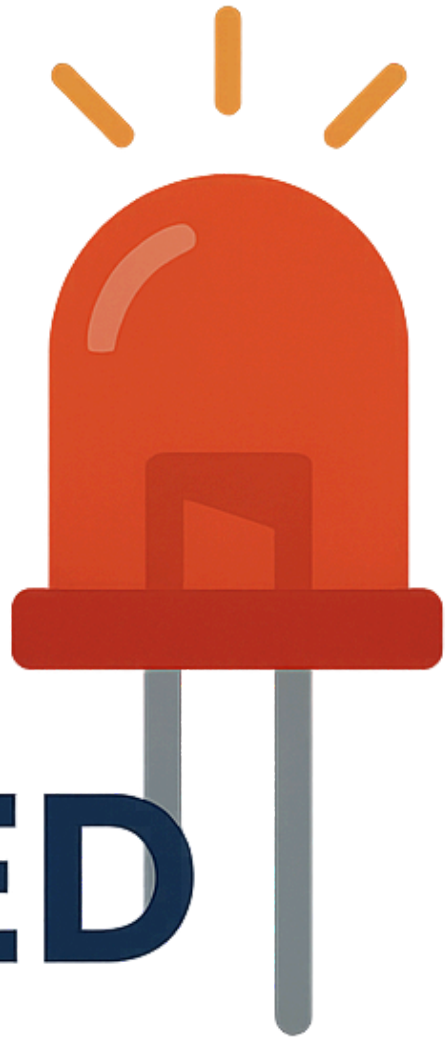
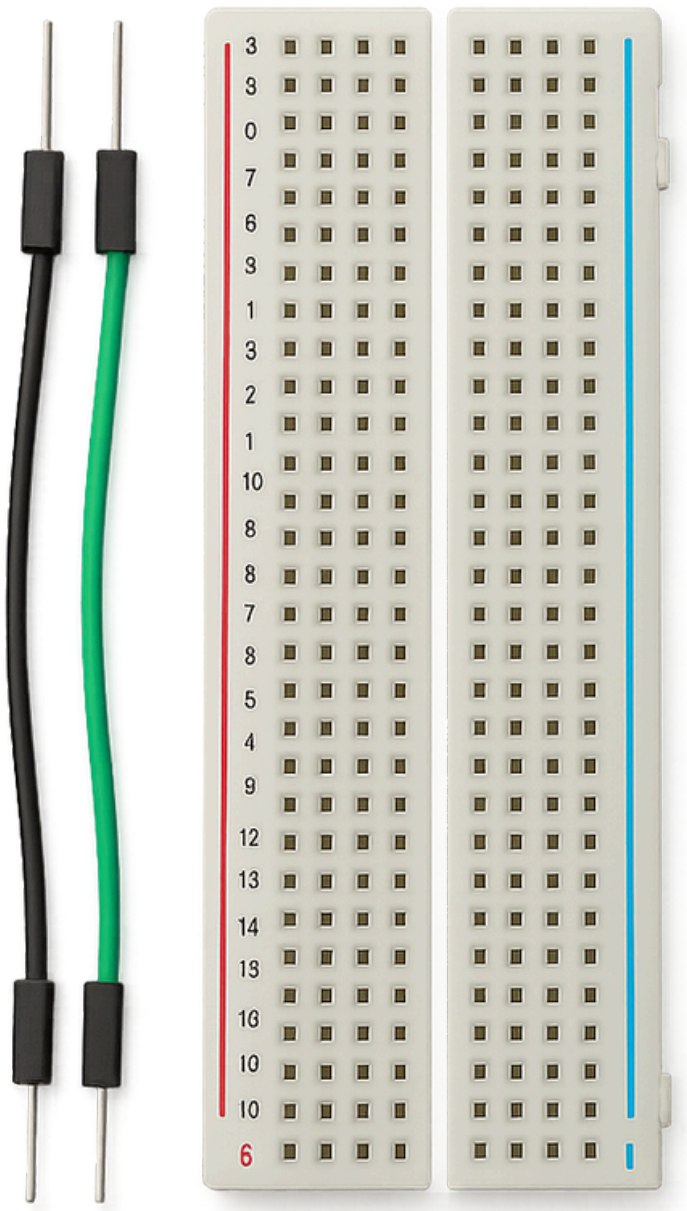
TA Embedded

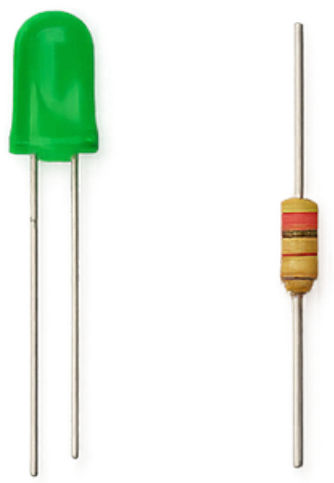
#AWeekAPeripheral

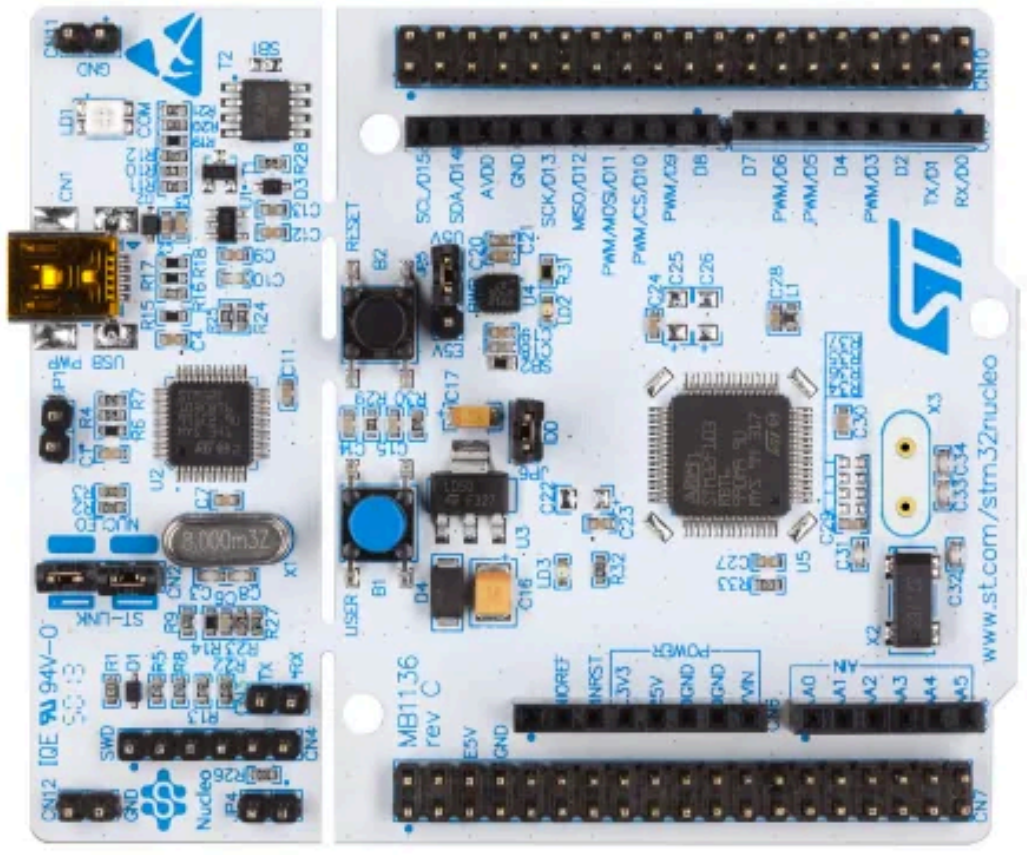
HELLO, WORLD OF EMBEDDED SYSTEMS!



REQUIREMENTS



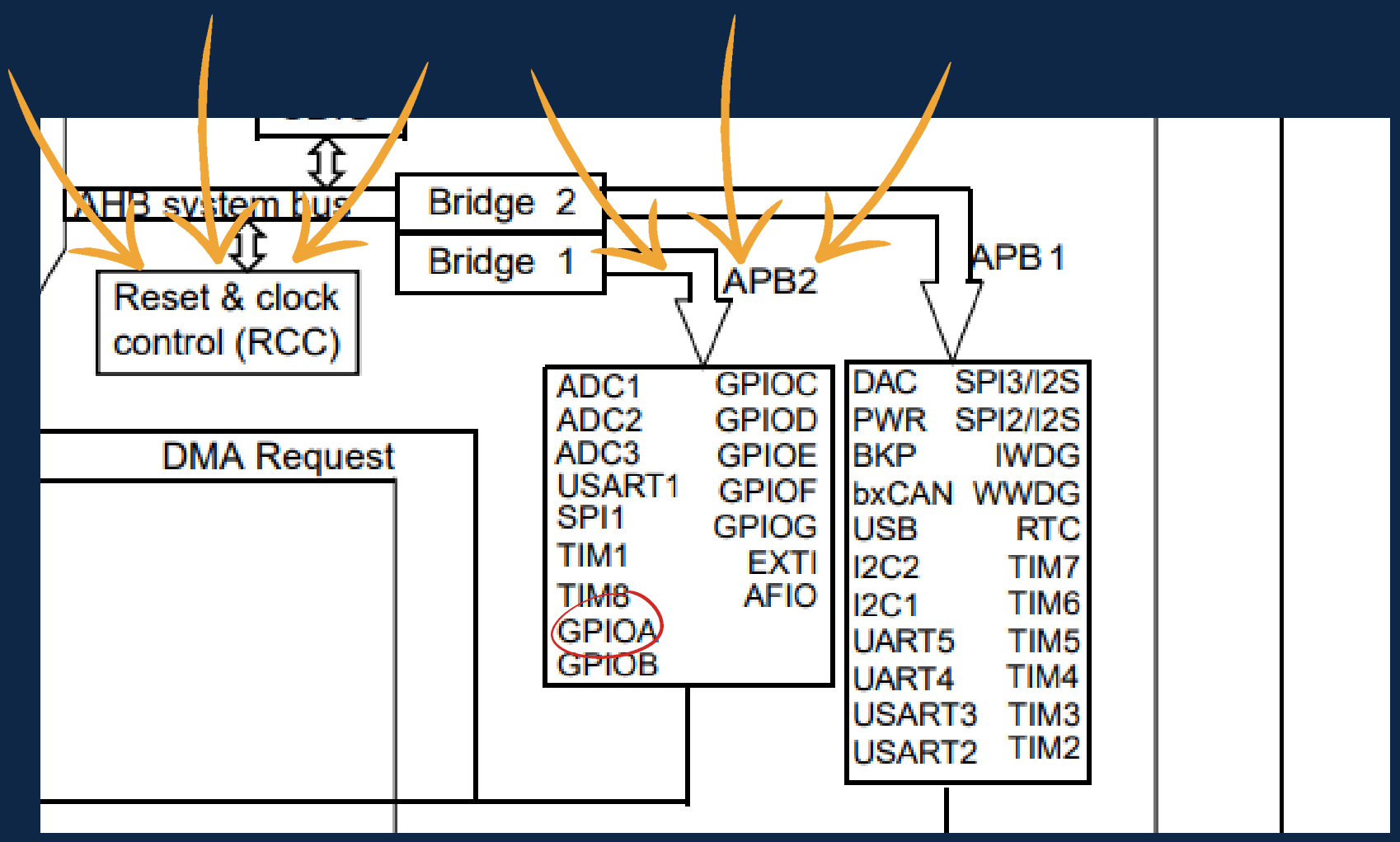




STM32 NUCLEO- F103RB

Step 1 – Enable Clock for GPIOA

BEFORE A MICROCONTROLLER PIN CAN DO ANYTHING, IT NEEDS POWER.
SO FIRST, WE ENABLE THE CLOCK FOR THE GPIO PORT.



GPIOA IS CONNECTED TO THE RCC THROUGH THE APB2 BUS.

TO ACTIVATE IT, YOU MUST SET BIT 2 IN THE RCC_APB2ENR REGISTER

TA Embedded

DETAILS:

- PERIPHERAL: RCC (RESET AND CLOCK CONTROL)
- BUS: APB2
- REGISTER: RCC_APB2ENR
- BASE ADDRESS: 0X40021000
- OFFSET: 0X18
- FINAL ADDRESS: 0X40021018

	0x4002 1000 - 0x4002 13FF	Reset and clock control RCC

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Bit 2 **IOPAEN**: I/O port A clock enable

Set and cleared by software.

0: I/O port A clock disabled

1:I/O port A clock enabled

Macro definition :

```
// ---------------- RCC REGISTER DEFINITIONS ----------------

// RCC (Reset and Clock Control) is responsible for managing clocks
// for all peripherals in the STM32 microcontroller.

// Define the base address of the RCC peripheral block.
// This is a fixed address from the STM32 memory map.
#define RCC_BASE_ADDR           0x40021000UL  // RCC base address in memory

// Define the offset to the APB2ENR register within the RCC register map.
// APB2ENR = APB2 Peripheral Clock Enable Register
// This register is used to enable/disable clocks to peripherals connected via the APB2 bus,
// such as GPIOA, GPIOB, TIM1, and AFIO.
#define RCC_APB2ENR_OFFSET      0x18UL        // Offset to APB2ENR from RCC base

// Define the full memory-mapped address of the RCC_APB2ENR register.
// This macro combines the base address and the offset to locate the exact address in memory
// where the APB2ENR register resides.
#define RCC_APB2ENR             (RCC_BASE_ADDR + RCC_APB2ENR_OFFSET)  // 0x40021018
```

IN MAIN :

```
 // Create a pointer to the RCC_APB2ENR register,
// which is responsible for enabling clocks to peripherals on the APB2 bus.
// 'volatile' is used because the value of this register can change due to hardware behavior
// or external events, so the compiler should not optimize access to it.
volatile uint32_t* ptrRccApb2Enr = (volatile uint32_t*) RCC_APB2ENR;

// Set bit 2 of the APB2ENR register to 1 to enable the clock for GPIOA.
// Each bit in this register corresponds to a different peripheral on the APB2 bus:
// Bit 2 = GPIOA
// Bit 3 = GPIOB
// Bit 4 = GPIOC, etc.
// This step is **mandatory** before accessing any GPIOA registers.
*ptrRccApb2Enr |= (1 << 2);  // Enable clock for GPIOA
```

Step 2 – Configure PA8 as Output

OBJECTIVE:

TELL THE PIN TO BEHAVE AS AN OUTPUT (NOT INPUT OR ANALOG).

DETAILS:

- PERIPHERAL: GPIOA
- REGISTER: GPIOA_CRH (BECAUSE PA8 IS PIN 8)

- BASE ADDRESS: 0X40010800

0x4001 0800 - 0x4001 0BFF	GPIO Port A

- OFFSET: 0X04
- FINAL ADDRESS: 0X40010804

- EACH PIN USES 4 BITS: MODE[1:0] AND CNF[1:0]
- FOR PA8: BITS 3:0
  - MODE = 10 → OUTPUT AT 2 MHZ
  - CNF = 00 → GENERAL PURPOSE PUSH-PULL

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 20: Port bit configuration table*.

In input mode (MODE[1:0]=00):

00: Analog mode

01: Floating input (reset state)

10: Input with pull-up / pull-down

11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull

01: General purpose output Open-drain

10: Alternate function output Push-pull

11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to *Table 20: Port bit configuration table*.

00: Input mode (reset state)

01: Output mode, max speed 10 MHz.

10: Output mode, max speed 2 MHz.

11: Output mode, max speed 50 MHz.

Macro definition :

```
#define GPIOA_BASE_ADRR                 0x40010800UL
#define GPIOA_CRH_OFFSET                0x04UL

#define GPIOA_CRH_ADRR          (GPIOA_BASE_ADRR + GPIOA_CRH_OFFSET)
```

IN MAIN :

```
// Create a volatile pointer to GPIOA's Output Data Register (ODR)
// ODR is used to set or clear the output state of GPIO pins (HIGH or LOW)
volatile uint32_t *ptrGpioaOdr = (volatile uint32_t *)GPIOA_ODR_ADRR;

// Create a pointer to GPIOA's Configuration Register High (CRH)
// CRH controls the configuration for GPIO pins 8 through 15
uint32_t *ptrGpioaCrh = (uint32_t *)GPIOA_CRH_ADRR;

// Clear the configuration bits for pin PA8 (bits 3:0 of CRH)
// This resets the MODE8[1:0] and CNF8[1:0] fields to 0
// 0xF in binary is 1111, so this clears all 4 bits related to PA8
*ptrGpioaCrh &= ~(0xF << 0);

// Set PA8 to Output mode, 2 MHz, Push-Pull configuration
// MODE8 = 10 (output at 2 MHz), CNF8 = 00 (general purpose push-pull)
// This enables PA8 to drive an LED or other logic-level devices
*ptrGpioaCrh |=  (0x2 << 0);
```

Step 3 – Connect Your LED to PA8

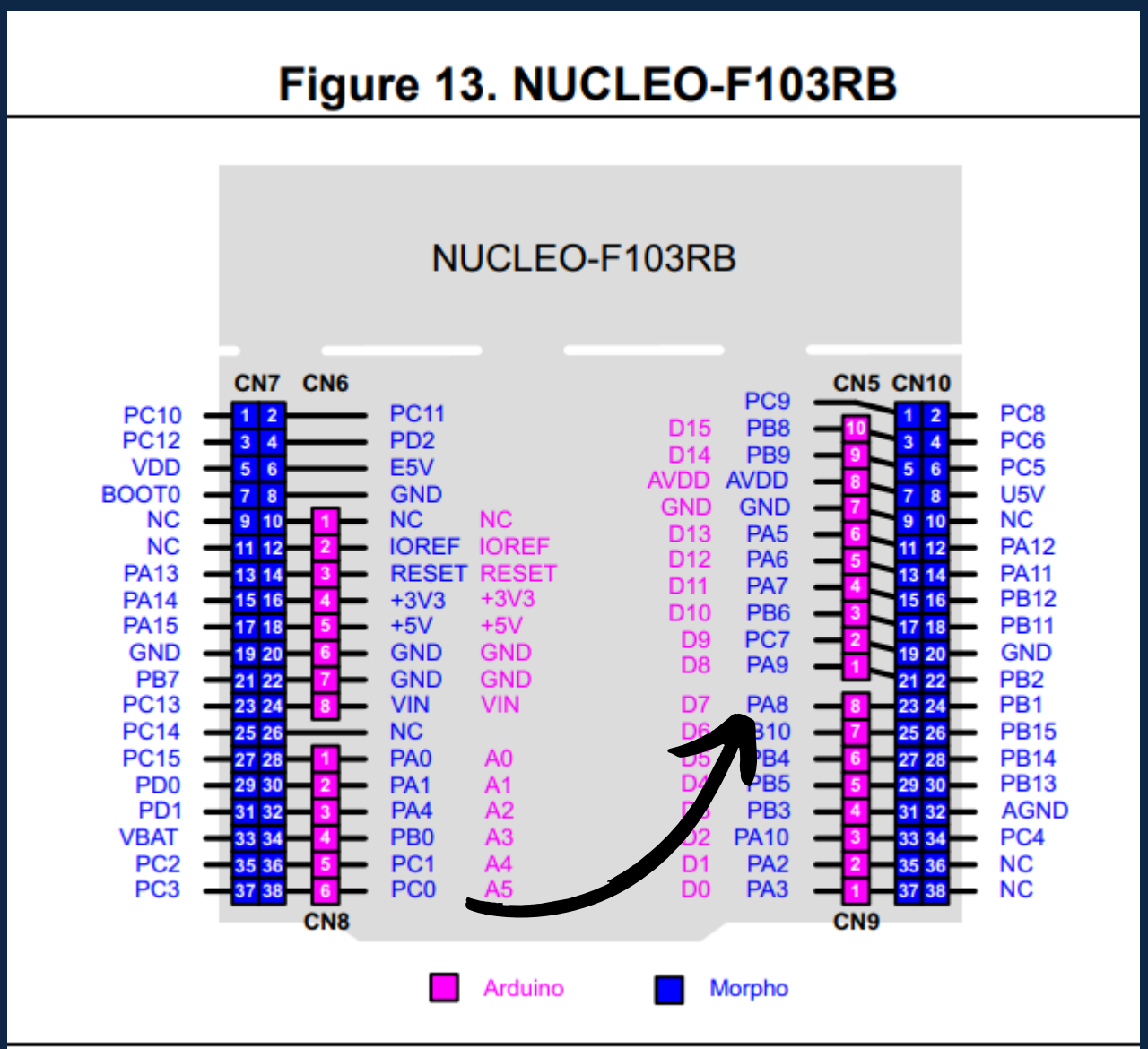
Objective:

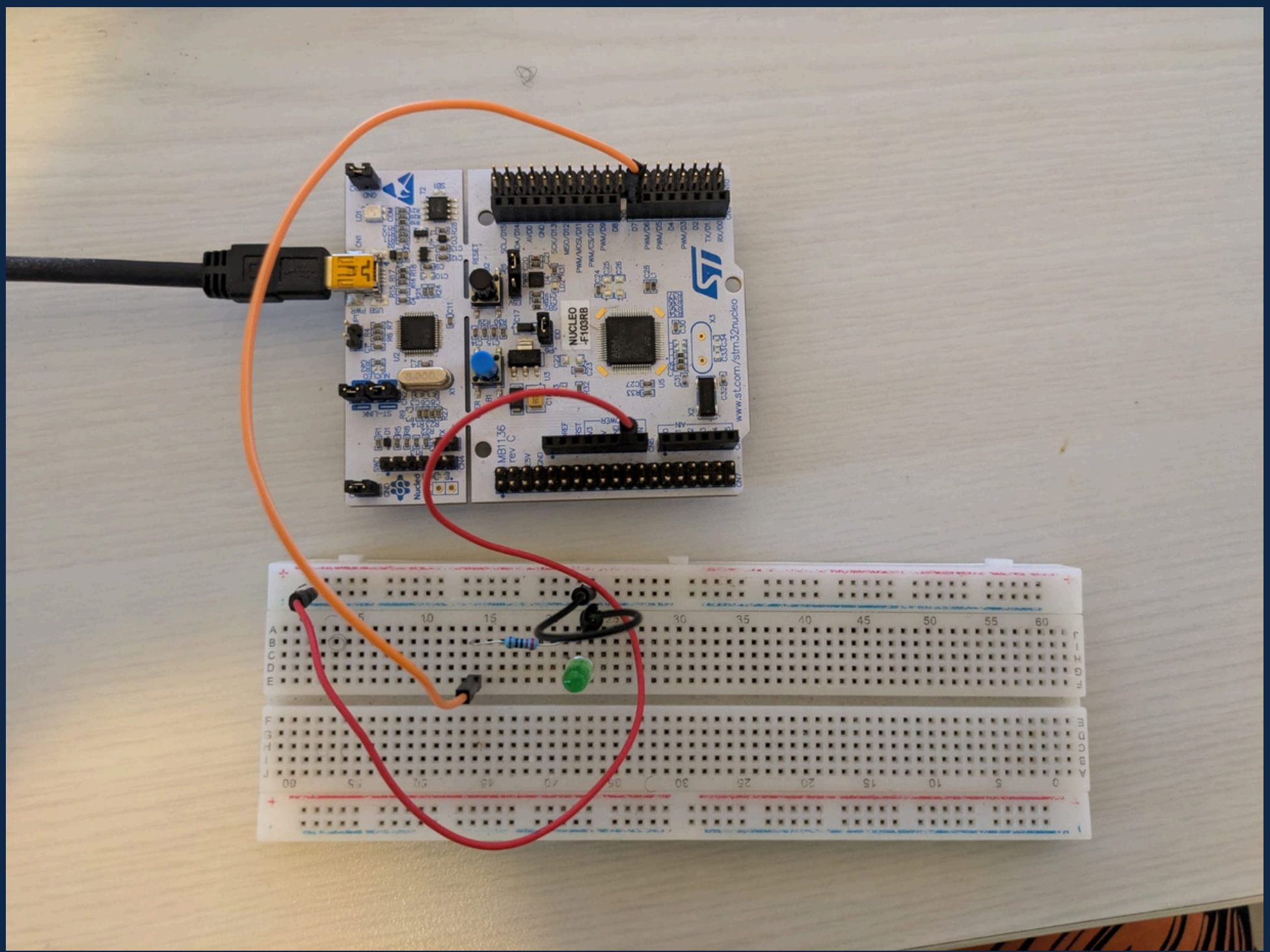
Wire your LED properly to PA8 to see it respond.

Connect:

- PA8 → Resistor (220Ω) → LED anode (long leg)
- LED cathode → GND

Figure 13. NUCLEO-F103RB




NUCLEO
-F103RB
MB1136
rev C
www.st.com/stm32nucleo

STEP 4 – TURN THE LED ON USING ODR

Objective:

Drive the pin HIGH to power the LED.

- Peripheral: GPIOA
- Register: GPIOA_ODR (Output Data Register)

- Base Address: 0x40010800
- Offset: 0x0C
- Final Address: 0x4001080C

- Each bit corresponds to a pin: Bit 8 = PA8

9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)
These bits can be read and written by software and can be accessed in Word mode only.

Macro Definition

```
#define GPIOA_ODR_OFFSET        0X0CUL


#define GPIOA_ODR_ADRR          (GPIOA_BASE_ADRR + GPIOA_ODR_OFFSET)
```

Pin A8 High

```
*ptrGpioaOdr |=  (1 << 8);  // Set PA8
```

Step 5 – Add a Delay

Objective:

Pause so the LED stays on long enough to see.

```
void delay(void) {
    for (volatile uint32_t i = 0; i < 100000; i++);
}
```

```
  *ptrGpioaOdr |=  (1 << 8);  // Set PA8
   delay();
```

Step 6 – Turn the LED OFF Using ODR

Objective:

Clear PA8 to 0 → turns off the LED.

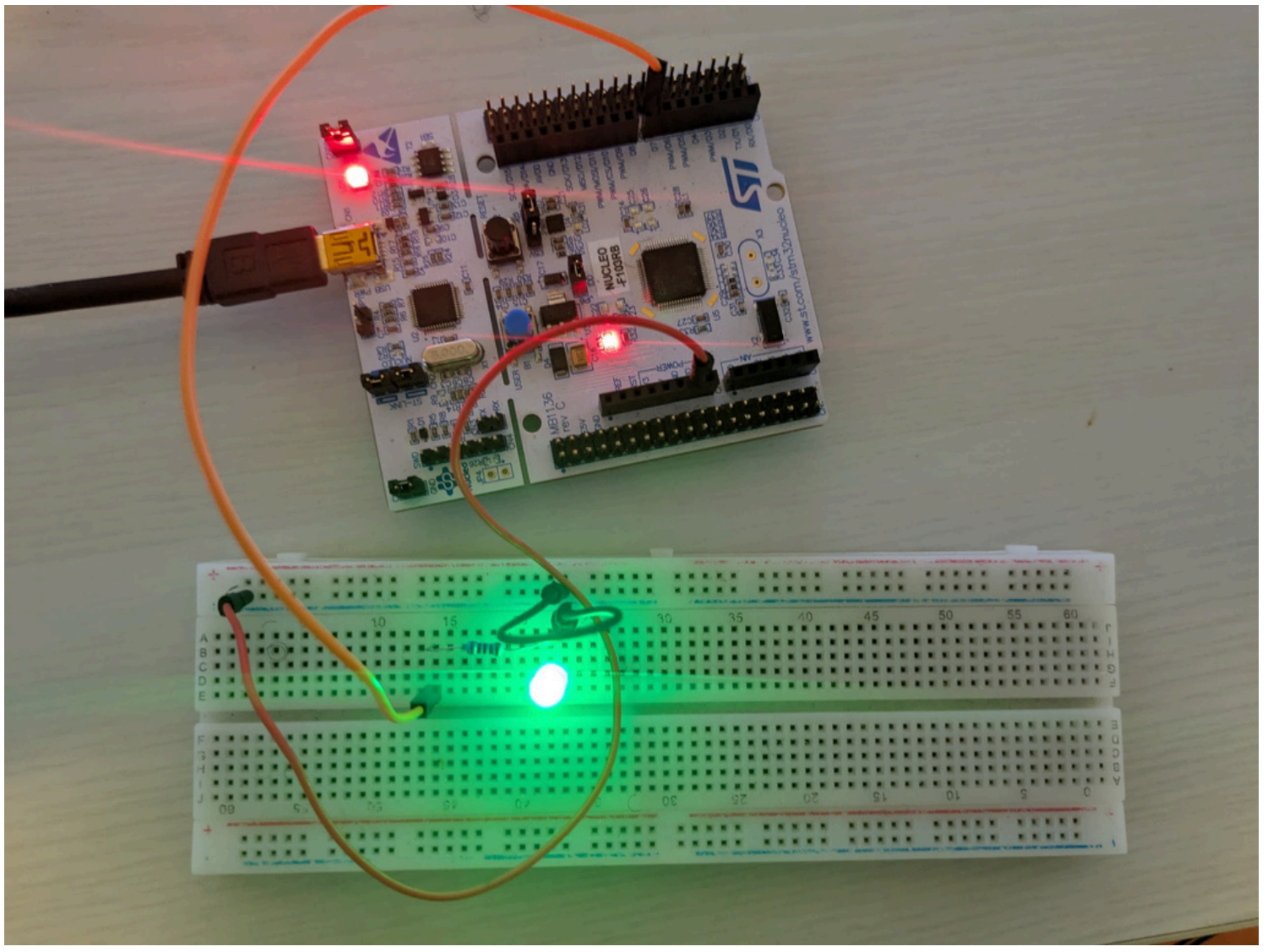
```
*ptrGpioaOdr &= ~(1 << 8);  // Clear PA8
```

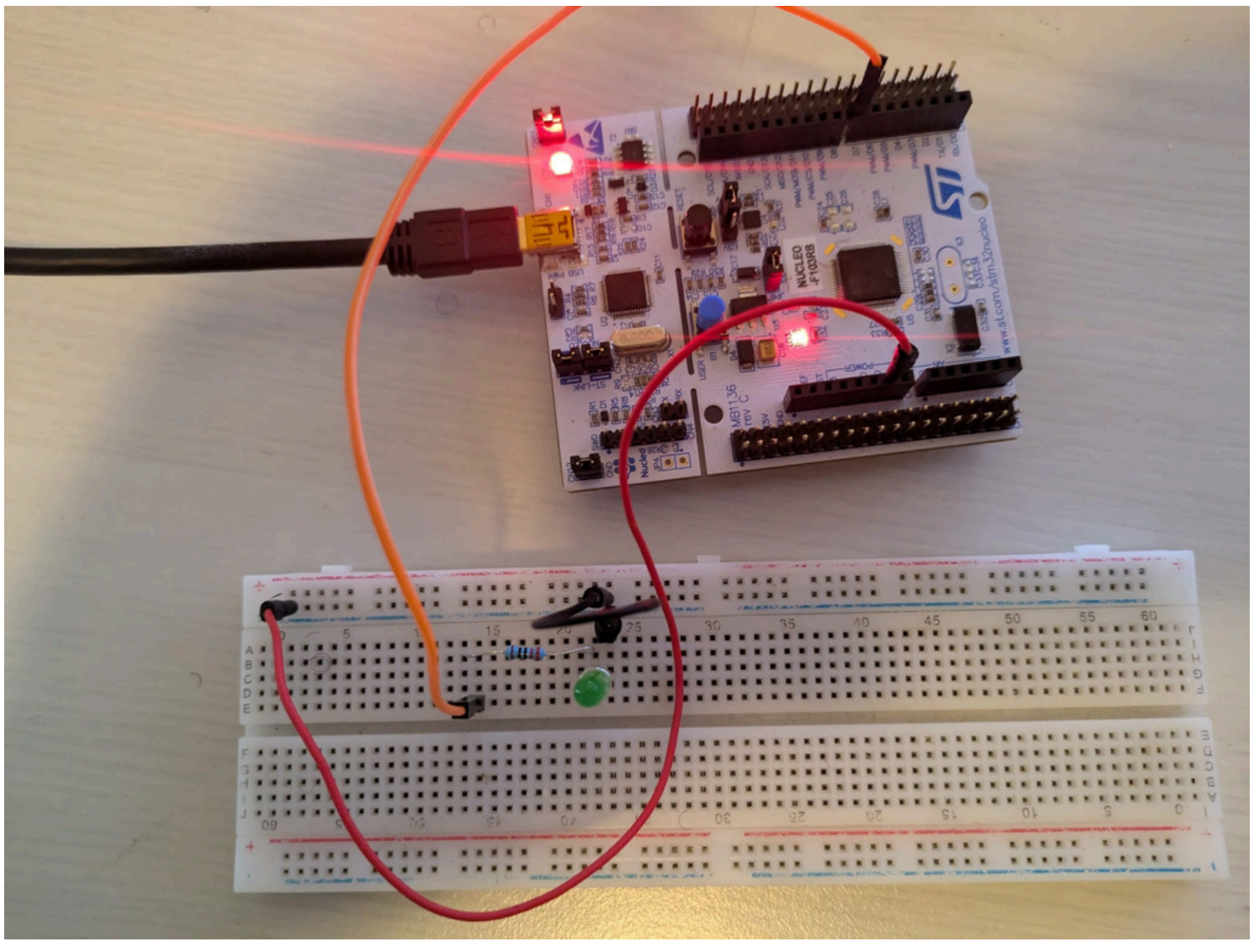
Step 7 – Repeat Forever (Blink Loop)

Objective:

Create an infinite loop to blink the LED ON/OFF with delay

```
while (1)
 {
   /* USER CODE END WHILE */
     *ptrGpioaOdr |=  (1 << 8);  // Set PA8
       delay();
     *ptrGpioaOdr &= ~(1 << 8);  // Clear PA8
       delay();
   /* USER CODE BEGIN 3 */
 }
```


NUCLEO
F103RB
MB1136
rev C
www.st.com/stm32nucleo


NUCLEO
F103RB
MB1136
rev C
www.st.com/stm32nucleo

Like and follow for more embedded systems related content