STM32 Bare-Metal Embedded C Drivers Mini Cookbook

Embedded Expert IO

TABLE OF CONTENT

INTRODUCTION : MUST READ

This is a mini-cookbook providing step-by-step instructions for writing bare-metal embedded-c peripheral drivers for the stm32f4 family of microcontrollers.

The solutions in this book have been tested on the stm32f411-nucleo development board. However, the solutions are expected to work the same way on all stm32f4 microcontrollers.

This book makes references to the RM0383 Reference Manual which is one of the official reference manuals provided for the stm32f4 microcontroller family by STMicroelectronics.

This document can be downloaded from this link:

https://www.st.com/resource/en/reference_manual/dm00119316-stm32f411xc-e-advanced-arm-based32-bit-mcus-stmicroelectronics.pdf

We also make references to the Cortex-M4 Generic User Guide which is the official Cortex-M4 guide provided by ARM Ltd.
This document can be downloaded from this link:

https://developer.arm.com/documentation/dui0553/latest/

This mini-cookbook is brought to you by
https://study.embeddedexpert.io/

EmbeddedExpertIO is an online embedded systems school focused on professional embedded systems software programming.

If you are new to embedded systems programming our community provides step-by-step courses that will take you from "blinky" to "build your own rtos".

If you are an embedded systems developer who wants to specialize in some specific aspect of embedded systems programming, we also provide a wide range of specialization courses to help you master different aspects of embedded firmware development.

We look forward to welcoming you to EmbeddedExpertIO. Visit us at : https://study.embeddedexpert.io/

INTRODUCTION : MUST DO

Watch this lesson collection for setup and live coding of this particular task. Link : Access Mini-Cook book Companion Video Lessons

TASK: WRITE A BARE-METAL DRIVER TO TOGGLE GPIOA PIN 5

```
#include "stm32f4xx.h"
int main(void) {
```

```
/*1. Enable GPIOA clock by Writing 1 to bit0 of AHB1ENR*/
RCC->AHB1ENR |=(1U<<0);
```

See Page 116 of RM0383 Reference manual

```
/*2. Set PA5 to output mode by writing 1 to bit10 of MODER*/
GPIOA->MODER |=(1U<<10);
while (1) {
```

See Page 156 of RM0383 Reference manual

```
/*3. Turn on PA5 by writing 1 to bit5 of ODR*/
GPIOA->ODR |=(1U<<5);
```

See Page 158 of RM0383 Reference manual

```
/*4. Delay for some time*/
for(int i =0; i<180000; i++){}
```

```
/*5. Turn off PA5 by writing 0 to bit5 of ODR*/
GPIOA->ODR &=~(1U<<5);
```

See Page 158 of RM0383 Reference manual

```
/*6. Delay for some time*/
for(int i =0; i<180000; i++) {}
        }
}
```

TASK: WRITE A BARE-METAL DRIVER TO READ INPUT FROM PC13 TO TOGGLE OUTPUT AT PA5

```
#include "stm32f4xx.h"
int main(void) {
```

```
/*1. Enable GPIOA clock by Writing 1 to bit0
of AHB1ENR*/
RCC->AHB1ENR |=(1U<<0);
```

```
/*2. Enable GPIOC clock by writing 1 to bit2
of AHB1ENR*/
RCC->AHB1ENR |=(1U<<2);
```

See Page 116 of RM0383 Reference manual

```
/*3. Set PA5 to output mode by writing 1 to
bit10 of MODER*/
GPIOA->MODER |=(1U<<10);
```

See Page 116 of RM0383 Reference manual

```
/*4. Set PC13 to input mode by writing 0 to
bit26 and bit27 of MODER*/
GPIOC->MODER |=(0U<<26);
GPIOC->MODER |=(0U<<27);
while (1) {
```

See Page 156 of RM0383 Reference manual

```
/*5. Check if input is high by checking if bit13
is 1*/
if(GPIOA->IDR & (1U<<13)) {
```

See Page 156 of RM0383 Reference manual

```
/*6. Turn off PA5 by writing 0 to bit5 of ODR*/
GPIOA->ODR &=~(1U<<5);
}
else {
```

```
/*7. Turn on PA5 by writing 1 to bit5 of ODR
*/
GPIOA->ODR |=(1U<<5);
      }
   }
 }
```

See Page 158 of RM0383 Reference manual

SYSTICK-Introduction

1. Found in all ARM Cortex-Microcontrollers, regardless of silicon manufacturer.
2. Used for taking actions periodically.
   Often used as time-base for real-time operating system.
3. The Systick is a 24-bit down counter driven by the processor clock.

SYSTICK-Counting

1. Counts from initial value down to zero
2. 24-bits imply maximum initial value of: 224 = 0xFFFFFF = 16,777,216
3. Initial value can be set to a value between 0x000000 to 0xFFFFFF

SYSTICK-Registers

1. Systick Current Value Register (STCVR)
   This register contains the current count value
2. Systick Control & Status Register (STCSR). This register allows us to configure the systick clock source,
   enable/disable interrupts and enable/disable the systick counter
3. Systick Reload Value Register (STRVR). This is where the initial count value is placed

SYSTICK-Count value computations

Compute the delay achieved by loading 11 in the Systick Reload Value Register (STRVR) given system clock = 16Mhz

Written as :

Systick->LOAD = 10

*written in the CMSIS standardt

*we write 10 although we want 11 because

the counter starts counting from 0

Solution

System clock = 16MHz = 16 000 000 cycles/second.

If 1 second executes 16 000 000 cycles, how many seconds execute 1 cycle ?

$$\frac{1}{16000000} = 62.5us = 62.5 \times 10\text{-}9\ s$$

Then 10 cycles => 10 x 62.5 x10-9 s = 625 x10-9 s = 625us

SYSTICK-Count value computations

System Clock (SYSCLK) is chosen as clock source.

If :

Systick->LOAD = N

$$\text{Delay achieved} = N \times \frac{1}{SYSCLK} = \frac{N}{SYSCLK}$$

SYSTICK-Delay computation

Compute N value for achieving a 1ms delay given SYSCLK as 16MHz

Solution

1ms = 0.001s

$$\text{Delay} = \frac{N}{SYSCLK}$$

$$0.001 = \frac{N}{16\ 000\ 000}$$

N = 0.001 x 16 000 000

N = 16000

SYSTICK-Documentation

Because SYSTICK is a core Cortex-M peripheral its references are found in the Cortex-M Generic User Guides provided by Arm

For more on systick you can download the Cortex-M4 Generic User Guide using this link:

Link Here

TASK: WRITE BARE-DRIVER FOR THE SYSTICK TIMER TO TOGGLE PA5 AT A RATE OF 1HZ I.E . ONCE EVERY SECOND.

```
#include "stm32f4xx.h"
int main(void) {
```

```
/*1. Enable GPIOA clock by Writing 1 to bit0 of AHB1ENR*/
RCC->AHB1ENR |=(1U<<0);
```

See Page 116 of RM0383 Reference manual

```
/*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
GPIOA->MODER |=(1U<<10);
```

See Page 156 of RM0383 Reference manual

```
/*3. Reload with number of clocks per second */
SysTick->LOAD = 16000000 - 1;
```

See Page 4-33 Cortex-M4 Generic User Guide

```
/*4.Clear Systick Current Value Register by writing any value to it*/*/
SysTick->VAL = 0;
```

```
/*5. Enable it, no interrupt, use system clock */
SysTick->CTRL = 0x5;
while (1) {
```

See Page 4-33 Cortex-M4 Generic User Guide

```
/*6. Wait for flag to be set if COUNT flag is set */
if (SysTick->CTRL & 0x10000) {
```

See Page 4-33 Cortex-M4 Generic User Guide

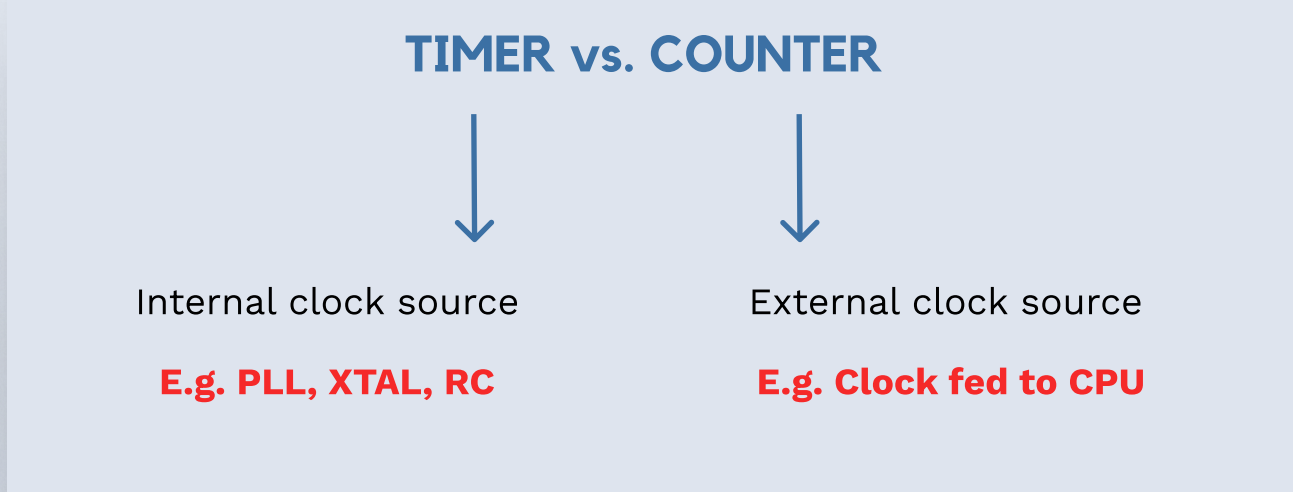
```
/*7. Toggle green LED */
GPIOA->ODR &=~(1U<<5);
        }
    }
}
```

See Page 116 of RM0383 Reference manual

TIMER-Uses

1. Counting Events
2. Creating Delays
3. Measuring time between event

TIMER- Timer vs. Counter



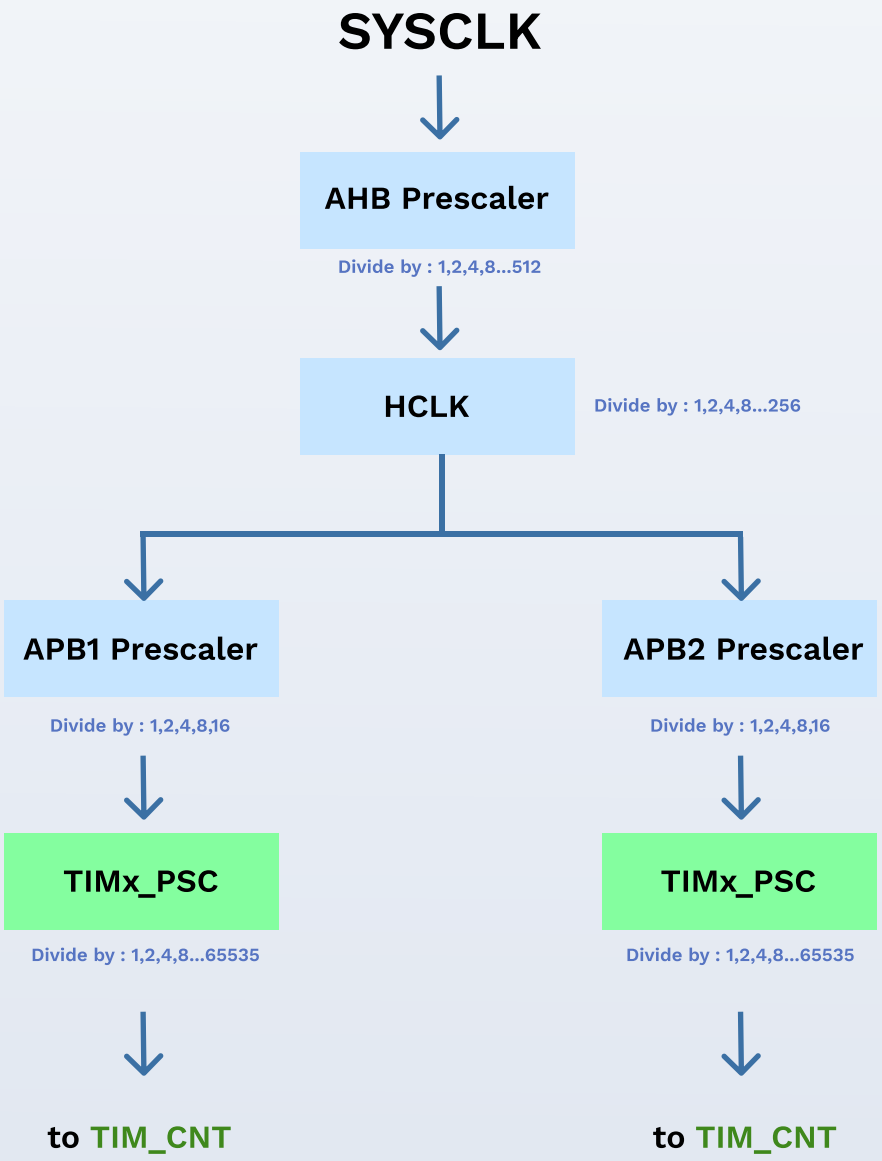
TIMER-STM32 Timers

1. Can be used as time base generator.
2. Can be used to measure the frequency of an external event – Input Capture Mode.
3. Control an output waveform, or to indicate when a period of time has elapsed - Output Compare Mode.
4. One pulse mode (OPM)- allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

TIMER-Registers

1. **Timer Count Register (TIMx_CNT)**
   Shows the current counter value. Size could be 32-bit or 16-bit depending on timer module used.
2. **Timer Auto-Reload Register (TIMx_ARR)**
   Timer raises a flag and the counter restarts automatically when counter value reaches the value in the auto-reload register. The counter is an up counter by default but can also be configured to be a down counter.
3. **Timer Prescaler Register (TIMx_PSC)**
   The prescaler slows down the counting speed of the timer by dividing the input clock of the timer.

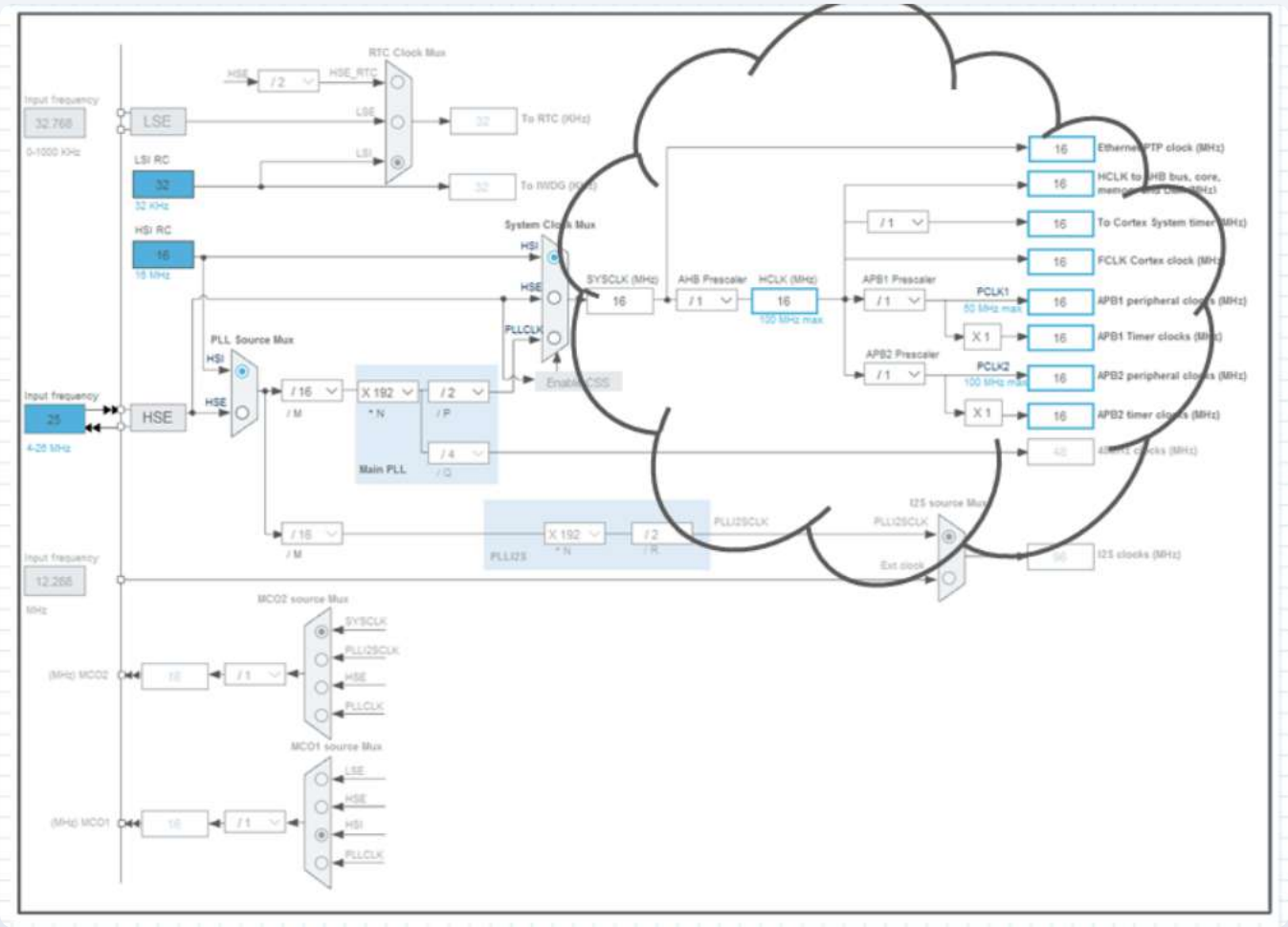
TIMER-Clock Pre-scaling



1. Timer prescaler (TIMx_PSC) determines how fast the timer counter(TIMx_CNT) increases/decreases.

2. With each change in the counter(TIMx_CNT) value, the new value is compared to the value in the timer auto-reload register (TIM_ARR), when the values match, a flag is raised and an interrupt occurs.

TIMER-Clock Pre-scaling



TIMER- Some Terms

1. **Update Event**
   When timeout occurs or how long it takes for flag to be raised

2. **Period**
   Value loaded into auto-reload register(TIM_ARR)

3. **Up counter**
   Counts from zero to a set value.

4. **Down counter**
   Counts from a set value down to zero

TIMER- Computing Update Event

$$\text{Update Event} = \frac{\text{Timer}_{\text{clock}}}{\text{(Prescaler+1)(Period+1)}}$$

Example

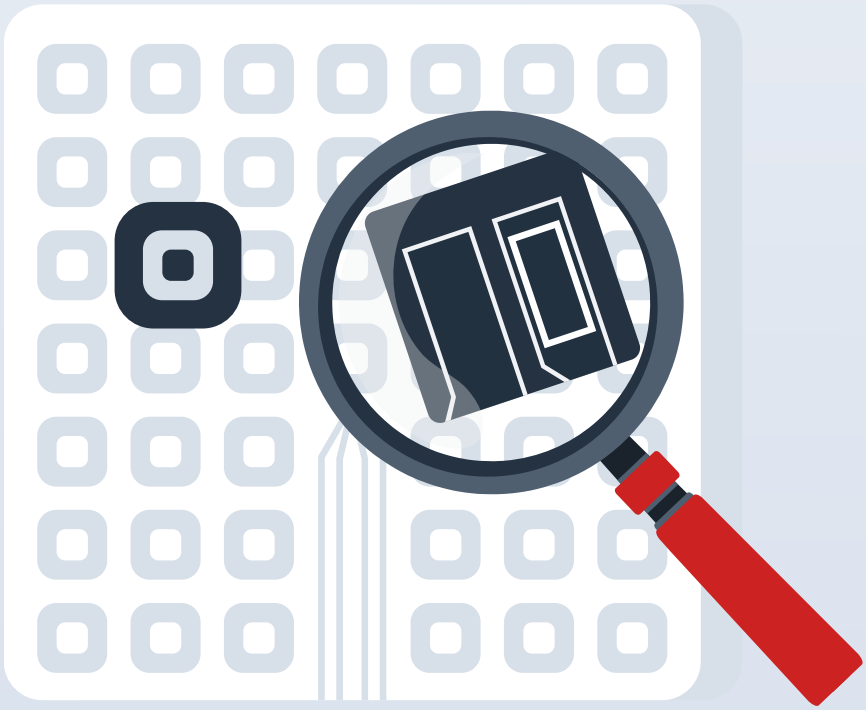
Let

Timer clock = APB1 clock = 48MHz

Prescaler = TIM_PSC value = 47999 + 1

Period = TIM_ARR value = 499 +1

$$\text{Update Event} = \frac{\text{48 000 000}}{\text{(47999+1)(499+1)}} = 2\text{Hz} = \frac{1}{2}\text{s} = 0.5\text{s}$$



TIM-Registers

TIM-Registers

1. **Prescaler (PSC)**

   -Prescaler value is put here

   Example

```
TIM2->PSC = 1600 -1;      // Set prescaler value to 1600
```

2. **Auto-Reload Register (ARR)**

   -Auto-reload value is put here

   Example

```
TIM2->ARR = 10000 ;       // Set prescaler value to 1600
```

3. **Control Register 1 (CR1)**

   -Enabling and disabling timer.

   Example

```
TIM2->CR1 = 1;            // Enable timer 2
```

4. **Status Register (SR)**

   -Checking, setting and clearing the flags of the timer

   Example

```
TIM2->SR & 1 ;            // Check update interrupt flag
TIM2->SR & = ~1 ;         // Clear update interrupt flag
```

5 **Capture/Compare Register (CCR1, CCR2, CCR4, CCR4)**

-One capture/compare register for each of the 4 channels.

Example

```
timestamp =TIM2->CCR1; // read captured value
```

6 **Capture Compare Mode Register 1 (CCMR1)**

-Configuring capture/compare functionality for CH1 and CH2.

7 **Capture Compare Mode Register 2 (CCMR2)**

-Configuring capture/compare functionality for CH3 and CH4.

Example

```
TIM2->CCMR1 = 0x41; // set CH1 to capture at every edge
```

8 **Capture/Compare Enable Register (CCER)**

Used to enable any of the timer channels either as input capture or output compare

Example

```
TIM2->CCER = 1;          // Enable channel 1
```

TASK: WRITE A BARE-METAL TIMER DRIVER TO TOGGLE PA5 AT A 1HZ RATE.

```
#include "stm32f4xx.h"
int main(void) {
```

```
/*1. Enable GPIOA clock by Writing 1 to bit0 of AHB1ENR*/
RCC->AHB1ENR |=(1U<<0);
```

See Page-116 of RM0383 Reference manual

```
/*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
GPIOA->MODER |=(1U<<10);
```

See Page-156 of RM0383 Reference manual

```
/*3. enable TIM2 clock */
RCC->APB1ENR |= (1U<<0);
```

See Page-117 of RM0383 Reference manual

```
/*4.Divide system clock by 1600*/
TIM2->PSC = 1600 - 1;
```

See Page-366 of RM0383 Reference manual

```
/*5. Divide the remainder by 10000*/
TIM2->ARR = 10000 - 1;
```

See Page-366 of RM0383 Reference manual

```
/*6. Clear Timer counter*/
TIM2->CNT = 0;
```

See Page-366 of RM0383 Reference manual

```
/*7. Enable TIM2*/
TIM2->CR1 = 1;
   while (1) {
```

See Page-351 of RM0383 Reference manual

```
/*8. Wait until UIF sett */
      while (! (TIM2->SR & 1)) {}
```

See Page-351 of RM0383 Reference manual

```
/*9. Clear UIF*/
      TIM2->SR &= ~1;
```

See Page-351 of RM0357 Reference manual

```
/*10. Toggle PA5*/
      GPIOA->ODR ^= (1U<<5);
      }
   }
}
```

See Page-158 of RM0383 Reference manual

ANALOG-TO-DIGTAL CONVERTER (ADC)

ADC Independent Modes

1. Single-channel, single conversion mode.
2. Multichannel(scan), single conversion mode.
3. Single-channel continuous conversion mode.
4. Multichannel continuous conversion mode.
5. Injected continuous conversion mode.

Single-channel, single conversion mode

1. **Simplest ADC mode.**
2. **ADC performs a single conversion of a single channel x and stops after conversion is complete.**

Example use case

Measurement of voltage level to determine if a system should be started on not.

ADC-Registers

1. **Control Register 1 (CR1)**
   -For setting the ADC resolution.

2. **Data Register (DR)**
   -Stores converted results.

3. **Control Register 2 (CR2)**
   -For enabling/disabling the ADC
   -Set trigger type

   Example

   ```
   ADC1->CR2 = 1 ;      // Enable channel 1
   ```

4. **Sampling Time Register 1 (SMPR1)**
   -For setting the number of clocks cycles for sampling time for CH10 - CH18.

5. **Sampling Time Register 2 (SMPR2)**
   -For setting the number of clocks cycles for sampling time for CH0 - CH9.

6. **Sequence Register 1 (SQR1)**
   -Setting total number of conversions

   Example

   ```
   ADC1->SQR1 = 0 ;      // set sequence length to 1
   ```

7 **Sequence Register 2 (SQR2)**

-Setting conversions sequence rank.

8 **Sequence Register 3 (SQR3)**

-Set conversion start channel.

TASK: WRITE A BARE-METAL ADC DRIVER TO CONVERT ANALOG INPUT FROM PA1 INTO A GLOBAL VARIABLE.

```
#include "stm32f4xx.h"
int main(void) {
```

```
/*1. Enable GPIOA clock by Writing 1 to bit0 of AHB1ENR*/
RCC->AHB1ENR |=(1U<<0);
```

See Page-116 of RM0383 Reference manual

```
/*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
GPIOA->MODER |=(1U<<2);
GPIOA->MODER |=(1U<<3);
```

See Page-156 of RM0383 Reference manual

```
/*3. enable ADC1 clock */
RCC->APB1ENR |= (1U<<8);
```

See Page-117 of RM0383 Reference manual

```
/*4.Set ADC to SW trigger and disable ADC before configuring it*/
ADC1->CR2 = 0;
```

See Page-228 of RM0383 Reference manual

```
/*5. Set start of conversion sequence to ch 1*/
ADC1->SQR3 = 1;
```

See Page-235 of RM0383 Reference manual

```
/*6. Set conversion sequence length to 1*/
ADC1->SQR1 = 0;
```

See Page-234 of RM0383 Reference manual

```
/*7. Enable ADC1 */
ADC1->CR2 |= 1;
while (1) {
```

See Page-230 of RM0383 Reference manual

```
/*8. Start a conversion*/
ADC1->CR2 |= 0x40000000;
```

See Page-230 of RM0383 Reference manual

```
/*9. Wait for conv complete*/
while(!(ADC1->SR & 2)) {}
```

See Page-227 of RM0383 Reference manual

```
/*10. Read conversion result */
result = ADC1->DR;
        }
    }
}
```

See Page-237 of RM0383 Reference manual

UART-Registers

1 **Baud Rate Register (BRR)**

- For setting uart baud rate.

Example

```
USART2->BRR = 0x0683;     //9600 baud operating at 16 MHZ
```

2 **Control Register 1 (CR1)**

-Setting transmission direction.

-Setting data size.

-Enabling and disabling uart.

Example

```
USART2->CR1 = 0x0008;      // 8-bit data, enable TX
USART2->CR1 |= 0x2000;     //enable uart2
```

3 **Control Register 2 (CR2)**

-Setting stop bit.

Example

```
USART2->CR2 = 0x0000;      // 1 stop bit
```

4 **Control Register 3 (CR3)**

-Setting flow control.

Example

```
USART2->CR3 = 0x0000;      // no flow control
```

5 **Status Register (SR)**

-Checking the state of rx and tx buffers.

Example

```
USART2->SR & 0x0080;      // check if tx buffer
```

TASK: WRITE A BARE-METAL DRIVER TO CONFIGURE PA2 AS UART2 TX AND PA3 AS UART2 RX. TEST CAN BE RUN WITH A TERMINAL EMULATION PROGRAM SUCH AS TERA TERM OR REALTERM.

```
#include "stm32f4xx.h"
#include <stdio.h>

void uart2_init(void);
int uart2_write(int c);
int uart2_read(void);

int main(void) {
        int number;
        char sentence[100];
        uart2_init();
        printf("Hello from STM32!! \n\r");
        while (1) {
                printf("Please enter a number: ");
                scanf("%d", &number);
                printf("the number entered is: %d\r\n", number);
                printf("please type a character string: ");
                gets(sentence);
                printf("the character string entered is: ");
                puts(sentence);
                printf("\r\n");
        }
  }
```

```
/* Initialize USART2 to transmit at
9600 Baud */
void uart2_init (void) {
```

`/*1. Enable GPIOA clock by writing 1 to bit0 of AHB1ENR*/` `RCC->AHB1ENR |= (1U<<0);`	See Page-116 of RM0383 Reference manual
`/*2. Enable USART2 clock by writing 1 to bit17 of APB1ENR*/` `RCC->APB1ENR |= (1U<<17);`	See Page-117 of RM0383 Reference manual
`/*3. Enable alt7 for USART2 */` `GPIOA->AFR[0] |= 0x7700;`	See Page-149 of RM0383 Reference manual
`/*4. Enable alternate function for PA2, PA3 */` `GPIOA->MODER |= 0x00A0;`	See Page-156 of RM0383 Reference manual
`/*5. Set UART: 9600 baud @ 16 MHz */` `USART2->BRR = 0x0683;`	See Page-551 of RM0383 Reference manual
`/*6. Enable TX, RX, 8-bit data */` `USART2->CR1 = 0x000C;`	See Page-551 of RM0383 Reference manual
`/*7. Set UART :1 stop bit */` `USART2->CR2 = 0;`	See Page-554 of RM0383 Reference manual
`/*8. Set UART: No flow control */` `USART2->CR3 = 0;`	See Page-555 of RM0383 Reference manual

```
/*9. Enable USART2 */
USART2->CR1 |= 0x2000;
}
```

See Page-551 of RM0383 Reference manual

```
/* Write a character to USART2 */
int uart2_write (int ch) {
    /*10. wait until Tx buffer empty*/
    while (!(USART2->SR & 0x0080)) {}
    /*11. Write character to DR */
    USART2->DR = (ch & 0xFF);
    return ch;
}
```

See Page-548 of RM0383 Reference manual

See Page-551 of RM0383 Reference manual

```
/* Read a character from USART2 */
int uart2_read(void) {
    /*12. Wait until character arrives*/
    while (!(USART2->SR & 0x0020)) {}
    /*13. Return the received character */
    return USART2->DR;
    }
```

See Page-548 of RM0383 Reference manual

See Page-551 of RM0383 Reference manual

```
/*Interface to the c standard I/O library*/
struct __FILE { int handle; };
FILE __stdin = {0};
FILE __stdout = {1};
FILE __stderr = {2};
/*fgetc is called by c library console input.
The function will echo the character received*/
```

```
int fgetc(FILE *f) {
    int c;
    /*1. read the character from console */
    /*2. If '\r', after it is echoed, a '\n' is appended*/
    if (c == '\r') {
        uart2_write(c);      /* echo */
        c = '\n';
    }

    /*3. Echoe*/
    uart2_write(c);

    /*4. Return character*/
    return c;
}
```

```
/*fputc is called by c library console output. */
int fputc (int c, FILE *f) {
        /*5. Write the character to console */
        return uart2_write(c);
}
```

Our Courses

ARM Assembly Programming Mastery Pack

Covering ARM Systems Design, Architecture and Practical Assembly Programming, this is the most comprehensive ARM ..

Learn More

Bare-Metal C/C++ Learning Path

1. Modern Bare-Metal Embedded-C From Ground Up (STM32F4) : Old and New Edition
2. Modern Bare-Metal ..

Learn More

Bluetooth Low Energy (BLE) From Ground Up™

Welcome to the Bluetooth Low Energy (BLE) From Ground Up™ course.
This practical Bluetooth Low Energy ..

Learn More

Embedded Ethernet Firmware Development Learning Path

3 Courses | 43+ Courses | Complete Source Included
1. Embedded Ethernet ..

Learn More

Embedded Systems IoT Learning Path

3 Courses | 36+ Courses | Complete Source Included

1. Bluetooth Low Energy (BLE) from Ground Up ....

Learn More

Embedded Wifi Bare-Metal Development From Ground Up™

Welcome to the Embedded WIFI Bare-Metal Development From Ground Up™ course..

Learn More

Extreme Embedded Firmware Engineering Learning Path

3 courses | 44+ hours | Complete Source Code Included

1. Embedded Build ..

Learn More

STM32 Development Learning Path

8 Courses | 90+ Courses | Complete Source Included

1. Mastering STM32CubeMX 5 and CubeIDE

2. Embedded Systems..

Learn More

Realtime Operating Systems (RTOS) Learning Path

4 Courses | 47+ hours | Complete Source Included

1. FreeRTOS from Ground Up
2. Arm Assembly Programming..

Learn More

STM32F3 Bare-Metal Peripheral Drivers Development

Welcome to the STM32F3 Bare-Metal Peripheral Drivers Programming course ..

Learn More

ARM GNU Assembly Programming From Ground Up™

Welcome to the ARM GNU Assembly Programming From Ground Up™ course ..

Learn More

Embedded Ethernet Essential Training With CubeMX

This course is the beginner course of a 3 course learning path teaching you how ..

Learn More

Embedded Systems Bare-Metal Programming Ground Up™ (STM32F4)

The goal of this course is to teach you how to navigate the microcontroller reference manual ..

Learn More

Embedded Systems STM32 HAL APIs Driver Development

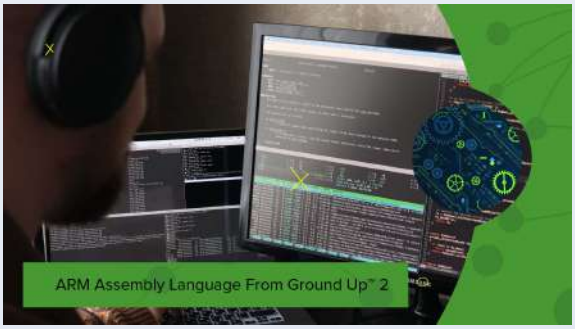
Welcome to the Embedded Systems STM32 Hardware Abstraction Layer (HAL) ..

Learn More

Embedded Systems STM32 Low-Layer APIs(LL) Driver Development

Welcome to the Embedded Systems STM32 Low-Layer APIs(LL) Driver Development course. .

Learn More



ARM Assembly Language From Ground Up™ 2

Welcome to the ARM Assembly Programming Ground Up™ 2 course. With a programming based approach, this course is designed ..

Learn More

Mastering STM32CubeMX 5 And CubeIDE - Embedded Systems

Hello Welcome to the Mastering STM32CubeMX 5 and CubeIDE course This course teaches you ..

Learn More

{C++}Build Your Own Realtime OS (RTOS) From Ground Up™ On ARM

Welcome to the {C++} Build Your Own RTOS From Ground Up™ course. This is a C++ version of..

Learn More

Embedded System IoT Systems Design

This course teaches you how build a complete Internet-of-Thing (IoT) system from scratch using just your development board ..

Learn More

Embedded Systems Bare-Metal Ethernet Programming

This course is the advanced level course of a 3 course learning path teaching you how to ..

Learn More

Embedded Systems Cellular Firmware Development(GSM)

This course teaches you how to develop drivers and libraries for adding cellular functionality to your embedded device. This course uses the STM32 ...

Learn More

Modern Bare-Metal Embedded C++ Programming From Ground Up™

Welcome to the Modern Embedded C++ Bare Metal course. This is a practical programming ..

Learn More

Embedded Systems Design Patterns From Ground Up™

Hello, welcome to the "Embedded Systems Design Patterns " course. This course teaches you how to apply design patterns to embedded firmware development. Design ..

Learn More

Embedded Ethernet Programming With HAL

This course is the intermediate level course of a learning path teaching you how to write/configure ..

Learn More

Deep Learning On ARM Processors - From Ground Up™

We are going to embark on a very exciting journey together. We are going to learn how to build deep neural networks from scratch..

Learn More

Build Your Own RealTime OS (RTOS 1) From Ground Up™ On ARM 1

This course teaches you how to build a Real-Time Operating Systems through intensive ..

Learn More

Build Your Own RealTime OS (RTOS 2) From Ground Up™ On ARM 2

Welcome to the Build Your Own RealTime OS (RTOS) From Ground Up™ on ARM 2 course ..

Learn More

FreeRTOS From Ground Up™ On ARM Processors

This course teaches you the foundations of real-time systems and how to build real-time applications using FreeRTOS ,one of the most popular real-time ..

Learn More

Embedded Systems Object-Oriented Programming In C

Welcome to the Embedded Systems Object-Oriented Programming course. This course is for anyone seeking to improve their ..

Learn More

Practical Low Cost Bare-Metal Bluetooth Development

Hello, welcome to the “Practical Low Cost Bare-Metal Bluetooth Development” course. ..

Learn More

Embedded Google Cloud <> Python Gateway Communication

Get Ready To Embark On A Transformative Journey With Our Practical Course That

Learn More

Modern Embedded GUI With TouchGFX

Introducing Modern Embedded GUI With TouchGFX. This Course Will Equip You With The Skills And Knowledge Needed

Learn More

Firmware Version Control With Git From Ground Up™

We shall delve into the world of Version Control Systems (VCS). We start by introducing ..

Learn More

USB Host Development Essential Training With CubeMX

This course complements our USB Device Development Essential Training, offering a holistic ..

Learn More

WiFi IoT Architecture: From Firmware To Full Stack Web Development

Welcome to the WiFi IoT Architecture course. This course is designed to transform you into a ..

Learn More

4G LTE IoT: Bare-Metal To HTTP, MQTT, SMS

Welcome to 4G LTE IoT: Bare-Metal to HTTP, MQTT, SMS, an immersive journey crafted to transform ...

Learn More

Flash Memory And EEPROM Drivers: A Hands-On Guide For Embedded Engineers

Are you an Embedded Engineer looking to master the fundamentals of memory storage and ..

Learn More

Advanced Digital Signal Processing On ARM Processors

Welcome to the “Advanced Digital Signal Processing on ARM Processors” course. Whether ..

Learn More

Embedded Systems Cryptography & Encryption

In the era of interconnected devices, every micro-bit of data is both an asset and a vulnerability..

Learn More

USB Device Development Essential Training With CubeMX

Discover the Art of USB Device Development: Harness the Power of Universal Connectivity

Learn More

Embedded Local Database Storage: MySQL

Enter the world of embedded database storage in our new course, "Embedded Local Database Storage: MySQL".

Learn More

Embedded Azure Cloud <> Python Gateway Communication

Step into the fascinating world of Microsoft Azure with this practical course designed to empower you to

Learn More

Embedded AWS Cloud <> Python Gateway Communication

This course seamlessly merges the realms of embedded systems and Amazon Web Services (AWS) ..

Learn More

Embedded Memory Security: Protecting Your System From Tampering And Unauthorized Access

Are you looking to take your embedded systems protection ..

Learn More

Custom Cloud <> Python Gateway Communication

Are you ready to redefine the future with IoT without the complexity of wireless radios?

Learn More

Embedded Audio Solutions: Developing An Audio Media Player

Welcome to the "Embedded Audio Media Player" course, your quickest way to developing a complete

Learn More

Master Firmware Updates With In-Application Programming(IAP)

you an embedded systems enthusiast or a professional engineer looking to level up your skills and

Learn More

Embedded Bootloader Development From Ground Up™

Get ready to dive into the exciting world of bootloader development with this beginner level course of our

Learn More

HAPPY CODING

Embedded
Expert IO