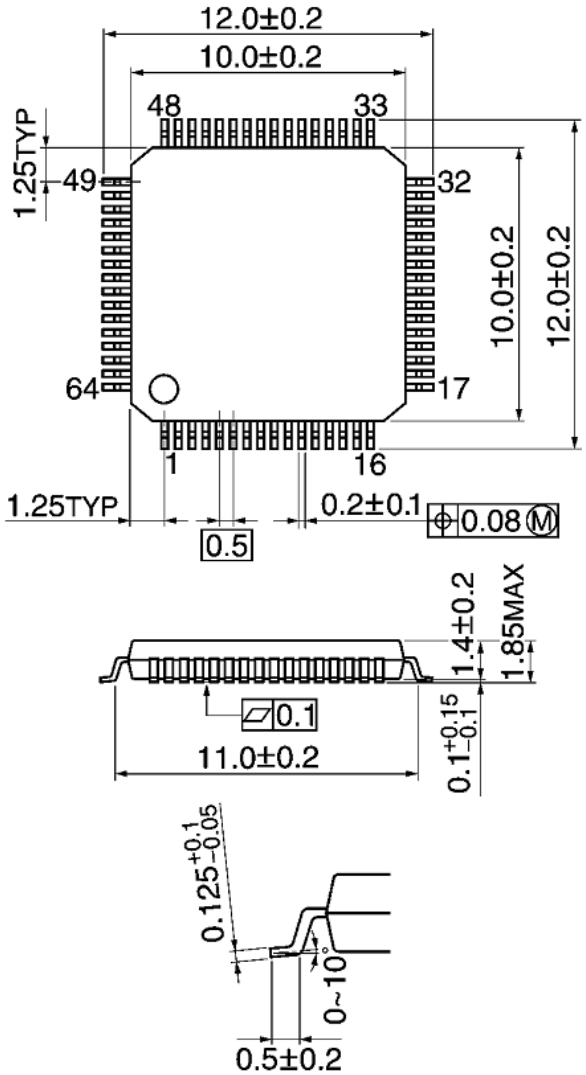
Microcomputer Engineering TMIK13 Lecture 5

INTERRUPTS CONT

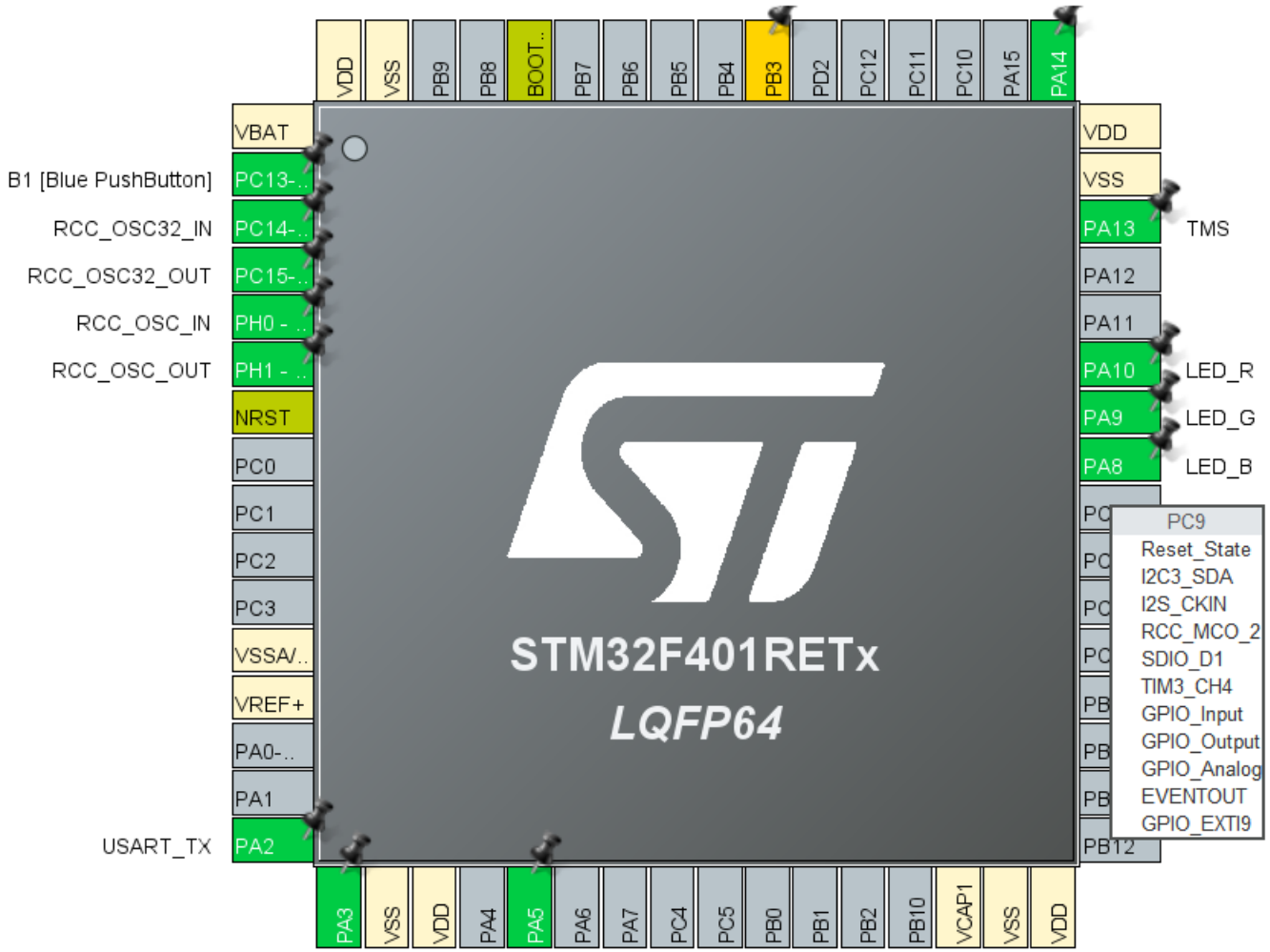
ANDREAS AXELSSON (ANDREAS.AXELSSON@JU.SE)

MCU Pinout

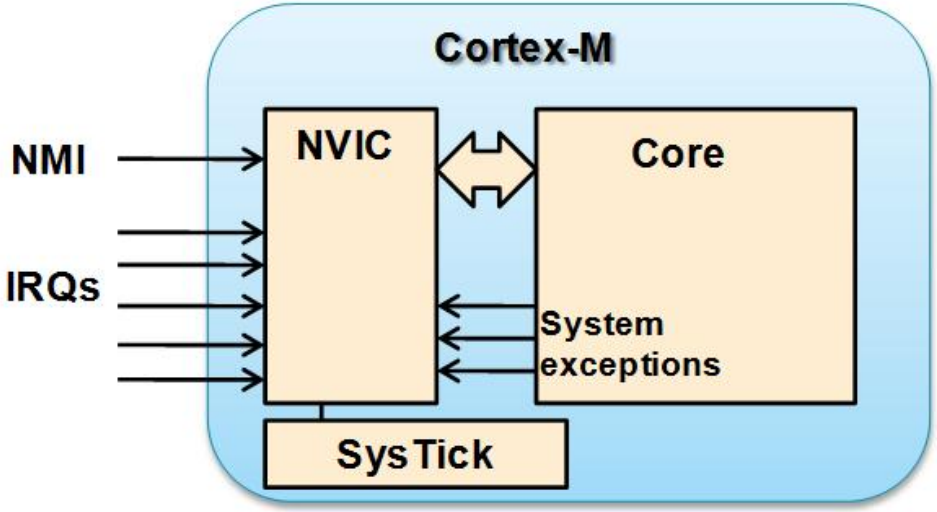

STM32
Cortex

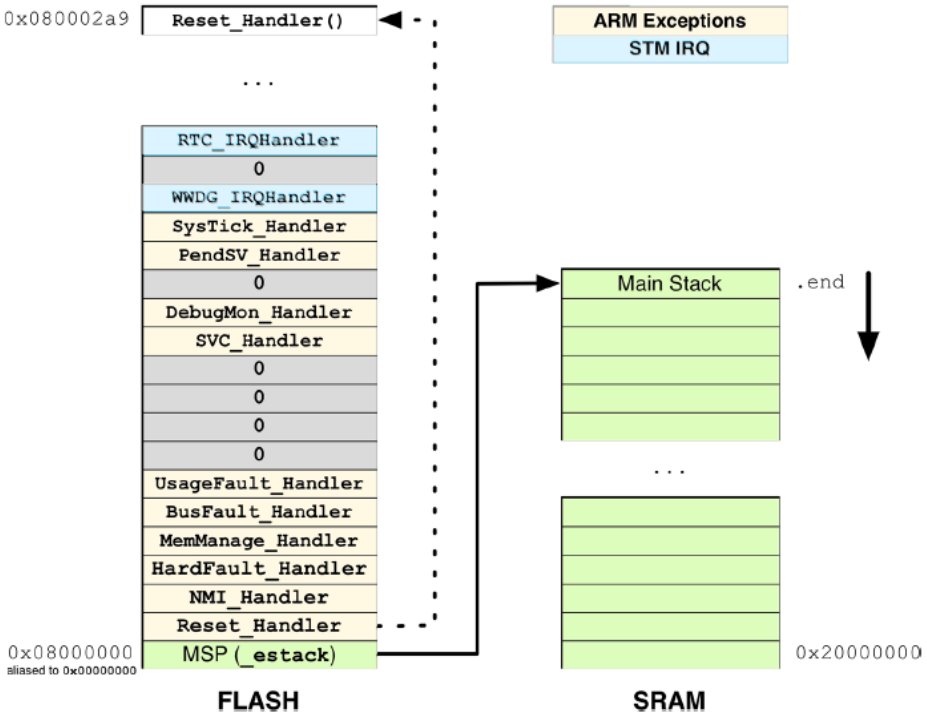

12.0±0.2
10.0±0.2
48
33
1.25TYP
49
32
10.0±0.2
12.0±0.2
64
17
1
16
1.25TYP
0.5
0.2±0.1
0.08 M
1.4±0.2
1.85MAX
0.1
11.0±0.2
0.1+0.15 −0.1
0.125+0.1 −0.05
0~10°
0.5±0.2

Pin Configuration

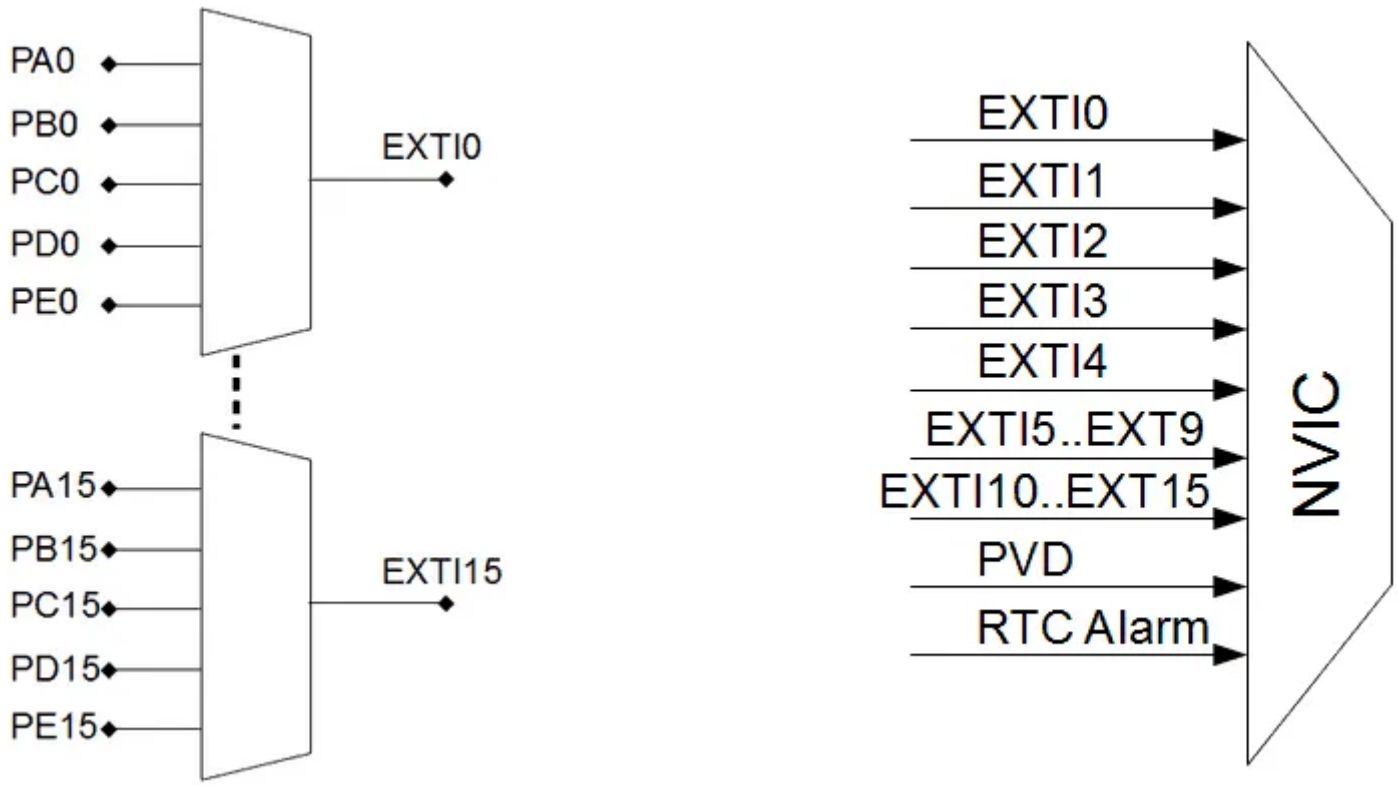


Interrupt vectors





GPIO – EXTI / NVIC



CubeMX – NVIC

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- WWDG

Analog

Timers

Connectivity

Multimedia

Computing

Middleware

NVIC | Code generation

Priority Group: 0 bits for pre-emption priority 4 bits for subpriority

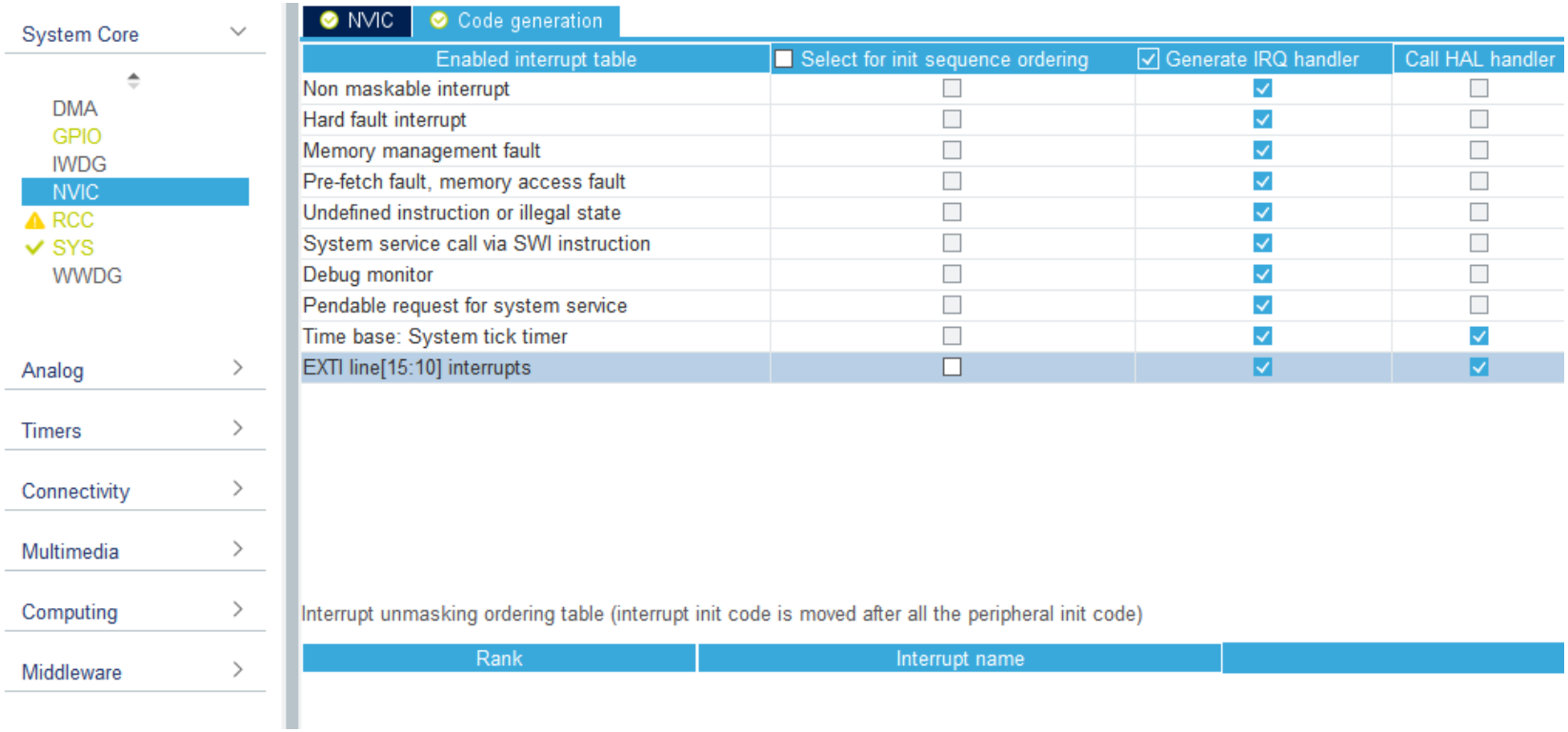
☐ Sort by Premption Priority and Sub Priority

Search: Search (Crtl+F)

☐ Show only enabled interrupts ☑ Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
TIM2 global interrupt	☐	0	0
USART2 global interrupt	☐	0	0
EXTI line[15:10] interrupts	☑	0	0
FPU global interrupt	☐	0	0

CubeMX – NVIC (Code generation)



STM32F4 Interrupts – GPIO

startup_stm32f401xe.s

```
.word     0                                  /* Reserved                          */
.word     EXTI9_5_IRQHandler                 /* External Line[9:5]s               */
.word     TIM1_BRK_TIM9_IRQHandler           /* TIM1 Break and TIM9               */
.word     TIM1_UP_TIM10_IRQHandler           /* TIM1 Update and TIM10             */
.word     TIM1_TRG_COM_TIM11_IRQHandler      /* TIM1 Trigger and Commutation and TIM11 */
.word     TIM1_CC_IRQHandler                 /* TIM1 Capture Compare              */
.word     TIM2_IRQHandler                    /* TIM2                              */
.word     TIM3_IRQHandler                    /* TIM3                              */
.word     TIM4_IRQHandler                    /* TIM4                              */
.word     I2C1_EV_IRQHandler                 /* I2C1 Event                        */
.word     I2C1_ER_IRQHandler                 /* I2C1 Error                        */
.word     I2C2_EV_IRQHandler                 /* I2C2 Event                        */
.word     I2C2_ER_IRQHandler                 /* I2C2 Error                        */
.word     SPI1_IRQHandler                    /* SPI1                              */
.word     SPI2_IRQHandler                    /* SPI2                              */
.word     USART1_IRQHandler                  /* USART1                            */
.word     USART2_IRQHandler                  /* USART2                            */
.word     0                                  /* Reserved                          */
.word     EXTI15_10_IRQHandler               /* External Line[15:10]s             */
.word     RTC_Alarm_IRQHandler               /* RTC Alarm (A and B) through EXTI Line */
.word     OTG_FS_WKUP_IRQHandler             /* USB OTG FS Wakeup through EXTI line */
.word     0                                  /* Reserved                          */
.word     0                                  /* Reserved                          */
```

STM32F4 Interrupts – GPIO

stm32f4xx_it.c

```
/******************************************************************************/
/* STM32F4xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                   */
/******************************************************************************/

/**
* @brief This function handles EXTI line[15:10] interrupts.
*/
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

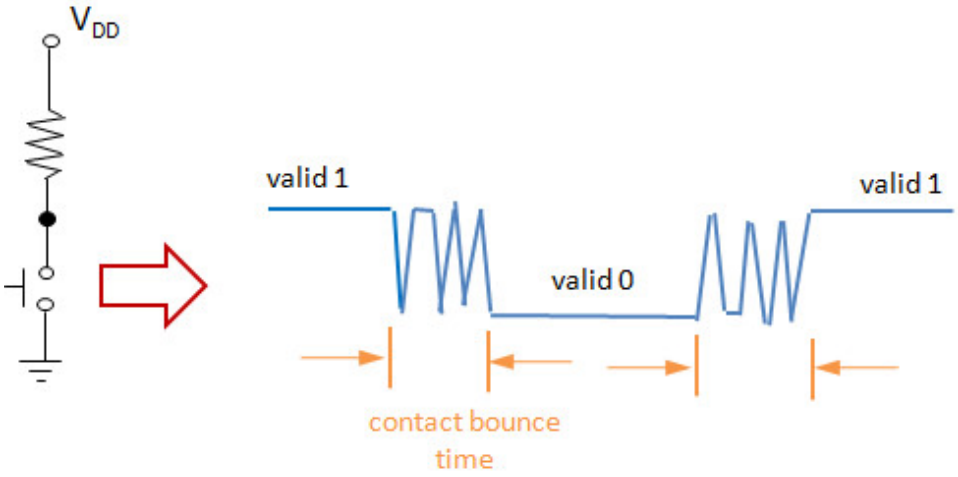
STM32F4 Interrupts – GPIO

Create callback to handle GPIO interrupt

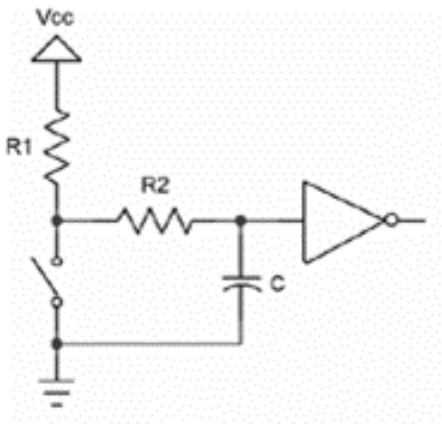
```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
 if (GPIO_Pin == GPIO_PIN_13)
 {
 /* DO the job when EXTI was detected on pin 13 */
 }
}
```

STM32F4 Interrupts – GPIO

What about contact bounces???

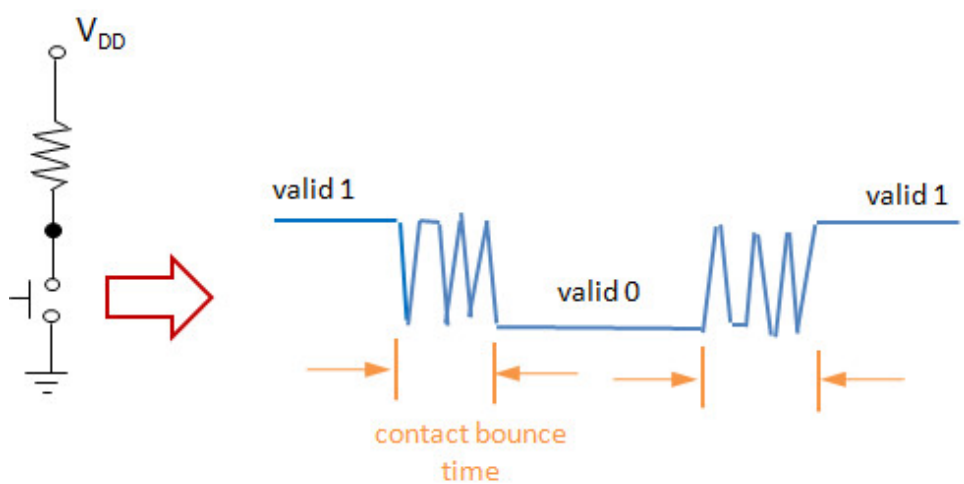


STM32F4 Interrupts – GPIO



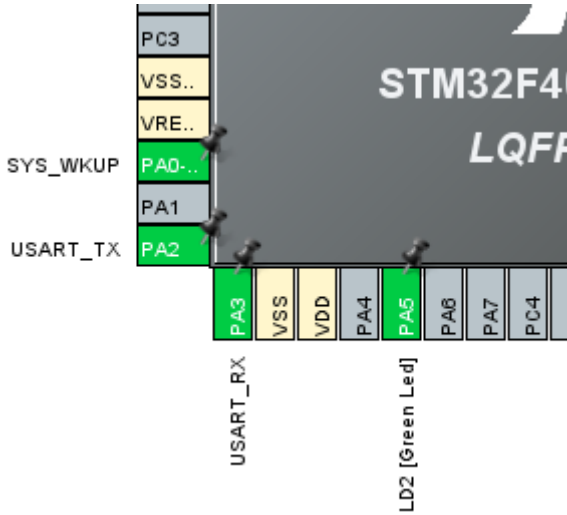
$$V_{IHmin} = V_{DD}\left(1 - e^{\frac{-t}{(R1+R2)C}}\right) => (R_1 + R_2) = \frac{-t}{Cln(1-\frac{V_{IHmin}}{V_{DD}})}$$

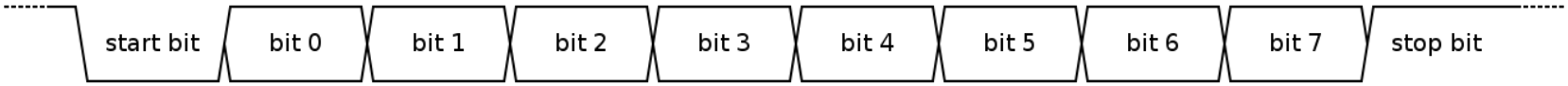
$$V_{ILmax} = V_{DD}\left(e^{\frac{-t}{R2C}}\right) => R2 = \frac{-t}{Cln(\frac{V_{ILmax}}{V_{DD}})}$$



UART – Serial communication

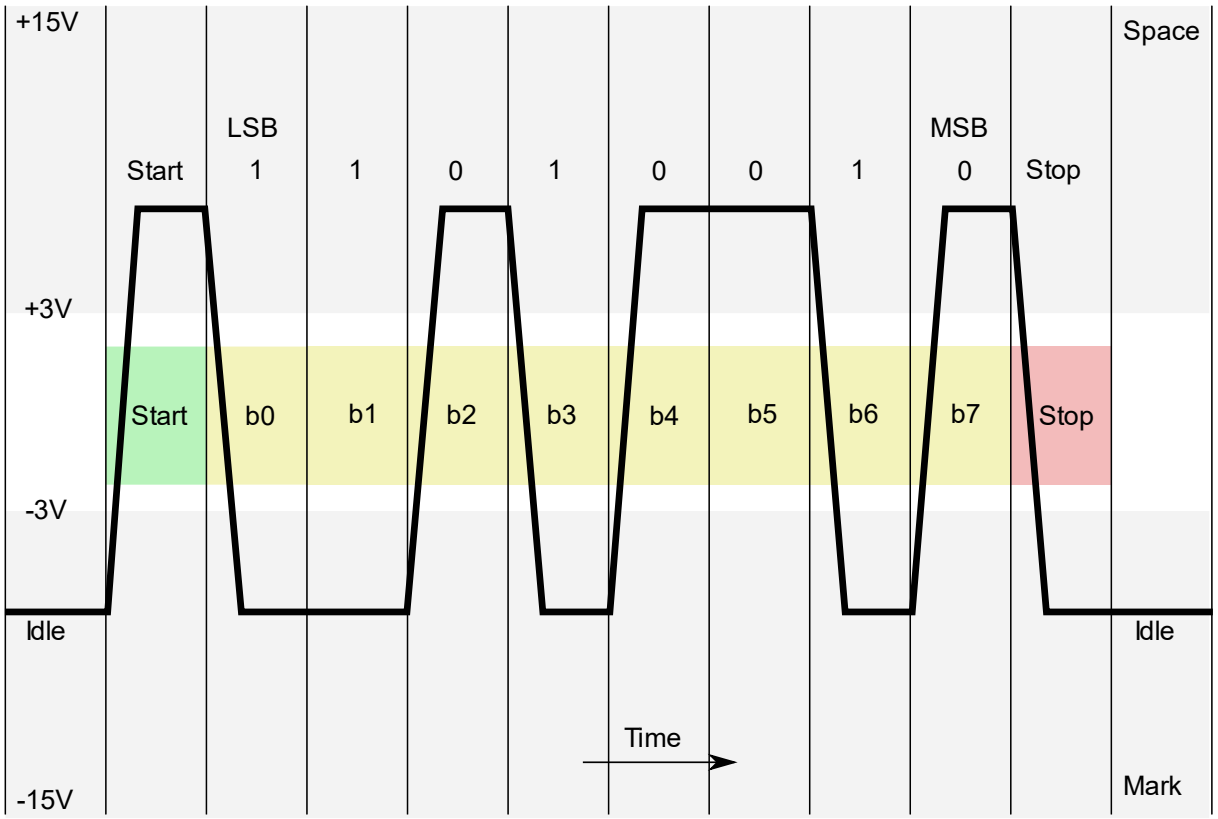
- Asynchronous Communication
  - Uses only RX and TX pins, no clock
- Modes
  - Simplex, Full Duplex and Half Duplex
- Typical bit rates:
  - *4800, 9600, 19200, 38400, 57600, 115200, 230400 bits/s*
- *Requires transmitter and receiver clock to be within 2-4% relative accuracy*





8N1 - format

UART – RS-232

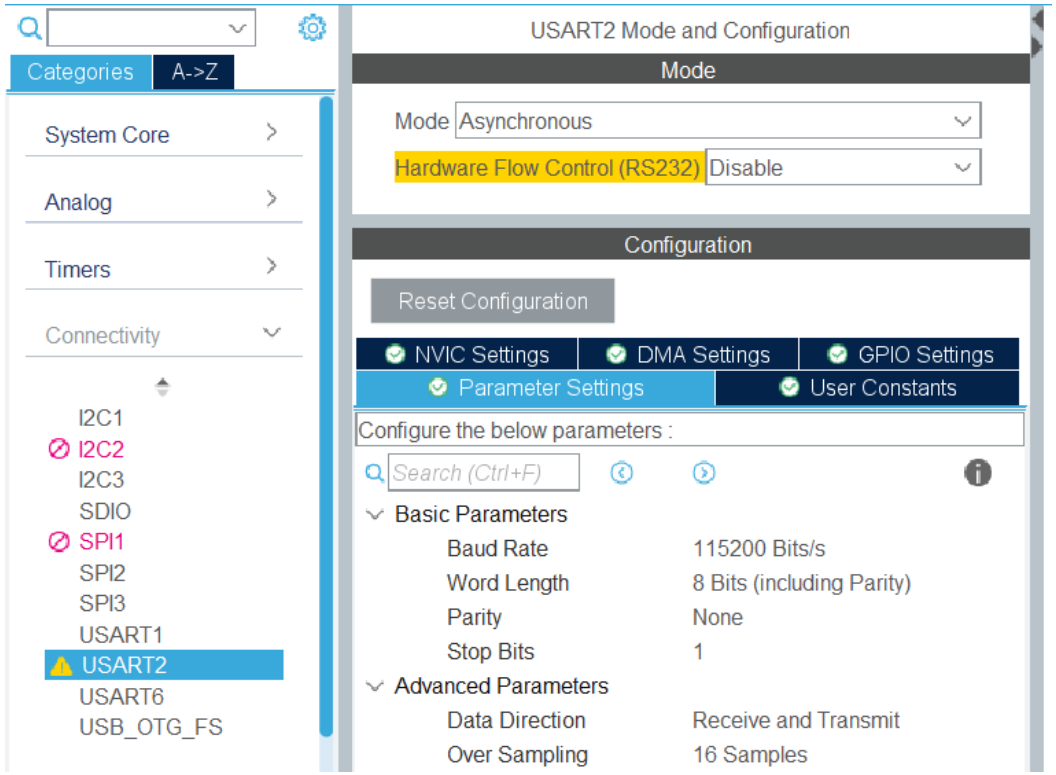


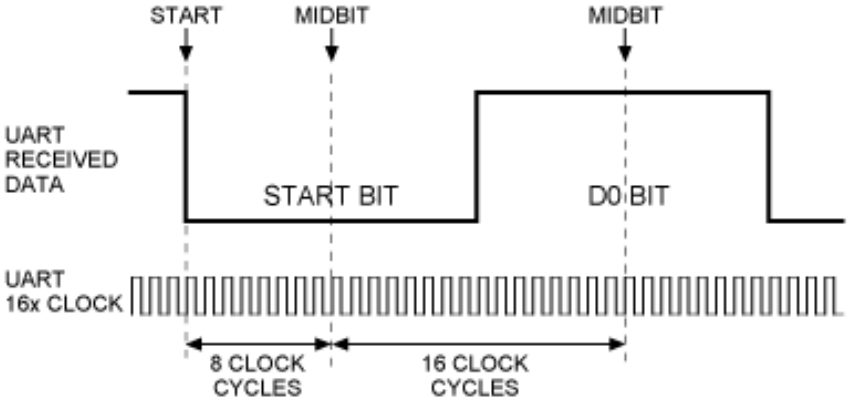
RS-232
”0” = 3 to 15V (Space)
”1” = -3 to -15V (Mark)

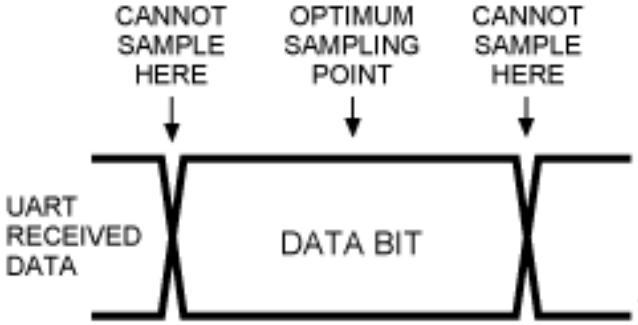
MCU (typical)
”0” = 0V
”1” = 3.3V

https://en.wikipedia.org/wiki/RS-232#Voltage_levels

UART – Serial communication







UART – STM32 HAL API

HAL_UART_Transmit

Function name	**HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Send an amount of data in blocking mode.
Parameters	• **huart:** UART handle. • **pData:** Pointer to data buffer. • **Size:** Amount of data to be sent. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

UM1725 User Manual – Description of STM32F4 HAL and LL drivers

UART – STM32 HAL API

HAL_UART_Receive

Function name	**HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function description	Receive an amount of data in blocking mode.
Parameters	• **huart:** UART handle. • **pData:** pointer to data buffer. • **Size:** amount of data to be received. • **Timeout:** Timeout duration.
Return values	• **HAL:** status

UM1725 User Manual – Description of STM32F4 HAL and LL drivers

UART – Serial with interrupt

```
HAL_UART_Receive_IT(huart, pData, Size);
HAL_UART_Transmit_IT(huart, pData, Size);

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart)
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart)
```

Other possible callbacks available to detect idle-condition, break-condition or any error-conditions.

stm32f4xx_hal_uart.c

```
   Three operation modes are available within this driver :

*** Polling mode IO operation ***
=================================
[..]
  (+) Send an amount of data in blocking mode using HAL_UART_Transmit()
  (+) Receive an amount of data in blocking mode using HAL_UART_Receive()

*** Interrupt mode IO operation ***
==================================
[..]
  (+) Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
  (+) At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_TxCpltCallback
  (+) Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()
  (+) At reception end of transfer HAL_UART_RxCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_RxCpltCallback
  (+) In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can
       add his own code by customization of function pointer HAL_UART_ErrorCallback

*** DMA mode IO operation ***
==============================
[..]
  (+) Send an amount of data in non blocking mode (DMA) using HAL_UART_Transmit_DMA()
  (+) At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
  (+) At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_TxCpltCallback
  (+) Receive an amount of data in non blocking mode (DMA) using HAL_UART_Receive_DMA()
  (+) At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback
  (+) At reception end of transfer HAL_UART_RxCpltCallback is executed and user can
       add his own code by customization of function pointer HAL_UART_RxCpltCallback
  (+) In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can
       add his own code by customization of function pointer HAL_UART_ErrorCallback
  (+) Pause the DMA Transfer using HAL_UART_DMAPause()
  (+) Resume the DMA Transfer using HAL_UART_DMAResume()
  (+) Stop the DMA Transfer using HAL_UART_DMAStop()
```

Microcomputer Engineering

Questions?

Contact information

Andreas Axelsson

Email: andreas.axelsson@ju.se

Mobile: 0709-467760