```c
/*-----------------WEEK 2 TASK 1------------------*/
#include "stm32f4xx.h"

int main(void){

    // ENABLE CLOCK TO GPIOA AND GPIOC (needed for PA5 internal LED and
PC pins)
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN;

    // SET PC5, PC6, PC8, PC9 AS OUTPUTS (external LEDs)
    GPIOC->MODER |= GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0 |
                    GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0 ;

    // SET PA5 AS OUTPUT (internal LED)
    GPIOA->MODER = GPIO_MODER_MODER5_0;

    // SET PC13 AS INPUT (USER BUTTON)
    GPIOC->MODER &= ~GPIO_MODER_MODER13_Msk;

    // MAIN LOOP
    while(1) {

        // CHECK IF USER BUTTON IS CLICKED (PC13 reads LOW when pressed)
        if(!(GPIOC->IDR & GPIO_IDR_ID13) == 0){

            // Turn ON all external LEDs
            GPIOC->BSRR |= GPIO_BSRR_BS5 | GPIO_BSRR_BS6 |
                           GPIO_BSRR_BS8 | GPIO_BSRR_BS9;

            // Turn OFF internal LED (PA5)
            GPIOA->BSRR |= GPIO_BSRR_BR5;
        }
        else {

            // Turn OFF external LEDs
            GPIOC->BSRR |= GPIO_BSRR_BR5 | GPIO_BSRR_BR6 |
                           GPIO_BSRR_BR8 | GPIO_BSRR_BR9;

            // Turn ON internal LED
            GPIOA->BSRR |= GPIO_BSRR_BS5;
        }
    }
}

/*---------------WEEK 2 TASK 2--------------------*/
#include "stm32f4xx.h"

// SIMPLE DELAY FUNCTION
void delay_ms(uint32_t ms) {
    volatile uint32_t i, j;
    for (i = 0; i < ms; i++) {
        for (j = 0; j < 1000; j++);
        // Crude software delay (depends on CPU speed)
    }
}

int main(void){
```

```
    // ENABLE CLOCK TO GPIOA AND GPIOC
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOCEN;

    // SET PC5, PC6, PC8, PC9 AS OUTPUTS (LEDs)
    GPIOC->MODER |= GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0 |
                    GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0 ;

    while(1) {

        // CHECK IF BUTTON ON PC10 IS *NOT PRESSED*
        // (pull-down → reads 0 normally)
        if(!(GPIOC->IDR & GPIO_IDR_ID10) == 1){

            // LEFT → RIGHT ORDER: PC5 → PC6 → PC8 → PC9

            GPIOC->BSRR |= GPIO_BSRR_BS5;    // Turn ON PC5
            delay_ms(100);
            GPIOC->BSRR |= GPIO_BSRR_BR5;    // Turn OFF PC5

            GPIOC->BSRR |= GPIO_BSRR_BS6;    // Turn ON PC6
            delay_ms(200);
            GPIOC->BSRR |= GPIO_BSRR_BR6;    // Turn OFF PC6

            GPIOC->BSRR |= GPIO_BSRR_BS8;    // Turn ON PC8
            delay_ms(300);
            GPIOC->BSRR |= GPIO_BSRR_BR8;    // Turn OFF PC8

            GPIOC->BSRR |= GPIO_BSRR_BS9;    // Turn ON PC9
            delay_ms(400);
            GPIOC->BSRR |= GPIO_BSRR_BR9;    // Turn OFF PC9
        }
        else {

            // BUTTON PRESSED → REVERSE ORDER (RIGHT → LEFT)
            // PC9 → PC8 → PC6 → PC5

            GPIOC->BSRR |= GPIO_BSRR_BS9;    // Turn ON PC9
            delay_ms(100);
            GPIOC->BSRR |= GPIO_BSRR_BR9;

            GPIOC->BSRR |= GPIO_BSRR_BS8;    // Turn ON PC8
            delay_ms(200);
            GPIOC->BSRR |= GPIO_BSRR_BR8;

            GPIOC->BSRR |= GPIO_BSRR_BS6;    // Turn ON PC6
            delay_ms(300);
            GPIOC->BSRR |= GPIO_BSRR_BR6;

            GPIOC->BSRR |= GPIO_BSRR_BS5;    // Turn ON PC5
            delay_ms(400);
            GPIOC->BSRR |= GPIO_BSRR_BR5;
        }

        delay_ms(200); // Delay between cycles
    }
}

/*--------------------WEEK 3----------------------*/
```

```
int myLed = 6;          // LED connected to digital pin 6
int input = 0;          // Variable to store serial input

void setup() {
    pinMode(myLed,OUTPUT);      // Configure pin 6 as an output

    Serial.begin(115200);       // Start serial communication at 115200
baud
    while(!Serial);             // Wait until serial port is ready (for
boards like Leonardo)

    Serial.println("Serial is ready!");   // Print startup message
}

void loop() {
    // Check if data is available in the Serial buffer
    if(Serial.available() > 0){

        input = Serial.read();   // Read a single character from the
serial port

        // Check if the input is a letter (uppercase or lowercase)
        if(input >= 'a' && input <= 'z' || input >= 'A' && input <= 'Z'){

            Serial.println("Letter Pressed ");   // Print that a letter
was received
            digitalWrite(myLed,HIGH);            // Turn ON the LED
        }
        else{
            Serial.println("Number Pressed ");   // Print that a
number/other char was received
            digitalWrite(myLed,LOW);             // Turn OFF the LED
        }

        Serial.flush();   // Wait for all outgoing serial data to be sent
    }

    delay(5);  // Small delay to avoid reading too fast
}
/*---------------------------WEEK 4 TASK 1 -------------------------------
---------*/

int pin = A0;   // Using analog input A0

void setup() {
  Serial.begin(9600);   // Start serial communication at 9600 baud
  while(!Serial);       // Wait for Serial monitor (important for some
boards)
}

void loop() {
  uint16_t rawADC = analogRead(pin);   // Read 10-bit ADC value (0–1023)

  // Convert ADC value to voltage (3.3V reference, 10-bit resolution)
  float voltage = 3.3 * rawADC / 1023.0;

  // Print raw ADC value
```

```
  Serial.print("ADC Value = ");
  Serial.print(rawADC);
  Serial.println(".");

  // Print corresponding voltage
  Serial.print("Voltage = ");
  Serial.println(voltage);
  Serial.println(" V.");

  delay(1000);   // 1-second delay
}

/*----------------------------WEEK 4 TASK 2--------------------------------
--*/

int tmp36 = A0;   // TMP36 sensor connected to analog pin A0

void setup() {
  Serial.begin(9600);   // Start serial communication
  while(!Serial);       // Wait for Serial monitor to open

  analogReadResolution(12);
  // Set ADC resolution to 12 bits (0-4095)
  // Needed for more precision on boards that support it
}

void loop() {
  uint16_t rawADC = analogRead(tmp36);  // Read 12-bit ADC value (0-4095)

  // Convert ADC value to voltage using 3.3V reference and 12-bit scale
  float voltage = 3.3 * rawADC / 4095.0;

  // TMP36 formula: Temperature (°C) = (Vout - 0.5V) × 100
  float temperature = (voltage - 0.5) * 100;

  // Print calculated temperature
  Serial.print("Room Temperature = ");
  Serial.print(temperature);
  Serial.println(" Degree Celcius.");

  delay(1000);  // 1-second delay
}
/*------------------------WEEK 5  Uart_driver c -------------------------
-*/

#include "uart_driver.h"
#include "stm32f4xx.h"
#include <stdbool.h>

// Timeout value (in ms)
#define UART_TIMEOUT_MS 1000

// Global millisecond counter
static volatile uint32_t sysTick_ms = 0;

// Initialise SysTick for 1ms interrupts
static void SysTick_Init(void) {
// Configure SysTick to interrupt every 1ms
```

```
SysTick->LOAD = (16000000 / 1000) - 1; // 16 MHz / 1000 = 16000 - 1
SysTick->VAL = 0; // Clear current value
SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
SysTick_CTRL_TICKINT_Msk |
SysTick_CTRL_ENABLE_Msk; // Enable SysTick, CPU clock, and interrupt
}

// SysTick Handler (increments ms counter)
void SysTick_Handler(void) {
sysTick_ms++;
}

// Return system tick in ms
inline uint32_t GetTick(void) {
return sysTick_ms;
}
// ===================== UART DRIVER =====================

// GPIO Configuration for USART2
// PA2 - TX (AF7)
// PA3 - RX (AF7)

int __io_putchar(int ch){
UART2_TransmitByte((uint8_t)ch);
return ch;
}

UART_Status_t UART2_Init(UART_Config_t* config) {
// Enable SysTick if not already running
SysTick_Init();

// Enable clocks
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // GPIOA clock
RCC->APB1ENR |= RCC_APB1ENR_USART2EN; // USART2 clock

// Configure GPIO pins PA2 (TX) and PA3 (RX)
GPIOA->MODER &= ~(GPIO_MODER_MODER2 | GPIO_MODER_MODER3);
GPIOA->MODER |= (GPIO_MODER_MODER2_1 | GPIO_MODER_MODER3_1);

GPIOA->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR2 | GPIO_OSPEEDER_OSPEEDR3);

GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR2 | GPIO_PUPDR_PUPDR3);
GPIOA->PUPDR |= GPIO_PUPDR_PUPDR3_0; // Pull-up on RX

GPIOA->AFR[0] &= ~(GPIO_AFRL_AFRL2 | GPIO_AFRL_AFRL3);
GPIOA->AFR[0] |= (7 << GPIO_AFRL_AFSEL2_Pos) | (7 <<
GPIO_AFRL_AFSEL3_Pos);

// Reset USART2
USART2->CR1 = 0;
USART2->CR2 = 0;
USART2->CR3 = 0;

// Configure USART2
USART2->CR1 |= config->word_length | config->parity;
USART2->CR2 |= config->stop_bits;

// Set baud rate
```

```
if (UART2_SetBaudRate(config->baudrate) != UART_OK) {
return UART_ERROR;
}

// Enable USART2, transmitter and receiver
USART2->CR1 |= USART_CR1_UE | USART_CR1_TE | USART_CR1_RE;

return UART_OK;
}


UART_Status_t UART2_DeInit(void) {
USART2->CR1 &= ~USART_CR1_UE;
RCC->APB1ENR &= ~RCC_APB1ENR_USART2EN;
GPIOA->MODER &= ~(GPIO_MODER_MODER2 | GPIO_MODER_MODER3);
return UART_OK;
}


UART_Status_t UART2_SetBaudRate(uint32_t baudrate) {
uint32_t pclk = 16000000; // APB1 at 16 MHz default
uint32_t tmp = (pclk + baudrate/2) / baudrate;

if (tmp < 16 || tmp > 0xFFFF) {
return UART_ERROR;
}
USART2->BRR = tmp;
return UART_OK;
}


UART_Status_t UART2_Transmit(uint8_t* data, uint16_t size, uint32_t
timeout) {
uint32_t start_tick = GetTick();

for (uint16_t i = 0; i < size; i++) {
while (!(USART2->SR & USART_SR_TXE)) {
if ((GetTick() - start_tick) > timeout) {
return UART_TIMEOUT;
}
}
USART2->DR = data[i];
}

while (!(USART2->SR & USART_SR_TC)) {
if ((GetTick() - start_tick) > timeout) {
return UART_TIMEOUT;
}
}

return UART_OK;
}


UART_Status_t UART2_Receive(uint8_t* data, uint16_t size, uint32_t
timeout) {
uint32_t start_tick = GetTick();
```

```
for (uint16_t i = 0; i < size; i++) {
while (!(USART2->SR & USART_SR_RXNE)) {
if ((GetTick() - start_tick) > timeout) {
return UART_TIMEOUT;
}
}
data[i] = (uint8_t)(USART2->DR & 0xFF);
}
return UART_OK;
}


UART_Status_t UART2_TransmitByte(uint8_t data) {
uint32_t start_tick = GetTick();

while (!(USART2->SR & USART_SR_TXE)) {
if ((GetTick() - start_tick) > UART_TIMEOUT_MS) {
return UART_TIMEOUT;
}
}
USART2->DR = data;
return UART_OK;
}


uint8_t UART2_ReceiveByte(void) {
while (!(USART2->SR & USART_SR_RXNE));
return (uint8_t)(USART2->DR & 0xFF);
}


bool UART2_IsDataAvailable(void) {
return (USART2->SR & USART_SR_RXNE) != 0;
}

bool UART2_IsTransmitComplete(void) {
return (USART2->SR & USART_SR_TC) != 0;
}

/*--------------UART_driver. H-----------*/

#ifndef UART_DRIVER_H_
#define UART_DRIVER_H_

#include "stm32f4xx.h"
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
// ================ UART CONFIGURATION ================
typedef struct {
uint32_t baudrate;
uint32_t word_length; // USART_CR1_M (0 = 8 bits, 1 = 9 bits)
uint32_t stop_bits; // USART_CR2_STOP (00 = 1 bit, 10 = 2 bits)
uint32_t parity; // USART_CR1_PCE and USART_CR1_PS
} UART_Config_t;

// Predefined configurations
#define UART_WORDLENGTH_8B 0x00000000U
```

```c
#define UART_WORDLENGTH_9B USART_CR1_M

#define UART_STOPBITS_1 0x00000000U
#define UART_STOPBITS_2 USART_CR2_STOP_1

#define UART_PARITY_NONE 0x00000000U
#define UART_PARITY_EVEN USART_CR1_PCE
#define UART_PARITY_ODD (USART_CR1_PCE | USART_CR1_PS)

// ================= UART STATUS =================
typedef enum {
UART_OK = 0,
UART_ERROR,
UART_BUSY,
UART_TIMEOUT
}
UART_Status_t;

// ================= UART DRIVER =================
UART_Status_t UART2_Init(UART_Config_t* config);
UART_Status_t UART2_DeInit(void);
UART_Status_t UART2_SetBaudRate(uint32_t baudrate);
UART_Status_t UART2_Transmit(uint8_t* data, uint16_t size, uint32_t
timeout);
UART_Status_t UART2_Receive(uint8_t* data, uint16_t size, uint32_t
timeout_ms);
UART_Status_t UART2_TransmitByte(uint8_t data);
uint8_t UART2_ReceiveByte(void);
bool UART2_IsDataAvailable(void);
bool UART2_IsTransmitComplete(void);

// ================= SYSTICK API =================
// SysTick is initialised inside UART2_Init()
// but these are provided for external use if needed.
void SysTick_Handler(void); // ISR (increments millisecond counter)
uint32_t GetTick(void); // Return current millisecond tick

#endif /* UART_DRIVER_H_ */

/*---------------------WEEK 6--------------*/
#include "stm32f4xx.h"

volatile uint32_t tim3_overflow = 0;
volatile uint32_t last_capture = 0;
volatile uint32_t period_ticks = 0;
volatile float period_ms = 0;

void inputCapture_TIMCH4(void){
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
RCC->APB1ENR |= RCC_AHB1ENR_TIM3EN;
GPIOC->MODER &= ~GPIO_MODER_MODER9;
GPIOC->MODER |= GPIO_MODER_MODER9_1;
GPIO->AFR[1] &= ~GPIO_AFRH_AFSEL9;
GPIO->AFR[1] |= 4 << GPIO_AFRH_AFSEL9_Pos;
TIM3->PSC = 1-1;
TIM3->ARR = 0xFFFF; // 4.1 ms
TIM3->CCMR2 |= TIM_CCMR2_CC4S_0;
TIM3->CCER &= ~TIM_CCER_CC4P;
```

```
TIM3->CCER &= ~TIM_CCER_CC4NP;
TIM3->CCER |= TIM_CCER_CC4E;
TIM3->DIER |= TIM_DIER_UIE | TIM_DIER_CC4IE;
TIM3->EGR |= TIM_EGR_UG;
TIM3->CR1 |= TIM_CR1_CEN;
NVIC_EnableIRQ(TIM3_IRQn);
}

int main(void){

while(1){};
}

void TIM3_IRQHandler(void){
if(TIM3->SR & TIM_SR_UIF){
tim3_overflow++;
TIM3->SR &= ~TIM_SR_UIF;
}

if(TIM->SR & TIM_SR_CC4IF){
TIM3_IC_callback();
TIM3->SR &= ~TIM_SR__CC4IF;
}
}

void TIM3_IC_callback(void){
uint32_t capture = TIM3->CCR4;
uint32_t timestamp = (tim3_overflow << 16) | capture;
period_ticks = timestamp - last_capture;
last_capture = timestamp;
period_ms =1000/period_ticks/16000000;
}

/*-----------------WEEK 7 --------------------*/

#include "stm32f4xx.h"

volatile uint32_t tim3_overflow = 0;
volatile uint32_t last_capture = 0;
volatile uint32_t period_ticks = 0;
volatile float period_ms = 0;

void inputCapture_TIMCH4(void){
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
RCC->APB1ENR |= RCC_AHB1ENR_TIM3EN;
GPIOC->MODER &= ~GPIO_MODER_MODER9;
GPIOC->MODER |= GPIO_MODER_MODER9_1;
GPIO->AFR[1] &= ~GPIO_AFRH_AFSEL9;
GPIO->AFR[1] |= 4 << GPIO_AFRH_AFSEL9_Pos;
TIM3->PSC = 1-1;
TIM3->ARR = 0xFFFF; // 4.1 ms
TIM3->CCMR2 |= TIM_CCMR2_CC4S_0;
TIM3->CCER &= ~TIM_CCER_CC4P;
TIM3->CCER &= ~TIM_CCER_CC4NP;
TIM3->CCER |= TIM_CCER_CC4E;
TIM3->DIER |= TIM_DIER_UIE | TIM_DIER_CC4IE;
TIM3->EGR |= TIM_EGR_UG;
TIM3->CR1 |= TIM_CR1_CEN;
```

```
NVIC_EnableIRQ(TIM3_IRQn);
}

int main(void){

while(1){};
}

void TIM3_IRQHandler(void){
if(TIM3->SR & TIM_SR_UIF){
tim3_overflow++;
TIM3->SR &= ~TIM_SR_UIF;
}

if(TIM->SR & TIM_SR_CC4IF){
TIM3_IC_callback();
TIM3->SR &= ~TIM_SR__CC4IF;
}
}

void TIM3_IC_callback(void){
uint32_t capture = TIM3->CCR4;
uint32_t timestamp = (tim3_overflow << 16) | capture;
period_ticks = timestamp - last_capture;
last_capture = timestamp;
period_ms =1000/period_ticks/16000000;
}
```

WEEK 7:

Week 7 TASK1

Create a program using the I2C protocol to read the acceleration from the ADXL345

MAIN.C

```
#include "stm32f4xx.h"
#include <stdio.h>
#include <stdint.h>
#include "adxl345.h"


int16_t x,y,z;
float xg, yg, zg;

extern uint8_t rec_data[6];

int main(void){

adxl_init();
```

```
while(1){
adxl_read_values(DATA_START_ADD);

x = ((rec_data[1] << 8) | rec_data[0]);
y = ((rec_data[3] << 8) | rec_data[2]);
z = ((rec_data[5] << 8) | rec_data[4]);

xg = x*0.0078;
yg = y*0.0078;
zg = z*0.0078;

}
}
```

I2c_drive.c

```
#include "i2c_driver.h"


/*
 * Pinout for the Nucleo-F411RE
 * PB8 ------ SCL
 * PB9 ------ SDA
 * I am using those for I2C1
 */

/*
 * From 18.6.8 in the reference manual
 * CCR = fclk/(2fscl)
 * CCR = 16E6/(2*100E3)
 * CCR = 80 or 0x50
 */

/*
 * From 18.6.9 in the reference manual
 * F = 16 MHz -> T = 62.5 ns
 * F = 100 kHz -> T
 * TRise = 1000 ns/ 62.5 ns +1
 * TRise = 17
 */

#define I2C_100kHz                0x50
#define Sm_MAX_RISE_TIME     17

void I2C1_init(void){


RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;


GPIOB->MODER &= (~GPIO_MODER_MODER8_Msk) | ~(GPIO_MODER_MODER9_Msk);
GPIOB->MODER |= GPIO_MODER_MODER8_1 | GPIO_MODER_MODER9_1;


GPIOB->AFR[1] |= (4U<<GPIO_AFRH_AFSEL8_Pos);
GPIOB->AFR[1] |= (4U<<GPIO_AFRH_AFSEL9_Pos);
```

```
GPIOB->OTYPER |= (GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9);
GPIOB->PUPDR |= (GPIO_PUPDR_PUPD8_0 | GPIO_PUPDR_PUPD9_0);

RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;

I2C1->CR1 |= I2C_CR1_SWRST;

I2C1->CR1 &= ~I2C_CR1_SWRST_Msk;

I2C1->CR2 |= (1U << 4);

I2C1->CCR = I2C_100kHz;

I2C1->TRISE = Sm_MAX_RISE_TIME;

I2C1->CR1 |= I2C_CR1_PE;

}

void I2C1_readByte(char saddr, char maddr, char* data){

volatile int tmp;

while(I2C1->SR2 & I2C_SR2_BUSY){}

I2C1->CR1 |= I2C_CR1_START;

while(!(I2C1->SR1 & I2C_SR1_SB)){}


I2C1->DR = (saddr << 1);

while(!(I2C1->SR1 & I2C_SR1_ADDR)){}

tmp = I2C1->SR2;


I2C1->DR = maddr;


while(!(I2C1->SR1 & I2C_SR1_TXE)){}

I2C1->CR1 |= I2C_CR1_START;

while(!(I2C1->SR1 & I2C_SR1_SB)){}

I2C1->DR = saddr << 1 | 1;

while(!(I2C1->SR1 & I2C_SR1_ADDR)){}

I2C1->CR1 &= ~I2C_CR1_ACK_Msk;

tmp = I2C1->SR2;

I2C1->CR1 |= I2C_CR1_STOP;
```

```
while(!(I2C1->SR1 & I2C_SR1_RXNE)){}

*data++ = I2C1->DR;
(void)tmp;

}

void I2C1_burstRead(char saddr, char maddr, int n, char* data){
volatile int tmp;


while(I2C1->SR2 & I2C_SR2_BUSY){}

I2C1->CR1 |= I2C_CR1_START;

while(!(I2C1->SR1 & I2C_SR1_SB)){}

I2C1->DR = (saddr << 1);


while(!(I2C1->SR1 & I2C_SR1_ADDR)){}

tmp = I2C1->SR2;

while(!(I2C1->SR1 & I2C_SR1_TXE)){}

I2C1->DR = maddr;

while(!(I2C1->SR1 & I2C_SR1_TXE)){}

I2C1->CR1 |= I2C_CR1_START;

while(!(I2C1->SR1 & I2C_SR1_SB)){}

I2C1->DR = saddr << 1 | 1;


while(!(I2C1->SR1 & I2C_SR1_ADDR)){}

tmp = I2C1->SR2;

I2C1->CR1 |= I2C_CR1_ACK;

while(n > 0U){

if(n == 1U){
I2C1->CR1 &= ~I2C_CR1_ACK;

I2C1->CR1 |= I2C_CR1_STOP;

while(!(I2C1->SR1 & I2C_SR1_RXNE)){}

*data++ = I2C1->DR;
break;
}
else{
while(!(I2C1->SR1 & I2C_SR1_RXNE)){}
```

```
*data++ = I2C1->DR;
n--;
}
}

(void)tmp;
}

void I2C1_burstWrite(char saddr, char maddr, int n, char* data){

volatile int tmp;

while(I2C1->SR2 & I2C_SR2_BUSY){}

I2C1->CR1 |= I2C_CR1_START;

while(!(I2C1->SR1 & I2C_SR1_SB)){}

I2C1->DR = (saddr << 1);

while(!(I2C1->SR1 & I2C_SR1_ADDR)){}

tmp = I2C1->SR2;

while(!(I2C1->SR1 & I2C_SR1_TXE)){}

I2C1->DR = maddr;



for(int i = 0; i < n; i++){
while(!(I2C1->SR1 & I2C_SR1_TXE)){}
I2C1->DR = *data++;
}

while(!(I2C1->SR1 & I2C_SR1_BTF)){}

I2C1->CR1 |= I2C_CR1_STOP;
(void)tmp;


}
```

Ic2_drive.h

```
#ifndef I2C_DRIVER_H_
#define I2C_DRIVER_H_

#include "stm32f4xx.h"

void I2C1_init(void);
void I2C1_readByte(char saddr, char maddr, char* data);
void I2C1_burstRead(char saddr, char maddr, int n, char* data);
void I2C1_burstWrite(char saddr, char maddr, int n, char* data);

#endif /* I2C_DRIVER_H_ */
```

adxl345.C

```
#include "adxl345.h"

char data;

uint8_t rec_data[6];

void adxl_read_address(uint8_t reg){

I2C1_readByte(DEVICE_ADDR, reg, &data);
}

void adxl_write(uint8_t reg, char value){

char data[1];
data[0] = value;
I2C1_burstWrite(DEVICE_ADDR, reg, 1, data);

}

void adxl_read_values(uint8_t reg){

I2C1_burstRead(DEVICE_ADDR, reg, 6, (char*)rec_data);

}

void adxl_init(void){

I2C1_init();

adxl_read_address(DEVID);

adxl_write(DATA_FORMAT, FOUR_G);

adxl_write(PWR_CTL, RESET);
adxl_write(PWR_CTL, SET_MEASURE_B);

}
```