STM32F103C8T6 (BLUE PILL)

By Adham EL-Serougy

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- ADC
- UART
- SPI
- I2C
- FreeRTOS

Content

- **Clock Speed Configuration**
- GPIO
- GPIO Interrupts
- Timers
- ADC
- UART
- SPI
- I2C
- FreeRTOS

Clock Speed Configuration

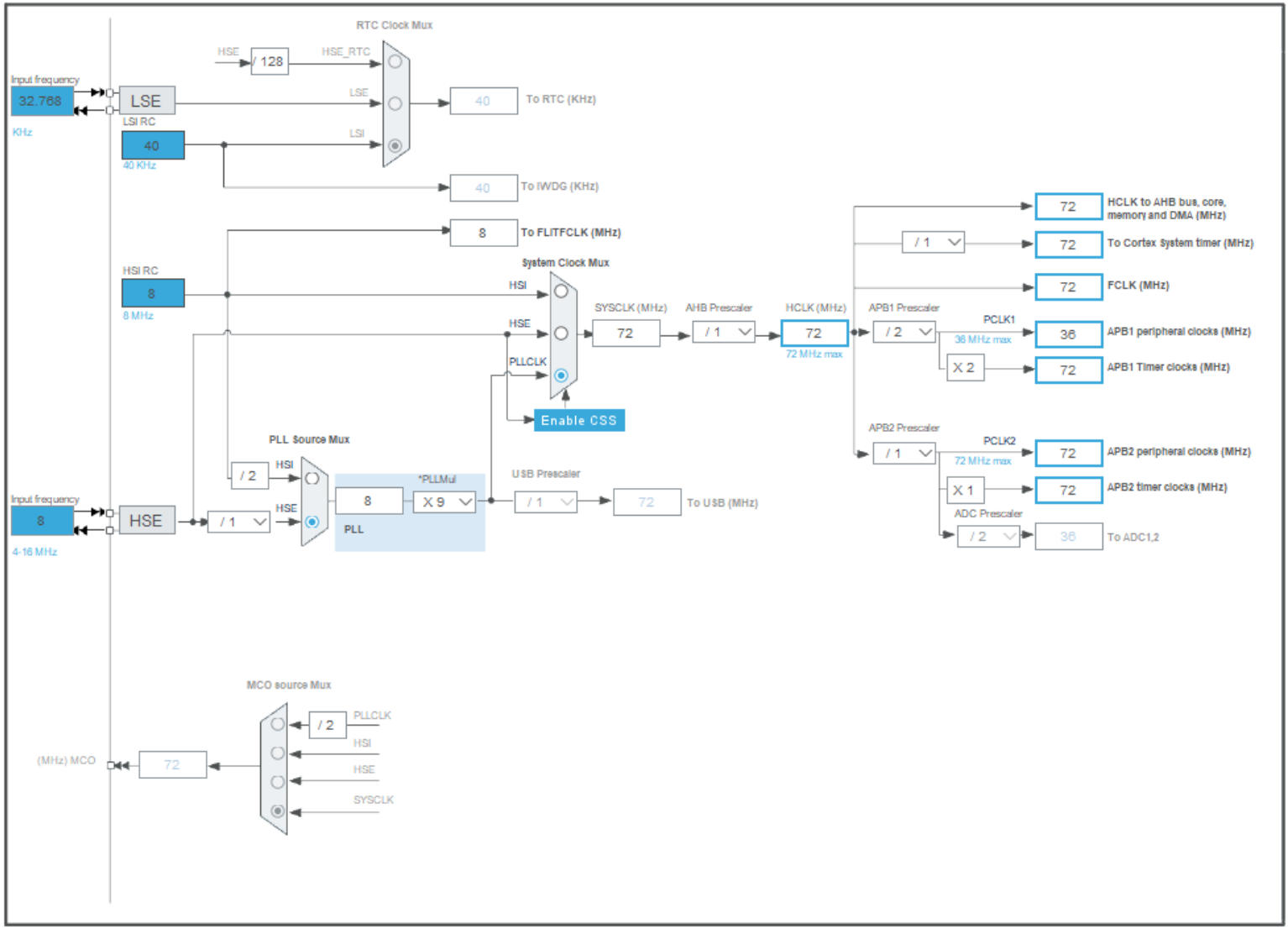
- The blue pill runs at 8 MHz on default settings.
- However it is advertised to a 72 MHz clock frequency.
- In order to utilize the full capability of the blue pill we will need to configure some of the registers related to the clock frequency.
- There are 3 clock sources used to drive system clock in the blue pill:
  - HSI (High Speed Internal) oscillator clock
    - 8 MHz Speed
  - HSE (High Speed External) oscillator clock
    - 4 – 16 MHz Speed (8 MHz crystals are found in commercial Blue Pills)
  - PLL (Phase Locked Loop) clock
    - Clock Multiplier

Clock Speed Configuration

- There are secondary clock sources which are not used to drive system clock but have other uses:
  - LSI (Low Speed Internal) Clock
    - Speed 40 kHz
    - Used to drive Independent Watchdog
  - LSE (Low Speed External) Clock
    - Speed 32.768 kHz
    - Used to drive RTC (Real Time Clock) accurately

Clock Speed Configuration

- So in order to get speeds above 8 MHz we need to use PLL clock multiplier to increase clock speeds.
- First we change the System Clock MUX to have PLLCLK as system clock.
- Then change the PLL Source MUX to take the HSE as its source.
- Then we adjust the value of PLLMul to x9 (values range from x2 to x16).
- This should Output a clock of 72 MHz.
- However since the maximum clock the APB1 Peripheral clock can withstand is 36 MHz we need to adjust APB1 Prescaler to /2.



Clock Speed Configuration

PLL Configuration in terms of registers:

7.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bit 25 **PLLRDY:** PLL clock ready flag
Set by hardware to indicate that the PLL is locked.
0: PLL unlocked
1: PLL locked

Bit 24 **PLLON:** PLL enable
Set and cleared by software to enable PLL.
Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock.
0: PLL OFF
1: PLL ON

Bit 17 **HSERDY:** External high-speed clock ready flag
Set by hardware to indicate that the HSE oscillator is stable. This bit needs 6 cycles of the HSE oscillator clock to fall down after HSEON reset.
0: HSE oscillator not ready
1: HSE oscillator ready

Bit 16 **HSEON:** HSE clock enable
Set and cleared by software.
Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.
0: HSE oscillator OFF
1: HSE oscillator ON

Clock Speed Configuration

PLL Configuration in terms of registers:

Bits 21:18 **PLLMUL:** PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 72 MHz.

0000: PLL input clock x 2
0001: PLL input clock x 3
0010: PLL input clock x 4
0011: PLL input clock x 5
0100: PLL input clock x 6
0101: PLL input clock x 7
0110: PLL input clock x 8
0111: PLL input clock x 9
1000: PLL input clock x 10
1001: PLL input clock x 11
1010: PLL input clock x 12
1011: PLL input clock x 13
1100: PLL input clock x 14
1101: PLL input clock x 15
1110: PLL input clock x 16
1111: PLL input clock x 16

Bit 16 **PLLSRC:** PLL entry clock source

Set and cleared by software to select PLL clock source. This bit can be written only when PLL is disabled.

0: HSI oscillator clock / 2 selected as PLL input clock
1: HSE oscillator clock selected as PLL input clock

Bits 10:8 **PPRE1:** APB low-speed prescaler (APB1)

Set and cleared by software to control the division factor of the APB low-speed clock (PCLK1).

Warning: the software has to set correctly these bits to not exceed 36 MHz on this domain.

0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

7.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					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
ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 3:2 **SWS:** System clock switch status

Set and cleared by hardware to indicate which clock source is used as system clock.

00: HSI oscillator used as system clock
01: HSE oscillator used as system clock
10: PLL used as system clock
11: not applicable

Bits 1:0 **SW:** System clock switch

Set and cleared by software to select SYSCLK source.

Set by hardware to force HSI selection when leaving Stop and Standby mode or in case of failure of the HSE oscillator used directly or indirectly as system clock (if the Clock Security System is enabled).

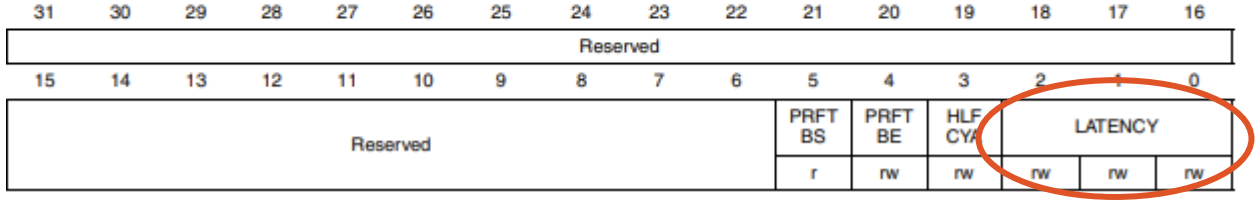
00: HSI selected as system clock
01: HSE selected as system clock
10: PLL selected as system clock
11: not allowed

Clock Speed Configuration

- Lastly we need to add wait states to flash memory reading since max speed is 24 MHz

3.1 Flash access control register (FLASH_ACR)

Address offset: 0x00
Reset value: 0x0000 0030



Bits 2:0 **LATENCY**: Latency

These bits represent the ratio of the SYSCLK (system clock) period to the Flash access time.

000 Zero wait state, if 0 < SYSCLK≤ 24 MHz

001 One wait state, if 24 MHz < SYSCLK ≤ 48 MHz

010 Two wait states, if 48 MHz < SYSCLK ≤72 MHz

Clock Speed Configuration

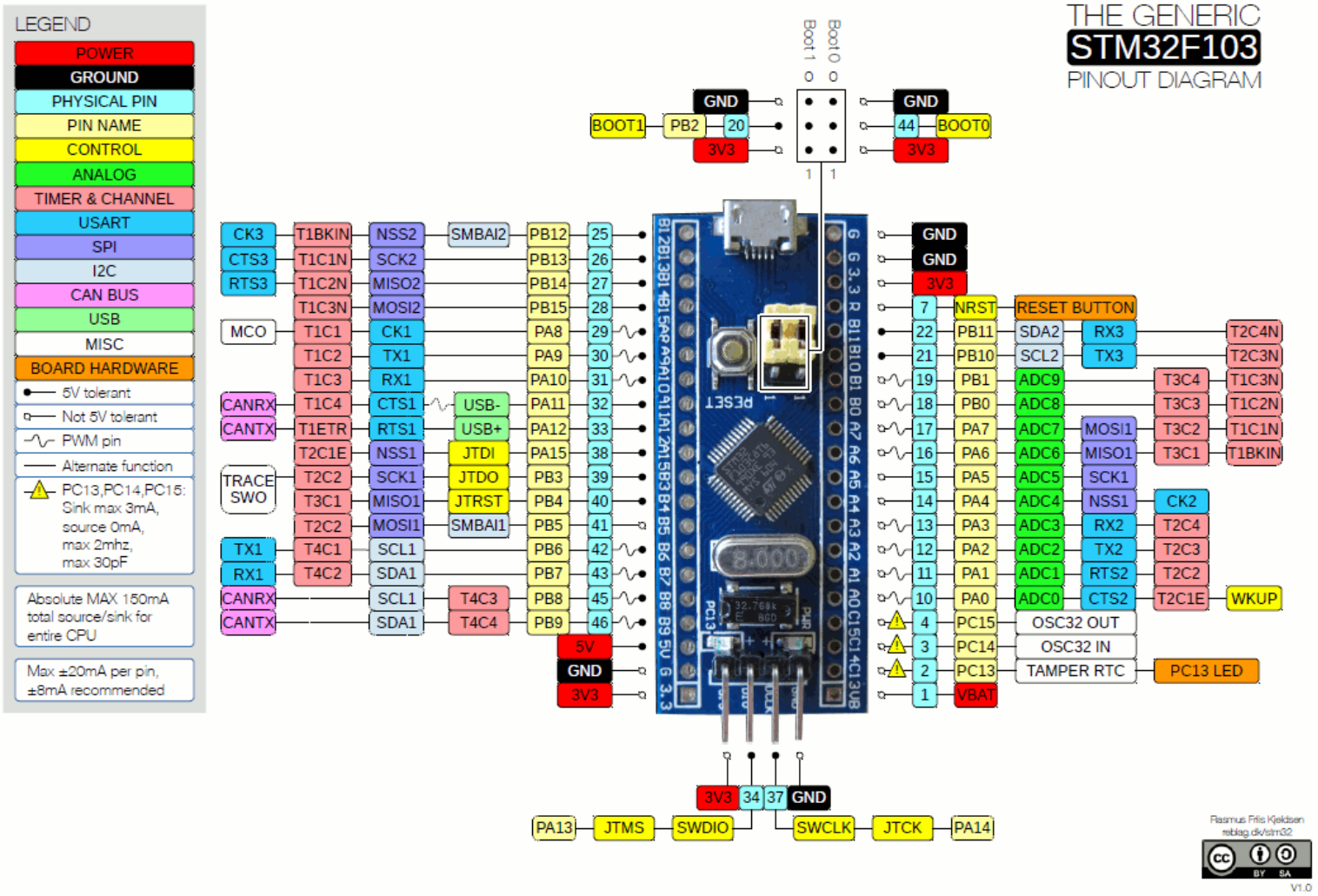
Code example for initializing clock speed to 72 MHz

```
FLASH->ACR |= (1<<1); //Adding Latency (2 Wait State) since between 48-72 MHz

RCC->CFGR |= (1<<10); //Configuring APB1 peripherals speed to be at max 36 MHz by
dividing by 2 since clock is 72 MHz

RCC->CR |= (1<<16); //Enabling HSE

while(!(RCC->CR & (1<<17))){} //Waiting for HSE to be ready

RCC->CFGR |= (1<<16); //Configuring HSE as PLL source

RCC->CFGR |= (1<<20)|(1<<19)|(1<<18); //Setting PLL multiplier to x9

RCC->CR |= (1<<24); //Enabling PLL

while(!(RCC->CR & (1<<25))){} //Waiting for PLL to be ready

RCC->CFGR |= (1<<1); //Set system clock to PLL clock

while(!(RCC->CFGR & (1<<3))){} //Waiting for system clock to be ready
```

Content

- Clock Speed Configuration
- **GPIO**
- GPIO Interrupts
- Timers
- ADC
- UART
- SPI
- I2C
- FreeRTOS

GPIO


LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
THE GENERIC
STM32F103
PINOUT DIAGRAM
Boot 1
Boot 0
BOOT1
PB2
20
GND
3V3
44
BOOT0
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T2C4N
T2C3N
T3C4
T3C3
T3C2
T3C1
T1C3N
T1C2N
T1C1N
T1BKIN
CK2
T2C4
T2C3
T2C2
T2C1E
WKUP
PC13 LED
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Friis Kjeldsen
rsblog.dk/stm32
V1.0

GPIO

There are 4 Ports in the blue pill:

- Port A with 16 pins
- Port B with 16 pins
- Port C with 3 pins
- Port D with 2 pins used to connect to HSE oscillator and are inaccessible to user

With bigger STM32F1xx packages there will be more ports with pins up to 16 in each port.

GPIO

Before Using any Peripheral we first must enable its clock:

Bit 4 **IOPCEN:** IO port C clock enable
Set and cleared by software.
0: IO port C clock disabled
1: IO port C clock enabled

Bit 3 **IOPBEN:** IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled

Bit 2 **IOPAEN:** IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

GPIO

Pins can be configured as:

- General Purpose output
- Alternate Function output
  - Such as communication or timers
- Input

Output mode speed is used to determine maximum speed output by pin (useful for Alternate function output).

Any Pins that are configured with Alternate Function require the AFIOEN bit in the RCC_APB2ENR register to be set with 1.

Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see *Table 21*		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
	Input floating		1			Don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 21. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Maximum output speed 10 MHz
10	Maximum output speed 2 MHz
11	Maximum output speed 50 MHz

GPIO

To configure pins 0 – 7 we will use GPIOx_CRL Register (x stands for port letter.)

For each pin there are 4 bits to be configured.

2 bits in Mode to determine whether pin is input or output.

2 bits in CNF to determine properties of the input mode or output mode.

Pins 8 – 15 are configured the same way but in the GPIOx_CRH Register.

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

Address offset: 0x00

Reset value: 0x4444 4444

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

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

GPIO

Reading Input:

Each pin has a designated bit in the GPIOx_IDR register to indicate whether a pin is receiving high or low input.

Ex:

```
int input = GPIOA->IDR &
(1<<5); //Reading input of
Pin A5
```

9.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy:** Port input data (y= 0 .. 15)

These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

GPIO

Generating Output:

In order to output a signal, the GPIOx_ODR must be set to 1 for the corresponding pin.

Ex:

- `GPIOA->ODR |= (1<<4);`
  `//Set pin A4 to High`
- `GPIOB->ODR &= ~(1<<2);`
  `//Clear pin B2 to Low`

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

Address offset: 0x0C

Reset value: 0x0000 0000

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

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

Content

- Clock Speed Configuration
- GPIO
- **GPIO Interrupts**
- Timers
- ADC
- UART
- SPI
- I2C
- FreeRTOS

GPIO Interrupts

To enable interrupts for GPIO we first need to enable the Alternative Function clock

Bit 0 **AFIOEN:** Alternate function IO clock enable
Set and cleared by software.
0: Alternate Function IO clock disabled
1: Alternate Function IO clock enabled

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

GPIO Interrupts

Now we need to tell the MCU that we want said pin to receive interrupts.

There are 4 registers, 1 for each set of 4 pins.

0 – 3 in register AFIO_EXTICR1

4 – 7 in register AFIO_EXTICR2

8 – 11 in register AFIO_EXTICR3

12 – 15 in register AFIO_EXTICR4

EX:

```
AFIO->EXTICR[0] |= (1<<0);
//Enable Pin B0 Interrupt
```

9.4.3 External interrupt configuration register 1 (AFIO_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

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
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]:** EXTI x configuration (x= 0 to 3)

These bits are written by software to select the source input for EXTIx external interrupt.
Refer to *Section 10.2.5: External interrupt/event line mapping*

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

GPIO Interrupts

Now we need to enable the line interrupt.

There are 20 interrupt lines:

0 – 15 are for pins 0 – 15 where PA15, PB15 and PC15 share the same line 15.

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB Wakeup event
- EXTI line 19 is connected to the Ethernet Wakeup event (available only in connectivity line devices)

10.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

GPIO Interrupts

Now to decide whether an interrupt is triggered due to rising edge, falling edge or both.

Ex:

```
EXTI->FTSR |= (1<<0); //Trigger falling edge
```

10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 used in connectivity line devices and is reserved otherwise.

GPIO Interrupts

Now we need to tell NVIC (Nested vectored interrupt controller) that an interrupt is initialized:

We will usePB0 for our example.

Since PB0 is in line 0 we will use EXTI0 line.

We need to find its corresponding position for EXTI0 in the vector table provided in the reference manual.

Position	Priority	Type of priority	Acronym	Description	Address
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060

GPIO Interrupts

We need to enable the NVIC to see our interrupt:

Table 41. Mapping of interrupts to the interrupt variables

Interrupts	CMSIS array elements(1)				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]
64-67	ISER[2]	ICER[2]	ISPR[2]	ICPR[2]	IABR[2]

Since our position is 6 we are in the first set so we enable interrupt in ISER[0]

4.3.2 Interrupt set-enable registers (NVIC_ISERx)

Address offset: 0x00 - 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Bits 31:0 **SETENA[31:0]**: Interrupt set-enable bits.

Write:

0: No effect

1: Enable interrupt

Read:

0: Interrupt disabled

1: Interrupt enabled.

GPIO Interrupts

Now we need to assign a priority to our interrupt, if this is not done then the priority will be left as default which is found in the vector table:

Each line can receive a priority between 0 and 3 with 0 being highest priority.

IP[interrupt position]

After an interrupt occurs the Pending register receives a 1 on the line that caused the interrupt

10.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit
0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a '1' into the bit.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

GPIO Interrupts

Ex:

```
AFIO->EXTICR[0] |= (1<<0); //Enable Pin B0 Interrupt
EXTI->IMR |= (1<<0); //Enable line 0 interrupt
EXTI->FTSR |= (1<<0); //Trigger falling edge
NVIC->ISER[0] |= (1<<6); //Enable EXTI0 interrupt
NVIC->IP[6] |= (1<<0); //High priority for EXTI0 Interrupt

void EXTI0_IRQHandler(void){
   if (EXTI->PR & (1 << 0)) { //Checking if line 0 is the cause of interrupt
      EXTI->PR |= (1 << 0); //Clearing the Pending register
      GPIOA->ODR |= 0b110; // Setting Pin A1 and A2 with high
   }
}
```

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- **Timers**
- ADC
- UART
- SPI
- I2C
- FreeRTOS

Timers

There are 7 timers in the blue pill:

- 1 Advanced Timer
- 3 General Purpose Timers
- 2 Watchdog Timers
- 1 SysTick timer 24-bit downcounter

For our purposes we will use the general purpose timers, advanced timers and SysTick Timer.

The advanced timer is the same as the general purpose timer but with extra capabilities.

We will consider the advanced timer as a general purpose timer.

Timers

As done with any peripheral we need to initialize its clock.

Bit 11 **TIM1EN**: TIM1 Timer clock enable
Set and cleared by software.
0: TIM1 timer clock disabled
1: TIM1 timer clock enabled

Bit 2 **TIM4EN**: Timer 4 clock enable
Set and cleared by software.
0: Timer 4 clock disabled
1: Timer 4 clock enabled

Bit 1 **TIM3EN**: Timer 3 clock enable
Set and cleared by software.
0: Timer 3 clock disabled
1: Timer 3 clock enabled

Bit 0 **TIM2EN**: Timer 2 clock enable
Set and cleared by software.
0: Timer 2 clock disabled
1: Timer 2 clock enabled

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

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

8.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5E N	UART4E N	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					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

Timers

General purpose timers are capable of:

- **Counters**
- Input Capture
- Output Compare
- PWM Generation

Timers

For Counters we will need to first set prescalar value which will be used to reduce the clock speed of the timer.

Then the Auto Reload Register where when the counter reaches this value it will restart automatically.

15.4.11 TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

15.4.12 TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Prescaler value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 15.3.1: Time-base unit* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

Timers

When the counter overflows we want it to generate an interrupt event since we are counting a certain time to for example blink an LED at certain intervals or creating a delay function.

Now we enable the NVIC interrupt as we did in the GPIO interrupt where the only change is the position of the timer depending on which timer you use.

Then you will create the timer interrupt function to handle the interrupt.

15.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled.
1: Update interrupt enabled.

Timers

Now that everything is setup all that is needed is to start the counter.

To view the current value of the counter just read the counter register.

Lastly when an interrupt occurs and you want to find out what caused the timer interrupt check the status register then clear the bit.

Bit 0 **UIF**: Update interrupt flag
- This bit is set by hardware on an update event. It is cleared by software.
  0: No update occurred.
  1: Update interrupt pending. This bit is set by hardware when the registers are updated:
- At overflow or underflow and if the UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

15.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

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

Bit 0 **CEN**: Counter enable
0: Counter disabled
1: Counter enabled
Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.
CEN is cleared automatically in one-pulse mode, when an update event occurs.

15.4.10 TIMx counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bits 15:0 **CNT[15:0]**: Counter value

15.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Timers

Delay using counter example:

```
void Initialize Timers(void){
RCC->APB1ENR |= (1<<0); //Enable Timer 2 clock
NVIC->ISER[0] |= (1<<28); //Enable Timer 2 Interrupt
NVIC->IP[28] |= (1<<3); //High priority for Timer 2 Interrupt
TIM2->PSC = 7199; //Set prescaler 7200 - 1
TIM2->DIER |= (1<<0); //Enable interrupt
}
```

Timers

```
void delay(int del){
delay_Flag = 1;
TIM2->ARR = del*10; //Reload at desired delay
TIM2->CR1 |= (1<<0); //Enable Counter
while(delay_Flag){} //Wait for Timer
}
void TIM2_IRQHandler(){
if(TIM2->SR & (1<<0)){ //Check if interrupt is due to counter update
TIM2->SR &= (0<<0); //Reset register
delay_Flag = 0; //Reset flag
TIM2->CR1 &= (0<<0); //Disable Counter
}
}
```

Timers

SysTick Timer:

The SysTick Timer is a downcounting timer that can be used for the delay function however its downside is there is only 2 prescalar options:

Either no prescalar or /8 which means that depending on our system clock we can either have long delays or short delays.

Ex: if clock speed is 8 MHz then using the /8 prescalar we get 1 MHz.

Since the SysTick Timer is 24 bits then our limit in the register is 16,777,216.

Meaning our maximum count value is around 16 seconds.

Timers

SysTick Timer:

First we initialize the control register.

We set the clock source to AHB/8 and the Tick interrupt to 1.

Unlike previous interrupts the SysTick Timer does not require setting up the NVIC.

The interrupt handler for the SysTick Timer is SysTick_Handler.

Load the value of the timer should start from in the SysTick reload value register.

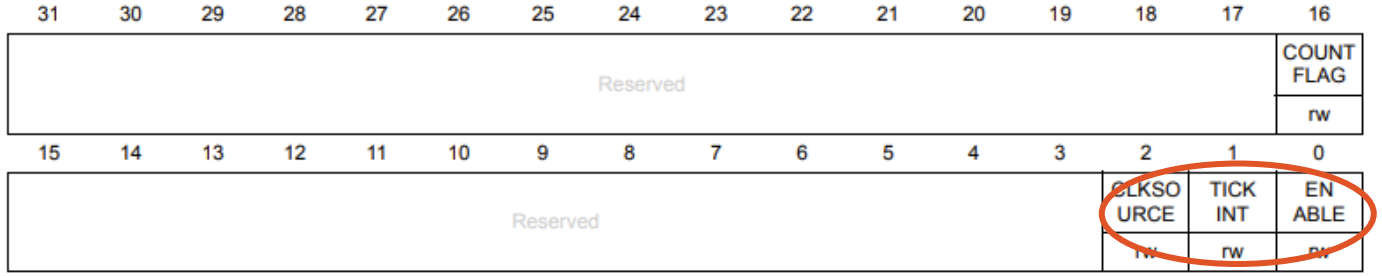
4.5.1 SysTick control and status register (STK_CTRL)

Address offset: 0x00

Reset value: 0x0000 0000

Required privilege: Privileged

The SysTick CTRL register enables the SysTick features.



Bit 2 **CLKSOURCE:** Clock source selection
Selects the clock source.
0: AHB/8
1: Processor clock (AHB)

Bit 1 **TICKINT**: SysTick exception request enable
0: Counting down to zero does not assert the SysTick exception request
1: Counting down to zero to asserts the SysTick exception request.
Note: Software can use COUNTFLAG to determine if SysTick has ever counted to zero.

Bit 0 **ENABLE**: Counter enable
Enables the counter. When ENABLE is set to 1, the counter loads the RELOAD value from the LOAD register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.
0: Counter disabled
1: Counter enabled

Timers

Code Ex for blinking LED:

```
int flag = 0;


int main(void){
RCC->APB2ENR |= (1<<2); //Enable GPIOA Clock
GPIOA->CRL &= 0; //Clear CRL Register
GPIOA->CRL |= (1<<5); //Set PA1 to Output
SysTick->CTRL |= (1<<1); //Enable SysTick Interrupt
while(1){
GPIOA->ODR ^= (1<<1); //Toggle PA1
delay(200); //Delay
}
}
```

Timers

```
void delay(int delay){
SysTick->LOAD = delay*1000 - 1; //Load Delay value to Load Register
SysTick->CTRL |= (1<<0); //Enable Counter
flag = 1;
while(flag);
}
void SysTick_Handler(void){
if(SysTick->CTRL & (1<<16)){
SysTick->CTRL &= ~(1<<16);
SysTick->CTRL &= ~(1<<0);
flag = 0;
}
}
```

Timers

General purpose timers are capable of:

- Counters
- **Input Capture**
- Output Compare
- PWM Generation

Timers

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}
0001: $f_{SAMPLING}=f_{CK\_INT}$, N=2
0010: $f_{SAMPLING}=f_{CK\_INT}$, N=4
0011: $f_{SAMPLING}=f_{CK\_INT}$, N=8
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8
1000: $f_{SAMPLING}=f_{DTS}/8$, N=6
1001: $f_{SAMPLING}=f_{DTS}/8$, N=8
1010: $f_{SAMPLING}=f_{DTS}/16$, N=5
1011: $f_{SAMPLING}=f_{DTS}/16$, N=6
1100: $f_{SAMPLING}=f_{DTS}/16$, N=8
1101: $f_{SAMPLING}=f_{DTS}/32$, N=5
1110: $f_{SAMPLING}=f_{DTS}/32$, N=6
1111: $f_{SAMPLING}=f_{DTS}/32$, N=8

Bits 3:2 **IC1PSC**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).
The prescaler is reset as soon as CC1E=0 (TIMx_CCER register).
00: no prescaler, capture is done each time an edge is detected on the capture input
01: capture is done once every 2 events
10: capture is done once every 4 events
11: capture is done once every 8 events

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.
00: CC1 channel is configured as output
01: CC1 channel is configured as input, IC1 is mapped on TI1
10: CC1 channel is configured as input, IC1 is mapped on TI2
11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

15.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. Take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]					IC1F[3:0]			IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

To configure input capture mode, first we must go to the capture/compare mode register.

There are 2 of them that control 2 channels each.

As you can see this register can be seen as either an input capture register or output compare register depending on the CCxS bits.

Each channel can be configured differently.

Set CCxS bits to 01 to map the capture compare register to its respective channel.

Timers

As you saw in the previous slide we can add a prescalar to our capture as well as adding a filter to the input signal if using a high noise device.

Don’t forget to set the polarity for either rising edge or falling edge as well as enabling the capture.

To setup the interrupt we will set the NVIC the same way as we did in the counter mode since every timer has 1 interrupt handle that handles all its interrupts as well as setting the bits for their respective channels.

15.4.9 TIMx capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
		rw	rw			rw	rw			rw	rw			rw	rw

Bit 1 **CC1P**: Capture/Compare 1 output polarity
CC1 channel configured as output:
0: OC1 active high.
1: OC1 active low.
CC1 channel configured as input:
This bit selects whether IC1 or IC1 is used for trigger or capture operations.
0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted.
1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.

Bit 0 **CC1E**: Capture/Compare 1 output enable
CC1 channel configured as output:
0: Off - OC1 is not active.
1: On - OC1 signal is output on the corresponding output pin.
CC1 channel configured as input:
This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.
0: Capture disabled.
1: Capture enabled.

15.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable
0: CC1 interrupt disabled.
1: CC1 interrupt enabled.

Timers

Be advised, if you enter the interrupt and do not clear its flag, no new interrupt can occur for that flag until its cleared, however some bits in the status register can be cleared by reading a certain register such as for input capture interrupt reading the CCR register clears the bit of its respective flag.

To handle different causes of interrupt for the same handler can be done

15.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag
refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag
refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag
refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

If channel CC1 is configured as output:
This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.
0: No match.
1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.

If channel CC1 is configured as input:
This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.
0: No input capture occurred.
1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Timers

General purpose timers are capable of:

- Counters
- Input Capture
- **Output Compare**
- PWM Generation

Timers

Bit 7 **OC1CE:** Output compare 1 clear enable

OC1CE: Output compare 1 Clear Enable
0: OC1Ref is not affected by the ETRF input
1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.
000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).
001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).
011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.
100: Force inactive level - OC1REF is forced low.
101: Force active level - OC1REF is forced high.
110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).
111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.
1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

*Note: **1:** These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=00 (the channel is configured in output).*
__2:__ The PWM mode can be used without validating the preload register only in one-pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.

15.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. Take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Unlike in the input capture mode, to set the output compare you have to clear the CCxS bits.

We will focus mainly on 3 modes:

Set channel high on match, set channel low on match and toggle.

Setting OCxM to 001, 010, 011 will set their respective modes.

Timers

Next you will need to set the CCRx register of the channel to the value you wish to compare with the counter.

The rest of the setup is the same as for the input capture in terms of the CCER, DIER and SR registers.

Do not forget to setup the Prescalar, ARR and counter registers for both input capture and output compare since they will not function without the counter.

Setting up the counter is the same as in counter mode.

Timers

General purpose timers are capable of:

- Counters
- Input Capture
- Output Compare
- **PWM Generation**

Timers

For PWM only 2 things change in comparison to the output compare.

The 2 modes for PWM are active from start of counter till compare or inactive from start of counter till compare. In other words duty cycle and 1 – duty cycle respectively.

The other that is changed is the OCxFE bit in the CCMRx registers for enabling fast PWM.

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.
0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.
1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Now to get the duty cycle for the PWM it will be the value assigned CCRx register over the value assigned to the ARR register.

The rest of the setup is the same as for the output compare.

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- **ADC**
- UART
- SPI
- I2C
- FreeRTOS

ADC

- To use the ADC, first we need to enable the ADC clock.
- Things to keep in mind is that the ADC has a maximum clock speed of 14 MHz
- So the ADCPRE bits in the Clock Configuration Register (RCC_CFGR) need to be configured to not exceed 14 MHz.

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

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

Bit 10 **ADC2EN**: ADC 2 interface clock enable
Set and cleared by software.
0: ADC 2 interface clock disabled
1: ADC 2 interface clock enabled

Bit 9 **ADC1EN**: ADC 1 interface clock enable
Set and cleared by software.
0: ADC 1 interface disabled
1: ADC 1 interface clock enabled

ADC

- After enabling the ADC clock we need to begin setup of the ADC.
- First we need to make sure ADC is off.
- Then we configure the ADC to our choosing.
- The ADC has multiple parameters that can be configured:
  - Trigger
  - Data Allignment
  - Interrupts
  - Mode of operation

11.12.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

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

Bit 0 **ADON**: A/D converter ON / OFF

This bit is set and cleared by software. If this bit holds a value of zero and a 1 is written to it then it wakes up the ADC from Power Down state.

Conversion starts when this bit holds a value of 1 and a 1 is written to it. The application should allow a delay of t_{STAB} between power up and start of conversion. Refer to *Figure 23.*

0: Disable ADC conversion/calibration and go to power down mode.

1: Enable ADC and to start conversion

Note: If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.

ADC

- We can trigger the ADC by writing a 1 to the ADON bit or configuring an external trigger.
- There are timer trigger events, external interrupt event or software start(SWSTART) which is technically the same as writing a 1 to ADON bit.
- When using SWSTART as trigger you will trigger the ADC by writing one to SWSTART bit.
- For alignment there is left and right alignment.
- By default ADC is right alignment.

Bit 11 **ALIGN**: Data alignment

This bit is set and cleared by software. Refer to *Figure 27.*and *Figure 28.*

0: Right Alignment

1: Left Alignment

11.12.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

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

Bit 20 **EXTTRIG**: External trigger conversion mode for regular channels

This bit is set and cleared by software to enable/disable the external trigger used to start conversion of a regular channel group.

0: Conversion on external event disabled

1: Conversion on external event enabled

Bits 19:17 **EXTSEL[2:0]**: External event select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:

For ADC1 and ADC2, the assigned triggers are:

000: Timer 1 CC1 event

001: Timer 1 CC2 event

010: Timer 1 CC3 event

011: Timer 2 CC2 event

100: Timer 3 TRGO event

101: Timer 4 CC4 event

110: EXTI line 11/TIM8_TRGO event (TIM8_TRGO is available only in high-density and XL-density devices)

111: SWSTART

For ADC3, the assigned triggers are:

000: Timer 3 CC1 event

001: Timer 2 CC3 event

010: Timer 1 CC3 event

011: Timer 8 CC1 event

100: Timer 8 TRGO event

101: Timer 5 CC1 event

110: Timer 5 CC3 event

111: SWSTART

ADC

- For interrupts we will use the end of conversion interrupt so the interrupt will occur when conversion ends.
- You should clear the flag by writing a 0 to the EOC bit in the status register.

11.12.2 ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bit 5 **EOCIE:** Interrupt enable for EOC

This bit is set and cleared by software to enable/disable the End of Conversion interrupt.

0: EOC interrupt disabled

1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

11.12.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

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 1 **EOC:** End of conversion

This bit is set by hardware at the end of a group channel conversion (regular or injected). It is cleared by software or by reading the ADC_DR.

0: Conversion is not complete

1: Conversion complete

ADC

- ADC modes of operation.
- Single conversion mode: converts the sequence of channels once upon trigger.
- Continuous mode: continuously converts the sequence of channels upon trigger until ADC is turned off.
- Discontinuous mode: Converts a subset of the sequence (up to 8 conversions) upon trigger.
- Scan mode: Required only when having multiple channels in sequence.

11.12.2 ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 15:13 **DISCNUM[2:0]**: Discontinuous mode channel count
These bits are written by software to define the number of regular channels to be converted in discontinuous mode, after receiving an external trigger.
000: 1 channel
001: 2 channels
.....
111: 8 channels

Bit 11 **DISCEN**: Discontinuous mode on regular channels
This bit set and cleared by software to enable/disable Discontinuous mode on regular channels.
0: Discontinuous mode on regular channels disabled
1: Discontinuous mode on regular channels enabled

Bit 8 **SCAN:** Scan mode
This bit is set and cleared by software to enable/disable Scan mode. In Scan mode, the inputs selected through the ADC_SQRx or ADC_JSQRx registers are converted.
0: Scan mode disabled
1: Scan mode enabled
Note: An EOC or JEOC interrupt is generated only on the end of conversion of the last channel if the corresponding EOCIE or JEOCIE bit is set

ADC

- ADC modes of operation continued.
- To use single conversion mode have the discontinuous and continuous modes disabled.

11.12.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

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

Bit 1 **CONT:** Continuous conversion

This bit is set and cleared by software. If set conversion takes place continuously till this bit is reset.

0: Single conversion mode

1: Continuous conversion mode

ADC

- Next we configure the sequence of conversions.
- We need to tell the ADC how many conversions are in the sequence as well as giving it the sequence (this must be done even when using one channel only).
- There are 3 sequence registers which the sequence starts from the least significant bit in the third register until right before the length bits in the first sequence register.
- The sequence is designated by SQx where x is the position in sequence.

11.12.9 ADC regular sequence register 1 (ADC_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

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 31:24 Reserved, must be kept at reset value.

Bits 23:20 **L[3:0]**: Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

.....

1111: 16 conversions

ADC

- You will write the channel number in each SQx until you reach the length of your sequence.
- Not writing till you reach the length of sequence will leave the default value of channel 0 and writing more than the length will be discarded (will still be in the register but not read by microcontroller).

11.12.11 ADC regular sequence register 3 (ADC_SQR3)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		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
SQ4_0	SQ3[4:0]					SQ2[4:0]					SQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:25 **SQ6[4:0]**: 6th conversion in regular sequence
These bits are written by software with the channel number (0..17) assigned as the 6th in the sequence to be converted.

Bits 24:20 **SQ5[4:0]:** 5th conversion in regular sequence

Bits 19:15 **SQ4[4:0]**: fourth conversion in regular sequence

Bits 14:10 **SQ3[4:0]**: third conversion in regular sequence

Bits 9:5 **SQ2[4:0]:** second conversion in regular sequence

Bits 4:0 **SQ1[4:0]**: first conversion in regular sequence

ADC

- Next you will need to assign the number of cycles needed for conversion.
- This will depend on the device you are using.
- SMPx where the x is for its respective channel and there are 2 registers.

$$Tconv = \frac{\text{Sampling time + 12.5 cycles}}{\text{ADC CLOCK}}$$

- Tconv is the desired conversion time.
- Sampling time will be in ADC cycles.
- Ex: if sampling time resulted in 72 cycles you will choose the option above it.

11.12.5 ADC sample time register 2 (ADC_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
Res.		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
SMP 5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:0 **SMPx[2:0]:** Channel x Sample time selection

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

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

Note: ADC3 analog input Channel9 is connected to V_{SS}.

ADC

- Somethings to note:
- Anything related to injected channels is out of scope.
- When having a sequence bigger than 1, the EOC bit will only trigger on the end of the sequence, so only the last channel will be in the Data register of the ADC. DMA is used with this but DMA is out of scope so to fix this, enable discontinuous mode and set limit to one so you can go through the sequence one by one with each trigger. Or you can just have the sequence of length 1 and change the ADC channel before every conversion.
- Continuous mode is out of scope.
- Don’t forget to set the GPIO pins as analog.
- Reading the data register also clears the EOC flag.
- ADC requires 14 cycles after turning on.
- There is a calibration option in the ADC control register 2.
- ADC is 12bit resolution
- Don’t forget to set NVIC to use interrupts.

ADC

- In this example we have are using single conversion mode with sequence length 1, using SWSTART as external trigger.
- Our device is a potentiometer.
- `int main(void){`
- `int potentiometer = 0;`
- `RCC->APB2ENR |= (1<<2) | (1<<9); //Enable GPIOA and ADC 1 clock`
- `GPIOA->CRL &= 0; //Clear CRL Register`
- `GPIOA->CRH &= 0; //Clear CRH Register`
- `GPIOA->CRL |= 0x00000000; //Set PA0 as analog input`
- `ADC1->CR1 |= (1<<5); //EOC enable`
- `ADC1->CR2 |= (7<<17) | (1<<20); //Enable SWSTART Trigger / Enable External Trigger`

ADC

```
ADC1->SMPR2 &= ~(7<<3); //Set sampling to 1.5 cycles
ADC1->SQR1 &= ~(15<<20); //Set to 1 conversion
ADC1->CR2 |= (1<<0); //ADC On
for(int i = 0; i<1000000;i++); //Wait for ADC to initialize
ADC1->CR2 |= (1<<2); //Start Calibration
while(ADC1->CR2 & (1<<2)); //Wait for calibration to finish
while(1){
ADC1->CR2 |= (1<<22); //Start conversion
while(ADC1->SR == ~(1<<1)); //Wait for conversion to finish
ADC1->SR &= ~(1<<1); //Reset Status Register
potentiometer = ADC1->DR; //Read ADC data
}
}
```

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- ADC
- **UART**
- SPI
- I2C
- FreeRTOS

UART

- UART is serial communication protocol which is found in the STM32 called USART.
- USART just means that it is capable of synchronous as well as asynchronous communication however synchronous is out of scope.
- As always the first thing is enable the clock for the UART as well as the AFIO clock since alternate functions are required.
- There is only USART1, USART2, USART3 in the blue pill.

Bit 14 **USART1EN:** USART1 clock enable
Set and cleared by software.
0: USART1 clock disabled
1: USART1 clock enabled

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

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.	USART1EN	Res.	SPI1EN	TIM1EN	ADC2EN	ADC1EN	Reserved		IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Res.	AFIOEN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

7.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DACEN	PWREN	BKPEN	Res.	CANEN	Res.	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	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
SPI3EN	SPI2EN	Reserved		WWDGEN	Reserved		TIM14EN	TIM13EN	TIM12EN	TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

UART

- First we turn off the UART.
- We can’t modify UART while it is on.
- Only Control register 1 is need for asynchronous mode.
- Choose word length of 8 data bits.
- By default number of stop bits is 1 but can be changed in Control Register 2, however it is not recommended since most devices use 1 stop bit.
- Enable Transmitter and receiver as needed.
- You don’t need both the transmitter and receiver enabled, you could enable one as your application requires.

27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

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		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 13 **UE**: USART enable

When this bit is cleared the USART prescalers and outputs are stopped and the end of the current
byte transfer in order to reduce power consumption. This bit is set and cleared by software.
0: USART prescaler and outputs disabled
1: USART enabled

Bit 12 **M**: Word length

This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 Data bits, n Stop bit
1: 1 Start bit, 9 Data bits, n Stop bit
Note: The M bit must not be modified during a data transfer (both transmission and reception)

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled
Note: 1: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in Smartcard mode.
2: When TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit

UART

- For interrupts there is:
  - Transmission complete interrupt (TCIE)
  - Transmission data register empty (TXEIE)
  - Receive register not empty (RXNEIE)
- There respective bits that are set in the Status Register are TC, TXE, RXNE respectively.
- Don’t forget clear flag by writing 0.

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by a software sequence (a read from the USART_SR register followed by a write to the USART_DR register). The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.

0: Transmission is not complete

1: Transmission is complete

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_DR register. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. This clearing sequence is recommended only for multibuffer communication.

0: Data is not received

1: Received data is ready to be read.

27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

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		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **TXEIE**: TXE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TXE=1 in the USART SR register

Bit 6 **TCIE**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TC=1 in the USART_SR register

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_SR register

27.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0

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						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register

1: Data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

UART

- Next we set the baud rate.
- The Equation is: $\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$
- Where Fck is microcontroller clock.
- Calculate the USARTDIV.
- Take the number before the decimal point and put it in the DIV_Mantissa section.
- The value after decimal point is multiplied by 16 then rounded to nearest whole number and put in the DIV_Fraction section.

27.6.3 Baud rate register (USART_BRR)

Note: *The baud counters stop counting if the TE or RE bits are disabled respectively.*

Address offset: 0x08

Reset value: 0x0000

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
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV)

UART

- Lastly enable the UART by writing 1 to UE bit in Control register 1.
- Somethings to note:
- USART1 Max clock speed is 72 MHz while USART2 and USART3 have a max clock of 36 MHz.
- GPIO Pins need to be set correctly for UART functionality

USART pinout	Configuration	GPIO configuration
USARTx_TX(1)	Full duplex	Alternate function push-pull
	Half duplex synchronous mode	Alternate function push-pull
USARTx_RX	Full duplex	Input floating / Input pull-up
	Half duplex synchronous mode	Not used. Can be used as a general IO

- Don’t forget to set NVIC to use interrupts.

UART

- In this example no interrupts and only transmission is required using USART3 and baud rate of 115200.

```
void Initialize_UART3(void) {
RCC->APB1ENR |= RCC_APB1ENR_USART3EN; //Enable USART3
USART3->CR1 &= 0; //Clear control register
uint16_t uartdiv = 36000000 / 115200; //Set baud rate
USART3->BRR = (((uartdiv / 16) << USART_BRR_DIV_Mantissa_Pos)
| ((uartdiv % 16) << USART_BRR_DIV_Fraction_Pos));
USART3->CR1 |= (USART_CR1_TE | USART_CR1_UE);
}
```

UART

```
void UART3_SendChar(char c) {
USART3->DR = c;    // Load the Data
while (!(USART3->SR & (1 << 7)))
; // Wait for TC to SET.. This indicates that the data has been transmitted
}

void UART3_SendString(char *string) {
while (*string) {
UART3_SendChar(*string++);
}
}
```

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- ADC
- UART
- **SPI**
- I2C
- FreeRTOS

SPI

- SPI is serial communication protocol which is found in the STM32.
- First we enable SPI clock.

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

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

7.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

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	UART5 EN	UART4 EN	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

Bit 12 **SPI1EN**: SPI 1 clock enable

Set and cleared by software.

0: SPI 1 clock disabled

1: SPI 1 clock enabled

SPI

- Disable SPI before configuration.
- You can use internal slave select or use the GPIO pins as slave select pins.
- Use Frame format as needed.
- Baud rate control is for setting the output clock.
- Master selection is for selecting master or slave mode.
- Clock polarity is for output level when idle.
- Clock phase is for setting the first clock transition as data capture or second transition.

25.5.1 SPI control register 1 (SPI_CR1) (not used in I²S mode)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	*LSB FIRST*	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 9 **SSM:** Software slave management
When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.
0: Software slave management disabled
1: Software slave management enabled
Note: This bit is not used in I²S mode

Bit 8 **SSI:** Internal slave select
This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the IO value of the NSS pin is ignored.
Note: This bit is not used in I²S mode

Bit 7 **LSBFIRST:** Frame format
0: MSB transmitted first
1: LSB transmitted first
Note: This bit should not be changed when communication is ongoing. It is not used in I²S mode

Bit 6 **SPE:** SPI enable
0: Peripheral disabled
1: Peripheral enabled
Note: This bit is not used in I²S mode.
When disabling the SPI, follow the procedure described in Section 25.3.8.

ts 5:3 **BR[2:0]:** Baud rate control
000: $f_{PCLK}/2$
001: $f_{PCLK}/4$
010: $f_{PCLK}/8$
011: $f_{PCLK}/16$
100: $f_{PCLK}/32$
101: $f_{PCLK}/64$
110: $f_{PCLK}/128$
111: $f_{PCLK}/256$
Note: These bits should not be changed when communication is ongoing. They are not used in I²S mode.

Bit 2 **MSTR:** Master selection
0: Slave configuration
1: Master configuration
Note: This bit should not be changed when communication is ongoing. It is not used in I²S mode.

Bit1 **CPOL:** Clock polarity
0: CK to 0 when idle
1: CK to 1 when idle
Note: This bit should not be changed when communication is ongoing. It is not used in I²S mode

Bit 0 **CPHA:** Clock phase
0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge
Note: This bit should not be changed when communication is ongoing. It is not used in I²S mode

SPI

- For interrupts there is TX buffer empty and RX buffer not empty and their respective flags in status register.
- SSOE dictates whether slave select is active.
- Now that configuration is done

25.5.2 SPI control register 2 (SPI_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Res.	Res.	*SSOE*	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

Bit 7 **TXEIE:** Tx buffer empty interrupt enable
0: TXE interrupt masked
1: TXE interrupt not masked. Used to generate an interrupt request when the TXE flag is set.

Bit 6 **RXNEIE:** RX buffer not empty interrupt enable
0: RXNE interrupt masked
1: RXNE interrupt not masked. Used to generate an interrupt request when the RXNE flag is set.

Bit 2 **SSOE:** SS output enable
0: SS output is disabled in master mode and the cell can work in multimaster configuration
1: SS output is enabled in master mode and when the cell is enabled. The cell cannot work in a multimaster environment.
Note: This bit is not used in I^2S mode

SPI

- Somethings to note:
- AFIO clock is needed.
- GPIO configuration for SPI.
- Configuration must be the same master and slave devices.
- Don’t forget to set NVIC for interrupts.

SPI pinout	Configuration	GPIO configuration
SPIx_SCK	Master	Alternate function push-pull
	Slave	Input floating
SPIx_MOSI	Full duplex / master	Alternate function push-pull
	Full duplex / slave	Input floating / Input pull-up
	Simplex bidirectional data wire / master	Alternate function push-pull
	Simplex bidirectional data wire/ slave	Not used. Can be used as a GPIO
SPIx_MISO	Full duplex / master	Input floating / Input pull-up
	Full duplex / slave (point to point)	Alternate function push-pull
	Full duplex / slave (multi-slave)	Alternate function open drain
	Simplex bidirectional data wire / master	Not used. Can be used as a GPIO
	Simplex bidirectional data wire/ slave (point to point)	Alternate function push-pull
	Simplex bidirectional data wire/ slave (multi-slave)	Alternate function open drain
SPIx_NSS	Hardware master /slave	Input floating/ Input pull-up / Input pull-down
	Hardware master/ NSS output enabled	Alternate function push-pull
	Software	Not used. Can be used as a GPIO

SPI

- In our example when a button is pressed on one device the other turns on LED and if button is released the LED turns off.
- Master code.

```
int main(void){
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_AFIOEN | RCC_APB2ENR_SPI1EN; //Enable GPIOA / Alt Function / SPI1
GPIOA->CRL &= 0; //Clear CRL Register
GPIOA->CRL |= 0xB4BB2000; //Set PA7 - PA4 as SPI / PA3 as input pull down //Check SS pin
SPI1->CR1 |= (7<<2); //Master mode / baud rate 500 khz
SPI1->CR2 |= (1<<2); //SS enable
SPI1->CR1 |= (1<<6); //Enable SPI
Delay(1000);
```

SPI

```
while(1){
if(GPIOA->IDR & 0b1000){
SPI1->DR = 1; //Send on for LED
while(!(SPI1->SR & 0b10)); //Wait for transfer to finish
}else{
SPI1->DR = 0; //Send off for LED
while(!(SPI1->SR & 0b10)); //Wait for transfer to finish
}
Delay(100);
}
}
```

SPI

- Slave code:
- `int main(void){`
- `RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_AFIOEN | RCC_APB2ENR_SPI1EN;`
  `//Enable GPIOA / Alt Function / SPI1`
- `GPIOA->CRL &= 0; //Clear CRL Register`
- `GPIOA->CRL |= 0x4B442000; //Set PA7 - PA4 as SPI / PA3 as output//Check SS pin`
- `SPI1->CR1 |= (1<<6); //Enable SPI`

SPI

- Slave code:

```
while(1){
if(SPI1->SR & 0b1){
int status = SPI1->DR; //Load status from data registry
if(status){
GPIOA->ODR |= 0b1000; // Turn LED on
}else{
GPIOA->ODR &= ~0b1000; //Turn LED off
}
}
}
}
```

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- ADC
- UART
- SPI
- **I2C**
- FreeRTOS

I2C

- I2C is serial communication protocol which is found in the STM32.
- First we enable I2C clock.

7.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

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	UART5 EN	UART4 EN	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

I2C

- First reset I2C by writing 1 to SWRST and then clearing this bit.
- PE bit must be cleared.
- ACK bit is for returning acknowledgement.
- Start generation is for generating start bit and Stop generation is for stop bit. (works differently for slave mode).

26.6.1 I²C Control register 1 (I2C_CR1)

Address offset: 0x00
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bit 15 **SWRST**: Software reset
When set, the I2C is under reset state. Before resetting this bit, make sure the I2C lines are released and the bus is free.
0: I²C Peripheral not under reset
1: I²C Peripheral under reset state
Note: This bit can be used to reinitialize the peripheral after an error or a locked state. As an example, if the BUSY bit is set and remains locked due to a glitch on the bus, the SWRST bit can be used to exit from this state.

Bit 10 **ACK**: Acknowledge enable
This bit is set and cleared by software and cleared by hardware when PE=0.
0: No acknowledge returned
1: Acknowledge returned after a byte is received (matched address or data)

Bit 9 **STOP**: Stop generation
The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.
In Master Mode:
0: No Stop generation.
1: Stop generation after the current byte transfer or after the current Start condition is sent.
In Slave mode:
0: No Stop generation.
1: Release the SCL and SDA lines after the current byte transfer.

Bit 8 **START**: Start generation
This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.
In Master Mode:
0: No Start generation
1: Repeated start generation
In Slave mode:
0: No Start generation
1: Start generation when the bus is free

I2C

- Set FREQ bits with the APB1 frequency in MHz.
- Next set TRISE register with the max rise time of the SCL feedback loop.
- TRISE is equal to FREQ + 1.

26.6.2 I²C Control register 2 (I2C_CR2)

Address offset: 0x04
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]					
			rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Bits 5:0 **FREQ[5:0]**: Peripheral clock frequency

The FREQ bits must be configured with the APB clock frequency value (I2C peripheral connected to APB). The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The minimum allowed frequency is 2 MHz, the maximum frequency is limited by the maximum APB frequency and cannot exceed 50 MHz (peripheral intrinsic maximum limit).

0b000000: Not allowed
0b000001: Not allowed
0b000010: 2 MHz
...
0b110010: 50 MHz
Higher than 0b101010: Not allowed

26.6.9 I²C TRISE register (I2C_TRISE)

Address offset: 0x20
Reset value: 0x0002

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

Bits 5:0 **TRISE[5:0]**: Maximum rise time in Fm/Sm mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode.
The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.
These bits must be programmed with the maximum SCL rise time given in the I²C bus specification, incremented by 1.
For instance: in Sm mode, the maximum allowed SCL rise time is 1000 ns.
If, in the I2C_CR2 register, the value of FREQ[5:0] bits is equal to 0x08 and T_{PCLK1} = 125 ns therefore the TRISE[5:0] bits must be programmed with 09h.
(1000 ns / 125 ns = 8 + 1)
The filter value can also be added to TRISE[5:0].
If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note: TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

I2C

- For clock control there is standard mode(Sm) and fast mode (Fm).
- For CCR bits we need to get some information about I2C timings from STM32F103C8T6 Datasheet.

Table 40. I^2C characteristics

Symbol	Parameter	Standard mode I^2C(1)(2)		Fast mode I^2C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{w(SCLL)}$	SCL clock low time	4.7	-	1.3	-	µs
$t_{w(SCLH)}$	SCL clock high time	4.0	-	0.6		
$t_{su(SDA)}$	SDA setup time	250	-	100	-	ns
$t_{h(SDA)}$	SDA data hold time	-	3450(3)	-	900(3)	
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time	-	1000	-	300	
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time	-	300	-	300	
$t_{h(STA)}$	Start condition hold time	4.0	-	0.6	-	µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	0.6	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	0.6	-	µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7	-	1.3	-	µs
C_b	Capacitive load for each bus line	-	400	-	400	pF
t_{SP}	Pulse width of spikes suppressed by the analog filter	0	50(4)	0	50(4)	ns

1. Specified by design, not tested in production.
2. f_{PCLK1} must be at least 2 MHz to achieve standard mode I^2C frequencies. It must be at least 4 MHz to achieve fast mode I^2C frequencies. It must be a multiple of 10 MHz to reach the 400 kHz maximum I2C fast mode clock.
3. The maximum Data hold time must be met if the interface does not stretch the low period of SCL signal.
4. The minimum width of the spikes filtered by the analog filter is above t_{SP}(max).

26.6.8 I^2C Clock control register (I2C_CCR)

Address offset: 0x1C
Reset value: 0x0000

Note: *f_{PCLK1} must be at least 2 MHz to achieve Sm mode I^2C frequencies. It must be at least 4 MHz to achieve Fm mode I^2C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I^2C Fm mode clock.*

The CCR register must be configured only when the I2C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **F/S:** I2C master mode selection
0: Sm mode I2C
1: Fm mode I2C

Bit 14 **DUTY:** Fm mode duty cycle
0: Fm mode t_{low}/t_{high} = 2
1: Fm mode t_{low}/t_{high} = 16/9 (see CCR)

Bits 13:12 Reserved, must be kept at reset value

Bits 11:0 **CCR[11:0]:** Clock control register in Fm/Sm mode (Master mode)
Controls the SCL clock in master mode.
Sm mode or SMBus:
$T_{high} = CCR * T_{PCLK1}$
$T_{low} = CCR * T_{PCLK1}$
Fm mode:
If DUTY = 0:
$T_{high} = CCR * T_{PCLK1}$
$T_{low} = 2 * CCR * T_{PCLK1}$
If DUTY = 1:
$T_{high} = 9 * CCR * T_{PCLK1}$
$T_{low} = 16 * CCR * T_{PCLK1}$
For instance: in Sm mode, to generate a 100 kHz SCL frequency:
If FREQ = 08, T_{PCLK1} = 125 ns so CCR must be programmed with 0x28
(0x28 <=> 40d x 125 ns = 5000 ns.)

Note: The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01
$t_{high} = t_{r(SCL)} + t_{w(SCLH)}$. See device datasheet for the definitions of parameters.
$t_{low} = t_{f(SCL)} + t_{w(SCLL)}$. See device datasheet for the definitions of parameters.
I2C communication speed, fSCL ~ 1/(thigh + tlow). The real frequency may differ due to the analog noise filter input delay.
The CCR register must be configured only when the I^2C is disabled (PE = 0).

I2C

- For example if our APB1 clock is 8 MHz and we are using Sm mode then T_{PCLK} is equal to 125ns.
- T_{High} is equal to $T_{r(SCL)} + T_{w(SCLH)}$
- $T_{r(SCL)}$ from datasheet equals to 1000ns and $T_{w(SCLH)}$ equals to 4000ns.
- T_{High} equals 5000ns
- So 5000÷128 equals 40
- So CCR bits equal to 40

26.6.8 I²C Clock control register (I2C_CCR)

Address offset: 0x1C
Reset value: 0x0000

Note: *f_{PCLK1} must be at least 2 MHz to achieve Sm mode I²C frequencies. It must be at least 4 MHz to achieve Fm mode I²C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I²C Fm mode clock.*

The CCR register must be configured only when the I2C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **F/S:** I2C master mode selection
0: Sm mode I2C
1: Fm mode I2C

Bit 14 **DUTY:** Fm mode duty cycle
0: Fm mode $t_{low}/t_{high} = 2$
1: Fm mode $t_{low}/t_{high} = 16/9$ (see CCR)

Bits 13:12 Reserved, must be kept at reset value

Bits 11:0 **CCR[11:0]:** Clock control register in Fm/Sm mode (Master mode)
Controls the SCL clock in master mode.
Sm mode or SMBus:
$T_{high} = CCR * T_{PCLK1}$
$T_{low} = CCR * T_{PCLK1}$
Fm mode:
If DUTY = 0:
$T_{high} = CCR * T_{PCLK1}$
$T_{low} = 2 * CCR * T_{PCLK1}$
If DUTY = 1:
$T_{high} = 9 * CCR * T_{PCLK1}$
$T_{low} = 16 * CCR * T_{PCLK1}$
For instance: in Sm mode, to generate a 100 kHz SCL frequency:
If FREQ = 08, T_{PCLK1} = 125 ns so CCR must be programmed with 0x28
(0x28 <=> 40d x 125 ns = 5000 ns.)

Note: *The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01*
$t_{high} = t_{r(SCL)} + t_{w(SCLH)}$. See device datasheet for the definitions of parameters.
$t_{low} = t_{f(SCL)} + t_{w(SCLL)}$. See device datasheet for the definitions of parameters.
I2C communication speed, fSCL ~ 1/(thigh + tlow). The real frequency may differ due to the analog noise filter input delay.
The CCR register must be configured only when the I²C is disabled (PE = 0).

I2C

- Lastly set PE bit to enable I2C.
- Somethings to note:
- Debug mode can cause issues with I2C.
- Start generation, stop generation and acknowledge bits are not set in configuration but are set when needed in data frame.
- For an example on I2C visit: https://controllerstech.com/stm32-i2c-configuration-using-registers/
- Don’t forget setting NVIC for interrupts.

Content

- Clock Speed Configuration
- GPIO
- GPIO Interrupts
- Timers
- ADC
- UART
- SPI
- I2C
- **FreeRTOS**

FreeRTOS

- This section will only show how to add freeRTOS to your project.
- First create a new project and open the .ioc file generated in your project.
- Click on middleware on the left under the Pinout and configuration tab.
- FreeRTOS will be the second option under FATFS.
- In interface choose CMSIS_V1 then save.
- Click yes on any prompt until the STM32CubeIDE is done generating code.
- Navigate to the Middlewares->Third_Party->FreeRTOS->Source->CMSIS_RTOS folder created in your project.
- Copy the includes at the top of the cmsis_os.h file and add them to your main.c file.
- Finally remove cmsis_os.h include from main.c.
- For FreeRTOS configs go to Core->Inc->FreeRTOSConfig.h.
- For examples go to: https://controllerstech.com/freertos-tutorials/

Referneces

STM32F103xx Reference manual:

https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Cortex M3 Programming manual:

https://www.st.com/resource/en/programming_manual/cd00228163-stm32f10xxx20xxx21xxxl1xxxx-cortexm3-programming-manual-stmicroelectronics.pdf

Flash Memory Programming manual:

https://www.st.com/resource/en/programming_manual/pm0075-stm32f10xxx-flash-memory-microcontrollers-stmicroelectronics.pdf

STM32F103C8T6 Datasheet:

https://www.st.com/resource/en/datasheet/stm32f103c8.pdf