La Salle Campus Barcelona

Equips Perifèrics

Mesurador del Temps de Resposta

Pràctica 1

Samuel Tavares da Silva (ls30759)

Manel Manchón Gascó (ls31343)

19 de març de 2018

Índex

1 Descripció de l'arquitectura del programa

L'arquitectura plantejada per a la resolució d'aquesta pràctica és molt senzilla. Tot el programa està en un sol fitxer .c. Aquesta arquitectura, per tenir un referent, pot tenir algunes similituds amb una arquitectura dirigida per esdeveniments. Un esdeveniment pot ser definit com ün canvi significatiu en el nostre estat". Per explicar com hem dissenyat la pràctica ens recolzarem sobre aquesta arquitectura.

En el nostre cas, tenim un esdeveniment principal: l'usuari prement el polsador blau. Això és donarà quan un esdeveniment de menys importància, com s'encendre un LED (més concretament el LED3) en un interval entre 1 i 5 segons succeeixi. Abans de continuar, vegem com hem implementat i gestionat aquests dos esdeveniments que corresponen al punt 1 de la pràctica.

La primera funció que és cridada en el nostre programa, com qualsevol programa, és la funció main. Aquesta funció té un seguit de declaracions de variables necessàries per obtenir un seguit de valors. Un cop acabada la declaració comencem primer amb les configuracions que necessita la nostra placa per funcionar del mode desitjat. La primera funció l'utilitzem per a configurar tots els pins de mode output que utilitzarem al llarg

```
305
306         INIT_OutputPins(gpio);
307         INIT_Button(gpio);
308
```

Funcions que configuren la nostra placa STM32F4

de la nostra pràctica. La segona funció, com molt bé indica el seu nom s'usa per detectar quan l'usuari prem el botó. En el següent apartat d'aquest document entrarem més en com s'han implementat i veurem la seva configuració.

Un cop realitzada la configuració del ports d'output - més endavant veurem els ports configurats com a input i explicarem perquè ho fem més tard - ara generarem el valor ràndom de 32bits el qual utilitzarem per encendre el LED3 en un moment donat entre 1 i 5 segons. Llavors en el main cridem un seguit d'instruccions les quals podem veure en la figura 2.

La primera línia d'aquesta porció de codi es la inicialització del ràndom. Aquesta configuració és necessària per generar un valor ràndom de 32 bits. L'explicació de les configuracions necessàries es troben en l'apartat de mòduls, tal i com hem indicat amb els pins i el polsador.

Per treure els valors col·locats dintre del *while* vam realitzar una comprovació i vam mirar quan era el temps corresponent que ens donava la funció *Delay* amb un counter d'1 proporcionada pel projecte base.

```
36  void delay(int counter)
37  {
38          int i;
39          for (i = 0; i < counter * 10000; i++) {}
40  }
```

Funció utilitzada per esperar-nos un valor entre 1 i 5 segons.

Un cop vist el valor d'aquest vam fer una regla de tres per obtenir els valors necessaris per 1 i 5 segons. Un segon correspon al valor 334 i 5 segona al 1670. Com que la variable *rand_rled* està inicialitzada a 0 la primera iteració entrarem dintre del while. Un cop dins, utilitzam la variable auxiliar *arand_tled* per agafar el valor ràndom de 32 bits. Aquest valor creiem que el bit de més pes és de peritat, ja que donava valors negatius. Per això la necessitat d'utilitzar una variable auxiliar. La funció per obtenir aquest valor ràndom de 32 bit té la següent implementació:

```
51  uint32_t TM_RNG_Get(void) {
52          /* Wait until one RNG number is ready */
53          while (!(RNG->SR & (RNG_SR_DRDY)));
54
55          /* Get a 32-bit Random number */
56          return RNG->DR;
57  }
```

Funció que genera un valor ràndom de 32bits.

Com podem veure pels mateixos comentaris de la funció, aquesta espera fins que el nombre està llest i quan el té llavors agafa el seu valor ràndom. Tot seguit es fa una crida a la funció *map* per ajustar aquest valor entre els desitjats. La variable rand és l'obtinguda per la funció anterior, la variable min és l'interval mínim

```
27  /*
28   * Function for scale a number to the desired range.
29   */
30  int map(int rand, int min, int max)
31  {
32    return (rand % (max + 1 - min)) + min;
33  }
```

Funció la qual ajusta el valor entre els desitjats.

que volem i finalment, la variable max el límit gran de l'interval. Un cop sortim del while i tenim el valor ràndom dintre dels valors desitjats llavors, fem un reset el pin 13 del port GPIOG (el qual correspon al LED3), esperem els segons equivalents al valor ràndom i fem un set del pin 13. La següent imatge és una representació de com quedaria totes les funciona nomenades fins ara.

```
309         //Inicializamos el Random
310         TM_RNG_Init();
311
312         //Generamos valores random y mapeamos a los valores que cosnideramos válidos
313         //para poder hacer un delay de entre 1s y 5s
314         while(rand_tled<=334 || rand_tled >=1670){
315                 arand_tled = TM_RNG_Get();
316                 if(arand_tled < 0 )arand_tled *= -1;
317                 rand_tled = map(arand_tled,334,1670);
318         }
319         GPIO_ResetBits(GPIOG, GPIO_Pin_13);
320         delay(rand_tled);
321         GPIO_SetBits(GPIOG, GPIO_Pin_13);
---
```

Seguit de crides que és fan per generar un ràndom d'entre 1 i 5 segons.

Un cop vist això ja sabem com s'ha implementat l'esdeveniment d'encendre el LED3 entre 1 i 5 segons. L'esdeveniment principal, l'usuari prem el polsador, el detectem de la següent forma: En aquesta captura

```
323         TIM2_INT_Init();
324
325         while(!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)) {
326
327         }
328         timer_value= timer_value/100; //convertimos a ms
329
330         tm4_periode = (timer_value * 288)/100;//escalamos el periodo necesario para el pwm en ms
331
332         GPIO_ResetBits(GPIOG, GPIO_Pin_13);
333         GPIO_ResetBits(GPIOG, GPIO_Pin_5);
334
335         dutyCycle = (double)(tm4_periode*50.0)/(timer_value);
```

Part del main per detectar que l'usuari a premut el polsador i realització de càlculs pertinents.

inicialitzem el timer 2, el qual anirà a 10 micros. Aquest timer l'utilitzarem per contar quant de temps passa des que s'encén el LED3 fins que l'usuari prem el polsador. Tenim un marge d'error, ja que des que s'encén el LED3 fins que iniciem el comptatge de temps passen 2 instruccions. Aquest marge de temps és menyspreable perquè el temps d'instrucció és de l'ordre de nanosegons i no ens afectarà.

El principal problema que tenim amb el polsador són els rebots. Al principi el valor que ens donava del temps que l'usuari havia trigat era molt petit. Això es deia a què teníem el bucle dins del while(1) i provocava que agafés valors dels rebots. Llavors, quant a sortim de la condició de llegir si l'input del polsador és a 1, encara que hi hagi rebots, haurem agafat el valor "boï tindrem el temps que ha trigat l'usuari en preme el polsador.

Els càlculs que es veuen a la captura són els següents:

- **Pas de microsegons a nanosegons:** Per passar a mili-segons dividim entre 100 i no 1000, ja que estem contant quants cops fem 10 microsegons

  $Tms = N^{o}vegades 10micros/100$

- **Període PWM:** Com ja veurem en la configuració, en ficar un preescaler de 9, el període que configurarem serà de 288 que equivaldrà a 100 microsegons que és el mínim que pot trigar una persona. Es una simple regla de tres que hem tret a partir de l'enunciat.

  $Periode = (Tms * 288)/100$

- **DutyCycle PWM:** Aquest representa el temps que ha d'estar 1 respecte a 0 del PWM. Segon l'enunciat el temps a 1 ha de ser sempre 50 microsegons. El càlcul que farem equivaldrà al % que necessitem perquè sempre siguin 50 microsegons en un període variable.

  $DutyCycle = (double)(Periode * 50.0)/Tms$

Per acabar, només comentar que aquí fem un reset dels pins corresponents al LED3 (Pin 13) i el pin que rebrà a la interrupció del PWM (Pin 5), el dos del port GPIOG.

En aquest moment ja hem explicat com hem implementat la gestió dels esdeveniments mencionats a l'inici d'aquest document. Només queda explicar un esdeveniment el qual serà l'últim dels tres esdeveniments principals. Aquest fa referencia a la generació del PWM i implementació d'una RSI la qual detecti aquest com una interrupció. Primer de tot vegem el codi restant del main el qual correspon a les dues implementacions:

```
337         //Rutinas de inicializacion del PWM
338         TM_LEDS_PWM_Init();
339
340         //Configuramos la interrupcion externa
341         EXTI_Pwm();
342         //Inicializamos la interrupcion para el PWM
343         TIM4_INT_Init(tm4_periode);
344         TM_PWM_Init(dutyCycle);
345
346         while (1)
347         {
348                 GPIO_ResetBits(GPIOG, GPIO_Pin_5);
349         }
```

Part del main que genera PWM.

Les dues primeres funcions que es criden són: *TM_LEDS_PWM_Init*, la qual serveix per inicialitzar el pin per on treure'm el PWM (aquest pin té una configuració especial per anar amb el timer explicada al següent apartat), i *EXTI_PWM* la qual ens configurarà un segon pin que rebrà una interrupció sempre que es generi un PWM.

Un cop tinguem la interrupció externa configurada (per satisfer l'apartat 3) podem passar el període calculat anteriorment per configurar el timer 4. La configuració d'aquest timer és igual que la de tots els altres a diferència que el seu període dependrà del temps que hagi trigat l'usuari en prema el botó.

Finalment, com el PWM funciona sobre el timer 4 ja configurat anteriorment, inicialitzem la funció que generarà el PWM indicant-li quin ha de ser el DutyCycle. Aquesta funció no està explicada en l'apartat següent, ja que no considerem que no sigui part de cap módul.

```
244  void TM_PWM_Init(int dutycycle) {
245
246          TIM_OCInitTypeDef TIM_OCStruct;
247
248
249          TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
250          TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
251          TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
252          TIM_OCStruct.TIM_Pulse = dutycycle;
253          TIM_OC1Init(TIM4, &TIM_OCStruct);
254          TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);
255  }
```

Funció per generar el PWM.

Finalment, per acabar resetegem els pin 5 dintre del while(1) per detectar quan sortim de la interrupció.

2 Descripció dels mòduls

Pel correcte desenvolupament de la pràctica ens hem ajudat d'una sèrie de mòduls proporcionats per la placa STM32F.

Alguns d'aquests mòduls són:

- RNG (Random Number Generator)
- Ports i Pins I/O
- Timers
- Interrupcions

2.1 RNG (Random Number Generator)

Mòdul utilitzat per obtenir un nombre aleatori, que ens permeti engegar el LED3 en un temps d'entre 1 i 5 segons. En el nostre cas obtenim un valor i l'escalem al valor necessari per realitzar una espera vàlida de l'1 a 5 s.

Sent un perifèric encarregat de generar un número de 32 bits totalment aleatori. Aquest perifèric es troba en les majories de sistemes STM32F. El principal avantatge és que genera un número realment aleatori independent de qualsevol altre timer de l'usuari.

En el nostre cas, com que no necessitem un número de 32 bits hem realitzat un map, abreviació de mapping. Aquesta funció sol ser molt típica en sistemes Arduino, el que fa és a partir d'un valor mínim i un màxim, retorna un valor en aquest rang desitjat.

2.2 Ports i Pins I/O

Sent aquest mòdul essencial pel funcionament de les entrades i sortides del sistema, la correcta configuració dels pins és fonamental pel funcionament general de la pràctica.

Per realitzar la configuració d'un pin s'ha d'omplir un registre, mitjançant el qual s'indica el pin a configurar i el mode en què es desitja que funcioni el pin, finalment es passa aquesta estructura a una funció indicant a quin port fa referència. Destacar que l'estructura de configuració de ports consta de més camps, els quals s'utilitzen en funció de l'ús que es vulgui a donar al pin en concret.

En el nostre cas hem configurat per un costat els inputs i els outputs. En el nostre cas els inputs serien, els pins on tenim el polsador i la interrupció externa on introduïm el PWM. Respecte a els outputs tenim el pwm, senyal de latència de la RSI i els LEDs.

Per un costat ens agradaria destacar que els inputs tenen una configuració diferent, no sols canviant el mode del pin, sinó que pel polsador, s'ha d'indicar la finestra de rebots i configurar el polsador com push-pull. Referent al pin amb la interrupció externa, a part de la configuració estàndard, s'ha d'afegir la configuració d'un registre per la interrupció i un altre per la interrupció externa, de manera que es pugui definir el tipus d'interrupció que es vol realitzar, tal com es pot mostrar en la següent imatge:

```c
/* Configure pins to be interrupts */
void EXTI_Pwm(void) {
    /* Set variables used */
    GPIO_InitTypeDef GPIO_InitStruct;
    EXTI_InitTypeDef EXTI_InitStruct;
    NVIC_InitTypeDef NVIC_InitStruct;

    /* Enable clock for GPIOD */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    /* Enable clock for SYSCFG */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

    /* Set pin as input */
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOD, &GPIO_InitStruct);

    /* Tell system that you will use PD0 for EXTI_Line0 */
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOD, EXTI_PinSource0);

    /* PD0 is connected to EXTI_Line0 */
    EXTI_InitStruct.EXTI_Line = EXTI_Line0;
    /* Enable interrupt */
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    /* Interrupt mode */
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    /* Triggers on rising and falling edge */
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
    /* Add to EXTI */
    EXTI_Init(&EXTI_InitStruct);

    /* Add IRQ vector to NVIC */
    /* PD0 is connected to EXTI_Line0, which has EXTI0_IRQn vector */
    NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
    /* Set priority */
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
    /* Set sub priority */
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    /* Enable interrupt */
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    /* Add to NVIC */
    NVIC_Init(&NVIC_InitStruct);
}
```

Respecte els outputs, la configuració és més senzilla, ja que pels LEDs els pins només s'indica el pin, port i mode output. Però per un altre costat tenim la configuració del PWM, ja que es configura vinculant-lo a un timer que realitzarà un update del pin, sense cap manipulació.
En la següent imatge podem observar la configuració general dels pins de LEDs:

```c
void INIT_OutputPins(GPIO_InitTypeDef gpio){
        //RCC -> Reset and Clock Control
        // All GPIO's are on AHB1 bus.
        //GPIOG port porque los LEDs estan conectados a los pins PG13 y PG14 (supongo que estructura interna)
        //The function below enables the clock
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);
        GPIO_StructInit(&gpio); //supongo inicializa la structura

        /*
         *  GPIO_Pin: Choose pins you will use for set settings
            GPIO_Mode: Choose mode of operation. Look down for options
            GPIO_OType: Choose output type
            GPIO_PuPd: Choose pull resistor
            GPIO_Speed: Choose pin speed
         *
         */
        gpio.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_5;
        gpio.GPIO_Mode = GPIO_Mode_OUT;
        //una vez digamos que hemos indicado que "usaremos" en las linias anteriores
        GPIO_Init(GPIOG, &gpio);
}
```

2.3 Timers

Fonamentals pel correcte funcionament del sistema, ens permetran contar el temps que l'usuari té de reacció i realitzar el senyal de PWM. En el nostre cas hem utilitzat dos timers, un per calcular el temps de reacció i un altre per generar el senyal de PWM.

Dins de tots els timers que tenim disponibles, hem utilitzat el timer 2 pel càlcul de temps de reacció i per facilitar el càlcul del període del timer del PWM.

Referent a les configuracions d'aquest, hem introduït el període i preescaler necessari per poder obtenir un temps de 100us, hem configurat la interrupció de temps del timer i hem inicialitzat la variable on contem el temps que transcorre.

En aquest timer cal destacar que capturem la interrupció per poder incrementar el temps transcorregut en una variable global.

Per un altre costat tenim el timer 4, encarregat de generar el senyal PWM. Destacar que hem escollit aquest timer, ja que permet generar senyals PWM de manera més senzilla, tal com es pot observar en aquesta imatge:

```
258 void TIM4_INT_Init(int periode)
259 {
260     // Enable clock for TIM
261     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
262 
263     TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
264     TIM_TimeBaseInitStruct.TIM_Prescaler = 9;
265     TIM_TimeBaseInitStruct.TIM_Period = periode;
266     TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
267     TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
268 
269     // TIM4 initialize
270     TIM_TimeBaseInit(TIM4, &TIM_TimeBaseInitStruct);
271     // Enable TIM4 interrupt
272     TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
273     // Start TIM4
274     TIM_Cmd(TIM4, ENABLE);
275     //timer_value = 0;
276 
277     NVIC_InitTypeDef NVIC_InitStruct;
278     NVIC_InitStruct.NVIC_IRQChannel = TIM4_IRQn;
279     NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
280     NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
281     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
282 
283     timer_value = 0;
284 
285     NVIC_Init(&NVIC_InitStruct);
286     GPIO_ResetBits(GPIOG, GPIO_Pin_14);
287 }
```

Respecte les configuracions d'aquest timer, només destacar que el període es calcula en funció del temps obtingut del timer 2, ja que el període d'aquest ha de ser el contingut de la variable global gestionada amb la interrupció del timer 2. En el moment que el timer 4 està configurat, passem a la configuració del PWM, que requereix el dutycycle necessari per realitzar el temps a 1 de 50us. Finalment passem a la configuració del pin per on sortirà el PWM, indicant que el timer4 actualitzarà el valor del pin.

2.4 Interrupcions

Aquest mòdul és utilitzat per diversos mòduls, directament o indirectament se solen utilitzar les interrupcions, ja que és l'única manera de detectar esdeveniments del sistema. En aquesta pràctica podem separar les interrupcions en externes i internes. Les internes serien les interrupcions del timer o el polsador i les externes fan referencies a les que vénen de fora com és el cas del pin on posem el pwm.

En general les interrupcions es configuren amb el registre NVIC, Nested Vector Interrupt Controller, registre que configura els paràmetres necessaris perquè el sistema capturi la interrupció. La part fonamental de la configuració del NVIC és quan li indiquem el canal d'interrupció que s'utilitzarà, ja que es vincularà el corresponent timer amb la interrupció.

Per cada interrupció configurem un *IRQ_handler()*, realitzem un handler per cada interrupció, de manera que cada cop que salti una interrupció puguem realitzar una tasca concreta. En el nostre cas això ens permet incrementar una variable cada cop que el timer dongui un període, veure el temps que triga la RSI, entre altres.

Com a exemple de handler, en aquesta imatge es pot observar el handler de la interrupció externa.

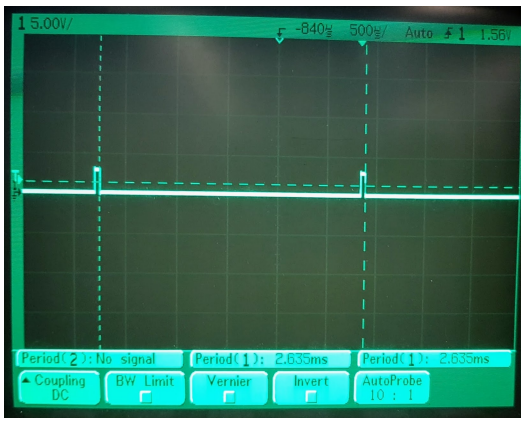
```
180 /* Set interrupt handlers */
181 /* Handle PD0 interrupt */
182 void EXTI0_IRQHandler(void) {
183     /* Make sure that interrupt flag is set */
184     if (EXTI_GetITStatus(EXTI_Line0) != RESET) {
185         EXTI_ClearITPendingBit(EXTI_Line0);
186         GPIO_SetBits(GPIOG, GPIO_Pin_5);
187         if(tm4_periode > MAX_PERIODE) {
188             GPIO_SetBits(GPIOG, GPIO_Pin_14);
189             GPIO_ResetBits(GPIOG, GPIO_Pin_13);
190         }else{
191             GPIO_SetBits(GPIOG, GPIO_Pin_13);
192         }
193
194     }
195 }
196
```

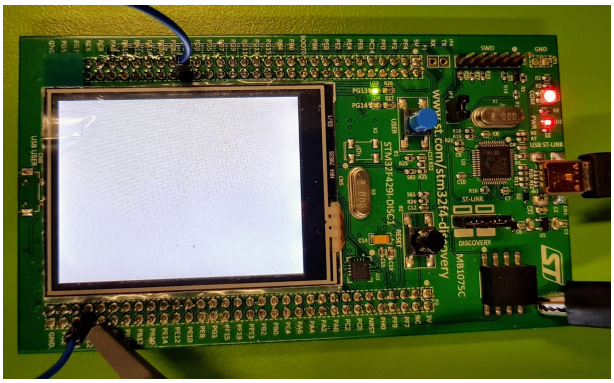
3 Captures de pantalla

3.1 Comprovació Leds

Temps mig

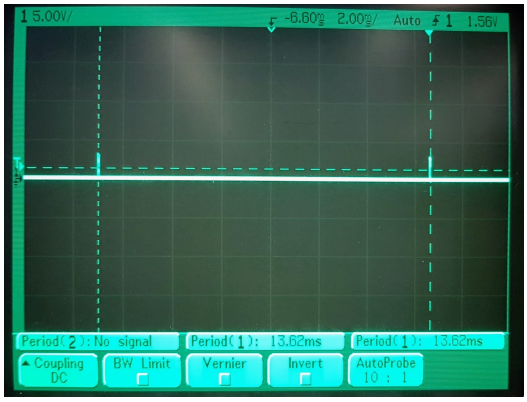
Realitzant un temps normal, comprès en la mitja de reacció definit per l'enunciat, podem observar un període de 2.63ms. Corresponentment per indicar que és un bon temps de reacció es manté el LED2 (LED VERD) engegat.

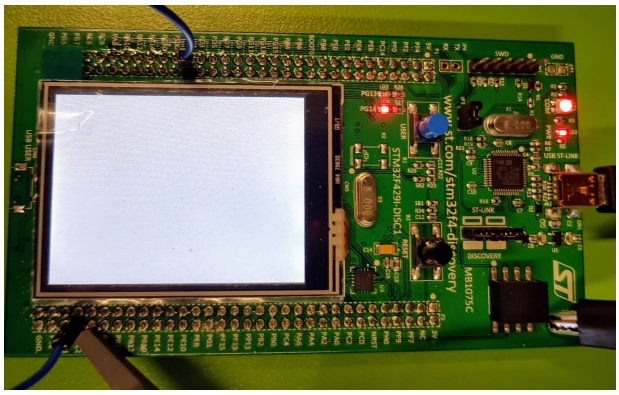




Temps superior a la mitja

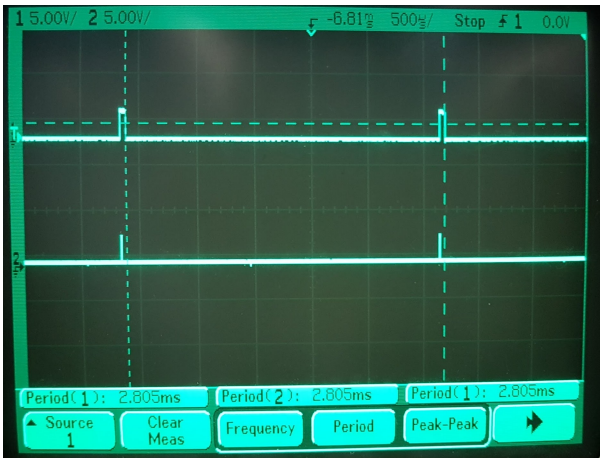
En aquestes dues imatges podem observar com actua el sistema en cas de realitzar un temps de reacció superior a 10 s. Per un costat obtenim un període de 13.62ms i per un altre costat engeguem el LED3, indicant que s'ha superat el temps màxim.





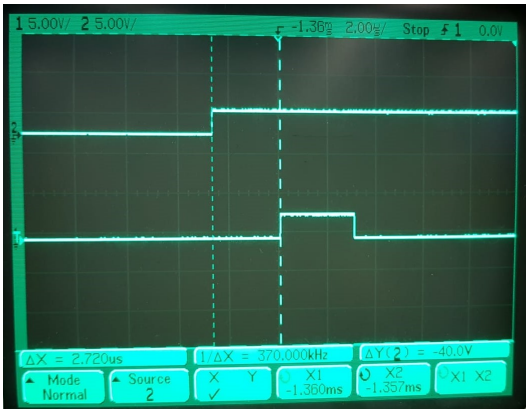
Relació PWM i RSI

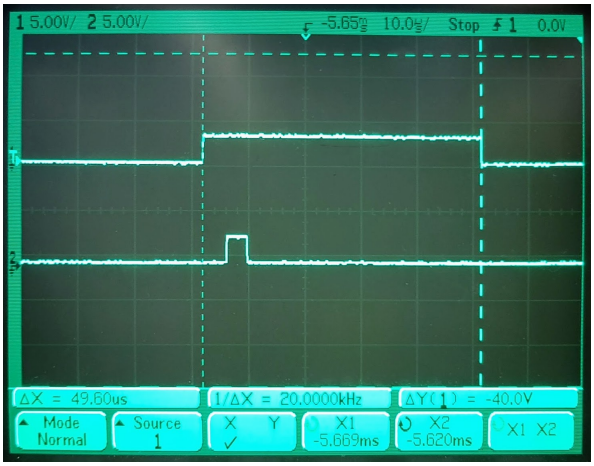
En aquesta imatge de l'osciloscopi, es pot observar dos senyals. En el canal 1, canal superior, es pot observar el senyal de PWM amb un període variable i 50us a 1. En el segon canal, l'inferior, es pot veure el senyal de la RSI i com el període coincideix.



Temps de latència de la RSI i temps a 1 del PWM

Utilitzant els dos canals de l'oscil·loscopi i els cursors, podem comprovar el temps que passa entre els dos senyals. Aquest temps és el que correspon amb la latència de la RSI.





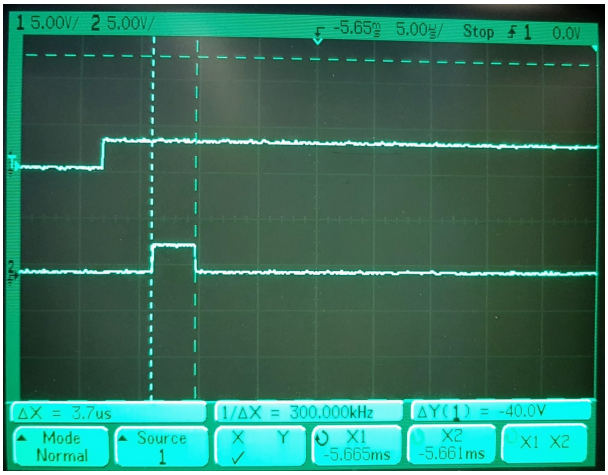
4 Resultats de les mesures sol·licitades

El nombre d'interrupcions és igual al nombre de peticions de servei (cada petició genera una interrupció). Sabent això i que el band width de la CPU és calcula de la següent forma: $BWint = Cavg * t3$
C average correspon a la mitjana del nombre de peticions de servei que caldrà sincronitzar per unitat de temps. Y t3 serà la fracció de temps de CPU dedicat a la sincronització del dispositiu per interrupcions. Llavors pels diferents casos que ens planteja la pràctica tindrem:

$BWint = Cavg * t3 = 1/100us * 3.7us = 3.7\%$

$BWint = Cavg * t3 = 1/250us * 3.7us = 1.48\%$

$BWint = Cavg * t3 = 1/10ms * 3.7us = 0.037\%$



Captura del temps que triga RSI.

5 Conclusions

En general considerem que la relació de teoria respecte la practica és adequada. Ja que el gran gruix de la teoria l'hem vist a la pràctica, ens hi ha permet visualitzar millor i provar pels nostres propis medis conceptes vistos en els apunts de teoria. Destacar que el fet que l'enunciat estigui explicat per punts, ens ha ajudat durant la implementació de la pràctica.

Inicialment el problema principal que hem tingut ha sigut endinsar-se en les profunditats de la documentació oficial de STM32F4. Ja que la placa compta amb tants perifèrics, registres i diferents configuracions, ens ha suposat un petit repte, ja que no estem acostumats a treballar amb aquest tipus de documentació. Però cal destacar, que existeix una gran comunitat al voltant de STM32F4, facilitant molt l'accés a solució d'errors i exemples de funcionament o configuracions.

Finalment considerem que aquesta pràctica ens ha ajudat a solidificar conceptes vistos a teoria i ens ha servit per veure clar configuració, control i diversos tipus d'interrupcions, generar senyals PWM amb un Duty Cycle variable, configuració dels timers, entre altres. També hem pogut observar de primera mà com s'han d'anar agrupant els diferents mòduls per poder crear una bona pràctica.

Un exemple d'això és entendre com amb un handler d'interrupcions la programació és més senzilla. Això és gracies a que si no fem pulling, podem anar fent altres coses. Aquesta técnica l'utilitzem pels timers i de forma oposada amb el botó si que vam fer pulling, ja que ens insteressava.

Destacar que també hem vist maneres d'interactuar amb la placa com utilitzar un polsador o configurar els pins I/O, per veure els senyals que generem o engegar LEDs. Per tant creiem que és una pràctica curta amb què podem posar en practica un munt de conceptes i perifèrics que inclou la placa, de manera que podem tenir una perspectiva general, realitzant una tasca concreta per una finalitat concreta.