GPIO Application

CSE 3105 – Computer Interfacing & Embedded System

Md. Nasif Osman Khansur

Lecturer

Dept. of CSE, RUET

Lighting up an LED using GPIO

- The software initialization involves **two** key steps:
  - First, it enables the clock of the GPIO port B via the RCC module.
  - Second, it configures pin 2 of GPIO port B as a general-purpose output pin, with the output type as **push-pull**.
  - To light up the red LED, we need to output logic "1" to pin 2.

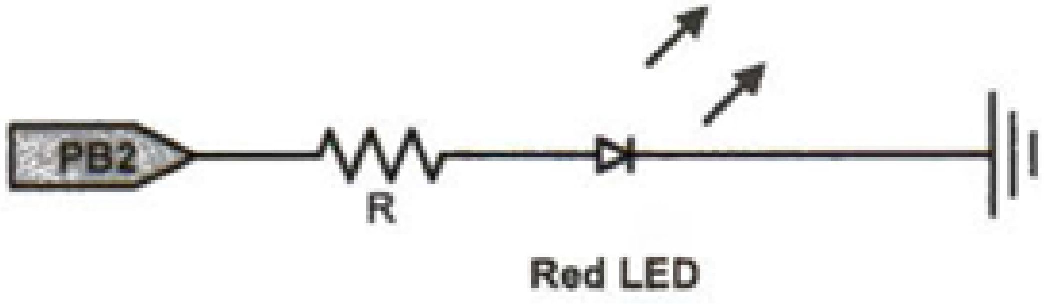

Figure: Connection diagram between a processor pin and LED

Lighting up an LED using GPIO

- In assembly, a load-modify-store sequence is required to change the register value stored in memory.
- Also, we can use "EQU" directive to create symbols for the GPIO B base address and ODR register offset, which make the assembly program more readable and self-documenting.
- The following is an example:

```
GPIOB_BASE EQU 0x48000400        ; Base memory address
GPIO_ODR EQU 20                  ; Byte offset of ODR from the base
LDR r7, =GPIOB_BASE              ; Load GPIO port B base address
LDR rl, [r7, #GPIO_ODR]          ; Read GPIOB->ODR
ORR rl, rl, #(1«6)               ; Set bit 6
STR rl, [r7, #GPIO_ODR]          ; Write to GPIOB->ODR
```

Lighting up an LED using GPIO

- We need to enable the clock of GPIO port B.
- To save energy, every peripheral's clock is turned off by default.
- We can enable the clock of a peripheral by setting the corresponding bit of the clock control register defined in the reset and clock control (RCC) structure.

```
// Reset and clock control
typedef struct {
__ IO uint32_t CR;          // Clock control register
__ IO uint32_t ICSCR;       // Internal clock sources calibration register
__ IO uint32_t CFGR;        // Clock configuration register
__ IO uint32_t AHBlENR;     // AHB 1 peripheral clocks enable register
__ IO uint32_t AHB2ENR;     // AHB 2 peripheral clocks enable register
__ IO uint32_t AHB3ENR;     // AHB 3 peripheral clocks enable register
} RCC_TypeDef;

#define RCC ((RCC_TypeDef *) 0x40021000))
```

Lighting up an LED using GPIO

Field	Full Name	Purpose
CR	**Clock Control Register**	Enables/disables oscillators (HSI, HSE, PLL), and system reset flags.
ICSCR	**Internal Clock Sources Calibration Register**	Calibrates internal oscillators (like HSI or MSI). Improves accuracy.
CFGR	**Clock Configuration Register**	Selects system clock source and sets prescalers for AHB, APB buses.
AHB1ENR	**AHB1 Clock Enable Register**	Enables/disables clocks to peripherals on the AHB1 bus (e.g., GPIOA, DMA1).
AHB2ENR	**AHB2 Clock Enable Register**	Enables clocks for AHB2 peripherals (e.g., GPIOB, USB, RNG).
AHB3ENR	**AHB3 Clock Enable Register**	Enables clocks for peripherals on AHB3 (e.g., external memory controller).

Lighting up an LED using GPIO

- The following C statements enable the clock of GPIO port B.

```
#define RCC_AHB2ENR_GPIOBEN (0x00000002)
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

- The flowchart of initializing a GPIO pin as digital output with push pull can be designed as:

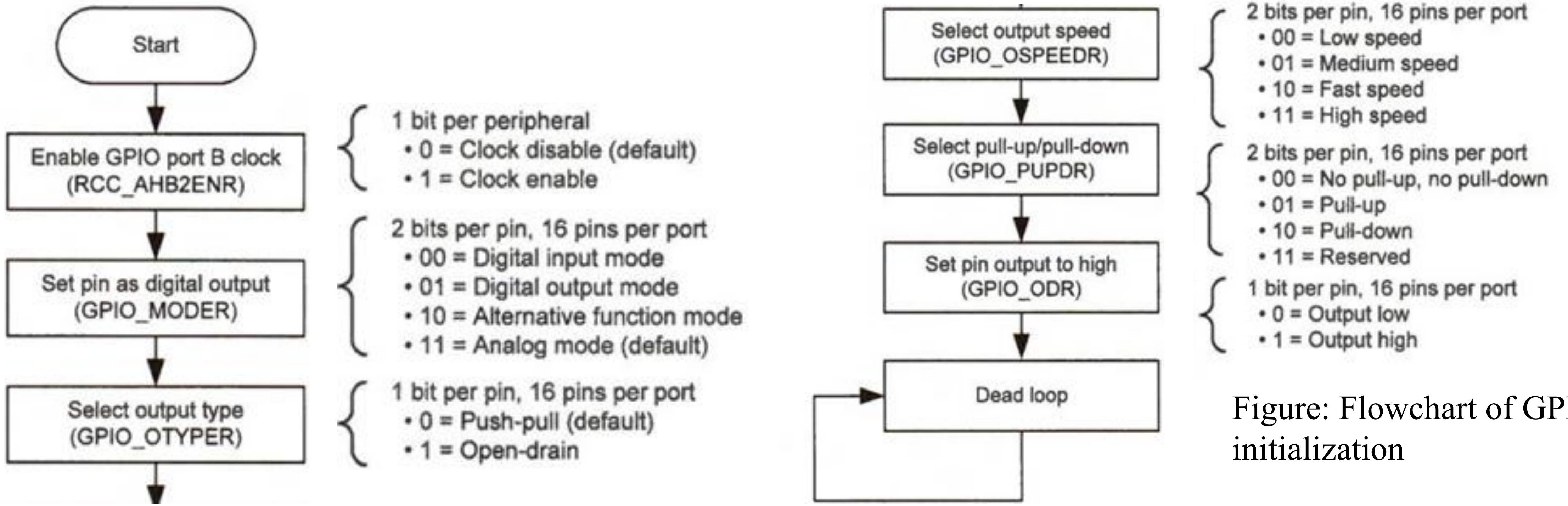

Figure: Flowchart of GPIO initialization

Lighting up an LED using GPIO

- When we change the value of specific bits in a register, we need to preserve the value of the other bits in this register to avoid creating unexpected negative impacts. For example, if we want to set the least significant bit in register R, "**R = 0x1**;" is incorrect because it also clears all the other bits. Instead, we should use a bitwise logical OR operation "**R |= 0x1**;"
- When we change the value of multiple bits, it is a good practice to reset these bits before updating them. For example, if we want to set the least significant four bits *b3b2b1b0* in register R to 1001, we need to clear these four bits first by running "**R &= ~0xF; R |= 0x9;**"
- If we do not clear these four bits first, we may fail to set the register correctly if their initial values are not **0**. For example, if the value of *b3b2b1b0* is 0111 initially, "**R |= 0x9**;" will lead a binary result of 1111.

Lighting up an LED using GPIO

- Each GPIO port has a *data output register* (ODR) and a *data input register* (IDR).
- Each bit in ODR controls the output of a corresponding GPIO pin in this port. In a push-pull setting, if the bit value is 1, the output voltage on its corresponding GPIO pin is high; if the bit value is 0, the output voltage then is low.
- The IDR register records the input of all pins of a GPIO port.

Lighting up an LED using GPIO

- The following C program demonstrates how to set up a GPIO pin and light up an LED in detail. Suppose we use the GPIO pin PB 2 to drive a red LED.

```c
// Red LED is connected PB 2 (GPIO port B pin 2)
void GPIO_Clock_Enable(){
    // Enable the clock to GPIO port B
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
}
```

```c
void GPIO_Pin_Init(){

    // Set mode of pin 2 as digital output
    // 00 = digital input,          01 = digital output
    // 10 = alternate function,     11 = analog (default)
    GPIOB->MODER &= ~(3UL<<4);      // Clear mode bits
    GPIOB->MODER |= 1UL<<4;         // mode = 01, digital output
```

Lighting up an LED using GPIO

- The following C program demonstrates how to set up a GPIO pin and light up an LED in detail. Suppose we use the GPIO pin PB 2 to drive a red LED.

```c
    // Set output type of pin 2 as push-pull
    // 0 = push-pull (default)
    // 1 = open-drain
    GPIOB->OTYPER &= ~(1<<2);

    // Set output speed of pin 2 as Low
    // 00 = Low speed,       01 = Medium speed
    // 10 = Fast speed,      11 = High speed
    GPIOB->OSPEEDR &= ~(3UL<<4);        // Clear speed bits

    // Set pin 2 as no pull-up, no pull-down
    // 00 = no pull-up, no pull-down  01 = pull-up
    // 10 = pull-down,                11 = reserved
    GPIOB->PUPDR &= ~(3UL<<4);          // no pull-up, no pull-down
}
```

Lighting up an LED using GPIO

- The following C program demonstrates how to set up a GPIO pin and light up an LED in detail. Suppose we use the GPIO pin PB 2 to drive a red LED.

```
int main(void){
    GPIO_Clock_Enable();
    GPIO_Pin_Init();
    GPIOB->ODR |= 1UL<<2;     // Set bit 2 of output data register (ODR)
    while(1);                 // Dead Loop & program hangs here
}
```

Lighting up an LED using GPIO

- The implementation in assembly is like the above C program. In the program,
  - *GPIOB_BASE* and *RCC_BASE* are pre-defined memory addresses
  - *GPIO_MODER, GPIO_OTYPER, GPIO_OSPEEDR, GPIO_PUPDR*, and *GPIO_ODR* are byte offset of its corresponding variable in the data structure *GPIO_ TypeDef* defined previously.

```
; Constants defined in file stm32l476xx_constants.s
;
; Memory addresses of GPIO port B and RCC (reset and clock control) data
; structure. These addresses are predefined by the chip manufacturer.
GPIOB_BASE        EQU    0x48000400
RCC_BASE          EQU    0x40021000

; Byte offset of each variable in the GPIO_TypeDef structure
GPIO_MODER        EQU    0x00
GPIO_OTYPER       EQU    0x04
GPIO_RESERVED0    EQU    0x06
GPIO_OSPEEDR      EQU    0x08
GPIO_PUPDR        EQU    0x0C
GPIO_IDR          EQU    0x10
GPIO_RESERVED1    EQU    0x12
GPIO_ODR          EQU    0x14
```

```
GPIO_RESERVED2    EQU    0x16
GPIO_BSRRL        EQU    0x18
GPIO_BSRRH        EQU    0x1A
GPIO_LCKR         EQU    0x1C
GPIO_AFR0         EQU    0x20  ;  AFR[0]
GPIO_AFR1         EQU    0x24  ;  AFR[1]
GPIO_AFRL         EQU    0x20
GPIO_AFRH         EQU    0x24

; Byte offset of variable AHB2ENR in the RCC_TypeDef structure
RCC_AHB2ENR       EQU    0x4C
```

Lighting up an LED using GPIO

- The implementation in assembly is like the above C program.

```
    INCLUDE stm32l476xx_constants.s

    AREA    main, CODE, READONLY
    EXPORT      __main               ; make __main visible to linker
    ENTRY

__main PROC
    ; Enable the clock to GPIO port B
    ; Load address of reset and clock control (RCC)
    LDR r2, =RCC_BASE                ; Pseduo instruction
    LDR r1, [r2, #RCC_AHB2ENR]       ; r1 = RCC->AHB2ENR
    ORR r1, r1, #2                   ; Set bit 2 of AHB2ENR
    STR r1, [r2, #RCC_AHB2ENR]       ; GPIO port B clock enable

    ; Load GPIO port B base address
    LDR r3, =GPIOB_BASE              ; Pseudo instruction
```

Lighting up an LED using GPIO

- The implementation in assembly is like the above C program.

```
; Set pin 2 I/O mode as general-purpose output
LDR r1, [r3, #GPIO_MODER]     ; Read the mode register
BIC r1, r1, #(3 << 4)         ; Direction mask pin 6, clear bits 5 and 4
ORR r1, r1, #(1 << 4)         ; Set mode as digital output (mode = 01)
STR r1, [r3, #GPIO_MODER]     ; Save to the mode register

; Set pin 2 the push-pull mode for the output type
LDR r1, [r3, #GPIO_OTYPER]    ; Read the output type register
BIC r1, r1, #(1<<2)           ; Push-pull(0), open-drain (1)
STR r1, [r3, #GPIO_OTYPER]    ; Save to the output type register

; Set I/O output speed value as low
LDR r1, [r3, #GPIO_OSPEEDR]   ; Read the output speed register
BIC r1, r1, #(3<<4)           ; Low(00), Medium(01), Fast(01), High(11)
STR r1, [r3, #GPIO_OSPEEDR]   ; Save to the output speed register
```

Lighting up an LED using GPIO

- The implementation in assembly is like the above C program.

```
  ; Set I/O output speed value as low
  LDR r1, [r3, #GPIO_OSPEEDR]   ; Read the output speed register
  BIC r1, r1, #(3<<4)           ; Low(00), Medium(01), Fast(01), High(11)
  STR r1, [r3, #GPIO_OSPEEDR]   ; Save to the output speed register

  ; Set I/O as no pull-up, no pull-down
  LDR r1, [r3, #GPIO_PUPDR]     ; r1 = GPIOB->PUPDR
  BIC r1, r1, #(3<<4)           ; No PUPD(00), PU(01), PD(10), Reserved(11)
  STR r1, [r3, #GPIO_PUPDR]     ; Save pull-up and pull-down setting

  ; Light up LED
  LDR r1, [r3, #GPIO_ODR]       ; Read the output data register
  ORR r1, r1, #(1<<2)           ; Set bit 2
  STR r1, [r3, #GPIO_ODR]       ; Save to the output data register

stop
  B   stop  ; dead loop & program hangs here
  ENDP
  END
```

Self Study

- Article 14.8 (Pushbutton) and 14.9 (Keypad Scan)
- **Ref. Book**: Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C 3rd Ed - Yifeng Zhu - Eman (2018) **[Chapter 14]**

Thank you