Lab 01

Beverly Yee

u0770041

Notes

When connecting the board to a computer, check to see if the LD1 light is steady, no blinking of any kind. If it is blinking, then it cannot fully connect to the computer and may not be discoverable. This may be due to the system missing the necessary drivers (as is my case). Drivers can be downloaded at: <https://www.st.com/en/development-tools/stsw-link009.html>

**Example — Setting a Pin to Output Mode**

The following steps configure pin PB3 as an output. Figure 1.5 outlines these using the GPIOx\_MODER register map in the peripheral reference manual.

1. The name PB3 indicates that the pin belongs to the GPIOB peripheral and is the fourth pin in the control register (pin numbering begins at zero).
2. The fourth cell in the GPIOx\_MODER register map is “MODER3” and contains bits 6-7.
3. Examine the bit patterns listed below the register map; one of these {01} corresponds to “general purpose output mode.”
4. The output mode bit pattern indicates that for the two configuration bits; set the lower bit to ‘1’ and clear the upper bit to ‘0’.
5. Since bits 6-7 in the GPIOx\_MODER register control pin PB3, setting bit 6 and clearing bit 7 configures the pin to output mode.

**1.1 — Configuring GPIO Pin Modes**

As demonstrated in the previous example, find and list the bit numbers to set or clear in the GPIOx\_MODER register (and indicate if you would set or clear those bits) to put the following pins into the requested modes:

* PC6 (RED LED) – General Purpose Output
  + GPIOC\_MODER6[1:0]
  + Set bit 12, clear 13
* PC7 (BLUE LED) – General Purpose Output
  + GPIOC\_MODER7[1:0]
  + Set 14, clear 15
* PC8 (ORANGE LED) – General Purpose Output
  + GPIOC\_MODER8[1:0]
  + Set 16, clear 17
* PC9 (GREEN LED) – General Purpose Output
  + GPIOC\_MODER9[1:0]
  + Set 18, clear 19
* PA0 (USER Button) – Digital Input
  + GPIOA\_MODER0[1:0]
  + Clear both bits (?)

**1.2 — Configuring GPIO Pin Details**

Use the GPIO register documentation in the peripheral reference manual to determine the bits to modify in order to configure the listed parameters for the following pins:

* PC6, PC7, PC8, PC9 (LED PINS)
  + GPIOx\_OTYPER – Push-Pull Output Type
    - GPIOC\_OTYPER OT6-9
    - Bits 6-9 cleared
  + GPIOx\_OSPEEDR – Low Speed
    - GPIOC\_OSPEEDR6-9
    - Bits 12-19 cleared
  + GPIOx\_PUPDR – No Pull-Up or Pull-Down
    - GPIOC\_PUPDR6-9
    - Bits 12-19 cleared
* PA0 (USER Button)
  + GPIOx\_OSPEEDR – Low Speed
    - GPIOA\_OSPEEDR0
    - Bits 0-1 cleared
  + GPIOx\_PUPDR – Pull-Down Resistor
    - GPIOA\_PUPDR0
    - Bit 1 set, bit 0 cleared

**1.3 — Initializing the GPIO Pins**

We recommend that you begin with the provided “blinky” example code and make incremental changes from the HAL library calls to bitwise operations on peripheral registers.

Using the bit numbers and registers that you found in the earlier exercises, configure the GPIO pins to the mode and parameters.

* Remove the GPIO\_InitTypeDef struct and all HAL\_GPIO\_Init() function calls.
* Make sure to reference the correct GPIO peripheral indicated by the pin name.
* Configure LED pins (PC6-PC9) in the following way:
  + General-purpose output mode using the MODER register.
  + Push-pull output type using the OTYPER register.
  + Low speed using the OSPEEDR register.
  + No pull-up/down resistors using the PUPDR register.
* USER Button pin (PA0) should be configured to:
  + Digital input mode using the MODER register.
  + Low speed using the OSPEEDR register.
  + Pull-down resistor using the PUPDR register.

Blinky example

int main(void) {

HAL\_Init(); // Reset of all peripherals, init the Flash and Systick

SystemClock\_Config(); //Configure the system clock

/\* This example uses HAL library calls to control the GPIOC peripheral.

You’ll be redoing this code with hardware register access. \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE(); // Enable the GPIOC clock in the RCC

// Set up a configuration struct to pass to the initialization function

GPIO\_InitTypeDef initStr = {GPIO\_PIN\_8 | GPIO\_PIN\_9,

GPIO\_MODE\_OUTPUT\_PP,

GPIO\_SPEED\_FREQ\_LOW,

GPIO\_NOPULL};

HAL\_GPIO\_Init(GPIOC, &initStr); // Initialize pins PC8 & PC9

HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_8, GPIO\_PIN\_SET); // Start PC8 high

while (1) {

HAL\_Delay(200); // Delay 200ms

// Toggle the output state of both PC8 and PC9

HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_8 | GPIO\_PIN\_9);

}

}

Blinky without HAL

int main(void) {

HAL\_Init(); // Reset of all peripherals, init the Flash and Systick

SystemClock\_Config(); //Configure the system clock

// enable the GPIOC peripheral clock

RCC->AHBENR |= RCC\_AHBENR\_GPIOCEN;

// set the general purpose output for green and orange LEDs, PC8 and PC9 respectively

// bits = 01

GPIOC->MODER |= GPIO\_MODER\_MODER8\_0;

GPIOC->MODER |= GPIO\_MODER\_MODER9\_0;

// both LEDs have push-pull output type = both bits cleared

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_8);

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_9);

// low speed = both bits cleared

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR8);

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR9);

// no pull-up/down resistors = both bits cleared

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR8);

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR9);

// write 1 pin high, write other pin low using either ODR or BSRR register

// start with 8 high and 9 down

GPIOC->ODR |= GPIO\_ODR\_8;

GPIOC->ODR &= ~(GPIO\_ODR\_9);

while (1) {

// Toggle the output state of both PC6 and PC7

// without using HAL toggle function

HAL\_Delay(300); // Delay 200ms

GPIOC->ODR &= ~(GPIO\_ODR\_8);

GPIOC->ODR |= GPIO\_ODR\_9;

HAL\_Delay(300); // Delay 200ms

GPIOC->ODR |= GPIO\_ODR\_8;

GPIOC->ODR &= ~(GPIO\_ODR\_9);

}

}

Reading a Button

User button is PA0

int main(void) {

HAL\_Init(); // Reset of all peripherals, init the Flash and Systick

SystemClock\_Config(); //Configure the system clock

SetPins();

// write 1 pin high, write other pin low using either ODR or BSRR register

GPIOC->ODR |= GPIO\_ODR\_6;

GPIOC->ODR &= ~(GPIO\_ODR\_7);

GPIOC->ODR &= ~(GPIO\_ODR\_8);

GPIOC->ODR &= ~(GPIO\_ODR\_9);

uint32\_t debounce = 0;

while (1) {

// monitor the button pin input state using IDR register

// the button pin is on 0

// make sure to do the debouncing as well

debounce = (debounce << 1); // shift every loop iteration

// if the input signal is set/high

if(GPIOA->IDR & 0x1) {

// set the lowest bit of bit-vector

debounce |= 0x01;

}

if(debounce == 0x7FFFFFFF) {

// if the button is pressed, switch the LED

if(GPIOC->ODR & GPIO\_ODR\_6){

GPIOC->ODR |= GPIO\_ODR\_7;

GPIOC->ODR &= ~(GPIO\_ODR\_6);

GPIOC->ODR &= ~(GPIO\_ODR\_8);

GPIOC->ODR &= ~(GPIO\_ODR\_9);

}

else if(GPIOC->ODR & GPIO\_ODR\_7){

GPIOC->ODR |= GPIO\_ODR\_8;

GPIOC->ODR &= ~(GPIO\_ODR\_7);

GPIOC->ODR &= ~(GPIO\_ODR\_6);

GPIOC->ODR &= ~(GPIO\_ODR\_9);

}

else if(GPIOC->ODR & GPIO\_ODR\_8){

GPIOC->ODR |= GPIO\_ODR\_9;

GPIOC->ODR &= ~(GPIO\_ODR\_8);

GPIOC->ODR &= ~(GPIO\_ODR\_6);

GPIOC->ODR &= ~(GPIO\_ODR\_7);

}

else if(GPIOC->ODR & GPIO\_ODR\_9){

GPIOC->ODR |= GPIO\_ODR\_6;

GPIOC->ODR &= ~(GPIO\_ODR\_9);

GPIOC->ODR &= ~(GPIO\_ODR\_7);

GPIOC->ODR &= ~(GPIO\_ODR\_8);

}

}

}

}

**Red/Blue Assignment**

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2024 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void) {

HAL\_Init(); // Reset of all peripherals, init the Flash and Systick

SystemClock\_Config(); //Configure the system clock

// enable the GPIOC clock in the RCC

RCC->AHBENR |= RCC\_AHBENR\_GPIOCEN;

// set up red led (PC6) to low-speed, push-pull output, no push/pull resistor

GPIOC->MODER |= GPIO\_MODER\_MODER6\_0;

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR6);

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_6);

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR6);

// set up blue led (PC7) to low-speed, push-pull output, no push/pull resistor

GPIOC->MODER |= GPIO\_MODER\_MODER7\_0;

GPIOC->OTYPER &= ~(GPIO\_OTYPER\_OT\_7);

GPIOC->OSPEEDR &= ~(GPIO\_OSPEEDER\_OSPEEDR7);

GPIOC->PUPDR &= ~(GPIO\_PUPDR\_PUPDR7);

// write 1 pin high, write other pin low using either ODR or BSRR register

// start with 6 high and 7 down

GPIOC->ODR |= GPIO\_ODR\_6;

GPIOC->ODR &= ~(GPIO\_ODR\_7);

while (1) {

// Toggle the output state of both PC6 and PC7

// without using HAL toggle function

// use XOR to toggle a pin from high to low and vice versa

HAL\_Delay(300);

GPIOC->ODR ^= GPIO\_ODR\_6;

GPIOC->ODR ^= GPIO\_ODR\_7;

}

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

Lab 01

Beverly Yee

u0770041

Postlab Questions

1. What are the GPIO control registers that the lab mentions? Briefly describe each of their functions.

**MODER – sets the type of mode the programmer wants it to be. There are 4 modes: General Purpose Output (push-pull or open-drain logic), Digital Input (floating or internal pull-up/down), Analog Input (connect to analog peripherals instead of digital), Alternate Function (connects to selection of internal peripherals).**

**OSPEEDR – controls the speed of the peripherals, which also determines how much power the peripherals use. Lower speeds use less power. Higher speeds use higher power.**

**OTYPER – sets the output mode of the pin.**

**PUPDR – connects a pin to internal pull-up/down resistors.**

**IDR – a read-only register. Reports the logical state of each pin in the GPIO port.**

**ODR – output register. Sets the logical state of configured output pins.**

**BSRR – write-only register. Sets and clear bits quickly in the output register. Lower half of the register sets bits in the output. Upper half clears/resets them.**

1. What values would you want to write to the bits controlling a pin in the GPIOx\_MODER register in order to set it to analog mode?

**0x3**

1. Examine the bit descriptions in GPIOx\_BSRR register: which bit would you want to set to clear the fourth bit in the ODR?

**Bit 19**

1. Perform the following bitwise operations:
   * 0xAD | 0xC7 = ?

**0xBF**

* + 0xAD & 0xC7 = ?

**0x85**

* + 0xAD & ~(0xC7) = ?

**0x28**

* + 0xAD ^0xC7 = ?

**0x6A**

1. How would you clear the 5th and 6th bits in a register while leaving the others alone?

**Bitmask the 5th and 6th registers only.**

**Reg &= ~((1 << 5) | (1 << 6))**