Cortex-M4 / STM32F4 Bare-Metal Cheat Sheet

1. Basic Data Types

Type	Size	Notes
`uint8_t`	1B	Unsigned 8-bit
`int8_t`	1B	Signed 8-bit
`uint16_t`	2B	Unsigned 16-bit
`int16_t`	2B	Signed 16-bit
`uint32_t`	4B	Unsigned 32-bit
`int32_t`	4B	Signed 32-bit
`uint64_t`	8B	Unsigned 64-bit
`int64_t`	8B	Signed 64-bit

---

2. Register Access

```
#define REG_ADDR 0x40021000
#define REG      (*(volatile uint32_t *)REG_ADDR)

REG = 0x01;        // Write
uint32_t val = REG; // Read
```

- `volatile` prevents compiler optimizations.
- Direct memory-mapped access (specific to STM32F4 reference manual).

---

3. Bit Manipulation

```
REG |= (1 << 5);   // Set bit 5
REG &= ~(1 << 3);  // Clear bit 3
REG ^= (1 << 7);   // Toggle bit 7
if (REG & (1 << 2)) { /* bit 2 is set */ }
```

- Useful for peripheral configuration and status checks.

---

4. Delays

```
void delay(volatile uint32_t count) {
    while(count--) { __asm__("NOP"); }
}
```

- Simple busy-wait loop.
- For precise timing, use **SysTick timer**.

---

5. GPIO Initialization Example

```
#include "stm32f4xx.h"

// Enable GPIOA clock
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

// Set PA5 as output
GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;
GPIOA->MODER |= GPIO_MODER_MODE5_0;
```

- Use STM32F4 reference manual for correct register bits.

---

6. Inline Assembly

```
__asm__("NOP");            // No operation
__asm__("MOV R0, #1");     // ARM-specific instruction
__asm__("VADD.F32 S0, S0, S1"); // FPU single-precision add
```

- Only use FPU instructions if your MCU has an FPU.

---

7. Interrupts

```
void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_UIF) {
        TIM2->SR &= ~TIM_SR_UIF; // Clear interrupt flag
    }
}

// Enable TIM2 interrupt in NVIC
NVIC_EnableIRQ(TIM2_IRQn);
```

- Cortex-M4 uses NVIC for nested interrupts.
- Always clear the peripheral’s interrupt flag in the handler.

---

8. Memory Sections (Linker Usage)

```
extern uint32_t _estack; // Defined in linker script
__attribute__((section(".my_section"))) int my_var;
```

- Place variables in specific memory (SRAM, CCM, or Flash).
- Useful for bootloader, DMA buffers, or critical code.

---

9. Startup / Main

```
void Reset_Handler(void) {
    // Initialize .data and .bss sections
```

```
    main();
}

int main(void) {
    while(1) {
        // Main loop
    }
}
```

- Cortex-M4 requires correct stack pointer (`_estack`) in vector table.
- No OS, no `stdio` by default.

---

10. Common Macros

```
#define BIT(x) (1U << (x))
#define SET_BIT(REG,BIT) ((REG) |= (BIT))
#define CLEAR_BIT(REG,BIT) ((REG) &= ~(BIT))
#define TOGGLE_BIT(REG,BIT) ((REG) ^= (BIT))
```

- Useful shorthand for register operations.

---

11. CMSIS / STM32F4 Examples

```
#include "stm32f4xx.h"

// Turn on GPIOA clock and set PA5 as output
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
GPIOA->MODER |= GPIO_MODER_MODE5_0;

// Toggle PA5
GPIOA->ODR ^= BIT(5);
```

- CMSIS provides structured access and avoids magic addresses.

---

12. Tips for Cortex-M4

- Use `volatile` for all hardware registers.
- Prefer **SysTick** or timers for accurate delays.
- Understand memory map: Flash, SRAM, CCM (Core-Coupled Memory), peripherals.
- Minimize work in interrupt handlers.
- Initialize FPU if using floating-point math.
- Keep startup code minimal: stack pointer, .data, .bss, vector table.