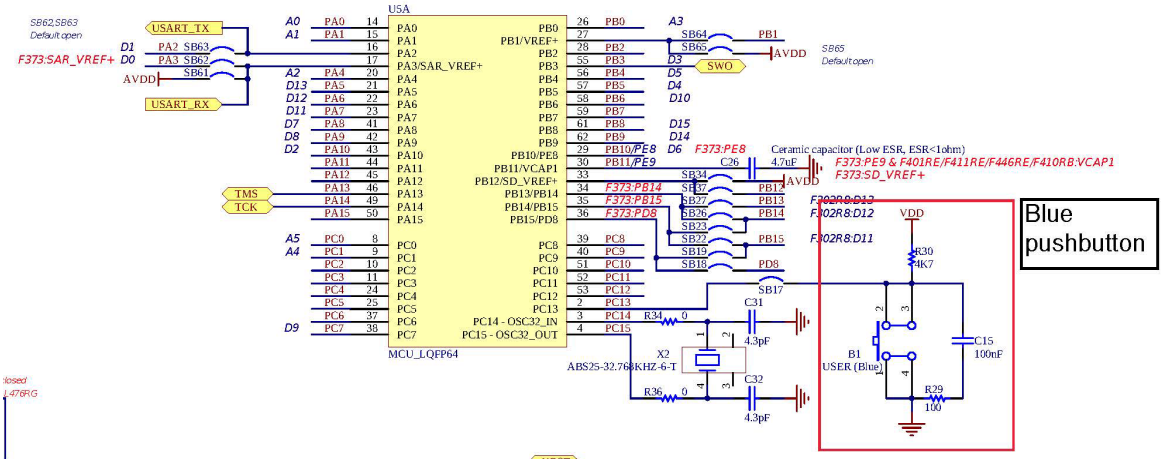
STM32 – Projects

1 M04 | Pushbutton to trigger LED



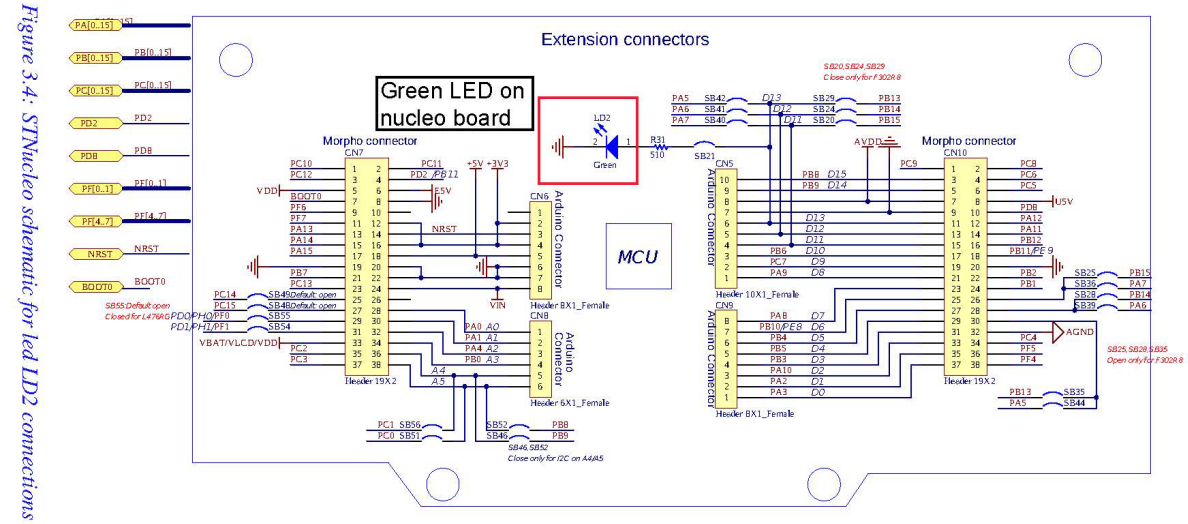


Figure 3.4: STNucleo schematic for led LD2 connections

1A: pushbutton – polling

Objective of this project is to switch on the green LED on NUCLEO board (LD2), every time the blue pushbutton is pressed. A polling operation will be used to monitor the state of the pushbutton.

CUBE:

- Set the button as GPIO input (PC13)
- Set LED as output (PA5)

Keil:

- Write a simple program in the while(1) to check the state of the button and light up the LED accordingly.

1B: Pushbutton -interrupt

Objective of this project is to switch on an LED every time the blue pushbutton is pressed, and to switch it off when the pushbutton is released. The LED input will be used in interrupt mode.

CUBE:

- Set the button as GPIO interrupt (GPIO_EXTI_13) (PC13)
- Set the interrupt priority lower than zero (e.g. 1)
- On GPIO tab, under PC13: GPIO mode = external interrupt mode with rising/falling edge trigger detection
- Set LED as output (PA5)

Keil:

- Write a simple program in the CODE begin 4, exploit the interrupt callback and use the toggle on the rising and falling edge of the button.

---

2 M05 | Blinking LED

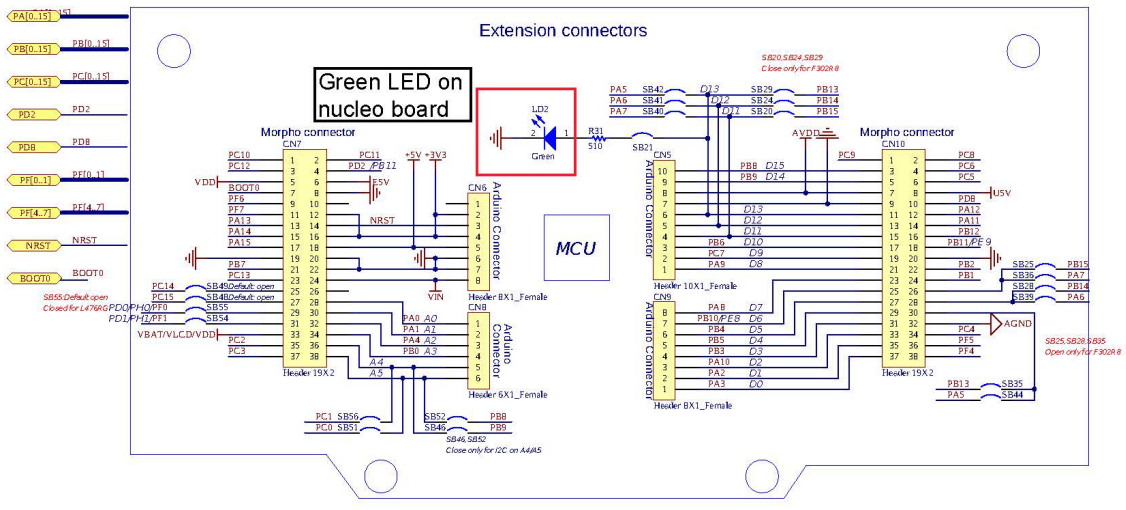


Figure 3.4: STNucleo schematic for led LD2 connections

1C: Blinking LED – PWM

Objective of this project is to blink the NUCLEO board green LED at 1 Hz with 50% DC, using a PWM.

CUBE:

- Connect the LED on PA5 to the timer 2, channel 1
- Sys clock to timer 2

- PWM generation on channel 1

Frequency of 1Hz:

- Prescaler set to 2’000-1
- Counter period 42’000-1
- Pulse: 21’000-1

Keil:

- Start a PWM signal on timer 2, channel 1
```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

1C: Blinking LED V2 – Interrupt (my variation)

This time I will make the LED blink at 1 Hz with 50% DC, using a period elapsed callback.

CUBE:

- Connect the LED on PA5 to the timer 2, channel 1
- Sys clock to timer 2
- Enable TIM2 global interrupt
- NVIC priority 1 to timer 2

Frequency of 1Hz (2Hz) in TIM2:

- Prescaler set to 2’000-1
- Counter period 21’000-1

Keil:

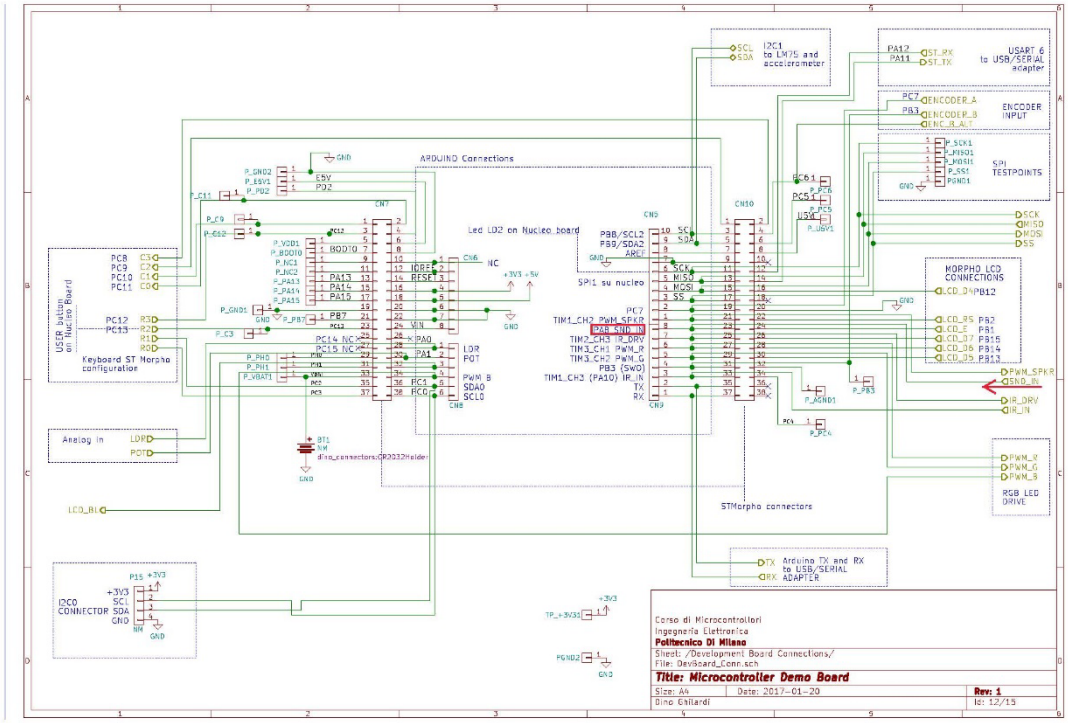
- Start TIM2 in interrupt mode
```
HAL_TIM_Base_Start_IT(&htim2);
```
- Toggle the LED every time a callback is generated by TIM2
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
      if(htim->Instance == TIM2)
            HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

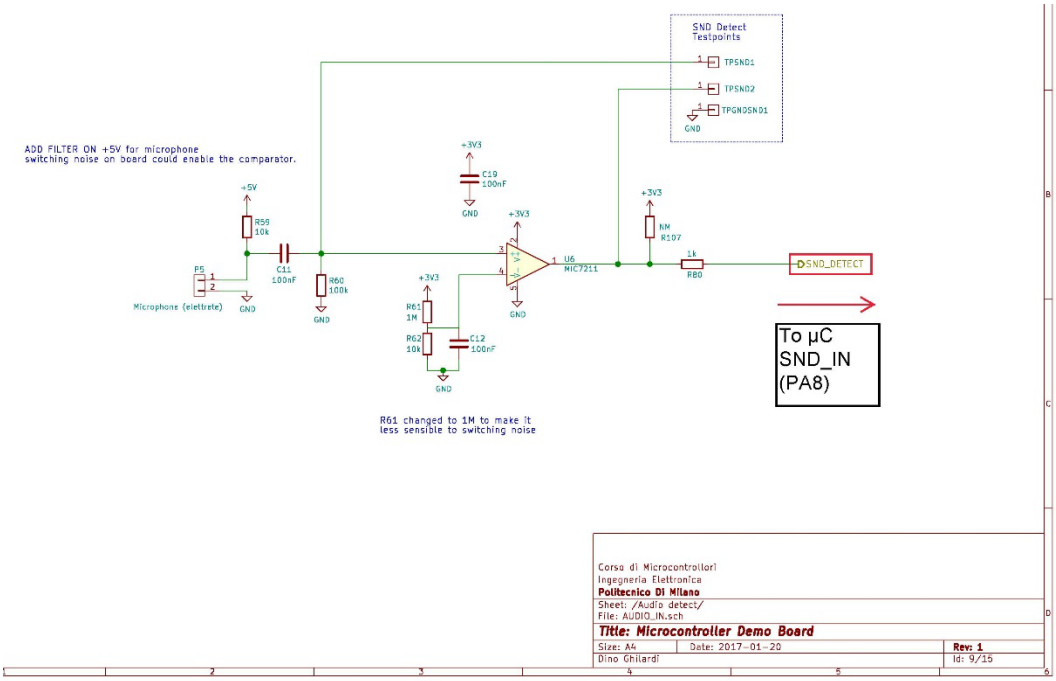
---

3 HW4 | Playing tones

2A: Snap fingers → toggle LED

Modify the status (switch on / off) of the NUCLEO green LED, every time you snap your fingers. (Use the pin connected to the microphone as an External Interrupt).





CUBE:

- Connect microphone to PA8 as external interrupt (GPIO_EXTI8)
- Connect built-in LED to PA5 as output (GPIO_Output)
- TIM1 as system timer
- Enable interrupt for lines 9:5
- Set interrupt priority to 5

Keil:

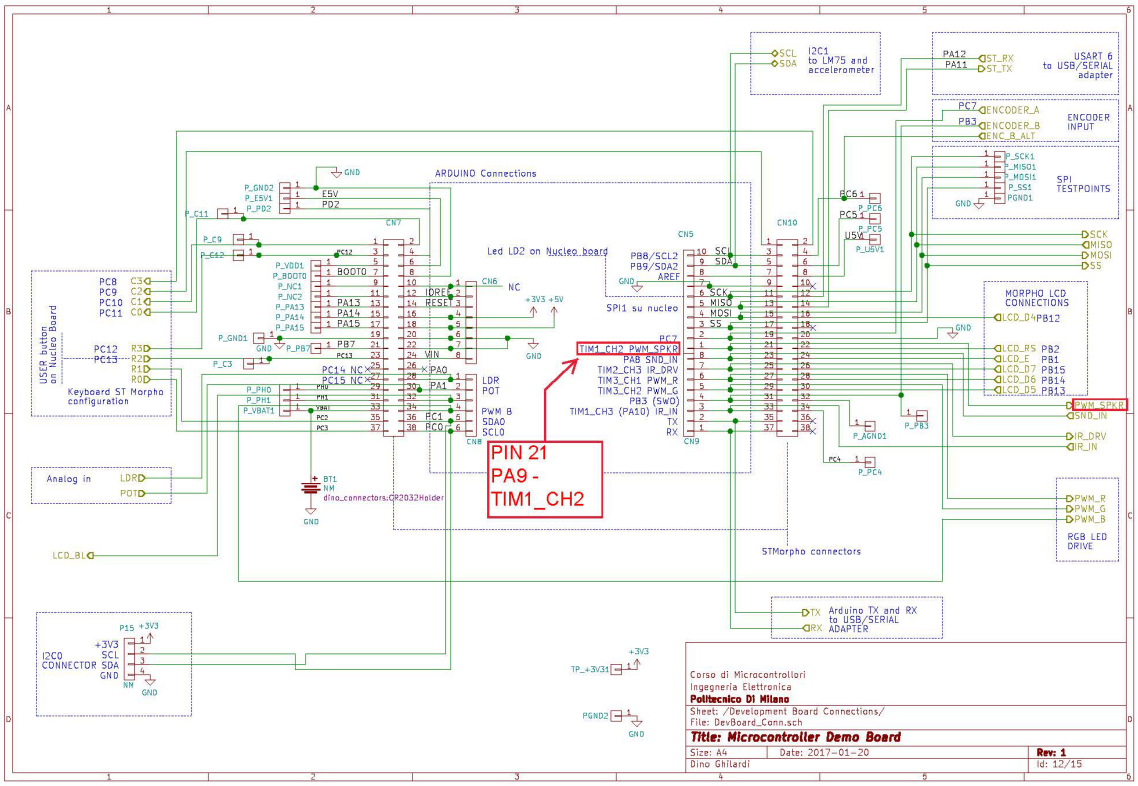
In USER CODE BEGIN 4

- Set a callback for external interrupts
- Check that it was triggered by PIN8
- Toggle PIN5 (LED)
- Wait half a second to make sure that the sound stopped
- Clear interrupt flag

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	switch (GPIO_Pin)
	{
		case GPIO_PIN_8:
			HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
			HAL_Delay(500);
			__HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_Pin);
		break;
	}
}
```

2B: Press button → Play pure tone

Play a single note with the speaker every time you press the NUCLEO blue button. (Provide a PWM square wave to the speaker pin, with the frequency of the note you want to play and DC 50%; stop the PWM after a delay equal to the note duration).



CUBE:

Connections:

- Connect speaker, set PA9 to TIM1_CH2
- Connect blue button as external interrupt, set PC13 to GPIO_EXTI13
- (OPTIONAL) Connect built-in LED, set PA5 to LD2

System:

- Enable interrupt for button
- Set priority to 5
- Sys tick as system timer

Timer 1 controls speaker frequency (C4 – central C, 262Hz)

84MHz/**100** = 840KHz

840KHz/**3206** = 262 Hz

- Clock source: internal clock
- Channel 2: PWM generation CH2
- Prescaler: **100**-1
- Counter period: **3206**-1
- Pulse 1603-1

Keil:

- Callback in USER CODE BEGIN 4
- Check that it was called by PIN13 (blue button)
- (Optional) Turn on LED
- Start PWM on speaker
- Wait 1 second through a **DELAY** function
- Stop PWM on speaker
- (Optional) Turn off LED
- Clear flag

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
	switch(GPIO_Pin){
		case GPIO_PIN_13:
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
			HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
			HAL_Delay(1000);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
			HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
			__HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_Pin);
	}
}
```

2B V2: Press button → Play pure tone

Previous exercise modified such that no delay function is used.

CUBE:

Same steps (at least it seems like it's the same)

Keil:

- Create a counter variable to keep track of the time flow
  `int counter = 0;`
- Start TIM in interrupt mode
  `HAL_TIM_Base_Start_IT(&htim1);`

Interrput:

- When the button is pressed start the PWM and reset the counter to zero.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
                        // button pressed
{
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
                              // start tone
```

```
    counter = 0;

                // initialize counter
}
```

- Count the cycles and stop the counter after a second (equivalent to 262 cycles)

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
    // 1 cycle elapsed
{
    if(htim->Instance == TIM1)    // check call by Timer 1
    {
          counter ++;             // increase counter
          if (counter == 262){    // a second passed (262 cycles)
                HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
                // stop tone
                HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
                // toggle LED2 for visual reference
          }
    }
}
```

3A: USART transmit

Cube:

- Nothing needs to be connected to the CPU
- Use SysTick as system timer
- Connectivity > USART2 > Mode: asynchronous
- 9600 bps (baud rate)

Keil:

- Add

```
#include "stdio.h"
#include "string.h"
```

- Prepare the variables to contain a person

```
struct person
{
      char name[10];
      int year;
};
struct person mestesso = {"alessio", 1996};
```

- Add support variables for the USART protocol

```
char str[30];
int len;
```

- In the main mash together the string we want to send and calculate its length (we usually calculate the string length directly in the transmit function)
  ```
  sprintf(str, "%s\t%04u\r\n", mestesso.name, mestesso.year);
  len = strlen(str);
  ```
- In the while(1) transmit the string every second
  HAL_UART_Transmit(&huart2, (uint8_t *) str, len, 100);
  HAL_Delay(1000);

2C: Play a Song

- Button
  See above
- Speaker
  See above
- LED
  See above

CUBE:

Connections:

- Connect button as external interrupt (PC13, GPIO_EXTI_13)
- Connect LED as output (PA5, LD2)
- Connect speaker to PWM (PA9, TIM1_CH2)

- SysTick as system timer
- Enable interrupt from button (priority 5)

Set timer 1 to count 0.5 milliseconds

- Internal clock
- PWM generation CH2
- Prescaler 84-1
- ARR: 480-1
- Pulse: 240-1
- Enable interrupts

Set timer 2 to count 1 millisecond (internally)

- Clock source: internal clock
- Prescaler: 2000
- ARR: 42-1
- Enable interrupts

Keil:

- Define milliseconds, seconds and BPM
- Define all the notes on a piano (I decided to define them by the period)
- Array variables to describe the song
    - Notes
    - Duration
- More variables to manage the song playback (“beat” section)
- Setup:
    - Function header
    - Define beat as (duration of each beat, battuta, in ms): `beat = MIN*MS/BPM;`

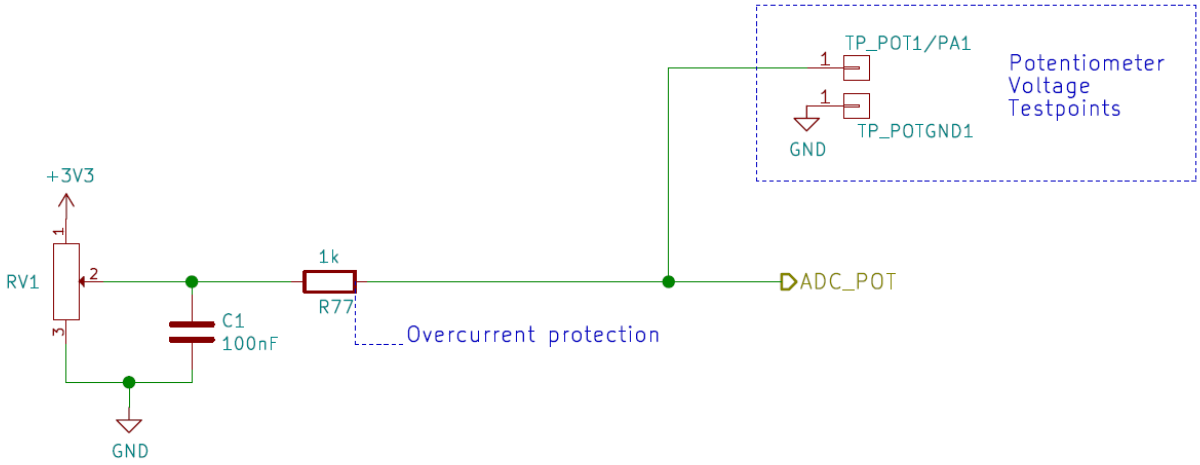
4A: ADC started by software

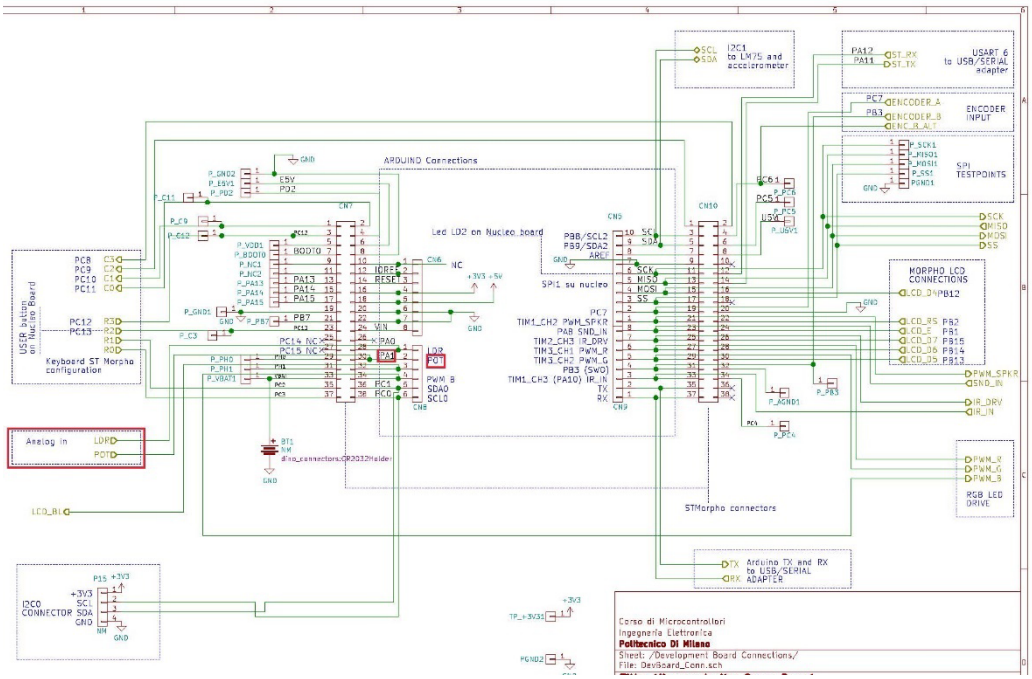
Objective of the project is to acquire the voltage of the potentiometer on the POLIMI board, starting the conversion by software and then send the value to PC on a remote terminal.

4A-I: ADC single acquisition polling

Objective of this project is to acquire the voltage of the potentiometer every 1 second and send this value to a remote terminal. The ADC will be used in polling mode.

POTENTIOMETER:





Cube:

- Connect potentiometer to ADC (PA1 as ADC1_IN1)
- Set USART baud rate to 9600
- ADC
  - Prescaler divided by 4
  - EOC flag at the end of conversion
  - External trigger conversion source: Conversion launched by software
  - Rank: 480 cycles (slowest mode)

Keil:

Important code in the while(1)

- Start ADC single conversion: (this one is also in the setup)
  `HAL_ADC_Start(&hadc1);`
- Check if the value is ready:
  `if(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK)`
- Get the value:
  `value = HAL_ADC_GetValue(&hadc1);`
- Convert it to voltage
  `conv = value/4095.000*3.30;`
- Send it via USART
  (standard procedure)
- Delay 1 second
  `HAL_Delay(1000);`

4A-II: ADC single acquisition interrupt

Objective of this project is to acquire the voltage of the potentiometer and send this value to a remote terminal every 1 s. The ADC will be used in interrupt mode.

Cube:

- Connect potentiometer to ADC (PA1 as ADC1_IN1)
- Set USART baud rate to 9600
- ADC
    - Prescaler divided by 4
    - Enable continuous conversion mode
    - EOC flag at the end of all conversion
    - External trigger conversion source: Conversion launched by software
    - Rank: 480 cycles (slowest mode)

Keil:

Code in the main():

- Start ADC in interrupt mode:
  `HAL_ADC_Start_IT(&hadc1);`

Important code in the USER CODE BEGIN 4:

- The ADC calls an interrupt when a value is ready:
  `HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)`
- Get the value:
  `value = HAL_ADC_GetValue(&hadc1);`
- Convert it to voltage
  `conv = value/4095.000*3.30;`
- Send it via USART
  (standard procedure)
- Delay 1 second the conversion of another value
  `HAL_Delay(1000);`

3B: ADC triggered by TIM

Objective of the project is to acquire the potentiometer voltage using a timer to trigger a conversion at a regular conversion rate of 1 Hz.

Cube:

- Connect potentiometer to ADC (PA1 as ADC1_IN1)
- Set USART baud rate to 9600
- ADC
    - Prescaler divided by 4
    - External trigger conversion source: Timer 3 Trigger out event

    - Trigger detection on rising edge
    - Rank: 3 cycles (fastest mode)
  - Timer 3 at 1Hz
    - Clock source: internal clock
    - Prescaler: 2000-1
    - ARR: 42000-1
    - Trigger event selection: Update event

Keil:

Code in the main():

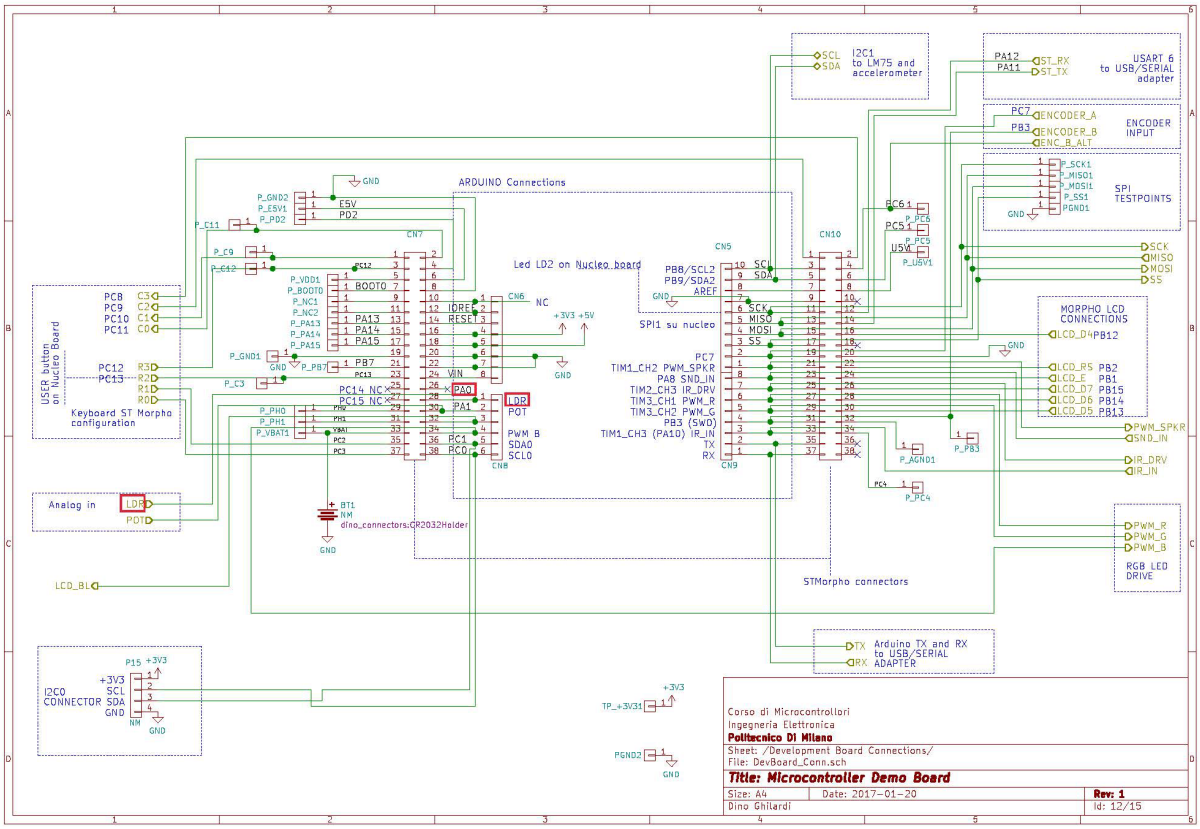
- Start ADC in interrupt mode:
  `HAL_ADC_Start_IT(&hadc1);`
- Start timer
  `HAL_TIM_Base_Start(&htim3);`

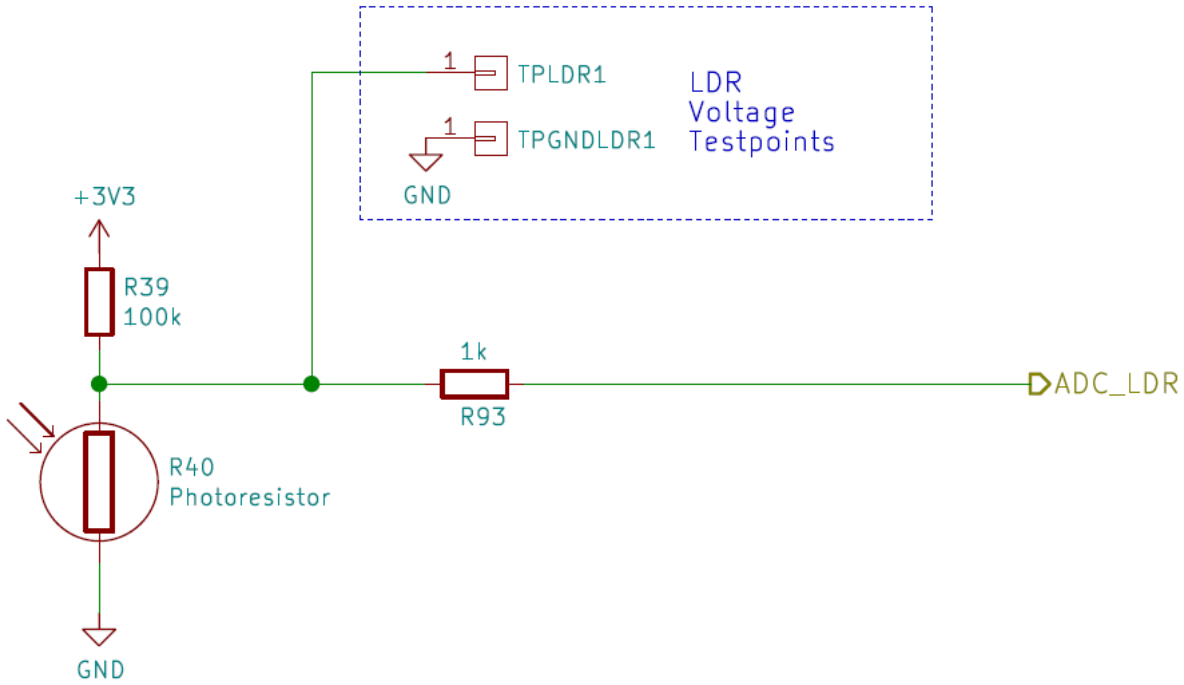
Important code in the USER CODE BEGIN 4: (exactly the same as before but without the delay(1000)

- The ADC calls an interrupt when a value is ready:
  `HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)`
- Get the value:
  `value = HAL_ADC_GetValue(&hadc1);`
- Convert it to voltage
  `conv = value/4095.000*3.30;`
- Send it via USART
  (standard procedure)

4C: Light Dependent Resistor DMA

Objective of the project is to acquire the signal from the LDR (by Sunrom technology) on the POLIMI board. Convert one value every 1ms and store 1,000 values using the DMA. Every 1s, display on a remote terminal the average light power (expressed in lux) of the previous second.





CUBE:

- Connect LDR to ADC (PA0 as ADC1_IN0)

- Set USART baud rate to 9600
- ADC
  - Prescaler divided by 4
  - DMA continuous request: enabled
  - EOC flag at the end of all conversions
  - External trigger conversion source: Timer 3 Trigger out event
  - Trigger detection on rising edge
  - Rank: 3 cycles (fastest mode)
    DMA
  - Mode: circular
  - Threshold: Half full
  - Use FIFO
  - Checkmark on memory
  - Data size: word
  - Burst size: single
- Timer 3 at 1000Hz
  - Clock source: internal clock
  - Prescaler: 2000-1
  - ARR: 42-1
  - Trigger event selection: Update event
  - TIM3 global interrupt: priority 1

Keil:

Variables:

- Create and array of *unit32_t* of length 2000
  `uint32_t value[2*LEN];`

Code in the main():

- Start timer
  `HAL_TIM_Base_Start(&htim3);`
- Start ADC in DMA mode
  `HAL_ADC_Start_DMA(&hadc1, (uint32_t *) value, 2*LEN);`

Important code in the USER CODE BEGIN 4:

The ADC calls an interrupt when DMA memory is half-full (0-999 values sampled) or complete (1000-1999 values sampled):
`HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)`

`HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)`

Inside the interrupt I want to average the 1000 acquired values and obtain a light measurement in LUX:

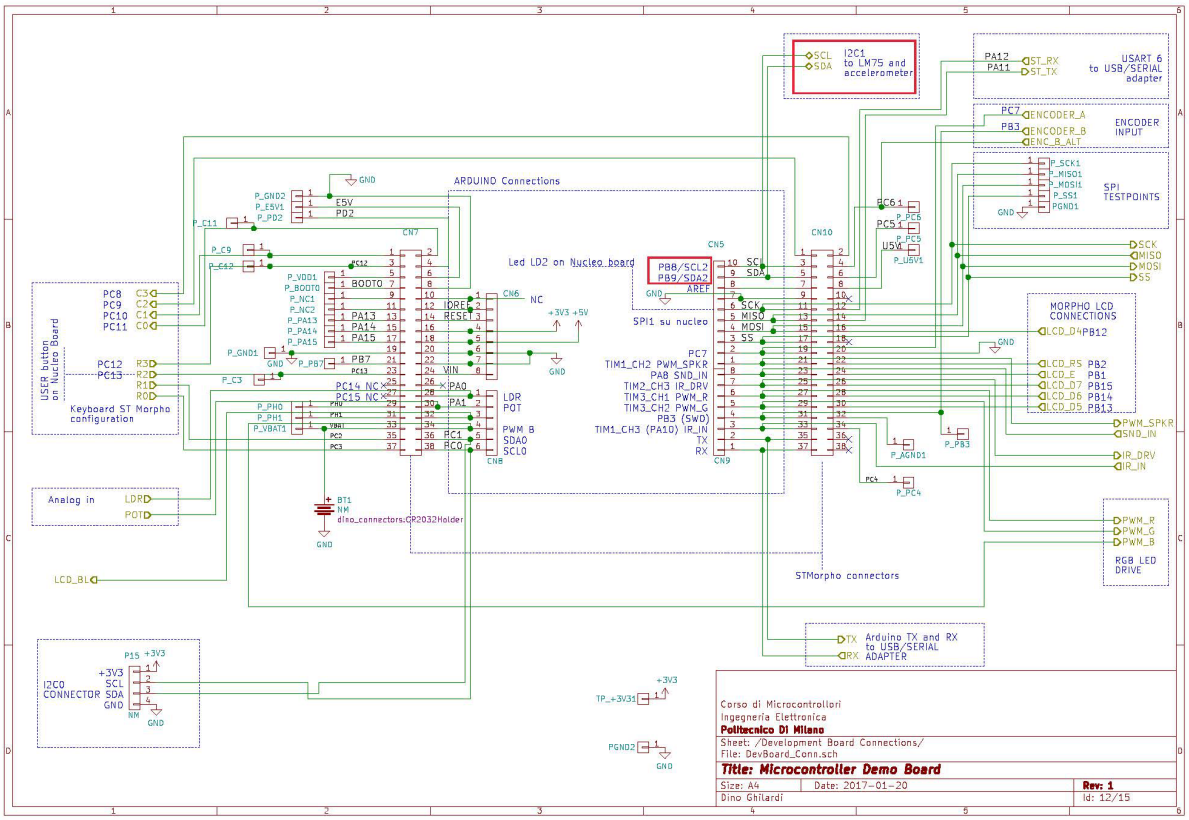
- Sum all the 1000 values and calculate their average:
  `for(i=LEN; i<2*LEN;i++)`

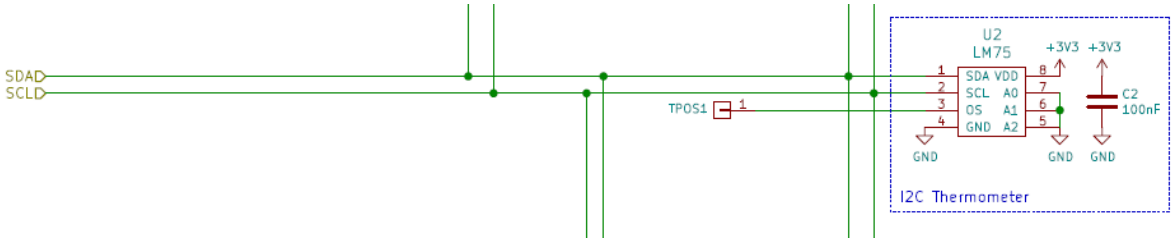
```
                sum+=value[i];
    mean=sum*1.000/LEN;
```

- Convert it to voltage
  `voltage = mean*3.3/4095;`
- Calculate LDR resistance as (see schematics above):
  `resistance = 100*voltage/(3.3-voltage);`
- Calculate LUX as (see graph on the datasheet):
  `lux = pow(100/resistance,4.00/3);`
- Send it via USART
  (standard procedure)

5A: Temp MSB

Read the temperature measured by the LM75 and send it to a remote terminal every 1 second. As a first step we will read only the MSB 8 bit.





CUBE:

I2C1

- I2C: I2C
- Speed: 100 000 (100KHz)
- Primary address length selection: 7 bits
- Connect PB9 to I2C1_SDA
- Connect PB8 to I2C1_SCL

USART

- 9600 bps

Keil:

- Define temperature sensor address:

```
#define LM_75_ADDRESS 0x48
```

0x48 = 0b1001 000

Table 4. Address table

1 = HIGH; 0 = LOW.

MSB						LSB
1	0	0	1	A2	A1	A0

A2, A1, A0 set to zero (ground) as in the previous schematics.

- Define the temperature register address
  `unsigned char temp_reg = 0x00;`

Table 6. Pointer register

B7	B6	B5	B4	B3	B2	B[1:0]
0	0	0	0	0	0	pointer value

Table 7. Pointer value

B1	B0	Selected register
0	0	Temperature register (Temp)
0	1	Configuration register (Conf)
1	0	Hysteresis register (Thyst)
1	1	Overtemperature shutdown register (Tos)

Main code in the while(1)

- Transmit to the temperature sensor the address of its temperature register
  `HAL_I2C_Master_Transmit(&hi2c1, LM_75_ADDRESS<<1, &temp_reg, 1, 100);`
- Receive from the temperature register the temperature value (MSByte only)
  `HAL_I2C_Master_Receive(&hi2c1, LM_75_ADDRESS<<1, &temperature, 1, 100);`
- `Send the value via USART`
- `Wait for 1000ms`

No conversion is needed. The MSByte already contains the temp in Celsius.

5B: Temp FSR

Now we will modify the code to read all 11 bits

CUBE:

Same as previous exercise

Keil:

Variables are mostly unchanged

- Now temperature is an array with 2 cells (2 bytes):
  `unsigned char temperature[2];`
- I also need to convert the temperature, therefore I create a float to contain the result:
  `float final_temp;`

In the while(1)

- Transmit remains unchanged
- Receive now gets back two values:
  `HAL_I2C_Master_Receive(&hi2c1, LM_75_ADDRESS<<1, temperature, 2, 100);`
- NOW I need to convert the temperature:

```
final_temp =      temperature[0]+
                  (temperature[1]>>5)/8.000;
                  // shift the LSByte by 5 to the right (right align)
                  // multiply by LSB (0.0125), or divide by 8
```

- Pack it in a nice string and send it via USART

5C: Temp acquired with interrupt

Modify the code to read the temperature with a timer interrupt.

CUBE:

Leave the USART and SPI unvaried

Setup TIM3 to work at 1Hz:

- Prescaler: 2000-1
- ARR:42000-1
- Enable TIM3 global interrupt

Keil:

- Start TIM3 in interrupt mode
  `HAL_TIM_Base_Start_IT(&htim3);`

  Below USER CODE 4 I can find the generic timer callback

- Add a specific case for TIM3
  `if (htim->Instance == TIM3)`

Inside this case it's essentially the same. I2C transmit, I2C receive, convert temperature, transmit via USART.

6A – Encoder

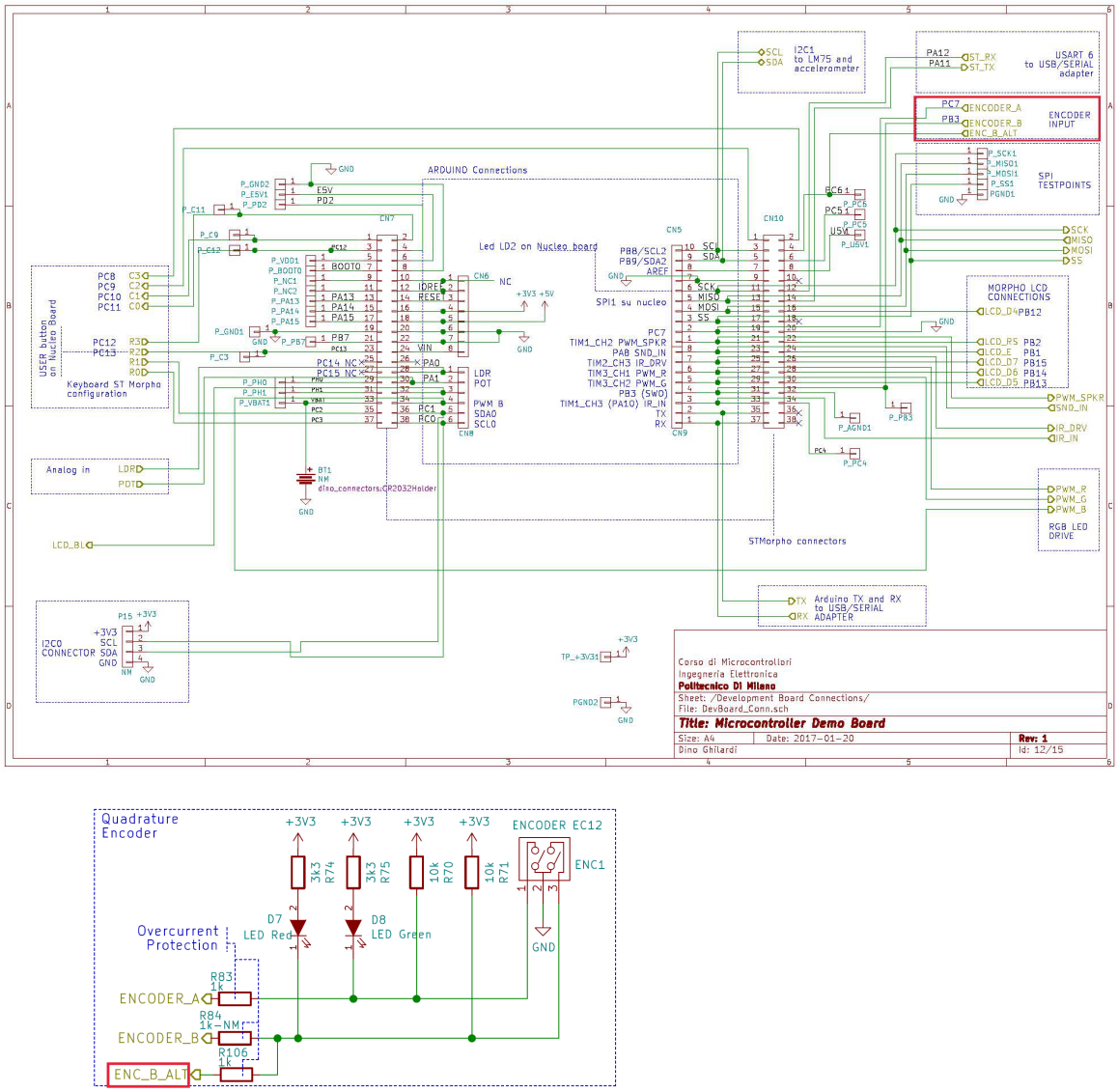


Figure 5.2: Encoder section schematic on PMDB16

<u>CUBE:</u>

Quadrature encoder connections (can be seen on the first schematics):

- Connect PC6 to TIM3_CH1
- Connect PC7 to TIM3_CH2

System

- Use TIM1 as system clock

Keep track of time using TIM2 at 1Hz. Use TIM2 at 1Hz

- Prescaler: 2000-1
- ARR 42000-1

Keep track of the encoder waves using TIM3:

- No clock source
- Channel 3: output compare, no output
- Combined channels: encoder mode
- ARR: 63335 $(2^{16} - 1)$
- Output compare no output CH3
    - Mode: active level on match
    - Pulse: 65535
- Encoder
    - CH1
        - Polarity: falling edge
        - Filter: 10
    - CH2
        - Polarity: rising edge
        - Filter: 10
- Enable TIM3 global interrupt

USART: 9600bps

Keil:

• Variables:

```
long value = 0, value_old = 0; // keeps track of the counter in TIM3
float freq; // frequency of the rotary encoder
int overflow = 0; // in case the counter in TIM3 overflows this gets raised
char string[16]; // used to send the frequency via uart
```

• Setup in the main()

```
HAL_TIM_Base_Start_IT(&htim2);                  // start timer 2 (1Hz)
HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL); // start TIM3 as encoder
HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_3);     // enable interrupt in case
of overflow
```

• In the USER CODE 4 setup a simple function to increase the variable *overflow* in case of overflow of TIM3:

```
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim){
      overflow++; // variable incremented in case of overflow
}
```

Within the already written function *period elapsed callback* add the case for TIM2 (1Hz)

First save the previous counter value into *value_old*

```
value_old = value;
```

Get the current counter value

```
value = __HAL_TIM_GET_COUNTER(&htim3);
```

If no overflow occurred just divide the difference of *value* and *value_old* by 24 (24 pulses in one full rotation)

```
freq = (float) (value-value_old)/24;
```

Otherwise check if the overflow was “positive”(value < value_old) or “negative” (value < value_old)

Positive formula: `freq = (float) (value+(htim3.Init.Period-value_old)+1)/24;`

Negative formula: `freq = (float) (-1)*((htim3.Init.Period-value)+value_old+1)/24;`

Don’t forget to reset *overflow*!

Now we are ready to send the frequency to the PC via USART!

7A – Accelerometer

Acquire the x, y, z acceleration and send the 3 values to a remote terminal, about every 2 seconds.

The value of acceleration must be expressed in g, with a precision of 0.01 g and the correct sign.

The full-scale range must be +/-2 g.

The output format of the data must be like the one in the example:

X: + 0.05 g

Y: -0.22 g

Z: + 1.00 g

<u>CUBE:</u>

Standard procedure for I2C peripherals:

I2C1

- I2C: I2C
- Speed: 100 000 (100KHz)
- Primary address length selection: 7 bits
- Connect PB9 to I2C1_SDA
- Connect PB8 to I2C1_SCL

USART

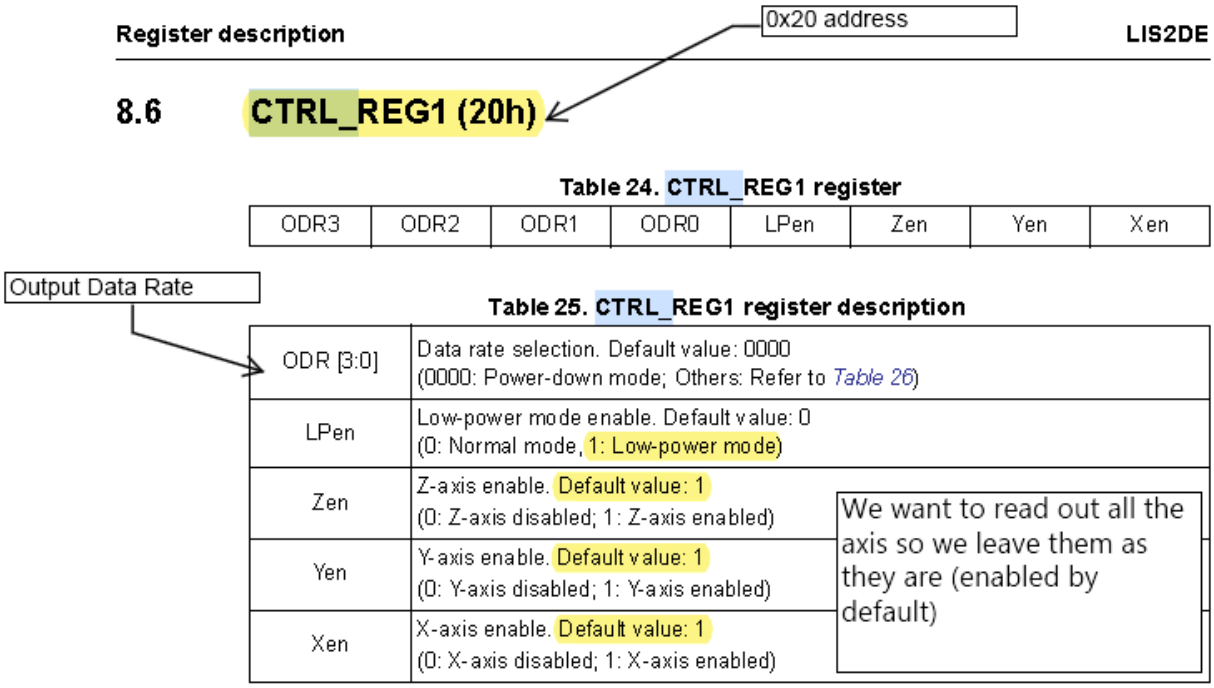
- 9600 bps

Keil:

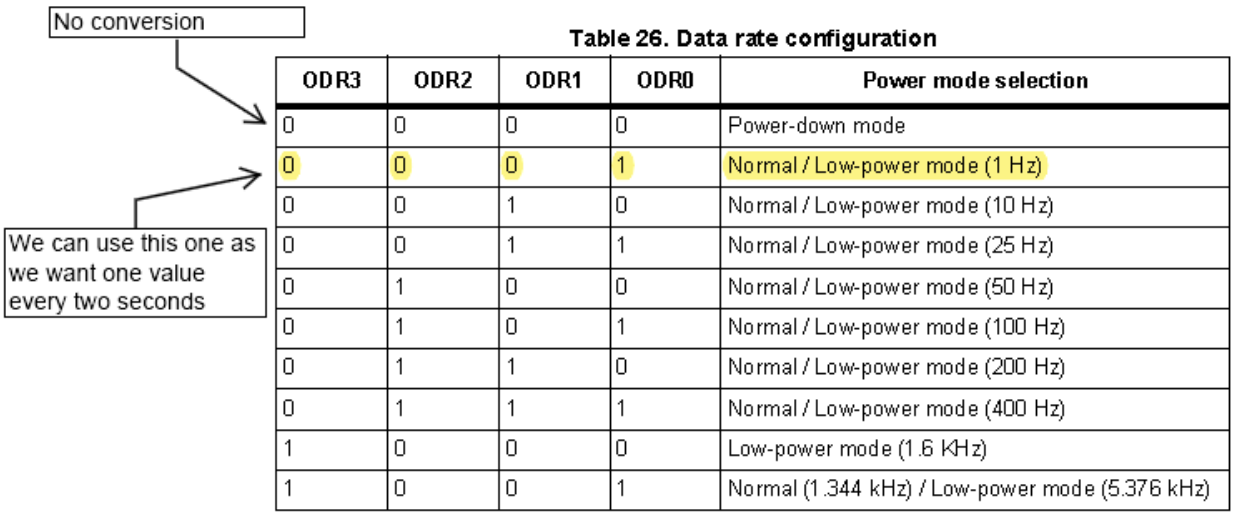
- Define the accelerometer address:
  `#define LIS2DE_ADDRESS 0x28 // 010100 0 (SAD+R/W)`

Variables:

- Setup the accelerometer control register 1 at the address 0x20 (we actually only need to define the output rate @1Hz):
  `uint8_t CTRL_REG_1[] = {0x20, 0b00011111}; // 0001 1111`



ODR [3:0] is used to set power mode and ODR selection. The following table provides all frequencies resulting from a combination of ODR [3:0].



- Create the variables that contain the address of the X, Y and Z axis registers and their values

```
uint8_t reg_x = 0x29;
uint8_t val_x;
uint8_t reg_y = 0x2B;
uint8_t val_y;
uint8_t reg_z = 0x2D;
uint8_t val_z;
```

- Create the variables to contain the converted values in g:

```
float a_X, a_Y, a_Z, A;
```

In the main()

- Actually write the CTRL_REG1

```
HAL_I2C_Master_Transmit(&hi2c1, LIS2DE_ADDRESS<<1, CTRL_REG_1, 2, 100);
```

In the while(1)

- Use the function read_axis
- Convert the acquired value in g
- Send it via USART

In the read_axis function I just condense the I2C transmit and receive

```
void read_axis(uint8_t *adress, uint8_t *value)
{
	HAL_I2C_Master_Transmit(&hi2c1, LIS2DE_ADDRESS<<1, adress, 1, 100);
	HAL_I2C_Master_Receive(&hi2c1, LIS2DE_ADDRESS<<1, value, 1, 100);
}
```