STM32

Bare-Metal Embedded-C Drivers

Mini Cookbook

TABLE OF CONTENTS

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-based-32-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-Cookbook 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);

/*2.  Set PA5 to output mode by writing 1 to bit10 of MODER*/

GPIOA->MODER |=(1U<<10);

while (1) {

/*3. Turn on PA5 by writing 1 to bit5 of ODR*/

GPIOA->ODR |=(1U<<5);

/*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);

/*6. Delay for some time*/

for(int i =0; i<180000; i++) {}

      }

}
```

See Page116 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page158 of RM0383 Reference manual

See Page158 of RM0383 Reference manual

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);
    /*3.  Set PA5 to output mode by writing 1 to bit10 of
     MODER*/
    GPIOA->MODER |=(1U<<10);
   /*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) {
    /*5. Check if input is high by checking  if bit13 is 1*/
     if(GPIOA->IDR & (1U<<13)) {
    /*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 Page116 of RM0383 Reference manual

See Page116 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page158 of RM0383 Reference manual

SYSTICK-*Introduction*

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

SYSTICK-*Counting*

- **Counts from initial value down to zero**
- **24-bits imply maximum initial value of:**
  2^{24} = 0xFFFFFF = 16,777,216
- **Initial value can be set to a value between**
  0x000000 to 0xFFFFFF

SYSTICK-*Registers*

- **Systick Current Value Register (STCVR)**

  This register contains the current count value

- **Systick Control & Status Register (STCSR)**

  This register allows us to configure the systick clock source, enable/disable interrupts and enable/disable the systick counter

- **Systick Reload Value Register (STRVR)**

  This is where the initial count value is placed

SYSTICK-*Count value computations*

Compute the delay achieved by **loading 10** in the Systick Reload Value Register (STRVR) given system clock = 16Mhz

Written as :

```
Systick->LOAD  =  9
```

```
*written in the CMSIS standard
*we write 9 although we want 10 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** ?

$\Rightarrow \frac{1}{16000000} = 62.5ns = 62.5 \times 10^{-9}\ s$

Then 10 cycles $\Rightarrow 10 \times 62.5 \times 10^{-9}\ s = 625 \times 10^{-9}\ s = 625ns$

SYSTICK-*Count value computations*

System Clock (**SYSCLK**) is chosen as clock source.

If :

```
Systick->LOAD  =  N
```

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

SYSTICK-*Delay computation*

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

Solution

$$1\text{ms} = 0.001\text{s}$$

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

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

$$N = 0.001 \times 16\,000\,000$$

$$N = 16\,000$$

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 systickyou can download the Cortex-M4 Generic User Guide using this link:

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

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);
    /*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
    GPIOA->MODER |=(1U<<10);
    /*3. Reload with number of clocks per second */
    SysTick->LOAD = 16000000 - 1;
   /*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) {
     /*6. Wait for flag to be set if COUNT flag is set */
      if (SysTick->CTRL & 0x10000) {
     /*7. Toggle green LED */
     GPIOA->ODR &=~(1U<<5);
      }
  }
}
```

See Page116 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page 4-33 Cortex-M4 Generic User Guide

See Page 4-33 Cortex-M4 Generic User Guide

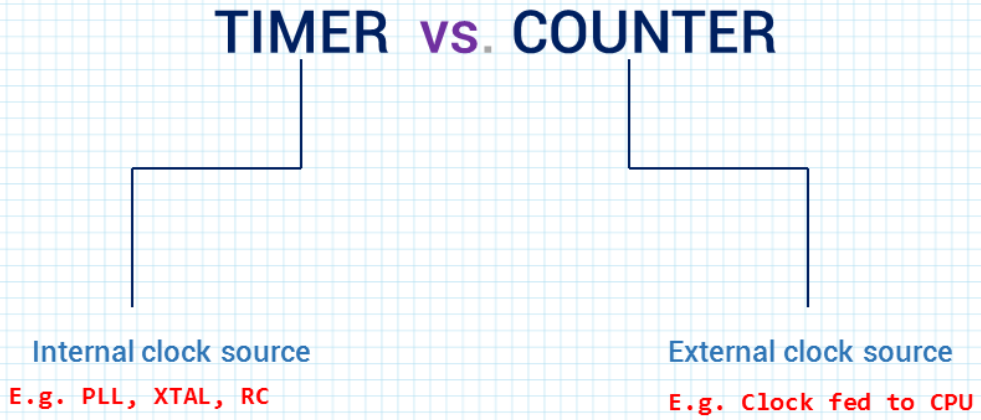
See Page 4-33 Cortex-M4 Generic User Guide

See Page158 of RM0383 Reference manual

TIMER-*Uses*

- **Counting Events**
- **Creating Delays**
- **Measuring time between event**

TIMER- *Timer vs. Counter*



TIMER-*STM32 Timers*

- **Can be used as time base generator**
- **Can be used to measure the frequency of an external event –** *Input Capture Mode*
- **Control an output waveform, or to indicate when a period of time has elapsed -** *Output Compare Mode*
- ***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*

- **Timer Count Register (TIMx_CNT)**

  Shows the current counter value. Size could be 32-bit or 16-bit depending on timer module used.

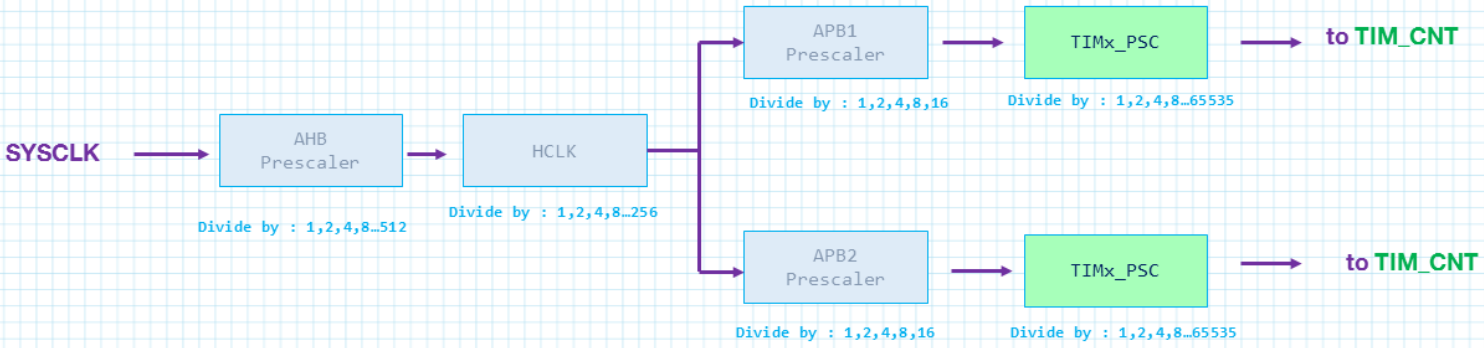
- **Timer Auto-Reload Register (TIMx_ARR)**

  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.

- **Timer Prescaler Register (TIMx_PSC)**

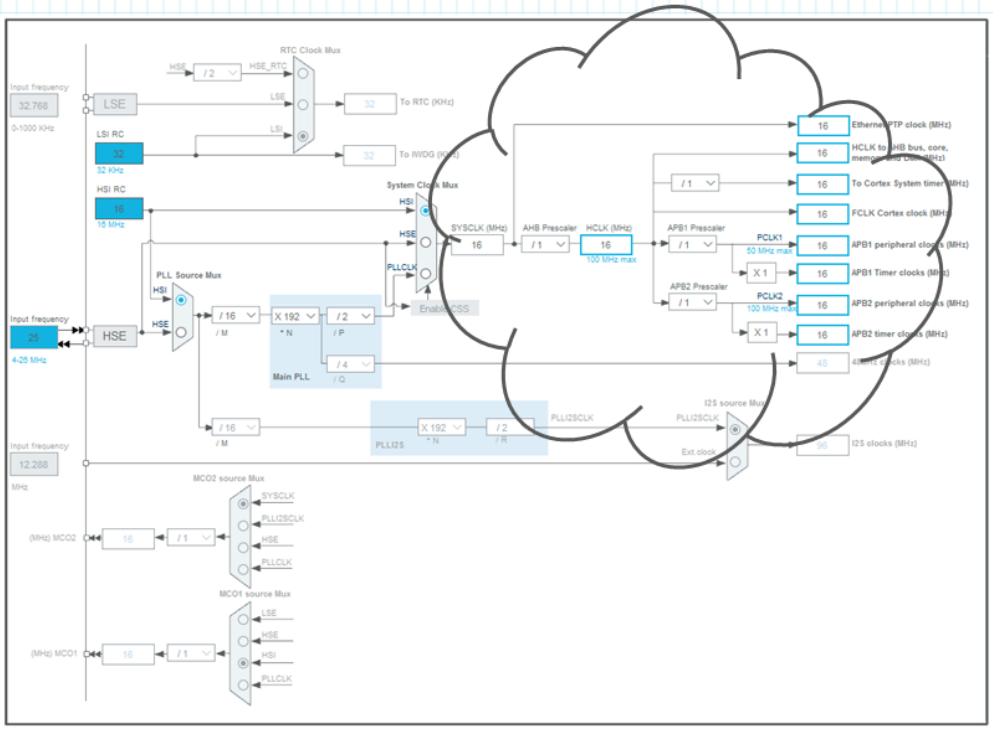
  The prescaler slows down the counting speed of the timer by dividing the input clock of the timer

TIMER-*Clock Pre-scaling*



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

- **Update Event**

  When timeout occurs or how long it takes for flag to be raised

- **Period**

  Value loaded into auto-reload register(TIM_ARR)

- **Up counter**

  Counts from zero to a set value.

- **Down counter**

  Counts from a set value down to zero

TIMER- *Computing Update Event*

$$Update\ Event = \frac{Timer_{clock}}{(Prescaler+1)(Period+1)}$$

Example

Let

Timer clock = APB1 clock = 48MHz
Prescaler = TIM_PSC value = 47999 + 1
Period = TIM_ARR value = 499 +1

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

TIM-*Registers*

- **Prescaler (PSC)**
  - *Prescaler value is put here*

EXAMPLE

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

- **Auto-Reload Register (ARR)**
  - *Auto-reload value is put here*

EXAMPLE

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

- **Control Register 1 (CR1)**
  - *Enabling and disabling timer*

EXAMPLE

```
TIM2->CR1 = 1 ;   // Enable timer2
```

- **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
```

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

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

EXAMPLE

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

- **Capture Compare Mode Register 1 (CCMR1)**
  - *Configuring capture/compare functionality for CH1 and CH2*
- **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
```

- **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);
    /*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
    GPIOA->MODER |=(1U<<10);
    /*3. enable TIM2 clock */
    RCC->APB1ENR |= (1U<<0);
   /*4.Divide system clock by 1600*/
  TIM2->PSC = 1600 - 1;
  /*5. Divide the remainder by 10000*/
  TIM2->ARR = 10000 - 1;
  /*6. Clear Timer counter*/
   TIM2->CNT = 0;
   /*7. Enable TIM2*/
    TIM2->CR1 = 1;
  while (1) {
     /*8. Wait until UIF sett */
      while (! (TIM2->SR & 1)) {}
     /*9. Clear UIF*/
      TIM2->SR &= ~1;
     /*10. Toggle PA5*/
     GPIOA->ODR ^= (1U<<5);
      }
  }
}
```

See Page116 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page117 of RM0383 Reference manual

See Page366 of RM0383 Reference manual

See Page366 of RM0383 Reference manual

See Page366 of RM0383 Reference manual

See Page351 of RM0383 Reference manual

See Page351 of RM0383 Reference manual

See Page351 of RM0357 Reference manual

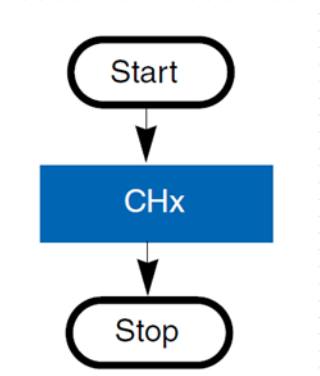
See Page158 of RM0383 Reference manual

ADC Independent Modes

- Single-channel, single conversion mode
- Multichannel(scan), single conversion mode
- Single-channel continuous conversion mode
- Multichannel continuous conversion mode
- Injected continuous conversion mode

Single-channel, single conversion mode

- Simplest ADC mode
- ADC performs a single conversion of a single channel x and stops after conversion is complete

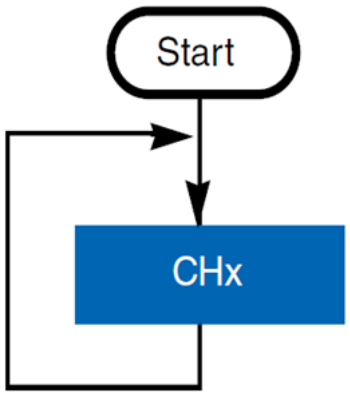


Example use case

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

Single-channel, continuous conversion mode

- Used to convert a single channel continuously
- Works in the background without intervention from the CPU

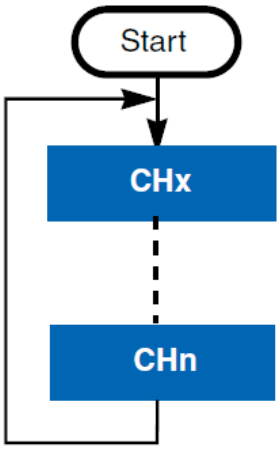


Example use case

Measurement of room temperature continuous to adjust air-conditioner

Multichannel, continuous conversion mode

- Used to convert multiple channels continuously
- Up to 16 different channels with different sampling times can be converted on the stm32f4

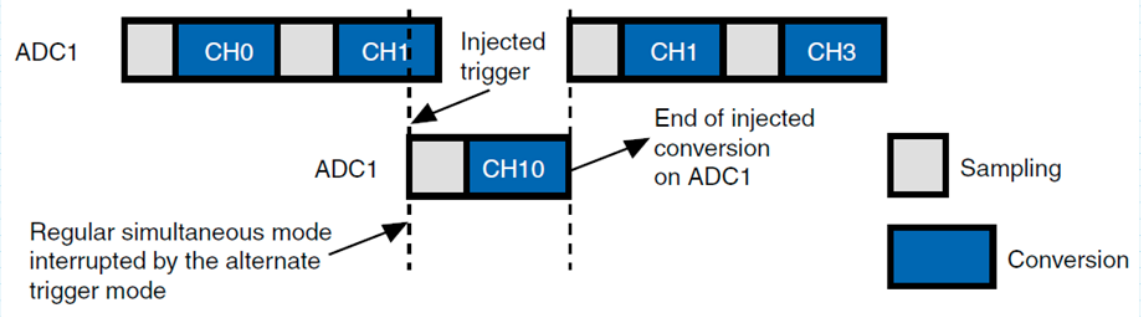


Example use case

Continuously measuring multiple accelerometers to adjust joints of a robotic arm.

Injected conversion mode

- Intended for use when conversion is triggered by an external event or by software.
- The injected group has priority over the regular channel group.
- Interrupts the conversion of the current channel in the regular channel group.



Example use case

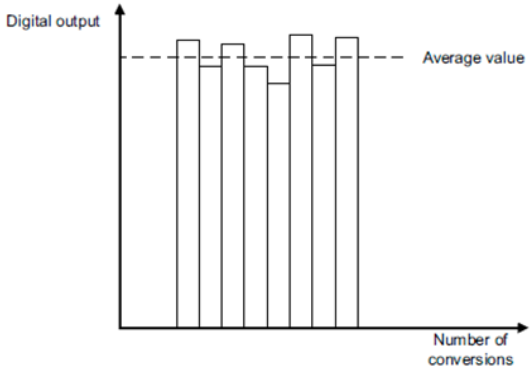
For synchronizing the conversion of channels to an event

Improving ADC Accuracy

- **Averaging N ADC samples**
- **Averaging N-X ADC sample**

Averaging N ADC samples

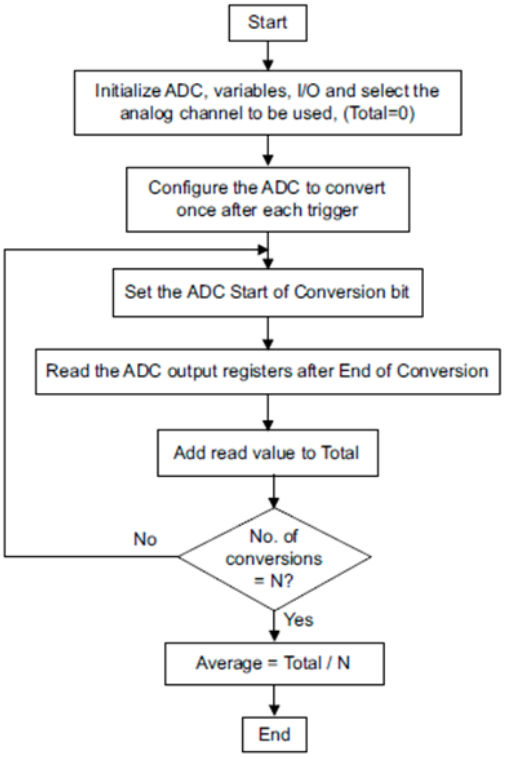
- **Collect samples in multiple of 2, meaning N should be a multiple of 2**
  - ✓ **This saves CPU time and code memory required for computing division**
  - ✓ **Division can be performed by right-shifting the sum of the values, this takes 1 CPU cycle**



Averaging N ADC samples

Total conversion time =
(number of samples * ADC conversion time) + computation time.

Computation time =
time taken to read the results, add them together and calculate the average by dividing the total by the number of samples.



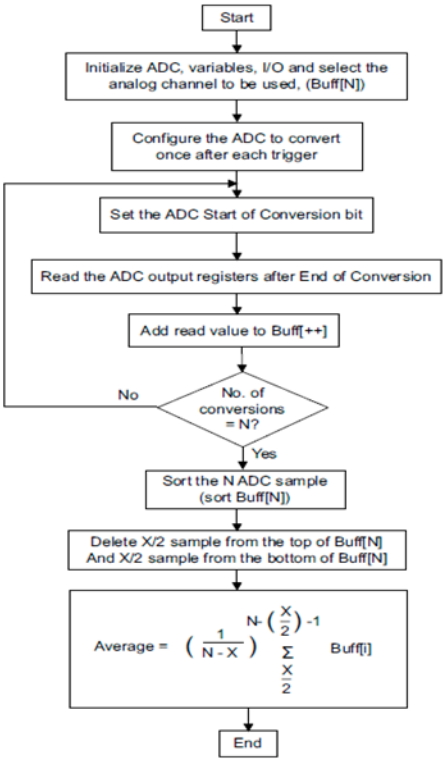
Averaging N-X ADC samples

- Based on taking N ADC samples, sorting them from the highest to the lowest value (or the reverse) and deleting the dispersed X samples.
- It is recommended to choose N and X as multiples of 2.
- This averaging method is more efficient than the previous one, as it deletes the most dispersed values which could impact the average, and it gives a good trade-off between the execution time and the conversion accuracy.

Averaging N-X ADC samples

Total conversion time =
(number of samples * ADC conversion time) + computation time.

Computation time =
time taken to read the results, sort the N ADC samples, delete the X most dispersed samples, add the rest of the ADC samples together and calculate the average by dividing the total by N-X.



- **Control Register 1 (CR1)**
  - *For setting the ADC resolution*
- **Data Register (DR)**
  - *Stores converted results*

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

EXAMPLE

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

- **Sampling Time Register 1 (SMPR1)**
  - *For setting the number of clocks cycles for sampling time for CH10 – CH18*
- **Sampling Time Register 2 (SMPR2)**
  - *For setting the number of clocks cycles for sampling time for CH0 – CH9*

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

EXAMPLE

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

- **Sequence Register 2 (SQR2)**
  - *Setting conversions sequence rank*
- **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);
    /*2.Set PA5 to output mode by writing 1 to bit10 of MODER*/
    GPIOA->MODER |=(1U<<2);
    GPIOA->MODER |=(1U<<3);
     /*3. enable ADC1 clock */
     RCC->APB1ENR |= (1U<<8);
    /*4.Set ADC to SW trigger and disable
     ADC before configuring it*/
     ADC1->CR2 = 0;
    /*5. Set start of conversion sequence to ch 1*/
     ADC1->SQR3 = 1;
   /*6. Set conversion sequence length to 1*/
    ADC1->SQR1 = 0;
    /*7. Enable ADC1 */
     ADC1->CR2 |= 1;
   while (1) {
     /*8. Start a conversion*/
      ADC1->CR2 |= 0x40000000;
     /*9. Wait for conv complete*/
      while(!(ADC1->SR & 2)) {}
     /*10. Read conversion result */
      result = ADC1->DR;
      }
  }
}
```

See Page116 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page117 of RM0383 Reference manual

See Page228 of RM0383 Reference manual

See Page235 of RM0383 Reference manual

See Page234 of RM0383 Reference manual

See Page230 of RM0383 Reference manual

See Page230 of RM0383 Reference manual

See Page227 of RM0383 Reference manual

See Page237 of RM0383 Reference manual

- **Baud Rate Register (BRR)**

  - For setting uart baud rate

EXAMPLE

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

- **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
```

- **Control Register 2 (CR2)**

  - *Setting **stop bit***

EXAMPLE

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

- **Control Register 3 (CR3)**
  - *Setting **flow control***

EXAMPLE

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

- **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 Page116 of RM0383 Reference manual

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 Page116 of RM0383 Reference manual

```
/*2. Enable USART2 clock by writing 1 to bit17 of APB1ENR*/
RCC->APB1ENR |= (1U<<17);
/*3. Enable alt7 for USART2 */
GPIOA->AFR[0] |=  0x7700;
/*4. Enable alternate function for PA2, PA3 */
GPIOA->MODER |= 0x00A0;
/*5. Set UART: 9600 baud @ 16 MHz */
USART2->BRR = 0x0683;
/*6. Enable TX, RX, 8-bit data */
USART2->CR1 = 0x000C;
/*7. Set UART :1 stop bit */
USART2->CR2 = 0;
/*8. Set UART: No flow control */
USART2->CR3 = 0;
/*9. Enable USART2 */
USART2->CR1 |= 0x2000;
}

/* 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;
}

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

See Page117 of RM0383 Reference manual

See Page149 of RM0383 Reference manual

See Page156 of RM0383 Reference manual

See Page551 of RM0383 Reference manual

See Page551 of RM0383 Reference manual

See Page554 of RM0383 Reference manual

See Page555 of RM0383 Reference manual

See Page551 of RM0383 Reference manual

See Page548 of RM0383 Reference manual

See Page551 of RM0383 Reference manual

See Page548 of RM0383 Reference manual

```
/*13. Return the received character */
   return USART2->DR;
}

/*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);
}
```

See Page551 of RM0383 Reference manual

Happy Coding

study.embeddedexpert.io