Hi

I have noticed a situation that does not make the chip behave in the same manner every time it boots up. Here is the code

```
#define HSE_VALUE (8000000)

#include <stddef.h>
#include <stdint.h>
#include "stm32f4xx.h"
#include <assert.h>
#include "stm32f4xx_exti.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_syscfg.h"
#include "misc.h"



#define configCPU_CLOCK_HZ (168000000)
#define APP_IRQ_PREEMPT_PRIO_CRITICAL (0)

#define DRIVER_START_FREQ           (40000)
#define SYNC_PULSE_MAX_FREQ         (80000)
#define SYNC_PULSE_MIN_FREQ         (10000)
#define MAX_FREQ_TICK_CHANGE        (11)
#define STABLE_TIMER                (TIM1)

#define SAFTEY_DISTANCE_STEP        (250)
#define DEAD_TIME                   (50)



/** @brief Return the largest of two numbers.
 * @param a First number.
 * @param b Second number.
 */
#define MAX(a, b) ((a >= b)? a : b)

/** @brief Return the smallest of two numbers.
 * @param a First number.
 * @param b Second number.
 */
#define MIN(a, b) ((a >= b)? b : a)

#define GREEN_PIN       (GPIO_Pin_7)
#define GREEN_PORT   (GPIOB)

#define GREEN_ON()  (GREEN_PORT->BSRRL = GREEN_PIN)
#define GREEN_OFF()     (GREEN_PORT->BSRRH = GREEN_PIN)
#define GREEN_GET()  (GREEN_PORT->ODR   & GREEN_PIN)

#define BLUE_PIN    (GPIO_Pin_5)
#define BLUE_PORT   (GPIOA)

#define BLUE_ON()   (BLUE_PORT->BSRRL = BLUE_PIN)
#define BLUE_OFF()  (BLUE_PORT->BSRRH = BLUE_PIN)
#define BLUE_GET()  (BLUE_PORT->ODR   & BLUE_PIN)

/*
 * OFF means no signal on the driver pins
 * IDLE means there are signals on the driver pins, but they produce as little power as possible
 * SOFT_START is soft start :-)
 * REGULATE_START is the short middle state between soft start and regulating. it is the request
state for regulating
```

```
 * REGULATING sets the period of the movable timer to a little longer then the stable one.
 * FREQ_ADJUST changes the frequency of both involved timers.
 */

typedef enum {
    SYNC_TIMER_OFF                = 0,
    SYNC_TIMER_IDLE               = 1,
    SYNC_TIMER_SOFT_START         = 2,
    SYNC_TIMER_REGULATE_START     = 3,
    SYNC_TIMER_REGULATING         = 4,
    SYNC_TIMER_FREQ_ADJUST        = 5
} synctimer_state;


// #define PULSE_FROM_PERIOD(period) (( (uint16_t) (((uint32_t) 5 * (period - 1)) / 10)    )) //
This is the old complex way of calculating the pulse
#define PULSE_FROM_PERIOD(period) ((period -1 )/ 2)

// (configCPU_CLOCK_HZ / DRIVER_START_FREQ) - 1
#define TICKS_FROM_FREQ(wanted_freq) ((configCPU_CLOCK_HZ / wanted_freq) - 1)

static uint16_t safty_before_pulse  = 0;
static uint16_t safty_after_pulse   = 0;
static uint16_t safty_before_period = 0;

#define MAX_NBR_RESTARTS (1)
static uint16_t too_long_feedback_restart = 0;

// These defines are used inorder to be able to calculate CCMR1_X variants once only
#define ALL_BITS_NOTOC2_MODE    (0x8FFF)
#define FORCE_ACTION_ACTIVE     (0x5000)
#define FORCE_ACTION_INACTIVE   (0x4000)
#define FORCE_ACTION_TOGGLE     (0x3000)
static uint16_t CCMR1_ACTIVE   = 0;
static uint16_t CCMR1_INACTIVE = 0;
static uint16_t CCMR1_TOGGLE   = 0;

static synctimer_state timer_state          = SYNC_TIMER_OFF;
static synctimer_state prev_timer_state     = SYNC_TIMER_OFF;

typedef enum {
    SYNC_DECREASE                   = -1,
    SYNC_STABLE                     = 0,
    SYNC_ACCELRERATE_REGULATE       = 1
} synctimer_movement;

static synctimer_movement next_move = SYNC_STABLE;

static int32_t movement = 0;

// This is here to disable the PWM outputs if there is ALOT of problems.

static inline void calculate_new_pulse_parameters(uint16_t timer_period) {
    uint16_t pulse  = PULSE_FROM_PERIOD(timer_period);
    safty_before_pulse  = pulse  - SAFTEY_DISTANCE_STEP;
    safty_after_pulse   = pulse  + SAFTEY_DISTANCE_STEP;
    safty_before_period = timer_period - SAFTEY_DISTANCE_STEP;
}


static void inline KILL_TIMER_INTERRUPT (uint16_t inter) {
    STABLE_TIMER->DIER &= ~inter; // Kill the interrupt
    STABLE_TIMER->SR    = ~inter; // Clear any stray interrupt
```

```
}

static void inline RESTART_TIMER_INTERRUPT (uint16_t inter) {
    STABLE_TIMER->SR    = ~inter; // Clear any stray interrupt
    STABLE_TIMER->DIER |= inter;  // re-enable the interrupt
}


static void inline CC2_STUFF_TO_DO(uint16_t new_ccr2) {
    if (new_ccr2 > STABLE_TIMER->CCR1) {
        // CC2 in second half period
        // STABLE_TIMER->CCR2 = MAX( MIN(STABLE_TIMER->CCR2 - STABLE_TIMER->CCR1 + next_move,
STABLE_TIMER->CCR1 - SAFTEY_DISTANCE_STEP), SAFTEY_DISTANCE_STEP);
        STABLE_TIMER->CCR2 = MIN(new_ccr2 - STABLE_TIMER->CCR1 + next_move, safty_before_pulse);

    }
    else {
        // CC2 in first half period
        // STABLE_TIMER->CCR2 = MAX(MIN(STABLE_TIMER->CCR2 + STABLE_TIMER->CCR1 + next_move,
STABLE_TIMER->ARR - SAFTEY_DISTANCE_STEP),  STABLE_TIMER->CCR1 + SAFTEY_DISTANCE_STEP) ;
        STABLE_TIMER->CCR2 = MIN(new_ccr2 +  STABLE_TIMER->CCR1 + next_move,
safty_before_period);
    }
    STABLE_TIMER->SR = ~TIM_IT_CC2;// Clear
}


void TIM1_CC_IRQHandler(void) {
    uint16_t timer_status = (STABLE_TIMER->SR & STABLE_TIMER->DIER);

    if ((timer_status & TIM_IT_CC3) != 0 ) {
        // CC3
        // This if statement can PERHAPS be removed in order to improve speed and efficiency
        // MAY not trigger in this example! Tie it to GND
        while(1);
    } // End of CC3
    else if ((timer_status & TIM_IT_CC2) != 0 ) {
        // CC2
        GREEN_ON();
        CC2_STUFF_TO_DO(STABLE_TIMER->CCR2); // This clears the interrupt aswell
        GREEN_OFF();

    } // END OF CC2
    if ((timer_status & TIM_IT_CC1) != 0 ) {
        BLUE_ON();
        RESTART_TIMER_INTERRUPT(TIM_IT_CC2);
        STABLE_TIMER->SR    = ~TIM_IT_CC1;// Clear
        BLUE_OFF();
    } // END of CC1
}

void TIM1_UP_TIM10_IRQHandler(void) {
    uint16_t timer_status = STABLE_TIMER->SR;
    if ((timer_status & TIM_IT_Update) != 0 ) {
        GREEN_ON();
        RESTART_TIMER_INTERRUPT(TIM_IT_CC1 | TIM_IT_CC2);
        STABLE_TIMER->SR = ~(TIM_IT_Update); // Clear this interrupt
        GREEN_OFF();
    }

}

int main (void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;
    TIM_ICInitTypeDef  TIM_ICInitStructure;
    TIM_BDTRInitTypeDef TIM_BDTRInitStructure;
    EXTI_InitTypeDef   EXTI_InitStructure;
    uint16_t tmp  = 0;

    uint16_t pulse, period;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1  | RCC_APB2Periph_SYSCFG, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;

    // TIM1_CH1 (LOW GATE 1)
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_8 ;
    GPIO_Init(GPIOA, &GPIO_InitStructure);   // This toggles the pin

    // TIM1_CH1N (HIGH GATE 1)
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_TIM1);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // TIM1_CH2 (LOW GATE 2)
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_TIM1);
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9 ;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // TIM1_CH2N (HIGH GATE 2)
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_TIM1);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;

    // Configuration TIM1_CH3 input capture

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_TIM1);
    GPIO_InitStructure.GPIO_Mode    = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed   = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_OType   = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin     = GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // A simple debug pin to understand what happens here
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
    GPIO_InitStructure.GPIO_Pin   = GREEN_PIN;
    GPIO_Init(GREEN_PORT, &GPIO_InitStructure);
    GREEN_OFF();


    // A simple debug pin to understand what happens here
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
    GPIO_InitStructure.GPIO_Pin   = BLUE_PIN;
    GPIO_Init(BLUE_PORT, &GPIO_InitStructure);
    BLUE_OFF();

    period = TICKS_FROM_FREQ(DRIVER_START_FREQ);
    pulse  = PULSE_FROM_PERIOD(period);

    TIM_Cmd(STABLE_TIMER, DISABLE);
    /* Time Base configuration */
    TIM_TimeBaseStructure.TIM_Prescaler         = 0;
    TIM_TimeBaseStructure.TIM_CounterMode       = TIM_CounterMode_Up;
    TIM_TimeBaseStructure.TIM_Period            = period;
    TIM_TimeBaseStructure.TIM_ClockDivision     = 0;
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(STABLE_TIMER,  &TIM_TimeBaseStructure);

    calculate_new_pulse_parameters(period);

    timer_state = SYNC_TIMER_OFF;

    // CC1 generates the stable pattern
    TIM_OCInitStructure.TIM_OCMode       = TIM_OCMode_PWM2;
    TIM_OCInitStructure.TIM_OutputState  = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
    TIM_OCInitStructure.TIM_Pulse        = pulse;
    TIM_OCInitStructure.TIM_OCPolarity   = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OCNPolarity  = TIM_OCNPolarity_High;
    TIM_OCInitStructure.TIM_OCIdleState  = TIM_OCIdleState_Reset;
    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Set;
    TIM_OC1Init(STABLE_TIMER, &TIM_OCInitStructure);

    // CC 2 is now toggle mode
    TIM_OCInitStructure.TIM_OCMode       = TIM_OCMode_Toggle;
    TIM_OCInitStructure.TIM_OutputState  = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
    TIM_OCInitStructure.TIM_Pulse        = SAFTEY_DISTANCE_STEP;
    TIM_OCInitStructure.TIM_OCPolarity   = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OCNPolarity  = TIM_OCNPolarity_High;
    TIM_OCInitStructure.TIM_OCIdleState  = TIM_OCIdleState_Reset;
    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Set;
    TIM_OC2Init(STABLE_TIMER, &TIM_OCInitStructure);

    TIM_ICInitStructure.TIM_Channel     = TIM_Channel_3;
    TIM_ICInitStructure.TIM_ICPolarity  = TIM_ICPolarity_Rising;
    TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
    TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    TIM_ICInitStructure.TIM_ICFilter    = 0x0;
    TIM_ICInit(STABLE_TIMER, &TIM_ICInitStructure);

    // These must be enabled as they are used
    TIM_ARRPreloadConfig(STABLE_TIMER, ENABLE);
    TIM_OC1PreloadConfig(STABLE_TIMER, TIM_OCPreload_Enable);
    TIM_OC2PreloadConfig(STABLE_TIMER, TIM_OCPreload_Enable);

    // The rest will be disabled as they must be changed FAST
    TIM_CCPreloadControl(STABLE_TIMER, DISABLE); // ENABLE
    // TIM_OC2PreloadConfig(STABLE_TIMER, TIM_OCPreload_Disable);
    TIM_OC3PreloadConfig(STABLE_TIMER, TIM_OCPreload_Disable);
    TIM_OC4PreloadConfig(STABLE_TIMER, TIM_OCPreload_Disable);

    /* Automatic Output enable, Break, dead time and lock configuration*/
    TIM_BDTRInitStructure.TIM_OSSRState         = TIM_OSSRState_Enable;
    TIM_BDTRInitStructure.TIM_OSSIState         = TIM_OSSIState_Enable;
    TIM_BDTRInitStructure.TIM_LOCKLevel         = TIM_LOCKLevel_OFF;
    TIM_BDTRInitStructure.TIM_DeadTime          = DEAD_TIME;
    TIM_BDTRInitStructure.TIM_Break             = TIM_Break_Disable;
    TIM_BDTRInitStructure.TIM_BreakPolarity     = TIM_BreakPolarity_High;
    TIM_BDTRInitStructure.TIM_AutomaticOutput   = TIM_AutomaticOutput_Enable ; //
TIM_AutomaticOutput_Enable TIM_AutomaticOutput_Disable

    TIM_BDTRConfig(STABLE_TIMER,  &TIM_BDTRInitStructure);


    // Interrupts must be running before starting the timer!
    STABLE_TIMER->SR       = ~(TIM_IT_CC1 | TIM_IT_CC2 | TIM_IT_CC3 | TIM_IT_CC4 |
TIM_IT_Update);
    // TIM_IT_CC3 cannot be enabled at start!
    // TIM_ITConfig(STABLE_TIMER, TIM_IT_CC1 | TIM_IT_CC2 | TIM_IT_Update, ENABLE);
    TIM_ITConfig(STABLE_TIMER, TIM_IT_Update, ENABLE);

    NVIC_SetPriority(TIM1_CC_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
APP_IRQ_PREEMPT_PRIO_CRITICAL, 2));
    NVIC_EnableIRQ(TIM1_CC_IRQn);
    NVIC_SetPriority(TIM1_UP_TIM10_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),
APP_IRQ_PREEMPT_PRIO_CRITICAL, 1));
    NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);

    // Prepare the CCMR1 variables for fast use.
    // These must be calculated before starting the timer.
    tmp             = (STABLE_TIMER->CCMR1 &  ALL_BITS_NOTOC2_MODE);
    CCMR1_ACTIVE    = tmp  | FORCE_ACTION_ACTIVE;
    CCMR1_INACTIVE  = tmp  | FORCE_ACTION_INACTIVE;
    CCMR1_TOGGLE    = tmp  | FORCE_ACTION_TOGGLE;

    // Interrupts must be running before we start the timer
    // BLUE_ON();

    // STABLE_TIMER->BDTR |= TIM_BDTR_MOE; // Enable outputs

    // STABLE_TIMER->CNT  = SAFTEY_DISTANCE_STEP + 10;
    STABLE_TIMER->CNT  = 0;
    TIM_Cmd(STABLE_TIMER, ENABLE);
    // STABLE_TIMER->CCMR1 = CCMR1_TOGGLE;
    // STABLE_TIMER->EGR   = TIM_EventSource_COM;

    timer_state = SYNC_TIMER_IDLE;

    TIM_OC2PreloadConfig(STABLE_TIMER, TIM_OCPreload_Disable);
    //BLUE_OFF();

    while(1);

}
```

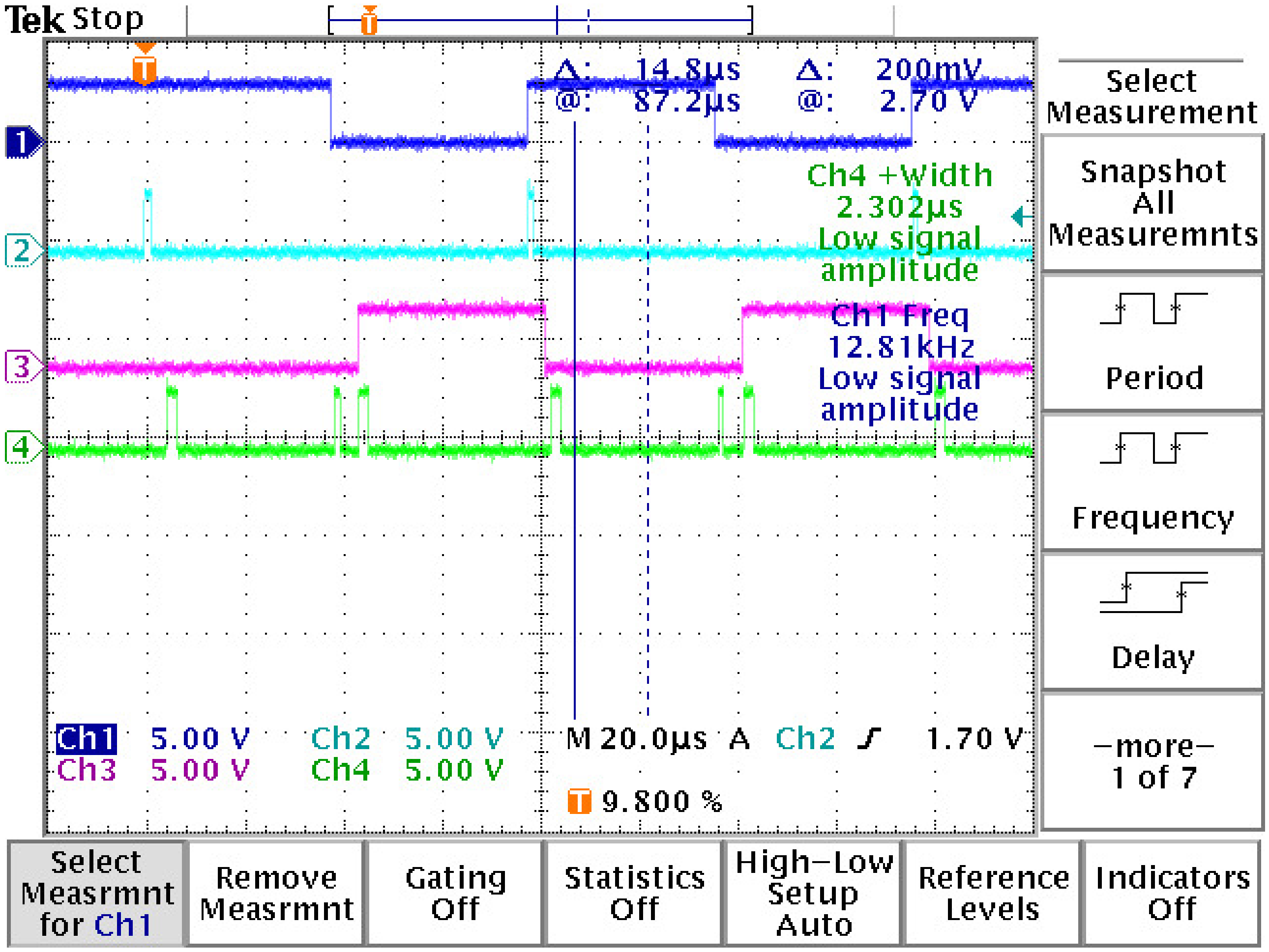
I have removed a lot of  application code from this example, but there might still be som bits and pieces of it.

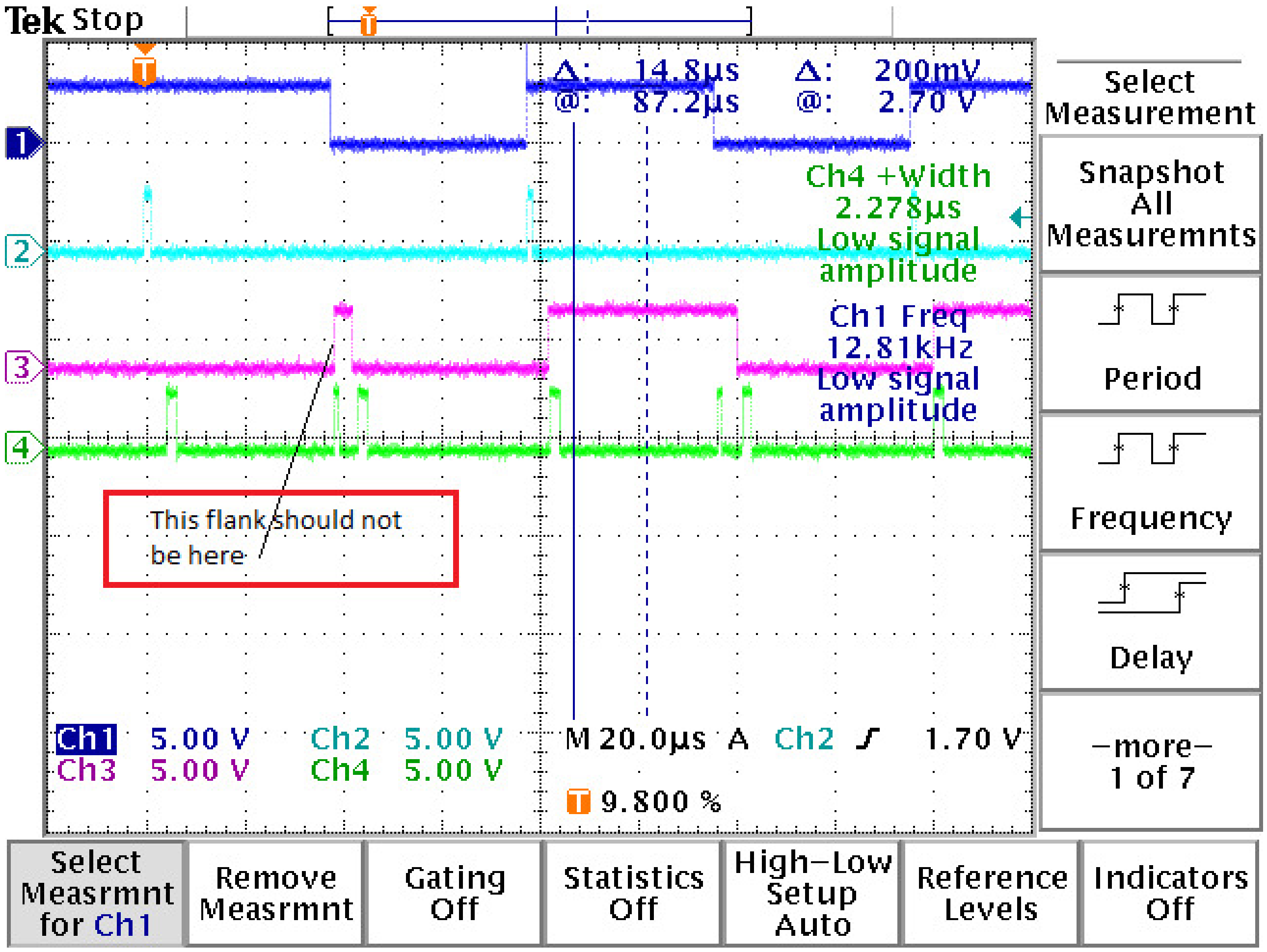
I have compiled and run the code on atollic 3.2. on a freshly started project. The code then runs on a STM32F4Discovery board that i have removed some resistors and caps on that ensures that the pinlists on the board are not affected by the USB OTG connector and chip.

I then connect the following pins to the following channels on my scope.

Blue: PA8 ( channel output)
Turquoise: PA5 (GPIO output for simple trace
Purple: PB14 ( channel output)
Green: PB7 (gpio again)

When i run this throu the debugger I will get two diffrent results
In the screenshots, i have triggered on the first positive flank on the blue channel.





This flank should not be here

The flank that i have marked as "Wrong" appears to me to be a toggle that has occured during the update event and not during the CC2.

If I use the reset button to trigger the boot of the system I never get the small pulse on the purple channel.

EDIT: better scope channel descriptions

Best regards
Martin