Taller de Sistemas Embebidos
STM32 MCU - GPIO

Información relevante

Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

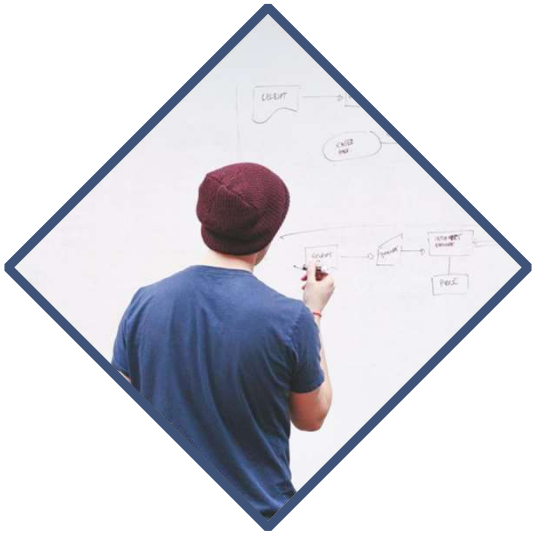
Estructura Curricular de la Carrera

El **Proyecto Intermedio** se desarrolla en la asignatura **Taller de Sistemas Embebidos**, la cual tiene un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**

Más información . . .

. . . sobre la **actualización 2023** . . . https://www.fi.uba.ar/grado/carreras/ingenieria-electronica/plan-de-estudios

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

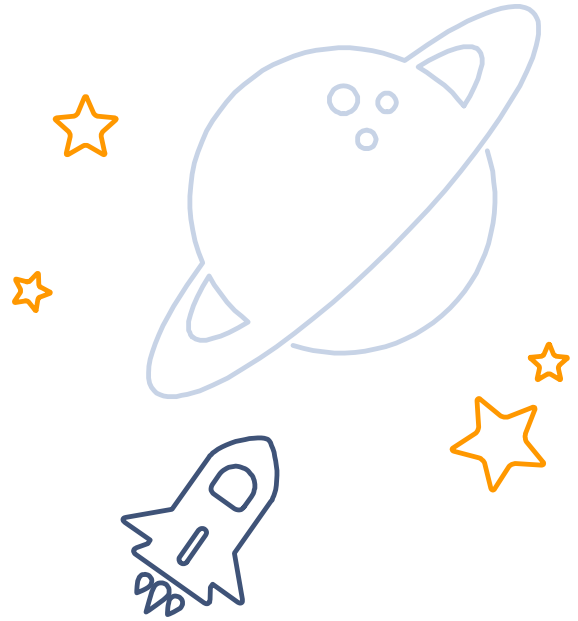
Introducción

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

- Referencia:
  - Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)
  - Chapter 5: General-Purpose Input Output (GPIO) and the STM Hardware Abstraction Layer (HAL)
    - Como dice el título, este capítulo trata sobre cómo funciona GPIO en una placa STM Nucleo-64 y cómo interviene el software HAL. Es importante estudiar primero la arquitectura STM MCU para comprender completamente cómo funcionan los puertos GPIO y su relación con HAL. También muchas de las figuras y el contenido de las tablas provienen de la hoja de datos STM32F302R8 o del manual de referencia. Le recomiendo encarecidamente que los descargue y los tenga disponibles para ayudar a llenar cualquier vacío en mis discusiones.

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Documentación debida

1er Cuatrimestre de 2024, dictado por primera vez . . .

Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)

Chapter 5: General-Purpose Input Output (GPIO) and the STM Hardware Abstraction Layer (HAL)

Block Diagram

- La Figura 5-1 es el diagrama de bloques STM32F302R8, que es la MCU utilizada en la placa de desarrollo Nucleo-64 utilizada en este libro.
- Tenga en cuenta que hay seis bancos de puertos GPIO que se muestran en el lado izquierdo de la Figura 5-1, que están conectados al bus bidireccional avanzado de alta velocidad 1 (AHB1). Este bus transfiere datos digitales internamente a una velocidad de hasta un máximo de 100 Mbps. HAL gestiona eficazmente las transferencias de datos desde los puertos GPIO, así como desde muchos otros componentes periféricos que se muestran en la figura.

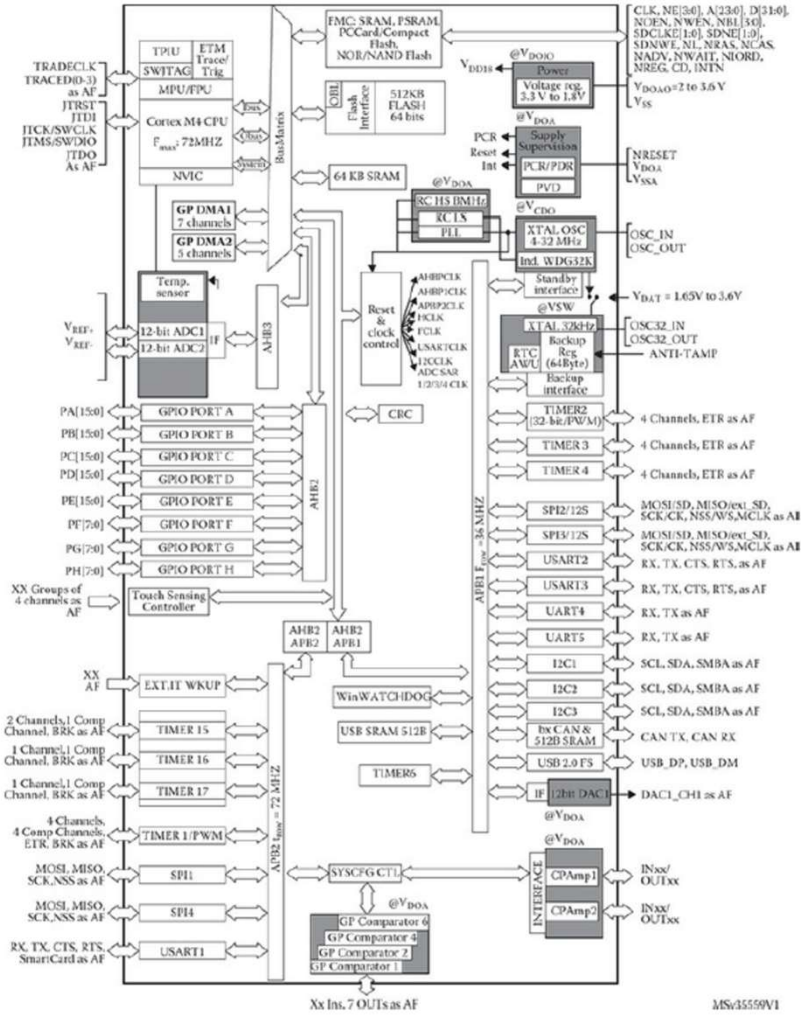


Figure 5-1 *STM32F302R8 block diagram.*

Block Diagram

- Comenzaré la discusión sobre HAL examinando quizás el módulo HAL más simple, que es HAL_GPIO.
- Este módulo ya se ha utilizado varias veces en programas analizados en capítulos anteriores.
- El módulo HAL_GPIO se utilizó sin explicación simplemente porque era necesario implementar una función requerida, como hacer parpadear un LED o leer el estado de un interruptor de botón.
- En breve examinaré de cerca el módulo HAL_GPIO, pero primero discutiré el concepto de periféricos mapeados en memoria.

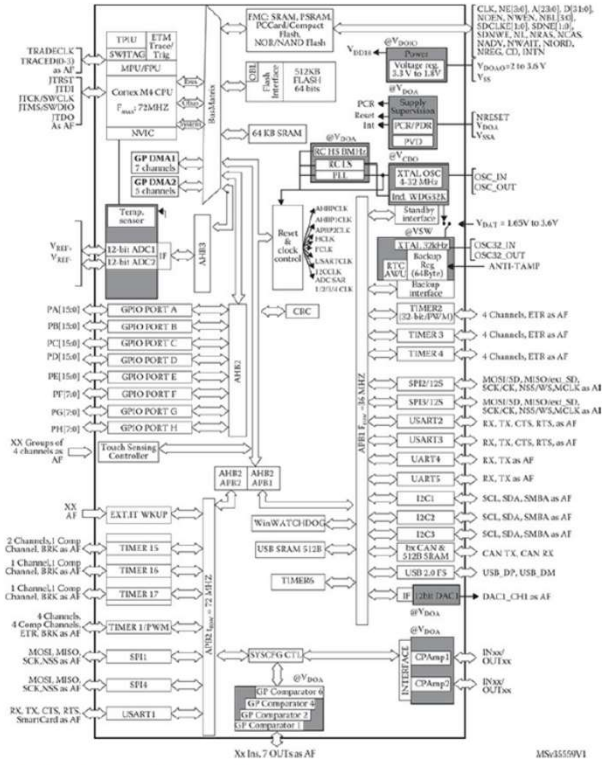


Figure 5-1 *STM32F302R8 block diagram.*

Memory-Mapped Peripherals

- Los periféricos STM MCU se controlan y los datos se transfieren utilizando direcciones dentro de un rango de 32 bits.
- Esto se conoce como mapeo de memoria porque, desde una perspectiva de programación, no hay diferencia si se comunica con una ubicación de memoria o con un registro periférico.
- El rango de direcciones de 32 bits generalmente está representado por ocho dígitos enteros, lo que significa que el rango total posible de ubicaciones es 0x0000 0000 a 0xFFFF FFFF.
- Tenga en cuenta que el número hexadecimal está precedido por un 0x y un espacio va seguido de cuatro de los primeros dígitos hexadecimales.

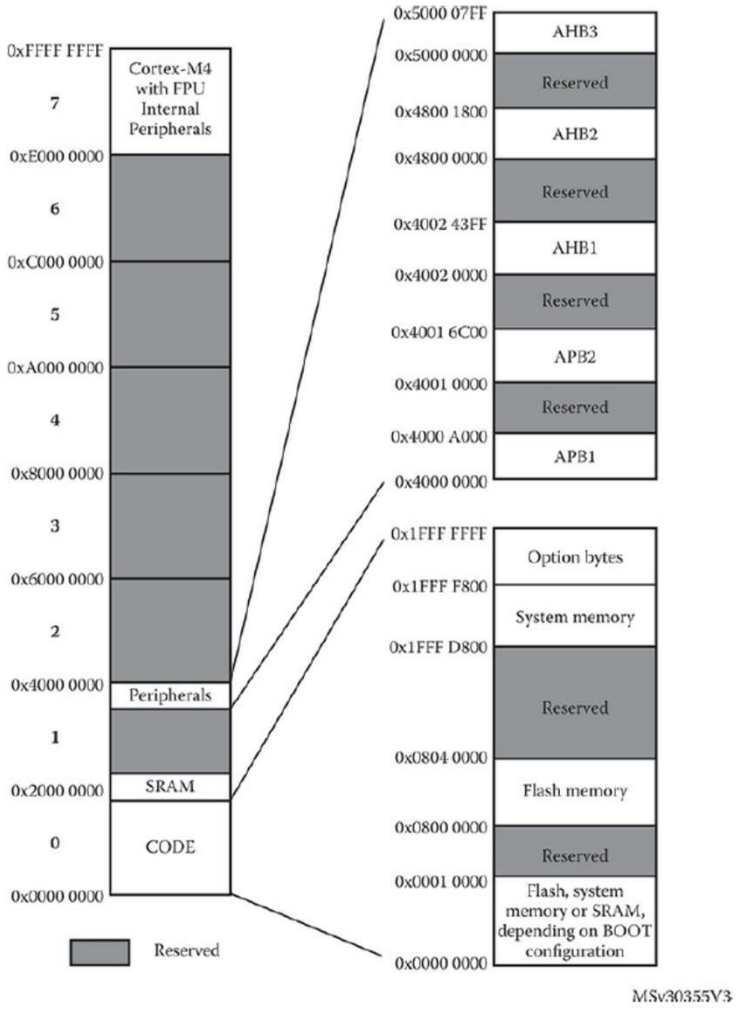


Figure 5-2 *Complete 32-bit STM address range.*

Memory-Mapped Peripherals

- El 0x es una construcción de programación común que indica que el número está en formato hexadecimal y el espacio entre los dos grupos de números hexadecimales es solo para hacer que el número sea más legible. Este formato de dirección también se utiliza en toda la documentación de STM.
- La Figura 5-2 muestra los bloques principales que constituyen todo el rango de direcciones de 32 bits. También se muestran en la Figura 5-2 dos bloques ampliados, que detallan las ubicaciones de la memoria del core, así como las ubicaciones de los periféricos/bus.
- A continuación discutiré brevemente las ubicaciones de la memoria del core, que son importantes por derecho propio, pero no muy relevantes para las ubicaciones de las direcciones de periféricos.

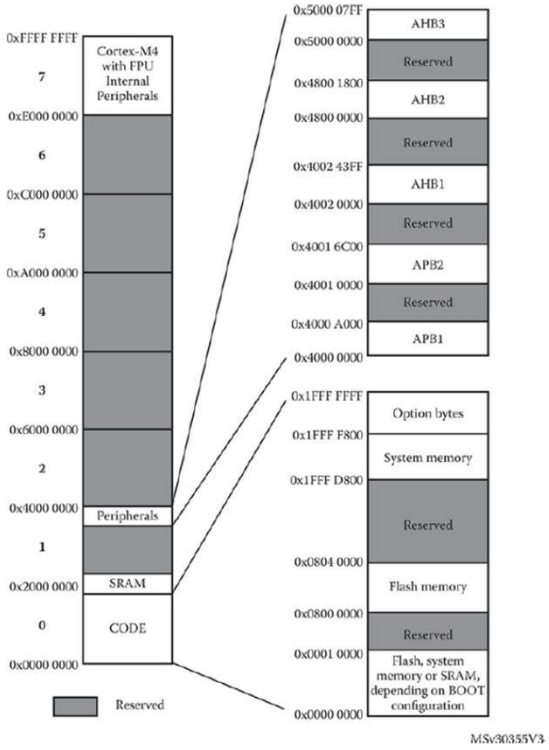


Figure 5-2 *Complete 32-bit STM address range.*

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Core Memory Addresses

- Una pregunta que muchos lectores podrían estar planteándose es ¿cómo una PC puede manejar 8, 16 o incluso 32 GB de SRAM con sólo 32 bits de capacidad de direccionamiento?
- La respuesta está en utilizar un sistema operativo diseñado para accesos extendidos a la memoria y una unidad de hardware conocida como Memory Management Unit (MMU).
- Todos los principales sistemas operativos de PC, como Windows, OSX y Linux, tienen incorporada una funcionalidad de memoria extendida que permite que el direccionamiento de 32 bits se extienda a múltiples bloques o páginas.
- Ud. podría pensar en utilizar un registro de 4 bits, donde cada bit activaría un bloque o página en particular.

Core Memory Addresses

- Por lo tanto, el bit 0 permitiría el acceso a todas las direcciones base comenzando en 0x0000 0000, mientras que el bit 1 permitiría que el direccionamiento comience virtualmente en 0x0001 0000 0000.
- Por supuesto, solo se utilizan 32 bits reales, pero el sistema operativo maneja las direcciones como si comenzaron una ubicación más allá de 0xFFFF FFFF.
- Se puede considerar que este registro de 4 bits forma parte de la MMU. Cuatro bits significa que se puede manejar un máximo de 64 GB en un PC de 32 bits, que casualmente suele ser la SRAM máxima que se puede instalar en un PC de este tipo.

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Core Memory Addresses

- Por supuesto, todas las limitaciones prácticas de la memoria física se alivian cuando se utiliza un sistema de 64 bits, pero eso requiere una CPU costosa y requisitos de energía adicionales.
- Por lo tanto, el bit 0 permitiría el acceso a todas las direcciones base comenzando en 0x0000 0000, mientras que el bit 1 permitiría que el direccionamiento comience virtualmente en 0x0001 0000 0000.
- Por supuesto, solo se utilizan 32 bits reales, pero el sistema operativo maneja las direcciones como si comenzaron una ubicación más allá de 0xFFFF FFFF.

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Core Memory Addresses

- Se puede considerar que este registro de 4 bits forma parte de la MMU. Cuatro bits significa que se puede manejar un máximo de 64 GB en un PC de 32 bits, que casualmente suele ser la SRAM máxima que se puede instalar en un PC de este tipo.
- Por supuesto, todas las limitaciones prácticas de la memoria física se alivian cuando se utiliza un sistema de 64 bits, pero eso requiere una CPU costosa y requisitos de energía adicionales.
- La discusión anterior no es aplicable a una MCU STM típica porque la memoria es bastante limitada y no se necesitan grandes cantidades para manejar aplicaciones integradas típicas.

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Core Memory Addresses

- Quiero mencionar que STM32F302R8 incorpora una unidad de protección de memoria (MPU). La MPU no es una MMU, pero se proporciona para administrar los accesos a la memoria de la CPU de manera que una tarea no se corrompa accidentalmente ni entre en conflicto con los accesos a la memoria de otra tarea.
- La memoria se puede organizar en hasta ocho áreas, cada área con un tamaño de entre 32 B y 4 GB; la última comprende todo el rango de memoria direccionable.
- La MPU normalmente está controlada por un sistema operativo en tiempo real (RTOS), que tiene una API contenida dentro del CMSIS, que se presentó en el último capítulo.

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Core Memory Addresses

- No usaré la MPU para ninguno de los proyectos de libros.
- Debe considerar su uso solo para las aplicaciones más críticas para la seguridad o de misión esencial.
- Piense en su uso en sistemas de seguridad para automóviles o vuelos.

Memory Type	Address Range	Size (KB)	Remark
Flash	0x0800 0000 to 0x0803 FFFF	64	–
System	0x1FFF D800 to 0x1FFF E7FF	24	Used with an operating system
SRAM	0x2000 0000 to 0x2000 3FFF	16	–
Aliased	0x0000 0000 to 0x0000 FFFF	64	Aliased to Flash, System, or SRAM depending on the configuration of the BOOT pins

Table 5-1 *STM Core Memory Address Ranges*

Peripheral Memory Addresses

- La Figura 5-3 muestra la parte inicial de las direcciones de memoria asignadas a los periféricos STM32F302R8.
- Debería poder ver que el bus AHB2 tiene el rango de direcciones 0x4800 0000 a 0x4FFF 17FF.
- Lo que es específico del enfoque de este capítulo es el rango GPIO total de 0x4002 0000 a 0x4002 0x1FFF, que se divide en seis bloques más pequeños de 1 KB.
- Cinco de los seis bloques de 1 KB están asignados a los puertos GPIO A, B, C, D y F, respectivamente.
- Hay un bloque no utilizado en el rango 0x4800 1000 a 0x4800 13FF, que no tiene un impacto significativo en las asignaciones de puertos.

Bus	Boundary address	Size (bytes)	Peripheral
AHB3	0x5000 0000 - 0x5000 03FF	1 K	ADC1
	0x4800 1800 - 0x4FFF FFFF	~132 M	Reserved
AHB2	0x4800 1400 - 0x4800 17FF	1 K	GPIOF
	0x4800 1000 - 0x4800 13FF	1 K	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 K	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 K	GPIOC
	0x4800 0400 - 0x4800 07FF	1 K	GPIOB
	0x4800 0000 - 0x4800 03FF	1 K	GPIOA
	0x4002 4400 - 0x47FF FFFF	~128 M	Reserved
AHB1	0x4002 4000 - 0x4002 43FF	1 K	TSC
	0x4002 3400 - 0x4002 3FFF	3 K	Reserved
	0x4002 3000 - 0x4002 33FF	1 K	CRC
	0x4002 2400 - 0x4002 2FFF	3 K	Reserved
	0x4002 2000 - 0x4002 23FF	1 K	Flash interface
	0x4002 1400 - 0x4002 1FFF	3 K	Reserved
	0x4002 1000 - 0x4002 13FF	1 K	RCC
	0x4002 0800 - 0x4002 0FFF	2 K	Reserved
	0x4002 0400 - 0x4002 07FF	1 K	Reserved
	0x4002 0000 - 0x4002 03FF	1 K	DMA1
	0x4001 8000 - 0x4001 FFFF	32 K	Reserved
APB2	0x4001 4C00 - 0x4001 7FFF	13 K	Reserved
	0x4001 4800 - 0x4001 4BFF	1 K	TIM17
	0x4001 4400 - 0x4001 47FF	1 K	TIM16
	0x4001 4000 - 0x4001 43FF	1 K	TIM15
	0x4001 3C00 - 0x4001 3FFF	1 K	Reserved
	0x4001 3800 - 0x4001 3BFF	1 K	USART1
	0x4001 3400 - 0x4001 37FF	1 K	Reserved
	0x4001 3000 - 0x4001 33FF	1 K	Reserved
	0x4001 0800 - 0x4001 2FFF	10 K	TIM1
	0x4001 0400 - 0x4001 07FF	1 K	EXTI
	0x4001 0000 - 0x4001 03FF	1 K	SYSCFG + COMP + OPAMP
	0x4000 7C00 - 0x4000 FFFF	33 K	Reserved

Figure 5-3 *Beginning portion of the STM32F302R8 peripheral allocations.*

Peripheral Memory Addresses

- No tengo idea de por qué hay un bloque reservado asignado en lo que era una asignación de bloque contiguo o por qué el puerto GPIO final está etiquetado como F en lugar de E, como cabría esperar de las designaciones de puertos anteriores.
- La cuestión de la designación y asignación de la memoria del puerto probablemente solo pueda ser respondida por los diseñadores de chips STM MCU. Depende de los usuarios de MCU aceptar las designaciones y asignaciones tal como están y trabajar con ellas lo mejor que puedan.
- Elegí incluir la siguiente figura sobre las ubicaciones de los periféricos para que esté completo y para proporcionar una referencia fácilmente disponible para periféricos adicionales.

Bus	Boundary address	Size (bytes)	Peripheral
AHB3	0x5000 0000 - 0x5000 03FF	1 K	ADC1
	0x4800 1800 - 0x4FFF FFFF	~132 M	Reserved
AHB2	0x4800 1400 - 0x4800 17FF	1 K	GPIOF
	0x4800 1000 - 0x4800 13FF	1 K	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 K	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 K	GPIOC
	0x4800 0400 - 0x4800 07FF	1 K	GPIOB
	0x4800 0000 - 0x4800 03FF	1 K	GPIOA
	0x4002 4400 - 0x47FF FFFF	~128 M	Reserved
AHB1	0x4002 4000 - 0x4002 43FF	1 K	TSC
	0x4002 3400 - 0x4002 3FFF	3 K	Reserved
	0x4002 3000 - 0x4002 33FF	1 K	CRC
	0x4002 2400 - 0x4002 2FFF	3 K	Reserved
	0x4002 2000 - 0x4002 23FF	1 K	Flash interface
	0x4002 1400 - 0x4002 1FFF	3 K	Reserved
	0x4002 1000 - 0x4002 13FF	1 K	RCC
	0x4002 0800 - 0x4002 0FFF	2 K	Reserved
	0x4002 0400 - 0x4002 07FF	1 K	Reserved
	0x4002 0000 - 0x4002 03FF	1 K	DMA1
	0x4001 8000 - 0x4001 FFFF	32 K	Reserved
APB2	0x4001 4C00 - 0x4001 7FFF	13 K	Reserved
	0x4001 4800 - 0x4001 4BFF	1 K	TIM17
	0x4001 4400 - 0x4001 47FF	1 K	TIM16
	0x4001 4000 - 0x4001 43FF	1 K	TIM15
	0x4001 3C00 - 0x4001 3FFF	1 K	Reserved
	0x4001 3800 - 0x4001 3BFF	1 K	USART1
	0x4001 3400 - 0x4001 37FF	1 K	Reserved
	0x4001 3000 - 0x4001 33FF	1 K	Reserved
	0x4001 0800 - 0x4001 2FFF	10 K	TIM1
	0x4001 0400 - 0x4001 07FF	1 K	EXTI
	0x4001 0000 - 0x4001 03FF	1 K	SYSCFG + COMP + OPAMP
	0x4000 7C00 - 0x4000 FFFF	33 K	Reserved

Figure 5-3 *Beginning portion of the STM32F302R8 peripheral allocations.*

- La Figura 5-4 muestra los periféricos conectados al bus interno de periféricos avanzados 1 (APB1). Los periféricos conectados al bus son los siguientes:
  - Timers (TIMxx)
  - Universal Synchronous/Asynchronous Receiver/Transmitters
  - (USARTx)
  - Serial peripheral interfaces (SPIx)
  - Inter-integrated interfaces (I2Cx)
  - Universal serial bus (USB)
  - Integrated sound interface (I2S)

Bus	Boundary address	Size (bytes)	Peripheral
APB1	0x4000 7800 - 0x4000 7BFF	9 K	I2C3
	0x4000 7400 - 0x4000 77FF	1 K	DAC1
	0x4000 7000 - 0x4000 73FF	1 K	PWR
	0x4000 6C00 - 0x4000 6FFF	1 K	Reserved
	0x4000 6800 - 0x4000 6BFF	1 K	Reserved
	0x4000 6400 - 0x4000 67FF	1 K	bxCAN
	0x4000 6000 - 0x4000 63FF	1 K	USB/CAN SRAM
	0x4000 5C00 - 0x4000 5FFF	1 K	USB device FS
	0x4000 5800 - 0x4000 5BFF	1 K	I2C2
	0x4000 5400 - 0x4000 57FF	1 K	I2C1
	0x4000 5000 - 0x4000 53FF	1 K	Reserved
	0x4000 4C00 - 0x4000 4FFF	1 K	Reserved
	0x4000 4800 - 0x4000 4BFF	1 K	USART3
	0x4000 4400 - 0x4000 47FF	1 K	USART2
	0x4000 4000 - 0x4000 43FF	1 K	I2S3ext
	0x4000 3C00 - 0x4000 3FFF	1 K	SPI3/I2S3
	0x4000 3800 - 0x4000 3BFF	1 K	SPI2/I2S2
	0x4000 3400 - 0x4000 37FF	1 K	I2S2ext
	0x4000 3000 - 0x4000 33FF	1 K	IWDG
	0x4000 2C00 - 0x4000 2FFF	1 K	WWDG
	0x4000 2800 - 0x4000 2BFF	1 K	RTC
	0x4000 1400 - 0x4000 27FF	5 K	Reserved
	0x4000 1000 - 0x4000 13FF	1 K	TIM6
	0x4000 0C00 - 0x4000 0FFF	1 K	Reserved
	0x4000 0800 - 0x4000 0BFF	1 K	Reserved
	0x4000 0400 - 0x4000 07FF	1 K	Reserved
	0x4000 0000 - 0x4000 03FF	1 K	TIM2
	0x2000 A000 - 3FFF FFFF	~512 M	Reserved
	0x2000 0000 - 0x2000 9FFF	40 K	SRAM
	0x1FFF F800 - 0x1FFF FFFF	2 K	Option bytes
	0x1FFF D800 - 0x1FFF F7FF	8 K	System memory
	0x0804 0000 - 0x1FFF D7FF	~384 M	Reserved
	0x0800 0000 - 0x0800 FFFF	64 K	Main Flash memory

Figure 5-4 *AHB1 peripheral allocations.*

Peripheral Memory Addresses

- Digital-to-analog converter (DAC)
- Controller area network interface (CAN)
- Real-time clock (RTC)
- Watch dog timers (IWDG, WWDG)

- Ahora volveré a la discusión sobre HAL_GPIO después de presentar el concepto de direcciones periféricas asignadas en memoria.

Bus	Boundary address	Size (bytes)	Peripheral
APB1	0x4000 7800 - 0x4000 7BFF	9 K	I2C3
	0x4000 7400 - 0x4000 77FF	1 K	DAC1
	0x4000 7000 - 0x4000 73FF	1 K	PWR
	0x4000 6C00 - 0x4000 6FFF	1 K	Reserved
	0x4000 6800 - 0x4000 6BFF	1 K	Reserved
	0x4000 6400 - 0x4000 67FF	1 K	bxCAN
	0x4000 6000 - 0x4000 63FF	1 K	USB/CAN SRAM
	0x4000 5C00 - 0x4000 5FFF	1 K	USB device FS
	0x4000 5800 - 0x4000 5BFF	1 K	I2C2
	0x4000 5400 - 0x4000 57FF	1 K	I2C1
	0x4000 5000 - 0x4000 53FF	1 K	Reserved
	0x4000 4C00 - 0x4000 4FFF	1 K	Reserved
	0x4000 4800 - 0x4000 4BFF	1 K	USART3
	0x4000 4400 - 0x4000 47FF	1 K	USART2
	0x4000 4000 - 0x4000 43FF	1 K	I2S3ext
	0x4000 3C00 - 0x4000 3FFF	1 K	SPI3/I2S3
	0x4000 3800 - 0x4000 3BFF	1 K	SPI2/I2S2
	0x4000 3400 - 0x4000 37FF	1 K	I2S2ext
	0x4000 3000 - 0x4000 33FF	1 K	IWDG
	0x4000 2C00 - 0x4000 2FFF	1 K	WWDG
	0x4000 2800 - 0x4000 2BFF	1 K	RTC
	0x4000 1400 - 0x4000 27FF	5 K	Reserved
	0x4000 1000 - 0x4000 13FF	1 K	TIM6
	0x4000 0C00 - 0x4000 0FFF	1 K	Reserved
	0x4000 0800 - 0x4000 0BFF	1 K	Reserved
	0x4000 0400 - 0x4000 07FF	1 K	Reserved
	0x4000 0000 - 0x4000 03FF	1 K	TIM2
	0x2000 A000 - 3FFF FFFF	~512 M	Reserved
	0x2000 0000 - 0x2000 9FFF	40 K	SRAM
	0x1FFF F800 - 0x1FFF FFFF	2 K	Option bytes
	0x1FFF D800 - 0x1FFF F7FF	8 K	System memory
	0x0804 0000 - 0x1FFF D7FF	~384 M	Reserved
	0x0800 0000 - 0x0800 FFFF	64 K	Main Flash memory

Figure 5-4 *AHB1 peripheral allocations.*

HAL_GPIO Module

- Sería aconsejable examinar un circuito GPIO de muestra antes de analizar cómo funciona con el software del módulo.
- Siempre es un paso importante para comprender el hardware subyacente que implementa el software.
- GPIO Pin Hardware
  - La Figura 5-5 es un diagrama de bloques, que también contiene esquemas que modelan un pin de puerto STM GPIO típico.
  - A partir del lado derecho de la Figura 5-5, puede ver el pin de E/S, que tiene conectados circuitos robustos de protección contra sobretensiones.

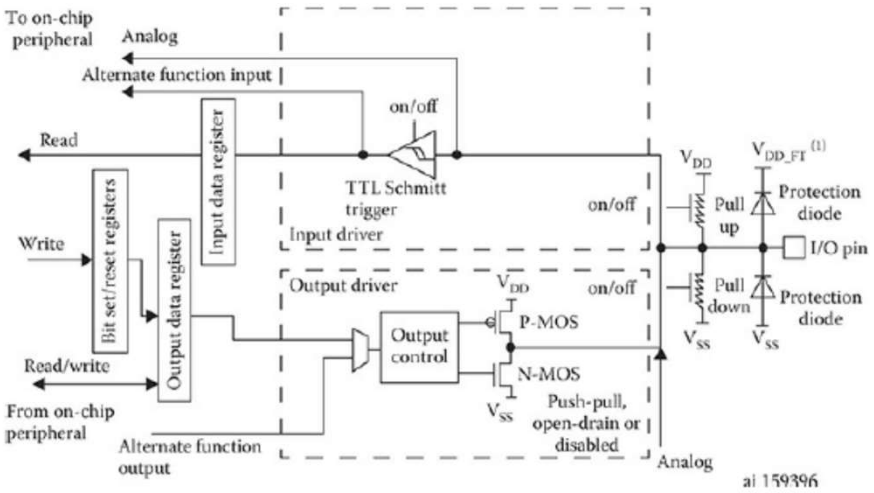


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- El pin GPIO está protegido hasta un nivel de entrada aplicado de 5 V a pesar de que el suministro de voltaje principal de la MCU es de 3,3 V y los voltajes del procesador central son de 1,8 V.
- Esto hace que la interfaz sea bastante sencilla sin la necesidad de emplear cambiadores de nivel de voltaje. o divisores de voltaje de resistencia.
- El pin de E/S está conectado simultáneamente a los controladores de entrada y salida, como se muestra en la figura.
- Sin embargo, sólo se puede activar un controlador a la vez.

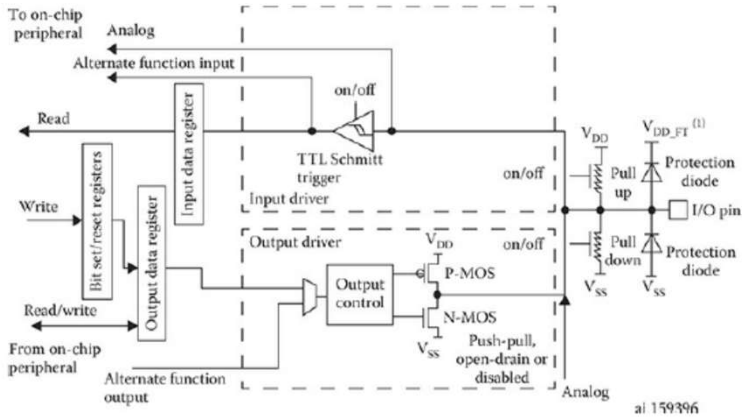


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- La inspección de la sección del controlador de entrada revela tres posibles entradas que pueden enviarse a la MCU central. Estos son los siguientes:
  - Analog: Bypasses the Schmitt trigger
  - Alternate function: Schmitt trigger output
  - Read: salida de una de las líneas de registro de lectura asociadas.
- Los ajustes de configuración del pin GPIO determinan qué entrada se utiliza en un momento dado.

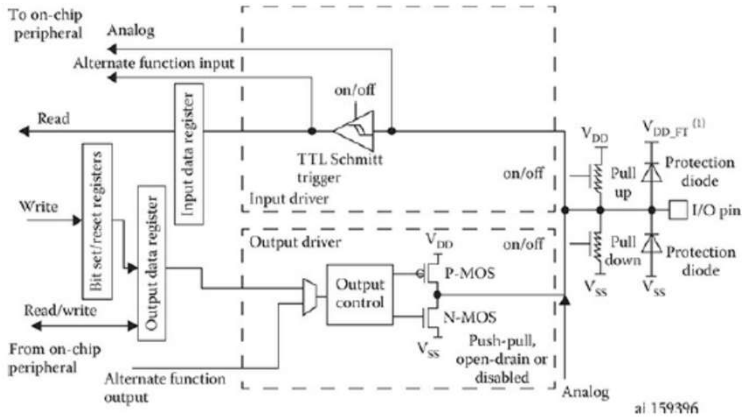


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- Asimismo, la sección de salida tiene múltiples formas de crear una salida de señal para el pin de E/S. Estos son los siguientes:
  - Write: Salida de una línea de registro de escritura asociada. El registro de salida también tiene una línea de control R/W.
  - Alternate function: Generada desde un periférico MCU interno.

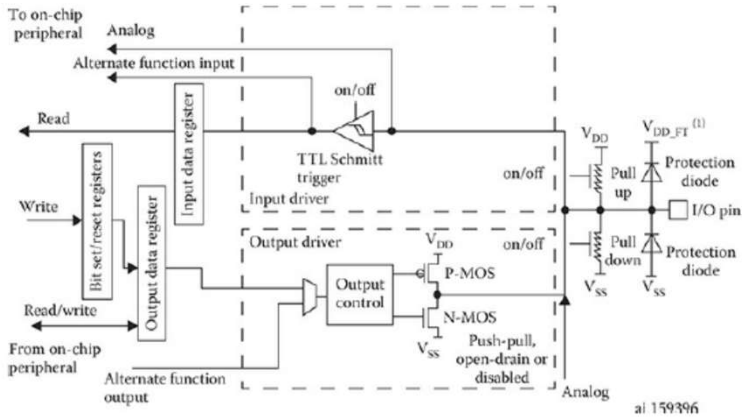


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- Tenga en cuenta que el bloque de control de salida tiene dos transistores MOSFET conectados al circuito de control de salida, que realizan las siguientes operaciones:
  - Push-pull
  - Open-drain
  - Disabled

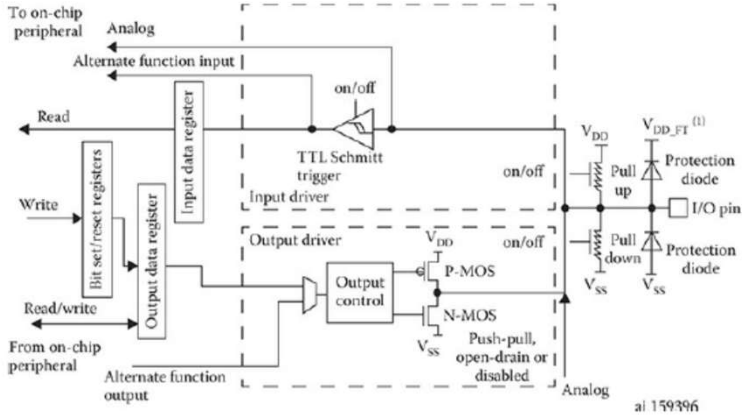


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- Finalmente, el pin de E/S tiene resistencias habilitadas programables conectadas al pin, lo que permitirá lo siguiente:
  - Pull-up
  - Pull-down
  - Neither
- Este circuito de muestra se replica para cada pin GPIO existente en el MCU.
- Cada puerto GPIO tiene varios registros adjuntos, que controlan todos los comportamientos de sus pines configurando o restableciendo bits asociados con los pines respectivos.

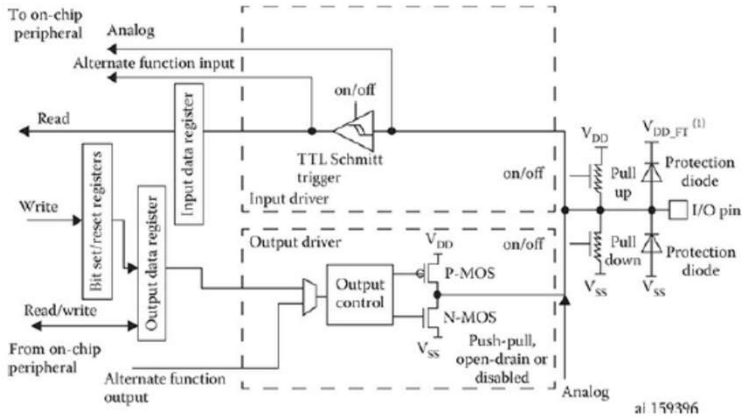
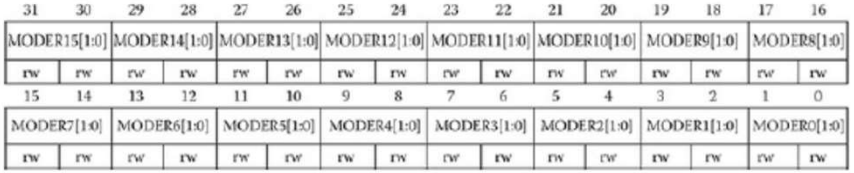


Figure 5-5 *Typical STM GPIO port pin block diagram/schematic.*

HAL_GPIO Module

- La Figura 5-6 muestra el formato del registro de modo de puerto GPIO para los puertos A… E y H.

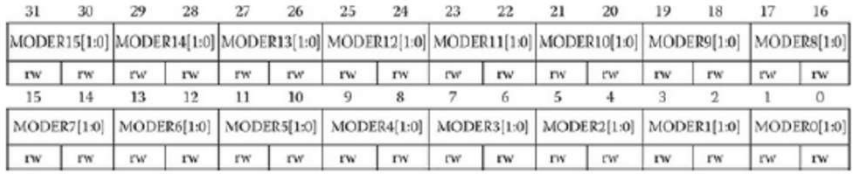


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5-6 *GPIO port mode register.*

- Se accedería a este registro mediante programación utilizando el símbolo GPIOx_MODER donde x = A … E y H para el puerto de destino. Los valores de reinicio de encendido (poweron reset) colocados automáticamente en todos los registros GPIO son los siguientes:
  - 0x0C00 0000 para el puerto A
  - 0x0000 0280 para el puerto B
  - 0x0000 0000 para todos los demás puertos

HAL_GPIO Module



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5-6 *GPIO port mode register.*

- Los valores de di-bit o dos bits que se pueden almacenar en un registro tienen los siguientes significados:
  - 00—Input mode
  - 01—Output mode
  - 10—Alternate function mode
  - 11—Analog mode

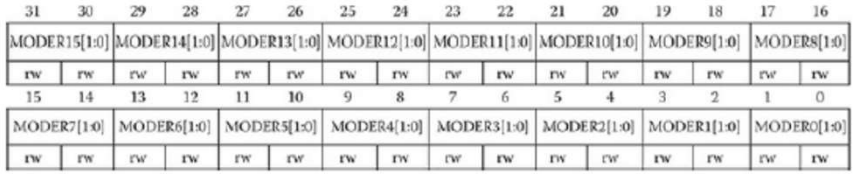
HAL_GPIO Module

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5-6 *GPIO port mode register.*

- Los registros GPIO adicionales configuran otros comportamientos más asociados con los pines del puerto, incluidos los siguientes:
  - Enabling push-pull, open-drain, or none (also called floating)
  - Enabling pull-up, pull-down, or none
  - Operational speed

HAL_GPIO Module



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5-6 *GPIO port mode register.*

- Las siguientes dos figuras muestran de manera concisa todos los registros involucrados en todas las configuraciones GPIO.
- Estas cifras son muy importantes y servirán como base para configurar los puertos GPIO para todos los proyectos futuros.
- Tuve que dividir los listados del registro GPIO en dos cifras separadas debido a la cantidad de registros involucrados.

- La Figura 5-7 muestra los primeros nueve registros en orden de sus direcciones de desplazamiento con respecto a la dirección base MODER.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOA_ MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00	GPIOB_ MODER	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0x00	GPIOx_ MODER (where x = C..E and H)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	GPIOx_ OTYPER (where x = A..E and H)	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOx_ OSPEEDER (where x = C..E and H)	OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]		OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOA_ OSPEEDER	OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]		OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]	
	Reset value	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOB_ OSPEEDER	OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]		OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Figure 5-7 *First nine GPIO registers' details in order of offset addresses.*

- Los últimos nueve registros en el orden de dirección de desplazamiento se muestran en la Figura 5-8.
- La clave para leer estas figuras es comprender qué se supone que debe controlar cada registro, su dirección de compensación y cómo se accede a los pines GPIO individuales en el registro.
- A continuación demostraré un programa simple que controlará un LED conectado al puerto A, pin PA10.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0C	GPIOA_PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOB_PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0x0C	GPIOx_PUPDR (where x = C..E and H)	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..E and H)	Reserved																IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x14	GPIOx_ODR (where x = A..E and H)	Reserved																ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_BSRR (where x = A..E and H)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	GPIOx_LCKR (where x = A..E and H)	Reserved															LCKK	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	GPIOx_AFRL (where x = A..E and H)	AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]				AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	GPIOx_AFRH (where x = A..E and H)	AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]				AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-8 *Last nine GPIO registers' details in order of offset addresses.*

LED Test Demonstration

- El propósito de la siguiente demostración es mostrar cómo configurar el puerto que registra un GPIO para controlar directamente un LED. El programa de demostración no hará parpadear el LED y deberá cambiar el código fuente, volver a compilarlo y recargarlo para afectar el estado del LED.
- Sin embargo, se debe realizar un poco de preparación del hardware antes de poder ejecutar el programa de prueba. Eso implica conectar el LED con una resistencia limitadora de corriente al pin del puerto A seleccionado. Para facilitar este requisito, elegí utilizar una placa de prototipos Arduino, que se muestra en la Figura 5-9. Esta placa es bastante económica y está disponible en varias fuentes en línea.

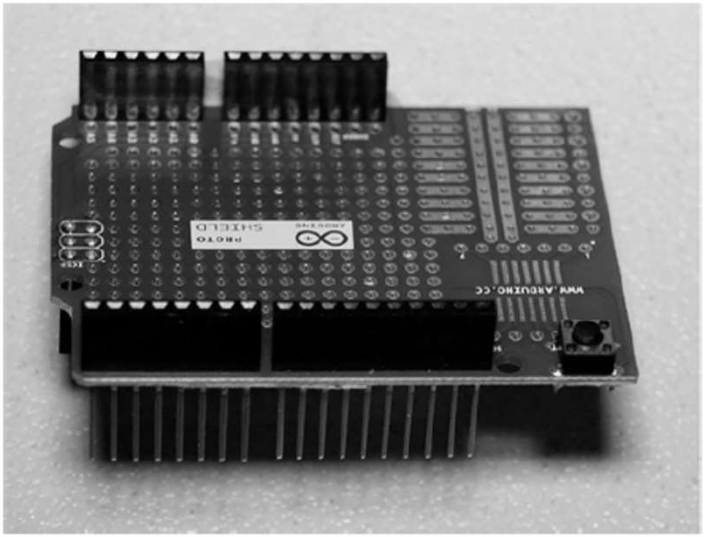


Figure 5-9 *Arduino prototype board.*

- También necesitará un LED y una resistencia de 330 Ω y 1/4 vatio para limitar la corriente que fluye a través del LED. También será importante saber qué pines GPIO están expuestos en el protoboard Arduino. Esto se responde fácilmente consultando el manual del usuario de la placa STM Nucleo-64.
- La Figura 5-10 proviene de ese manual y muestra los pines STM32F302R8 asociados con los conectores Arduino y Morpho. En nuestro caso, sólo se necesitan los pines de Arduino.
- Seleccioné arbitrariamente PA10 como salida LED, simplemente porque aún no estaba configurado para un periférico en el programa de plantilla main.c. La Figura 5-11 es el esquema utilizado para esta configuración.

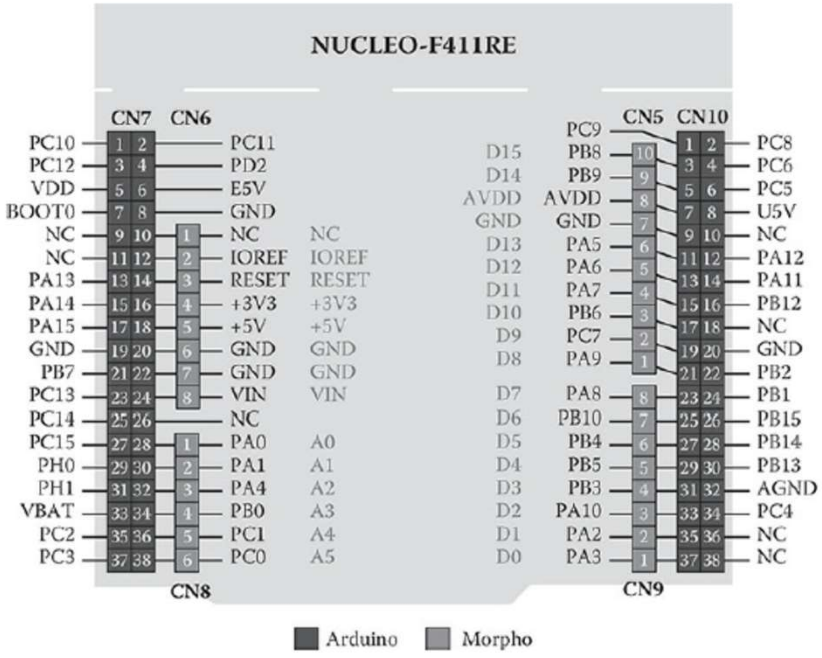


Figure 5-10 *Arduino/Morpho pin connectors.*

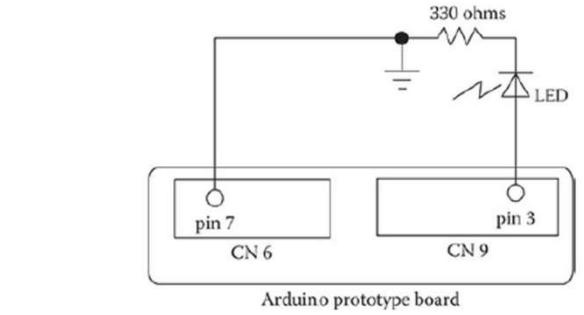


Figure 5-11 *LED test schematic.*

- El cable LED más largo está conectado al ánodo y el cable más corto está conectado al cátodo. Simplemente soldé la resistencia y el LED porque esta disposición me permitió conectar fácilmente la combinación LED/resistencia al protoboard, como se muestra en la Figura 5-12. El protoboard también está conectado a la placa Nucleo-64 para esta fotografía.
- Es hora de trabajar en el software una vez que el hardware esté configurado.

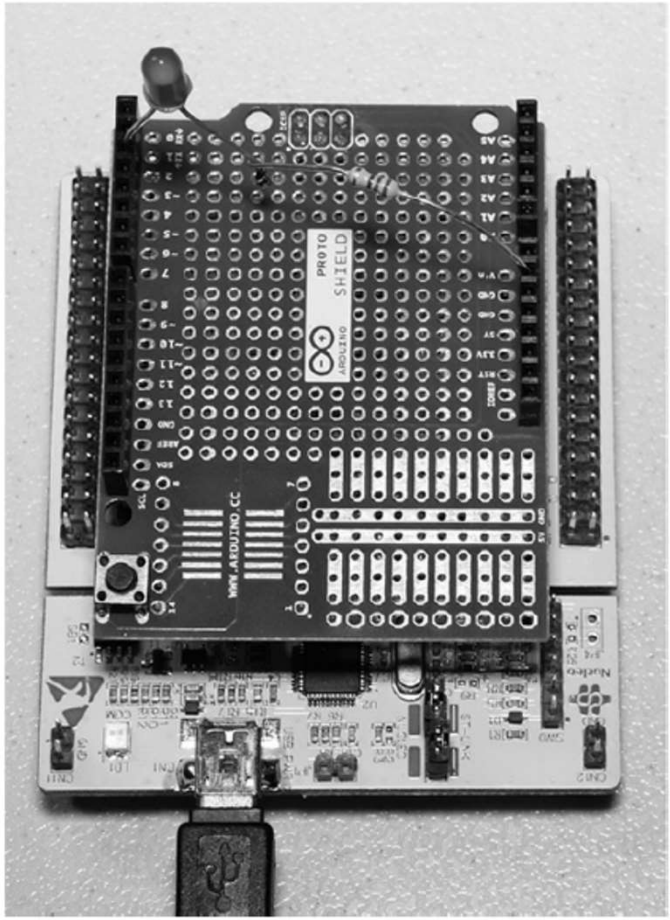


Figure 5-12 *LED/resistor combination plugged into the protoboard.*

LED Test Program

- Como mencioné anteriormente, todos los periféricos STM, incluidos los puertos GPIO, se controlan mediante la configuración de una serie de registros relacionados, como se detalla en las Figuras 5-7 y 5-8.
- En este programa de demostración configuraré el pin PA10, que se encuentra en el banco de pines GPIO del Puerto A.
- Será necesario configurar PA10 como salida.
- Usaré un método principal ligeramente modificado creado en el proyecto del capítulo anterior.
- Debe configurar un nuevo proyecto con este método principal y simplemente modificar el método ingresando las dos líneas siguientes y realizando dos cambios en el método MX_GPIO_Init(void) como se detalla a continuación.

LED Test Program

- Ingrese estas dos declaraciones inmediatamente después de la primera declaración MX_GPIO_Init():

```
HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_
SET);
while (1){}
```

Make the following two changes in the `static void MX_GPIO_Init(void)` method:

1. `HAL_GPIO_WritePin(LD2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);`

   to

   `HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_RESET);`

2. `GPIO_InitStruct.Pin = LD2_Pin;`

   to

   `GPIO_InitStruct.Pin = GPIO_PIN_10;`

LED Test Program

- La biblioteca HAL se basa en una estructura C llamada GPIO_InitStruct para configurar cualquier pin GPIO. El uso de esta estructura oculta la necesidad de utilizar direcciones reales asignadas en memoria para configurar un pin GPIO.
- El siguiente segmento de código del método MX_GPIO_Init(void) muestra cómo se inicializa la estructura para configurar PA10 en un pin de salida:

```
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
```

Structure Component	GPIO Register	Remark
GPIO_InitStruct.Pin	GPIOx_OTYPER	Pin id
GPIO_InitStruct.Mode	GPIOx_MODER	Input or output
GPIO_InitStruct.Pull	GPIOx_PUPDR_	Push-pull, open-drain, floating
GPIO_InitStruct.Speed	GPIOx_OSPEEDER	100, 75, or 50 MHz

Table 5-2 *GPIO_InitStruct to GPIO Register Equivalence*

- La última declaración de lista no forma parte de la inicialización de la estructura, pero es necesaria para aplicar la estructura recién inicializada dentro del marco HAL.
- Los componentes de la estructura están directamente relacionados con los registros GPIO asignados en memoria, como se muestra en la Tabla 5-2.

LED Test Progra

```
/** The STM disclaimer goes here */

// Includes
#include "main.h"
#include "stm32f3xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

// Private variables

/* USER CODE BEGIN PV */
// Private variables

/* USER CODE END PV */

// Private function prototypes
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
/* USER CODE BEGIN PFP */
// Private function prototypes

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  // MCU Configuration

  // Reset of all peripherals, Initializes the Flash interface and the
  // Systick.
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* USER CODE BEGIN 2 */
  // The next statement will turn-on GPIOA pin 10
  HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_SET);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */
}
```

```
// System Clock Configuration
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

  // Initialize the CPU, AHB and APB buss clocks
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  // Initialize the CPU, AHB and APB buss clocks
  RCC_ClkInitStruct.ClockType =
  RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
  |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
  HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  // Configure the Systick interrupt time
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  // Configure the Systick
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
     PA2   ------> USART2_TX
     PA3   ------> USART2_RX
*/
```

LED Test Pro

Debe notar que la lista de códigos del método principal termina con una instrucción while (1), que simplemente obliga a la MCU a realizar un bucle indefinido después de que se enciende el LED

La única forma de apagar el LED es desconectar la alimentación de la placa. Reiniciar la placa solo encenderá inmediatamente el LED.

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  // Configure GPIO_PIN_10 pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : USART_TX_Pin USART_RX_Pin */
  GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

// Configure GPIO pin: GPIO_Pin_10
  GPIO_InitStruct.Pin = GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  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

/**
  * @}
  */

/**
  * @}
*/

/**** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

LED Test Program

Test Run

- Primero deberá compilar el proyecto en preparación para cargarlo en la placa Nucleo-64. Luego utiliza la utilidad ST-LINK para transferir el archivo hexadecimal desde el directorio Debug del proyecto al Nucleo, como se describió anteriormente en el Capítulo 4.
- El LED debe encenderse inmediatamente una vez que el archivo hexadecimal se carga en la placa. Vuelva a verificar el código main.c si el LED no se enciende.
- Por cierto, vuelva a verificar que haya insertado el hardware en los zócalos de pines adecuados. A veces, es tan sencillo como solucionar un problema.

LED Test Program

Enabling Multiple Outputs

- Es relativamente fácil modificar el archivo main.c para habilitar múltiples salidas. Seleccioné GPIO PA6 como salida adicional para controlar un LED.
- Por cierto, me referí a las definiciones contenidas en el archivo main.h para determinar los pines ya comprometidos en el proyecto.
- He incluido la parte relevante del archivo en la siguiente lista para su uso.Tenga en cuenta que PA10 no se agregó como definición; habría agregado eso si estuviera construyendo un programa permanente en lugar de un programa de demostración transitorio. Lo mismo ocurre con el pin PA6.

```
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
```

LED Test Program

- Realice los siguientes cambios para habilitar las múltiples salidas de LED.
- En el método main.
- Deberá volver a compilar y cargar el nuevo archivo hexadecimal en la placa Nucleo-64 para probar la nueva funcionalidad.

1. `HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_SET);`

   to

   `HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_6 | GPIO_PIN_10, GPIO_PIN_SET);`

In the `static void MX_GPIO_Init(void)` method change:

1. `HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_PIN_RESET);`

   to

   `HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_6 | GPIO_PIN_10, GPIO_PIN_RESET);`

2. `GPIO_InitStruct.Pin = GPIO_PIN_10;`

   to

   `GPIO_InitStruct.Pin = GPIO_PIN_6 | GPIO_PIN_10;`

LED Test Program

Test Run

- Dejé la combinación de LED/resistencia conectada entre PA10 y tierra para volver a comprobar que se encendía cuando se cargaba el programa.
- Siempre es aconsejable hacer ese tipo de verificación porque a veces suceden cosas no deseadas cuando se realiza una modificación del programa, que se cree que no afecta una función existente y probada.
- Luego desconecté el circuito LED/resistencia y lo volví a conectar a PA6 y a tierra y confirmé que también encendía el LED.
- Sería sencillo habilitar aún más salidas utilizando el mismo procedimiento.
- Sólo tenga en cuenta que necesitará habilitar otro banco de puertos GPIO si desea utilizar todos los pines disponibles en el puerto GPIOA.

Push-Button Test Demonstration

Push-Button Test Demonstration

- Esta demostración le mostrará cómo habilitar una entrada de pin GPIO para detectar cuando un usuario presiona el botón azul de usuario. Un LED se iluminará mientras se presione el botón.
- No se requiere hardware adicional para esta demostración aparte del necesario para la demostración LED anterior. También usaré GPIOA PA10 como pin de salida para controlar el LED.
- El método MX_GPIO_Init(void) ya configuró el botón azul del usuario, por lo que todo lo que se necesita es leer el estado del botón en el método main(). Esto se logra usando la siguiente declaración:

```
HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13)
```

Push-Button Test Demonstration

- También debe recordar de la discusión del Capítulo 4 que el botón azul del usuario está conectado al puerto GPIO C, pin 13 y también se levanta constantemente.
- Esto significa que el estado del botón será alto cuando no se presione y pasará a un valor bajo cuando se presione. El fragmento de C correspondiente debe responder continuamente a este evento de la siguiente manera:

```
while(1)
{
  if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin)
  {
     HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_
PIN_RESET);
  }
  else
  {
     HAL_GPIO_WritePin(LD2_GPIO_Port, GPIO_PIN_10, GPIO_
PIN_SET);
  }
}
```

- Ingrese el segmento de código anterior en el método main() para continuar con la demostración. Luego compile y cargue el archivo hexadecimal en la placa Nucleo-64.

Push-Button Test Demonstration

- Test Run
  - Observé que el LED se encendía mientras se presionaba el botón.
  - Tenga en cuenta que tendrá que utilizar un destornillador pequeño para presionar el botón, ya que está oculto debajo del protoboard Arduino.
  - La siguiente demostración se refiere a una propiedad básica relacionada con todas las operaciones de señales de MCU, incluidos los pines GPIO.

Clock Speed Demonstration

Clock Speed Demonstration

- La velocidad del reloj en relación con las operaciones de la señal de la MCU es un nombre inapropiado.
- Normalmente se consideraría una medida del número máximo de transiciones de señal por segundo.
- Sin embargo, en realidad es una medida del desfase del reloj o la cantidad de tiempo que lleva hacer la transición de una señal del 10% al 90% del valor máximo.
- También se aplica en sentido inverso o pasando del 90% al 10% del valor máximo.
- El primero se conoce como leading edge delay y el segundo se llama trailing edge delay.

Clock Speed Demonstration

- La desviación del reloj está directamente relacionada con la velocidad real del reloj porque los retrasos del borde anterior y posterior determinan en última instancia el número máximo posible de transiciones de señal por segundo.
- Todavía me referiré a la desviación del reloj como velocidad del reloj porque así es como STM elige etiquetar esta importante propiedad de la señal.
- La pregunta importante que uno debería hacerse es por qué debería siquiera considerarse la velocidad del reloj.
- La respuesta está en ampliar la vida útil tanto del MCU como de la batería.
- La física detrás de la mejora de la velocidad del reloj dicta que se necesita energía o potencia, en este caso, para superar los efectos de capacitancia del circuito que determinan principalmente la velocidad del reloj.

Clock Speed Demonstration

- Se requieren pequeñas cantidades de energía adicional para mejorar los tiempos de conmutación de los circuitos.
- Esto implica un aumento en el flujo de corriente de la batería con el tiempo y, en consecuencia, una vida útil más corta.
- Además, una mayor disipación de energía inducirá pequeñas cantidades de estrés térmico adicional en la MCU y también acortará su vida útil.
- Las mencionadas son las razones principales por las que la velocidad del reloj siempre debe seleccionarse en el nivel más bajo pero aún así admitir la funcionalidad requerida.
- Esta demostración ilustrará los tiempos de velocidad de reloj reales asociados con las configuraciones de velocidad de reloj máxima y mínima.

Clock Speed Demonstration

- Hay cuatro configuraciones de velocidad disponibles con el marco HAL, que son las siguientes:
  - Bajo
  - Medio
  - Rápido
  - Muy rápido
- Usaré las configuraciones Baja y Muy Rápida y las aplicaré a las salidas de señal de los pines PC0 y PC1, respectivamente.
- No se requiere hardware MCU adicional.

Clock Speed Demonstration

- Sin embargo, necesitará un osciloscopio multicanal rápido para replicar esta demostración.
- El osciloscopio también debe tener un ancho de banda mínimo de al menos 100 MHz.
- Utilicé un osciloscopio USB Picoscope, modelo 3406B, para capturar las medidas.
- Este dispositivo de prueba funciona con una PC para medir y mostrar hasta cuatro canales independientes.

Configuración de las velocidades del reloj PIN

- Las velocidades de reloj de los pines PC0 y PC1 se configuran creando dos estructuras de datos GPIO_InitStruct C más y colocándolas en el método MX_GPIO_Init(void).
- Además, se requerirá una pequeña cantidad de código en el método main().

Clock Speed Demonstration

- Las dos nuevas estructuras de datos GPIO_InitStruct C se enumeran a continuación:
- Las nuevas estructuras C configuran los pines como salidas GPIO idénticas con la excepción de que una es de baja velocidad y la otra es de muy rápida velocidad.
- El segmento de código en main() simplemente alterna las salidas del pin entre alta y baja a la velocidad más rápida posible porque no hay ningún otro código intermedio entre las declaraciones de salida.

```
// Configure GPIO pin PC0
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
// Configure GPIO pin PC1
GPIO_InitStruct.Pin = GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

The new code statements to be placed in the `main()` method are as follows:

```
while (1) {
    GPIOC->ODR = 0x3;
    GPIOC->ODR = 0x0;
}
```

Clock Speed Demonstration

Test Results

- La Figura 5-13 es una captura de pantalla que muestra una captura instantánea de las formas de onda de la señal del pin GPIO.
- Además, se requerirá una pequeña cantidad de código en el método main().

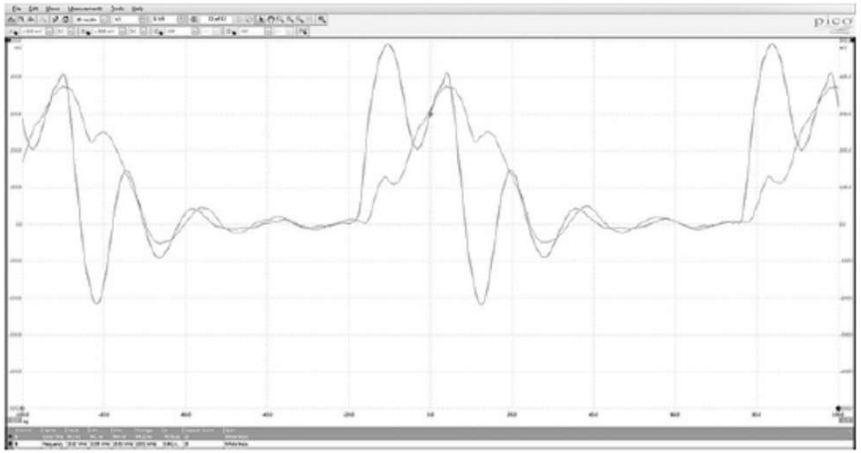


Figure 5-13 *PC0 and PC1 pins' signal waveforms.*

Clock Speed Demonstration

- Es fácil ver que las formas de onda de la señal no son señales digitales nítidas con transiciones bruscas, sino que están ligeramente distorsionadas con pendientes definidas de borde ascendente y descendente.
- Es probable que las distorsiones se deban a la alta carga capacitiva de la sonda del osciloscopio y a la ligera inductancia debida a los cables de puente que conecté entre las sondas del osciloscopio y los zócalos del protoboard Arduino.
- También utilicé configuraciones de sonda de alta impedancia 10× para ayudar a reducir la carga en los pines GPIO.
- Sé que esto ayudó porque repetí las mediciones usando la configuración directa de la sonda X1, lo que distorsionó aún más la configuración.
- La escala de amplitud de voltaje vertical debe multiplicarse por 10 para tener en cuenta la configuración de la sonda 10×.

Clock Speed Demonstration

- Si examina cuidadosamente la figura, debería ver dos mediciones automáticas que fueron calculadas por el osciloscopio en las formas de onda mostradas.
- He repetido estas medidas a continuación para una fácil visualización:
  - Período de pulso: 94,1 ns
  - Frecuencia: 10,62MHz
- También examiné de cerca las formas de onda en la Figura 5-13 para estimar los parámetros que utilicé para calcular el ancho de banda equivalente necesario que refleja los bordes de ataque de las formas de onda medidas.
- También utilicé una regla general para estimar el ancho de banda equivalente utilizando el tiempo de subida del borde de ataque.

Clock Speed Demonstration

- Estos cálculos siguen:

Low speed:

Leading edge rise time for 10% to 90% of peak = 18 ns

BW (GHz) = 0.35/rise time (nS) = 0.35/18 = 0.019.4

BW (MHz) = 19.4

Very Fast speed:

Leading edge rise time for 10% to 90% of peak = 5 ns

BW (GHz) = 0.35/rise time (nS) = 0.35/5 = 0.070

BW (MHz) = 70.0

Clock Speed Demonstration

- El ancho de banda de 70 MHz calculado para la configuración de velocidad Muy rápida refleja varios factores, incluida la velocidad máxima posible del bus AHB1, que es de 100 MHz, y la velocidad de reloj de alta velocidad STM32F302R8, que es de 72 MHz para la placa de desarrollo Nucleo-64 que controla.
- En general, un ancho de banda de 70 MHz es bastante decente, mientras que los 19,4 MHz para la configuración de baja velocidad podrían resultar problemáticos para aplicaciones exigentes como las comunicaciones de datos de muy alta velocidad.
- En realidad, mis resultados experimentales concuerdan bastante con las estimaciones de STM de 75 MHz para la configuración Muy rápida y 25 MHz para la configuración Baja.

Summary

Summary

- El enfoque de este capítulo fue cómo trabajar con pines GPIO y HAL. Comencé con una revisión de cómo se asignan la memoria a los periféricos STM porque esa es la base fundamental de cómo los configura y controla la MCU.
- A continuación, procedí a analizar cómo se configura un pin de hardware GPIO típico y las diversas funciones mediante las cuales se puede configurar, incluidas las de entrada, salida, función analógica o función alternativa. También se discutieron los diversos registros de control asociados con un pin GPIO.
- A continuación se mostró una demostración de prueba de LED simple en la que utilicé una combinación de LED/resistencia, que se insertó en los zócalos del protoboard Arduino. El protoboard, a su vez, se insertó en la placa de desarrollo Nucleo-64. Revisé todos los cambios de programa necesarios para ingresarlos en un archivo main.c para encender el LED. Se proporcionó un listado completo y anotado de main.c para su referencia y uso.

Summary

- El programa anterior se modificó a continuación para acomodar múltiples salidas, que controlaban un LED adicional.
- La siguiente demostración ilustró cómo el botón azul de usuario ubicado en la placa de desarrollo Nucleo-64 podría programarse con un pin de entrada GPIO para encender un LED cuando se presiona. El propósito de este programa era mostrarle cómo configurar un pin GPIO como entrada.
- La última demostración se centró en cómo el parámetro de velocidad del reloj GPIO afectaba las formas de onda de salida de la señal GPIO. Le expliqué que la velocidad del reloj era un nombre inapropiado y que la verdadera intención era controlar la desviación de la señal GPIO. El sesgo es el tiempo que lleva pasar de un estado a otro.

Referencias

- Programming with STM32: Getting Started with the Nucleo Board and C/C++ 1st Edición - Donal Norris (Author)
- Nucleo Boards Programming with the STM32CubeIDE, Hands-on in more than 50 projects - Dogan Ibrahim (Author)
- STM32 Arm Programming for Embedded Systems, Using C Language with STM32 Nucleo - Muhammad Ali Mazidi (Author), Shujen Chen (Author), Eshragh Ghaemi (Author)

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la práctica propia de la carrera más que en el desarrollo teórico disciplinar, con eje en la participación de las y los estudiantes


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar