Lab01:

Videos outcome

• Microcontrollers have CPU and i/o preference.

• There are 6 i/o preferences.

• The connections in the controller will be thru the pins.

• There is the default i/o is gpio

• The 6 are: gpio, adc, nvic, ssp, timer0, pwm.

• We can control the pins using the gpio registers.

• There are ports in the microcontroller that control the pins.

• The 5 ports have up to 32 pins.

• If we want to change pin we change the bit associated to it.

• The pointers is needed in changing pins bits.

• We use shifting to change the bits we need.

Ex:

Lab02:

* Video comments
* CMSIS:
* It allows the programmer to write the code without worrying about the address of the registers.
* LPC\_GPIO1 -> FIODIR |= (1<<5) | (1<<13);
* The part needed of the CPU: LPC
* Choose the register: GPIO1
* Port number can be changed depending on the pin: RIODIR
* Any input device must give 2 logics.

1. #define x \*(int \*) 0x2009c000 → the right star refers to casting (int \*) we are casting to int temporarily,, the left star refers to the red highlighted is type casting an address to an integer (need to ask Dr. Hazim about the meaning of this) and the green highlighted is the variable that whenever it appears in the code it will be replaced by the red.
2. someRegister = 3 → assuming that the variable someRegister is replaced by the address of port 0, then this means that I’m setting all pins within this port to be inputs except for the last two pins since we have 3 is 00000000000…………11
3. someRegister = 1 << 9 | 1 << 13 will set pins 9 and 13 to be outputs.
4. someRegister |= 1 << 9 | 1 << 13 will set pins 9 and 13 to be outputs and the rest of the pins will

Lab03:

1. By now, we know that if we need to access a register, we need to know its address and how to use pointers in C language
2. In fact, this is not the best way to do it, we may utilize other people work
3. It is better to go one level up to the CMSIS instead of accessing the microcontroller directly.
4. 4- In the API that we are going to use, we will utilize the following statement:

Ex:

LPC\_GPIO2 -> FIODIR |= (1<<5) | (1<<13);

PART OF THE CPU TO BE ACCESSED

EXACT REGISTER IN THE PART OF THE CPU TO WRITE TO

VALUES TO BE WRITTEN

5- Eclipse IDE provide the list of registers that can be accessed in the targeted part of the CPU after completing writing the ((LPC\_GPIO2 ->))

#include "mbed.h"

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

int main()

{

        while(true){

                int x =LPC\_GPIO1 -> FIOPIN;

                x = x>>31;

                int aaa = x & 1;

                if(aaa == 0){

                    LPC\_GPIO1 -> FIODIR |= 1<<23;

                    LPC\_GPIO1 -> FIOSET |= 1<<23;

                    // wait\_us(1000000);

                    // LPC\_GPIO1 -> FIOCLR |= 1<<23;

                    // wait\_us(1000000);

        }

                else{

                    LPC\_GPIO1 -> FIOCLR |= 1<<23;

                }

    }

}

Lab04:

Interrupts are subsets of exceptions.

2- Recall: exceptions are when the CPU stops execution to jump to another more prior instruction (usually an error)

3- Interrupts are intentional, unlike the exceptions.

4- External interrupts are generated by one of the external devices via one of the bins

5- Don’t forget to enable the NVIC to be able to deal with interrupts.

6- Steps for enabling the GPIO external interrupts: if you are using bin 0 or 2, then interrupts can happen only from EINT3

Ex:

1. Enabling rising edge interrupts on bin 0 of port 2 only → LPC\_GPIOINT->IO2IntEnR = 1;
2. Enable the required interrupt in the NVIC → for example: write the command: NVIC\_EnableIRQ(IRQn); note that almost every type of intruppts (interanal or external) has the same way (the previous statement) for setting the NVIC, where you insert the interrupt ID depending on the source of the interrupt
3. Clear the interrupt request at the end of the ISR through the command:

LPC\_GPIOINT-> IO0IntClr = 1; port number (0,2)

#include "mbed.h"

int x;

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

 extern "C" {

    void EINT3\_IRQHandler(){

        // clr

        LPC\_GPIOINT-> IO2IntClr |= (1<<5);

        //

        LPC\_GPIO1 -> FIOSET |= 1<<23;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<23;

        LPC\_GPIO1 -> FIOSET |= 1<<21;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<21;

        LPC\_GPIO1 -> FIOSET |= 1<<20;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<20;

        LPC\_GPIO1 -> FIOSET |= 1<<18;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<18;

    }

}

int main()

{

    LPC\_GPIO1 -> FIODIR |= 1<<23;

    LPC\_GPIO1 -> FIODIR |= 1<<21;

    LPC\_GPIO1 -> FIODIR |= 1<<20;

    LPC\_GPIO1 -> FIODIR |= 1<<18;

        LPC\_GPIOINT-> IO2IntEnR |= (1<<5);

        NVIC\_EnableIRQ(EINT3\_IRQn);

        while(true){

    }

}

**2 interrupts code:**

#include "mbed.h"

int x;

int y;

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

 extern "C" {

    void EINT3\_IRQHandler(){

        // clr

        LPC\_GPIOINT-> IO2IntClr |= (1<<5);

        LPC\_GPIOINT-> IO0IntClr |= (1<<11);

        //

        int stat;

        int stat2;

        stat = LPC\_GPIOINT -> IO2IntStatR;

        stat2 =LPC\_GPIOINT -> IO0IntStatR;

        if(stat){

        x = x>>5;

        int aaa = x & 1;

        if(aaa == 0){

        LPC\_GPIO1 -> FIOSET |= 1<<23;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<23;

        LPC\_GPIO1 -> FIOSET |= 1<<21;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<21;

        LPC\_GPIO1 -> FIOSET |= 1<<20;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<20;

        LPC\_GPIO1 -> FIOSET |= 1<<18;

        wait\_us(500000);

        LPC\_GPIO1 -> FIOCLR |= 1<<18;

        }

        else{

            LPC\_GPIO1 -> FIOSET |= 1<<23;

            LPC\_GPIO1 -> FIOSET |= 1<<21;

            LPC\_GPIO1 -> FIOSET |= 1<<20;

            LPC\_GPIO1 -> FIOSET |= 1<<18;

        }

        }else if(stat2){

            LPC\_GPIO1 -> FIOSET |= 1<<23;

            LPC\_GPIO1 -> FIOSET |= 1<<21;

            LPC\_GPIO1 -> FIOSET |= 1<<20;

            LPC\_GPIO1 -> FIOSET |= 1<<18;

        }

    }

}

int main()

{

    LPC\_GPIO1 -> FIODIR |= 1<<23;

    LPC\_GPIO1 -> FIODIR |= 1<<21;

    LPC\_GPIO1 -> FIODIR |= 1<<20;

    LPC\_GPIO1 -> FIODIR |= 1<<18;

        x =LPC\_GPIO2 -> FIOPIN;

        y =LPC\_GPIO2 -> FIOPIN;

        LPC\_GPIOINT-> IO2IntEnR |= (1<<5);

        LPC\_GPIOINT-> IO0IntEnR |= (1<<11);

        NVIC\_EnableIRQ(EINT3\_IRQn);

        while(true){

    }

}

Lab05:

1. Balling is a waste of resource
2. **PLEASE NOTE THAT THE DIFFERENCE BETWEEN THE NON-GPIO INTRRUPTS AND THE GPIO INTRRUPTS will come in the next quiz**
3. To convert from
4. Some paraphilas are powered on by default, while some are powered off by default
5. If you put 1 in bit 16 in the A/D control register, then you are configureing the ADCR to operate continuously,
6. Bit 27 is a power pin, hence 1 should be written there for the register to work
7. Whenever we need to change the functionality of the pin, refer to chapter 8

Ex:

While(true){

x = (LPC\_ADC -> ADDR4 >> 4) & 0xFFF;

// printf("%d\n", x);

if (x == 0) {

LPC\_GPIO1 -> FIOCLR |= (1 << 23) | (1 << 21) | (1 << 20) | (1 << 18);

}

else if ((x > 0) && (x < 1024)) {

LPC\_GPIO1 -> FIOPIN = (1 << 18);

}

else if ( (1024 <= x) && (2048 > x) ) {

LPC\_GPIO1 -> FIOPIN = (1 << 18) | (1 << 20);

}

else if ((2048 <= x) && (3072 > x)) {

LPC\_GPIO1 -> FIOPIN = (1 << 18) | (1 << 20) | (1 << 21);

}

else if ((3072 <= x) && (4095 >= x)) {

LPC\_GPIO1 -> FIOPIN = (1 << 23) | (1 << 21) | (1 << 20) | (1 << 18);

}

}

#include "mbed.h"

//Experiment 4: Analog Input and Output

int main()

{

    //POWER UP

    LPC\_SC -> PCONP |= (1 << 12);

    // pin number p20

    LPC\_PINCON -> PINSEL3 |= (1<<30) | (1<<31); //adc 0.5

    LPC\_ADC->ADCR |= (1<<5) | (1<<21) | (1<<16)| (1<<8);

    int x;

    LPC\_GPIO1 -> FIODIR |= (1<<23) | (1<<21)| (1<<20)| (1<<18);

    while (true) {

        x = (LPC\_ADC->ADDR5 >>4)& 0xFFFF;

        if(x>3500){

            LPC\_GPIO1 -> FIOPIN = (1<<23);

        }

        else if(x>3000){

            LPC\_GPIO1 -> FIOPIN = (1<<21);

        }

        else if(x>2000){

            LPC\_GPIO1 -> FIOPIN = (1<<20);

        }

        else {

            LPC\_GPIO1 -> FIOPIN = (1<<18);

        }

    }

}

Lab06:

1. Timers are part of the board functions like GPIO
2. Timers do not run by default, they need to be sat
3. The main component of the timer is the counter
4. When the counter is ticking the timer is running
5. In chapter 21 in the manual, the details of running timers are there, ion addition to the pdf file of the experiment
6. The register needed to run timers is TCR

Ex:

int main() {

// a 1 in TCR, the TC will start counting

NVIC\_EnableIRQ(TIMER0\_IRQn);

LPC\_TIM0

while (true) {}

}

#include "mbed.h"

//Experiment 5: digital to analog  Input and Output

int main()

{

    // pin number p18

    LPC\_PINCON -> PINSEL1 |= (0<<20) | (1<<21); //DCA 0.5

// for timer next time

    // LPC\_TIM0 -> TC = 1;

    while (true) {

        LPC\_DAC->DACR |= (1000<<8);

        wait\_us(1000);

    }

}

Lab07:

1. We can make the timer counter running using TCR register
2. Refer to page 509 from the user manual to know what value of (00, 01, 10, 11) to put in the EMR register to do what task
3. If you configured a pin to be MAT2.0, then you are connecting the timer 2 to the this pin while connecting the match register 0 to this pin.
4. EMx can be driven via the MATx on the chip, but first, MAT needs to be configured to do another functionality as described in the previous labs.

Ex:

#include "mbed.h"

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

extern "C"{

void TIMER0\_IRQHandler() {

    // LPC\_GPIOINT-> IO2IntClr;   //Clear at the beginning

    // to reset counter

    LPC\_TIM0 -> IR |= 1;

    // to put 1 in the led and keep turnig it on and off

    LPC\_GPIO1 -> FIOPIN ^= (1<<23);

    }

}

int main()

{

    //task: creating an interupt in oreder to blink the led with timer

    NVIC\_EnableIRQ(TIMER0\_IRQn);

    // LPC\_GPIOINT-> IO2IntEnR;

    // to enble the counter

    LPC\_TIM0 -> TCR |= 01 ;

    // using match register 2

    LPC\_TIM0 -> MR2 =20000000;

    // BY looking at table 430 you can see that MCR related to MR2 iscontroled by putting 1 in bit 6 and 7

    //so if you want to use for instance MR0 you have to put 1 in bit 0 and 1 in MCR T control IT

    LPC\_TIM0 -> MCR =(1<<7) | (1<<6);

    //for direction

    LPC\_GPIO1 -> FIODIR |= 1<<23;

    while (true) {

    }

}

Ex2:

#include "mbed.h"

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

int main()

{

    //ENBLE THE TIMER 2 BECOUSE ITS 0 BY DEFOUT

    LPC\_SC -> PCONP |= (1 << 22);

    LPC\_PINCON -> PINSEL0 |= (1<<12) | (1<<13); //CONFIG TO mat2.0 (functioality is 11) (8.5.1)    // to enble the counter

    LPC\_TIM2 -> TCR |= 01 ;

    // using match register 2

    LPC\_TIM2 -> MR0 =20000000;

    // BY looking at table 430 you can see that MCR related to MR2 iscontroled by putting 1 in bit 6 and 7

    //so if you want to use for instance MR0 you have to put 1 in bit 0 and 1 in MCR T control IT

    LPC\_TIM2 -> MCR =3;

    //for direction

    //TO TOGGEL MAT2.0

    LPC\_TIM2-> EMR |= (1<<5)|(1<<4);

    while (true) {

    }

}

Lab08:

1. The ADC circuit is powered off by default
2. Use this statement to turn on the timers:

LPC\_SC -> PCONP |= (1<<12); // Why 12? this is for powering on sice it is powered off by default! --> from AC to DC

1. We use the ADCR to configure the operations of the ADC.
2. LPC\_ADC -> ADCR = (1<<4) | (1<<21) | (1<<16) | (1<<11);

// 16 to make ADC always working

1. refer to page 113 table 75 to see how to change the configuration of pins to MAT or G INTERRUPTS

Ex:

Lab09:

1. Pulse Width Modulation (PWM) aims at generating a periodic signal indefinitely in a high and low manner with the ability to control the width.
2. The first thing we need to do to work with PWM is to configure one of the pins to work as PWM
3. We have 2 types of PWM signals: the single edge (at the beginning of the period it will go high and will go low somewhere in the middle and come back to be high again at the end of the period) and the double edge (at the beginning of the period it will remain as is (high or low) then in the middle it will go the other way.
4. We need 2 MRs for the single edge PWM signal, one of them is always MR0 which will contain the period of the signal (if was filled with 100, then this is 100 pulses of the parapharel clocks. Assume the clock is 1 MHz (1 uSecond), the period is 100 uSecond)
5. We have several pins that can use PWM, which we will refer to as PWM1, 2,3 and so on. You select the second MR1,2,3 and so on depending on the PWMx that you choose to place the value of the output (where the signal will go low in the single edge, then it will come back high at the end of the period).

Ex:

#include "mbed.h"

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

int main()

{

    LPC\_PINCON -> PINSEL3 |= (1<<5) | (0<<4); //PWM1.1

    LPC\_PWM1->MR0 = 2000000;

    LPC\_PWM1->MR1 = 1000000;

    LPC\_PWM1->TCR |= (1<<3)|(1<<0) ;

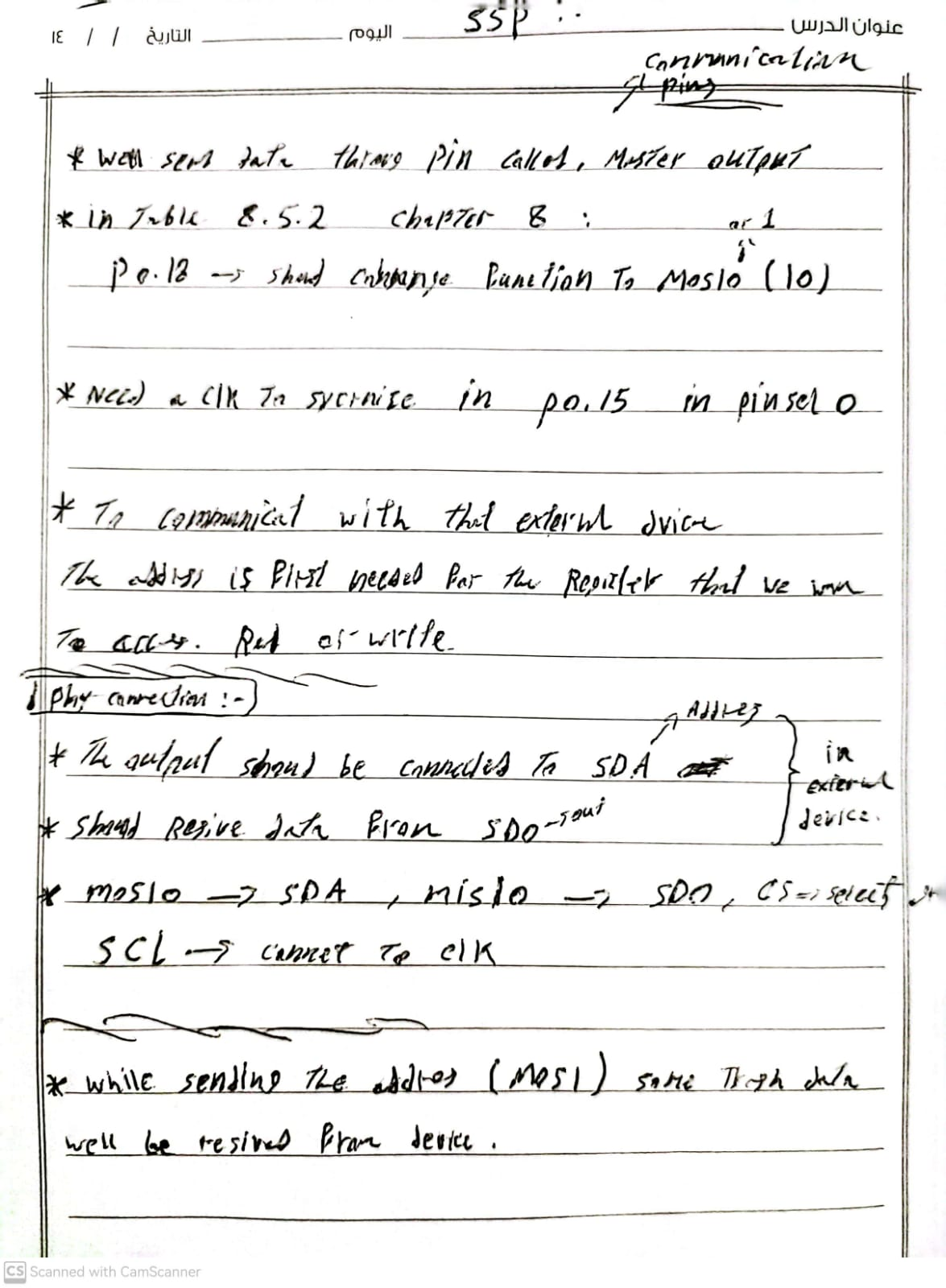
    LPC\_PWM1->PCR |= (1<<9) ;

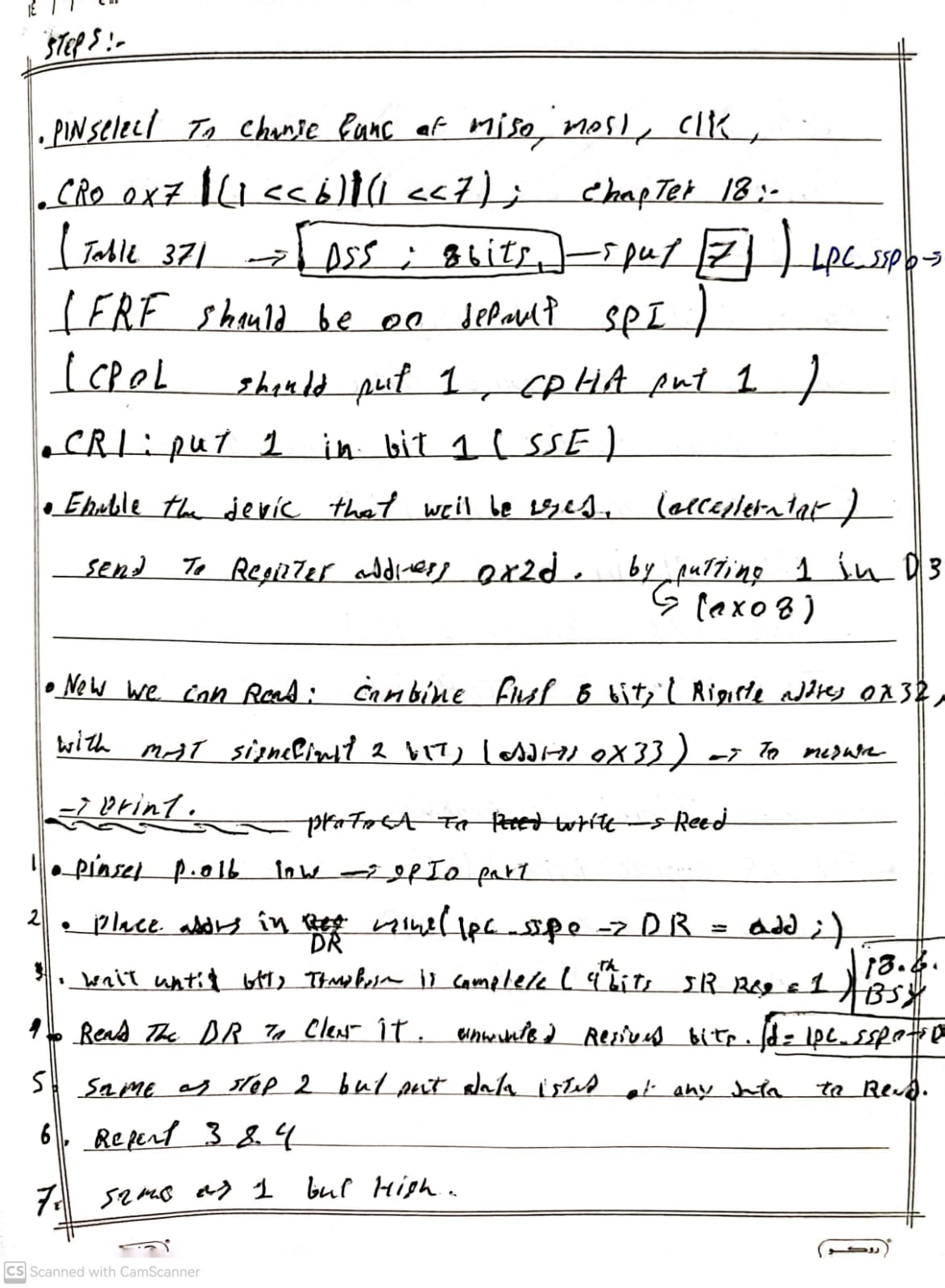
    while (true) {

    }

}

Lab 10:





#include "mbed.h"

// Blinking rate in milliseconds

#define BLINKING\_RATE     500ms

int main()

{

    //TO CHANGE THE DRIECTION OF P0.16 OUTPUT

    LPC\_GPIO0 -> FIODIR |= 1<<16;

    //MAKE THE PIN LOW

    LPC\_GPIO0 -> FIOCLR |= 1<<16;

    //MISO0

    LPC\_PINCON-> PINSEL1 |= 1<<3;

    //MOSI0

    LPC\_PINCON-> PINSEL1 |= 1<<5;

    //CLK (U FOR OVERFLOW) p0.15

    LPC\_PINCON-> PINSEL0 |= 1U<<31;

    //CR0 CONFIG (sending 8 bits)

    LPC\_SSP0 -> CR0 |= (1<<0)| (1<<1)| (1<<2)| (1<<6)| (1<<7);

    //CR1 CONFIG

    LPC\_SSP0 -> CR1 |= (1<<1);

    // send: //

    // SEND TO ADDERSS 0X2D THE VALUE 0X08

    LPC\_SSP0 -> DR = 0X2D;

    //checking if they are send or not (1 means busy, 0 idle)

    while(LPC\_SSP0 -> SR & (1<<4)) {}

    // acces the reg to clear the unwanted data coming from the device to the DR

    int d;

    d = LPC\_SSP0 -> DR;

    // data to send to the device

    LPC\_SSP0 -> DR = 0X08;

    //

    while(LPC\_SSP0 -> SR & (1<<4)) {}

    d = LPC\_SSP0 -> DR;

    //MAKE THE PIN HIGH

    LPC\_GPIO0 -> FIOSET |= 1<<16;

    //receive //

    // x value  lest sig 8 bits

    LPC\_SSP0 -> DR = 0XB2;

    //checking if they are send or not (1 means busy, 0 idle)

    while(LPC\_SSP0 -> SR & (1<<4)) {}

    // acces the reg to clear the unwanted data coming from the device to the DR

    d = LPC\_SSP0 -> DR;

    // send any data to recive the clean ones

    LPC\_SSP0 -> DR = 0X00;

    d = LPC\_SSP0 -> DR;

    int x1 = d;

    // x value  most significant 2 bits

    LPC\_SSP0 -> DR = 0XB3;

    //checking if they are send or not (1 means busy, 0 idle)

    while(LPC\_SSP0 -> SR & (1<<4)) {}

    // acces the reg to clear the unwanted data coming from the device to the DR

    d = LPC\_SSP0 -> DR;

    // send any data to recive the clean ones

    LPC\_SSP0 -> DR = 0X00;

    d = LPC\_SSP0 -> DR;

    int x2 = d;

    while (true) {

    //concainate both the 2 bits and 8 bits

    x1 = x1 & 0xff;

    x2 = x2 & 0x300;

    int x = x1 | x2;

    printf("%d ",x);

    wait\_us(500000);

    }

}

Lab 11:

#define BLINKING\_RATE     500ms

int main()

{

    // Initialise the digital pin LED1 as an output

    DigitalOut led(LED1);

    while (true) {

        led = !led;

        ThisThread::sleep\_for(BLINKING\_RATE);

    }

}