Cheat sheet

Embedded C

Bit manipulation, registers, volatile, interrupts, ring buffers and fixed-point maths for microcontrollers.

1 Fixed-width types

Include

```
#include <stdint.h>
#include <stdbool.h>
```

Types

```
uint8_t  int16_t  uint32_t
int64_t  bool
```

Same size on every compiler, unlike int

Unsigned literals

```
1U   0xFFU   1UL << 31
```

Shifting a signed 1 into bit 31 is undefined

2 Bit manipulation

Set bit n

```
reg |= (1U << n);
```

Clear bit n

```
reg &= ~(1U << n);
```

Toggle bit n

```
reg ^= (1U << n);
```

Read bit n

```
bool on = (reg >> n) & 1U;
```

Write a field

```
reg = (reg & ~(0x7U << 4)) |
((val & 0x7U) << 4);
```

Clear the 3-bit field at bit 4, then set it

Is power of two

```
bool p2 = x && !(x & (x -
1));
```

3 Registers and volatile

Memory-mapped register

```
#define GPIOA_ODR (*(volatile
uint32_t *)0x40020014U)
GPIOA_ODR |= (1U << 5);
```

STM32F4 GPIOA output register

Why volatile

```
volatile bool data_ready;
```

Tells the compiler the value can change outside normal code (hardware, ISR)

Read-only constant data

```
static const uint8_t table[]
= {1, 2, 4, 8};
```

const data can stay in flash

4 Interrupts

Keep ISRs short

```
void EXTI0_IRQHandler(void) {
  EXTI->PR = 1U;        //
clear flag
  button_pressed = true;
}
```

Set a flag; do the work in the main loop

Main loop

```
while (1) {
  if (button_pressed) {
    button_pressed = false;
    handle_button();
  }
}
```

Protect shared data

```
__disable_irq();
uint32_t copy = shared_count;
__enable_irq();
```

Multi-byte reads can be torn by an ISR

5 Non-blocking timing

Instead of delay()

```
uint32_t last = 0;
if (millis() - last >= 500) {
  last += 500;
  toggle_led();
}
```

Subtraction stays correct when the counter wraps

6 Ring buffer

Buffer

```
#define N 64   // power of two
uint8_t buf[N];
volatile uint8_t head, tail;
```

Put (in ISR)

```
uint8_t next = (head + 1) & (N - 1);
if (next != tail) { buf[head] = c; head = next; }
```

Get (in main)

```
if (tail != head) {
  c = buf[tail];
  tail = (tail + 1) & (N - 1);
}
```

One writer and one reader need no lock

7 Pointers and structs

Function pointer

```
void (*callback)(uint8_t);
callback = on_rx;
callback(42);
```

Struct over registers

```
typedef struct {
  volatile uint32_t SR, DR, BRR, CR1;
} UART_Regs;
#define UART1 ((UART_Regs *)UART1_BASE)
```

Order and gaps must match the reference manual

Size of an array

```
sizeof(arr) / sizeof(arr[0])
```

Only works on the array, not a pointer to it

8 Fixed-point (Q15)

Format

```
int16_t, value = x / 32768
```

Range −1 to just under +1

Multiply

```
int16_t y = (int16_t)(((int32_t)a * b) >> 15);
```

Use a 32-bit product, then shift back

Convert

```
int16_t q = (int16_t)(0.5f * 32768);   // 16384
```