**Parte 1: ¿Cómo funciona un LED como sensor de luz?**

Un **LED puede actuar como un fotodiodo** cuando se conecta de forma inversa (polarización inversa). Al recibir luz, genera una pequeña corriente. Esto permite detectar luz **sin sensores adicionales**.

* Para leer esta corriente, se puede usar el pin **ADC** del Blue Pill.
* Cada LED será conectado en un pin ADC para leer su “nivel de luz”.

**🔌 Parte 2: Esquema eléctrico (por LED)**

Cada LED tendrá **dos funciones**:

1. **Sensor de luz** (cuando está apagado y conectado al ADC).
2. **Emisor de luz** (cuando se activa como salida digital).

Para lograr esto, cada pin del microcontrolador debe ser **configurable dinámicamente** como entrada analógica o salida digital. ¡El Blue Pill lo permite!

**🧠 Parte 3: Lógica del programa**

1. Cíclicamente se apaga cada LED.
2. Se configura su pin como entrada analógica.
3. Se hace una lectura ADC para ver si está recibiendo luz.
4. Si recibe luz (ADC > umbral), entonces se prenden los otros tres LEDs.
5. Se vuelve a configurar como salida y se apaga o prende según lógica.
6. Se repite este ciclo.

**📟 Parte 4: Pines sugeridos**

Supongamos los 4 LEDs están conectados a estos pines del Blue Pill:

| **LED #** | **Pin STM32** |
| --- | --- |
| LED1 | PA0 |
| LED2 | PA1 |
| LED3 | PA2 |
| LED4 | PA3 |

Todos estos pines tienen ADC.

**🧾 Parte 5: Código básico (con STM32 HAL)**

Version **simplificada** con HAL (que podés generar desde STM32CubeMX):

uint16\_t leer\_luz(uint8\_t canal\_adc) {

ADC\_ChannelConfTypeDef sConfig = {0};

sConfig.Channel = canal\_adc;

sConfig.Rank = ADC\_REGULAR\_RANK\_1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_71CYCLES\_5;

HAL\_ADC\_ConfigChannel(&hadc1, &sConfig);

HAL\_ADC\_Start(&hadc1);

HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY);

uint16\_t value = HAL\_ADC\_GetValue(&hadc1);

HAL\_ADC\_Stop(&hadc1);

return value;

}

void controlar\_leds() {

uint16\_t lectura[4];

uint8\_t canales[4] = {ADC\_CHANNEL\_0, ADC\_CHANNEL\_1, ADC\_CHANNEL\_2, ADC\_CHANNEL\_3};

GPIO\_TypeDef\* puertos[4] = {GPIOA, GPIOA, GPIOA, GPIOA};

uint16\_t pines[4] = {GPIO\_PIN\_0, GPIO\_PIN\_1, GPIO\_PIN\_2, GPIO\_PIN\_3};

for (int i = 0; i < 4; i++) {

// Configurar como entrada analógica (para "sensor")

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = pines[i];

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

HAL\_GPIO\_Init(puertos[i], &GPIO\_InitStruct);

lectura[i] = leer\_luz(canales[i]);

// Volver a configurar como salida

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(puertos[i], &GPIO\_InitStruct);

}

for (int i = 0; i < 4; i++) {

if (lectura[i] > 1000) { // Umbral de luz

for (int j = 0; j < 4; j++) {

HAL\_GPIO\_WritePin(puertos[j], pines[j], (j != i) ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET);

}

}

}

}

**🔄 Parte 6: Loop principal**

while (1) {

controlar\_leds();

HAL\_Delay(100); // Evita parpadeo excesivo

}

**🛠️ Paso 1: Configuración en STM32CubeMX**

**🔷 Microcontrolador**

Seleccioná el **STM32F103C8T6** (Blue Pill).

**🔷 Configuración de pines**

Activá los siguientes pines como entradas/salidas con función analógica:

| **Pin** | **Función en CubeMX** | **Descripción** |
| --- | --- | --- |
| PA0 | GPIO\_Analog (ADC\_IN0) | LED1/Sensor1 |
| PA1 | GPIO\_Analog (ADC\_IN1) | LED2/Sensor2 |
| PA2 | GPIO\_Analog (ADC\_IN2) | LED3/Sensor3 |
| PA3 | GPIO\_Analog (ADC\_IN3) | LED4/Sensor4 |

No selecciones “ADC1\_INx” directamente desde el ADC, solo dejalos como GPIO\_ANALOG para poder cambiar entre entrada analógica y salida digital manualmente.

**🔷 Configurar ADC1**

* **Scan Conversion Mode**: Disable
* **Continuous Conversion Mode**: Disable
* **Discontinuous Mode**: Disable
* **Data Alignment**: Right
* **Resolution**: 12-bit
* **Number of conversions**: 1

No agregues canales todavía. Eso lo hacemos por software dinámicamente.

**🔷 Configurar el reloj**

* Activá el HSE (si usás cristal externo).
* Asegurate de que los clocks estén bien configurados (puede ser HSI si no usás cristal).

**🔷 Activar USART2 (opcional para debug)**

**🔧 Paso 2: Generar el proyecto**

* Toolchain: **STM32CubeIDE**
* Project name: BluePill\_LightMatrix
* Finish y abrí el proyecto.

**📄 Paso 3: Código en main.c**

Agregá estas funciones al archivo main.c:

**✅ Inicialización de variables**

Justo después de las definiciones de GPIO:

ADC\_HandleTypeDef hadc1;

// Pines usados (GPIOA, PA0-PA3)

GPIO\_TypeDef\* puertos[4] = {GPIOA, GPIOA, GPIOA, GPIOA};

uint16\_t pines[4] = {GPIO\_PIN\_0, GPIO\_PIN\_1, GPIO\_PIN\_2, GPIO\_PIN\_3};

uint32\_t canales[4] = {ADC\_CHANNEL\_0, ADC\_CHANNEL\_1, ADC\_CHANNEL\_2, ADC\_CHANNEL\_3};

**✅ Función para leer luz con ADC**

uint16\_t leer\_luz(uint32\_t canal\_adc) {

ADC\_ChannelConfTypeDef sConfig = {0};

sConfig.Channel = canal\_adc;

sConfig.Rank = ADC\_REGULAR\_RANK\_1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_71CYCLES\_5;

HAL\_ADC\_ConfigChannel(&hadc1, &sConfig);

HAL\_ADC\_Start(&hadc1);

HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY);

uint16\_t value = HAL\_ADC\_GetValue(&hadc1);

HAL\_ADC\_Stop(&hadc1);

return value;

}

**✅ Función de control de matriz**

void controlar\_leds() {

uint16\_t lectura[4];

for (int i = 0; i < 4; i++) {

// Configurar como entrada analógica

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = pines[i];

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

HAL\_GPIO\_Init(puertos[i], &GPIO\_InitStruct);

HAL\_Delay(5); // Pequeña pausa

lectura[i] = leer\_luz(canales[i]);

// Volver a configurar como salida

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(puertos[i], &GPIO\_InitStruct);

HAL\_GPIO\_WritePin(puertos[i], pines[i], GPIO\_PIN\_RESET); // apagar por defecto

}

for (int i = 0; i < 4; i++) {

if (lectura[i] > 1000) { // umbral de luz

for (int j = 0; j < 4; j++) {

if (j != i)

HAL\_GPIO\_WritePin(puertos[j], pines[j], GPIO\_PIN\_SET);

}

}

}

}

**🔁 En el while(1) del main()**

while (1) {

controlar\_leds();

HAL\_Delay(200); // 200 ms de ciclo

}