Taller de Sistemas Embebidos
STM32 MCU - Timers

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

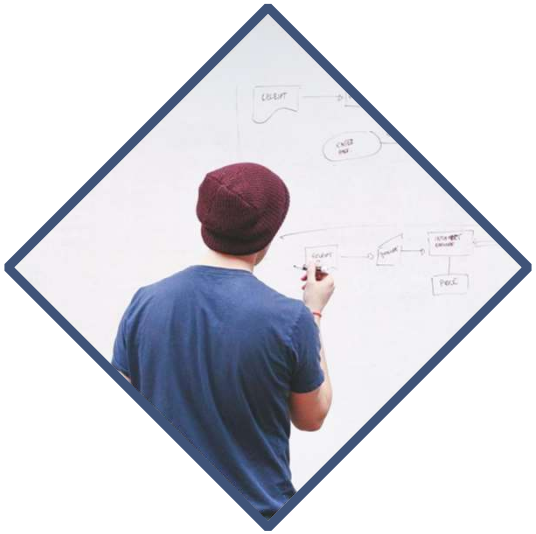
Estructura Curricular de la Carrera

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

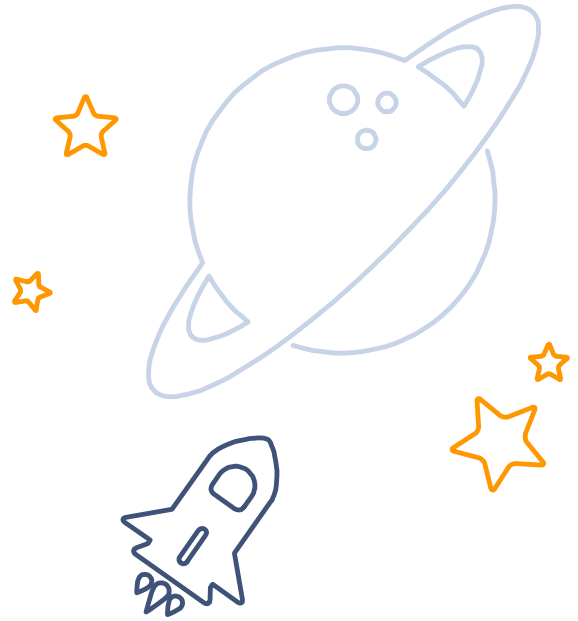
Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

Referencia:

- Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)
- Chapter 7: Timers
  - Los periféricos temporizadores son componentes muy importantes dentro de una MCU. Muchas aplicaciones embebidas dependen del tiempo y los temporizadores son el medio principal por el cual la MCU controla la aplicación. Si bien es posible utilizar una MCU para cronometrar procesos directamente, sería un gran desperdicio de potencia de procesamiento y un enfoque altamente ineficiente. El uso de temporizadores de hardware junto con interrupciones es realmente la única forma práctica de implementar aplicaciones integradas dependientes del tiempo. Afortunadamente, STM ofrece una poderosa variedad de temporizadores en su línea de MCU.

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 7: Timers

STM Timer Peripherals

- Un temporizador es simplemente un contador de funcionamiento libre que cuenta los pulsos de una fuente de reloj.
  - Dado que el tren de impulsos de una fuente de reloj tiene un período o intervalo conocido entre impulsos, el tiempo transcurrido está directamente relacionado con el número de impulsos contados.
  - La fuente del reloj temporizador utilizada con una MCU típica generalmente se deriva de un reloj maestro interno.
  - Sin embargo, el reloj maestro generalmente funciona a una velocidad muy rápida, por lo que los pulsos del temporizador deben dividirse mediante hardware en valores más razonables que puedan ser utilizados por los temporizadores.
  - El hardware utilizado para "reducir" la velocidad del reloj maestro varía y puede ser divisores binarios (preescalador) o bucles de bloqueo de fase (PLL).

STM Timer Peripherals

- Los contadores temporizadores pueden contar o acumular conteos hasta que se desborden según la cantidad de bits utilizados en el contador.
- Un contador de 16 bits se desbordará cuando se alcance el recuento máximo de 65535. Una interrupción por desbordamiento es a menudo el evento que ocurre en esta situación.

- Por el contrario, un contador temporizador puede realizar una cuenta regresiva desde algún valor preestablecido y desencadenar una interrupción cuando alcanza un valor 0.
  - Este tipo de contador se conoce como contador decreciente y tiene muchos usos.
- Los temporizadores tienen muchos usos, incluidos, entre otros, los siguientes:
  - Generar una base de tiempo precisa. Todos los temporizadores STM pueden hacer esto.
  - Medición de la frecuencia de un tren de pulsos digitales entrante.

STM Timer Peripherals

- Medición del tiempo transcurrido en una señal de salida. Esto se denomina comparación de salida para los temporizadores STM.
- Generar señales precisas de modulación de ancho de pulso (PWM) utilizadas para el control de servomotores y motores.
- Generación de pulso único con longitud programable y características de retardo.
- Generar señales periódicas de acceso directo a memoria (DMA) en respuesta a eventos de actualización, activación, captura de entrada y comparación de salida.

Existen las siguientes cinco categorías amplias de temporizadores STM:

- Básico: Temporizadores simples de 16 bits. No tienen pines de entrada ni de salida.
  - Se utilizan principalmente como "maestros" para otros cronómetros. También se utilizan como fuente de reloj del convertidor digital a analógico (DAC). Pueden generar una base de tiempo como pueden hacer todos los temporizadores STM.

STM Timer Peripherals

- Propósito general (GP): Son temporizadores de 16 o 32 bits con pines de entrada y salida.
  - Pueden realizar todas las funciones descritas en la lista anterior. Los temporizadores GP pueden tener hasta cuatro canales programables, los cuales se pueden configurar de la siguiente manera:
    - Uno o dos canales
    - Uno o dos canales con salida complementaria. La salida complementaria dispone de un generador de “tiempos muertos”, que proporciona una base de tiempos independiente.
- Avanzado: Todas las características del temporizador GP con funciones adicionales relacionadas con el control del motor y la conversión de energía digital. Hay tres salidas complementarias disponibles con esta categoría de temporizador con una entrada de apagado de emergencia.

STM Timer Peripherals

- Alta resolución: Múltiples salidas de alta resolución posibles gracias a seis subtemporizadores, un maestro y cinco esclavos.
  - Con este temporizador son posibles múltiples inserciones de tiempo muerto. También hay cinco entradas de falla y 10 entradas de eventos externos.
  - Este temporizador tiene el acrónimo HRTIM1 en terminología STM.
- Bajo consumo: Temporizadores utilizados en aplicaciones excepcionales de bajo consumo.
  - Estos temporizadores están diseñados para seguir funcionando en casi todos los modos de MCU STM, excepto en el modo de espera.
  - Incluso pueden seguir funcionando sin una fuente de reloj interna.

STM Timer Peripherals

- La MCU STM32F302R8 utilizada en la placa de demostración tiene temporizadores básicos, GP y avanzados.
- Me centraré en cómo funcionan y se programan estos temporizadores. Los conceptos discutidos también se aplicarán a los otros tipos de temporizadores si los encuentra en otros proyectos. Como siempre, las hojas de datos del fabricante son su mejor recurso para determinar cómo funciona un temporizador en particular y cómo se debe programar.
- La MCU STM32F302R8 tiene seis temporizadores, como se muestra en la Tabla 6-1. Esta tabla tiene una referencia útil a tener en cuenta al seleccionar y programar un temporizador.
- Ahora discutiré cómo configurar un temporizador STM usando el framework HAL.

Timer Type		Name
Advanced		TIM1
GP	16-bit	TIM15 TIM16 TIM17
GP	32-bit	TIM2
Basic		TIM6

Table 6-1 *STM32F302R8 MCU Timers*

STM Timer Configuration

- Los temporizadores STM se configuran de la misma manera que los pines GPIO y las interrupciones, como mencioné anteriormente.
- Los temporizadores STM se configuran con una estructura C denominada TIM_HandleTypeDef, que contiene los siguientes miembros:

```
TIM_HandleTypeDef,

uint32_t Prescaler;
uint32_t CounterMode;
uint32_t Period;
uint32_t ClockDivision
uint32_t RepetitionCounter;
```

- Los miembros de la estructura representan los siguientes parámetros y configuraciones del temporizador:
  - Preescalador: El factor de división utilizado para escalar la velocidad del reloj maestro.
    - Sólo se utilizan preescaladores de 16 bits en todos los temporizadores.

STM Timer Configuration

- Un registro de 16 bits puede contener valores de preescalador que van desde 1 a 65535. Por ejemplo, un valor de preescalador de 40.000 aplicado a un reloj maestro de 80 MHz significaría que se introduciría una frecuencia de reloj de 2 kHz en el temporizador.

- CounterMode: establece la dirección del conteo. Los modos de contador disponibles son los siguientes:
  - TIM_COUNTERMODE_UP
  - TIM_COUNTERMODE_DOWN
  - TIM_COUNTERMODE_CENTERALIGNED1
  - TIM_COUNTERMODE_CENTERALIGNED2
  - TIM_COUNTERMODE_CENTERALIGNED3

STM Timer Configuration

- Periodo: Este es un número que representa el tiempo máximo que transcurrirá antes de que el contador del temporizador se recargue con este número.
  - El número máximo es 0xffff para contadores de 16 bits y 0xffff ffff para contadores de 32 bits. Un valor de 0x0 garantizará que no se inicie el temporizador.
- ClockDivision: Este es un campo específico de bits usado para configurar la relación entre la frecuencia del reloj del temporizador interno y un reloj de muestreo usado para filtros digitales.
  - También se utiliza para configurar parámetros de tiempo muerto.

STM Timer Configuration

- Los modos ClockDivision disponibles son los siguientes:
  - TIM_CLOCKDIVISION_DIV1
  - TIM_CLOCKDIVISION_DIV2
  - TIM_CLOCKDIVISION_DIV4
- RepetitionCounter: establece un límite para la cantidad de veces que un temporizador puede desbordarse o no.
  - El registro de actualización del temporizador se configurará cuando se alcance el límite. También se puede generar un evento junto con la actualización del registro.

Update Event Calculation

- Se debe utilizar la siguiente ecuación para calcular el tiempo entre eventos de actualización para una frecuencia de reloj de alta velocidad determinada, un valor de preescalador y un valor de período:

$$Update\ event = \frac{High\ speed\ clock}{(Prescaler + 1)(Period + 1)}$$

- A continuación se ve un cálculo de muestra:
  - Dado:

High-speed clock = 80 MHz
Prescaler = 39999
Period = 1999

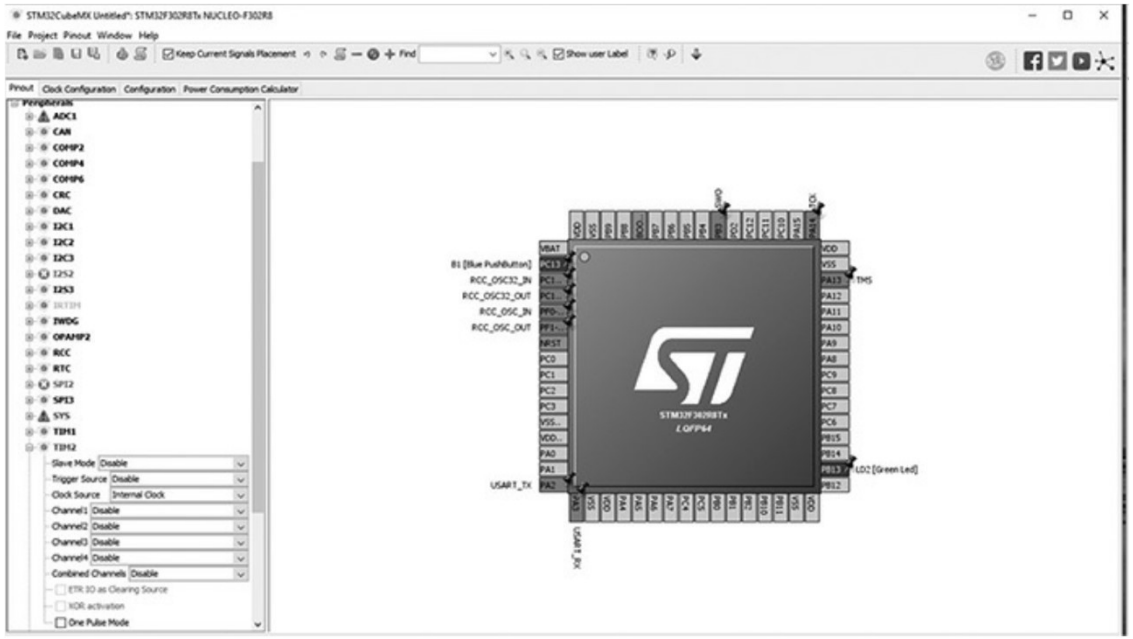
$$Update\ event = \frac{80000000}{(39999 + 1)(1999 + 1)} = \frac{80000000}{80000000} = 1.0\ s$$

- La discusión sobre el proyecto de demostración que se presenta a continuación le mostrará cómo configurar un temporizador GP en modo sondeado, que a su vez hará parpadear un LED.

Polled or Non-interrupt Blink LED Timer Demonstration

- Esta es la primera de dos demostraciones que le muestran cómo configurar el temporizador TIM2 GP para controlar LD2, el LED verde integrado.
  - Esta demostración no utilizará una interrupción, por lo que puedo concentrarme en la discusión sobre cómo configurar un temporizador.
  - El LED parpadeará a una velocidad de 0,5 s según la selección de valores de período y preescalador, así como la frecuencia de reloj de alta velocidad de 64 MHz preestablecida de la placa Nucleo-64.
- Debe iniciar este proyecto utilizando la aplicación CubeMX proporcionando un nombre de proyecto apropiado.
  - Usé el nombre TIM2_Example1 para este proyecto.
  - En la vista Pinout, debe seleccionar el reloj interno de alta velocidad como fuente de reloj del temporizador TIM2, que se muestra en la Figura 7-1.

- La selección de la fuente del reloj también hace que la aplicación CubeMX incluya todas las declaraciones de inicialización de TIM2 que se incluirán en el nuevo proyecto. Este paso es vital para generar un proyecto de trabajo.
- No olvide que todos los demás pasos anteriores de creación del proyecto siguen siendo apropiados, como incluir solo los archivos de biblioteca necesarios y asegurarse de que se genere un archivo hexadecimal. Discutiré las partes clave del código del proyecto después de la siguiente lista.



```
// STM disclaimer goes here
// Includes
#include "main.h"
#include "stm32f3xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

// Private variables
TIM_HandleTypeDef htim2;

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

/* USER CODE END PV */

// Private function prototypes
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_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 peripherals, initialize the flash interface and Systick.
 HAL_Init();
```

k LED T

```
  /* 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();
  MX_TIM2_Init();

  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

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

  /* USER CODE BEGIN 3 */
        int timerValue = __HAL_TIM_GET_COUNTER(&htim2);
        if(timerValue == 150)
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        if(timerValue == 399)
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  }
  /* 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;
```

Blink LED T

```
  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);
}

/* TIM2 init function */
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 63999; // Establishes a 1 KHz clock input
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 500; // Establishes the 0.5 second period
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  HAL_TIM_Base_Start(&htim2);  // This starts the TIM2 timer
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
     PA2   ------> USART2_TX
     PA3   ------> USART2_RX
*/
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 Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, 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 : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

Blink LED Timer

```
  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 *****/
```

- La mayor parte de este código es una repetición del código main.c que se muestra en el proyecto de demostración de interrupciones discutido en el capítulo anterior.
  - Las diferencias clave son la adición de un método MX_GPIO_Init y algún código operativo nuevo requerido en la parte del bucle permanente del método main.

Polled or Non-interrupt Blink LED Timer Demonstration

- El método MX_GPIO_Init configura TIM_HandleTypeDef, una estructura C escrita con valores específicos para que el temporizador se reinicie cada 0,5 s.
  - La estructura se denomina htim2 y se inicializa e inicia en este método.
- El bucle permanente en el método main tiene un código que verifica continuamente el conteo del temporizador y encenderá el LED cuando el conteo llegue a 150.
  - Permanecerá encendido hasta que el conteo avance a un valor de 399, momento en el cual el LED se reiniciará.
  - Estas declaraciones dan como resultado que el LED esté encendido durante 0,25 s y apagado durante 0,25 s.

Polled or Non-interrupt Blink LED Timer Demonstration

Test Run

- Construí el proyecto y lo subí a la placa Nucleo-64.
- El LED del usuario inmediatamente comenzó a parpadear a una velocidad de 0,5 s, lo que confirmó que el programa funcionó según lo deseado.
- El siguiente paso en esta serie de demostraciones de temporizadores es implementar una versión controlada por interrupciones que también hará parpadear el LED, pero que no requerirá ningún código especializado en el bucle permanente del método main.

Interrupt-Driven Blink LED Timer Demonstration

- Esta demostración también hará parpadear un LED.
  - En esta demostración, usaré un enfoque de interrupción, eliminando así cualquier código especializado en el bucle principal para siempre que lee continuamente el valor del contador del temporizador y reacciona cuando se alcanzan ciertos valores.
- Deberá generar un nuevo proyecto utilizando la aplicación CubeMX. Llamé a este proyecto Timer_int para reflejar su propósito.
  - También deberá seleccionar la fuente del reloj del temporizador como se hizo en la demostración anterior.
  - Simplemente use el reloj interno de alta velocidad como fuente, lo que activará la aplicación para incluir todo el código de inicialización y configuración del temporizador TIM2.

Interrupt-Driven Blink LED Timer Demonstration

- Sin embargo, esta vez también tendrás que configurar el temporizador para que active una interrupción cuando finalice su período.
- Esto se hace fácilmente usando la pestaña Configuración de CubeMX.
- Debería ver aparecer el temporizador TIM2 en la pestaña, pero primero asegúrese de haber seleccionado la fuente del reloj.
- De lo contrario, el temporizador TIM2 no aparecerá.
- Haga clic en el icono de TIM2 y luego haga clic en la casilla de verificación habilitar interrupción global.
- La pestaña Configuración y la casilla de verificación habilitar interrupción global se muestran en la Figura 7-2.

Interrupt-Driven Blink LED Timer Demonstration

- No repetiré la extensa lista de códigos mostrada anteriormente, sino que simplemente presentaré los segmentos de código necesarios para implementar el código basado en interrupciones.
- La primera parte es fácil, donde el bucle eterno se cambia a una acción nula:
- while(1);
- El único otro cambio que se debe hacer es configurar el período del temporizador en 250 para garantizar que el LED parpadee dos veces por segundo. El temporizador ahora se interrumpirá en el intervalo apropiado sin necesidad de ningún código especial para monitorear los valores del contador en tiempo real.

Interrupt-Driven Blink LED Timer Demonstration

- Las siguientes tres declaraciones deben agregarse justo antes del bucle eterno en el método principal:

```
// Start the TIM2 timer in the interrupt mode
HAL_TIM_Base_Start_IT(&htim2);
// Set the interrupt priority
HAL_NVIC_SetPriority(TIM2_IRQn, 0, 0);
// Enable the peripheral IRQ
HAL_NVIC_EnableIRQ(TIM2_IRQn);
```

- El método TIM2 IRQ handler debe definirse en el archivo main.c de la siguiente manera:

```
void TIM2_IRQHandler(void) {
    HAL_TIM_IRQHandler(&htim2);
}
```

- Finalmente, también se debe definir un método callback, que alterna el pin del LED cada vez que el temporizador TIM2 genera una interrupción:

```
// This callback is automatically called by the HAL on the TIM2 event:

void HAL_TIM_PeriodElapsedCallback(*htim2) {
   if(htim2->Instance == TIM2)
   HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
}
```

Interrupt-Driven Blink LED Timer Demonstration

Test Run

- Hice las modificaciones al archivo main.c, construí el proyecto y luego lo subí a la placa Nucleo-64.
- El LED del usuario inmediatamente comenzó a parpadear a un ritmo de 0,5 segundos, lo que confirmó que el programa funcionó como se deseaba.

Multi-rate Interrupt-Driven Blink LED Timer Demonstration

- Esta demostración es una ampliación de la demostración anterior en la que el LED del usuario parpadeará repetidamente a dos velocidades diferentes, una a 0,5 s y la otra a una velocidad de 1,0 s.
  - En este programa también se utilizará una interrupción del temporizador, pero con una diferencia significativa entre las implementaciones de velocidad única y múltiple.
  - En el programa de velocidad única, el LED se controla directamente en la función de callback.
  - En la versión de múltiples velocidades, la función de callback simplemente incrementa una variable de contador, que luego se usa en el bucle eterno para controlar el LED.
  - La variable del contador incremental timCnt hace que el LED se active o restablezca dependiendo de su valor. La siguiente lista es la lista completa de main.c.También agregué comentarios adicionales sobre el fragmento de código del bucle permanente que sigue a la lista.

Multi-rate I… e Demonstra…

```
// STM disclaimer goes here
// Includes
#include "main.h"
#include "stm32f3xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

// Private variables
TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */
// Private variables
int timCnt = -1;
/* USER CODE END PV */
```

```
// Private function prototypes
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_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 peripherals, initialize the flash interface and 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();
  MX_TIM2_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

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

  /* USER CODE BEGIN 3 */
```

```
    if(timCnt == 0) // start the 0.5 second intervals
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 2)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }
        if( timCnt == 4)
        {
      HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 6)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }
        if(timCnt == 8)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 10)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }
        if(timCnt == 12)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 14)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }
        if( timCnt == 16) // start the 1.0 second intervals
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
         }
        if(timCnt == 20)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }
        if( timCnt == 24)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 28)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
     }

  /* USER CODE END 3 */
  } // end of while loop
} // end of main method
```

ven Blink LE

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

  // Initializes 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__);
  }

  // Initializes 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);
}

/* TIM2 init function */
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 63999;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 250; // now a 0.25 second period
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
      if(htim -> Instance == TIM2)
      {
        timCnt++;
          if(timCnt == 32) // 8 seconds overall
          {
              timCnt = -1;
          }
      }
}

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

n Blink LED Timer

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, 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 : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  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 ****/
```

Multi-rate Interrupt-Driven Blink LED Timer Demonstration

- El bucle eterno contiene una serie de declaraciones if que se activarán en función de valores de contador específicos.
  - Las declaraciones dentro de las declaraciones if configuran o restablecen el LED de usuario.
  - Cada incremento del contador tarda 0,25 s en completarse, lo que significa que dos incrementos tardarán 0,5 s.
  - El LED alternará dos veces por segundo durante los primeros 4 segundos.
  - El segundo conjunto de 4 s alternará el LED una vez por segundo.
- Este formato de código es bastante largo, pero intercambia espacio de código para simplificar.
- Es bastante fácil seguir el algoritmo paso a paso para determinar cómo se controla el LED utilizando este formato.

Multi-rate Interrupt-Driven Blink LED Time Demonstration

```
  while (1)
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
      if(timCnt == 0) // start the 0.5 second intervals
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 2)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
        if( timCnt == 4)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 6)
        {
         HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
        if(timCnt == 8)
        {
         HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 10)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
        if(timCnt == 12)
        {
       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 14)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
        if( timCnt == 16) // start the 1.0 second intervals
        {
       HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 20)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
        if( timCnt == 24)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
        }
        if(timCnt == 28)
        {
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
      }

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

Multi-rate Interrupt-Driven Blink LED Timer Demonstration

Test Run

- Construí el proyecto y luego lo subí a la placa Nucleo-64.
- El LED del usuario inmediatamente comenzó a parpadear a una velocidad de 0,5 s durante 4 s y luego a una velocidad de 1,0 s durante 4 s, lo que confirmó que el programa funcionó como se deseaba.

Modification to the Multi-rate Program

- Cambié el fragmento de código en el código permanente anterior para hacerlo mucho más compacto. Esa modificación se enumera a continuación.

```
while (1)
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
  if((timCnt % 2 == 0) && (timCnt < 16))
        {
              HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
        }
        if((timCnt % 4 == 0) && (timCnt >= 16))
        {
              HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
        }
  }
  /* USER CODE END 3 */
}
```

- El programa funcionó casi igual con este código modificado, que usaba el operador de módulo junto con un método de alternancia para controlar el LED.

Modification to the Multi-rate Program

- El código es mucho más pequeño que el original, pero es más complejo y más difícil de entender y, en consecuencia, de mantener.
- Este cambio destaca un desafío interesante al que se enfrentan los desarrolladores constantemente. Este desafío es si usar o no un código más simple, que puede ocupar más espacio de código, en lugar de usar un código compacto y “eficiente”, que probablemente sea más difícil de entender y modificar.
- Test Run
  - Prueba de funcionamiento
  - Construí el proyecto modificado y luego lo subí a la placa Nucleo-64.
  - El LED del usuario inmediatamente comenzó a parpadear a una velocidad de 0,5 s durante 4 s y luego a una velocidad de 1,0 s durante 4 s, lo que confirmó que el programa modificado funcionó igual que el programa original.

Modification to the Multi-rate Program

- Noté que el LED parecía parpadear más usando esta versión que la original. Sólo puedo atribuir el parpadeo al uso del método de alternancia frente a los métodos de configuración y reinicio firmes.
- Esta última demostración completa todas las discusiones y demostraciones que deseaba transmitir sobre interrupciones y temporizadores.
- Este capítulo realmente es sólo la punta del proverbial iceberg porque hay una enorme cantidad de información sobre estos temas que no pude cubrir.
- Solo el manual de usuario de HAL de STM tiene casi 2000 páginas. Como siempre, las hojas de datos de STM y los manuales de usuario y de referencia serán un recurso invaluable para conocer las muchas características y funciones adicionales que puede llevar a cabo con interrupciones y temporizadores.

Summary

- Comencé las discusiones del capítulo enfocándome en los temporizadores, que son periféricos importantes porque evitan que la MCU tenga que realizar directamente funciones de sincronización. Expliqué las cinco categorías disponibles de temporizadores STM, de los cuales hay tres tipos proporcionados en la MCU del proyecto. Estos son básicos, de propósito general (GP) y avanzados.
- A continuación, le mostré cómo configurar un temporizador para realizar un evento de sincronización específico. Esta discusión fue seguida por una demostración en la que el LED integrado parpadeó una vez por segundo. Esta demostración no utilizó una interrupción, que fue el foco de la segunda demostración del cronómetro.
- En la segunda demostración del temporizador, mostré lo fácil que era integrar una operación de temporización con un proceso de interrupción. En esta demostración también el LED integrado parpadeó, pero parpadeó usando solo el proceso de interrupción.

Summary

- La tercera demostración del temporizador mostró cómo reestructurar el proceso de interrupción de modo que se pudiera implementar un parpadeo del LED de múltiples velocidades. El método de callback de interrupción controlaba una variable de contador, que a su vez controlaba la activación real del LED dentro del bucle permanente.
- En la demostración final del temporizador, cambié el código original del bucle eterno por una versión mucho más compacta y compleja con la misma funcionalidad.
- Hice esto para ilustrar cómo los desarrolladores deben decidir constantemente cómo escribir código, más simple y consumiendo más espacio de código o muy compacto y complejo, que es más difícil de entender.

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