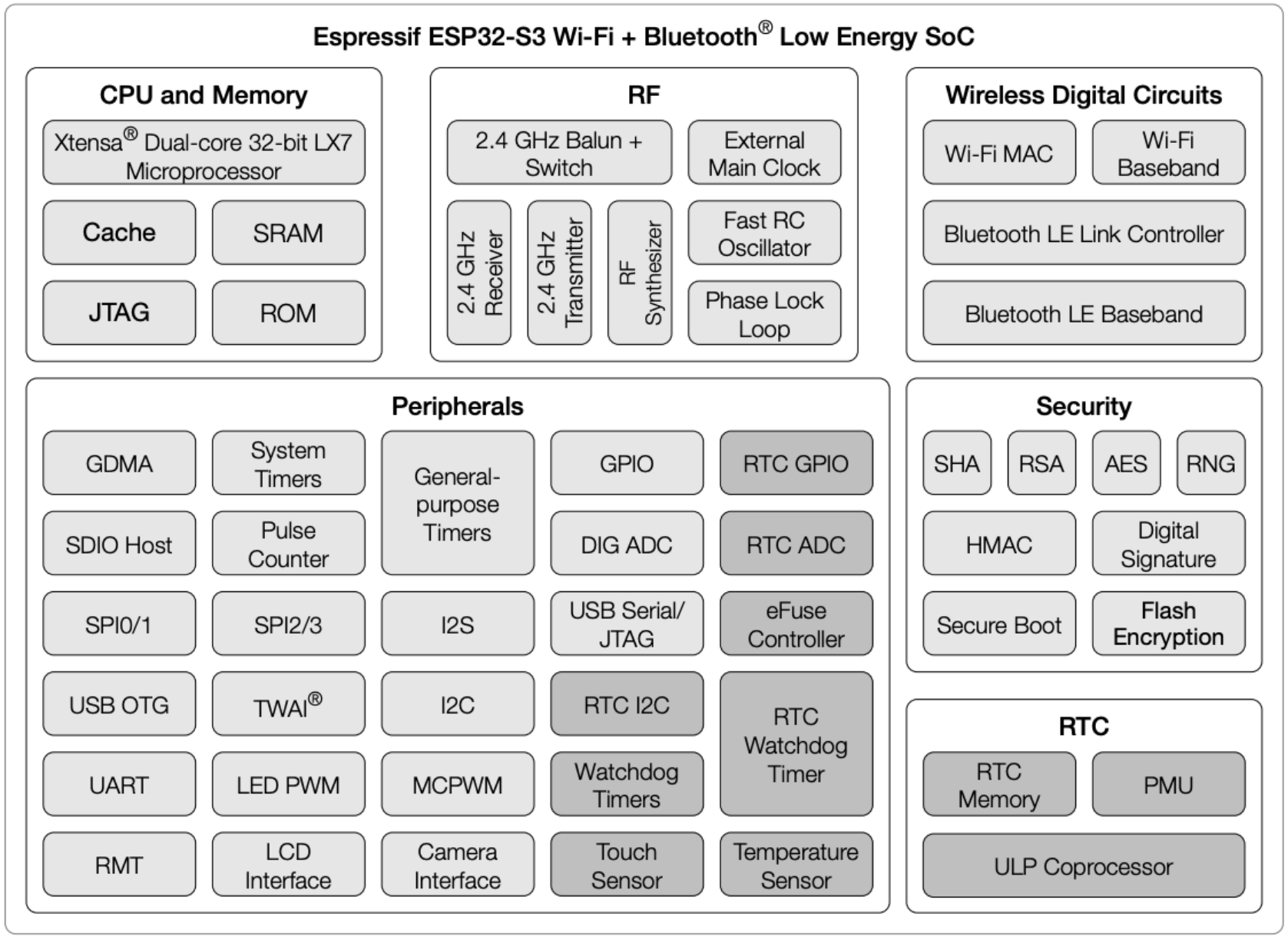
Project Workflow

- Specify the requirements
- Design your software
- Program your software
- Upload your program
  - Connected using a “programmer”
  - Connected with USB using a bootloader
  - Connected over Wi-Fi using OTA
- Monitor and direct your robot using WebSockets

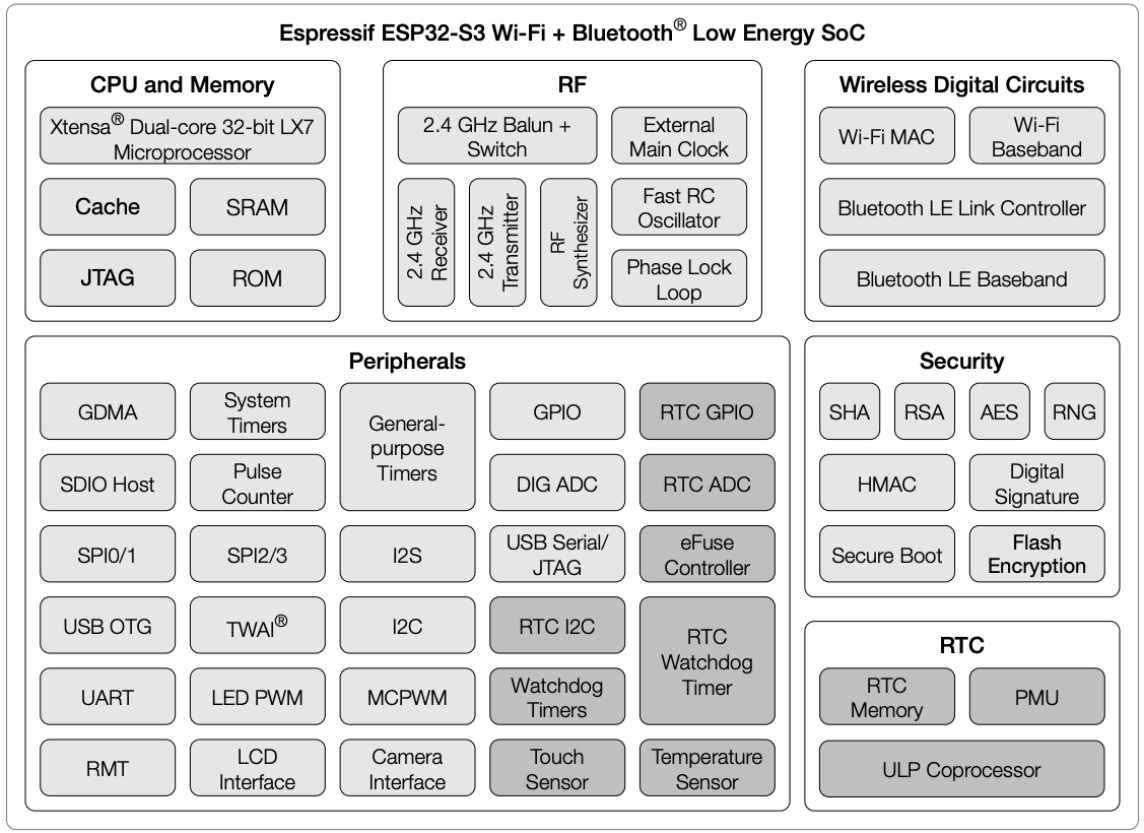
Embedded Systems

Hardware System

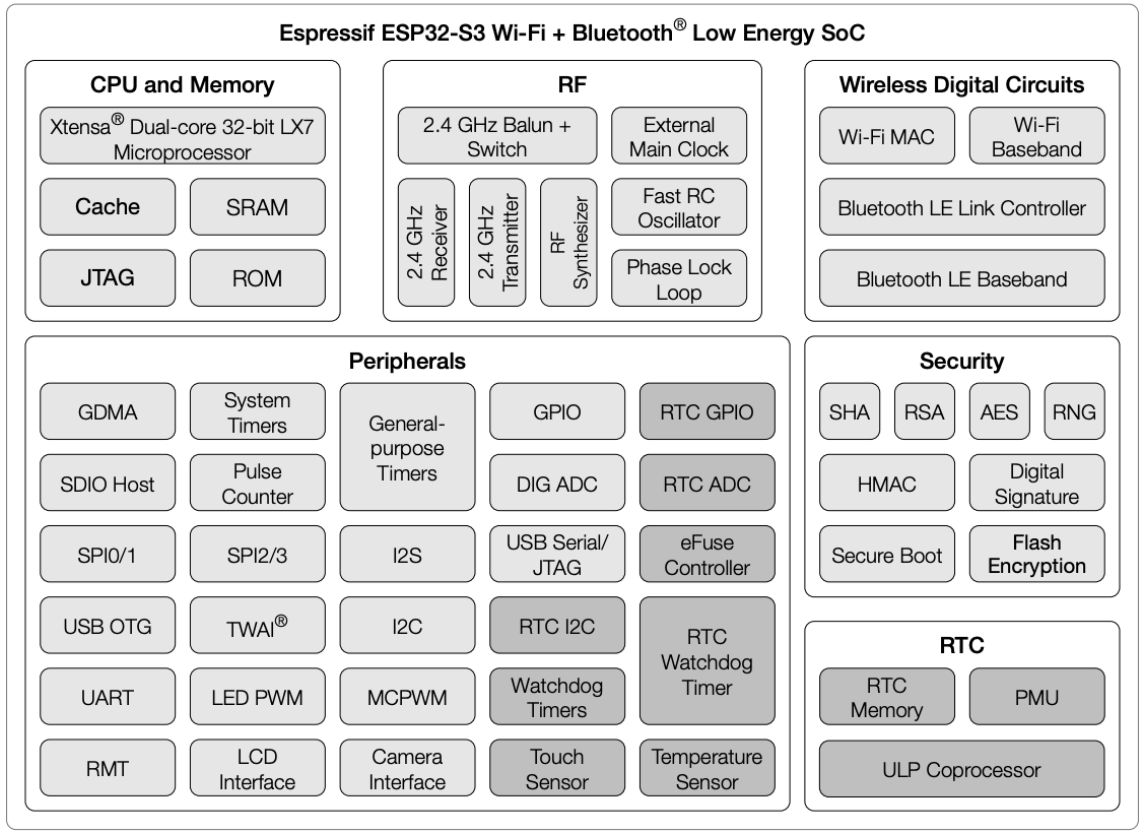
- CPU: Xtensa® dual-core 32-bit LX7 microprocessor (240 MHz)
- Cache: 16 KB instruction and 32 KB data
  - four-way set associative
  - 16 bytes block size
- Internal memory: 384 KB ROM and 512 KB SRAM
- External memory: 8 MB PSRAM and 8 MB Flash


Espressif ESP32-S3 Wi-Fi + Bluetooth® Low Energy SoC
CPU and Memory
Xtensa® Dual-core 32-bit LX7 Microprocessor
Cache
SRAM
JTAG
ROM
RF
2.4 GHz Balun + Switch
External Main Clock
2.4 GHz Receiver
2.4 GHz Transmitter
RF Synthesizer
Fast RC Oscillator
Phase Lock Loop
Wireless Digital Circuits
Wi-Fi MAC
Wi-Fi Baseband
Bluetooth LE Link Controller
Bluetooth LE Baseband
Peripherals
GDMA
System Timers
General-purpose Timers
GPIO
RTC GPIO
SDIO Host
Pulse Counter
DIG ADC
RTC ADC
SPI0/1
SPI2/3
I2S
USB Serial/ JTAG
eFuse Controller
USB OTG
TWAI®
I2C
RTC I2C
RTC Watchdog Timer
UART
LED PWM
MCPWM
Watchdog Timers
RMT
LCD Interface
Camera Interface
Touch Sensor
Temperature Sensor
Security
SHA
RSA
AES
RNG
HMAC
Digital Signature
Secure Boot
Flash Encryption
RTC
RTC Memory
PMU
ULP Coprocessor

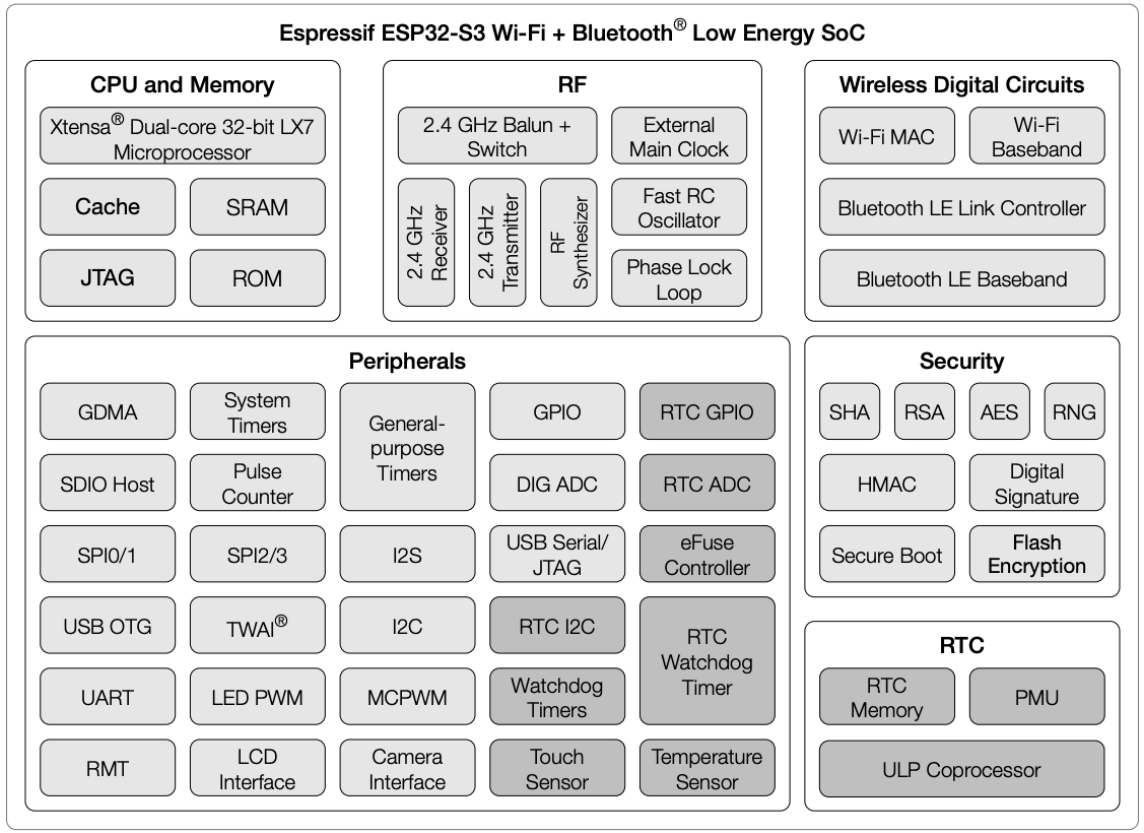
Motor Driver Interface



Other Peripherals Using I²C

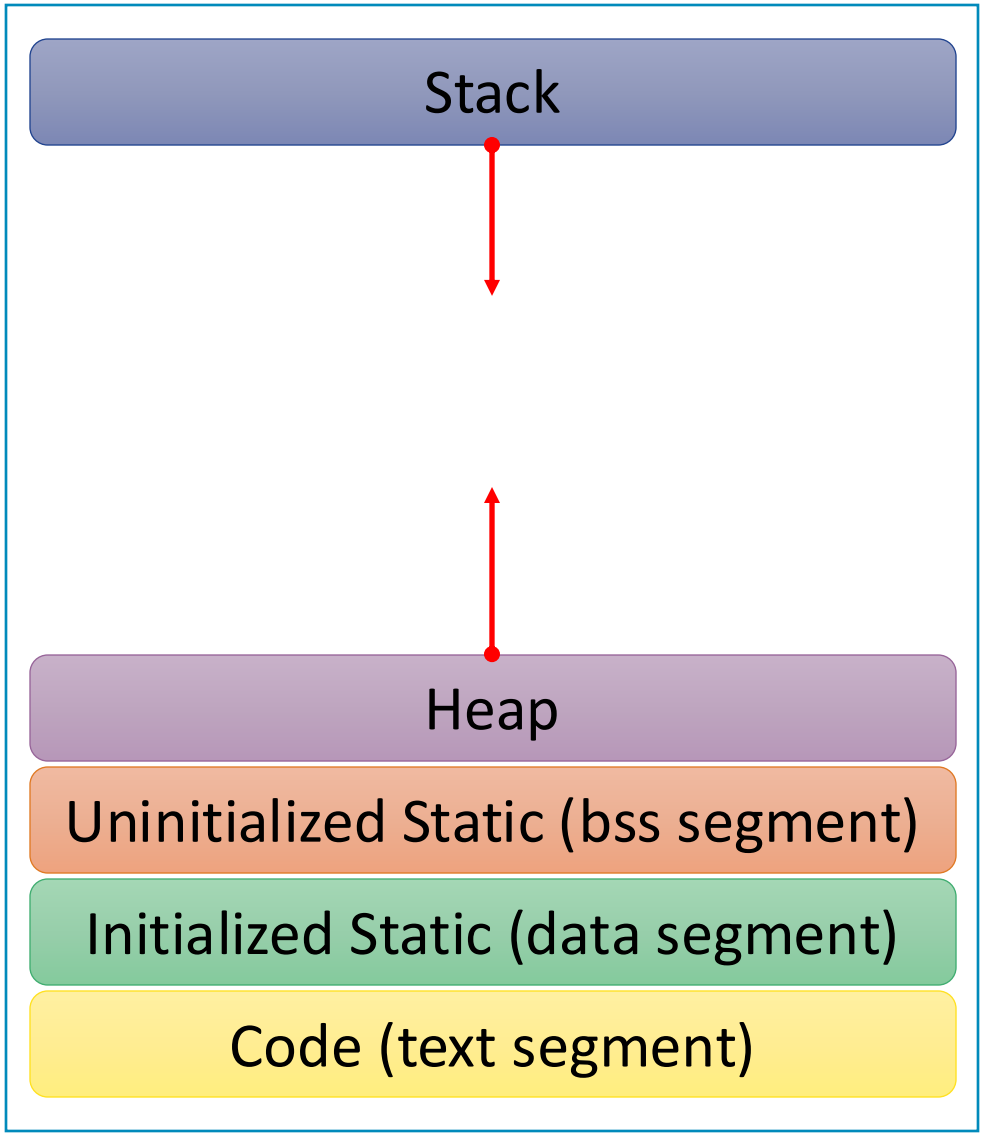


Wi-Fi Communication



Hardware/Software Interface

High Memory Address



Low Memory Address

External Memory

- Flash
- SPI RAM

Internal Memory

- Cache
- dRAM
- iRAM
- ROM

On Reset (Jumps to Bootloader)

Toit (and Jaguar) vs. Bare Metal (ESP-IDF)

Blinking With Toit

```
import gpio

main:
  pin := gpio.Pin 32 --output
  while true:
    pin.set 0
    sleep --ms=500
    pin.set 1
    sleep --ms=500
```

Fibers: Independent Control Flow

```
import gpio

LED1 ::= gpio.Pin.out 17
LED2 ::= gpio.Pin.out 18

main:
  task:: my-task-1
  task:: my-task-2

my-task-1:
  while true:
    sleep --ms=500
    LED1.set 1
    sleep --ms=500
    LED1.set 0

my-task-2:
  while true:
    sleep --ms=123
    LED2.set 1
    sleep --ms=123
    LED2.set 0
```

Why Leave Toit?

```
[jaguar] INFO: program 8358aee2-45c6-0193-6318-b75de00f59f3 stopped
[jaguar] INFO: program c6925192-2d45-548f-7f78-726e3c457cad started
Heap report @ out of memory in primitive 6:2:
```

Bytes	Count	Type
40960	12	toit processes
16384	4	system 0 0ff85ae2-63d2-e530-c9fe-c844785c0276
20480	5	other 1 31709de1-906c-ce7d-8698-87e163711ebb
4096	1	current 7 c6925192-2d45-548f-7f78-726e3c457cad
667648	1	heap metadata
4096	1	spare new-space
16808	7	heap overhead
192	4	thread/other

```
Total: 729704 bytes in 16 allocations (8%), largest free 7452k, total free 7471k

********************************************************************************
Decoding by `jag`, device has version <2.0.0-alpha.159>
********************************************************************************
MALLOC_FAILED error.
0: Bus.test <sdk>/i2c.toit:65:5
1: Bus.scan <sdk>/i2c.toit:60:10
2: main display.toit:23:18
********************************************************************************

[jaguar] ERROR: program c6925192-2d45-548f-7f78-726e3c457cad stopped - exit code 1
```

Blinking With ESP-IDF

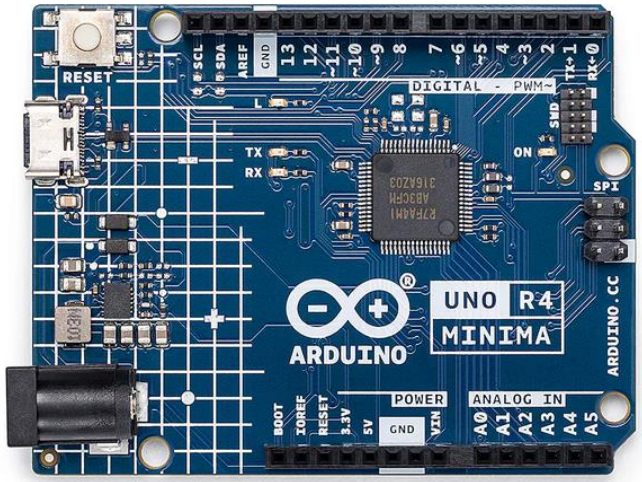
```c
void app_main(void) {
    configure_led();

    while (1) {
        blink_led();
        s_led_state = !s_led_state;
        vTaskDelay(CONFIG_BLINK_PERIOD / portTICK_PERIOD_MS);
    }
}
```

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "led_strip.h"
#include "sdkconfig.h"

#define BLINK_GPIO CONFIG_BLINK_GPIO
static uint8_t s_led_state = 0;

static void blink_led(void) {
    gpio_set_level(BLINK_GPIO, s_led_state);
}

static void configure_led(void) {
    gpio_reset_pin(BLINK_GPIO);
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
}
```

https://github.com/espressif/esp-idf/blob/master/examples/get-started/blink/main/blink_example_main.c

Arduino Ecosystem

- Bootloader (otherwise, must use a hardware programmer)
- Open-source hardware (kits, boards, and shields)
- Open-source software (many Arduino libraries)



Arduino® UNO R4 Minima

Blinking an LED (Light-Emitting Diode)

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000)
}
```

Arduino Bootloader

```
#include <Arduino.h>

int atexit(void (* /*func*/ )()) { return 0; }
void initVariant() __attribute__((weak));
void initVariant() { }


void setupUSB() __attribute__((weak));
void setupUSB() { }

int main(void) {
    init();

    initVariant();

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }

    return 0;
}
```

https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/main.cpp

https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/main.cpp

Embedded Software

- Don’t block other tasks with delays
- Deal with hardware and time constrains
- Use an event-driven approach

Blinking is Blocking

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000)
}
```

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  setupMotors();
}

void loop() {
  if shouldHaltMotors() haltMotors();
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000)
}
```

Blinking Without Blocking

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000)
}
```

```
const int blinkInterval = 1000;
int ledState = LOW;
unsigned long previousMillis = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= blinkInterval) {
    previousMillis = currentMillis;
    ledState = (ledState == LOW) ? HIGH : LOW;
    digitalWrite(LED_BUILTIN, ledState);
  }
}
```

Blinking Without Blocking

```
const int blinkInterval = 1000;
int ledState = LOW;
unsigned long previousMillis = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  setupMotors();
}

void loop() {
  if shouldHaltMotors() haltMotors();

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= blinkInterval) {
    previousMillis = currentMillis;
    ledState = (ledState == LOW) ? HIGH : LOW;
    digitalWrite(LED_BUILTIN, ledState);
  }
}
```

Pseudocode For Software Architecture

```
global state for module 1
global state for module 2
…

set update period for module 1
set update period for module 2
…

setup
  initialize non-trivial global states

loop
  short update/poll for module 1
  short update/poll for module 2
  …
```

Pseudocode For Software Architecture

```
global state for module 1
global state for module 2
…

set update period for module 1
set update period for module 2
…

setup
  initialize non-trivial global states

loop
  short update/poll for module 1
  short update/poll for module 2
  …
```

```
const int blinkInterval = 1000;
int ledState = LOW;
unsigned long previousMillis = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  setupMotors();
}

void loop() {
  if shouldHaltMotors() haltMotors();

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= blinkInterval) {
    previousMillis = currentMillis;
    ledState = (ledState == LOW) ? HIGH : LOW;
    digitalWrite(LED_BUILTIN, ledState);
  }
}
```