Sergio Mancha Cerezuela i Ferran Granell Jurado - G12

Sistemes Encastats. Pràctica 4.
Interrupcions i mesures de temporització. Estudi de laboratori.

L'objectiu d'aquest estudi de laboratori ha estat aprender a configurar interrupcions a través de linies del port GPIO, així com aprendre a utilitzar l'analitzador lògic del laboratori per a fer mesures al microcontrolador.

1. Configuració i control d'interrupcions.

En aquesta secció, hem configurat les interrupcions associades a un port GPIO, per a poder controlar una subrutina que executi lectures a l'acceleròmetre per a analitzar la seva temporització.

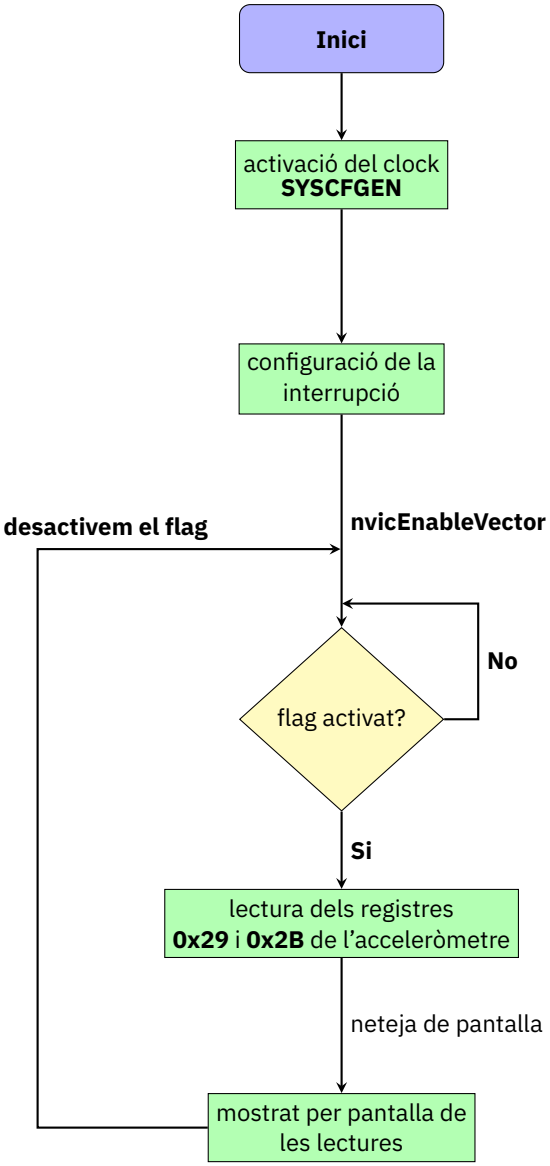

Figura 1: Diagrama de flux corresponent a la implementació d'un mètode per a relitzar crides arbitràries a lectures a l'acceleròmetre per a mesurar els seus paràmetres de temporització, mitjançant interrupcions al port GPIO.

El mètode implementat ha estat *void interruptTest(void)*, el qual no retorna ni rep cap variable com a paràmetre. El diagrama de flux de la *figura 1* ve a il·lustrar la lògica implementada.

```
volatile int switchFlag;

void interruptTest(void) {
    char *string[5]; // Per a fer servir itoa

    // Activació del clock
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Configuració de EXTI
    SYSCFG->EXTICR[0] = (SYSCFG->EXTICR[0] & (~SYSCFG_EXTICR1_EXTI0)) | SYSCFG_EXTICR1_EXTI0_PA;
    EXTI->IMR |= EXTI_IMR_MR0;
    EXTI->RTSR |= EXTI_RTSR_TR0; // Nosaltres el configurem com a flanc de pujada
    EXTI->FTSR &= ~EXTI_FTSR_TR0; // Desactivem la configuració de flan de baixada
    EXTI->PR = EXTI_PR_PR0; // Netejem PR

    // Activació de NVIC
    nvicEnableVector(EXTI0_IRQn,CORTEX_PRIORITY_MASK(STM32_EXT_EXTI0_IRQ_PRIORITY));


    while(1) {
      switchFlag = 0; // Netejem el flag.

      while(switchFlag == 0); // Esperem a que s'activi.

      LCD_ClearDisplay(); // Netejem la pantalla.

      // Llegim l'acceleròmetre i escrivim el valor de l'eix x.
      LCD_SendString("Eix x:");
      LCD_SendString(itoa(readAccel(0x29, 1), string, 10));

      LCD_GotoXY(0, 1); // Apuntem a l'inici de la segona fila amb el cursor.

      // Llegim l'acceleròmetre i escrivim el valor de l'eix y.
      LCD_SendString("Eix y:");
      LCD_SendString(itoa(readAccel(0x2B, 1), string, 10));
    }
}
```

En primer lloc, hem configurat les interrupcions que farem servir. Abans de sel·leccionar com volem que es comportin les interrupcions, hem d'assignar el port GPIO que farà de *trigger* d'aquesta. Per a això, fem servir el perifèric *System Configuration Controller*, que ens permetrà sel·leccionar el port d'on penja el botó USR de la placa de laboratori per a accionar les interrupcions. Abans de cap res, com aquest perifèric penja del bus APB2, al igual que en el cas de l'acceleròmetre amb el perifèric SPI, hem d'activar el seu rellotge accedint al registre que controla totes les habilitacions de rellotge d'aquest bus, *RCC_APB2ENR*, activant el bit **RCC_APB2ENR_SYSCFGEN**, constant corresponent al bit en la posició del mapa del registre per al rellotge del perifèric SYSCFG definida a *stm32f4xx.h*.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVAT															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figura 2: Diagrama del mapa del registre *SYSCFG_EXTICR1*, a partir del que es troba a les especificacions del microcontrolador.

La sel·lecció de les linies que poden accinar una interrupció ve agrupada en 16 blocs, conformats cadascún per totes les linies tipus *PAX...PIX*. Aquests blocs estan agrupats mitjançant multiplexors, pel que només

podrem utilitzar simultàniament un sol tipus de linia PAX...PIX. Per això, el perifèric que estem utilitzant per a la configuració de les linies que ens permetran accionar interrupcions disposa de quatre registres per a especificar quina linia de cada multiplexor és la que voldrem configurar per activar una interrupció. Aquests registres van enumerats del *SYSCFG_EXTICR1* fins al *SYSCFG_EXTI4*, i contenen dins dels seus mapes de registre quatre d'aquests blocs multiplexors a configurar. La *figura 2* es correspon amb el mapa del registre *SYSCFG_EXTICR1*, que serà el que haurem de configurar en el nostre cas en particular.

De manera consecutiva, els quatre bits associats al mapa del registre per *EXTI0* corresponen al multiplexor que sel·lecciona les linies tipus PY0, amb *EXTI1* sel·leccionem les linies tipus PY1, etc

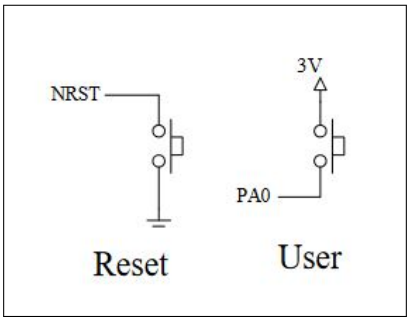

Figura 3: Connexio dels botons USER i RESET, que es troben a la placa de laboratori, amb les linies GPIO del microcontrolador STM32.

A la *figura 3* podem veure que el botó que pretenem utilitzar per a accionar les interrupcions, el botó USER, està connectat a la linia *PA0*, per tant configurem els bits de *EXTI0*, del registre *SYSCFG_EXTICR1*, a "0000". Per a flexibilitzar el nostre codi, hem fet servir les constants de l'arxiu de capçalera *stm32f4xx.h* que corresponen a **SYSCFG_EXTICR1_EXTI0**, constant 0xFF en la posició de *EXTI0*, i **SYSCFG_EXTICR1_EXTI0_PA**, constant amb la cambinació de quatre bits associada al port PA dins de *EXTI0*. Primer netejem el contigut de *EXTI0* aplicant una máscara tipus *and* bit a bit, de la constant negada **SYSCFG_EXTICR1_EXTI0**, que ens posarà els quatre bits a zero. Posteriorment, simplement aplicant una màscara tipus *or* bit a bit amb el contigut del registre i la constant **SYSCFG_EXTICR1_EXTI0_PA**, encenem els bits que hagin d'estar a '1' per a replicar el valor de la constant als quatre bits de *EXTI0*.

Ara ja estem en condicions de configurar com volem que siguin aquestes interrupcions. Per això, hem configurat el *External Interrupt Controller* (EXTI). El primer registre que hem configurat ha estat el *EXTI_IMR* ò *interrupt mask registre*, que té associat el mapa de registre de la *figura 4*, indica si les interrupcions estan habilitades o emmascarades per a cada multiplexor configurat.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVAT									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figura 4: Diagrama del mapa del registre *EXTI_IMPR*, a partir del que es troba al document d'especificacions del microcontrolador.

Els 16 LSB del registre es corresponen amb l'habilitació d'interrupció dels multiplexors amb les linies GPIO, com el que hem configurat abans. Com únicament volem habilitar l'interrupció per al primer multiplexor, que es correspon amb la linia *PA0*, i atenent que el valor predeterminat després de reseteig és *0x0000 0000*, aplicant una màscara tipus *or* bit a bit amb la constant **EXTI_IMR_MR0**, que es troba a *stm32f4xx.h* i correspon a la posició del *MR0* al mapa del registre, podem encendre la posició associada al nostre multiplexor.

A continuació, hem configurat l'activació de la interrupció pel que respecta a la configuració del clock. Els registres *rising trigger selection* (*EXTI_RTSR*) i *falling trigger selection* (*EXTI_FTSR*), ens configurar l'accionament per flanc de pujada o baixada.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVAT									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figura 5: Diagrama del mapa dels registres *EXTI_RTSR* i *EXTI_FTSR*, a partir del que es troba al document d'especificacions del microcontrolador.

La *figura 5* correspon als mapes d'ambdós registres, que són iguals. Posant un '1' o un '0' als 16 LSB, de la mateix manera que en l'anterior registre, podem activar o desactivar la configuració específica de cada registre. Per això, com en el nostre cas busquem una configuració d'activació per flanc de pujada, i com el fabricant ens assegura que després del reset de manera predeterminada el registre es troba a *0x0000 0000*, hem aplicat una màscara tipus *or* bit a bit amb la constant de *stm32f4xx.h EXTI_RTSR_TR0*, que es correspon a un bit en la posició de *TR0* al registre, al registre *EXTI_RTSR* per a activar el flanc de pujada. Per tant, tot i que no caldria, ens hem assegurat de desactivar la configuració per flanc de baixada aplicant una màscara tipus *and* bit a bit amb la constant negada **EXTI_FTSR_TR0**, que es correspon a un bit en la posició de *TR0* al registre, per a desactivar la configuració d'accionament com a flanc de baixada a *TR0* del registre *EXTI_FTSR*.

Finalment, hem netejat el registre *Pending Register*, l'encarregat d'indicar si s'ha generat petició d'interrupció o no en algún dels multiplexors habilitats. La *figura 6* es correspon al mapa de registre associat, on els 16 LSB són els corresponents a les peticions d'interrupció de cada multiplexor.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVAT									PR22	PR21	PR20	PR19	PR18	PR17	PR16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Figura 6: Diagrama del mapa del registre *EXTI_PR*, a partir del que es troba al document d'especificacions del microcontrolador.

Aquest registre té un comportament diferent en escriptura. Per a netejar els bits que es vulguin, simplement igualem el registre a una seqüència de bits que contingui '1's en les posicions que pretenem netejar. D'aquesta manera, igualant el registre a la constant **EXTI_PR_PR0**, que es correspon a un bit alt '1' en la posició de *PR0* que es correspon en el mapa del registre *EXTI_PR*.

Per acabar, hem de programar la interrupció als registres interns del *Nested vectored interrupt controller* (*NVIC*), l'element que rep i gestiona les interrupcions en funció de les seves prioritats.

nvicEnableVector(Linia, CORTEX_PRIORITY_MASK(Prioritat));

La *linia*, en aquest cas, es correspon a la constant **EXTI0_IRQn**, que s'associa amb el *EXTI GPIO 0*. La prioritat l'hem programat amb la constant **STM32_EXT_EXTI0_IRQ_PRIORITY**. Aquestes constants estan declarades a l'arxiu de capçalera *core_cm4.h*.

En un bucle tipus *while*, que s'executarà infinitamento bé fins que s'elimini l'alimentació del microcontrolador, hem implementat les crides a l'acceleròmetre. Per això, fora del mètode hem inicialitzar una variable tipus *volatile int* que es correspondrà amb el senyal de *flag* que indicarà si s'ha produit una interrupció. Per tant, quan s'activi el flag es de llegiran els registres de l'acceleròmetre amb adreçes *0x29*, acceleració en l'eix X, i *0x2B*, acceleració en l'eix Y, per posteriorment netejar el LCD i mostrar els valors per pantalla fent servir la funció, implementada al fitxer de capçalera *lcd.h*, *void LCD_SendString(char *)*. Finalment, tornem a l'inici del bucle, netejem el valor del flag i esperem a que es torni a activar per a repetir la lectura i mostrat per pantalla.

Per a definir la funció associada a un vector d'interrupció hem utilitzat la macro *CH_IRQ_HANDLER(interrupció)*, on *interrupció* es correspon a al vector d'interrupció que cridem mitjançant el NVIC, per tant en el nostre cas fent servir la constant *EXTI0_IRQHandler*.

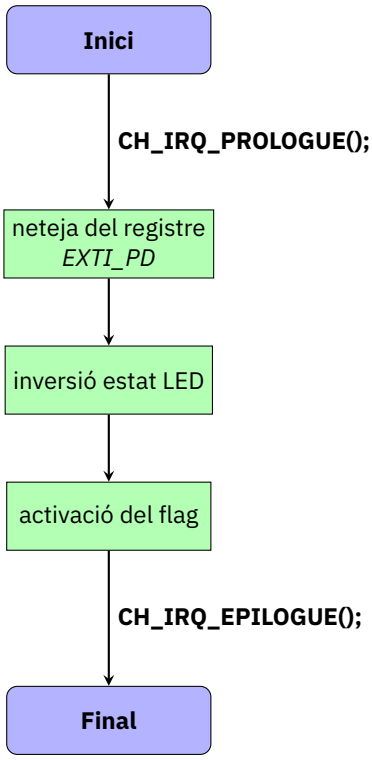

Figura 7: Diagrama de flux de la macro implementada per a la definició de la interrupció que associem al botó USR.

La *figura 7* es correspon amb el diagrama de flux associat a la implementació de la macro d'interrupció. Igualant el contingut del registre *EXTI_PR* amb la constant **EXTI_PR_PR0**, definida a l'arxiu de capçalera *stm32f4xx.h*, aconseguim netejar la petició d'interrupció al registre. A continuació, invertim l'estat del LED verd de la placa aplicant una màscara tipus *XOR* al registre *ODR* del port on es troben els LEDS, associat mitjançant la constant **LEDS_PORT** que ve definida a l'arxiu de capçalera *labBoard12.h*, amb la constant **GREEN_LED_BIT**, que correspon a un bit activat en la mateixa posició en que es troba la linia del led verd dins el registre *ODR*. Finalment, activem el flag per a indicar que s'ha produït una interrupció, i per tant es pot procedir amb la lectura de l'acceleròmetre.

```
CH_IRQ_HANDLER(EXTI0_IRQHandler) {
  CH_IRQ_PROLOGUE();

  EXTI->PR = EXTI_PR_PR0; // Esborrat posició PR0 del registre EXTI_PR.

  // Invertim l'estat LED.
  LEDS_PORT->ODR ^= GREEN_LED_BIT;

  // Activem el flag.
  switchFlag = 1; // Valor diferent de 0.

  CH_IRQ_EPILOGUE();
}
```

2. Mesures de temporització de l'acceleròmetre.

En aquesta secció hem utilitzat el mètode implementat per a generar interrupcions, *void interruptTest(void)*, per a recollir mesures sobre la temporització en els protocols de comunicació microcontrolador-acceleròmetre.

```
#include "Base.h"
#include "lcd.h"
#include "accel.h"
#include "int.h"

int main(void) {
   baseInit(); // Inicialització de la placa
   LCD_Init(); // Inicialització del LCD
   initAccel(); // Inicialització de l'acceleròmetre

   interruptTest(); // Rutina per les interrupcions

   return 0;
}
```

Per a realitzar les mesures de temporització hem implementat al mètode principal, *int main(void)*, la inicialització de la placa, cridant a *void baseInit(void)*, la inicialització del LCD, cridant a *void LCD_Init(void)* implementat a *lcd.h*, i la inicialització de l'acceleròmetre, cridant *void intAccel(void)* implementat a *accel.h*. Posteriorment, simplement cridant al mètode implementat *void interruptTests(void)* accedim a la interrupció a través del boto USR per a fer crides de l'ectura a l'acceleròmetre.

Per a realitzar les mesures, hem fet servir l'analitzador lògic disponible amb l'oscil·loscopi del laboratori. Configurant el trigger manualment, per a fer captures en mode *single*, i en cada cas en funció de si ens interessing un flanc de pujada o baixada per al senyal de referència que utilitzem. Compil·lant el codi, establint la connexió per un servidor tipus D i carregant el programa, hem activat la interrupció amb el polsador USER, obtenint la captura corresponent a la *figura 8*.

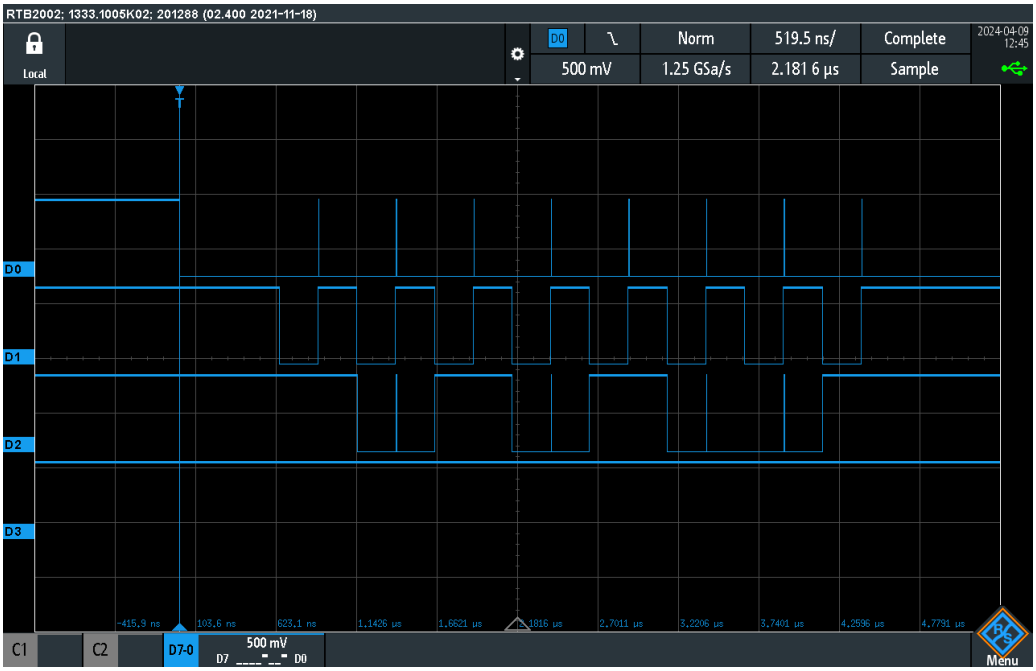

Figura 8: Captura de l'oscil·loscopi per al trigger configurat per a disparar en mode *single*, en el primer flanc de baixada de l'entrada D0. Les entrades *D0...D3* es corresponen, respectivament, amb *Chip Select* (CP), senyal del clock (SPC), senyal d'entrada a l'acceleròmetre (SDI) i la sortida de l'acceleròmetre (SDO).

Com el mètode d'interrupcions que hem implementat, *void interruptTest(void)*, en activar-se el polsador fa una primera lectura del registre *0x29* de l'acceleròmetre, el que es correspon amb el valor de l'acceleracció en aquell instant per a l'eix X, el senyal que haurem capturat per a l'SDI haurà de ser l'instrucció de lectura que enviem des del microcontrolador.

R/$\overline{W}$	MS	REG_ADRESS [5:0]
1	0	0b101001

A partir de la *figura 8*, si ens fixem en la linia del *SDI*, D2, i en els polsos de lectura del senyal de *chip select*, D0, veiem que, efectivament, la cadena de bits que llegeix l'acceleròmetre es correspon a 0b10101001.

La primera mesura de temporització que hem verificat ha estat el període del rellotge, SPC. La *figura 9* es correspon amb la mesura que hem pres, utilitzant cursors verticals amb l'ajut de l'oscil·loscopi. Experimentalment hem determinat un valor de període $t_{c(SPC)} = 499.2\text{ns}$, que contrastat amb els valors nominals proporcionats pel fabricant es verifiquen les especificacions ja que $t_{c(SPC)} \geq 100\text{ns}$.

Figura 9: En ordre descendent, mesura amb l'oscil·loscopi del període de rellotge SPC, $t_{c(SPC)} = 499.2\text{ns}$, i mesura amb l'oscil·loscopi del temps de *setup* pel senyal de *chip select*, $t_{su(CS)} = 632.5\text{ns}$. Les entrades *D0...D3* es corresponen, respectivament, amb *Chip Select* (CP), senyal del clock (SPC), senyal d'entrada a l'acceleròmetre (SDI) i la sortida de l'acceleròmetre (SDO).

La segona mesura que hem verificat ha estat el temps de *set up* del senyal de *chip select*. La *figura 9* es correspon amb les mesures que hem obtingut experimentalment, $t_{su(CS)} = 632.5\text{ns}$. El valor obtingut compleix amb les especificacions del fabricant, que imposen una condició $t_{su(CS)} \geq \text{ns}$.

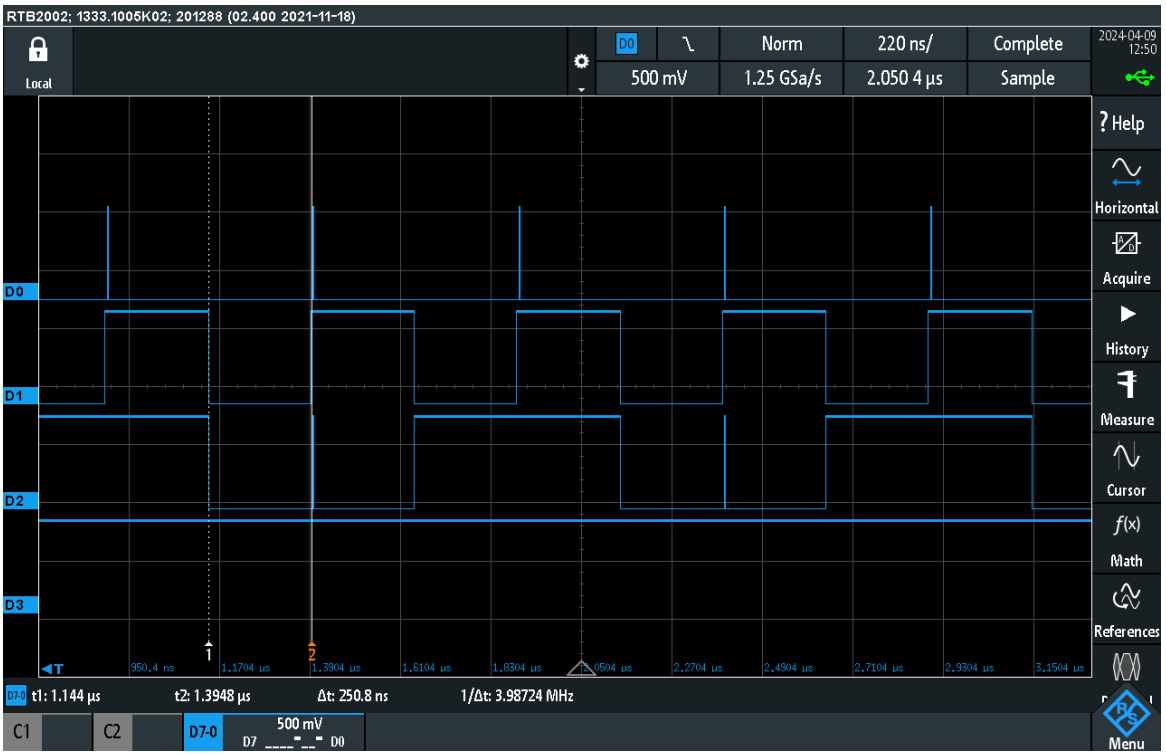

Figura 10: En ordre descendent, mesura amb l'oscil·loscopi del temps de *set up* pel senyal *SDI*, $t_{su(SDI)} = 250.8\text{ns}$, i mesura del temps de *hold* del mateix senyal, $t_{h(SDI)} = 251.42\text{ns}$. Les entrades *D0...D3* es corresponen, respectivament, amb *Chip Select* (CP), senyal del clock (SPC), senyal d'entrada a l'acceleròmetre (SDI) i la sortida de l'acceleròmetre (SDO).

La tercera i quarta mesura que hem verificat ha estat el temps de *set up* i temps de *hold*, del senyal SDI. La *figura 10* correspon als resultats experimentals que hem obtingut, modificant la base de temps de l'oscil·loscopi per a millorar la precisió de la mesura i utilitzant els cursors verticals per a recollir les mesures. El temps de $t_{su(SDI)} = 250.8\text{ns}$, el qual compleix amb els requeriments del fabricant ja que verifica $t_{su(SDI)} \geq 5\text{ns}$. El temps

de $t_{h(SDI)} = 251.42\text{ns}$, que verifica l'especificació del fabricant, $t_{h(SDI)} \geq 15\text{ns}$.

Les darreres mesures de temporització que hem verificat han estat el temps de validació del SDO, i el temps de hold del mateix senyal. La *figrua 11* es correspon amb els resultats experimentals obtinguts. El temps de validació que hem mesurat ha estat $t_{v(SDO)} = 17.617\text{ns}$, que compleix amb els requisits del fabricant ja que $t_{v(SDO)} \leq 50\text{ns}$. Per el temps de hold hem mesurat $t_{h(SDO)} = 18.3484\text{ns}$, que també es correspon al marge donat pel fabricant en les seves especificacions ja que $t_{ht(SDO)} \geq 6\text{ns}$.

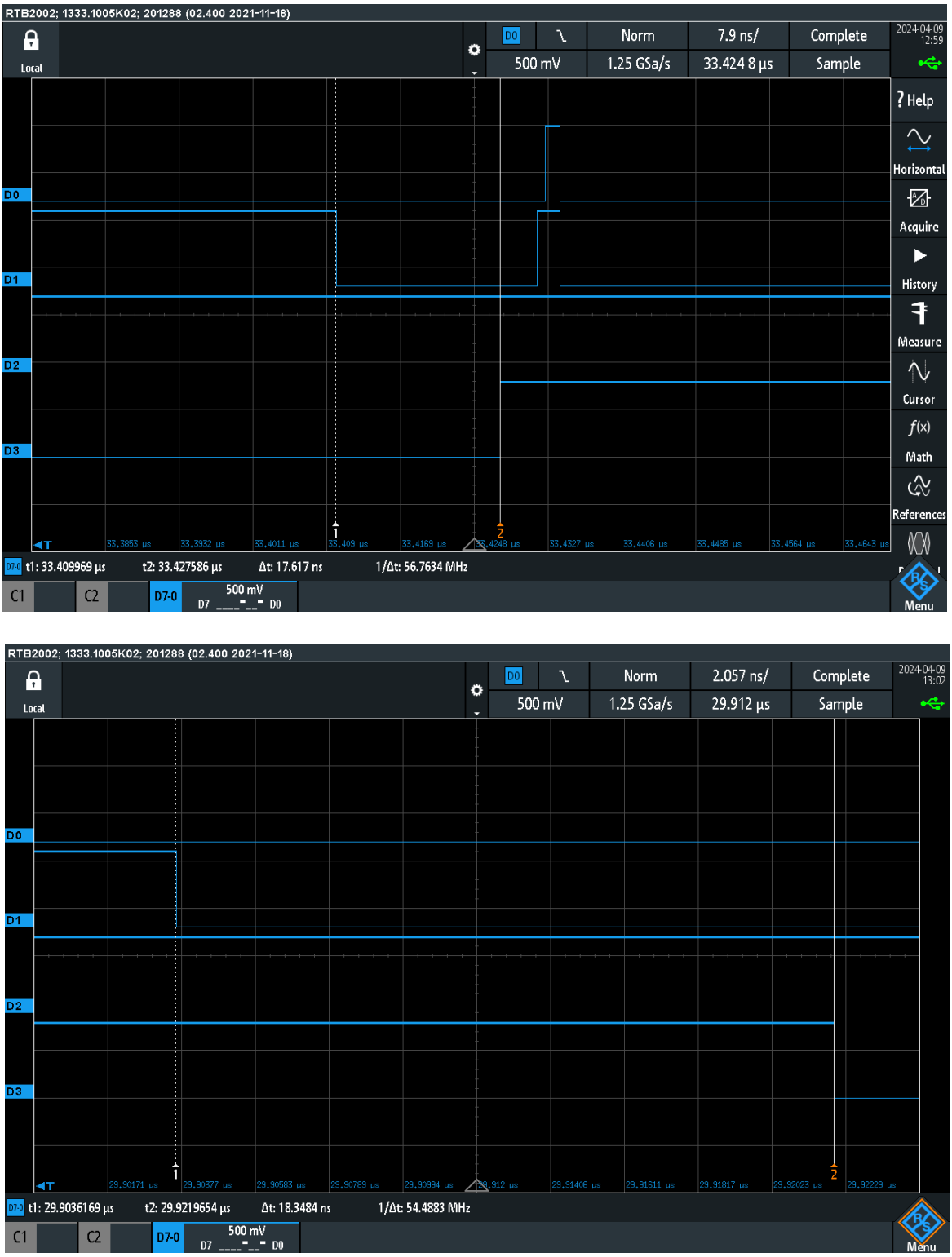

Figura 11: En ordre descendent, mesura amb l'oscil·loscopi del temps de *valid output* pel senyal *SDO*, $t_{v(SDO)} = 17.617\text{ns}$, i mesura del temps de *hold* del mateix senyal, $t_{h(SDO)} = 18.3484\text{ns}$. Les entrades *D0...D3* es corresponen, respectivament, amb *Chip Select* (CP), senyal del clock (SPC), senyal d'entrada a l'acceleròmetre (SDI) i la sortida de l'acceleròmetre (SDO).

3. Configuració d'interrupcions i mesures de temporització de l'LCD.

En aquesta secció, hem implementat un nou mètode per a la verificació de les mesures de temporització del LCD, utilitzant interrupcions a través del polsador USER com en el cas anterior.

El mètode implementat ha estat *void interruptLCDTest(void)*, declarat al fitxer capçalera *int.h*, que no rep ni retorna cap variable.

```
void interruptLCDTest(void){
  // Activació del clock
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;


  // Configuració de EXTI
  SYSCFG->EXTICR[0] = (SYSCFG->EXTICR[0] & (~SYSCFG_EXTICR1_EXTI0)) | SYSCFG_EXTICR1_EXTI0_PA;
  EXTI->IMR |= EXTI_IMR_MR0;
  EXTI->RTSR |= EXTI_RTSR_TR0; // Nosaltres el configurem com a flanc de pujada
  EXTI->FTSR &= ~EXTI_FTSR_TR0; // Desactivem la configuració de flanc de baixada.
  EXTI->PR = EXTI_PR_PR0; // Netejem PR.


  // Activació de NVIC.
  nvicEnableVector(EXTI0_IRQn,CORTEX_PRIORITY_MASK(STM32_EXT_EXTI0_IRQ_PRIORITY));


  while(1) {
    switchFlag = 0; // Netejem el flag.

    while(switchFlag == 0); // Esperem a que s'activi.
      LCD_Init();

      // lcdNibble(0b1011,0);
      // lcdNibble(0b0100,1);
  }
}
```

El mètode implementat configura de la mateixa manera que en el cas de l'acceleròmetre, les interrupcions per utilitzar el polsador USER, connectat a la linia PA0 GPIO, com a trigger per les interrupcions. En aquest cas, però, com volem realitzar mesures sobre la temporització en la inicialització del LCD i en les operacions de comuniacació microcontrolador-LCD, hem modificat les accions de la funció una vegada s'activa la interrupció.

En activar-se el flag d'interrupció, en el primer cas el mètode invocarà la funció d'inicialització del LCD, *void LCD_Init(void)* declarada al fitxer de capçalera *lcd.h*, i una vegada finalitzat netejarà el flag i tornarà a esperar a una nova interrupció. Per a les segones mesures, comentariem la invocació al mètode d'inicialització del LCD i descomentariem les dues crides al *void lcdNibble(int32_t nibbleCmd, int32_t RS)*, mètode també delcarat al fitxer de capçalera *lcd.h*.

```
#include "Base.h"
#include "lcd.h"
#include "accel.h"
#include "int.h"

int main(void) {
  baseInit(); // Inicialització de la placa
  // LCD_Init(); // Inicialització del LCD
  initAccel(); // Inicialització de l'acceleròmetre

  interruptLCDTest(); // Rutina per les interrupcions

  return 0;
}
```

Pel que respecta al *main loop*, fitxer *main.c*, de manera predeterminada inicialitzem tant la placa, invocant el mètode *void baseInit(void)* declarat al fitxer de capçalera *Base.h*, com l'acceleròmetre, cridant *void initAccel(void)* declarat al fitxer de capçalera *accel.h*. Posteriorment, es crida la funció d'interrupció *void interruptLCDTest(void)*. Per al primer cas, com volem verificar mesures sobre la temporització en la inicialització de la pantalla comentem la invocació a *void LCD_Init(void)*, i l'invoquem en el seu lloc al mètode d'interrupció, comentant les linies que criden als *void lcdNibble(int32_t nibbleCmd, int32_t RS)*.

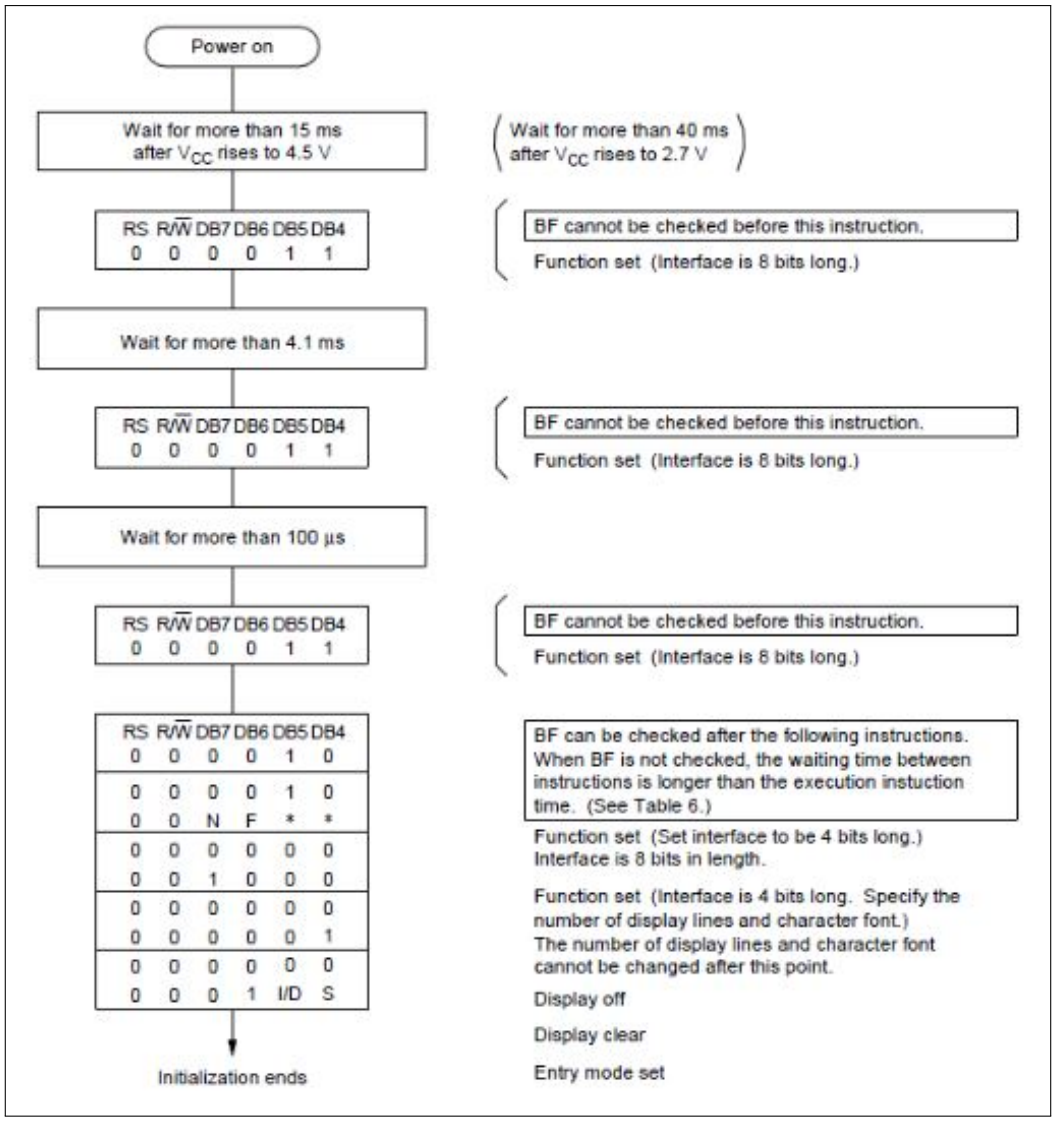

Figura 12: Diagrama corresponent al procés d'inicialització del LCD instal·lat a la placa de laboratori.

La primera mesura de temporització que hem verificat ha estat, esperar $\geq 4.1\text{ms}$ després d'enviar 0b0011 amb RS = '0' i R/$\overline{W}$ = '0'. Configurant l'oscil·loscopi per a capturar una única vegada al detectar un flanc de pujada en el senyal *d'enable*, i ajustant la base de temps a 1ms, hem obtingut els resultats experimentals de la *figura 13*. Per un costat, podem verificar que efectivament estem enviant la cadena de bits per les linies de dades corresponent a 0b0011.

Pel que fa a la mesura de temporització, amb l'ajut de cursors hem mesurat l'interval entre els dos polsos dels senyals d'*enable*, que corresponen a les dues crides consecutives de *void lcdNibble(int32_t nibbleCmd, int32_t RS)*, obtenint experimentalment $t_{wait(1)} = 26.25\text{ms} \geq 4.1\text{ms}$, per tant es verifiquen les especificacions.

Per a la segona mesura de temporització, hem procedit de la mateixa manera. Els resultats experimentals es corresponen amb la segona imatge de la *figura 13*, en que aquesta vegada hem pogut ampliar més la base temporal ja que l'intèrval de separació era significantment inferior a l'anterior. Hem mesurat amb l'ajuda dels cursors verticals de l'oscil·loscopi $t_{wait(2)} = 1.47\mu\text{s} \geq 100\mu\text{s}$, i per tant efectivament complim amb les especificacions.

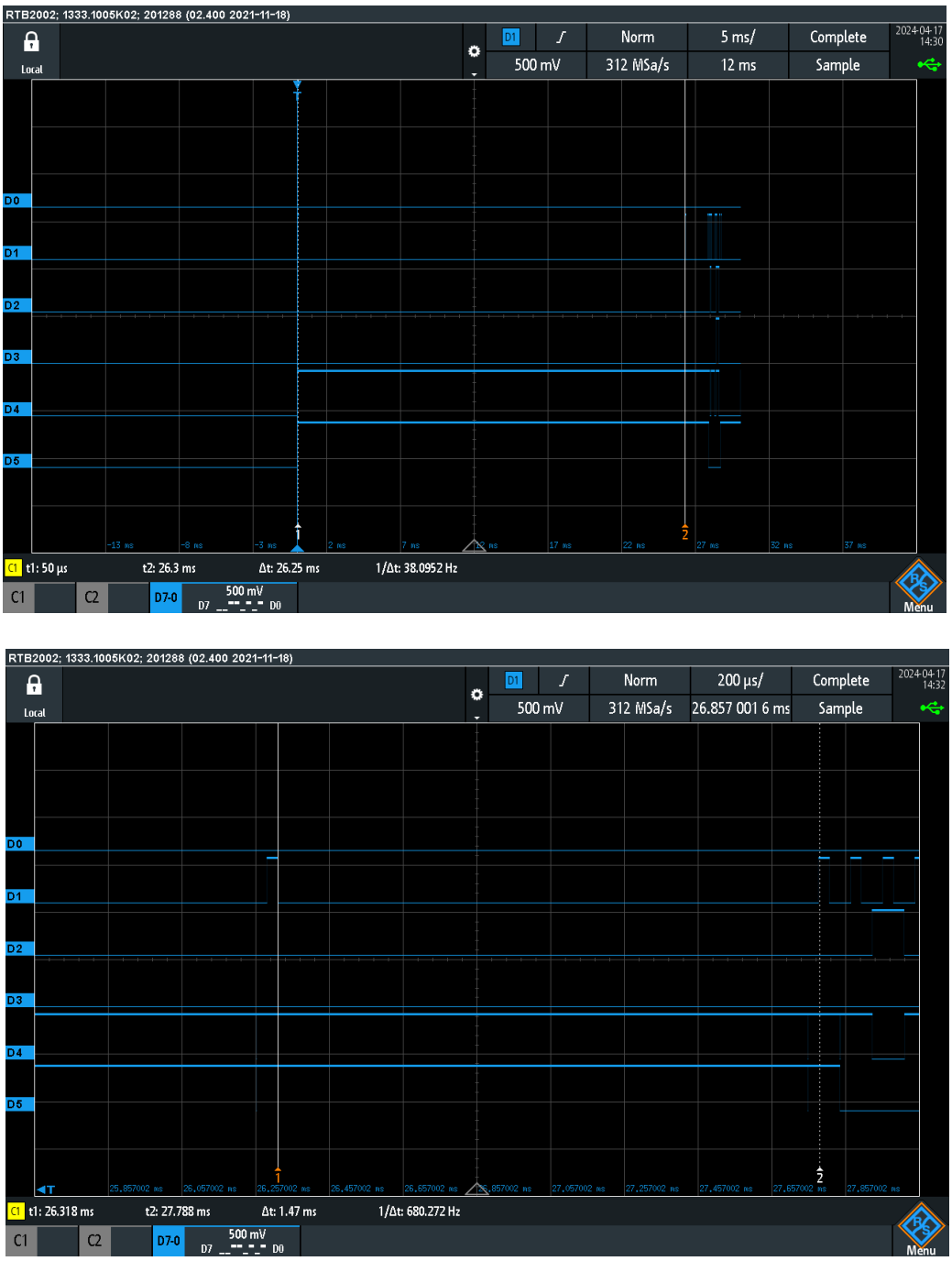

Figura 13: En ordre descendent, mesura de temporització del senyal d'espera $t_{wait(1)} \geq 4.1\text{ms}$ i mesura del senyal d'espera $t_{wait(2)} \geq 100\mu\text{s}$. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

Un resultat que hem observat, també, és en comparar els intèrvals temporals mesurats amb el codi d'inicialització de la funció *void LCD_Init(void)*, a *l'annex B* es pot consultar el seu codi. Tot i que en el primer intèrval s'especifica *SLEEP_MS(10)*, entre invocació i invocació de *lcdNibble*, hem mesurat un temps aproximadament del doble. De mateixa manera amb *DELAY_US(500)*, per a la segona espera. D'aquí hem conclòs a que això ha de ser degut a una disparitat entre el clock que utilitza el microcontrolador, i la freqüència amb la que el compilador fa els càlculs per a programar en codi màquina aquests períodes d'espera.

Per a les següents mesures de temporització, hem analitzat la comunicació des del microcontrolador cap al LCD mitjançat l'enviat d'una instrucció completa de 8 bits, això és invocant dues vegades *void lcdNibble(int32_t nibbleCmd, int32_t RS)*. Per això, al fitxer *main.c* dins de *int main(void)*, hem descomentat la inicialització de la pantalla, i al mètode d'interrupcions descomentat les crides al *lcdNibble*, tot comentant la invocació de *void LCD_Init(void)*.

La primera mesura de temporització que hem verificat ha estat el temps de *adress setup*. La *figura 14* correspon a la mesura experimental obtinguda a l'oscil·loscopi, $t_{AS} = 29\mu s$. La mesura verifica el criteri del fabricant, que estableix $t_{AS} \geq 20ns$.

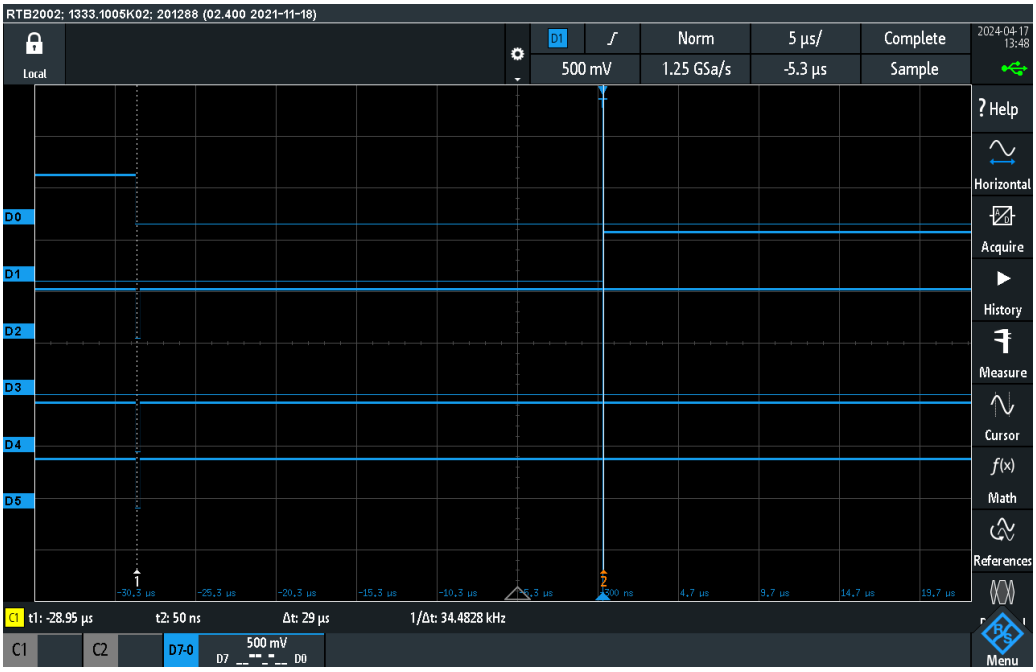

Figura 14: Mesura del senyal d'espera t_{AS}. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

La segona mesura que hem verificat ha estat el temps d'*adress hold*. La *figura 15* mostra els resultats obtinguts al laboratori, $t_{AS} = 29.8\mu s$. Aquesta temporització també compleix amb els requisits del fabricant, que imposa $t_{AS} \geq 10ns$.

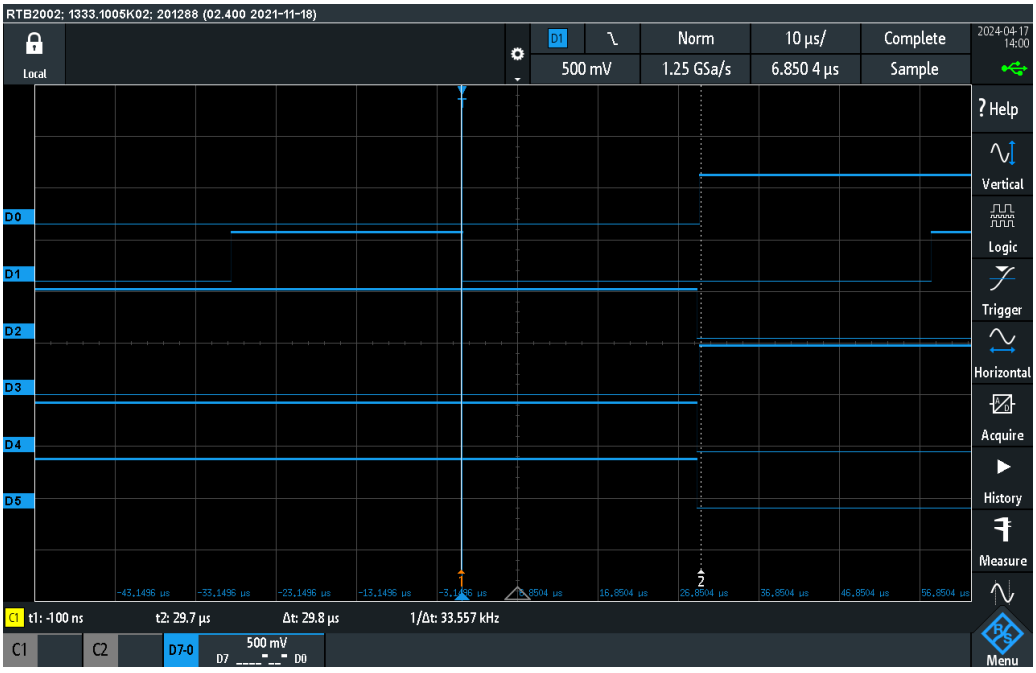

Figura 15: Mesura del senyal d'espera t_{AH}. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

La tercera mesura que hem verificat ha estat el temps de *data setup*. Experimentalment hem obtingut les mesures de la *figura 16*, $t_{DSW} = 58 - 2\mu s$. Atenent les especificacions del fabricant, com $t_{DSW} \geq 80ns$ concluim que es compleixen els requisits de temporització.

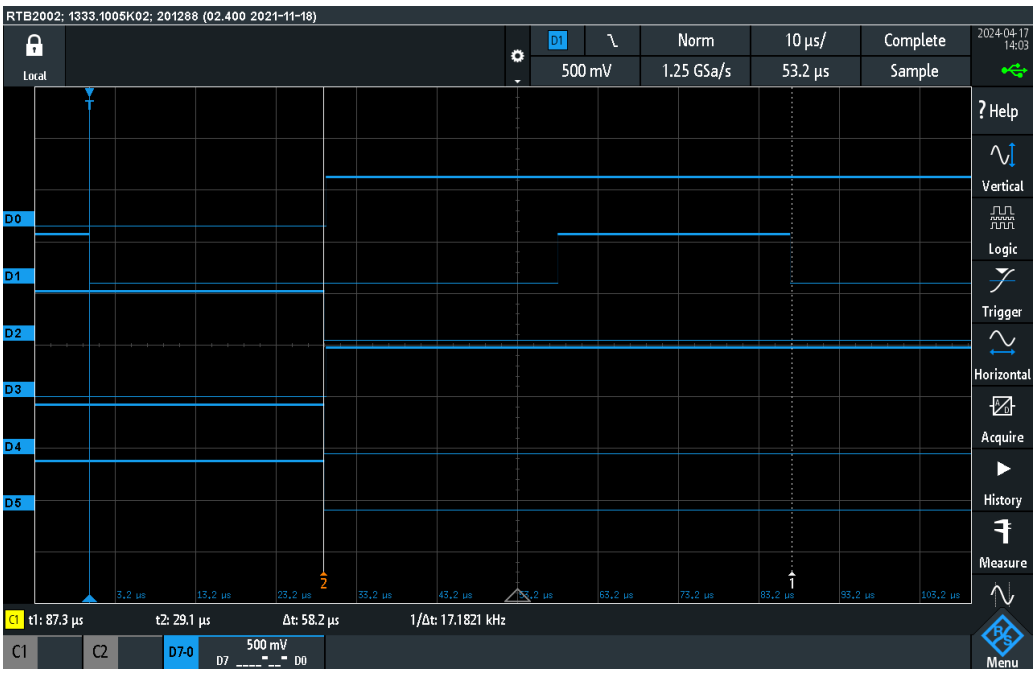

Figura 16: Mesura del senyal d'espera t_{DSW}. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

La quarta mesura de temporització que hem verificat ha estat el temps de *data hold*. La *figura 17* es correspon amb els resultats obtinguts experimentalment, $t_H = 29.5\mu s$. Observem que es verifiquen els requisits de temporització imposats pel fabricant, $t_H \geq 10ns$.

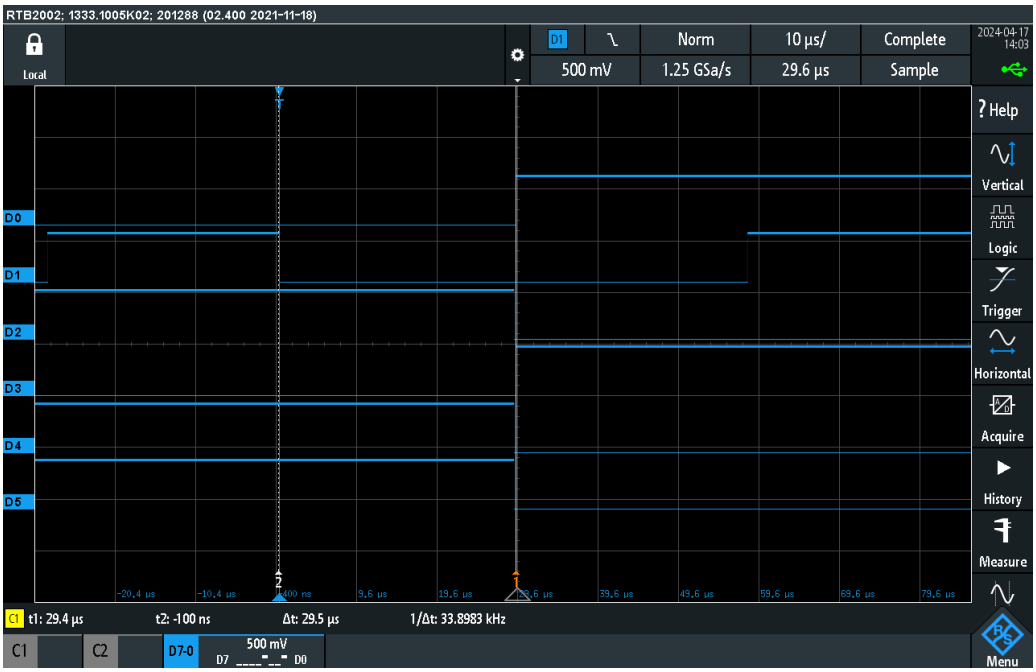

Figura 17: Mesura del senyal d'espera t_H. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

La darrera mesura de temporització que hem verificat ha estat el temps de *enable pulse width*. La *figura 18* es correspon amb els resultats obtinguts al laboratori, $t_{pWEH} = 28.7\mu s$. Com que $t_{pWEH} \geq 230ns$, es compleixen els requisits del fabricant.

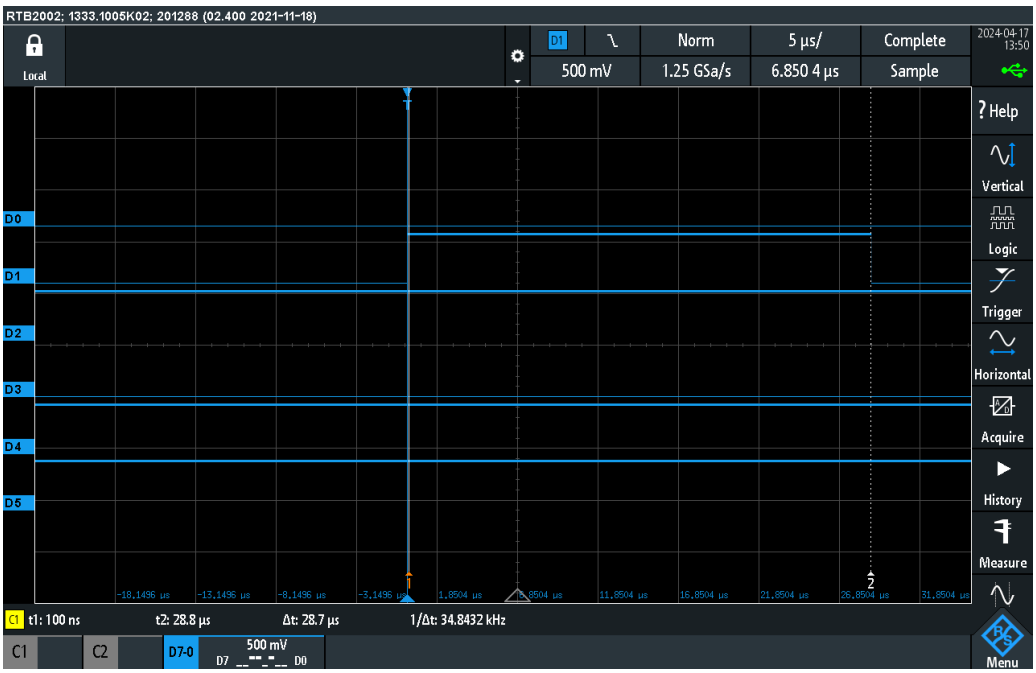

Figura 18: Mesura del senyal d'espera t_{pWEH}. Les entrades D0...D5 corresponen, respectivament, als senyals *Regiser Select* (RS), *enable* (E), i les linies de dades DB3...DB0.

Annexos.

Annex A. Fitxer de capçelera *int.h*

```
/************************************************************

i n t . h

Interrupt test source file

************************************************************/

#ifndef _INT_H   // Definitions so that the
#define _INT_H   // file is included only once

/****** Public function prototypes *******/

void interruptTest(void);
void interruptLCDTest(void);

#endif   // int.h
```

Annex B. Mètode *void LCD_Init(void)*

```
void LCD_Init(void) {
// Initializes the LCD GPIOs
lcdGPIOInit();

// Wait 50ms for vdd to stabilize
SLEEP_MS(50);

// Init command 1
lcdNibble(0x03,0);
SLEEP_MS(10); // Wait > 4.1ms

// Repeat command
lcdNibble(0x03,0);

// Wait > 100us
DELAY_US(500);

// Repeat command
lcdNibble(0x03,0);

// Init command 2
lcdNibble(0x02,0);

// Mode 4 bits 2 lines 5x8 chars
lcdNibble(0x08,0);
lcdNibble(0x02,0);
DELAY_US(50);

// Display on blink on and cursor on
lcdNibble(0x00,0);
lcdNibble(0x0E,0);
DELAY_US(50);

// Clear Display
lcdNibble(0x00,0);
lcdNibble(0x01,0);
```

```
SLEEP_MS(5);

// Cursor set direction
lcdNibble(0x00,0);
lcdNibble(0x06,0);
DELAY_US(50);

// Set ram address
lcdNibble(0x08,0);
lcdNibble(0x00,0);
DELAY_US(50);

// Send OK
lcdNibble(0x04,1);
lcdNibble(0x0F,1);
DELAY_US(50);
lcdNibble(0x04,1);
lcdNibble(0x0B,1);
DELAY_US(50);
}
```