Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Cuautitlán

Ingeniería en Telecomunicaciones, Sistemas y Electrónica

Manual del Laboratorio de Microprocesadores

Departamento de Ingeniería
Sección: Electrónica

Clave Asignatura: 0586
Clave Carrera: 130

Mendoza Andrade, Nidia
nidia.mendoza@cuautitlan.unam.mx

Gersenowies Rosas, Jorge Ricardo
jorge_gersenowies@comunidad.unam.mx

Ahumada Sierra, Santiago
Servicio Social SS-2025-12/147-979

Fecha de Elaboración: 15 de agosto de 2025

Objetivo general de la asignatura

Al finalizar el curso, el alumno será capaz de comprender la estructura y el funcionamiento de los microprocesadores y de los circuitos de apoyo necesarios para la creación de un sistema de cómputo digital y podrá aplicarlos en la solución de problemas de ingeniería.

Objetivos del curso experimental

- Comprender el funcionamiento y aplicaciones de un sistema digital basado en un microprocesador.
- Reconocer la función de cada componente que completa el sistema: Elementos internos del microprocesador (reloj, registros, ALU, buses de datos), memoria de solo lectura, memoria de acceso aleatorio estática, puertos de entrada y salida de propósito general.
- Conocer distintos protocolos de comunicación (paralela y serial), identificar sus diferencias y diseñar implementaciones.
- Diseñar soluciones que implementen sistemas embebidos para problemas de ingeniería.

Introducción

Un microprocesador (o procesador) es esencialmente una CPU (*Central Processing Unit*). La unidad central de procesamiento es el componente funcional principal de una computadora; está compuesta por la unidad de control, registros de propósito general y específico, canales de intercomunicación (*buses*) y una unidad encargada de las operaciones aritméticas y lógicas (*ALU*).

Desde un punto de vista didáctico, el diseño de sistemas basados en microprocesadores puede ser abordado utilizando microcontroladores, pues internamente un microcontrolador cuenta con una Unidad Central de Procesamiento (CPU o microprocesador). A su vez, de esta manera se puede acceder a herramientas hardware y software de bajo costo, indispensables en la etapa iterativa del diseño.

La arquitectura de un microprocesador se refiere a diversas metodologías de diseño y organización de los componentes de la CPU. Otro factor importante es la arquitectura del conjunto de instrucciones, ISA, del inglés (*Instruction Set Architecture*) que el procesador puede realizar. Las arquitecturas de computadora de conjunto de instrucciones reducidas (*RISC*) del inglés (*Reduced Instruction Set Computer*) y computadora de conjunto de instrucciones complejas (*CISC*), del inglés *Complex Instruction Set Computer* proporcionan varios métodos para el procesamiento de datos, ofreciendo diferentes niveles de rendimiento, confiabilidad y velocidad dependiendo de la aplicación.

Los microprocesadores varían en potencia, rendimiento, metodologías de arquitectura, tamaño, consumo de energía y muchas otras variables. Los microprocesadores de uso general son habituales en computadoras personales y dispositivos móviles, mientras que las unidades especializadas de alto rendimiento se diseñan para tareas exigentes. Los siguientes son algunos de los principales tipos de microprocesadores:

- Microprocesadores de uso general,
- Microcontroladores,
- Procesadores de señales digitales (DSP),
- Unidades de procesamiento de gráficos (GPU),

- Procesadores de red,
- Coprocesadores.

Las dos empresas más importantes en este campo son Intel y AMD (*Advanced Micro Devices*). Cada una utiliza un tipo diferente de arquitectura de conjunto de instrucciones (ISA). Los procesadores Intel utilizan una arquitectura de conjunto de instrucciones complejo (*CISC*), mientras que los procesadores AMD siguen una arquitectura de conjunto de instrucciones reducido (*RISC*).[8]

Otro jugador importante es **arm** (*Advanced RISC Machine*). Aunque no fabrica equipos, sí alquila sus diseños de procesadores de alta gama y otras tecnologías patentadas a otras empresas que sí fabrican equipos. Apple, por ejemplo, ya no utiliza chips Intel en sus computadoras, sino que fabrica sus propios procesadores personalizados basados en diseños **arm**.

Es una familia de arquitecturas de conjunto de instrucciones (*ISA*) *RISC* para procesadores, diseñada por Arm Holdings, ampliamente utilizada por su arquitectura eficiente. Tiene cuatro líneas de unidades centrales de procesamiento:

- Cortex (procesadores de 32/64-bit),
- Neoverse (procesadores de 64-bit para cómputo de alto rendimiento),
- GPUs (procesadores en tiempo real),
- NPUs (Microcontroladores).

Existen muchas revisiones de arquitectura **arm** diferentes (ARMv6, ATMv6-M, ARMv7-M, ARMv7-A, ARMv8-M, etc.) y muchas arquitecturas de núcleo (Cortex-M3, Cortex-M1, Cortex-M0, Cortex-M4,Cortex-M0+, etc.). La arquitectura sigue evolucionando de tal manera que, por ejemplo, la versión siete (ARMv7) define tres perfiles de arquitectura:

- ARM Cortex serie Ax.
- ARM Cortex serie Rx.
- ARM Cortex serie Mx,

Distintas empresas son titulares de licencias ARM, entre las más conocidas están: Samsung, Apple Inc., Microchip, Broadcom, STMicroelectronics, LG, MediaTek, Qualcomm, Texas Instruments. Las empresas fabrican chips de acuerdo con alguna revisión y núcleo de arquitectura. Por ejemplo, STMicroelectronics [7] tiene una línea de procesadores Cortex-M para aplicaciones de alto desempeño, los más usados, de bajo consumo de energía, aplicaciones inalámbricas y otros (figura 0.1).

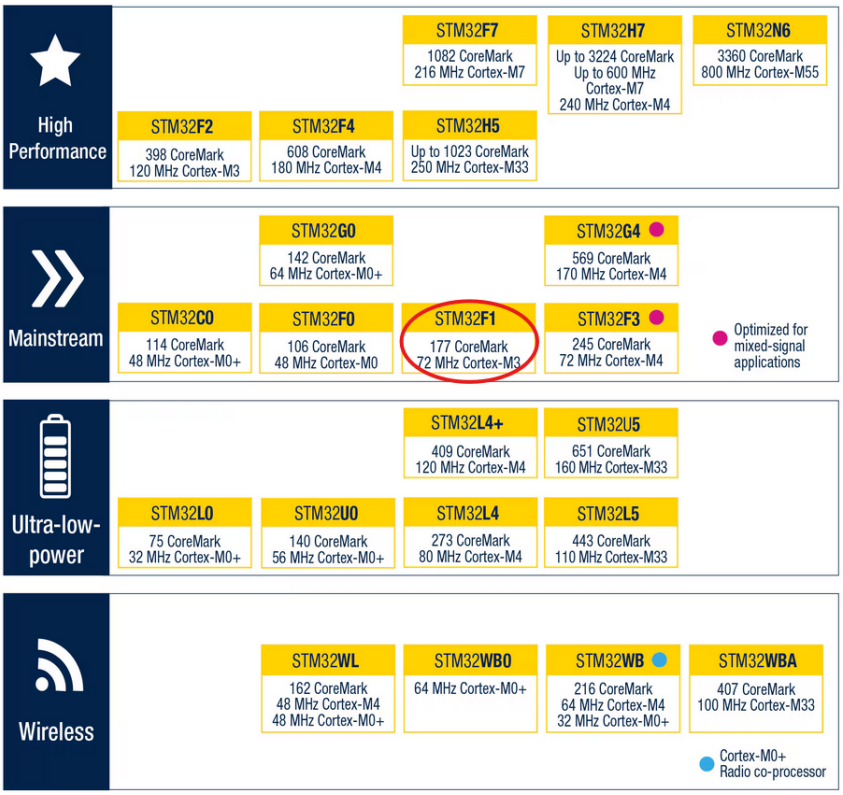

Figura 0.1: Procesadores Arm Cortex-M de STMicroelectronics.

Considerando el costo, el tamaño, las características y la disponibilidad, para este manual de prácticas se ha elegido el **STM32F103C8**[1], que es un procesador basado en el núcleo Cortex-M3 diseñado sobre la arquitectura ARMv7-M y una arquitectura clásica Harvard.

Criterios de Evaluación

No. Criterio	Criterio de Evaluación	Porcentaje
C1	Actividades previas a la práctica.	20
C2	Habilidad en el armado y funcionalidad de los sistemas.	30
C3	Habilidad para el manejo del equipo e interpretación correcta de las lecturas.	20
C4	Reporte entregado con todos los puntos anteriores.	30

Cuadro 1: Criterios de evaluación.

[1] Debido a que existen clones, verificar que el chip tenga el logo de ST .

UNAM
FESC

Laboratorio de: _____
Grupo: _____ No. de Práctica: _____
Nombre de la Práctica: _____
Profesor: _____
Alumno: _____
Fecha de realización: _____ Fecha de entrega: _____
Semestre: _____

Figura 0.2: Datos para la portada.

Para la entrega en cada reporte de práctica es necesario incluir una portada con la siguiente información:

Instrucciones para la elaboración del reporte

El reporte escrito deberá cumplir con los siguientes requisitos:

- Portada con los datos anteriormente indicados.
- Introducción (con información investigada por el alumno y debidamente citada).
- Materiales y Equipo.
- Procedimiento experimental redactado por el alumno.
- Fotografías del circuito armado (si aplica según la práctica).
- Elementos gráficos de apoyo (Tablas de datos, gráficas, fotografías, imágenes del osciloscopio,etc...).
- Observaciones y comentarios sobre el desarrollo experimental.
- Análisis de los resultados obtenidos.
- Cuestionario resuelto, si es necesario debe citarse la información obtenida.
- Bibliografía (formato IEEE).

Atributos de Egreso

ATRIBUTO POR EVALUAR: AEITSE 5. Diseña, desarrolla e implementa software y hardware.
La evaluación del atributo de egreso se llevará a cabo durante la Práctica No. 7 “Manejo de Interrupciones”, mediante una lista de cotejo que será aplicada en la sesión correspondiente. Está evaluación permitirá valorar el desempeño del estudiante en relación con los atributos de egreso establecidos en el perfil del ingeniero y será de su conocimiento por el profesor y publicada en el listado único de calificación.

Referencias

[1] Cortex-m3 devices generic user guide.

[2] Getting started with GPIO - stm32mcu.

[3] I2c quick guide.

[4] Interrupts.

[5] Nonvolatile memory technologies with emphasis on flash: a comprehensive guide to understanding and using NVM devices.

[6] Serial vs parallel communication in microprocessor. Section: Electronics Engineering.

[7] STM32 microcontrollers (MCUs) - STMicroelectronics.

[8] Types of central processing unit (CPU) \textbackslashtextbackslashtextbar IBM.

[9] Ben Eater. Reliable data transmission.

[10] Chris Coleman. A practical guide to ARM cortex-m exception handling.

[11] Dawoud Shenouda Dawoud and Peter Dawoud. *Serial communication protocols and standards: RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX*. River Publishers.

[12] Randall Hyde. *The book of I²C: a guide for adventurers*. No Starch Press.

[13] Larry D. Pyeatt Ph.D. *Modern Assembly Language Programming with the ARM Processor*. Newnes, 2nd edition edition.

[14] Ronald J. Tocci. *Digital systems: principles and applications*. Prentice-Hall, 3. ed edition.

[15] Neal S. Widmer, Gregory L. Moss, and Ronald J. Tocci. *Digital systems: principles and applications*. Pearson, twelfth edition edition.

[16] Joseph Wu. A basic guide to i2c.

	UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO **FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN** **DEPARTAMENTO DE INGENIERÍA** **SECCIÓN ELECTRÓNICA**
	REGLAMENTO INTERNO DE LABORATORIOS

El presente reglamento de la sección electrónica tiene por objetivo establecer los lineamientos para el uso y seguridad de laboratorios, condiciones de operación y evaluación, que deberán de conocer y aplicar, estudiantes y profesores en sus cuatro áreas: comunicaciones, control, sistemas analógicos y sistemas digitales.

1. Queda estrictamente prohibido, al interior de los laboratorios

   a) Correr, jugar, gritar o hacer cualquier otra clase de desorden.
   b) Dejar basura en las mesas de trabajo y/o pisos.
   c) Fumar, consumir alimentos y/o bebidas.
   d) Realizar o responder llamadas telefónicas y/o el envío de cualquier tipo de mensajería.
   e) La presencia de personas ajenas en los horarios de laboratorio.
   f) Dejar los bancos en desorden y/o sobre las mesas.
   g) Mover equipos o quitar accesorios de una mesa de trabajo.
   h) Usar o manipular el equipo sin la autorización del profesor.
   i) Rayar y/o sentarse en las mesas del laboratorio.
   j) Energizar algún circuito sin antes verificar que las conexiones sean las correctas (polaridad de las fuentes de voltaje, multímetros, etc.).
   k) Hacer cambios en las conexiones o desconectar el equipo estando energizado.
   l) Hacer trabajos pesados (taladrar, martillar, etc.) en las mesas de trabajo.
   m) Instalar software y/o guardar información en los equipos de cómputo de los laboratorios.
   n) El uso de cualquier aparato o dispositivo electrónico ajeno al propósito para la realización de la práctica.
   o) Impartir clases teóricas, su uso es exclusivo para las sesiones de laboratorio.

2. Es responsabilidad del profesor y de los estudiantes revisar las condiciones del equipo e instalaciones del laboratorio al inicio de cada práctica (encendido, dañado, sin funcionar, maltratado, etc.). El profesor deberá generar el reporte de fallas de equipo o de cualquier anomalía y entregarlo al responsable de laboratorio o al jefe de sección.

3. Los profesores deberán de cumplir con las actividades y tiempos indicados en el "cronograma de actividades de laboratorio".

4. Es requisito indispensable para la realización de las prácticas que el estudiante:

   a) Descargue el manual completo y actualizado al semestre en curso, el cual podrá obtener en (https://virtual.cuautitlan.unam.mx/ingenieriafesc/)
   b) Presente su circuito armado en la tableta de conexiones para poder realizar la práctica (cuando aplique), de no ser así, tendrá una evaluación de cero en la sesión correspondiente.
   c) Realizar las actividades previas y entregarlas antes del inicio de la sesión de práctica, de no ser así, tendrá una evaluación de cero en la sesión correspondiente.

5. Estudiante que no asista a la sesión de práctica de laboratorio será evaluado con cero.

6. La evaluación de cada sesión debe realizarse con base en los criterios de evaluación incluidos en los manuales de prácticas de laboratorio y no podrán ser modificados. En caso contrario, el estudiante deberá reportarlo al jefe de sección.

7. La evaluación final del estudiante en los laboratorios será con base en lo siguiente:

   a) **(Aprobado) Cuando el promedio total de todas las prácticas de laboratorio sea mayor o igual a 6 siempre y cuando tengan el 90% de asistencia y el 80% de prácticas acreditadas con base en los criterios de evaluación.**
   b) **(No Aprobado) No cumplió con los requisitos mínimos establecidos en el punto anterior.**
   c) **(No Presentó) Cuando no asistió a ninguna sesión de laboratorio o que no haya entregado actividades previas o reporte alguno.**

8. Profesores que requieran hacer uso de las instalaciones de laboratorio para realizar trabajos o proyectos, es requisito indispensable que las soliciten por escrito al jefe de sección. Siempre y cuando no interfiera con los horarios de los laboratorios.

9. Estudiantes que requieran realizar trabajos o proyectos en las instalaciones de los laboratorios, es requisito indispensable que esté presente el profesor responsable del trabajo o proyecto. En caso contrario no podrán hacer uso de las instalaciones.

10. Correo electrónico del buzón para quejas y sugerencias para cualquier asunto relacionado con los laboratorios (seccion_electronica@cuautitlan.unam.mx).

11. El incumplimiento a estas disposiciones faculta al profesor para que instruya la salida del infractor y en caso de resistencia, la suspensión de la práctica.

12. A los usuarios que, por su negligencia o descuido inexcusable, cause daños al laboratorio, materiales o equipo deberá cubrir los gastos que se generen con motivo de la reparación o reposición, indicándose en el reporte de fallas correspondiente.

13. Los usuarios de laboratorio que sean sorprendidos haciendo uso indebido de equipos, materiales, instalaciones y demás implementos, serán sancionados conforme a la legislación universitaria que le corresponda, según la gravedad de la falta cometida.

14. Los casos no previstos en el presente reglamento serán resueltos por el Jefe de Sección, de acuerdo con los lineamientos generales para el uso de los laboratorios en la Universidad Nacional Autónoma de México.

SECCIÓN ELECTRÓNICA
“POR MI RAZA HABLARÁ EL ESPÍRITU”
Cuautitlán Izcalli, Estado de Méx. a 16 de junio de 2025

Índice

Índice de figuras

1. Práctica: Entorno de desarrollo

1.1. Objetivos

- Instalar el entorno de desarrollo que nos permita escribir código, depurar programas y programar el procesador; tanto en lenguajes ensamblador como en C y C++.
- Escribir un programa en ensamblador para probar el entorno.
- Probar el programa, mediante la simulación del procesador.

1.2. Introducción

Además de la elección del hardware para la creación de sistemas integrados, se requiere del proceso de desarrollo de firmware[2], que implica una serie de pasos, herramientas y tecnologías para desarrollar productos desde su inicio hasta la producción. Tener una cadena de herramientas de software tiene como objetivo conectar y optimizar cada una, de modo que la salida que produce una herramienta sirva como entrada para la siguiente herramienta. Las cadenas de herramientas básicas suelen incluir: ensamblador, enlazador, depurador, compilador, bibliotecas. Estas herramientas pueden estar incluidas en un solo entorno de desarrollo o ser ejecutadas de manera individual por el desarrollador. Si bien hay una variedad de entornos de desarrollo integrados (*IDE*) disponibles para STM32, los más utilizados son STM32CubeIDE de STMicroelectronics, EWARM de IAR y μVision (*MDK*) de Keil. Este último será utilizado durante el desarrollo para estas prácticas.

1.3. Actividades previas

1. Leer la explicación introductoria al contenido y alcance de esta práctica.
2. Leer detenidamente y comprender cada una de las secciones de esta práctica.
3. Instalar el entorno de desarrollo siguiendo las indicaciones descritas en este enlace. Y, una vez completada la instalación, activar la licencia siguiendo estas otras indicaciones.

1.4. Material

- No se requiere.

1.5. Equipo

- Computadora portátil o de escritorio, con sistema operativo Windows 8 (o mayor).

1.6. Procedimiento experimental

1.6.1. Crear el proyecto

1. Abrir el entorno de desarrollo Keil μVision.
2. En la barra de herramientas, seleccionar *Project* y luego *μVision Project*. Crear la carpeta, escribir el nombre del proyecto y salvar (figura 1.3).
3. A continuación, aparece otra ventana para localizar el dispositivo con el que se trabajará, seleccionar **stm32f103c8** para esta práctica (ver figura 1.4). Luego oprimir el botón *OK*.

[2] Es el software que tiene directa interacción con el hardware, que reside en la memoria no volátil.

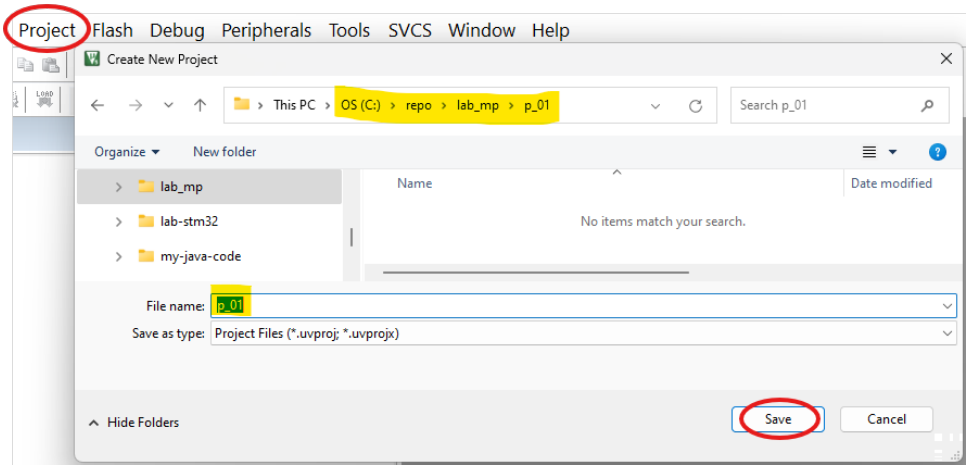

Figura 1.3: Crear carpeta y nombre para el proyecto.

4. Aparece una nueva ventana para seleccionar las interfaces estándar de software para el procesador (*CMSIS* por sus siglas en inglés). Seleccionar las básicas, como indica la figura 1.5 y oprimir *OK*.

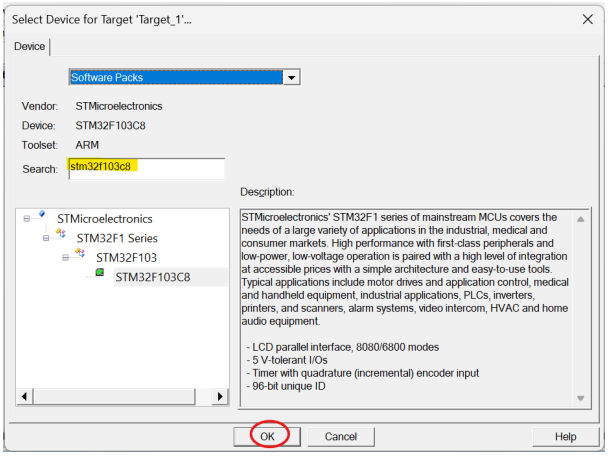

Figura 1.4: Seleccionar el procesador.

Al incluir las interfaces de software (*CMSIS*), se añade código que se utiliza para inicializar (*startup*) el procesador (establecer una pila para las operaciones push/pop, establecer un vector de excepciones, relojes y demás) e inicializar el entorno de ejecución de Ç". Aun si se utiliza lenguaje ensamblador, el llamado a estos programas se hace a través de la función *main*.

1.6.2. Crear un archivo de código

1. En la sección del proyecto (ver figura 1.6), seleccionar *Source Group 1*, botón derecho y *Add New Item*.

2. Aparece una nueva ventana (figura 1.7) para seleccionar el tipo de archivo fuente (ensamblador para esta práctica), la carpeta donde se alojará y asignar el nombre. Posteriormente, oprimir añadir (*Add*).

3. En la figura 1.8, se muestra un código escrito en lenguaje ensamblador. Analícelo y compréndalo; para luego escribirlo en el archivo recién creado. Tome en cuenta las sangrías

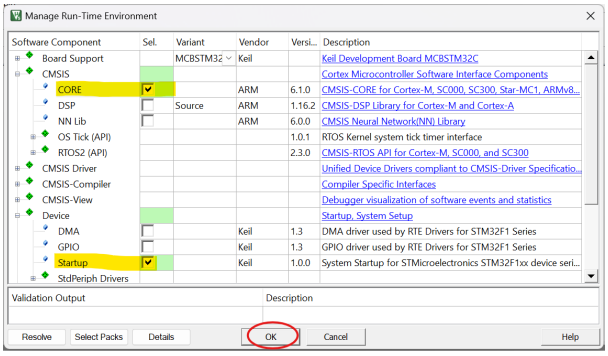

Figura 1.5: Incluir interfaces de software.

después de la columna 1 (también llamadas tabulación o indentación).

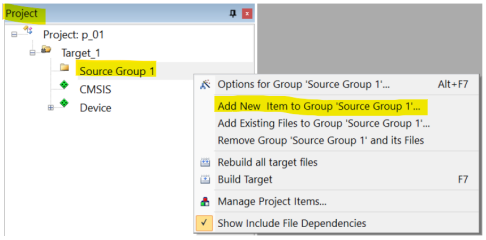

Figura 1.6: Añadir un nuevo archivo.

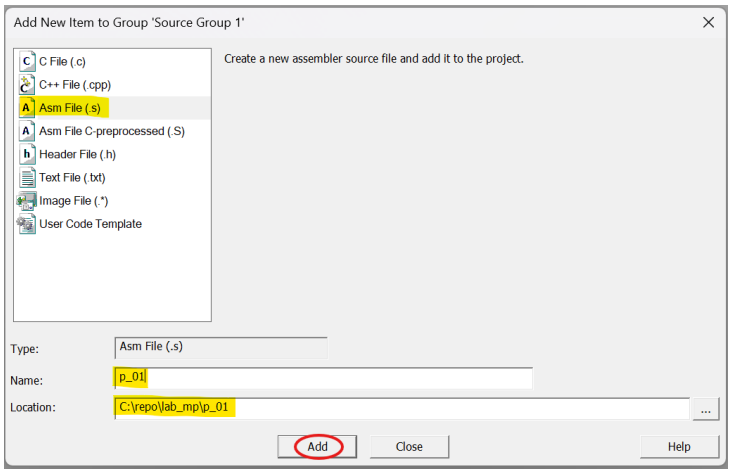

Figura 1.7: Crear el archivo fuente.

p_01.s

```
;directivas de ensamblador
;-------------------------
            area constantes, data, readonly     ;area de código de solo lectura
RCC_APB2ENR equ 0x40021018                      ;registro reloj de puertos
GPIOC_CRL   equ 0x40011000                      ;registro para configuración de puerto C (parte baja)
GPIOC_CRH   equ 0x40011004                      ;registro para configuración de puerto C (parte alta)
GPIOC_ODR   equ 0x4001100C

;area de programa
            area p_01, code, readonly   ; area de código de solo lectura
            export __main               ; se exporta a startup_stm32f10x_md.s
;inicio del programa
;-------------------
__main
    ldr r0, =0x00000010     ;r0 valor para habilitar reloj de puerto C (bit4=1)
    ldr r1, =RCC_APB2ENR    ;r1 apunta al registro de reloj de puertos
    str r0, [r1]            ;almacena '1' en el bit 4 del registro

    ldr r0,=0x44444444      ;r0 valor para reset
    ldr r1,=GPIOC_CRL
    str r0,[r1]             ;reset al puerto

    ldr r0,=0x44344444      ;r0 valor para reset
    ldr r1,=GPIOC_CRH
    str r0,[r1]             ;reset al puerto
ciclo
    ldr r0,=0x00002000      ;r0 valor para reset
    ldr r1,=GPIOC_ODR
    str r0,[r1]             ;reset al puerto

    ldr r0,=0x00000000      ;r0 valor para reset
    ldr r1,=GPIOC_ODR
    str r0,[r1]             ;reset al puerto

    b ciclo     ; ciclo infinito
;fin del programa
;----------------
    end
```

Figura 1.8: Código en lenguaje ensamblador.

1.6.3. Configurar el simulador

1. Seleccionar el icono (*Options for Target*, ver figura 1.9) para configurar la herramienta de depuración.

2. Aparece una nueva ventana. En esta, seleccionar la pestaña *Debug* (1.10) y activar la opción para usar el simulador (*Use Simulator*). Además, si se desea una vista estilizada de las ventanas de simulación, se pueden modificar las opciones *Dialog DLL* y *Parameter*. Oprimir *OK*.

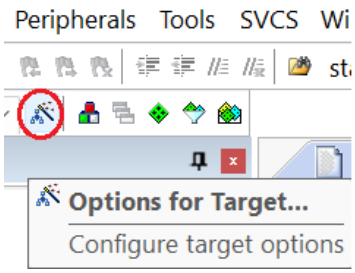

Figura 1.9: Configuración de herramientas software.

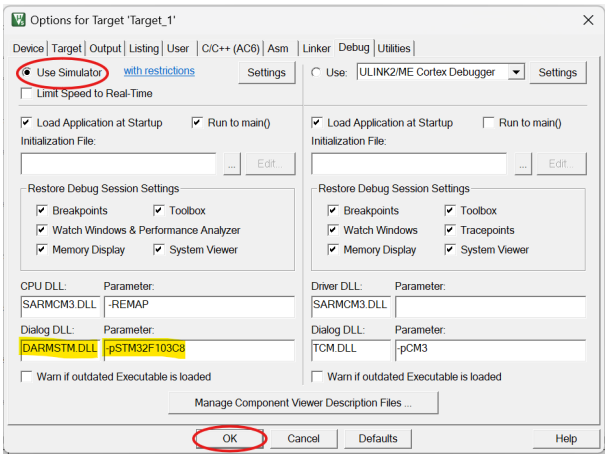

Figura 1.10: Configuración de la herramienta de depuración.

1.6.4. Ejecutar el programa

1. Seleccionar el icono (figura 1.11) para ensamblar el proyecto. Lleva a cabo las funciones del pre-procesador, compilador, ensamblador y enlace (*linker*). Verificar que no haya errores.

2. Iniciar la sesión para depurar (marcado con el número 1 en la figura 1.12).

3. Colocar un punto de prueba (marcado con el número 2 en la figura 1.12).

4. Iniciar la ejecución del programa (marcado con el número 3 en la figura 1.12). Notar que el programa se ejecuta hasta detenerse en el punto de prueba.

5. A partir del punto de prueba (*breakpoint*), se pueden utilizar otras opciones (marcadas con el número 4 en la figura 1.12) para ejecutar cada instrucción, ejecutar toda una función o salir de ella.

6. Para terminar la sesión, oprimir el icono marcado con el número 1 en la figura 1.12.

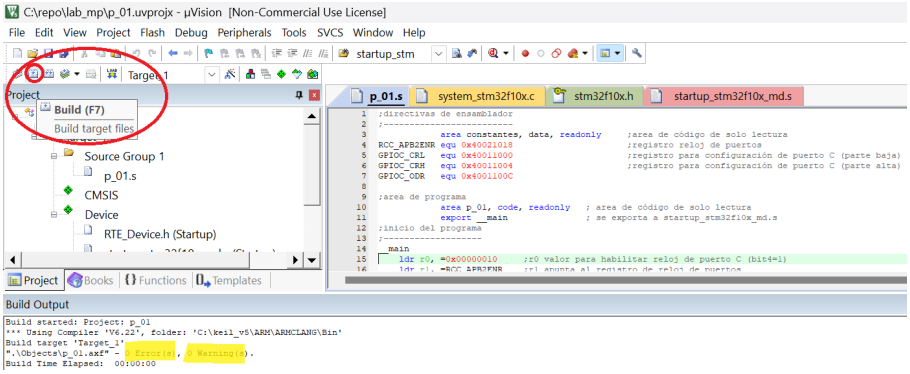

Figura 1.11: Construir el proyecto.

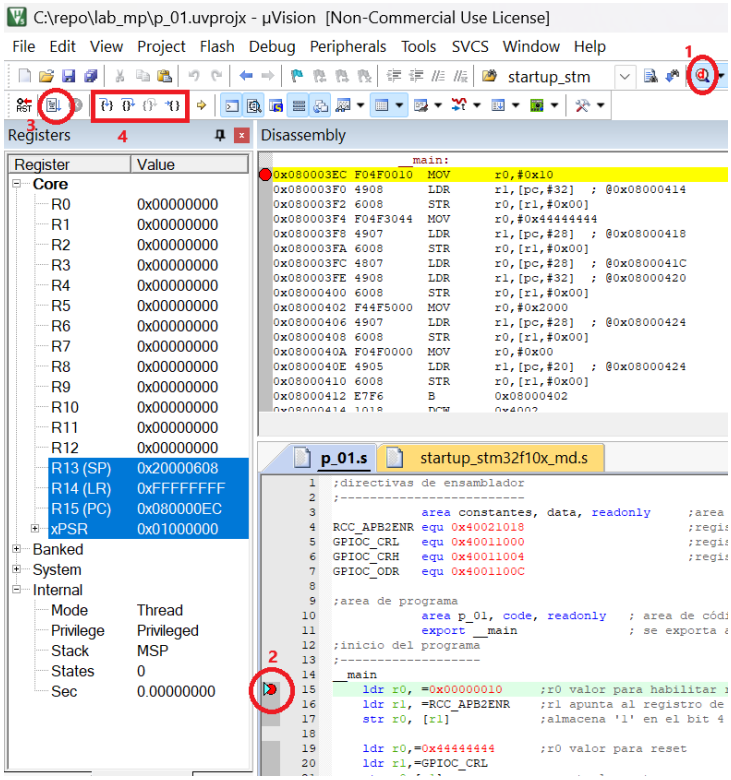

Figura 1.12: Ejecutar, simular y depurar.

1.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

1.8. Cuestionario

1. Explique sus resultados y observaciones al realizar la simulación de este programa.
2. Explique la importancia del proceso de depuración en el diseño de sistemas basados en procesadores.
3. Investigue otras herramientas, software o hardware, que nos ayudan en el proceso de diseño de sistemas basados en procesadores.

2. Práctica: Arquitectura del procesador

2.1. Objetivos

- Mediante la escritura de código en lenguaje ensamblador, identificar los elementos de la unidad de procesamiento central (*CPU*).
- Probar el programa mediante el uso de un emulador.

2.2. Introducción

La unidad central de procesamiento de un ARM es el componente funcional principal. Como se muestra en la figura 2.13, se pueden seleccionar dos registros de origen utilizando los buses A y B. Los datos del bus B se enrutan a través de un registro de desplazamiento (*shifter*) antes de llegar a la unidad aritmética-lógica (*ALU*). Los datos del bus A van directamente a la ALU, pero además los buses A y B pueden proporcionar datos para el multiplicador(*multiplier*). Los datos que vienen de la memoria (*memory*) o de los dispositivos de entrada y/o salida (*I/O*) se introducen directamente en el bus de la ALU, se pueden almacenar en uno de los registros de propósito general, se pueden procesar por la ALU o se pueden tomar directamente del bus B para escribirlos en la memoria o en los dispositivos de E/S. La CPU utiliza el registro de dirección siempre que necesita leer o escribir en la memoria o en los dispositivos de E/S, cada vez que se obtiene una instrucción y en todas las operaciones de carga y almacenamiento. El registro de dirección se puede cargar desde el contador de programa (obteniendo la siguiente instrucción), desde la ALU (en modos de direccionamiento) y se puede incrementar y almacenar nuevamente (para los modos de direccionamiento con apuntadores y contador de programa).

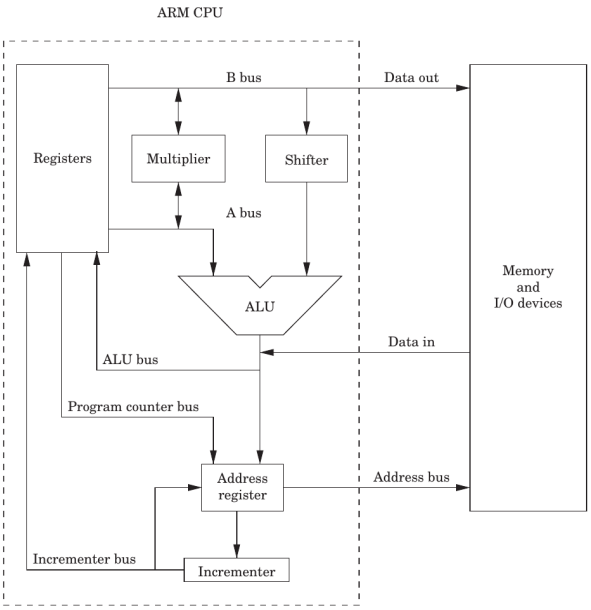

Figura 2.13: ARM CPU.[13]

El procesador cuenta con los siguientes registros de 32 bits (ver figura 2.14):

- Los registros **r0 a r12** son de propósito general. Los registros **r0 a r7** son accesibles por todas las instrucciones, mientras que los registros **r8 a r12** solo pueden ser accedidos por las instrucciones de 32 bits.
- Registros para funciones especiales:

El registro r13 es utilizado como apuntador a la pila (*Stack Pointer*).

El registro de enlace r14 (*Link Register*) recibe la dirección del contador de programa cuando se ejecutan instrucciones de bifurcación con enlace (*Branch and link*).

El Contador de programa r15 (*Program Counter*). Es utilizado por la CPU para apuntar a la dirección de la siguiente instrucción a ejecutar. El contador de programa de 32 bits puede acceder a un máximo de 4 GBytes (2^{32}) de código.

Registro de estados xPSR (*Program Status Register*). El estado del procesador a nivel del sistema se divide en tres categorías:

Aplicación, contiene el estado de banderas N (*Ngative*), Z (Zero), C (*Carry*) y V (*Overflow*).

Interrupciones, contiene el número de la rutina de atención a la interrupción (*ISR*).

Ejecución, contiene dos campos: uno para la continuidad o interrupción de la instrucción (*ICI*) y otro para la instrucción *IF-THEN*.

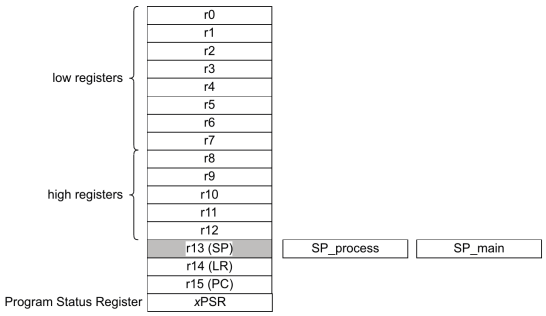

Figura 2.14: Registros del procesador.[1]

El lenguaje ensamblador es un lenguaje de bajo nivel con el que se puede acceder directamente a la estructura interna de la CPU. Las distintas formas en que se especifican los operandos en una instrucción se denominan modos de direccionamiento. En esta práctica se escribirá un programa aplicando algunos de los modos de direccionamiento clásicos como: de registro, inmediato y de registro indirecto.

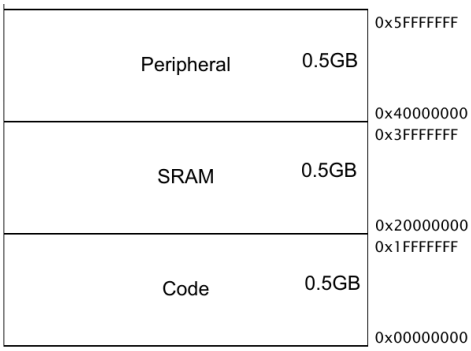

Figura 2.15: Mapa de memoria del procesador.[1]

Sin embargo, el programa que será ejecutado por la CPU debe ser alojado en una memoria externa a la CPU (ver la figura 2.13), para que de ahí vaya tomando las instrucciones que conformarán una tarea específica. Cuando se diseña un sistema de microcomputador, a cada recurso externo, como las memorias, puertos de interfaz y demás (que serán abordados en prácticas posteriores) se les asigna una dirección única (ver el mapa de memoria 2.15),

mediante la cual se accede a ese recurso en particular. El procesador **STM32F103C8** cuenta con **64Kbytes** para el código e inicia en la dirección **0x00000000**. La placa de desarrollo **STM32F103C8T6 Blue Pill** utiliza un procesador **STM32F103C8** con núcleo **Cortex-M3** sobre la arquitectura **ARMv7-M**.

2.3. Actividades previas

1. Leer detenidamente y comprender la sección introductoria a la práctica.
2. Comprender los objetivos y el procedimiento experimental.
3. Haber realizado satisfactoriamente la práctica anterior.

2.4. Material

- Placa de desarrollo STM32F103C8T6 Blue Pill.
- Programador, emulador ST-LINK V2.
- 4 cables hembra-hembra tipo Dupont.

2.5. Equipo

- Computadora portátil o de escritorio, con sistema operativo Windows 8(o mayor).

2.6. Procedimiento experimental

2.6.1. Estructura de un programa

1. Crear un nuevo proyecto, siguiendo el procedimiento experimental de la práctica anterior.
2. En el archivo para el código, escribir, con **comentarios**[3], la estructura que tendrá el programa:

```
;area de constantes y variables
;------------------------------

;area de programa
;----------------

;inicio del programa
;-------------------

;fin del programa
;----------------
```

3. Escribir las **directivas del ensamblador**[4], para definir las áreas de memoria (**area**), exportar el programa (**export**) e indicar el final del programa (**end**). En este enlace se encuentra la descripción detallada de todas las directivas soportadas por el ensamblador.

[3] En ensamblador, los comentarios inician con **";"** (signo de punto y coma), no son interpretados por la CPU. Constituyen una buena práctica de programación y son de gran ayuda para el programador.

[4] Son instrucciones para el proceso del ensamblador, no las ejecuta la CPU.

4. Escribir la **etiqueta**[5] (**main**), que indica el inicio del programa.

```
;area de constantes y variables
;-------------------------------
    area constantes, data, readonly  ;area de código de solo lectura

;area de programa
;----------------
    area p_02, code, readonly  ; area de código de solo lectura
 export __main    ; se exporta a startup_stm32f10x_md.s

;inicio del programa
;-------------------
__main

;fin del programa
;----------------
 end
```

2.6.2. Preparar el emulador y la placa de desarrollo

1. Interconectar la placa de desarrollo con el emulador, como muestra la figura 2.16.

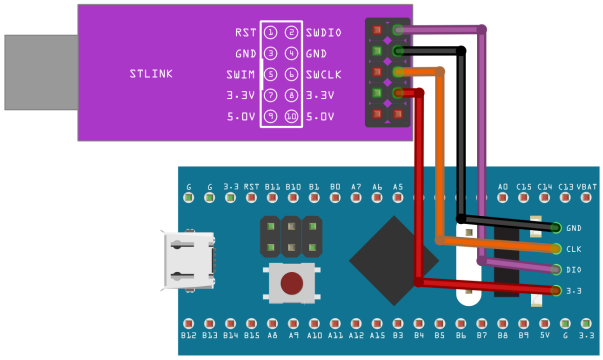

Figura 2.16: Conexión de tarjeta a emulador.

2. Conectar el emulador al puerto USB de la computadora.
3. Abrir el entorno de desarrollo Keil μVision.
4. Seleccionar *Options for Target*, luego *Debug* y configurar como indica la figura 2.17.

[5]Las etiquetas son muy útiles y versátiles en el ensamblador, dan nombre a las ubicaciones de memoria que serán descifradas posteriormente por el ensamblador o el enlazador.

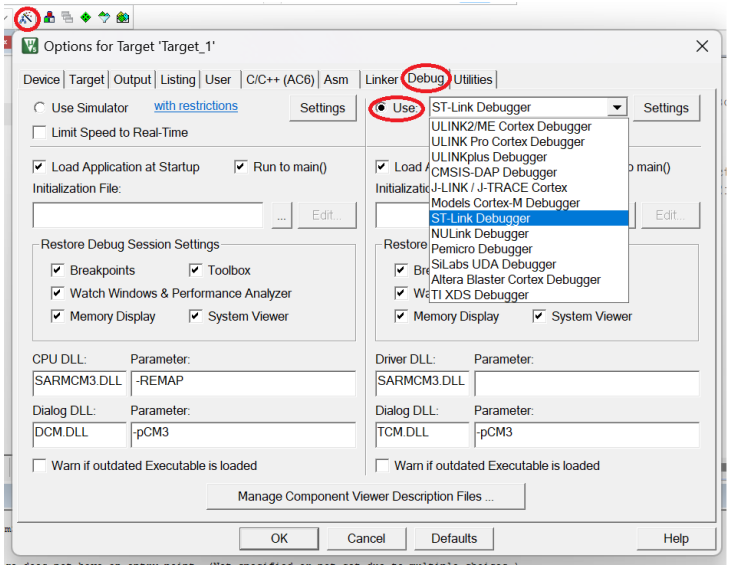

Figura 2.17: Configurar el emulador.

2.6.3. Instrucciones en ensamblador

1. Escribir las instrucciones de ensamblador usando los modos de direccionamiento inmediato, de registro y registro indirecto. Para el modo de registro indirecto, declarar una constante que contenga una dirección cualquiera (ver figura 2.15). Además, escribir las instrucciones para que el programa se ejecute y permanezca en un ciclo infinito[6].

```
;area de constantes y variables
;-------------------------
   area constantes, data, readonly  ;area de código de solo lectura
DIR_SRAM  equ 0x20000000
;area de programa
;----------------
   area p_02, code, readonly  ; area de código de solo lectura
   export __main    ; se exporta a startup_stm32f10x_md.s
;inicio del programa
;-------------------
__main
 ;direccionamiento inmediato
 mov r0,#0x69  ;mover 1 byte
 mov r1,#0x1234  ;mover 2 bytes(word)
 movw r2,#0x6655  ;otra manera de mover 2 bytes
 ldr r3,=0x12345678 ;mover 4bytes(dword)
 ;direccionamiento de registro
 mov r4,r3   ;copia el valor de r3 en r4
 ;direccionamiento de registro indirecto
 ldr r5,=DIR_SRAM ;cargar la dirección de la RAM en r5
 str r4,[r5]   ;almacenar el valor que contiene r4
      ;en la dirección a donde apunta r5
ciclo
 b ciclo    ;Ciclo infinito
```

[6]Los programas para sistemas de procesadores y microprocesadores, deberán ejecutarse siempre que esté encendido el sistema.

```
;fin del programa
;----------------
 end
```

2. Construir el proyecto (ver figura 1.11), abrir el emulador[7] e ir ejecutando las instrucciones, siguiendo el proceso descrito en la figura 1.12. Cuando utiliza por primera vez el emulador (**st-link**), aparecerá un mensaje (figura 2.18) con opción para actualizar el software. Por lo tanto, se procede a actualizarlo.

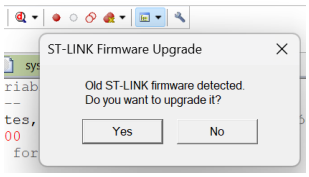

Figura 2.18: Actualizar software del emulador.

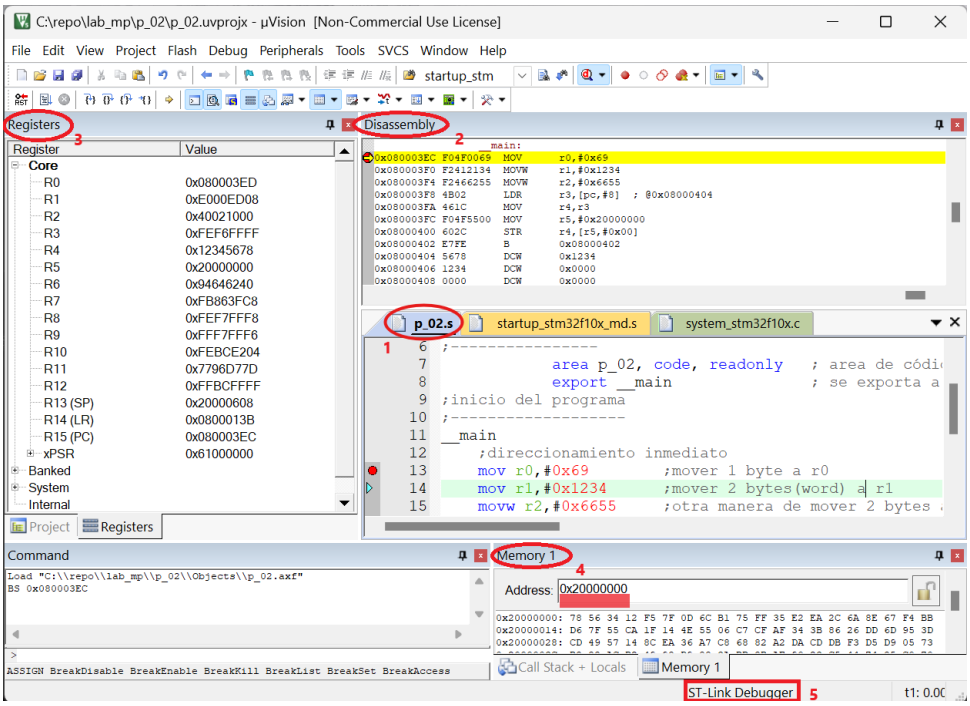

Figura 2.19: Emulador.

3. Colocar un punto de prueba al inicio del programa y ejecutar hasta ese punto (como en el procedimiento experimental de la práctica anterior). En la figura 2.19 se identifica lo siguiente:

    1. El programa escrito (código fuente).

    2. La ejecución paso a paso del programa y el área de memoria del programa.

    3. Los registros de propósito general (**r0 a r12**), el apuntador a la pila (**SP**), registro de enlace (**LR**), contador de programa (**PC**) y registro de estados del programa (**xPSR**).

    4. Ventana para mostrar el contenido de la memoria, de acuerdo con la dirección introducida.

[7] El proceso de emulación es similar a la simulación; sin embargo, la gran diferencia es que ahora se está interactuando con el procesador a través del st-link.

5. Indica que está conectado el emulador.

4. Ensamblar y ejecutar el programa en el emulador. Observe y haga sus anotaciones respecto al comportamiento de los recursos internos del procesador.

5. Agregar código para que la *ALU* realice una simple operación aritmética:

```
;registro de estados
 ldr r0,=0xFFFFFFFF ;carga el número máximo en r0
 mov r2,#0x55  ;carga un valor cualquiera en r2
 mov r3,#0x11  ;carga un valor cualquiera en r3
 adds r5,r0,#1  ;al valor de r0 suma 1 y resultado en r5
 adc r6,r2,r3  ;r6=r2+r3+C(acarreo)
```

6. Ensamblar y ejecutar en el emulador. En la figura 2.20 se observan algunas recomendaciones:
    1. Añadir un punto de prueba.
    2. Antes de ejecutar la instrucción **ADDS**[8], poner en "0"(punto 3) las banderas.

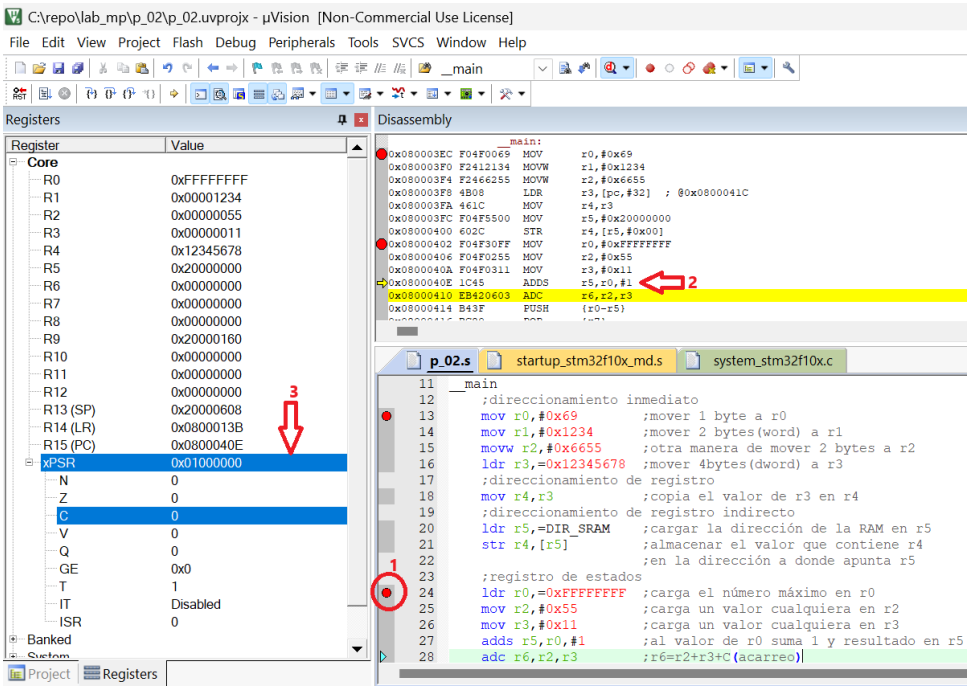

Figura 2.20: Emulador.

7. Continuar ejecutando el programa y haga sus anotaciones respecto al comportamiento de los bits del registro de estados.

8. Agregar código para observar el comportamiento de la pila y apuntador (*SP*):

```
;registro apuntador a la pila
 push {r0,r1-r5}  ;guarda el contenido de r0 y r1 a r5 en
      ;la pila de datos a partir de la dirección
      ;a donde apunta SP
 pop {r7}   ;Extrae un dato de la pila y lo deposita en r7
```

[8]La descripción detallada de esta y todas las instrucciones del procesador, se encuentran en el manual de programación.[**?**]

9. Ensamblar y ejecutar en el emulador. En la figura 2.20 se observan algunas recomendaciones:

    1. Antes de ejecutar la instrucción **PUSH**, observe y anote la dirección que tiene el registro *SP* (punto 2).

10. Escribir la dirección **0x200005f0**.

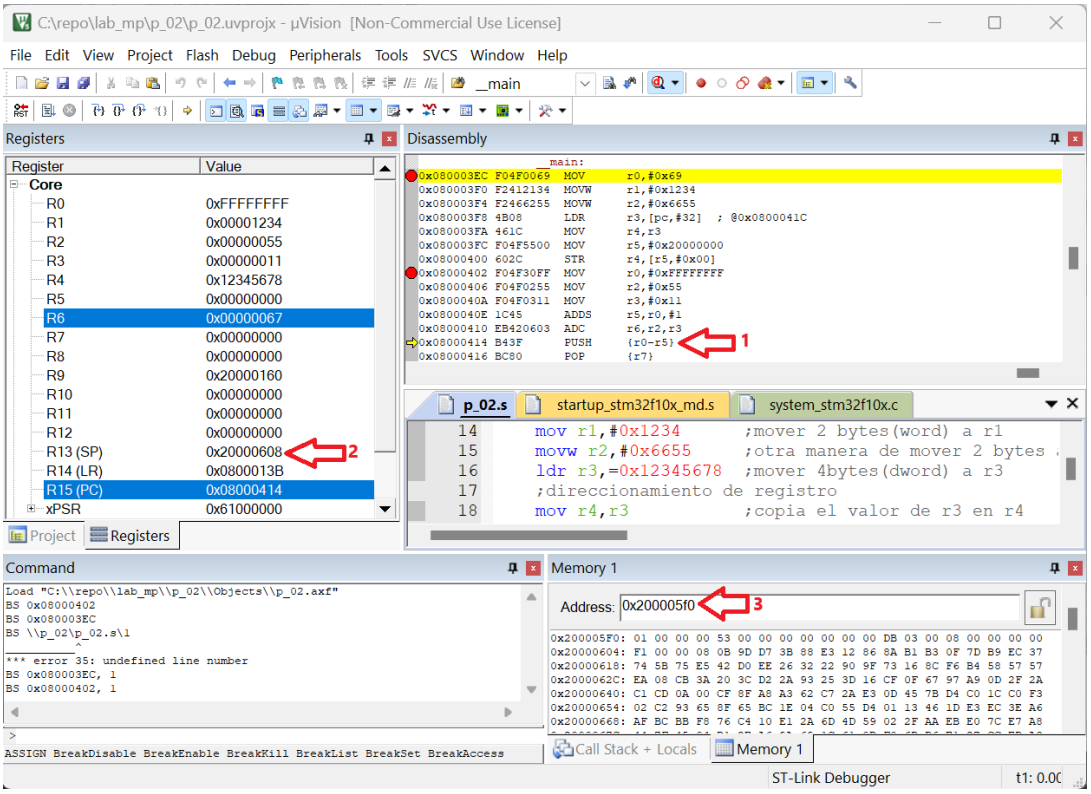

Figura 2.21: Emulador.

11. Continuar ejecutando el programa y haga sus anotaciones respecto al comportamiento del contenido en la dirección de memoria seleccionada.

2.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

2.8. Cuestionario

1. Para cada paso del procedimiento experimental, explique sus resultados y observaciones al utilizar el emulador como herramienta para depurar programas.

2. Escribir, ensamblar, emular y depurar un programa en lenguaje ensamblador para el procesador ARM Cortex-M3 que escriba 100 números a partir de la dirección de memoria 0x20000000.

3. Práctica: Unidad aritmética lógica

3.1. Objetivos

- Conocer las diferentes operaciones aritmético-lógicas que la ALU integrada en el microprocesador puede ejecutar.
- Identificar la aplicación de las distintas operaciones de la ALU.

3.2. Introducción

En una arquitectura de computadora básica la **ALU** es la parte de un procesador en la que se ejecutan operaciones aritmeticas y logicas con datos. La unidad de control determinara por medio de señales la operación a realizar. Los datos a utilizar pueden provenir de la unidad de memoria o de la unidad de entrada del sistema.[14]

Su propósito principal es aceptar datos binarios que están almacenados en la memoria y ejecutar las operaciones en estos datos de acuerdo a las instrucciones recibidas.

En su funcionamiento básico, una ALU contiene *al menos* dos registros de flip-flop, un registro B y un registro acumulador. También cotiene la lógica combinacional que realiza las operaciones en los números binarios los cuales se almacenan en el registro B y en el acumulador.[15]

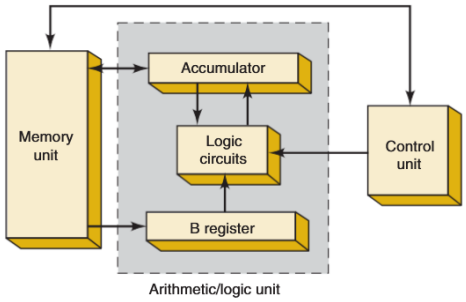

Figura 3.22: Diagrama de bloques básico de una ALU.[15]

Una secuencia de operaciones puede ocurrir de la siguiente manera:

- La unidad de control recibe una instrucción de la unidad de memoria especificando un número almacenado en una dirección de memoria particular el cual sera agregado al número almacenado en el registro acumulador.
- El número a ser sumado se transfiere de la memoria al registro B.
- El número en el registro B y el número en el acumulador se suman juntos en los circuitos lógicos. La sumatoria restante se guarda en el acumulador.
- El nuevo número en el acumulador puede permanecer en el registro a espera de que otro número se pueda sumar a el.

Esto unicamente describe una ALU de legado, en la arquitectura ARM, la ALU toma las variables directamente de los registros del procesador (r0 a r12), aceptando así, operaciones aritméticas y lógicas en datos de 32 bits. Ademas, la ALU en ARM (y arquitecturas modernas en general) tienen función de afectarrreo, etc... las banderas de estado, por lo que mediante estas operaciones podemos ejecutar comparaciones, detecciones de aca

3.3. Actividades previas

1. Investigue el set de instrucciones aritemicas y logicas que puede ejecutar la tarjeta de desarrollo STM32F103C8.

2. Con las instrucciones investigadas, desarrolle un programa en lenguaje ensamblador que resuelva los siguientes problemas:

   Calcular la elevación a la 5ta potencia del binomio $(9+7)^5$, cuyos coeficientes deberán ser almacenados en los registros r0-r5.

   Dividir los resultados obtenidos en los registros r3 y r4 y almacenar el resultado en r6.

   Sume todos los resultados y almacene la suma resultante en el registro r7

   Reste el resultado en el registro r8 con la suma de los valores que se ubican en los registros r2,r3 y r4.

   Aplique una mascara de 16 bits a los valores en r0 y r5 y únalos en un solo registro, el valor resultante se deberá encontrar en el registro 9.

   Multiplicar y Dividir por 2 los resultados obtenidos en r1 y r2 haciendo uso de desplazamientos lógicos, almacenar estos resultados en los registros r10 y r11.

3.4. Material

- 1 Placa de desarrollo STM32F103C8 “Blue Pill”.
- 1 Programador, emulador ST-Link V2.
- 4 cables hembra-hembra tipo Dupont.

3.5. Equipo

- PC con el software Keil μVision instalado

3.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Ejecutar el programa desarrollado en las actividades previas en modo debug.

4. Realice una tabla con los resultados obtenidos en cada registro.

Registro	Resultado
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Cuadro 2: Tabla de resultados.

3.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

3.8. Cuestionario

1. ¿Por qué la división se considera una operación pesada en microprocesadores?
2. Investigue que instrucciones de punto flotante tienen disponibles los microprocesadores ARM Cortex-M3.
3. Compare las instrucciones aritmeticas-logicas de los procesadores ARM Cortex-A9 con las de la arquitectura amd64 (x64) y mencione sus principales diferencias en cuanto a ciclos de reloj y complejidad.
4. ¿Cómo haría el calculo de una división **con punto decimal** en ensamblador ARM?. Incluya fotografías de los resultados obtenidos y su código desarrollado.

4. Práctica: Memoria estática de acceso aleatorio

4.1. Objetivos

- Comprender la importancia de la memoria de acceso aleatorio en sistemas basados en microprocesadores.
- Conocer y aplicar las funcionalidades de la memoria de acceso aleatorio.

4.2. Introducción

Las memorias RAM (memoria de acceso aleatorio por sus siglas en inglés) se usan en sistemas de computadoras para almacenamiento ***temporal*** de programas y datos. En esta se almacenan los datos que cambian a lo largo de la ejecución del programa principal, así que se usa como memoria de lectura y escritura, por ese motivo es que este tipo de memoria deben tener altas velocidades de acceso.

La memoria RAM es un tipo de memoria volátil, esto quiere decir que una vez que pierden conexión a voltaje se pierde la información almacenada.

Existen dos tipos de memoria RAM, memoria RAM estática (SRAM) y memoria RAM dinámica (DRAM). Las SRAM son en esencia un arreglo de multivibradores biestables (flip-flops) que permanecerán en un estado determinado indefinidamente, mientras que las DRAM solo mantienen los datos por un tiempo limitado (comúnmente 2ms después del cual se pierden los datos), por lo que requieren que los datos a almacenar sean actualizados en intervalos regulares para mantenerlos en memoria, esto se debe a que dependen de la carga de un capacitor que alimenta a un transistor MOSFET el cual actúa como interruptor (1 y 0), gracias a este comportamiento las memorias DRAM tienen un menor consumo de energía y menor costo a comparación de las memorias SRAM pero pierden la velocidad que las SRAM ofrecen[14].

Su acceso es mediante direcciones de memoria y cada dirección de memoria tiene una longitud de palabra definida. Así que la cantidad de direcciones de memoria indicarán el espacio disponible de almacenamiento. Al igual que, el tamaño limite de almacenamiento para estas memorias esta definido por la arquitectura de cada procesador, en el caso de la tarjeta de desarrollo STM32F103C8, contamos con un microprocesador de 32 bits, por lo que el tamaño de la memoria RAM máxima sería de $2^{32} = 4,294,967,29$ bytes.

4.3. Actividades previas

1. Investigue que utilidad tiene la memoria SRAM para códigos que tienen distintas subrutinas.
2. Anote las direcciones de la memoria SRAM embebida en la tarjeta de desarrollo STM32F103C8.
3. Implemente las diferentes formas de almacenar y cambiar datos en la memoria SRAM dentro de un programa en lenguaje ensamblador con las siguientes características:

   Asigne un espacio de 4 bytes en memoria SRAM para almacenar el valor a usar en una subrutina de delay.

   Programar una subrutina de delay que le tome al microprocesador 1 min en completar.

   Una vez que termine la subrutina de delay almacenar en una dirección de memoria SRAM la fecha y hora de la realización de la práctica, la cual se programará en el momento de la realización de la práctica en el laboratorio.

4.4. Material

- 1 Placa de desarrollo STM32F103C8 “Blue Pill”.
- 1 Programador, emulador ST-Link V2.
- 4 cables hembra-hembra tipo Dupont.

4.5. Equipo

- PC con el software Keil μVision instalado

4.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Ingresar en el área de su código desarrollado durante las actividades previas la fecha y hora de la realización de la práctica.
4. Ejecutar el programa desarrollado en las actividades previas en modo debug.
5. En alguna plataforma digital, subir un video que muestre el funcionamiento del código desarrollado en modo debug junto con un temporizador externo y proporcionar el enlace a este en su reporte de práctica.
6. En modo debug, con la herramienta “disassembler” identificar los llamados a subrutina y anotar la dirección que se otorga al Stack Pointer, ¿es una dirección que corresponde a la memoria SRAM?, anote sus observaciones.

4.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

4.8. Cuestionario

1. ¿Por qué se usa una memoria SRAM en sistemas embebidos como STM32F103C8 y no DRAM?
2. ¿Qué otros tipos de memoria volátil se usa en microprocesadores modernos y cuáles son sus diferencias con las memorias RAM?
3. Describa otras formas de almacenar datos en sistemas de microprocesadores.

5. Práctica: Memoria de solo lectura

5.1. Objetivos

- Identificar las diferencias entre memoria activa y pasiva.
- Comprender la importancia de la memoria de solo lectura para aplicaciones en microprocesadores.

5.2. Introducción

Las memorias de solo lectura **no volátiles** almacenan datos incluso cuando pierden la energía, a diferencia de las memorias de acceso aleatorio (RAM), que en su mayoría son volátiles, lo que significa que una vez se desconecta de la alimentación toda la información almacenada se borra. Es por eso que las memorias de solo lectura se han utilizado para almacenar el código a ejecutar en sistemas de microprocesadores.
La tecnología de las memorias de solo lectura ha evolucionado a lo largo de los años, pasando por diferentes tipos de circuitos integrados para el almacenamiento de datos. Hoy en día, en la mayoría de sistemas embebidos se usan memorias del tipo **Flash**, las cuales permiten una alta velocidad de lectura de datos aleatorios y una segmentación en páginas, lo que permite leer y escribir datos en diferentes secciones sin la necesidad de borrar y escribir toda la memoria. Esto permite, por ejemplo, almacenar el código a ejecutar y datos obtenidos durante su ejecución, esto para evitar la perdida de los mismos en caso de un corte en la alimentación.
Las memorias flash usan transistores de puerta flotante (FGT) para almacenar el voltaje y lograr el almacenamiento de datos no volátil, la cantidad de carga en la puerta flotante determinará si el transistor conducirá cuando un ajuste de “lectura” se aplica.
El modelo de almacenamiento **datos + código** requiere atributos que permitan la asignación eficiente del espacio entre el código y los datos, disponibilidad de rutinas para operaciones de escritura, y métodos para proteger el código almacenado y a los datos de corromperse. [5]

5.3. Actividades previas

1. Investigue que otros tipos de memoria de solo lectura existen y realice una tabla comparativa.
2. Investigue que tipo de memoria flash está integrada en la placa de desarrollo STM32F103C8 y su capacidad de memoria.
3. Investigue que directivas de lenguaje ensamblador se usan para almacenar datos constantes en memoria de solo lectura
4. Desarrolle un programa en lenguaje ensamblador que calcule la raíz cuadrada de Newton de la sumatoria de su número de cuenta.

   La ecuación a calcular será la siguiente:

$$x_{n+1} = \frac{1}{2}\left[x_n + \frac{\sum \text{N° de Cuenta}}{x_n}\right]$$

Hasta que el valor absoluto de la resta de $x_{n+1} - x_n$ sea menor o igual a 1, o sea:

$$\left|x_{n+1} - x_n\right| \leq 1$$

Debe inicializar el registro con el que compare el resultado absoluto de la resta con cualquier número mayor a 1 para evitar entrar en un ciclo infinito.
Dónde:

El primer valor de x_n será 1.

x_{n+1} es el resultado de la raíz cuadrada y es el ultimo valor que deberá aparecer en la memoria flash de forma descendente.

La operación $\frac{\sum \text{N° de Cuenta}}{x_n}$ debe realizarse con la instrucción udiv.

La operación $\frac{1}{2}\left[x_n + \frac{\sum \text{N° de Cuenta}}{x_n}\right]$ debe realizarse con un desplazamiento lógico hacia la derecha de un bit, o sea, con la instrucción lsr.

Su número de cuenta deberá ser almacenado en una look-up table haciendo uso de alguna de las directivas de ensamblador previamente investigadas.

Mediante apuntadores obtenga cada uno de los dígitos de su número de cuenta y ejecute la suma para resolver su raíz cuadrada.

En cualquier registro, almacene las direcciones de memoria en las que se encuentren cada uno de los dígitos de su número de cuenta.

5.4. Material

- 1 Placa de desarrollo STM32F103C8 “Blue Pill”.
- 1 Programador, emulador ST-Link V2.
- 4 cables hembra-hembra tipo Dupont.

5.5. Equipo

- PC con el software Keil μVision instalado

5.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Ejecute el programa desarrollado en modo debug.
4. Anote las direcciones donde se encuentran almacenados los dígitos de su número de cuenta, según el mapa de memoria del microprocesador ¿corresponden a una dirección de memoria flash?.
5. Con las direcciones de memoria obtenidas en el punto anterior, haciendo uso del mapa de memoria del modo debug de Keil μVision ingrese dichas direcciones e identifique cada uno de los dígitos de su número de cuenta, incluya fotografías donde señalice donde se encuentra almacenado.

5.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

5.8. Cuestionario

1. Investigue que otro tipo de operaciones puede realizar sobre la memoria flash interna.
2. ¿Por qué existen memorias de solo lectura y memorias de acceso aleatorio en sistemas de microprocesadores?
3. ¿Qué otro tipo de datos se pueden almacenar mediante programación en la memoria flash?
4. ¿Es posible conectar una memoria EEPROM externa a la tarjeta de desarrollo? y ¿de qué forma podría conectar e interactuar con este tipo de dispositivos?

6. Práctica: Puertos de propósito general

6.1. Objetivos

- Reconocer los registros involucrados en el manejo de los puertos de entrada y salida en su función de propósito general
- Generar código en lenguaje ensamblador para el manejo de datos de entrada y salida mediante los puertos de propósito general

6.2. Introducción

GPIO o por sus siglas en inglés ***general purpose input/output*** es un tipo de pin que se encuentra en un circuito integrado que no tiene una función especifica. Mientras que la mayoría de pines tienen un propósito dedicado, como enviar una señal a un componente en especifico, la función de un pin GPIO es personalizable y puede ser controlado mediante software. [2]

- **Modo del Pin**: cada bit de puerto de los puertos de propósito general I/O puede configurarse individualmente mediante software de diferentes maneras:

  Salida o entrada

  Analógico

  Función alterna (AF por sus siglas en inglés).

- **Características de pin**:

  Entrada: sin pull-up y sin pull-down o con pull-up o pull-down internos

  Salida: push-pull o open drain con capacidades de pull-up o pull-down

  Función alterna: push-pull o open drain con capacidades de pull-up o pull-down

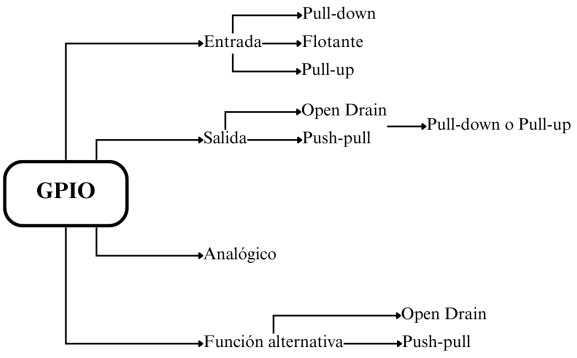

Figura 6.23: Diagrama de funciones para los puertos GPIO.

6.3. Actividades previas

1. Investigar que registros están involucrados en la configuración de los puertos GPIO y sus direcciones de memoria.
2. Investigar como es el diagrama del circuito interno de un teclado matricial 4x4.

3. Desarrollar un código en lenguaje ensamblador que muestre una secuencia de LED's conectados en un puerto X de salida, distinta para cada botón del teclado matricial 4x4 conectado en un puerto Y de entrada.

4. Dibujar el diagrama de flujo del programa desarrollado en el punto anterior.

5. Escribir el pseudocódigo del programa desarrollado.

6. Armar el circuito acorde al código desarrollado en el punto anterior.

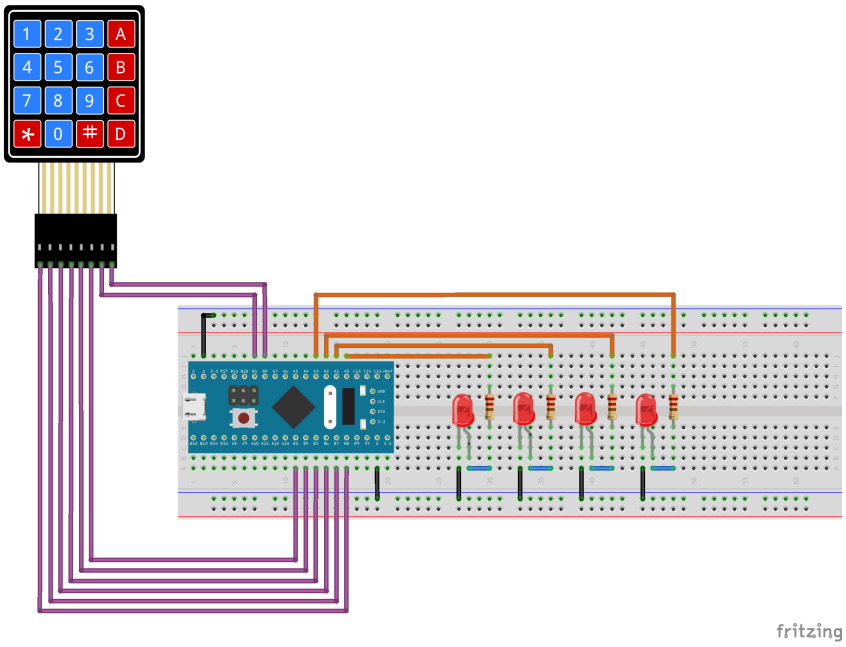

Figura 6.24: Circuito sugerido.

6.4. Material

- 1 Placa de desarrollo STM32F103C8 "Blue Pil"
- 1 Programador, emulador ST-Link V2
- 4 cables hembra-hembra tipo Dupont
- Un teclado matricial 4x4
- 4 LED's
- 4 resistencias de 330Ω a $\frac{1}{2}W$
- Cables para conexión

6.5. Equipo

- PC con el software Keil μVision instalado

6.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Con el código desarrollado en las actividades previas, programe la placa de desarrollo STM32F103C8.
4. En modo debug, observe los registros de control investigados en las actividades previas con su palabra respectiva para la función de GPIO. Incluir capturas de pantalla correspondientes en su reporte de práctica.
5. Incluya fotografías del funcionamiento del programa en su reporte de práctica.

6.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

6.8. Cuestionario

1. ¿Qué tipo de comunicación se usa con los puertos GPIO en su función básica?
2. Modifique el código desarrollado en las actividades previas para que cuando presione un botón en el teclado matricial muestre su valor binario en los LED's de salida.
3. Investigue que funciones alternativas tienen los puertos GPIO y que registros debe modificar para habilitarlos.

7. Práctica: Manejo de interrupciones

7.1. Objetivos

- Comprender el funcionamiento de las interrupciones en microprocesadores.
- Identificar que usos se le podrían dar a las interrupciones en aplicaciones reales.

7.2. Introducción

El termino interrupción se refiere a un tipo de evento de bajo nivel o de software que fuerza a los microprocesadores a interrumpir el flujo de ejecución de un programa, como una instrucción de llamado lo haría para redirigir a una rutina llamada *interrupt handler*, la cual es el conjunto de instrucciones que ejecutará el microprocesador en caso de recibir una interrupción. [10]

En el modelo de computadora Von Neumann, el procesador ejecuta instrucciones de forma secuencial. Antes de iniciar una instrucción verificar la ausencia de solicitudes de interrupción. Por lo que se puede decir que una solicitud de interrupción por hardware tiene mayor prioridad sobre la ejecución de una instrucción.[4]

Los procesadores ARM Cortex M-3 garantizan que aquellas tareas criticas e interrupciones sean atendidas de la forma mas rápida posible pero en un número conocido de ciclos. Estos procesadores incluten un controlador de interrupciones llamado NVIC o controlador de interrupciones vectorial anidado por sus siglas en inglés (Nested Vectored Interrupt Controller).[1]

El NVIC soporta:

- Un número de interrupciones definido por implementación, en el rango de 1-240 interrupciones.
- Un nivel de prioridad programable del 0-255 para cada interrupción. Un nivel alto corresponde a una prioridad baja, así que el nivel 0 es la interrupción con el nivel más alto de prioridad.
- Detección de nivel y pulso de las señales de interrupción.
- Re-priorización dinámica de interrupciones.
- Agrupación de valores de prioridad en campos de grupo de prioridad y sub-prioridad.
- Tail-chaining de interrupciones.
- Una interrupción externa no mascarable (NMI).
- WIC opcional, lo que provee soporte para un modo de reposo de ultra bajo consumo de energía.

7.3. Actividades previas

1. Investigar que registros están involucrados para habilitar el uso de interrupciones externas.
2. Investigar en que dirección de memoria se encuentra el registro NVIC para el manejo de interrupciones en el STM32F103C8.

3. Desarrollar un programa en lenguaje ensamblador para el manejo de interrupciones externas. En este caso, escoger un puerto X de salidas en el que conectará 4 LED's en los cuales mostrará una secuencia de encendido y apagado.

   Para la rutina de interrupción, aplicar un contador que indique cuantas interrupciones se han detectado y mostrar su valor binario en los 4 LED's durante al menos 2 segundos para que sea posible visualizarlo. Cuando el contador llegue al valor 0x0Fh, reiniciar el contador a 0.

4. Dibujar el diagrama de flujo de su programa desarrollado.

5. Escribir el pseudocódigo del programa desarrollado.

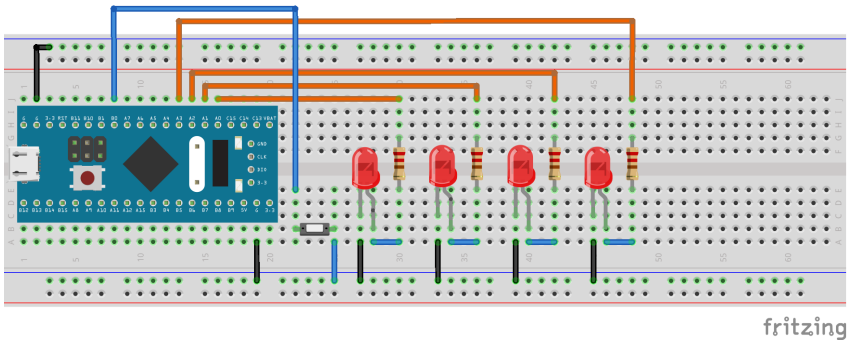

Figura 7.25: Circuito sugerido.

7.4. Material

- 1 Placa de desarrollo STM32F103C8 "Blue Pil"
- 1 Programador, emulador ST-Link V2
- 4 cables hembra-hembra tipo Dupont
- 1 Push-Button
- 4 LED's
- 4 resistencias de 330Ω a $\frac{1}{2}W$
- Cables para conexión

7.5. Equipo

- Osciloscopio
- PC con el software Keil μVision instalado

7.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Haciendo uso del programa desarrollado en las actividades previas, compile y programe la placa de desarrollo STM32F103C8.

4. Con el osciloscopio observe las señales provenientes de los puertos de salida según su programación.
5. Presione el botón generando la interrupción externa y comente sus observaciones.
6. En modo debug, observe el comportamiento de los registros investigados en las actividades previas, ademas del registro donde se almacena su contador de la rutina de interrupción y comente sus observaciones.

7.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

7.8. Cuestionario

1. Investigue que otro tipo de interrupciones puede manejar el STM32F103C8 y que registros debe modificar para hacer uso de las mismas.
2. Modifique el código anterior para que mediante un contador, cuando llegue a cierto limite definido por el programador, genere una interrupción de software que cambie la secuencia de LED's.
3. ¿En qué se diferencia una interrupción y una excepción?

8. Práctica: Interfaz de comunicación serial

8.1. Objetivos

- Crear un sistema de comunicación serial mediante el uso de señales digitales (puertos de propósito general).
- Implementar diferentes funciones de un microprocesador para resolver problemas de sincronía.
- Identificar que señales internas del microprocesador juegan un papel importante en la sincronía.
- Comprender la importancia de la sincronía en comunicaciones seriales.

8.2. Introducción

Una de las aplicaciones más importantes de los sistemas con microprocesadores son las comunicaciones, la emisión, recepción y el procesamiento de los datos son usos que se les han dado a estos sistemas desde su creación.
Las comunicaciones pueden ser en serie o en paralelo. La comunicación en serie se usa principalmente para transmitir datos a larga distancia, esto debido a que es mas barato enviar un unico cable de cobre a grandes distancias que enviar los multiples cables de cobre que se necesitarían para transmitir datos en paralelo.[11]
Las comunicaciones seriales se pueden clasificar de diferentes maneras:

- Sincronas
- Asincronas
- Simplex
- Duplex
- Half-Duplex

El microprocesador procesa los datos recibidos de forma individual bit a bit y los une para generar palabras completas que representen la información deseada. Este proceso generalmente se lleva a cabo en una serie de pasos:

1. **Inicia el proceso de comunicación:** este proceso se inicia desde el transmisor. El transmisor envía el primer bit al dispositivo receptor.
2. **Proceso de transmisión de datos:** El transmisor empieza a enviar los datos completos bit a bit en un orden especifico. Estos mensajes son largos y toman un tiempo para enviarse.
3. **Fin de la comunicación:** Cuando el transmisor termina su operación, se detiene el proceso de transmisión bit a bit.
4. **Detección de errores opcional:** Algunos protocolos de comunicación serial realizar una detección de errores del lado del receptor, esto se puede detectar mediante bits de paridad. Si hay errores el receptor solicita al transmisor que vuelva a enviar la información[6].

8.3. Actividades previas

1. Investigar cómo funciona la codificación “Not Return to Zero” o NRZ.

2. Según la hoja de datos del fabricante, que pin del circuito integrado del STM32F103C8 corresponde a la entrada de la señal de reloj proveniente del cristal de cuarzo incrustado en la placa de desarrollo.

   Esta práctica se realizará en parejas durante la sesión de laboratorio, sin embargo, ambos miembros del equipo deberán desarrollar su código de manera individual.

3. Desarrolle los diferentes códigos descritos a continuación:

   Todos los códigos solicitados durante esta práctica deberán desarrollarse en lenguaje C.

   Unicamente hacer uso de los paquetes de software CMSIS de arranque del dispositivo (los paquetes que se han usado en las prácticas anteriores). ***no está permitido*** el uso de cualquier otra librería CMSIS, el uso del software STM-CubeMX o de cualquier otro medio que contenga librerías previamente hechas por otros usuarios.

   Deberá desarrollar sus propias librerías para uso de puertos de entrada y salida (GPIO), manejo de interrupciones y delays con el modulo SysTick.

   El primer par de códigos tendrán las siguientes características:

   Emisor: Mediante el uso de delays, enviar el siguiente mensaje “ComunicacionSerial” a través de un puerto de propósito general, transmitiendo bit a bit el mensaje con una velocidad de 1 bit por segundo

   Receptor: Mediante el uso de delays, recibir en una constante el mensaje transmitido por el emisor, esto quiere decir que recibe bit a bit los datos siempre que el bit sea 1 y construye cada byte de la palabra. Para luego retransmitirlo mediante un puerto de entrada y salida (GPIO).

   El segundo par de códigos deberán tener las siguientes características:

   Emisor: Cambiar la velocidad de transmisión a 5 bits por segundo, ademas, mediante otro pin GPIO enviar una señal de reloj correspondiente a la velocidad de transferencia, la transmisión de cada bit debe de ocurrir en cada flanco de subida de la señal de reloj.

   Receptor: El método de recepción *ya no* hará uso de un delay para la sincronización, ahora mediante una interrupción externa generada por la señal de reloj del transmisor se recibirán los datos que se almacenarán en una constante, en el ciclo principal se representará la constante mediante un puerto GPIO bit a bit con la misma velocidad a la que se recibió (5 bits por segundo).

   Para la demostración visual de las señales de datos y la señal de reloj se harán uso de LED’s de la siguiente manera:

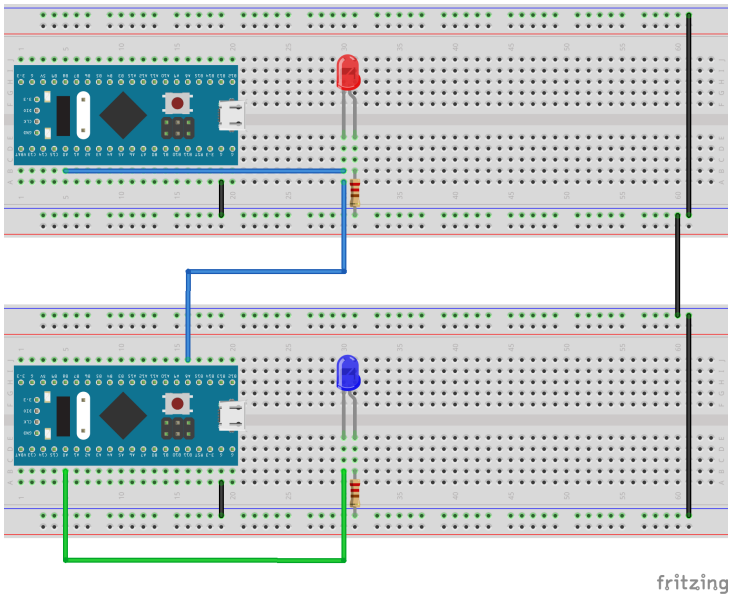

Figura 8.26: Circuito Sugerido.

8.4. Material

- 2 Placas de desarrollo STM32F103C8 “Blue Pill”.
- 2 Programadores, emuladores ST-Link V2.
- 4 cables hembra-hembra tipo Dupont.
- 4 LED’s (De preferencia de distintos colores).
- 4 Resistencias de 220Ω a $\frac{1}{2}W$
- Tableta de conexiones.
- Cables para conexiones.

8.5. Equipo

- PC con el software Keil μVision instalado
- Osciloscopio

8.6. Procedimiento experimental

Esta práctica debe desarrollarse en parejas

1. Conecte las placas de desarrollo STM32F103C8 a los programadores/emuladores ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Haciendo uso del primer par de programas desarrollados en las actividades previas, compile y programe las placas de desarrollo STM32F103C8.
4. Una vez programadas las placas de desarrollo, conecte un canal del osciloscopio a la linea de transmisión y el otro canal del osciloscopio a la linea que emite el dato recibido en el receptor.

5. Pulse el botón de RESET de ambas placas *al mismo tiempo* para intentar sincronizar los dispositivos. Según la señal observada en el osciloscopio decodifique los datos de ambas lineas haciendo uso del código ASCII incluido en el anexo B del manual de prácticas. Anote sus observaciones, ¿Se observa correctamente el mensaje en la línea del receptor?.

6. Modifique el código del receptor para ajustar el delay de forma que se consiga una mejor sincronía, ya sea que aumente el tiempo de delay o disminuya.

7. Repita el paso 5, comente sus observaciones y anote a que tiempo tuvo que ajustar el delay para conseguir una mejor sincronía.

8. Con la punta de precisión del osciloscopio, observe la señal de reloj que reciben cada microprocesador, indique que nivel de desfase en el tiempo tienen ambas señales y comente porque esto tiene efecto en la sincronía de la comunicación.

9. Ahora programe ambas tarjetas de desarrollo con el segundo par de códigos generados en las actividades previas.

10. Modifique el circuito armado inicialmente para incluir un LED en el receptor y en el emisor en la línea de reloj de sincronía.

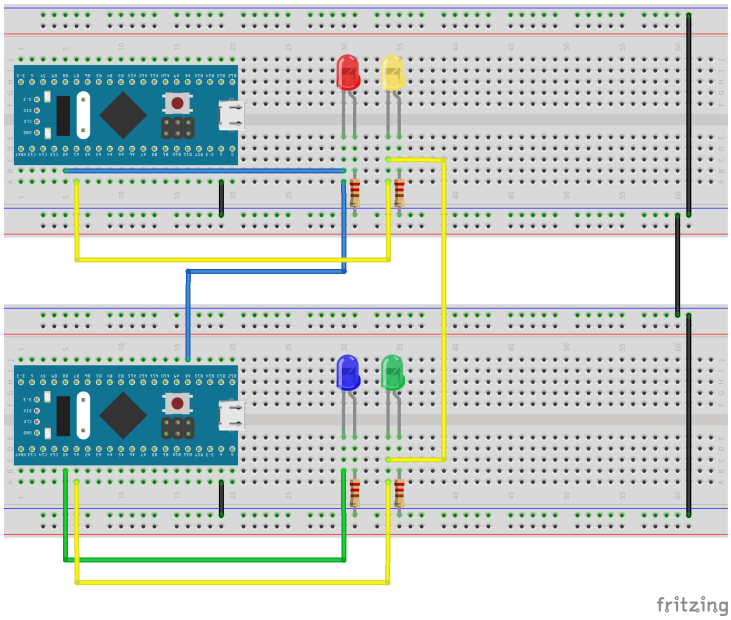

Figura 8.27: Modificación al circuito inicial.

11. Inicie la ejecución de los programas desarrollados, ¿es necesario hacer un RESET al mismo tiempo para conseguir sincronía? ¿Por qué?.

12. Observe la señal transmitida y la señal emitida por el receptor, decodifique ambas líneas con el código ASCII y compare estos resultados con los observados en los puntos 5 y 7, ¿se recibió correctamente el mensaje en el receptor?, Comente que diferencias observa con las señales anteriores.[9]

8.7. Análisis de resultados

En esta sección deberá escribir el análisis de los resultados obtenidos.

8.8. Cuestionario

1. Explique porque el delay es un mal metodo de sincronía del lado del receptor.
2. ¿Qué aplicaciones le daría a este sistema de comunicaciones seriales?
3. ¿Cómo mejoraría este sistema de comunicaciones para corregir posibles errores de transmisión?
4. Investigue los diferentes protocolos de comunicación serial mas usados en sistemas con microprocesadores y haga una tabla comparativa.
5. De los protocolos investigados, ¿cuál es el que más se parece al implementado durante esta práctica?.

9. Práctica: Comunicación entre circuitos (I^2C)

9.1. Objetivos

- Comprender el funcionamiento del protocolo de comunicación I^2C.
- Hacer uso del periférico I^2C de la placa de desarrollo STM32F103C8.

9.2. Introducción

El protocolo para circuitos inter-integrados o I^2C fue desarrollado en 1982 por Phillips Semiconductor, y es un protocolo comunicación de baja velocidad para conectar dispositivos controladores, como microcontroladores y procesadores que tengan dispositivos de destino. Su origen se dio dada la necesidad de reducir la cantidad de líneas de comunicación que se usaban en la comunicación paralela.[16]

I^2C es un protocolo de comunicación serial que hace uso de dos líneas: SCL, la cual es la señal de reloj serial (Serial Clock por sus siglas en inglés), y SDA, la cual es la señal de datos seriales (Serial Data por sus siglas en inglés). [12]

En este protocolo se tienen dos tipos de elementos principales: Maestro y Esclavo. El maestro es quien inicializa la comunicación, envía la señal de reloj a todos los elementos y procesa los datos que sean enviados y/o recibidos por los esclavos en la línea SDA. Un sistema I^2C típico consta de un maestro y uno o varios esclavos.[3]

En un solo bus I^2C se pueden conectar hasta 128 dispositivos, esto dado que el sistema de direcciones es de 7 bits. Cada dispositivo esclavo **debe** de tener una dirección única, ya que es el medio por el cual el maestro identifica cada dispositivo en el bus.

En sus especificaciones, en las líneas de comunicación tiene una resistencia pull-up, cuyo valor depende de cada dispositivo maestro y sus características eléctricas.[16]

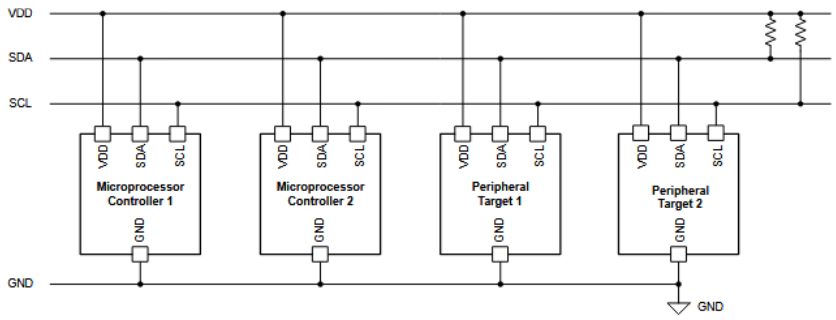

Figura 9.28: Aplicación típica del protocolo I^2C.[16]

9.3. Actividades previas

1. Describa un ciclo de comunicación I^2C, dibuje las señales involucradas y señalice cada parte del proceso.
2. Investigue que registros del STM32F103C8 se involucran en la configuración del periférico I^2C.
3. Investigue que dirección tiene el expansor I^2C PCF8574 por defecto y como se puede modificar.
4. Desarrolle un proyecto en Keil μVision con las siguientes características:

Todos los códigos deben ser desarrollados en lenguaje C y **unicamente** usando los paquetes de software CMSIS de arranque del dispositivo. (Los paquetes que se han usado durante las prácticas anteriores). **No está permitido** el uso de ninguna librería CMSIS, el uso de STM-CubeMX o de cualquier otro medio que contenga librerías previamente hechas por otros usuarios.

El alumno deberá desarrollar su propia librería de manejo de I^2C para el STM32F103C8, esto basándose en la hoja de datos técnicos proporcionada por el fabricante, así cómo una librería para generar delays con el modúlo SysTick.

Igualmente, se desarrollará una librería para el manejo del LCD mediante el bus I^2C, debe incluir por lo menos la función de escribir en cualquier posición de la pantalla y borrar la pantalla.

Haciendo uso de las librerías anteriormente creadas, desarrollé un programa principal en el que:

Se ingrese un código de 4 dígitos, los cuales correspondan a los últimos 4 dígitos de su número de cuenta y muestre el código en la pantalla LCD.

Si el código es incorrecto, mostrar en la pantalla una leyenda indicando que el acceso es denegado y un contador de intentos incorrectos. Si el contador supera los 3 intentos, muestre la leyenda “Acceso Bloqueado”, ademas de que debe mantenerse el microprocesador en un ciclo infinito, lo que obligaría al usuario a reiniciar la placa de desarrollo mediante el botón RESET.

Si el código es correcto, mostrar la leyenda “Bienvenido al Laboratorio"

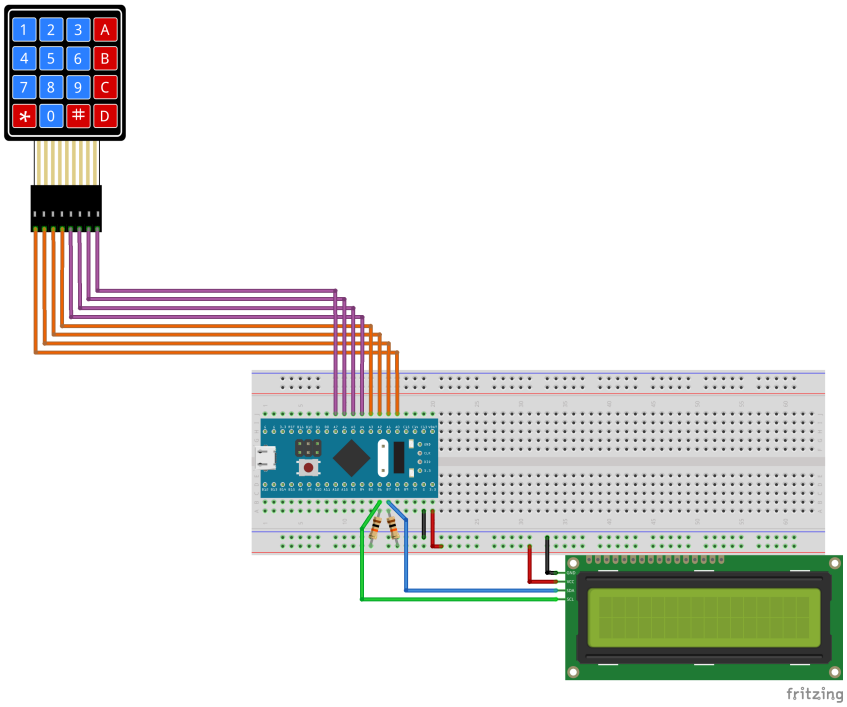

Figura 9.29: Circuito Sugerido.

9.4. Material

- 1 Placa de desarrollo STM32F103C8 “Blue Pill”.

- 1 Programador, emulador ST-Link V2.
- 4 cables hembra-hembra tipo Dupont.
- 1 LCD HD44780 con expansor I^2C PCF8574
- Tableta de conexiones.

9.5. Equipo

- PC con el software Keil μVision instalado
- Osciloscopio

9.6. Procedimiento experimental

1. Conecte la placa de desarrollo STM32F103C8 al programador/emulador ST-Link V2.
2. Conecte el programador ST-Link V2 a una computadora con el software Keil μVision.
3. Programe la placa de desarrollo con el proyecto desarrollado en las actividades previas.
4. Haciendo uso del osciloscopio, observe las señales SCL y SDA de la placa de desarrollo mientras esta conectada a la pantalla LCD, señale las partes del ciclo de comunicación I^2C investigadas en las actividades previas.
5. Compruebe el funcionamiento de su proyecto, incluyendo fotografías de todos los casos solicitados.

9.7. Cuestionario

1. ¿Qué modificaciones haría tanto al código como al circuito para incluir un sensor de temperatura BME280 con comunicación I^2C?
2. ¿Qué ventajas y desventajas tiene I^2C sobre otros protocolos de comunicación serial?
3. Investigue que otros circuitos integrados con comunicación I^2C se utilizan en dispositivos de consumo y en la industria.

A. Directivas del ensamblador

Conjunto de instrucciones Cortex-M3

Mnemonic	Operands	Brief description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with carry	N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V
ADD, ADDW	{Rd,} Rn, #imm12	Add	N,Z,C,V
ADR	Rd, label	Load PC-relative address	-
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C
ASR, ASRS	Rd, Rm, <Rs|#n>	Arithmetic shift right	N,Z,C
B	label	Branch	-
BFC	Rd, #lsb, #width	Bit field clear	-
BFI	Rd, Rn, #lsb, #width	Bit field insert	-
BIC, BICS	{Rd,} Rn, Op2	Bit clear	N,Z,C
BKPT	#imm	Breakpoint	-
BL	label	Branch with link	-
BLX	Rm	Branch indirect with link	-
BX	Rm	Branch indirect	-
CBNZ	Rn, label	Compare and branch if non zero	-
CBZ	Rn, label	Compare and branch if zero	-
CLREX	-	Clear exclusive	-
CLZ	Rd, Rm	Count leading zeros	-
CMN, CMNS	Rn, Op2	Compare negative	N,Z,C,V
CMP, CMPS	Rn, Op2	Compare	N,Z,C,V
CPSID	iflags	Change processor state, disable interrupts	-
CPSIE	iflags	Change processor state, enable interrupts	-
DMB	-	Data memory barrier	-
DSB	-	Data synchronization barrier	-
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C
ISB	-	Instruction synchronization barrier	-
IT	-	If-then condition block	-
LDM	Rn{!}, reglist	Load multiple registers, increment after	-
LDMDB, LDMEA	Rn{!}, reglist	Load multiple registers, decrement before	-
LDMFD, LDMIA	Rn{!}, reglist	Load multiple registers, increment after	-
LDR	Rt, [Rn, #offset]	Load register with word	-
LDRB, LDRBT	Rt, [Rn, #offset]	Load register with byte	-
LDRD	Rt, Rt2, [Rn, #offset]	Load register with two bytes	-
LDREX	Rt, [Rn, #offset]	Load register exclusive	-
LDREXB	Rt, [Rn]	Load register exclusive with byte	-
LDREXH	Rt, [Rn]	Load register exclusive with halfword	-
LDRH, LDRHT	Rt, [Rn, #offset]	Load register with halfword	-
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load register with signed byte	-
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load register with signed halfword	-
LDRT	Rt, [Rn, #offset]	Load register with word	-
LSL, LSLS	Rd, Rm, <Rs|#n>	Logical shift left	N,Z,C
LSR, LSRS	Rd, Rm, <Rs|#n>	Logical shift right	N,Z,C
MLA	Rd, Rn, Rm, Ra	Multiply with accumulate, 32-bit result	-
MLS	Rd, Rn, Rm, Ra	Multiply and subtract, 32-bit result	-
MOV, MOVS	Rd, Op2	Move	N,Z,C
MOVT	Rd, #imm16	Move top	-
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C
MRS	Rd, spec_reg	Move from special register to general register	-
MSR	spec_reg, Rm	Move from general register to special register	N,Z,C,V
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C
NOP	-	No operation	-
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C
POP	reglist	Pop registers from stack	-
PUSH	reglist	Push registers onto stack	-
RBIT	Rd, Rn	Reverse bits	-
REV	Rd, Rn	Reverse byte order in a word	-
REV16	Rd, Rn	Reverse byte order in each halfword	-
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	-
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate right	N,Z,C
RRX, RRXS	Rd, Rm	Rotate right with extend	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse subtract	N,Z,C,V
SBC, SBCS	{Rd,} Rn, Op2	Subtract with carry	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Signed bit field extract	-
SDIV	{Rd,} Rn, Rm	Signed divide	-
SEV	-	Send event	-
SMLAL	RdLo, RdHi, Rn, Rm	Signed multiply with accumulate (32 x 32 + 64), 64-bit result	-
SMULL	RdLo, RdHi, Rn, Rm	Signed multiply (32 x 32), 64-bit result	-
SSAT	Rd, #n, Rm {,shift #s}	Signed saturate	Q
STM	Rn{!}, reglist	Store multiple registers, increment after	-
STMDB, STMEA	Rn{!}, reglist	Store multiple registers, decrement before	-
STMFD, STMIA	Rn{!}, reglist	Store multiple registers, increment after	-
STR	Rt, [Rn, #offset]	Store register word	-
STRB, STRBT	Rt, [Rn, #offset]	Store register byte	-
STRD	Rt, Rt2, [Rn, #offset]	Store register two words	-
STREX	Rd, Rt, [Rn, #offset]	Store register exclusive	-
STREXB	Rd, Rt, [Rn]	Store register exclusive byte	-
STREXH	Rd, Rt, [Rn]	Store register exclusive halfword	-
STRH, STRHT	Rt, [Rn, #offset]	Store register halfword	-
STRT	Rt, [Rn, #offset]	Store register word	-
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V
SVC	#imm	Supervisor call	-
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	-
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	-
TBB	[Rn, Rm]	Table branch byte	-
TBH	[Rn, Rm, LSL #1]	Table branch halfword	-
TEQ	Rn, Op2	Test equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C
UBFX	Rd, Rn, #lsb, #width	Unsigned bit field extract	-
UDIV	{Rd,} Rn, Rm	Unsigned divide	-
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned multiply with accumulate (32 x 32 + 64), 64-bit result	-
UMULL	RdLo, RdHi, Rn, Rm	Unsigned multiply (32 x 32), 64-bit result	-
USAT	Rd, #n, Rm {,shift #s}	Unsigned saturate	Q
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	-
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	-
WFE	-	Wait for event	-
WFI	-	Wait for interrupt	-

B. Código ASCII

CÓDIGO ASCII

Binario	Dec	Hex	Representación	Binario	Dec	Hex	Representación	Binario	Dec	Hex	Representación
0010 0000	32	20	espacio ( )	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

Caracteres imprimibles del Código ASCII