DM tutoriel Timers

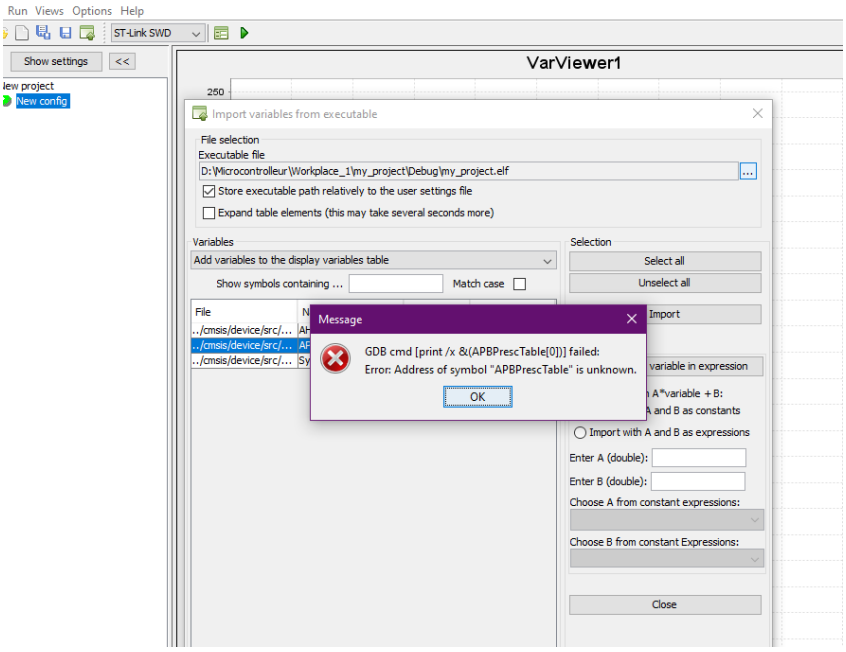
EI2I3-IIA **Julian BARKOUDEH**

Note 1 : Tous les codes de BSP.c relatifs à ce tutoriel se trouvent dans la partie Timer, à partir de la ligne 340.

Note 2 : La fonction de printf est appelée j_printf dans mes codes. Or dans le rendu de chaque main.c je les avait appelés mon_printf

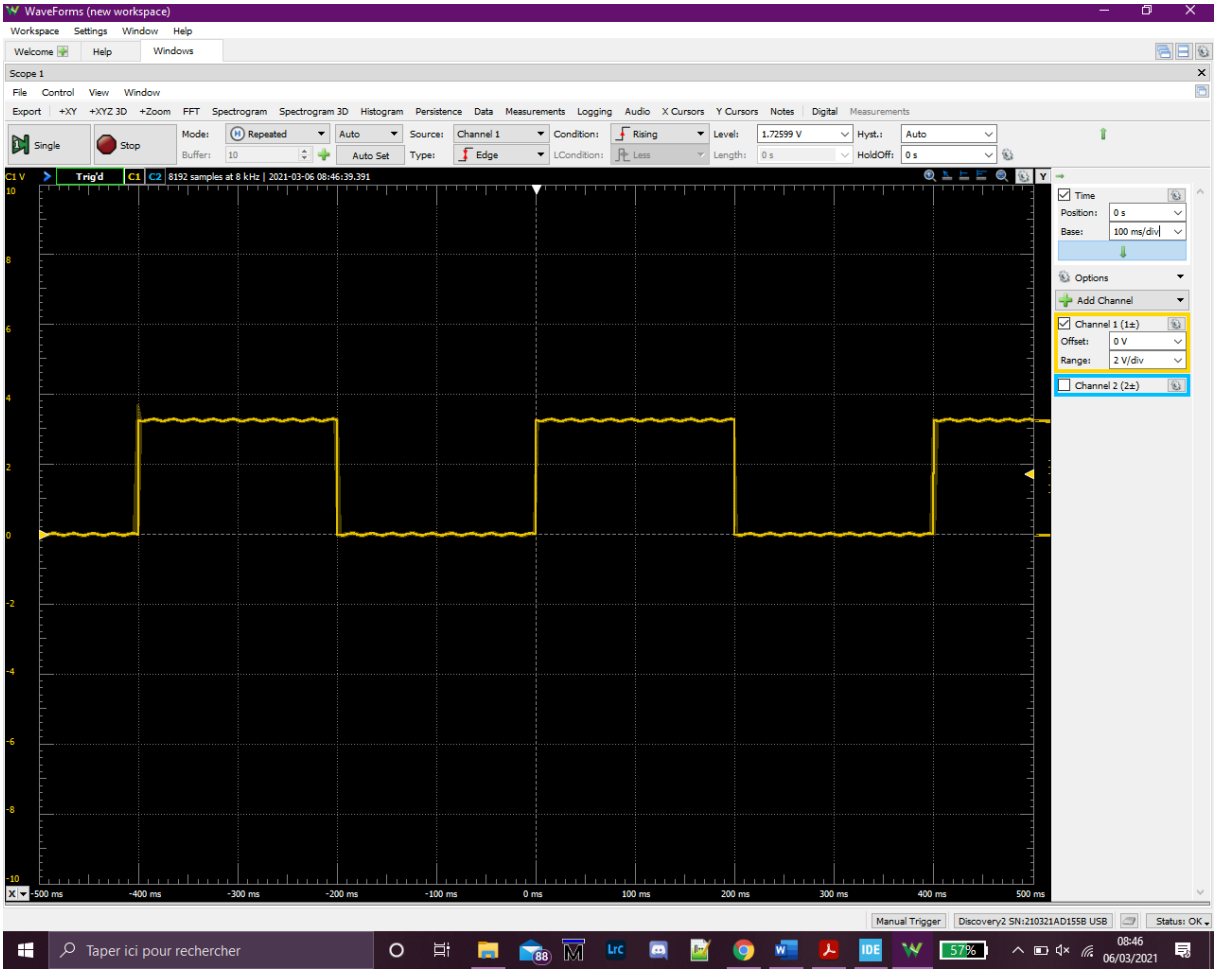
1.1 Base de temps simple : Mutineries

```c
/*
 * main.c
 * Created on: Feb 3, 2021
* Author: BARKOUDEH */

#include "stm32f0xx.h"
#include "bsp.h"
#include "delay.h"
 #include "main.h"
static void SystemClock_Config(void);

// Main program
int main()
    {
        // Configuration horloge du systeme
        SystemClock_Config();
        //Initialisation signal PWM
        BSP_LED_Init();
        BSP_TIMER_Timebase_Init();
        uint8_t TIM6_counter;
        uint8_t TIM6_UIF;
        while(1){
            TIM6_counter = TIM6->CNT;
            if(TIM6->SR == 1){
                if(TIM6_UIF == 0){
                    TIM6_UIF = 1024;
                }
                else if(TIM6_UIF == 1024){
                    TIM6_UIF = 0;
                }
                TIM6->SR = 0;
            }
        }
}
/*
```



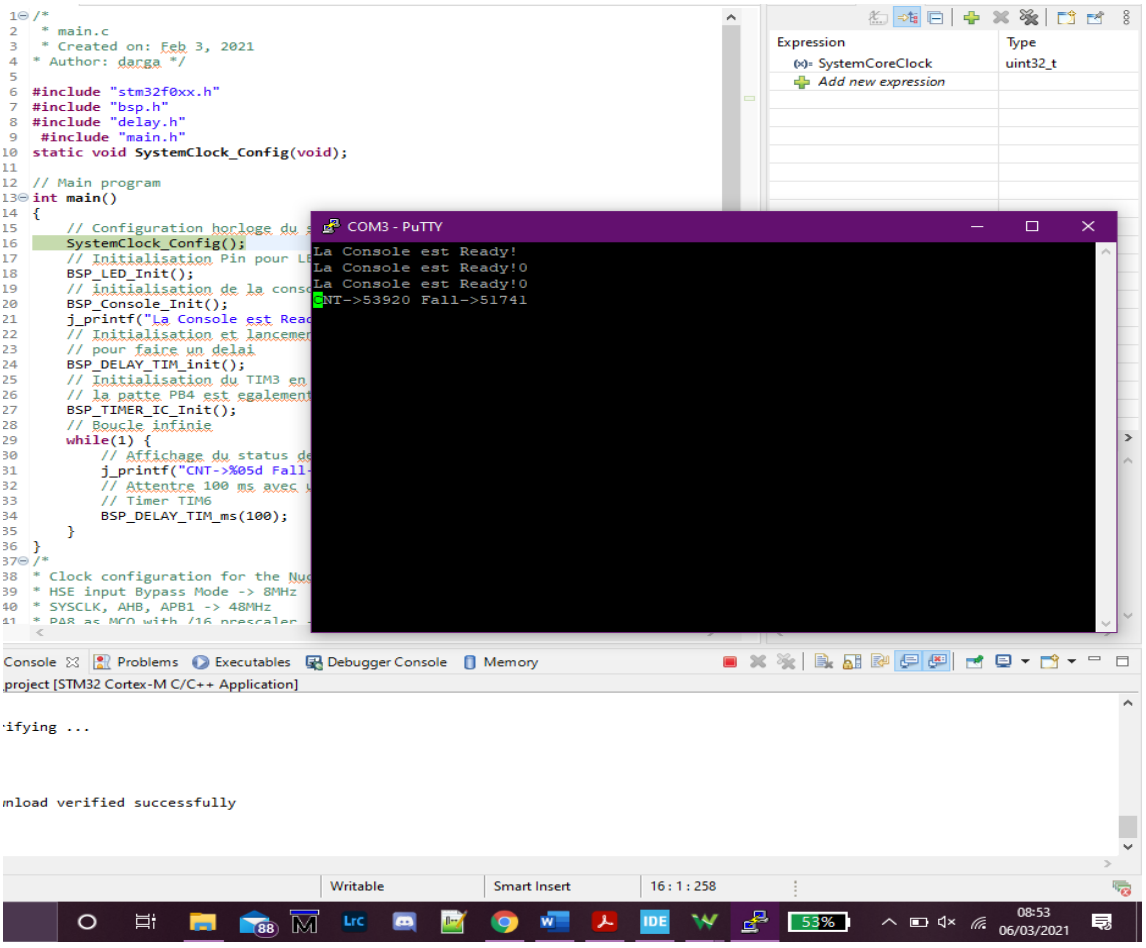
Le code ci-dessus permet de générer le signal en question. Or l'erreur suivant empêche de visualiser les variables sur STMstudio.

1.2) Utilisation du Timer pour gérer une fonction de temporisation robuste

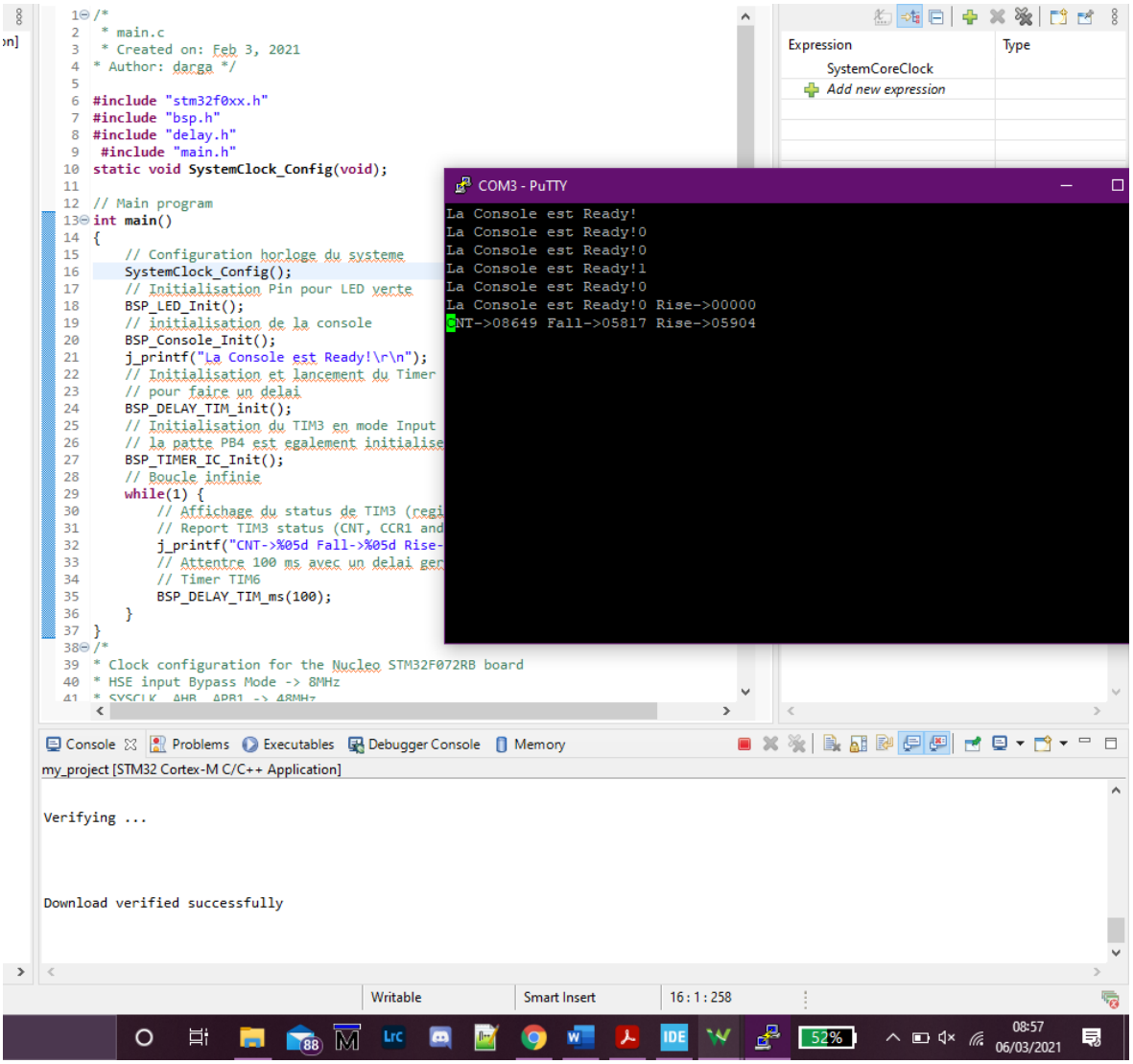


On branche l'oscilloscope sur le pin PA5 lié au LED afin de visualiser le basculement avec les fonctions delay introduites.

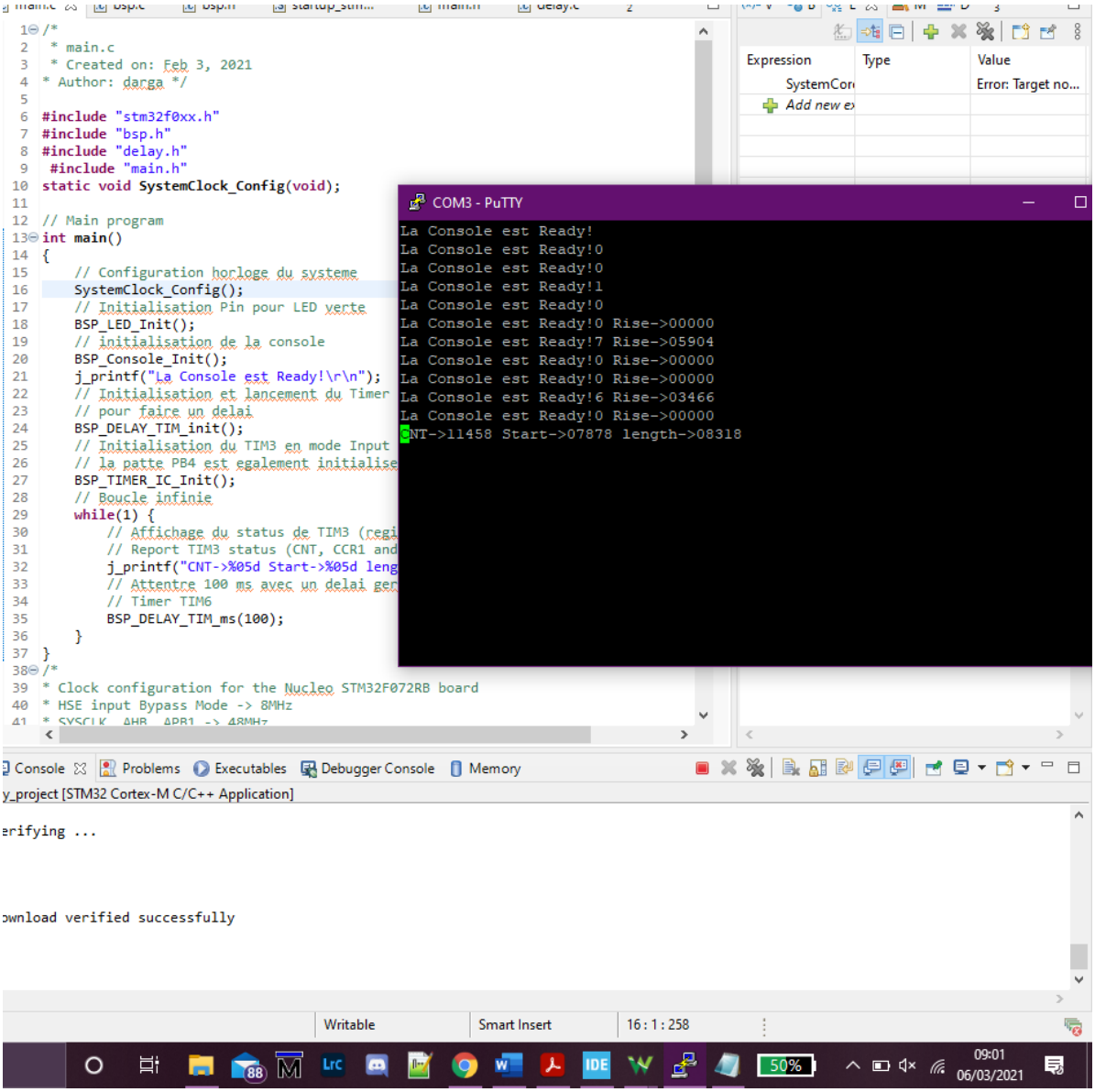
2.1 Mode Input Capture : Single edge – détection d'un seul front

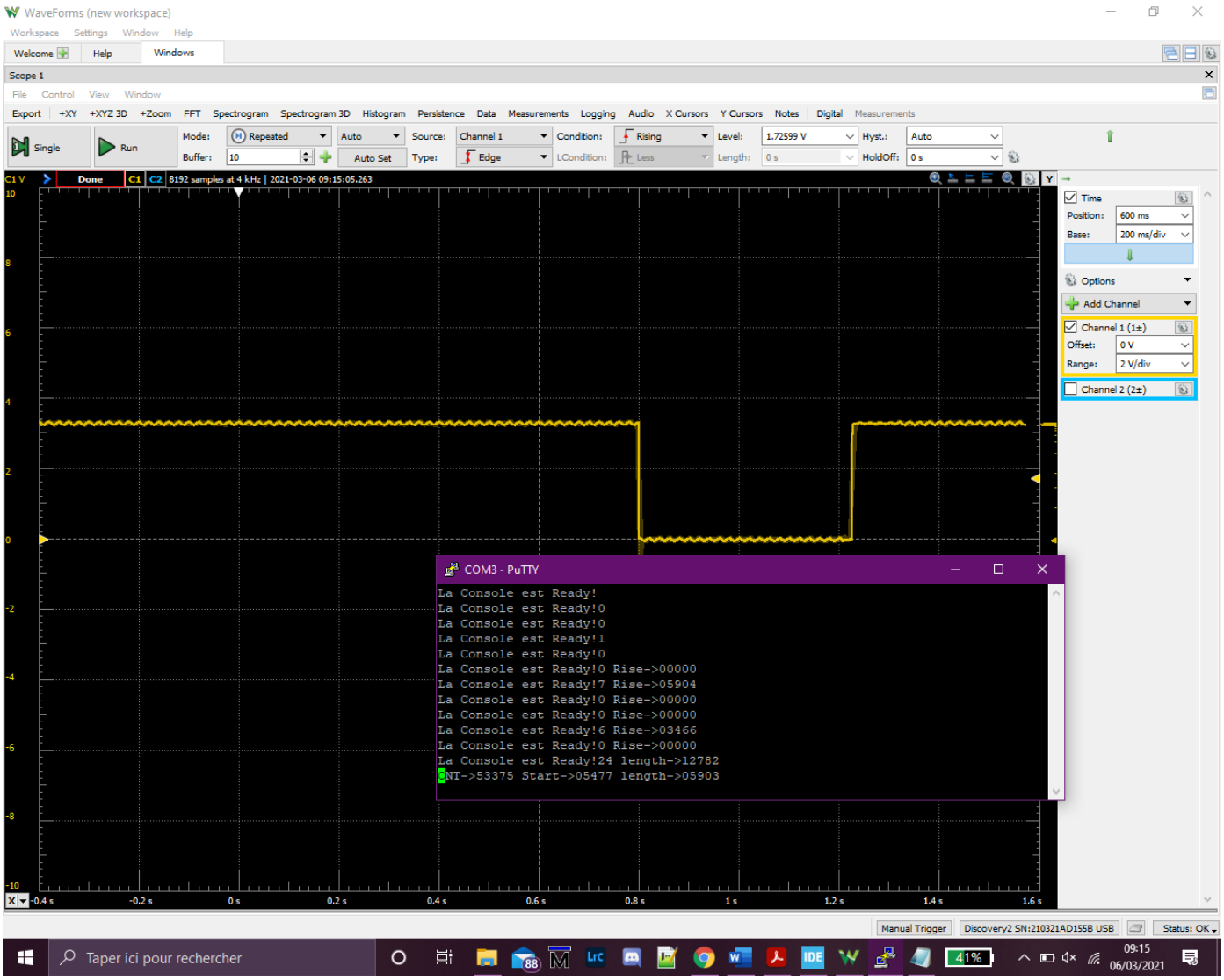


2.2 Mode Input Capture : Dual edges captures – détection de deux fronts



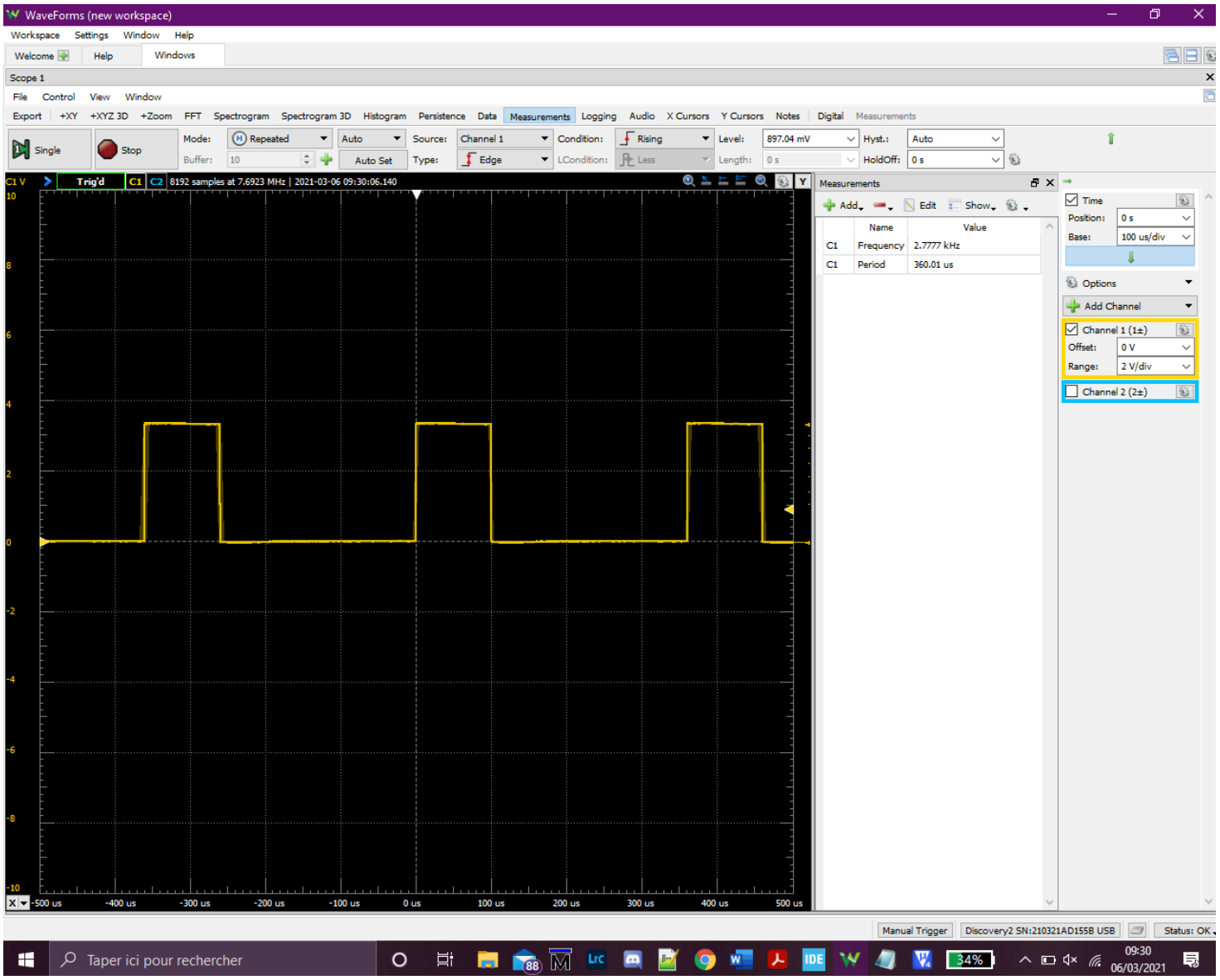
2.3 Mode Input Capture : Dual edges captures – détection de deux fronts avec reset automatique





Sur l'affichage de printf on mesure 0.59 s, ceci est cohérent avec la lecture graphique du temps du signal affiché sur l'oscilloscope. On mesure t= 1.3-0.8 = 0.5 s.

3.1 Qu'est-ce qu'un signal PWM



```c
//-----------------------Signal PWM--------------------------------
// BSP_PWM_Init()
//Fonction initialisation d'un signal PWM sur le pin PA9
void BSP_PWM_Init(){
    RCC->AHBENR |= (1<<17);
    /*
    // Ecrire "10" sur les bits b11b10 du registre (GPIOA_MODER) (AF)
    GPIOA->MODER |= (1<<17);
    GPIOA->MODER &= ~(1<<16);*/
    // Ecrire "10" sur les bits b11b10 du registre (GPIOA_MODER) (AF)
    GPIOA->MODER |= (1<<19);
    GPIOA->MODER &= ~(1<<18);
    //Def de l'AF
    //Ecrire "0010" (AF2) sur b3b2b1b0
    GPIOA->AFR[1] &= ~(1<<3);
    GPIOA->AFR[1] &= ~(1<<2);
    GPIOA->AFR[1] |= (1<<1);
    GPIOA->AFR[1] &= ~(1<<0);
    //APB2 timer1
    RCC->APB2ENR |= (1<<11);
    //Configuration des caractéristiques du signal PWM
    //Time on
    TIM1->PSC = 47;
    //Résolution
    TIM1->ARR = 180;
    // Rapport cyclique
    TIM1->CCR1 = 50;
    TIM1->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1| TIM_CCMR1_OC1PE;
    TIM1->CCER |= TIM_CCER_CC1E;
    TIM1->BDTR |= TIM_BDTR_MOE;
    TIM1->CR1 |= TIM_CR1_CMS_0 | TIM_CR1_CEN;
    TIM1->EGR |= TIM_EGR_UG;
}
```

En se basant sur la manuelle de la carte, le code ci-dessus présent dans BSP.c permet d'initialiser un signal PWM sur la pate PA8.

Tout d'abord en met la pate PA8 en mode Highspeed, ensuite on active le AF2 sur la pate PA8. On génère un signal PWM de rapport cyclique de 50%, et d'une fréquence de 2.7 KHZ. La configuration de la fréquence est faite avec le registre PSC, on le règle à 47 us qui définit la période du signal. On définie la résolution par le registre ARR, on le règle à 180 pour avoir 180 valeurs du rapport cyclique.