Button Read using HAL libraries

rev1.0 24/03/2020

Read the state of the button on the board and debounce it

PREREQUISITES

Software needed:

- STM32IDE

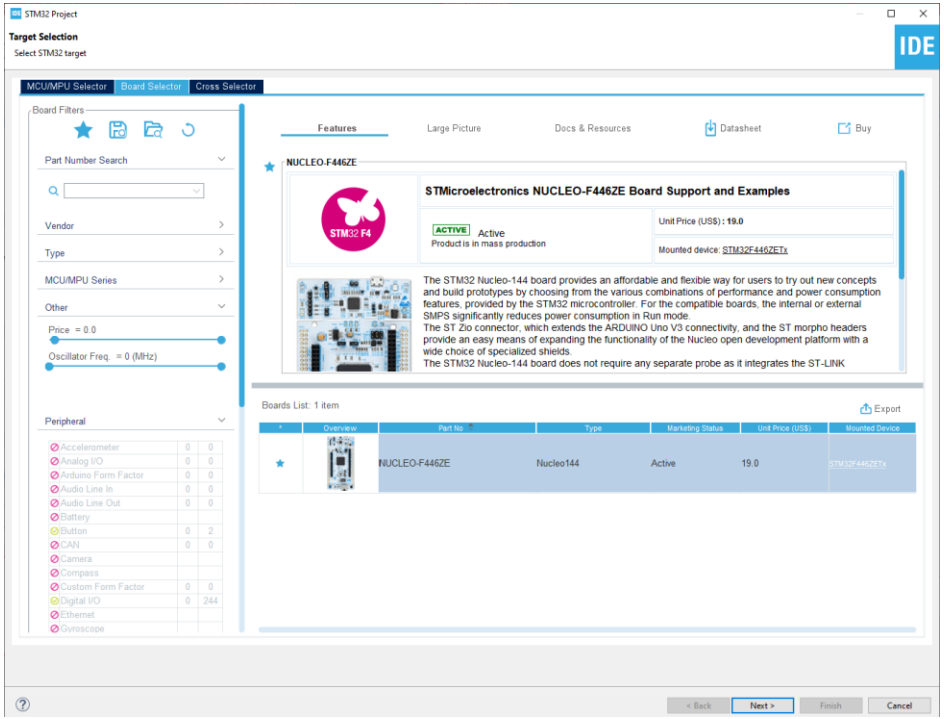
Hardware used in this example:

- **NUCLEO-F446ZE**

Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

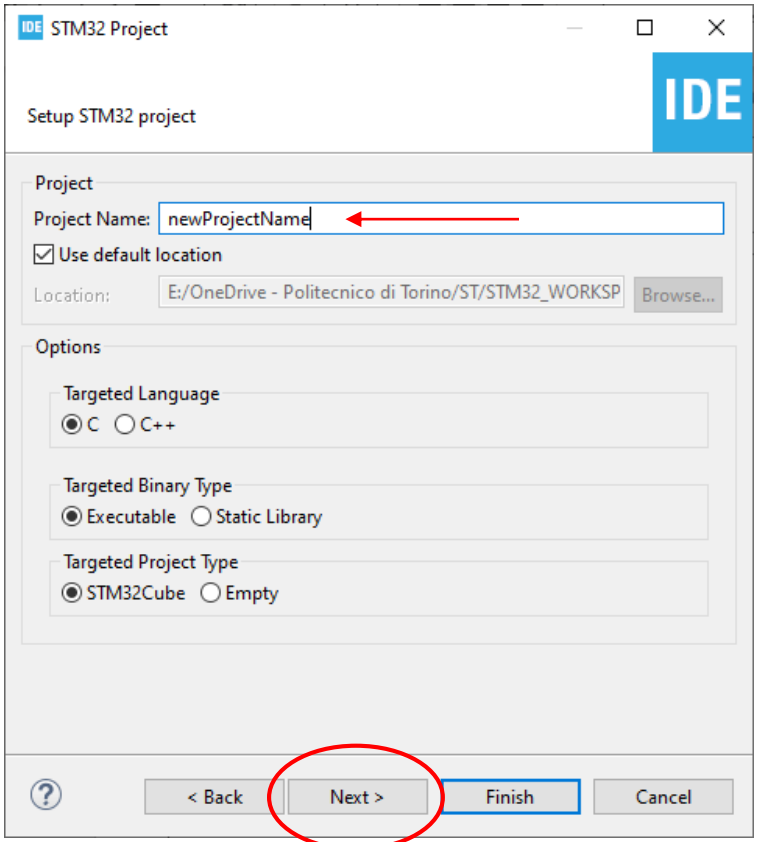
Select your board or your uC and click *next.*



Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



Start a new project

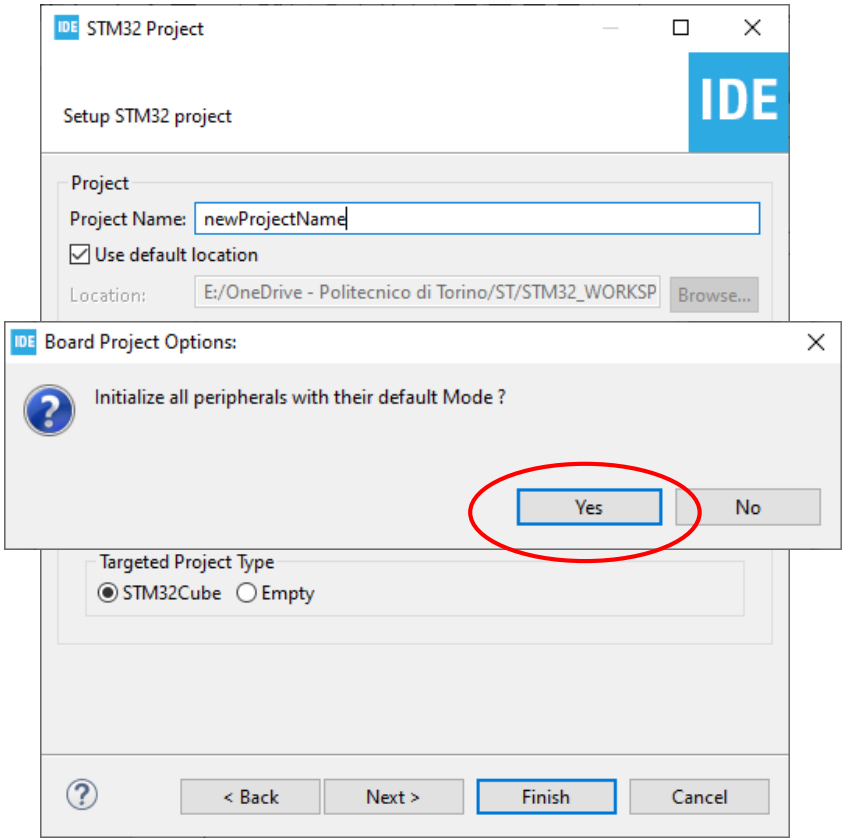
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

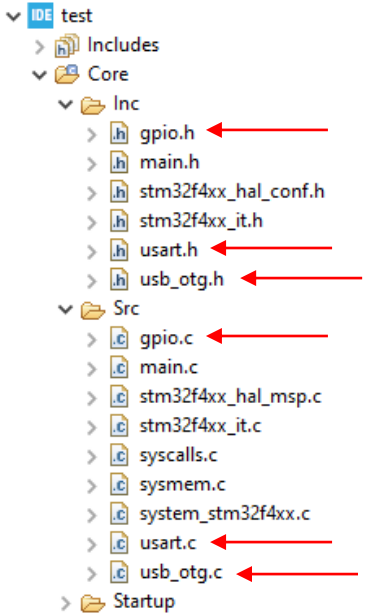
Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

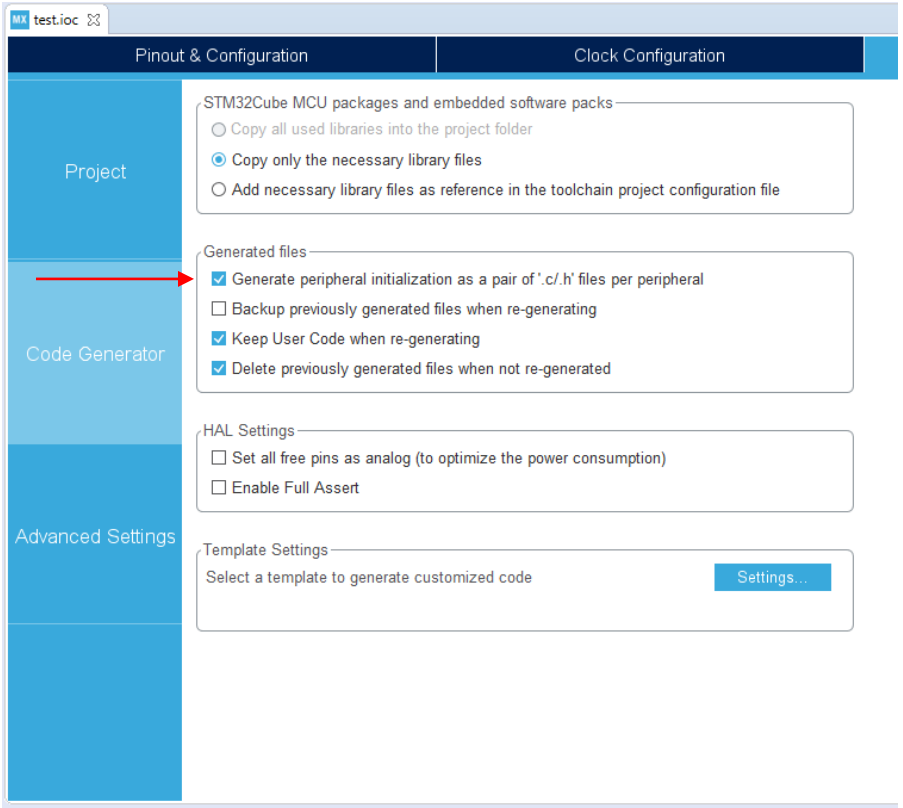
Click *Yes*.



Project Manager

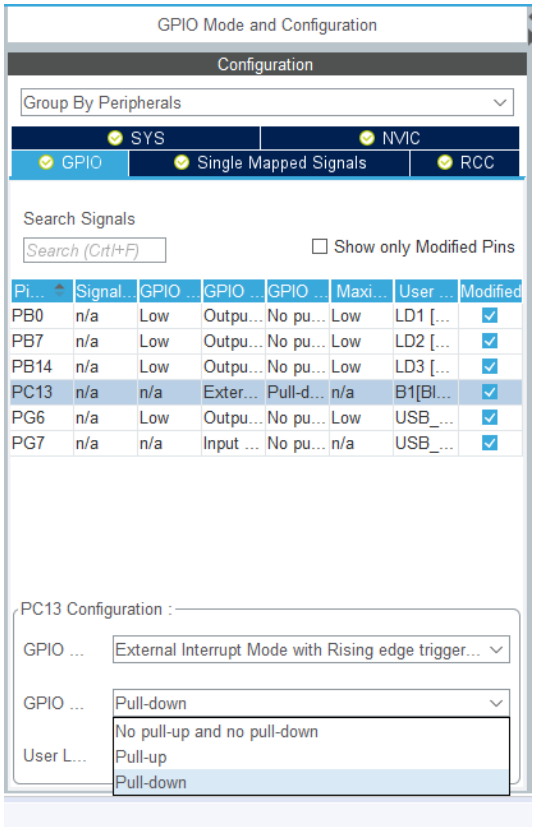
In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.





Pull-up o Pull-down ?

- After the last overview of the *GPIOs*, it should be noted that sometimes, especially for the *INPUT* pins, it is convenient to decide whether these should be ***pull-up*** or ***pull-down***, how to choose?



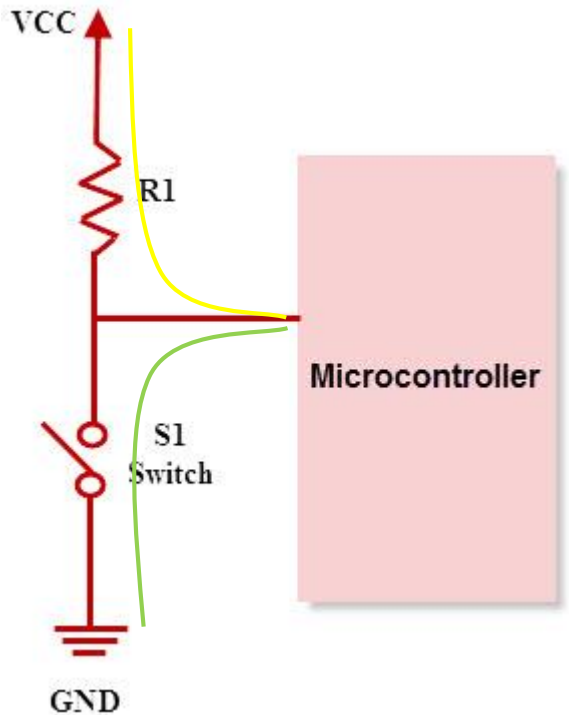
Pull-Up

In a digital circuit, the pull-up configuration is useful, when the switch that we are going to read is connected to GND:

- *OPEN* → VCC
- *CLOSED* → GND

In this way the input read will be:

- ‘1’ → Open Switch
- ‘0’ → Closed switch



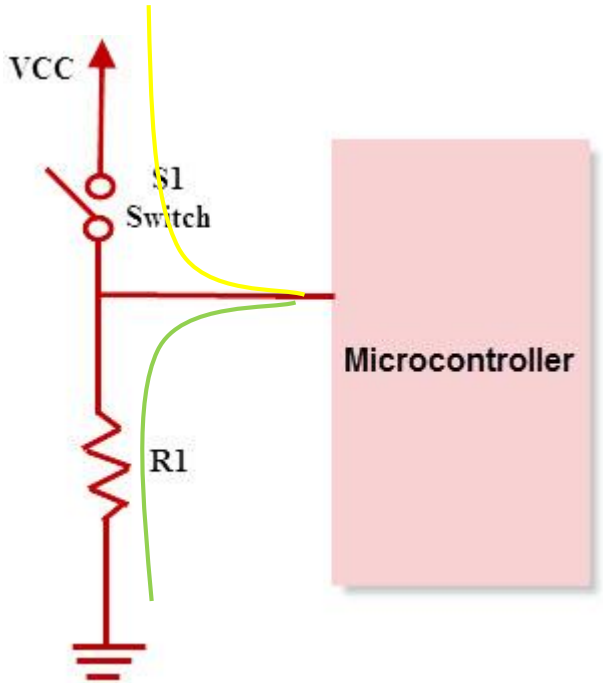
Pull-Down

n a digital circuit, the pull-down configuration is useful, when the switch we are going to read is connected to VCC:

- *CLOSED* → VCC
- *OPEN* → GND

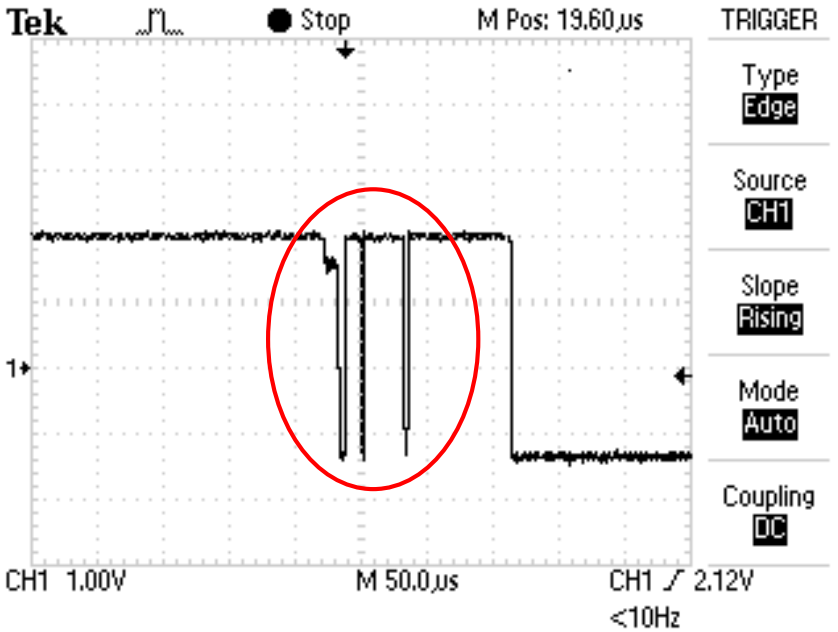
In this way the input read will be:

- '1' → Switch Closed
- '0' → Open switch



Debouncing, what is it?

- When reading an input, in particular a button, it is often useful to ***debounce*** it, that is, to prevent the input noise from causing spurious switching. How to do ? There are various ways, both hardware and software, here we will focus on the second type.



Switch Bouncing prodotto dalla pressione di un pulsante

Software debouce

- An example of software debounce can be the one shown here:
- Through the function ***HAL_GPIO_ReadPin()*** the status of the input is checked: in the event that this assumes the *'SET*' value then a counter is started which has the purpose of controlling how long the input is kept at a high level. The delay of 1ms causes the value assumed by the *now* variable to correspond to the time, in ms, during which our button was pressed.

main.h | main.c | *btnRead_HAL.ioc

```
256
257 /*debounce the button to prevent false readings*/
258 int readBtn(GPIO_TypeDef* GPIOx_PORT, uint16_t Bx_Pin){
259
260     B1_state=HAL_GPIO_ReadPin(GPIOx_PORT, Bx_Pin);
261
262     if (B1_state) {
263     int now=0;
264
265     do {
266         now++; //increment "now" to see for how long the Bx has been pressed
267         B1_state=HAL_GPIO_ReadPin(GPIOx_PORT, Bx_Pin); //read the Bx pin
268         HAL_Delay(1);
269     } while (B1_state);
270
271     if (now>DEBOUNCE_TIME) {
272           return 1; //correct reading, the user pressed the button for longer than the debounce time
273       }
274     }
275     return 0; //false reading
276 }
277
278 /* USER CODE END 4 */
```

- Consideriamo una pressione volontaria del bottone se questo è stato attivo continuamente per un tempo maggiore al **DEBOUNCE_TIME**, altrimenti viene considerata come spuria e quindi rumore.

Quante volte è stato premuto il pulsante ?

- It may be useful to understand how many times the button has been pressed and then evaluate the different cases, for example:
  - 1 press → Blink
  - 2 presses → LED on
- One strategy could be to define a USR_TIME time limit in which the user can press the button once or twice.
- During this first phase, the counter variable takes into account how many times the switch is pressed.

main.h | main.c | *btnRead_HAL.ioc

```
94  /* Infinite loop */
95  /* USER CODE BEGIN WHILE */
96  while (1) {
97
98      counter = 0 ;
99      int timer=USR_TIME; //max time for make the decision : press the button one or two times
100
101     while(timer>0 && counter<2){
102       if  (readBtn(B1_GPIO_Port,B1_Pin)) {
103           counter++; //check for how many times the button has been pressed, at most twice
104       }
105
106       timer--;
107       HAL_Delay(1);//wait 1ms
108     }
109
110     switch (counter) {
111       case 1: //blink led
112           blink_once(BLINK_TIME,LD1_GPIO_Port, LD1_Pin);
113       break;
114       case 2: //LEDs on
115            HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, SET);
116          break;
117       default: //do nothing
118
119       break;
120
121     }
122   /* USER CODE END WHILE */
123
124   /* USER CODE BEGIN 3 */
125 }
126 /* USER CODE END 3 */
```

Quante volte è stato premuto il pulsante ?

- As in the example above, the 1ms delay at the end of this pre-evaluation phase makes USR_TIME the ms that the user has available to make his choice.
- At the end of this phase it is possible to evaluate how many times the button has been pressed using the value contained in the counter variable

main.h | main.c | *btnRead_HAL.ioc

```
94  /* Infinite loop */
95  /* USER CODE BEGIN WHILE */
96  while (1) {
97
98      counter = 0 ;
99      int timer=USR_TIME; //max time for make the decision : press the button one or two times
100
101     while(timer>0 && counter<2){
102       if  (readBtn(B1_GPIO_Port,B1_Pin)) {
103           counter++; //check for how many times the button has been pressed, at most twice
104       }
105
106       timer--;
107       HAL_Delay(1);//wait 1ms
108     }
109
110     switch (counter) {
111       case 1: //blink led
112           blink_once(BLINK_TIME,LD1_GPIO_Port, LD1_Pin);
113       break;
114       case 2: //LEDs on
115            HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, SET);
116          break;
117       default: //do nothing
118
119       break;
120
121     }
122   /* USER CODE END WHILE */
123
124   /* USER CODE BEGIN 3 */
125 }
126 /* USER CODE END 3 */
```

blink_once()

- For completeness, the function that allows the LED to flash only once is also shown.
- The code is very simple to interpret but it is important to understand the ***RESET → SET*** passage on the first two lines which allows you to be sure of obtaining a flash **ON / OFF** and not vice versa.

```
246
247 /* toggle the led only 1 time*/
248 void blink_once(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin ){
249
250     HAL_GPIO_WritePin(GPIOx_PORT, LDx_Pin, RESET); //Led off
251     HAL_GPIO_WritePin(GPIOx_PORT, LDx_Pin, SET);    //Led on
252     HAL_Delay(time);
253     HAL_GPIO_WritePin(GPIOx_PORT, LDx_Pin, RESET);
254
255 }
256
```