Códigos de ejemplo de STM32

Encendido de un led por botón

```
#include "stm32f4xx.h"
int main(void) {
   RCC->AHB1ENR |=  4;                /* enable GPIOC clock */
   RCC->AHB1ENR |=  1;                /* enable GPIOA clock */

   GPIOA->MODER &= ~0x00000C00;       /* clear pin mode */
   GPIOA->MODER |=  0x00000400;       /* set pin to output mode */

   GPIOC->MODER &= ~0x0C000000;       /* clear pin mode to input
mode */

   while(1) {
      if (GPIOC->IDR & 0x2000)        /* if PC13 is high */
         GPIOA->BSRR = 0x00200000;   /* turn off green LED */
      else
         GPIOA->BSRR = 0x00000020;   /* turn on green LED */
   }
}
```

Generador de onda senoidal en salida DAC

```
#include "main.h"
#include <math.h>

#define USEMAXMHZ

/* Test for boards */
#if defined(STM32F446xx)
#define GLCD_RCC_M (360)
#else
/* No DAC available */
#undef USEMAXMHZ
#endif

#define LENGTH (256)
#define BASEANGLE (2*M_PI/LENGTH)

int main(void) {

        uint32_t i;
```

```
	uint32_t sintab[LENGTH];

	/* Calculate the sine table */
	for (i=0; i<LENGTH; i++) {
		// No output buffer
		//sintab[i] = 2047.5*sin(BASEANGLE*i)+2048.0;
		sintab[i] = 1820.0*sin(BASEANGLE*i)+2048.0;
	}

#ifdef USEMAXMHZ

	/* For the correct procedure to enable max core frequency,
	 * see RM0390, page 97 (Entering Over-drive Mode)
	 */

	/* 5 wait states */
	FLASH->ACR = (FLASH_ACR_LATENCY_5WS <<
FLASH_ACR_LATENCY_Pos);

	/* Caches enable, prefetch enable */
```

```
    FLASH->ACR |= (FLASH_ACR_PRFTEN | FLASH_ACR_ICEN
| FLASH_ACR_DCEN);


    /* Enable the power system */
    do {
        __IO uint32_t tmpreg = 0x00U;
        SET_BIT(RCC->APB1ENR, RCC_APB1ENR_PWREN);
        /* Delay after an RCC peripheral clock enabling */
        tmpreg = READ_BIT(RCC->APB1ENR,
RCC_APB1ENR_PWREN);
        UNUSED(tmpreg);
    } while(0U);


    /* Select Scale 1 voltage regulator (max clock frequency) */
    do {
        __IO uint32_t tmpreg = 0x00U;
        MODIFY_REG(PWR->CR, PWR_CR_VOS,
PWR_REGULATOR_VOLTAGE_SCALE1);
        /* Delay after an RCC peripheral clock enabling */
        tmpreg = READ_BIT(PWR->CR, PWR_CR_VOS);
        UNUSED(tmpreg);
```

```
    } while(0U);

    /* Select HSE bypass to use clock oscillator */
    RCC->CR |= RCC_CR_HSEBYP;

    /* Start the Main PLL to MAX MHz
     * FREQ = ((HSE/M)*N)/P
     * 1 MHz <= HSE/M <= 2 MHz
     * 50 <= N <= 432      2 <= M <= 63     P = 2(0),4(1),6(2),8(3)
     */
    RCC->PLLCFGR =
(GLCD_RCC_M<<RCC_PLLCFGR_PLLN_Pos) |
(4<<RCC_PLLCFGR_PLLM_Pos) |
                          (1<<RCC_PLLCFGR_PLLP_Pos) |
(1<<RCC_PLLCFGR_PLLSRC_Pos);

    /* Enable HSE, PLL */
    RCC->CR |= RCC_CR_HSEON | RCC_CR_PLLON;
    //RCC->CR |= RCC_CR_HSEON;

    /* Wait for HSE, PLL to become ready */
```

```
    while ((RCC->CR & RCC_CR_HSERDY) == 0);

    while ((RCC->CR & RCC_CR_PLLRDY) == 0);


    /* Enable the Over-drive to extend the clock frequency to 180
MHz */

    //PWR->CR |= PWR_CR_ODEN;

    __HAL_PWR_OVERDRIVE_ENABLE();


    /* Wait for Over-drive enable */

    // while (!(PWR->CSR & (PWR_CSR_ODRDY)) ==
(PWR_CSR_ODRDY));

    while(!__HAL_PWR_GET_FLAG(PWR_FLAG_ODRDY)) {}


    /* Enable the Over-drive switch */

    //(*(__IO uint32_t *) CR_ODSWEN_BB = ENABLE); // bit
banding

    //PWR->CR |= PWR_CR_ODSWEN;              // normal way

    __HAL_PWR_OVERDRIVESWITCHING_ENABLE();


    /* Wait for Over-drive ready */

    // while (!(PWR->CSR & (PWR_CSR_ODSWRDY)) ==
(PWR_CSR_ODSWRDY));
```

```
        while(!__HAL_PWR_GET_FLAG(PWR_FLAG_ODSWRDY))
{}



        /* APB2 clk div'd by 2, APB1 clk div'd by 4, AHB div'd by 1, use
PLL */

        RCC->CFGR = (4<<RCC_CFGR_PPRE2_Pos) |
(5<<RCC_CFGR_PPRE1_Pos) | (0<<RCC_CFGR_HPRE_Pos)|
(2<<RCC_CFGR_SW_Pos);



        /* Wait for PLL to lock on */

        while ((RCC->CR & RCC_CR_PLLRDY) == 0);


#endif



        /* Update the SystemCoreClock variable */

        SystemCoreClockUpdate();


        volatile uint32_t dummy = SystemCoreClock;


        /* Enable the GPIOA clock */
```

```
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
	/* Select pin PA4 analog mode (mode 3, output of DAC1) */
	GPIOA->MODER |= GPIO_MODER_MODER4;

	/* Enable DAC clock */
	RCC->APB1ENR |= RCC_APB1LPENR_DACLPEN;
	/* Enable DAC1 */
	DAC->CR |= DAC_CR_EN1;

	/* Generate a sine wave of 100 kHz on pin PA4 */
	while (1) {
		for (i=0; i<LENGTH; i++) {
			/* Write sine table sample to the Data Holding Register
*/
			/* for 12 bits, right aligned */
			DAC->DHR12R1 = sintab[i];
		}
	}
	return 0;
}
```

Ejemplo con LCD

```
#define USEMAXMHZ

#include <stdio.h>
#include <stdint.h>

#include "stm32f4xx.h"
#include "lcd.h"

/* Test for boards */
#if defined(STM32F446xx)
#define GLCD_RCC_M (378)
#elif defined(STM32F411xx)
#define GLCD_RCC_M (225)
#else
#undef USEMAXMHZ
#define GLCD_RCC_M (50)
#endif

int main(void) {
```

```
	int i;
	/* volatile for use with optimizer */
	volatile int dly;
	char buf[20];
	uint8_t chdef[] = {0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa,
0x55};


	SystemInit();


#ifdef USEMAXMHZ


	/* 5 wait states */
	FLASH->ACR = (FLASH_ACR_LATENCY_5WS <<
FLASH_ACR_LATENCY_Pos);


	/* Caches enable, prefetch enable */
	FLASH->ACR |= (FLASH_ACR_PRFTEN | FLASH_ACR_ICEN
| FLASH_ACR_DCEN);


	RCC->CR |= RCC_CR_HSEBYP;
```

```
	/* Start the Main PLL to 168 MHz
	 * FREQ = ((HSE/M)*N)/P
	 * 1 MHz <= HSE/M <= 2 MHz
	 * 50 <= N <= 432      2 <= M <= 63     P = 2,4,6,8
	 */
	RCC->PLLCFGR =
(GLCD_RCC_M<<RCC_PLLCFGR_PLLN_Pos) |
(6<<RCC_PLLCFGR_PLLM_Pos) |
                        (2<<RCC_PLLCFGR_PLLP_Pos) |
(0<<RCC_PLLCFGR_PLLSRC_Pos);

	/* Enable HSE, PLL */
	RCC->CR |= RCC_CR_HSEON | RCC_CR_PLLON;
	//RCC->CR |= RCC_CR_HSEON;

	/* Wait for HSE, PLL to become ready */
	while ((RCC->CR & RCC_CR_HSERDY) == 0);
	while ((RCC->CR & RCC_CR_PLLRDY) == 0);

	/* APB2 clk div'd by 2, APB1 clk div'd by 4 */
```

```
	RCC->CFGR = (4<<RCC_CFGR_PPRE2_Pos) |
(5<<RCC_CFGR_PPRE1_Pos);


	/* Select PLL clock */

	RCC->CFGR |= (2<<RCC_CFGR_SW_Pos);


#endif


	/* Get the CPU speed from the I/O registers */

	/* This function sets SystemCoreClock to 100000000 */

	SystemCoreClockUpdate();


	lcd_init();

	lcd_init_backlight();


	lcd_set_cgram(0x00, chdef);


	lcd_putc('H');

	lcd_putc('e');

	lcd_putc('l');
```

```
lcd_putc('l');
lcd_putc('o');

lcd_puts(" World! ");
lcd_goto(1,0);
lcd_puts("Char at pos 0: ");
lcd_putc(0x00);

for (dly=0; dly<3000000; dly++);
lcd_cls();

lcd_set_backlight(255/3);

for (dly=0; dly<3000000; dly++);
lcd_cursor(0, 0);

for (dly=0; dly<3000000; dly++);
lcd_home();
lcd_puts("Booting.");
for (i=0; i<6; i++) {
```

```
        for (dly=0; dly<300000; dly++);
        lcd_putc('.');
}

lcd_goto(1,6);
lcd_putc('X');

lcd_set_backlight(2*255/3);

for (dly=0; dly<3000000; dly++);
lcd_cls();
lcd_goto(1,0);
lcd_puts("From 0 to 32767");
lcd_home();

for (i=0; i<=32767; i++) {
        snprintf(buf, sizeof buf, "i = %5d", i);
        lcd_goto(0,0);
        lcd_puts(buf);
        lcd_set_backlight((uint8_t) (i>>2));
```

```
	}


	i = 0;
	while (1) {
		lcd_goto(1,0);
		lcd_puts("Now reset me!   ");
		i=(i+1)&0x7fff;
		lcd_set_backlight(i>>2);
	}

	return 0;
}
```

PWM

```
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* Needed for expf() */
#include <math.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
```

```
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */
/* To keep track of the Duty Cycle */
uint32_t counter = 0;
/* 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 */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
```

```
/* USER CODE BEGIN 0 */


/* USER CODE END 0 */


/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */


  /* USER CODE END 1 */


  /* MCU Configuration--------------------------------------------------------*/


  /* Reset of all peripherals, Initializes the Flash interface and the
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 */

/* Start the PWM generation */
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_Base_Start(&htim2);
```

```
  /* USER CODE END 2 */


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


    /* USER CODE BEGIN 3 */
                /* Write directly to Compare Register CH1 to avoid stopping
the PWM
                 * generation. Using the HAL functions, the PWM generation
will stop
                 * for a short time and then starts again. We don't want that,
so we
                 * update the compare register directly. See:
                 *
https://community.st.com/s/question/0D50X00009XkaM3SAJ/changing-
of-pwm-duty-cycle-in-an-efficient-manner
                 */


              /* Please note that the perception of the human eye is non-linear
and
```

```
        * is modeled with the Weber–Fechner law. See:
        *
https://en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law
        * So in real life the CCR1 register should be loaded by a
corrected
        * value. The formula below compensates for the human eye
perception.
        * Now the brightness of a led will almost appear linear to the
human
        * eye.
        */


#if 1
        float perception = 10000.0f *
((expf((float)(counter*counter)/(10000.0f*10000.0f))-
1.0f)/(2.718281828459045235 4f-1.0f));
        TIM2->CCR1 = perception;
#else
            TIM2->CCR1 = counter;
#endif


        HAL_Delay(1);
        if (counter>9999) {
```

```
            counter = 0;
        } else {
            counter += 10;
        }
      }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
```

```
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_
VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
parameters

  * in the RCC_OscInitTypeDef structure.

  */

  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE;

  RCC_OscInitStruct.HSEState = RCC_HSE_ON;

  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;

  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

  RCC_OscInitStruct.PLL.PLLM = 8;

  RCC_OscInitStruct.PLL.PLLN = 360;

  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;

  RCC_OscInitStruct.PLL.PLLQ = 2;

  RCC_OscInitStruct.PLL.PLLR = 2;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)

  {

    Error_Handler();

  }

  /** Activate the Over-Drive mode
```

```
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses 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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */
```

```
  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_OC1;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 2499;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */
```

```
  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}
```

```
/* USER CODE BEGIN 4 */


/* USER CODE END 4 */


/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */


  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```