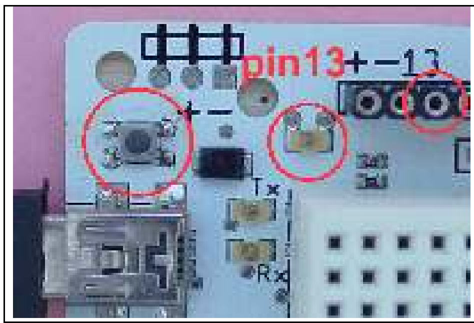
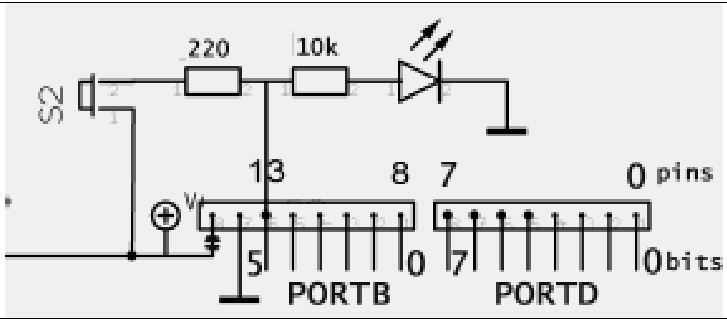
Push Button on Arduino pin 13

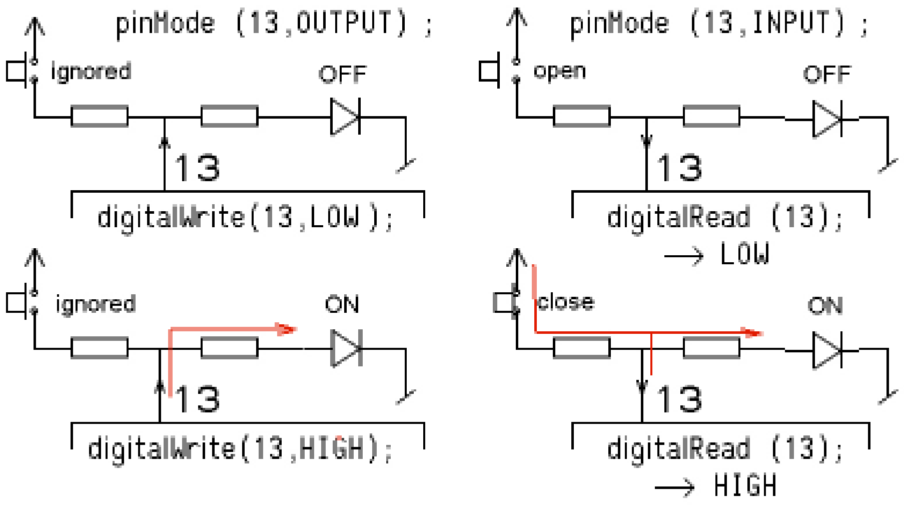
If you have an Arduino compatible boards, you can wire a push button of any make and benefit from the documentation below. Adding the protection resistor is not required if you avoid long depress actions in led mode.

1 Diduino push button

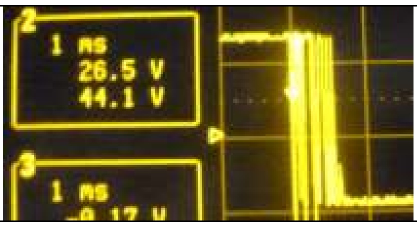
The Diduino has added a push button and a protection resistor on pin 13.

	 S2 220 10k 13 8 7 0 pins 5 0 7 0bits PORTB PORTD

Depending if pin13 is initialized as output or input, and button pushed or not, one have the following situations.



We do not need to explain how to blink the led. For the push button, it is important to understand the reading can be done different ways
1) the program wait till the button is depressed, execute the action and continues when the button is released. This is a blocking action
2) the program check regularly if the key is depressed. If yes, the action is executed,
3) the interrupt mode is iniotialized; when the button is depressed, the action is executed. Usually, a flag is set the program will check, do the action and clear the flag.

Bouncing of mechanical switches complicate the handling of the switch signal. Typically for a small switch, bouncing may create tens of pulses for several ms.	

We of course use functionnal names resulting of #define declarations, e.g.

```
    #define PushOn (digitalRead(13))   // active high
or  #define PushOn (PORTC & (1<<5))
```

PushOn is not an instruction; it is a boolean variable to be tested in a conditional instruction.
You surely know that `PushOff` is the same as !`PushOn`, no need to define it.

1) Wait on button depressed

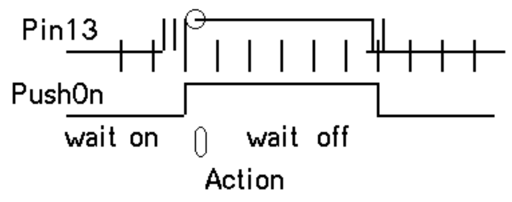
The program stops in a waiting loop

```
 while (!PushOn) {delay (20);}   // wait for action
```

After action, it wait for the button released

```
 while (PushOn) {delay (20);}   // wait released
```

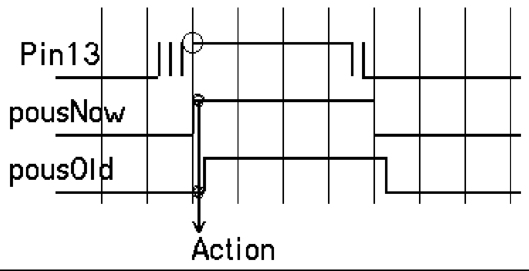
The delay is longer than the worse bouncing duration, typically 20ms.



2) Non blocking decision

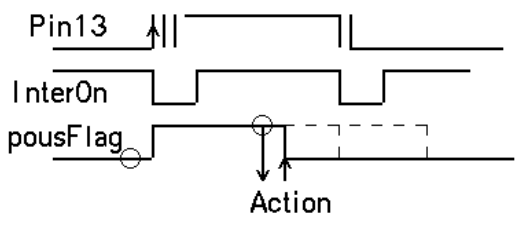
The program has several activities (read sensors, set moteur speed, blink), but every 20ms it checks if the button state has changed. This mean the previous value read has to be saved in an variable named e.g. `oldPush` and the present one under pushNow;

```
  pushNow = PushOn;
  if (pushNow && !oldPush)  {.. action .. }
  OldPush = PushOn ;
```



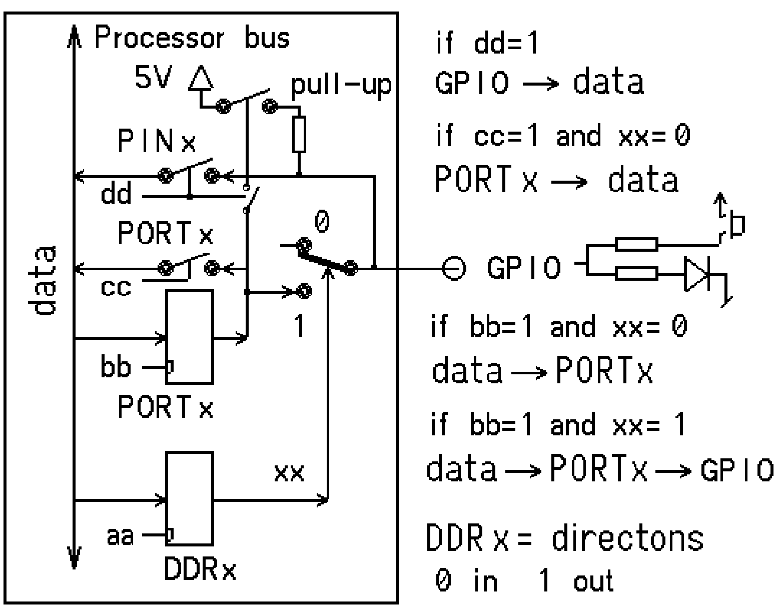
3) Interrupt

The AVR processors can be programmed to detect the change of value of any pin and call an interrupt routine. If the main program must know about the button, the interrupt activate a flag. Main program check the flag instead of the button.
The problem is one need to deactivate the interrupt during the bounces, that is use a timer that will interrupt after 20ms and restore the interrupt for handlind release situation.



Caution – internal pull-up

The Arduino designers were not aware apparently that with the AVR and most microcontrollers, there are good reasons to connect leds and button so they are active low. A pull-up is programmable inside the controller and simplify the wiring.
The pin13 schematic shows the led and its resistor plays the role of a pull-down.
On the AVR, a pull-up is connecter in input mode if the corresponding bit of the port register is HIGH. This brings a bad voltage of 2 volts on the processor pin.



Definition file

We need several definitions for which we can hesitate on the most clear name:
PushMode could be named Pin13In and LedMode Pin13Out

```
#define PushMode     pinMode (13,INPUT)
#define LedMode      pinMode (13,OUTPUT)
#define  PushOn      (digitalRead(13))
#define  LedOn  digitalWrite (13,1)
#define  LedOff digitalWrite (13,0)
```

```
#define PushMode  clearBit (DDRB,5)
#define LedMode    setBit  (DDRB,5)
#define  PushOn      (PORTB & (1<<5))
#define  LedOn  setBit  (PORTB,5)
#define  LedOff clearBit  (PORTB,5)
```

Remember, when reading the push button, one must be sure that the pull-up is not active.

Example 1: Alternate wait for button and led brinking periods

We want to blick n times after every push. Led must be Off (no pull-up when reading the button. This is easy because the two modes are one after the other.

The complete program can be found under www.didel.com/diduino/PushButton.zip

```
// Push1.ino   Button trigger a set of blinks
.. definitions and setup
void loop () {
       // wait push
  LedOff; PushMode; // pull-up not active
  while (!PushOn) {delay (20);}
  while (PushOn) {delay (20);}
     // blink
  delay (500);
  LedMode;
  for (byte i=0;i<3;i++) {
     LedOn;  delay (200);
     LedOff; delay (200);
  }
}
```

Example 2: Led13 is a realtime copy of the button.

A blocking function is not possible any more. One need to switch between the output and input mode. When in input mode (button read), the Led is off even if the program asks is to be on. The blinking must be as short as possible, which is not a problem.

Since we want to test the button while blinking, it will be necessary to save the led state and reestablish it. A function will do the work.

```
//Push2.ino| Led copy button
... #define ...
byte saveLed;
byte GetButton () {  // 1 if button on
  byte button=0;
  saveLed= (PORTB & (1<<5));
  LedOff;  // no pull-up in push mode
  PushMode;
  if (PushOn) {
     button=1;
  }
  PORTB |= saveLed;
  LedMode;
  return button;
}
```

```
void setup() {
}
byte butNow, butOld;
void loop () {
  delay (10);
  butNow= GetButton();
  if(butNow && !butOld) {
    LedMode;
    LedToggle;
  }
  butOld = butNow;
}
```

Example 3: Button toggle the led

We need now to take a non blocking decision.

```
//Push3.ino Button toggle the led
... #define ...
byte saveLed;
byte GetButton () {  // 1 if button on
  byte button=0;
  saveLed= (PORTB & (1<<5));
  LedOff;  // no pull-up in push mode
  PushMode;
  if (PushOn) {
     button=1;
  }
  PORTB |= saveLed;
  LedMode;
  return button;
}
```

```
void setup() {
}

byte butNow, butOld;
void loop () {
  delay (10);
  butNow= GetButton();
  if(butNow && !butOld) {
    LedMode;
    LedToggle;
  }
  butOld = butNow;
}
```

Example 4: Debugging help

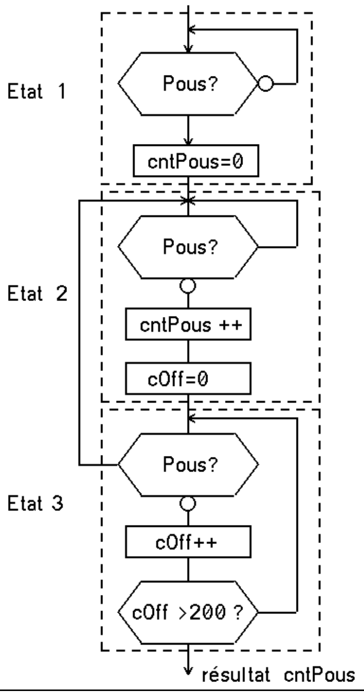
Insert in you program the function Stop (period); where you wish to stop and continue.
To make it simple her, the period of the blinking is the parameter. You can blink several times once or repetidively with simple functions.
	```
	.. definitions, setup and GetButton ()

	void Stop (byte nn) {
	    do {    // blink waiting for the
	push
	      LedToggle;
	      delay (nn);
	    } while (!GetButton());
	    while (GetButton()){delay (20);}
	//wait for release
	    LedOff;
	}

	void loop () {
	  delay (1000);
	  Stop(100);
	}
	```

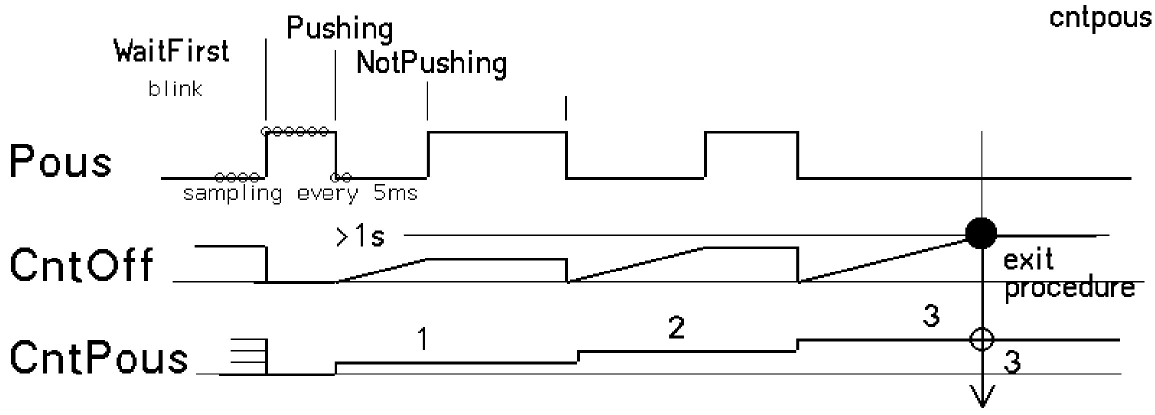
Example 5: Counting pushes

Counting how many depress on a switch has many applications. The mose useful is selecting a program or an operating mode at power up, or giving a parameter.

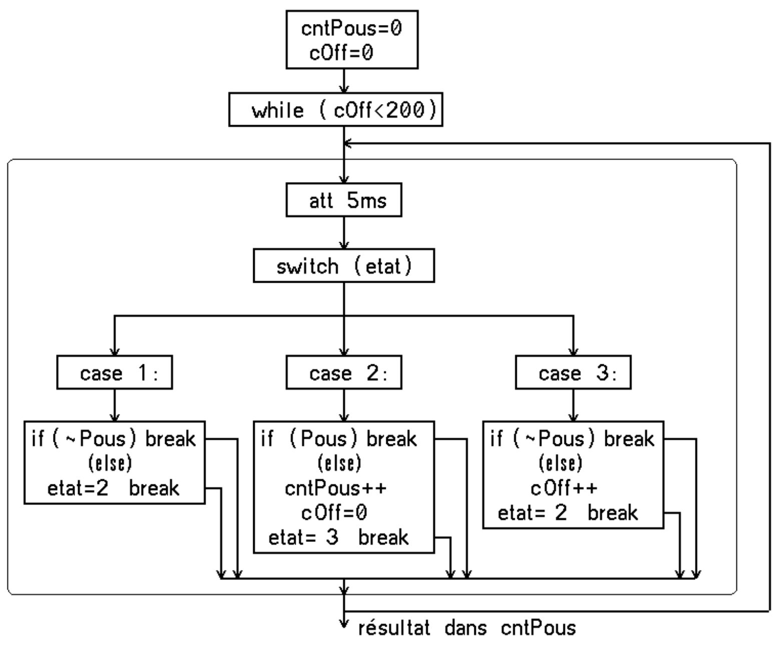
What the routine must do is:
1.- Wait for a first depress of the button, set cntPous = 0
When pressed, enter a loop
a) cntPous +=1
wait for release button and go to b
b) When released
initialize a timeout counter cOff
wait for depress
while checking timeout cOff++
if depressed, goto a)
if timeout, goto c
c) Blink cntPous times
	 Etat 1
	Pous?
	cntPous=0
	Etat 2
	Pous?
	cntPous ++
	cOff=0
	Etat 3
	Pous?
	cOff++
	cOff >200 ?
	résultat cntPous
		```
		(1)
		  while (!Pous) delay (5) ;
		  cntPous = 0 ;

		(2)
		  while (Pous) delay (5) ;
		  cntPous++ ;
		  cOff = 0 ;

		(3)
		  while (cOff++ < 200) {
		    delay (5) ;
		    if (Pous) goto (2) ;
		    cOff++ ;
		  }
		```



The program use a state machine. the 3 states 1, 2, 3 are named `WaitFirst, Pushing, NotPushing.`
A fast blinking show the user it is time to push the number of time he desires.
The instructions that confirm the number of pushes are very usefull.



Key instructions:

```
byte cntPous ;
byte cOff = 0;
while (cntOff < 200)
{
  delay (5);
  switch (etat)
    {
    case 1:
      if (!Pous) break;

      etat=2; break;
    case 2:
      if (Pous) break;
      cntPous++; cOff=0;
      etat=3; break;
    case 3:
      if (!Pous) break;
      cOff++;
      etat=2; break;
    }  // end switch
}  // end while
```

```
// TestCntPous.ino fast blink while waiting first push
//                  slow blinks to confuîrm number of pushes

#define PushMode  bitClear (DDRB,5); PORTB=0;
#define LedMode    bitSet  (DDRB,5)
#define  PushOn    (PINB & (1<<5))
#define  LedOn   bitSet   (PORTB,5)
#define  LedOff bitClear  (PORTB,5)
#define  LedToggle PORTB ^= (1<<5)

byte cntPous; // global variable

//<<<<<<<<<<<< Fonction GetPous --> numbe of
pushes 1 2 3..
byte cli, cntOff = 0;
enum { WaitFirst, Pushing, NotPushing }  next =
WaitFirst;
byte GetPous () {
  cntOff = 0;
  while (cntOff < 200) {
    switch (next) {
    case WaitFirst: // blink fast waiting for push
      LedOff; PushMode;
      if (!(PushOn))  {
        LedMode ;
        LedOn; delay (100); LedOff; delay (100);
        break;
      }
      cntPous  = 0 ;
      next = Pushing; break;
    case Pushing:   // wait release
      delay(5);
      PushMode ;
      if (PushOn) break;
      cntPous++ ; cntOff = 0 ;
      PORTC=cntPous;  // optional test
      if (cntPous == 10) cntPous=9; // saturate
      next = NotPushing ;  break ;
    case NotPushing:
      delay(5);
      cntOff++;  //200x5 = 1 sec
      if (!(PushOn)) break ; // release
      next = Pushing ; break;
    } // end switch
  }
  next = WaitFirst;
  LedMode ;   //  blink cntPous time
  for (cli=0; cli<cntPous; cli++) {
    LedOn ; delay(200);  LedOff ; delay(300);
  }
  PushMode ;
  return cntPous ;
}
void setup () {
  ModeLed;
  DDRC = 0b11111111;  // optional test
}

void loop () {
  PORTC = GetPous ();  // optional test
  delay (1000); // pas nécc
}
```

Library LedPush13.h

File `PushLed13.h` includes the definitions and the 3 functions used beforhands..
`TestCntPous.ino` can now be written;

```
//TestCntPousLib.ino
#include  "PushLed13.h"
void setup () {
  ModeLed;
  DDRC = 0b11111111;  // optional test
}
void loop () {
  PORTC = GetPous ();  // optional test
  delay (1000);
}
```

jdn 151202