Peripherals & Registers: Quick Program Flows

1) GPIO (Output / Input / Debounce)

Goal: drive LED, read button.

Flow

1. Enable GPIO clock
2. Configure pin mode (output push-pull / input + pull-up/down)
3. (Input) add debounce (timer or simple delay/state filter)
4. Use ODR/LAT to write; IDR/PORT to read

STM32 regs: RCC->APB2ENR, GPIOx->CRL/CRH (mode), GPIOx->ODR/IDR, (BSSR/BRR)
PIC18 regs: TRISx (dir), LATx (write), PORTx (read), INTCON2.RBPU (pull-ups on PORTB)

Debounce (software)

if (raw != last) { t_start = now; last = raw; }

if (now - t_start > 10ms) stable = raw;

2) External Interrupt (button → EXTI)

Flow

1. Enable GPIO & AFIO clock (STM)
2. Configure pin as input
3. Map pin to EXTI line (STM)
4. Configure edge (rising/falling)
5. Clear pending; enable EXTI + NVIC
6. ISR: set a flag

STM32 regs: AFIO->EXTICR, EXTI->IMR/RTSR/FTSR/PR, NVIC_EnableIRQ
PIC18 regs: INT0/1/2 edge control (INTCON2), INTCON/INTCON3 enable flags, ISR in high/low priority

3) Timer — Periodic Interrupt (SysTick-like)

Flow

1. Enable timer clock
2. Set prescaler + ARR/top
3. Enable update interrupt
4. NVIC enable (STM) / PIE bit (PIC)
5. Start counter
6. ISR: toggle flag / tick++

STM32 regs: RCC->APB1/2ENR, TIMx->PSC/ARR/DIER/SR/CR1
PIC18 regs: TMRx, TMRxIE/TMRxIF, T0CON/T1CON/T2CON (prescale), PR2 (for TMR2)

4) PWM (Timer Output Compare)

Flow

1. Timer clock enable
2. Configure PWM channel mode
3. Set period (ARR) + duty (CCR)
4. Route channel pin to AF (STM) / CCPx pin (PIC)
5. Enable channel + counter

STM32 regs: TIMx->PSC/ARR/CCRn/CCMRn/CCER/BDTR, GPIO AF mode
PIC18 regs: CCPxCON (PWM mode), PR2, T2CON, CCPRxL/CCP bits, TRIS for output pin

5) Input Capture (measure frequency/duty)

Flow

1. Timer clock enable
2. Configure capture on channel (edge)
3. Enable capture interrupt
4. In ISR: read CCR, compute delta

STM32 regs: TIMx->CCMRn (IC), CCER (edge), DIER/SR, CNT/CCRn
PIC18 regs: CCPxCON (capture modes), CCPxIF/IE, TMR1 as timebase

6) UART — Polling TX/RX

Flow

1. Enable GPIO & UART clocks
2. Set TX pin AF-push-pull, RX input
3. Configure baud, word length, parity, stop bits
4. Enable UE/TE/RE (STM) / TXEN/SPEN/CREN (PIC)
5. TX: wait TXE/TSR empty → write DR/TXREG
6. RX: wait RXNE/RCIF set → read DR/RCREG

STM32 regs: RCC->APB2/APB1ENR, GPIO AF, USARTx->BRR/CR1/CR2/SR/DR
PIC18 regs: SPBRG/BRGH (baud), TXSTA (TXEN), RCSTA (SPEN/CREN), TXREG/RCRG, PIR1.TXIF/RCIF

7) UART — Interrupt-Driven (ring buffer)

Flow

1. Same init as polling
2. Enable RXNE interrupt (and TXE if using TX IRQ)
3. NVIC/PIE enable
4. ISR:
    - RX: read DR → push to rx_ring
    - TX: if bytes pending → write DR else disable TXE int

STM32 regs: USARTx->CR1 (RXNEIE/TXEIE), SR/DR
PIC18 regs: PIE1.RCIE/TXIE, PIR1.RCIF/TXIF, TXREG/RCREG

8) ADC — Single Conversion (polling)

Flow

1. Enable ADC + GPIO clocks; set pin analog mode
2. Select channel, sample time
3. Turn on ADC; start conversion
4. Wait EOC; read data

STM32 regs: RCC ENR, GPIO analog mode, ADCx->SQR/SMPR/CR2, SR/DR
PIC18 regs: ANSEL/ANSELH (analog select), ADCON0/ADCON1/ADCON2, GO/DONE, ADRESH:ADRESL

9) ADC — Continuous with DMA

Flow

1. Configure ADC in scan/continuous
2. Set up DMA: src=ADC_DR, dst=buffer[], length=N, circular
3. Enable DMA req from ADC
4. Start ADC; process buffer in main/half-complete ISR

STM32 regs: ADCx->CR1/CR2 (SCAN/CONT/DMA), DMA1->ChannelX CCR/CPAR/CMAR/CNDTR, NVIC for DMA

10) I²C — Master Write/Read

Flow

1. Enable I²C + GPIO clocks; SDA/SCL open-drain + pull-ups
2. Program freq/clock (I²C speed)
3. START → send address+R/W → wait ACK
4. Write/Read bytes with ACK/NACK correctly
5. STOP

STM32 regs: I2C1/2->CR2/CCR/TRISE, SR1/SR2, DR; GPIO open-drain
PIC18 regs: SSPCON1/2, SSPSTAT, SSPBUF (MSSP module), PIR1.SSPIF, SEN/RSEN/PEN/ACKEN bits

11) SPI — Master Transfer

Flow

1. Enable SPI + GPIO clocks; SCK/MOSI AF, MISO input; CS (manual GPIO)
2. Configure CPOL/CPHA, baud prescaler, MSB/LSB
3. Enable SPI
4. Assert CS → write DR/SSPBUF → wait RXNE/BF → read; deassert CS

STM32 regs: SPIx->CR1/CR2, SR, DR; GPIO AF; manual CS via GPIO
PIC18 regs: SSPCON1 (SPI mode), SSPSTAT, SSPBUF, TRIS for pins

12) Watchdog (WDT)

Flow

1. Enable WDT with appropriate prescaler/timeout
2. In main loop, **kick** (reload) periodically
3. On timeout → system reset → log cause

STM32 regs: IWDG->KR/PR/RLR; RCC_CSR flags for reset cause
PIC18 regs: WDT enabled via configuration bits; CLRWDT() in code; RCON for reset flags

13) RTC (basic timekeeping)

Flow

1. Enable LSE/LSE-like clock for RTC
2. Enter RTC config mode, set prescalers/time/date
3. Enable; read time via registers; optionally second interrupt/alarm

STM32 regs: Backup/RTC domain, RCC->BDCR, RTC->CRH/CRL/PRL/COUNTER
PIC18: Use TMR1 + external 32.768 kHz crystal for “soft RTC” (no native RTC on many PIC18s)

14) NVIC / Interrupts (STM32) & Priority (PIC18)

Flow

1. Clear peripheral pending flag
2. Enable periph interrupt in peripheral
3. NVIC_EnableIRQ(Periph_IRQn) (STM) / set PIE & GIE (PIC)
4. Set priority if needed
5. ISR: clear flag first, then handle

STM32: NVIC_ISER, NVIC_IPR; periph SR flags
PIC18: RCON IPEN (priority), INTCON GIE/GIEH/GIEL, PIE*/PIR* bits

15) DMA (generic recipe, STM32)

Flow

1. Enable DMA clock
2. Disable channel; set CPAR (periph), CMAR (mem), CNDTR (len)
3. Configure CCR: dir, minc, pinc, circular/normal, interrupt enables
4. Enable channel
5. ISR: check HT/TC/TE; update pointers/flags

Regs: DMA1/2 ChannelX: CCR, CNDTR, CPAR, CMAR; ISR/IFCR; NVIC

16) Low-Power Entry/Exit

Flow

1. Ensure wake sources configured (EXTI/RTC/UART)
2. Disable unused clocks/peripherals
3. Execute WFI/WFE (STM) / SLEEP (PIC)
4. On wake ISR: clear flag, resume clocks if needed

STM32: PWR->CR, SCB->SCR (SLEEPDEEP), RCC clock gating
PIC18: Sleep instruction, peripherals with wake-up capability, OSCCON states

17) Error/Timeout Pattern (reliable polling)

```
#define WAIT_UNTIL(cond, TO) do{ uint32_t t=0; while(!(cond)){ if(++t>(TO)) break; } }while(0)
```

Use for flags like TXE/RXNE/EOC to avoid deadlocks.

18) ISR Template (minimal)

```
volatile uint8_t flag_evt;


void PERIPH_IRQHandler(void){
  if (PERIPH->SR & FLAG){ PERIPH->SR = ~FLAG;  // clear
    /* read DR if needed */
    flag_evt = 1;
  }
}
```

Mini Checklists (print these)

UART (polling)

- Clocks (GPIO, UART)
- TX AF, RX input
- Baud/format
- UE/TE/RE (or TXEN/SPEN/CREN)
- TX: wait TXE → DR; RX: wait RXNE → DR

I²C (master)

- Clocks + OD pulls
- Speed (CCR), TRISE
- START → Addr → ACK
- TX/RX bytes with ACK/NACK
- STOP

SPI (master)

- Clocks + AF pins
- CPOL/CPHA + prescaler
- CS low
- Write DR → wait RXNE → read
- CS high

ADC (single)

- Analog mode pin
- Channel + sample time
- ON → Start
- Wait EOC → read

Timer (PWM)

- PSC/ARR
- CCMR to PWM mode
- CCR duty
- CCER enable + Counter start

Embedded C Program Structure — Function Prototypes & Flow

1. GPIO (LED + Button)

```
// Function Prototypes
void gpio_init(void);
void led_on(void);
void led_off(void);
int button_read(void);

int main() {
    gpio_init();
    while(1) {
        if(button_read()) led_on();
        else led_off();
    }
}
```

---

2. Timer (Delay / PWM)

```
// Function Prototypes
void timer_init(void);
void delay_ms(unsigned int ms);
void pwm_init(void);
void pwm_set_duty(unsigned int duty);

int main() {
    timer_init();
    pwm_init();
    while(1) {
        pwm_set_duty(25);  // 25% duty
        delay_ms(1000);
```

```
        pwm_set_duty(75);  // 75% duty
        delay_ms(1000);
    }
}
```

---

3. UART

```
// Function Prototypes
void uart_init(void);
void uart_tx_char(char c);
void uart_tx_string(char *s);
char uart_rx_char(void);

int main() {
    uart_init();
    uart_tx_string("Hello World\r\n");
    while(1) {
        char c = uart_rx_char();
        uart_tx_char(c); // echo back
    }
}
```

---

4. ADC

```
// Function Prototypes
void adc_init(void);
unsigned int adc_read(unsigned char channel);

int main() {
    adc_init();
    while(1) {
```

```
        unsigned int value = adc_read(0); // channel 0
        // process value
    }
}
```

---

5. I²C (RTC or EEPROM)

```
// Function Prototypes
void i2c_init(void);
void i2c_start(void);
void i2c_stop(void);
void i2c_write(unsigned char data);
unsigned char i2c_read_ack(void);
unsigned char i2c_read_nack(void);

// Example: RTC DS1307 Init
void rtc_init(void);
void rtc_write(void);
void rtc_read(void);
```

---

6. SPI (Sensor / EEPROM)

```
// Function Prototypes
void spi_init(void);
unsigned char spi_transfer(unsigned char data);

int main() {
    spi_init();
    spi_transfer(0x55);  // example write
    while(1);
}
```

---

7. LCD (16x2 Character)

```
// Function Prototypes
void lcd_init(void);
void lcd_cmd(unsigned char cmd);
void lcd_data(unsigned char data);
void lcd_string(char *s);
void lcd_goto_xy(int row, int col);


int main() {
    lcd_init();
    lcd_string("Hello");
    while(1);
}
```

---

8. RTC (with I²C)

```
// Function Prototypes
void rtc_init(void);
void rtc_set_time(unsigned char hr, unsigned char min, unsigned char sec);
void rtc_get_time(unsigned char *hr, unsigned char *min, unsigned char *sec);
void rtc_display(void); // sends to LCD/UART
```

Universal Coding Flow

No matter the MCU, the embedded code flow is always:

1. **Peripheral Init** → (gpio_init, uart_init, i2c_init...)
2. **Helper Functions** → (lcd_cmd, uart_tx_char, adc_read...)
3. **Application Logic (main)** → where you combine them.

This way, for every new concept, you just **fill in the body with MCU-specific register code**, but structure stays the same.