Joint ICTP, SAIFR and UNESP School on Systems-on-Chip, Embedded Microcontrollers and their Applications in Research and Industry

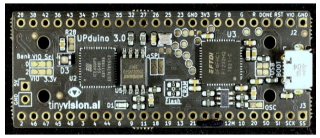
STM32 Evaluation and Development Boards

Dr. Luigi Calligaris (SPRACE/UNESP)

Day 2 - 19/10/2021

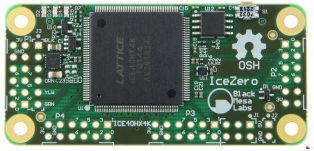
Evaluation and Development kits

- Chip manufacturers want to sell their product
  - It's in their interest to make them easy to use by developers
  - **Evaluation and Development kits** provide a platform:
    - Easy to set-up and ready to use out of the box
    - With an well-developed ecosystem of examples
    - (often) Which can be used as a design reference
- Devkits are made by the same company or by a partner
  - These kits are subsidized, so (in times of no chip shortage) price is attractive.
  - There are some rather cheap MCU, FPGA and (to a lesser extent) SoC kits
    - MCUs: 6 - 35$ FPGA: 25-40$ FPGA SoCs: less than 200 $
- Usually the devkits include an onboard debugger
  - More practical, cheaper to get started on specific hardware
- We focus on the **NUCLEO-STM32F411RE** MCU boards

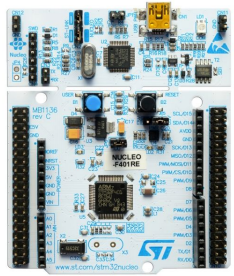


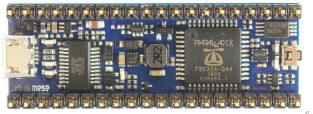
UPduino (Lattice ICE40UP5K FPGA)

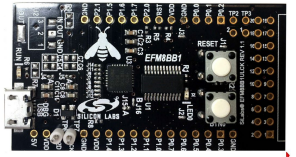

LPCXPresso (Many NXP MCU)



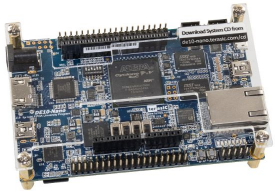
IceZero (Lattice ICE40HX4K FPGA)

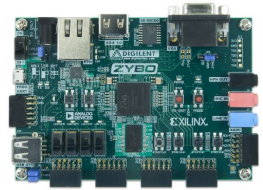

NUCLEO (Many STM32 MCU)


Propeller FLiP (an esoteric multicore MCU)



EFM8BB1LCK (SiLabs EFM8BB MCU)

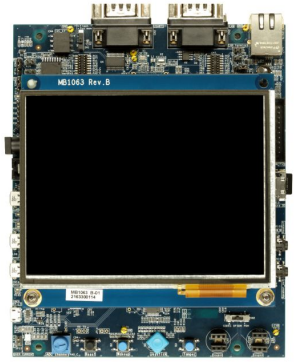

Terasic DE10-Nano (Cyclone V FPGA SoC)



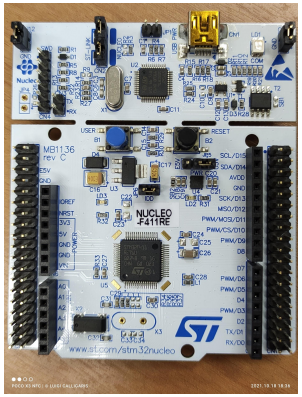
ZyBo Z7 (ZYNQ FPGA SoC)

ST development/eval kits

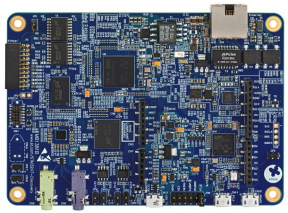
- ST offers three main lines of STM32 boards
  - **EVAL** → "demo in a kit", many onboard devices, all pins used, $$$
    - **Mainly oriented towards MCU SW development & as reference design**
    - **You cannot connect them to generic external devices (no pins)**
  - **Discovery** → some onboard devices, many pins available, $$
    - **Good compromise, depending on the board components**
  - **Nucleo** → almost all pins are broken out to headers, cheap $
    - **The most useful in a lab setting for new HW development**
- NUCLEOs mount TQFP-32 -64 e -144 MCUs by ST
  - All MCU families covered by NUCLEO
  - Some boards mount an Ethernet MAC & socket
  - Integrated ST-Link programmer
  - Arduino and Morph-compatible headers
    - **Pin breakout is somewhat uncomfortable**

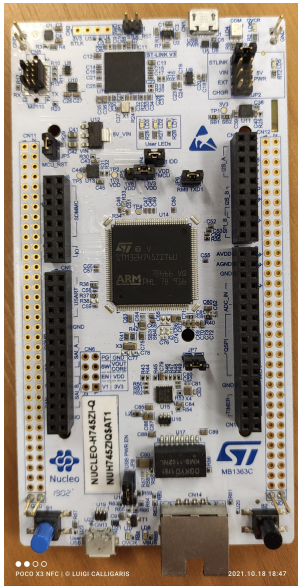


STM32H753-EVAL

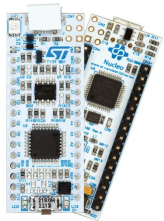


NUCLEO-64


STM32H745-DISCO



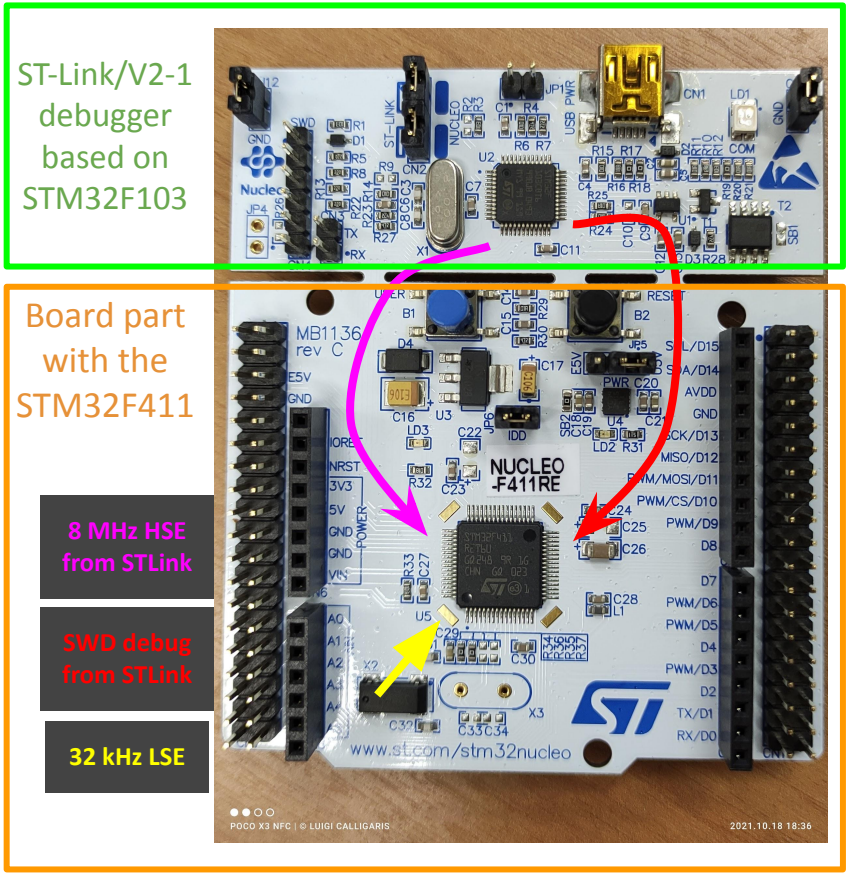
NUCLEO-144


NUCLEO-32

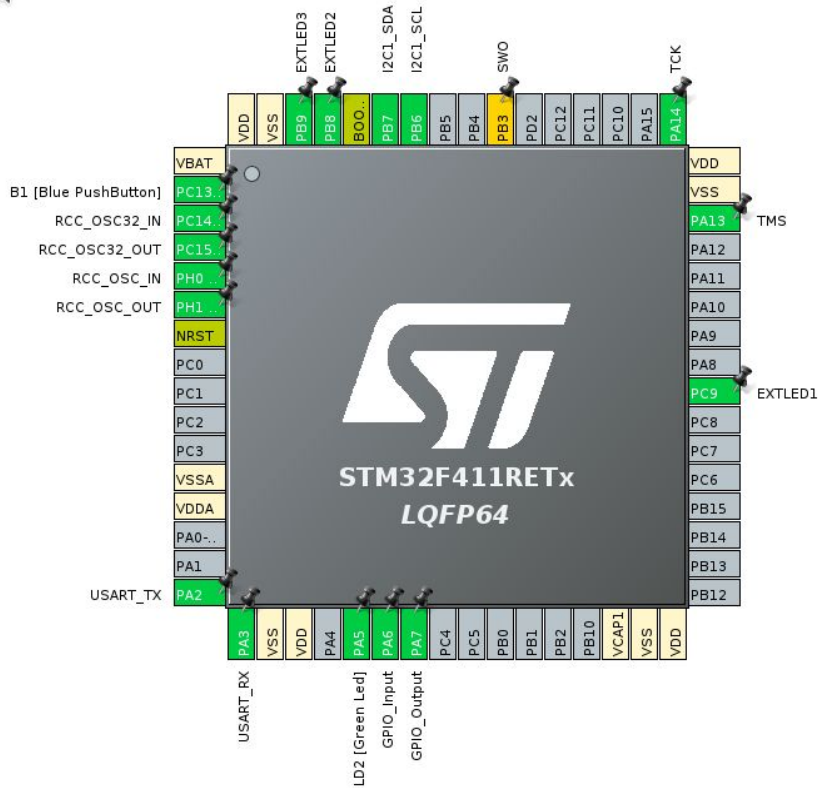
STM32F411-DISCO

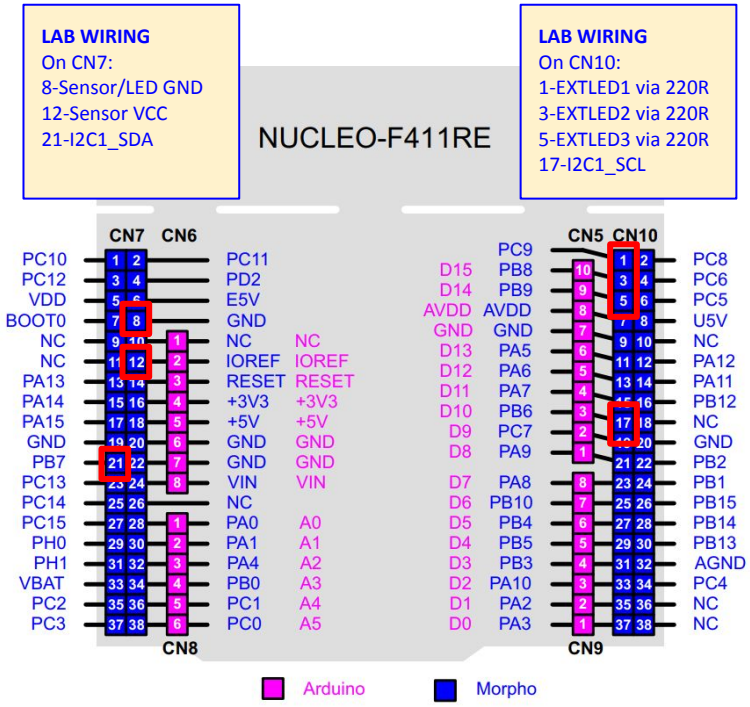
STM32 NUCLEO-F411RE

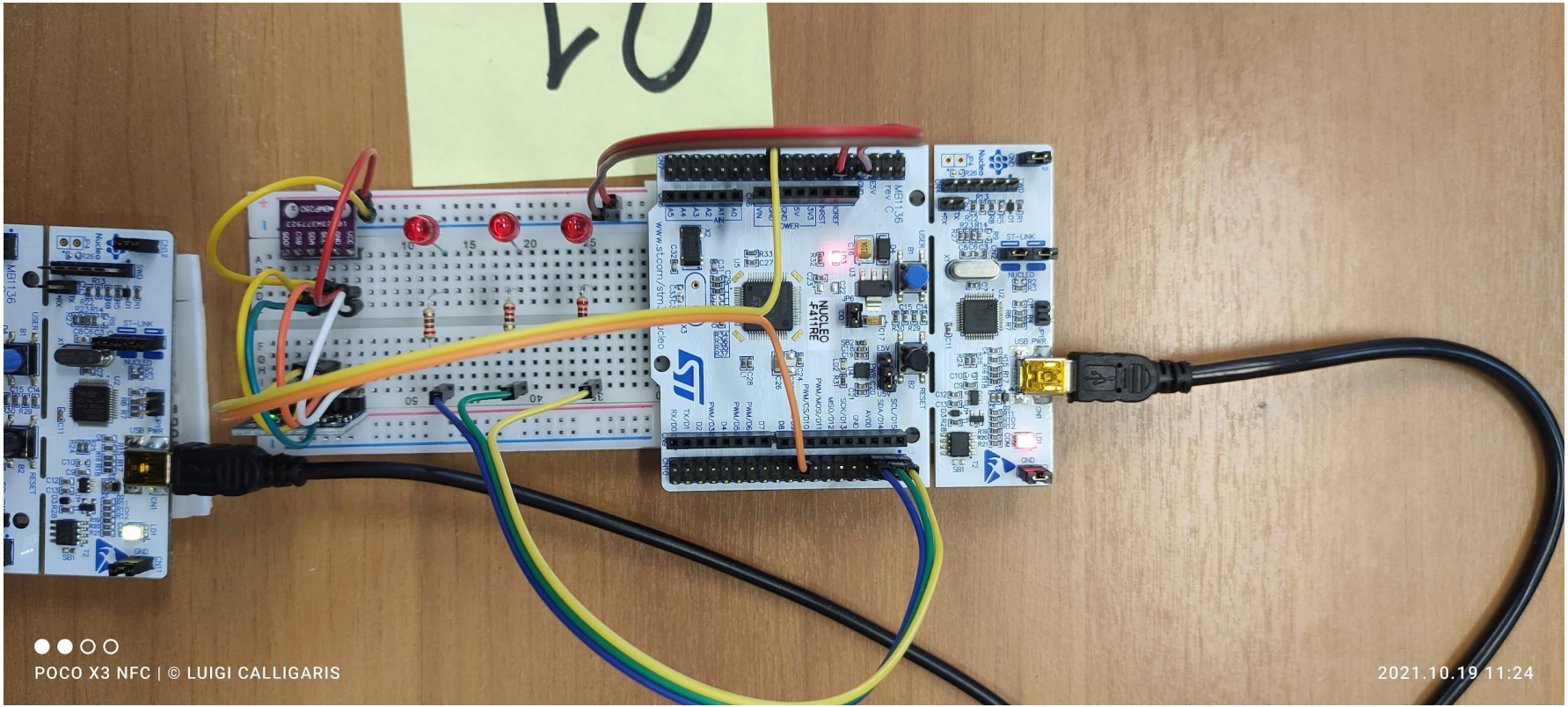
- This is the board we'll use in our labs
- It mounts an STM32F411RE MCU
- The board is composed of two parts
  - STLink/V2-1 debugger (top) based on STM32F103C8
  - The part of the board hosting the STM32F411
- The STLink provides to the main chip
  - Power from USB, dropped to 3.3V via a LDO
  - SWD debug connection to the Coresight DAP
  - An 8 MHz clock generated & controlled by the F103 from its own crystal. Can be used as HSE.



Our wiring for the labs

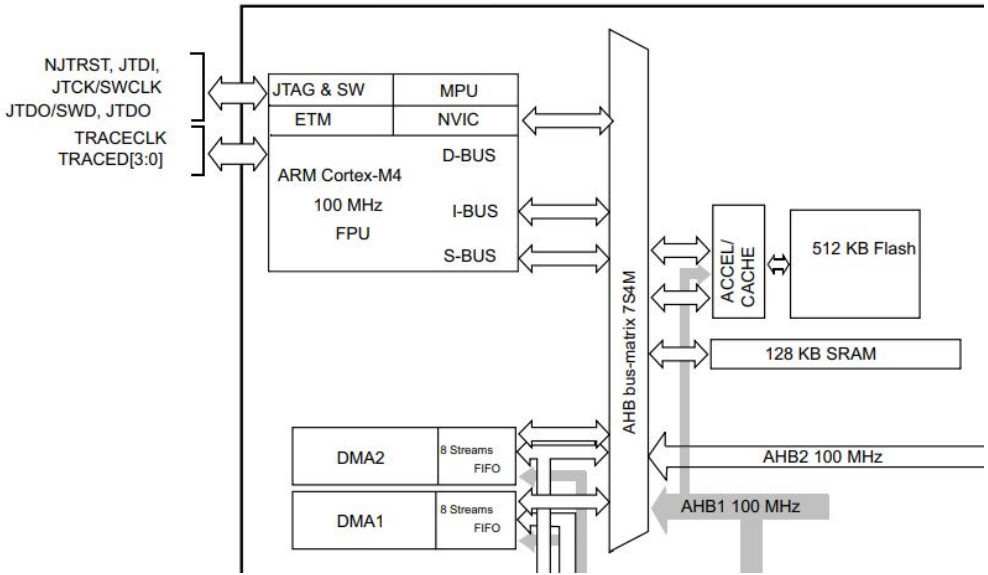





POCO X3 NFC | © LUIGI CALLIGARIS
2021.10.19 11:24

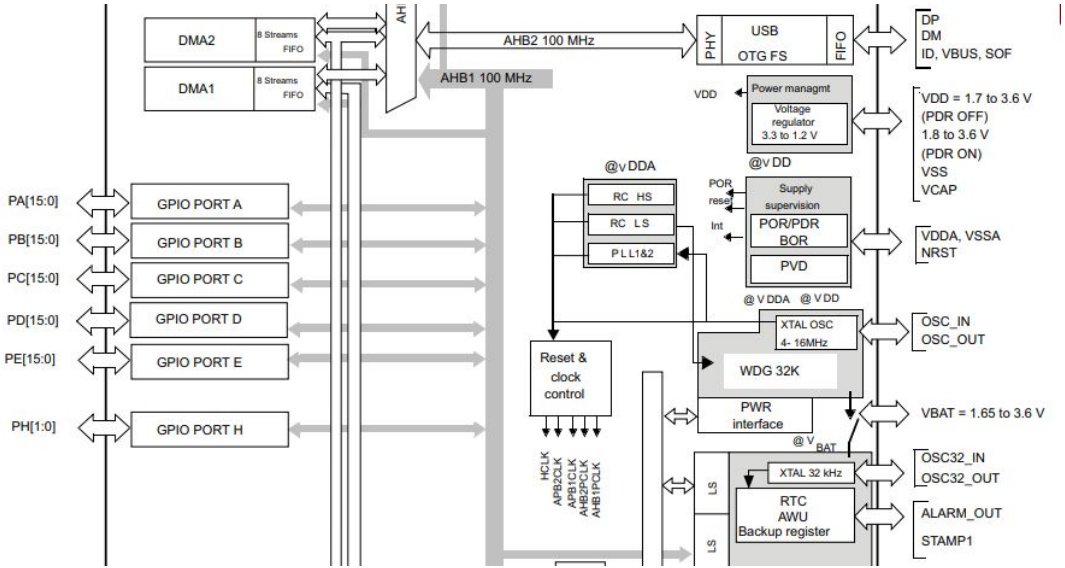
STM32F411RE: Core Part

- Arm Cortex-M4
  - max 100 MHz
  - with Floating Point Unit
- 128 KB SRAM
- 512 KB Flash w/cache
- Coresight DAP
  - JTAG/SWD
  - Embedded Trace Macrocell
- 2 DMA controllers



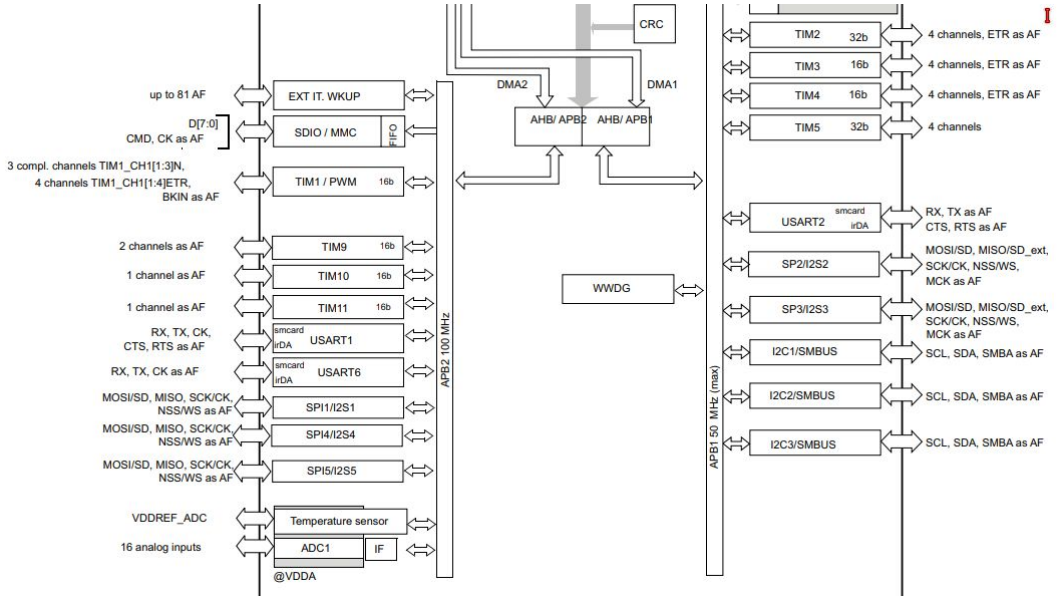
STM32F411RE: Auxiliary & Peripherals 1

- Watchdog timer
  - Protects against prog crashes
- Arm Cortex-M4
  - max 100 MHz
  - with Floating Point Unit
- USB 2.0 Full Speed
- 6 GPIO banks
- Real Time Clock



STM32F411RE: Peripherals 2

- CRC coprocessor
- Another watchdog
- 8 Timers
  - TIM1 advanced
  - TIM2-5 general purpose 4ch
  - TIM9 general purpose 2ch
  - TIM10-11 simple 1ch
- MMC SD car controller
- 3 USARTs
- 5 SPI / I2S(for audio)
- ADC block w/temp sensor



API access to the peripherals: GPIO

For simplicity, configuration is recommended via the IDE configurator (IOC editor), see Lab 2

```
GPIO_PinState     HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void              HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
void              HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void              HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin); // Called if GPIO interrupt enabled AND if it has fired.
```

The callback function is defined by default with the __weak attribute.
This allows you to override the function implementation with your own.
Just reimplement somewhere in your code a function with the same signature.

Example:

```
// in main.c
HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == 5)
  {
    char msg[] = "Pin 5 changed state! \r\n"
    HAL_UART_Transmit_IT(&huart2, msg, sizeof(msg));
  }
}
```

```
/**
  * @brief  This function handles EXTI interrupt request.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}

/**
  * @brief  EXTI line detection callbacks.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

API access to the peripherals: UART

```
For simplicity, configuration is recommended via the IDE configurator (IOC editor), see Lab 2
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout); // polling mode
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);  // polling mode
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size); // interrupt-driven
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);  // interrupt-driven
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size); // with DMA
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);  // with DMA

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint16_t *RxLen, uint32_t Timeout);
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_UART_Abort(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortTransmit(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortReceive(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_Abort_IT(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortTransmit_IT(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortReceive_IT(UART_HandleTypeDef *huart);

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);
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortTransmitCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortReceiveCpltCallback(UART_HandleTypeDef *huart);

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size);
```

API access to the peripherals: I2C

For simplicity, configuration is recommended via the IDE configurator (IOC editor)

```
// POLLING MODE
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout);

// INTERRUPT MODE
HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Write_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Read_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_EnableListen_IT(I2C_HandleTypeDef *hi2c);
HAL_StatusTypeDef HAL_I2C_DisableListen_IT(I2C_HandleTypeDef *hi2c);
HAL_StatusTypeDef HAL_I2C_Master_Abort_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress);
```

API access to the peripherals: I2C

```
// DMA MODE
HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);

// INTERRUPT CALLLBACKS (INTERRUPT AND DMA MODE)
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_AddrCallback(I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode);
void HAL_I2C_ListenCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_MemTxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_MemRxCpltCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_ErrorCallback(I2C_HandleTypeDef *hi2c);
void HAL_I2C_AbortCpltCallback(I2C_HandleTypeDef *hi2c);
```

A focus on the I2C _MEM functions

- It's very common to use this protocol for I2C reg device access:

MASTER DEVICE SLAVE DEVICE

```
Reg write example          |*|10010000|*|01000110|*|00000110|*|
                            S  I2C    W A  Reg     A Reg      S
                            t  Addr   r C  Addr    C Value    t
                            a         i K          K To Write o
                            r         t                       p
                            t         e
```

That is, you first write the reg address, then either

- the data you want to write in case of write into the device register
- you start an I2C read to read the device register content

```
HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

API access to the peripherals: Timers (many functions!)

For simplicity, configuration is recommended via the IDE configurator (IOC editor)

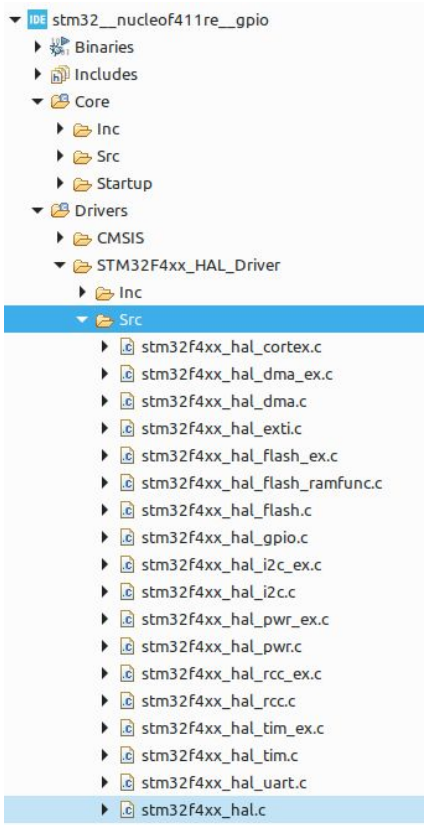
```
// BASE COUNTER MANAGEMENT
HAL_StatusTypeDef HAL_TIM_Base_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_DeInit(TIM_HandleTypeDef *htim);
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef *htim);
void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef *htim);
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_Stop(TIM_HandleTypeDef *htim);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim);
/* Non-Blocking mode: DMA */
HAL_StatusTypeDef HAL_TIM_Base_Start_DMA(TIM_HandleTypeDef *htim,
  uint32_t *pData, uint16_t Length);
HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA(TIM_HandleTypeDef *htim);
// OUTPUT COMPARE FUNCTION
HAL_StatusTypeDef HAL_TIM_OC_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_OC_DeInit(TIM_HandleTypeDef *htim);
void HAL_TIM_OC_MspInit(TIM_HandleTypeDef *htim);
void HAL_TIM_OC_MspDeInit(TIM_HandleTypeDef *htim);
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_OC_Start(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_OC_Stop(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_OC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_OC_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: DMA */
HAL_StatusTypeDef HAL_TIM_OC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel,
  uint32_t *pData, uint16_t Length);
HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel);
```

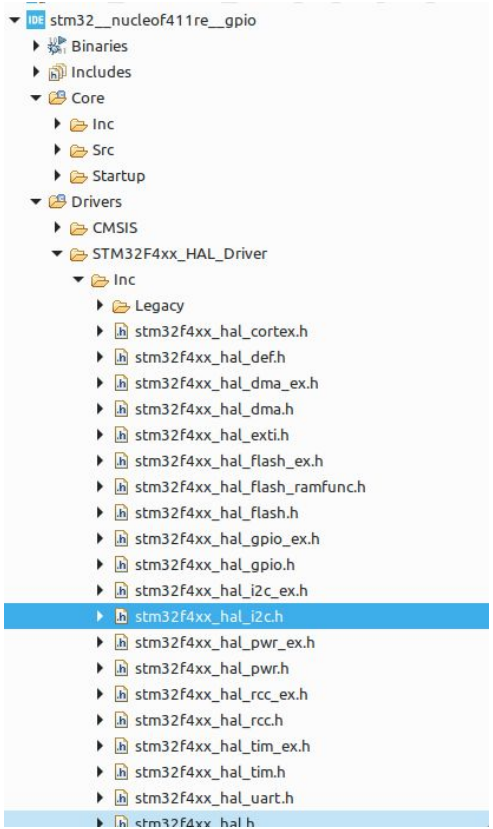
```
// PWM FUNCTION
HAL_StatusTypeDef HAL_TIM_PWM_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_PWM_DeInit(TIM_HandleTypeDef *htim);
void HAL_TIM_PWM_MspInit(TIM_HandleTypeDef *htim);
void HAL_TIM_PWM_MspDeInit(TIM_HandleTypeDef *htim);
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_PWM_Stop(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_PWM_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: DMA */
HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel,
uint32_t *pData, uint16_t Length);
HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel);
// INPUT CAPTURE
HAL_StatusTypeDef HAL_TIM_IC_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_IC_DeInit(TIM_HandleTypeDef *htim);
void HAL_TIM_IC_MspInit(TIM_HandleTypeDef *htim);
void HAL_TIM_IC_MspDeInit(TIM_HandleTypeDef *htim);
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_IC_Start(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: DMA */
HAL_StatusTypeDef HAL_TIM_IC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel,
uint32_t *pData, uint16_t Length);
HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel);
```

Where to find more info in your project

You can navigate the STM32 HAL code in these directories

You have lots of comments which can help you understand which function to use





School home

Questions? :)