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/**

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

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

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

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

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