Microcontroller Interfacing

CSE/EE 5/7385 Microcontroller Architecture and Interfacing

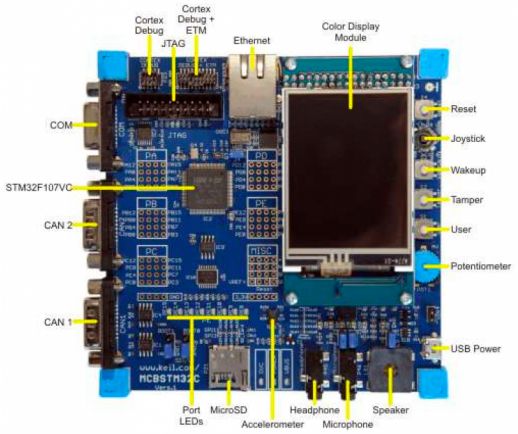
David Houngninou

Southern Methodist University

Table of content

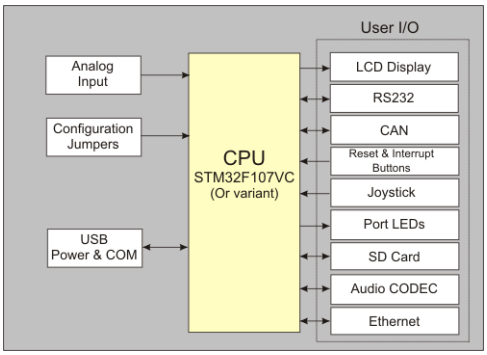
Target device: MCBSTM32C

- 72MHz ARM Cortex™-M3
- On-Chip Flash: 256KB
- On-Chip RAM: 64KB
- External Memory: 8KB I2C Flash
- Serial/UART Port
- 80 GPIO pins



Target device: MCBSTM32C

- 72MHz ARM Cortex™-M3
- On-Chip Flash: 256KB
- On-Chip RAM: 64KB
- External Memory: 8KB I2C Flash
- Serial/UART Port
- 80 GPIO pins



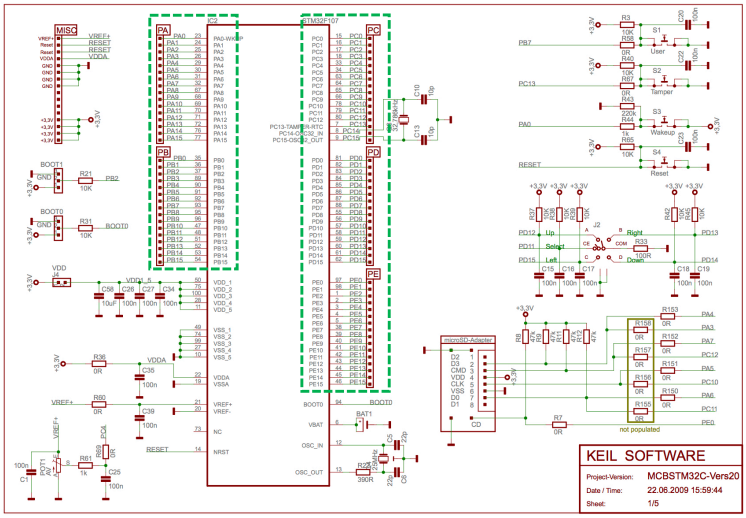
Useful references

- Evaluation board: MCBSTM32C
- Microcontroller: STM32F107VC
- Manufacturer: STMicroelectronics
- Board documentation: www.keil.com/dd/chip/4889.htm
- Board schematic: www.keil.com/mcbstm32c/mcbstm32c-base-board-schematics.pdf
- Microcontroller Reference manual: **RM0008** davidkebo.com/documents/RM0008_Reference_manual.pdf

What is a GPIO port ?

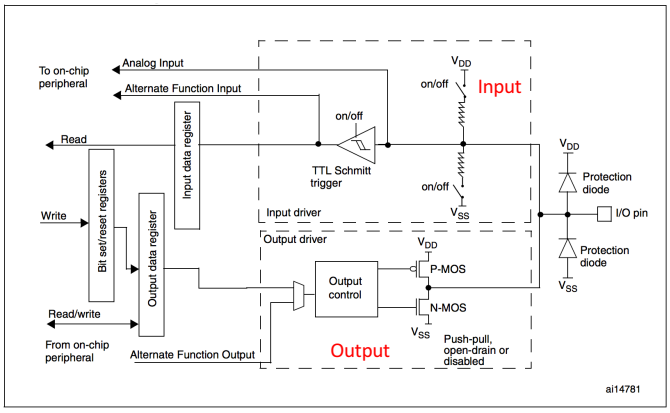
- **A GPIO port** is a group of user-configurable pins for **G**eneral **P**urpose **I**nput/**O**utput
- The STM32F107VC has 7 general purpose input/output (GPIO) ports
- The 7 GPIO Ports are A, B, C, D, E, F and G.
- Each port can have up to 16 pins
- Each port has 7 registers

What is a GPIO port ?



What is a GPIO pin ?

A GPIO pin is a generic pin controllable by the user at run time and configurable to be **input** or **output**.



What is a GPIO register ?

A GPIO register is a **32-bit** storage area that hold configuration bits for a GPIO pin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Example of a configurable 32-bit register

7 GPIO port registers (32-bit)

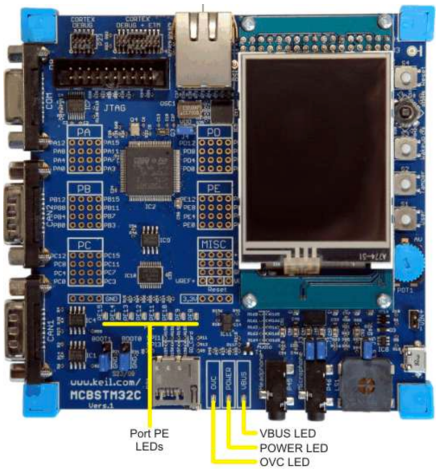
- Configuration register low GPIOx_CRL
- Configuration register high GPIOx_CRH
- Input data register GPIOx_IDR
- Output data register GPIOx_ODR
- Set/reset register GPIOx_BSRR
- Reset register GPIOx_BRR
- Locking register GPIOx_LCKR

*the **'x'** is the port name.*
e.g. GPIOA_IDR is the input data register associated with port A.

(Reference: RM0008 Page 170)

Tutorial: Interfacing with the LEDs

- Write a program that interfaces with the LEDs
- Toggle the 08 LEDs ON/OFF
- The toggle speed of the LEDs should depend on the input voltage of the ADC1 input.



Interfacing with the LEDs

Steps

1. Initialize the microcontroller
2. Enable the clock for the LEDs
3. Setup the GPIO for the LEDs
4. Setup and initialize the ADC
5. Read the ADC values
6. Blink the LEDs

Initialize the microcontroller

SystemInit() is a function defined in the source file system_stm32f10x_cl.c

The purpose of this function is to:

- Initialize the embedded flash interface
- Update the system clock frequency

Enable the clock for the LEDs

- The ARM microcontroller does not hold the clock active continuously
- Enable the clock for the LEDs manually **(Port E)**
- Set configuration bit(s) in register **RCC_APB2ENR**
- RCC_APB2ENR : Advanced Peripheral Bus 2 Enable Register

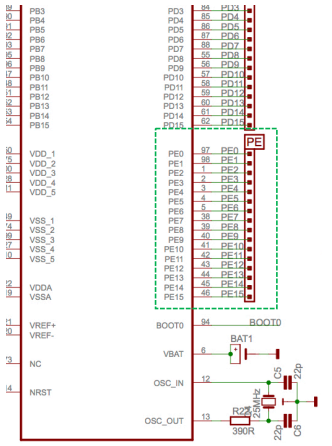
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
rw															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

```
RCC->APB2ENR |= 1 << 6;   // Enable GPIOE clock
```

Setup the GPIO for the LEDs

- LEDs pins are wired to GPIOE (Port E)
- The 16 pins are PE0 to PE15
- LEDs are wired to pins PE8,…PE15
- Configure the pins as **output**

GPIOE->CRH = **0x33333333**;

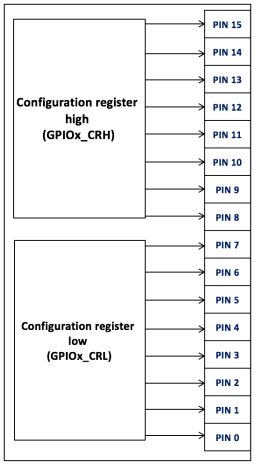


Setup the GPIO for the LEDs

	Name	r/w	Bits	Function
GPIO**x**_CRL	Port configuration register low	rw	CNF[1:0], MODE[1:0]	Configure lowest 8 bit of port *x*. IN or OUT
GPIO**x**_CRH	Port configuration register high	rw	CNF[1:0], MODE[1:0]	Configure highest 8 bit of port *x*. IN or OUT
GPIO**x**_IDR	port input data register	r	IDR[15:0]	Read state of pins configured for input on port *x* (Lowest 16 bits of word)
GPIO**x**_ODR	port output data register	rw	ODR[15:0]	Write to pins configured for Output on port *x* (Lowest 16 bits of word)
GPIO**x**_BSRR	Port bit set/reset register	w	BS[15:0], BR[15:0]	Atomic Set/Reset individual Pins configured for Output.
GPIO**x**_BRR	Port bit reset register	w	BR[15:0]	Atomic Reset individual Pins configured for Output. (Lowest 16 bits of word)
GPIO**x**_LCKR	Port configuration lock register	rw	LCK[15:0], LCKK	Lock individual pins of port *x* (Freeze CRL and CRH of ports)

Table 1: GPIO Configuration Registers

Setup the GPIO for the LEDs



Port configuration register to pins

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General Purpose Output	Push-Pull	0	0	01 10 11		0 or 1
	Open Drain		1			0 or 1
Alternate function Output	Push-Pull	1	0			Don't Care
	Open Drain		1			Don't Care
Input	Analog	0	0	00		Don't Care
	Floating		1			Don't Care
	Pull-Down	1	0			0
	Pull-Up					1

Table 2: Port bit configuration summary

Setup the GPIO for the LEDs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Port configuration register high (GPIOx_CRH) (x=A..G)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Port configuration register low (GPIOx_CRL) (x=A..G)

Setup the GPIO for the LEDs

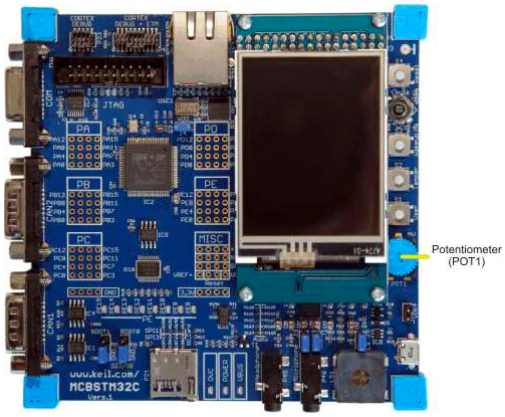
- GPIOE_CRH (Control Register High) controls bits 8 through 15.
- Configure GPIOE_CRH as a push pull output (See table 2)

CNFx[1:0] = 00 and MODEx[1:0] = 01,10 or **11**

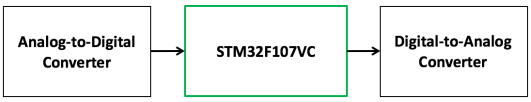
Register bit pattern: **0011 0011 0011 0011 0011 0011 0011 0011**

GPIOE->CRH = **0x33333333**;

Setup and initialize the ADC



The Analog to Digital Converter (ADC) converts an analog input to a digital input.



ADC to STM32F107VC to DAC

Setup and initialize the ADC

Enable the clock for the ADC.

RCC->APB2ENR |= **1** << **9**;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

APB2 peripheral clock enable register (RCC_APB2ENR)

Setup and initialize the ADC

- The potentiometer connects to port pin PC4 (ADC12_IN14)
- Pin PC4 must be configured as an **analog input**. (See table 2)
- Pin 4 is connected to CRL (Configuration register low)

To configure PC4 we use the following mask:

1111 1111 1111 0000 1111 1111 1111 1111

A bitwise AND of GPIOC->CRL and the mask clears CNF4 and MODE4

GPIOC->CRL &= **0xFFF0FFFF**;

Setup the ADC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Port configuration register high (GPIOx_CRH) (x=A..G)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Port configuration register low (GPIOx_CRL) (x=A..G)

ADC Sequence registers

The STM32F107 has 18 analog input channels.

Sequence registers configure the number of channels to sample

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]					SQ14[4:0]					SQ13[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC Sequence registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]					SQ14[4:0]					SQ13[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 23:20 L[3:0]: Regular channel sequence **length**. Number of conversions in the regular channel conversion sequence.

0000: 1 conversion, **0001**: 2 conversions, ... **1111**: 16 conversions

Bits 19:15 SQ16[4:0]: 16th conversion in regular sequence. Channel number assigned as the 16th in the conversion sequence.

Bits 14:10 SQ15[4:0]: 15th conversion in regular sequence

ADC Sequence registers

The STM32F107 has **18 analog input channels**.

The potentiometer is wired to channel **14**

For a single conversion:

```
ADC1->SQR1 = 0x00000000; // Regular channel single conversion
ADC1->SQR2 = 0x00000000; // Clear register
ADC1->SQR3 = (14<<0);  // channel 14 as 1st conversion
```

ADC sample time registers ADC_SMPR

The number of ADC_CLK cycles that the ADC samples the input voltage is modified using:

SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC sample time register ADC_SMPR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 23:0 SMPx[2:0]: Channel x Sample time selection

Select the sample time individually for each channel.

000: 1.5 cycles	**001**: 7.5 cycles	**010**: 13.5 cycles	**011**: 28.5 cycles
100: 41.5 cycles	**101**: 55.5 cycles	**110**: 71.5 cycles	**111**: 239.5 cycles

Table 3: ADC sample time configuration

ADC sample time register ADC_SMPR

The ADC should delay reading for **5.15 uS**

For an ADCCLK running at 14 MHz

Determine the sample time (Setting for SMPx)?

ADC sample time register ADC_SMPR

The ADC should delay reading for 5.15 uS

For an ADCCLK running at 14 MHz

Determine the sample time (Setting for SMPx) ?

T = 1 / 14 MHz

T = 0.07142 uS

# of cycles = 5.15 uS / T = 5.15 uS / 0.07142 uS

# of cycles = 72

ADC Sample Time Bits Register

Set channel **14** at a sample time of **72 cycles :**

The bit configuration for 72 cycles is **110** (Table 3)

ADC1->SMPR1 = **6** << **12**;

ADC1->SMPR2 = **0x00000000**; // Clear register

ADC Control Registers (ADC_CR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								AWDEN	JAWDEN	Reserved		DUALMOD[3:0]			
								rw	rw			rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The ADC is controlled using two control registers **ADC_CR1** and **ADC_CR2**

Bit 8 SCAN (Scan mode): In this mode, the inputs selected through ADC_SQRx registers are converted.

Bit 5 EOCIE (Interrupt enable for EOC): enable/disable the End of Conversion interrupt.

ADC block diagram (page 216) RM0008

ADC Control Registers **(ADC_CR1)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								AWDEN	JAWDEN	Reserved		DUALMOD[3:0]			
								rw	rw			rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

To interrupt the microcontroller at the end of conversion:

ADC1->CR1 |= (1<<5); // EOC interrupt

NIVC->ISER[0] |= (1<<18); //Interrupt number 18

ADC Control Registers (ADC_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

SWSTART: Start conversion of regular channels

EXTTRIG: Enable external signal

EXTSEL SWSTART as trigger.

ALIGN: Data alignment. 0: right alignment, 1: left alignment

RSTCAL: Reset calibration

CAL: A/D Calibration

CONT: Continuous conversion

ADON: A/D converter ON / OFF

ADC Control Registers (ADC_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

Configuration

```
ADC1->CR2 |= (7<<17);        // Set SWSTART as trigger
ADC1->CR2 |= (1<<20);        // Enable external trigger
ADC1->CR2 &= ~(1<<11);       // Right data alignment
ADC1->CR2 |= (1<<1);         // Continuous conversion
ADC1->CR2 |= (1<<0);         // Turn ADC ON
```

ADC Control Registers (ADC_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

Enable external trigger, EXTSEL = SWSTART, Continuous conversion, ADC enable

ADC1->CR2 = (**1** << **20**) | (**7** << **17**) | (**1** << **1**) | (**1** << **0**) ;

ADC Control Registers (ADC_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVREFE	SWSTART	JSWSTART	EXTTRIG	EXTSEL[2:0]			Res.
								rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTTRIG	JEXTSEL[2:0]			ALIGN	Reserved		DMA	Reserved				RSTCAL	CAL	CONT	ADON
rw	rw	rw	rw	rw	Res.		rw					rw	rw	rw	rw

Calibration

```
ADC1->CR2 |= (1<<3);                  // reset calibration
    while (ADC1->CR2 & (1<<3));       // wait until reset finished
ADC1->CR2 |= (1<<2);                  // start calibration
    while (ADC1->CR2 & (1<<2));       // wait until calibration finished
```

Conversion

```
ADC1->CR2 |= (1<<22);                 // start SW conversion
```

ADC Status and Data Registers (ADC_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											STRT	JSTRT	JEOC	EOC	AWD
											rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 4 STRT Regular channel Start flag

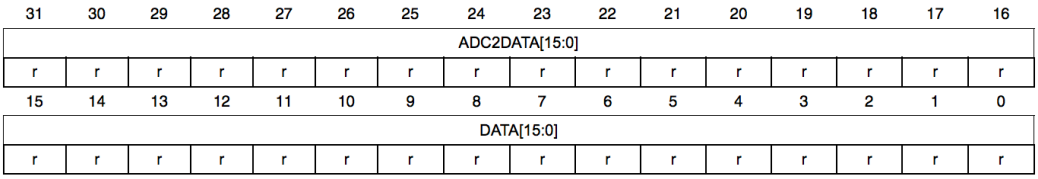
Bit 3 JSTRT Injected channel Start flag

Bit 2 JEOC Injected channel end of conversion

Bit 1 EOC End of conversion

Bit 0 AWD Analog watchdog flag

ADC Status and Data Registers (ADC_DR)



Bits 31:16
ADC2DATA[15:0] Contain the regular data of ADC2.

Bits 15:0
DATA[15:0] Contain the conversion result from the regular channels.

ADC Status and Data Registers (ADC_SR/ ADC_DR)

```
if (ADC1->SR & (1 << 1)) { // If conversion has finished (Check EOC bit)
AD_val = ADC1->DR & 0x0FFF; // Read AD converted value
ADC1->CR2 |= 1 << 22; // Start new conversion
}
```

Blink the LEDs (Read-modify-write method)

```
const long led_mask[] =
{1<<15, 1<<14, 1<<13, 1<<12, 1<<11, 1<<10, 1<<9, 1<<8};
GPIOE->ODR |= led_mask[num]; /*Turn LED num on*/
for (i = 0; i < ((AD_val << 8) + 100000); i++); /*ADC delay*/
GPIOE->ODR &= ~led_mask[num]; /*Turn LED num off*/
```


Blink the LEDs (Read-modify-write method)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Port output data register (GPIOx_ODR) (x=A..G)

```
GPIOE->ODR |= led_mask[num]; /*Turn LED num on*/
```

Blink the LEDs (Atomic method)

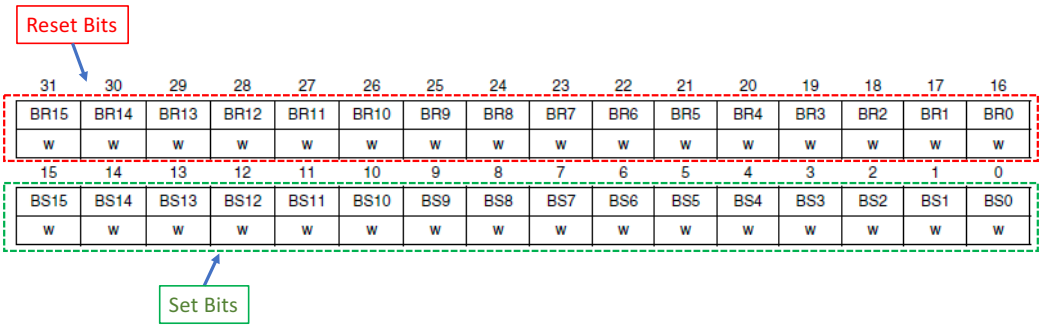
```
const long led_mask[] =
{1<<15, 1<<14, 1<<13, 1<<12, 1<<11, 1<<10, 1<<9, 1<<8};

GPIOE->BSRR = led_mask[num]; /*Turn LED num on*/

for (i = 0; i < ((AD_val << 8) + 100000); i++); /*ADC delay*/

GPIOE->BSRR = led_mask[num] << 16; /*Turn LED num off*/
```


Blink the LEDs (Atomic method)



Note: Shifting by 16 bits goes from **Set** bits to **Reset** bits

Timers

Timers are clock sources used as the 'heartbeats' for operations.

e.g. of applications using timers:

- Counting pulses
- Measuring time periods of waveforms
- Generating pulse width modulation (PWM) signals
- Triggering external devices
- Timing special events

Timers

03 groups of timers:

- Basic Timers
- General Purpose Timers
- Advanced Timers (TIM1&TIM8)

Basic timers

TIM6, TIM7

Applications:

no I/O channels for input capture

no PWM generation

Only used for time-base generation

General Purpose Timers

TIM2, TIM3, TIM4, TIM5

Applications:

- PWM generation
- Input capture
- Time-base generation
- Output compare

Advanced timers

TIM1, TIM8

Applications:

- Advanced PWM generation
- Input capture
- Time-base generation
- Output compare

Timers pin

Each timer is associated with an I/O pin

Configure the **alternate function** to use the timer pin

What is an alternate function?

In addition to general purpose input and output the ARM subsystem can implement other specialized input output functions.

e.g. **TIM4_Ch4** is an alternate function for **pin PB9**

TIMx functional description

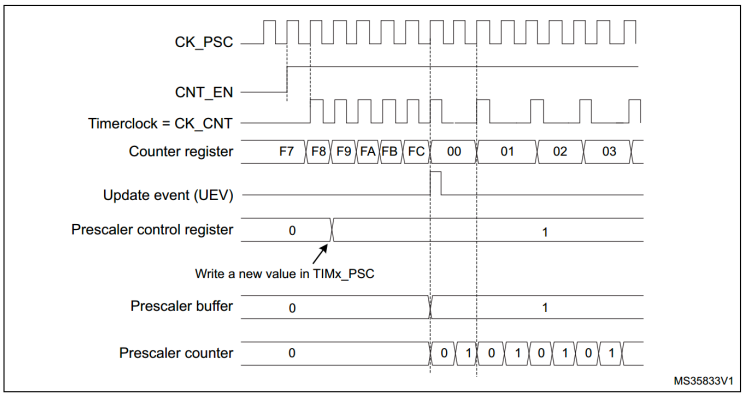
The main block of the timer is a 16-bit counter **TIMx_CNT** with an auto-reload register **TIMx_ARR**

The counter can count up, down or both up and down.

The counter clock can be divided by a pre-scaler **TIMx_PSC**

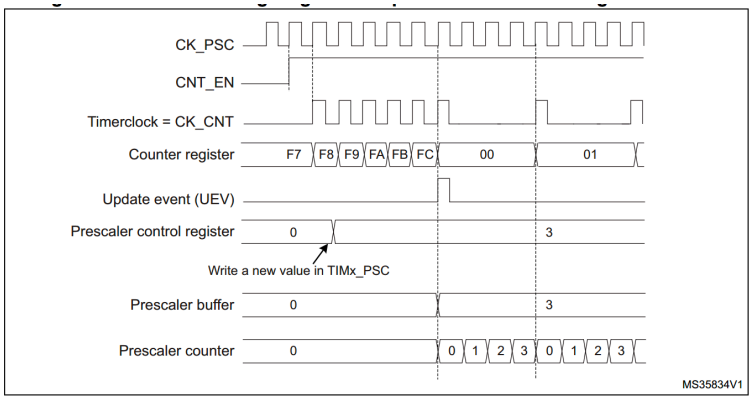
General-purpose timers (TIM2 to TIM5) RM0008 Manual 15.1 Page 364

TIMx functional description



Counter timing diagram with pre-scaler division change from 1 to 2

TIMx functional description



Counter timing diagram with pre-scaler division change from 1 to 4

Timer configuration

Steps:	**Registers**
1. Enable clock to the timer	RCC->APB1ENR
2. Enable the Alternate function and the GPIO clock	RCC->APB2ENR
3. Set the internal clock as system clock	(TIMx_SMCR)TIMx slave mode control register
4. Set the pre-scaler	(TIMx_PSC)TIMx pre-scaler register
5. Enable the timer	(TIMx_CR1)TIMx control register

Timer configuration

Steps:	**Notes about the clock usage:**
1. Enable clock to the timer 2. Enable the Alternate function and the GPIO clock 3. Set the internal clock as system clock 4. Set the pre-scaler 5. Enable the timer	The **HSI** clock signal is generated from an internal **8 MHz** RC oscillator and can be used directly as a system clock. (HSI) high-speed internal oscillator **To set the pre-scaler register:** Use the counter clock frequency (CK_CNT) **CK_CNT = f_{CK_PSC} / (PSC[15:0] + 1)**

Timer example

A developer is writing a function for a counter user a timer.

The internal HIS clock signal runs at **8 MHz**

What is value to set the pre-scaler to count every **millisecond**?

CK_CNT = f_{CK_PSC} / (PSC[15:0] + 1)

PSC[15:0] + 1 = f_{CK_PSC} / 1000

PSC[15:0] = (f_{CK_PSC} / 1000) – 1

PSC[15:0] = 7999

Timer clock

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Advanced Peripheral Bus 1 Enable Register

e.g: To use TIM4

RCC->APB1ENR |= (1<<2); //enable clock to TIM4.

Timer clock

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Advanced Peripheral Bus 2 Enable Register

e.g: To use TIM4

RCC->APB2ENR |= (1)|(1<<3); //Enable the A.F clock and the GPIO clock

Timer pre-scaler register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

TIM1&TIM8 pre-scaler (TIMx_PSC)

PSC[15:0]: Pre-scaler value

Timer slave mode register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	rw	rw	rw

TIM1&TIM8 slave mode control register (TIMx_SMCR)

SMS = 000, the pre-scaler is clocked directly by the internal clock.

Timer control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

TIM1&TIM8 control register 1 (TIMx_CR1)

CEN: Counter enable

0: Counter disabled

1: Counter enabled