Digital Instrumentation

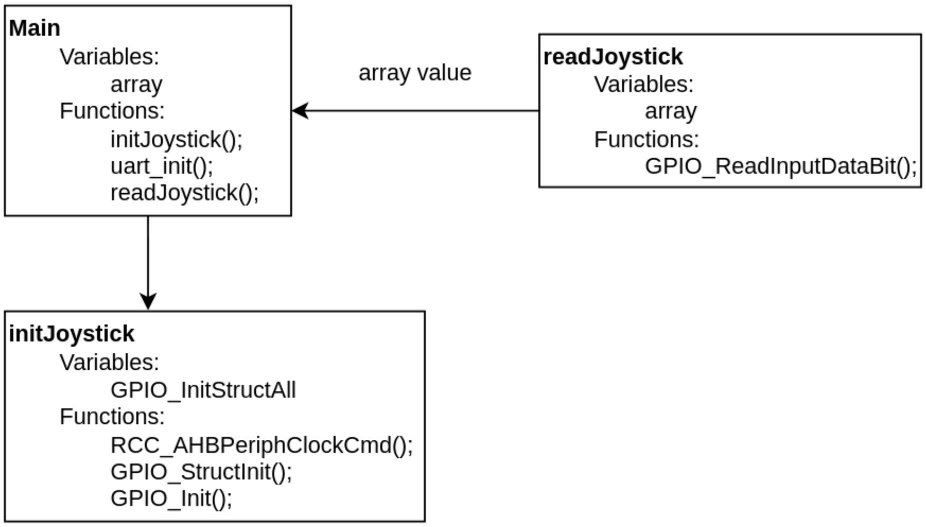
Journal for Exercise 1

Participants: **Tibor Illés**
Benedikt Klingebiel
Christian Jehle

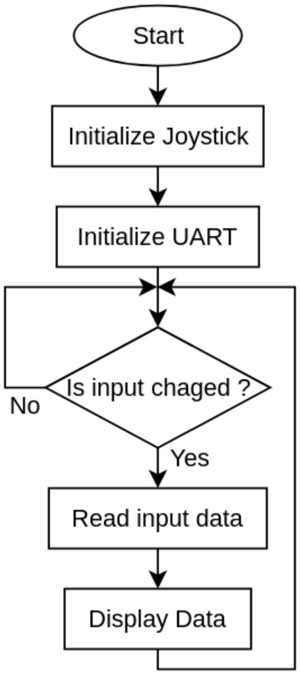
Sep 4, 2022

1. Detecting Joystick Interaction

Block Diagram:



Flowchart:



Function description:

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO ports, the required variables and the UART port. It contains the main program loop that is continuously executed.

initJoystick

Syntax `void initJoystick(void);`

Parameters -

The function initializes the required GPIO ports as input or output (as needed).

readJoystick

Syntax `uint8_t readJoystick(void);`

Parameters -

The function loads the input GPIO values into the array, then returns this variable.

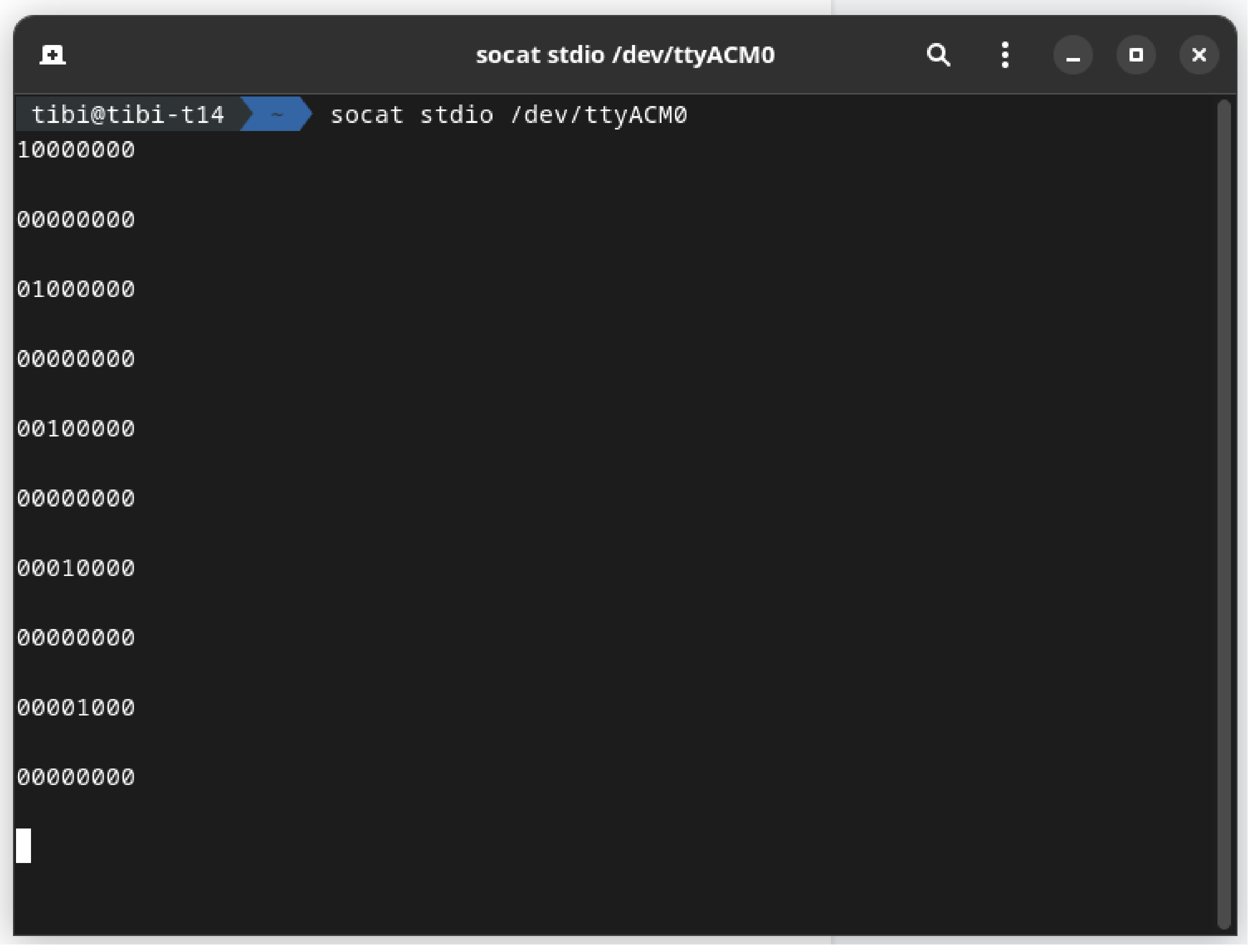
Source code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

void initJoystick(void){
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for GPIO
Port C
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO
Port B
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO
Port A

	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins

	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_4; // Set so the configuration is on
pin 4
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_0; // Set so the configuration is on
pin 0
	GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_5; // Set so the configuration is on
pin 5
	GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_0; // Set so the configuration is on
pin 0
	GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
```

```
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_IN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_1; // Set so the configuration is on
pin 1
	GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
}

uint8_t readJoystick(void){ //function to load input data to the array
	uint8_t array = (GPIO_ReadInputDataBit ( GPIOA, GPIO_Pin_4) << 7) |\
			(GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_0) << 6) |\
			(GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_1) << 5) |\
			(GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_0) << 4) |\
			(GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_5) << 3) |\
			(0 << 2) | (0 << 1) | (0 << 0); //Create array for serial data
	return(array);
}

int main(void)
{
	initJoystick(); //GPIO initialization
	uint8_t array=0; //variable to display
	uart_init( 9600 ); // Initialize USB serial at 9600 baud

	while(1){

		if(array!=readJoystick()){ //display only when input change occurs
			array=readJoystick(); //loads the input value
			for(int i=7; i>=0; i--) printf("%1d", (array & (1 << i)) >> i
); //print the input value on the serial terminal
			printf("\n");
		}
	}
}
```

Results:

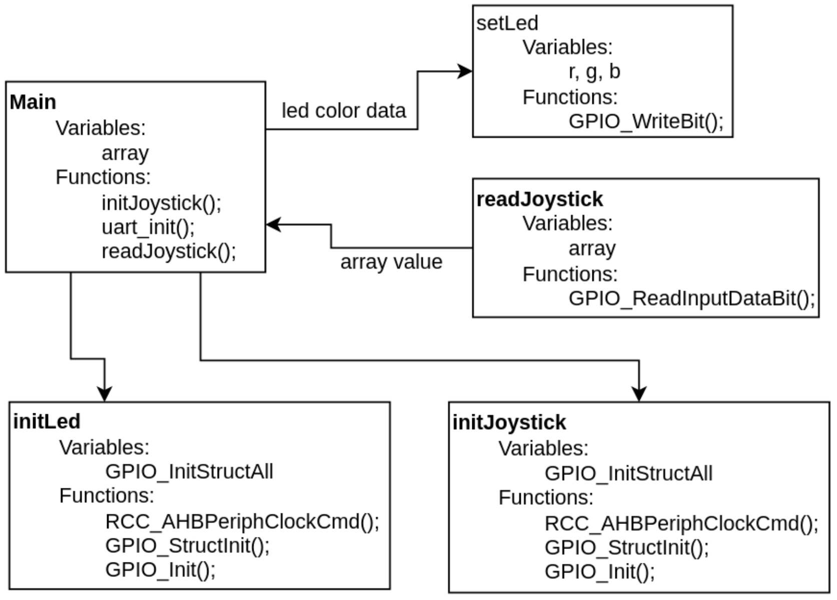
On the serial terminal it can clearly be seen that whenever the input changes, the array containing the right input value is displayed.



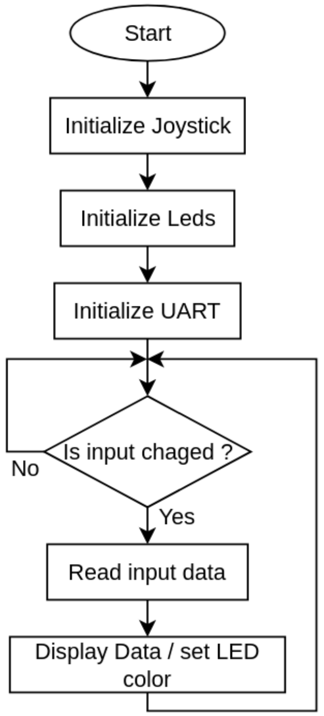
```
tibi@tibi-t14  ~  socat stdio /dev/ttyACM0
10000000

00000000

01000000

00000000

00100000

00000000

00010000

00000000

00001000

00000000
```

2. Controlling the RGB LED

Block Diagram:



Flowchart:



Function description: *(Only the changed functions are indicated)*

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO ports, the required variables and the UART port. It contains the main program loop that is continuously executed..

initLed

Syntax `void initLed(void);`

Parameters -

The function initializes the required GPIO ports as output.

setLed

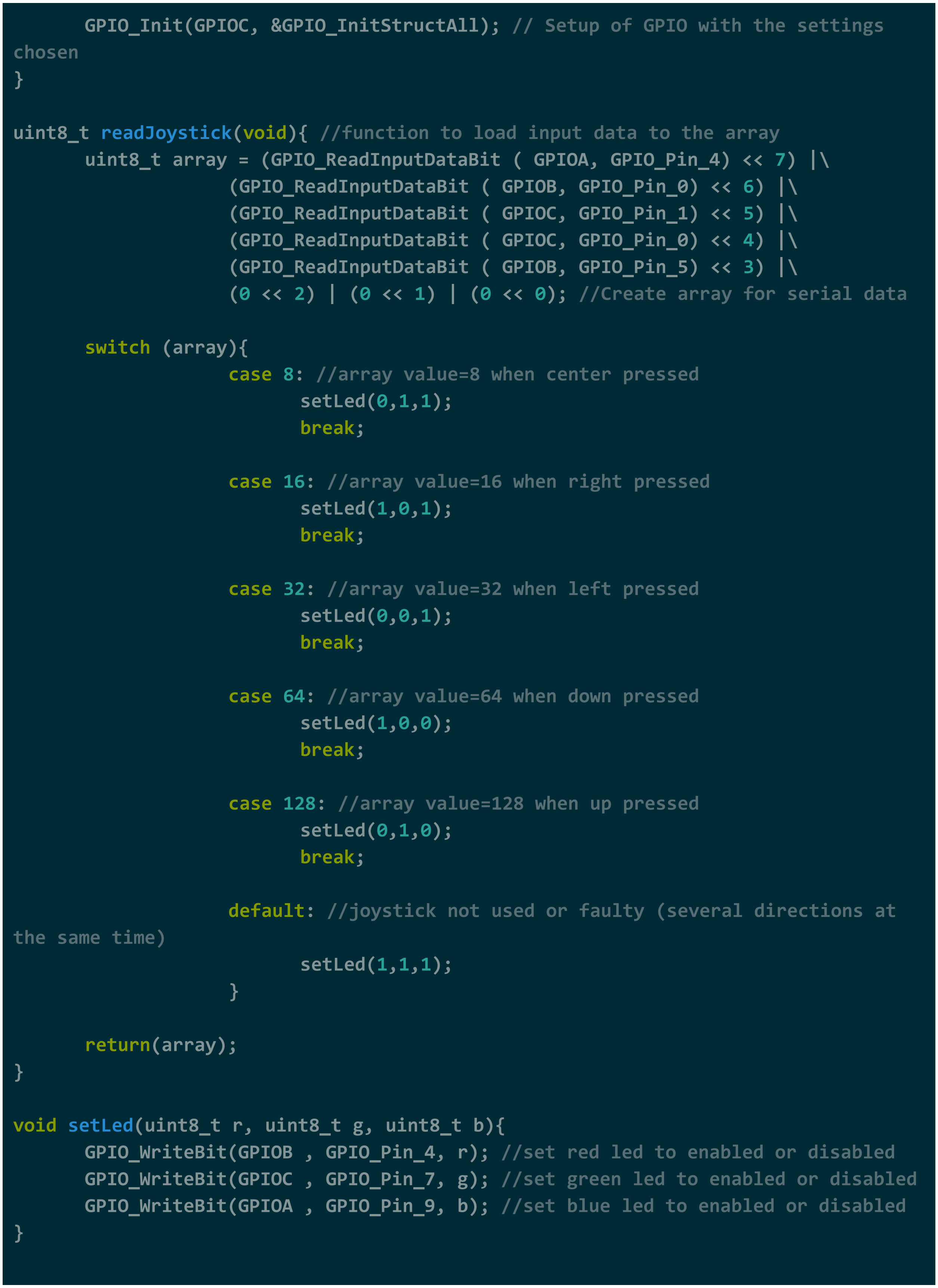
Syntax `void setLed(uint8_t r, uint8_t g, uint8_t b);`

Parameters uint8_t r, g, b

The function sets the GPIO outputs to set required LED colors.

Source code: *(only the changed functions are indicated)*

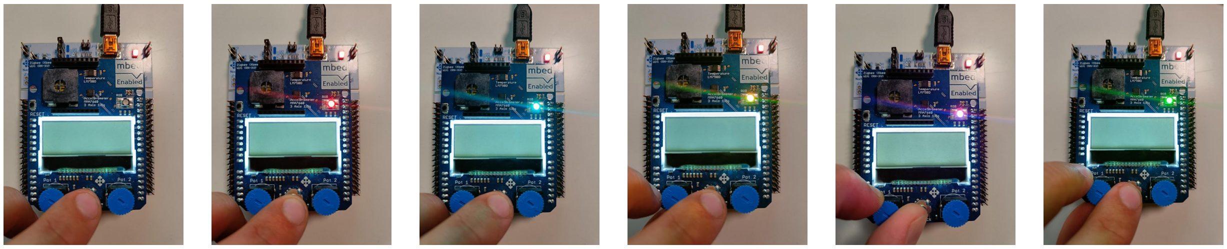
```c
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

void initJoystick(void){...

void initLed(void){
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for GPIO Port C
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO Port B
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO Port A

	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting pins

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_9; // Set so the configuration is on pin 9
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_4; // Set so the configuration is on pin 4
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_7; // Set so the configuration is on pin 7
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
```



```
    GPIO_Init(GPIOC, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
}

uint8_t readJoystick(void){ //function to load input data to the array
    uint8_t array = (GPIO_ReadInputDataBit ( GPIOA, GPIO_Pin_4) << 7) |\
            (GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_0) << 6) |\
            (GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_1) << 5) |\
            (GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_0) << 4) |\
            (GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_5) << 3) |\
            (0 << 2) | (0 << 1) | (0 << 0); //Create array for serial data

    switch (array){
            case 8: //array value=8 when center pressed
                setLed(0,1,1);
                break;

            case 16: //array value=16 when right pressed
                setLed(1,0,1);
                break;

            case 32: //array value=32 when left pressed
                setLed(0,0,1);
                break;

            case 64: //array value=64 when down pressed
                setLed(1,0,0);
                break;

            case 128: //array value=128 when up pressed
                setLed(0,1,0);
                break;

            default: //joystick not used or faulty (several directions at
the same time)
                setLed(1,1,1);
            }

    return(array);
}

void setLed(uint8_t r, uint8_t g, uint8_t b){
    GPIO_WriteBit(GPIOB , GPIO_Pin_4, r); //set red led to enabled or disabled
    GPIO_WriteBit(GPIOC , GPIO_Pin_7, g); //set green led to enabled or disabled
    GPIO_WriteBit(GPIOA , GPIO_Pin_9, b); //set blue led to enabled or disabled
}
```

```
int main(void)
{
	initJoystick();
	initLed();
	uint8_t array=0;
	uart_init( 9600 ); // Initialize USB serial at 9600 baud
	setLed(1,1,1); //set all leds to off state

	while(1){ // set led color depending on joystick position

		if(array != readJoystick()){ //display only when input change occurs
			array=readJoystick();
			for(int i=7; i>=0; i--) printf("%1d", (array & (1 << i)) >> i
);
			printf("\n");
		}
	}
}
```

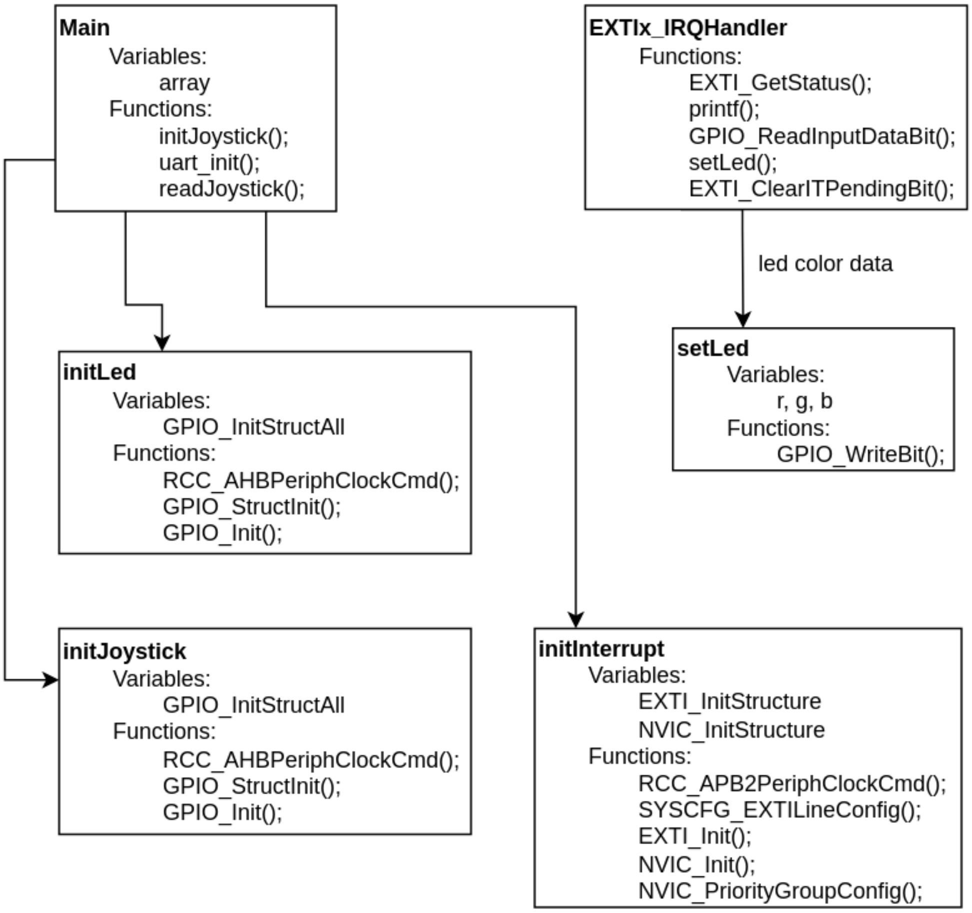
Results:



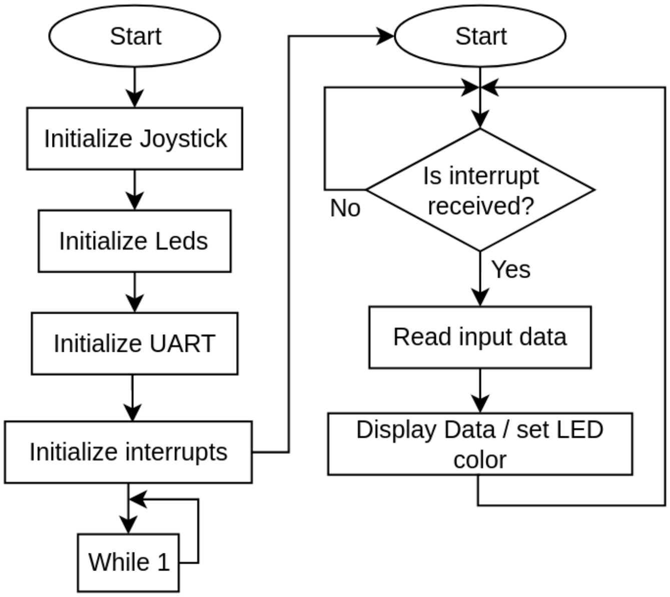
In the photos it can be seen that the RGB LED color changes based on the direction that the joystick is pressed (and held continuously).

3. Interrupt based input

Block Diagram:



Flowchart:



Function description: *(Only the changed functions are indicated)*

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO ports, the required variables, the interrupts and the UART port. It contains the main program loop that is continuously executed, but everything is handled by interrupt routines.

initInterrupt

Syntax `void initInterrupt(void);`

Parameters -

The function initializes the required interrupts.

EXTIx_IRQHandler

Syntax `void EXTIx_IRQHandler(void);`

Parameters -

The function is activated when an interrupt occurs. Changes the RGB LED colors based on the state of the input.

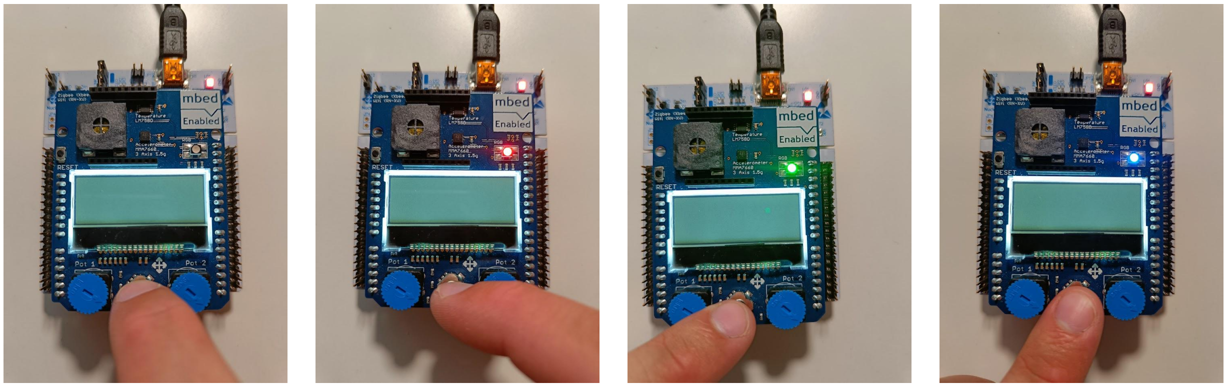
Source code: *(only the changed functions are indicated)*

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

void initJoystick(void){
        . . .
}

void initLed(void){
        . . .
}

void initInterrupt(void){
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,ENABLE);
        SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB,EXTI_PinSource5); // sets port B pin 5 to the IRQ
        // define and set setting for EXTI
        EXTI_InitTypeDef EXTI_InitStructure;
        EXTI_InitStructure.EXTI_Line = EXTI_Line5;
        EXTI_InitStructure.EXTI_LineCmd = ENABLE;
        EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
        EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
        EXTI_Init(&EXTI_InitStructure);
        // setup NVIC
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
        NVIC_InitTypeDef NVIC_InitStructure;
        NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_Init(&NVIC_InitStructure);

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,ENABLE);
        SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC,EXTI_PinSource0); // sets port C pin 0 to the IRQ
        // define and set setting for EXTI
        EXTI_InitStructure.EXTI_Line = EXTI_Line0;
        EXTI_InitStructure.EXTI_LineCmd = ENABLE;
        EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
        EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
        EXTI_Init(&EXTI_InitStructure);
        // setup NVIC
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
        NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_Init(&NVIC_InitStructure);

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,ENABLE);
        SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource4); // sets port A pin 4 to the IRQ
        // define and set setting for EXTI
        EXTI_InitStructure.EXTI_Line = EXTI_Line4;
        EXTI_InitStructure.EXTI_LineCmd = ENABLE;
        EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
        EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
        EXTI_Init(&EXTI_InitStructure);
        // setup NVIC
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
        NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_Init(&NVIC_InitStructure);

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,ENABLE);
        SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC,EXTI_PinSource1); // sets port C pin 1 to the IRQ
        // define and set setting for EXTI
        EXTI_InitStructure.EXTI_Line = EXTI_Line1;
```

```
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&EXTI_InitStructure);
    // setup NVIC
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&NVIC_InitStructure);
}

void setLed(uint8_t r, uint8_t g, uint8_t b){
    . . .
}

void EXTI9_5_IRQHandler(void){ //interrupt handler for joystick center input
    if(EXTI_GetITStatus(EXTI_Line5) != RESET){
        printf("Right : %d | Up : %d | Center : %d | Left : %d | Down :
%d\n",\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0),\
            GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_5),\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_0)); //prints the
direction data
        setLed(1,1,1); //set led to off
        EXTI_ClearITPendingBit(EXTI_Line5); //clear pending bit
    }
}

void EXTI0_IRQHandler(void){ //interrupt handler for joystick right input
    if(EXTI_GetITStatus(EXTI_Line0) != RESET){
        printf("Right : %d | Up : %d | Center : %d | Left : %d | Down :
%d\n",\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0),\
            GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_5),\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_0)); //prints the
direction data
        setLed(1,0,1); //set led to green color
        EXTI_ClearITPendingBit(EXTI_Line0); //clear pending bit
    }
}
```

```
void EXTI4_IRQHandler(void){ //interrupt handler for joystick up input
    if(EXTI_GetITStatus(EXTI_Line4) != RESET){
        printf("Right : %d | Up : %d | Center : %d | Left : %d | Down :
%d\n",\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0),\
            GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_5),\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_0)); //prints the
direction data
        setLed(1,1,0); //set led to blue color
        EXTI_ClearITPendingBit(EXTI_Line4); //clear pending bit
    }
}

void EXTI1_IRQHandler(void){ //interrupt handler for joystick left input
    if(EXTI_GetITStatus(EXTI_Line1) != RESET){
        printf("Right : %d | Up : %d | Center : %d | Left : %d | Down :
%d\n",\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0),\
            GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_5),\
            GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_1),\
            GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_0)); //prints the
direction data
        setLed(0,1,1); //set led to red color
        EXTI_ClearITPendingBit(EXTI_Line1); //clear pending bit
    }
}

int main(void)
{
    initJoystick(); //initialize the input pins
    initLed(); //initialize the output pins
    setLed(1,1,1); //turn off all leds
    initInterrupt(); //initialize the interrupts
    uart_init( 9600 ); // Initialize USB serial at 9600 baud

    while(1){

    }
}
```

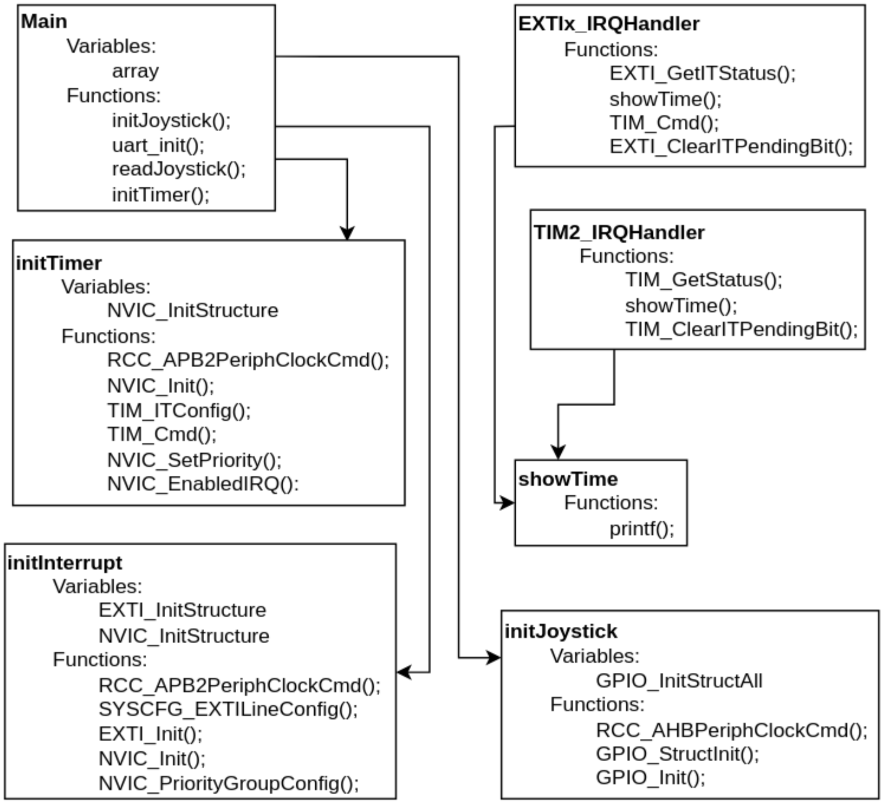
Results:



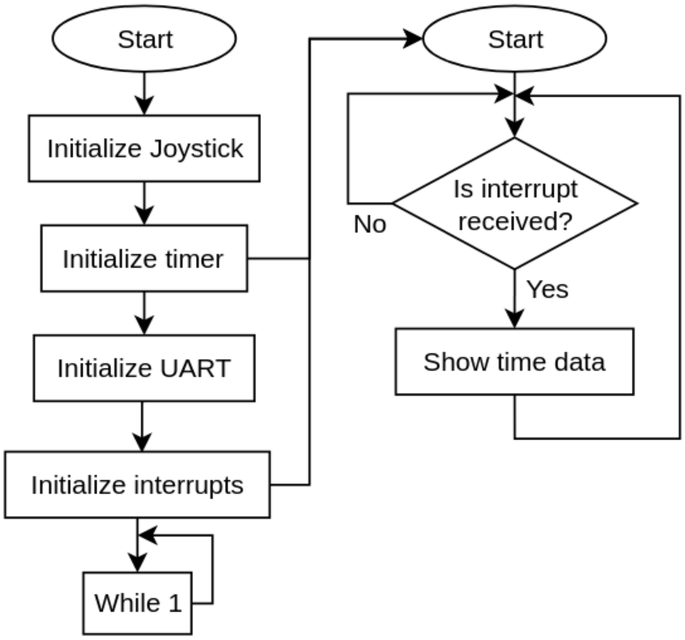
In the photos it can be seen that the color of the RGB LED changes based on the direction the joystick is pressed. The direction is detected by four separate interrupt routines.

4. Timers

Block Diagram:



Flowchart:



Function description: *(Only the changed functions are indicated)*

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO ports, the required variables, the interrupts and the UART port. It contains the main program loop that is continuously executed, but everything is handled by interrupt routines.

initTimer

Syntax `void initTimer(void);`

Parameters -

The function initializes the required timer.

EXTIx_IRQHandler

Syntax `void EXTIx_IRQHandler(void);`

Parameters -

The function is activated when an interrupt occurs.

TIM2_IRQHandler

Syntax `void TIM2_IRQHandler(void);`

Parameters -

The function is activated when a timer interrupt occurs. Updates the time data every 10 milliseconds.

showTime

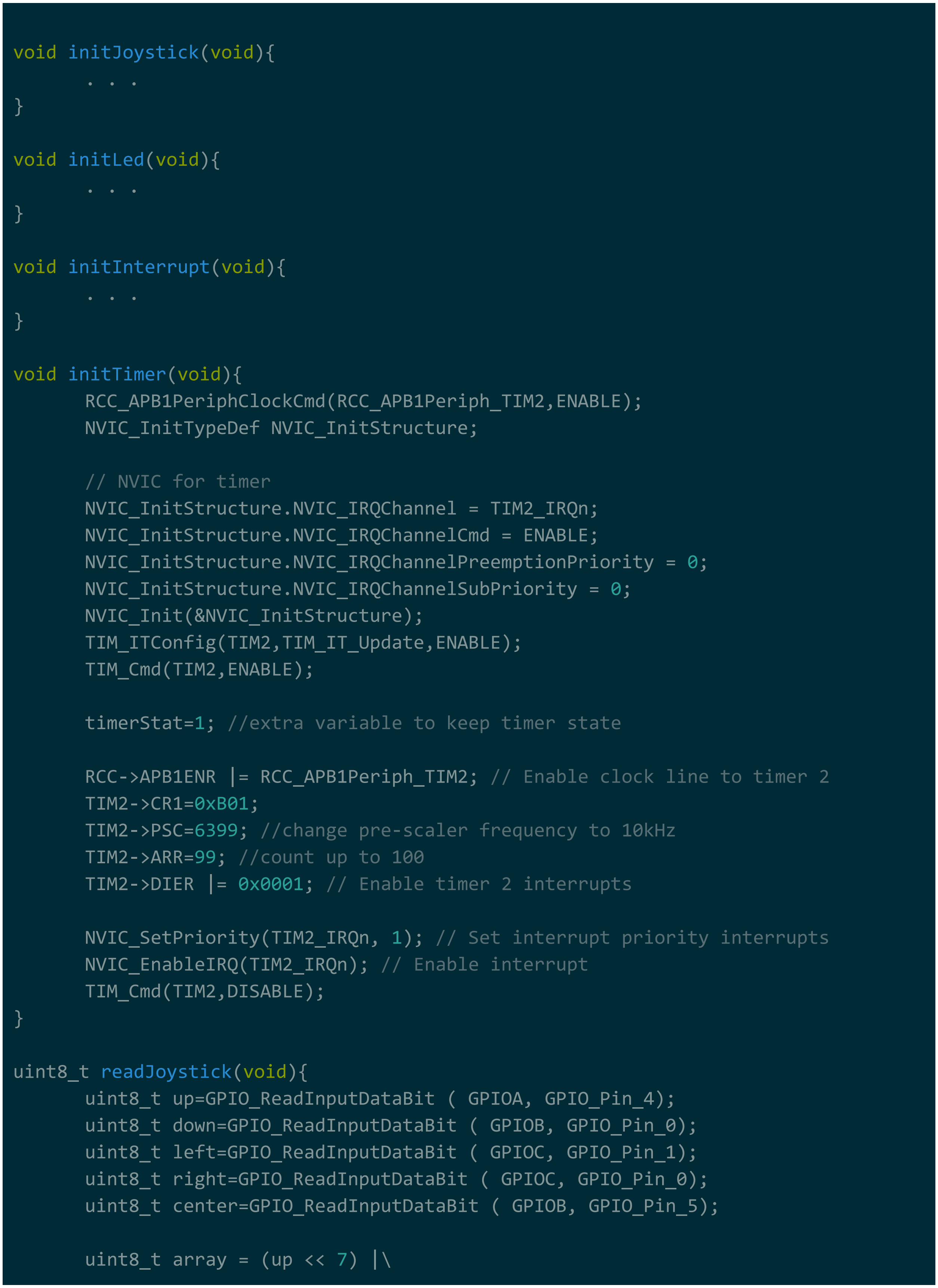
Syntax `void showTime(void);`

Parameters -

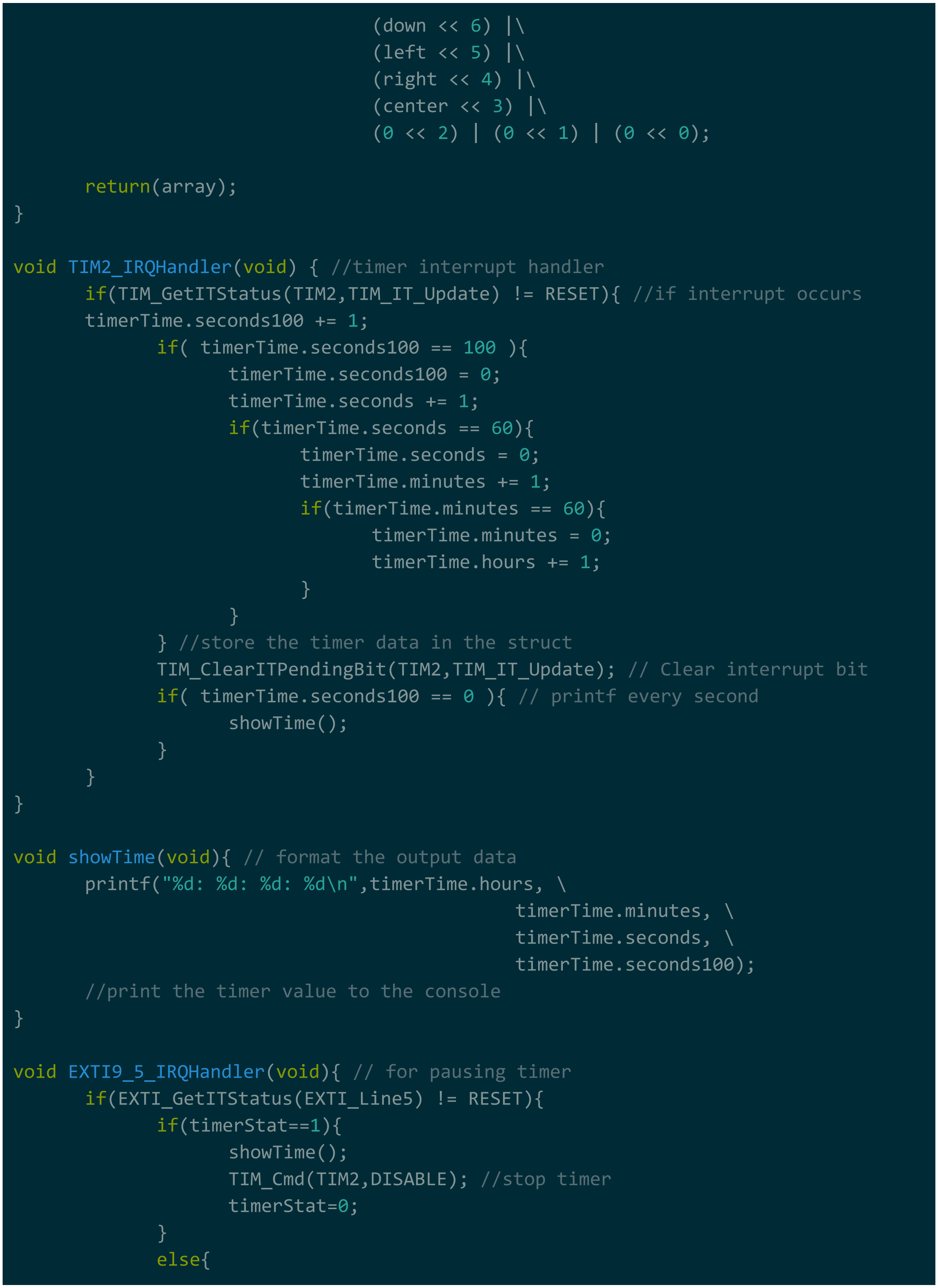
The function prints the current time to the console.

Source code: *(only the changed functions are indicated)*

```c
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

struct Time { //struct for timer data
	uint8_t volatile hours;
	uint8_t volatile minutes;
	uint8_t volatile seconds;
	uint8_t volatile seconds100;
};

struct Time timerTime; //declaration of timer struct

uint8_t timerStat; //variable for timer state
```

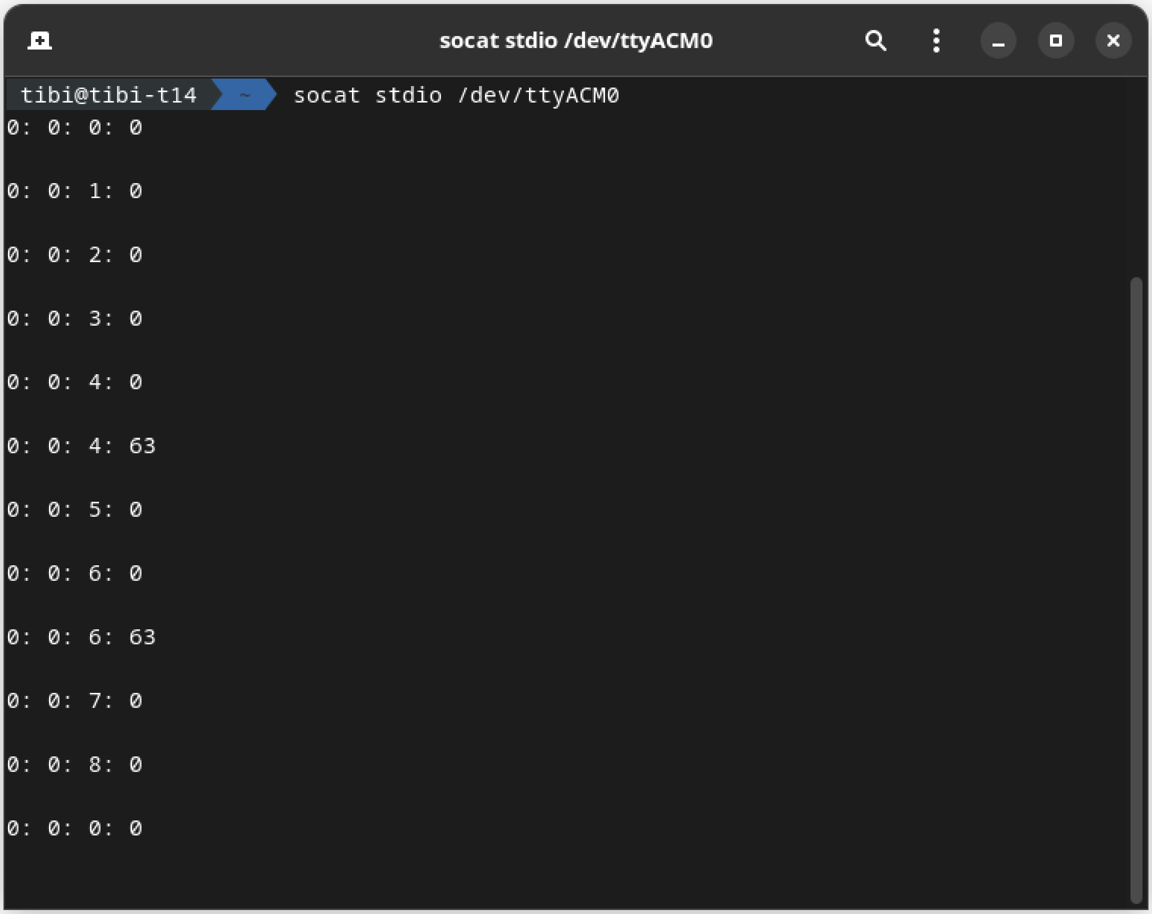


```c
void initJoystick(void){
       . . .
}

void initLed(void){
       . . .
}

void initInterrupt(void){
       . . .
}

void initTimer(void){
       RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
       NVIC_InitTypeDef NVIC_InitStructure;

       // NVIC for timer
       NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
       NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
       NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
       NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
       NVIC_Init(&NVIC_InitStructure);
       TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
       TIM_Cmd(TIM2,ENABLE);

       timerStat=1; //extra variable to keep timer state

       RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
       TIM2->CR1=0xB01;
       TIM2->PSC=6399; //change pre-scaler frequency to 10kHz
       TIM2->ARR=99; //count up to 100
       TIM2->DIER |= 0x0001; // Enable timer 2 interrupts

       NVIC_SetPriority(TIM2_IRQn, 1); // Set interrupt priority interrupts
       NVIC_EnableIRQ(TIM2_IRQn); // Enable interrupt
       TIM_Cmd(TIM2,DISABLE);
}

uint8_t readJoystick(void){
       uint8_t up=GPIO_ReadInputDataBit ( GPIOA, GPIO_Pin_4);
       uint8_t down=GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_0);
       uint8_t left=GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_1);
       uint8_t right=GPIO_ReadInputDataBit ( GPIOC, GPIO_Pin_0);
       uint8_t center=GPIO_ReadInputDataBit ( GPIOB, GPIO_Pin_5);

       uint8_t array = (up << 7) |\
```



```
                                        (down << 6) |\
                                        (left << 5) |\
                                        (right << 4) |\
                                        (center << 3) |\
                                        (0 << 2) | (0 << 1) | (0 << 0);

        return(array);
}

void TIM2_IRQHandler(void) { //timer interrupt handler
        if(TIM_GetITStatus(TIM2,TIM_IT_Update) != RESET){ //if interrupt occurs
        timerTime.seconds100 += 1;
                if( timerTime.seconds100 == 100 ){
                        timerTime.seconds100 = 0;
                        timerTime.seconds += 1;
                        if(timerTime.seconds == 60){
                                timerTime.seconds = 0;
                                timerTime.minutes += 1;
                                if(timerTime.minutes == 60){
                                        timerTime.minutes = 0;
                                        timerTime.hours += 1;
                                }
                        }
                } //store the timer data in the struct
                TIM_ClearITPendingBit(TIM2,TIM_IT_Update); // Clear interrupt bit
                if( timerTime.seconds100 == 0 ){ // printf every second
                        showTime();
                }
        }
}

void showTime(void){ // format the output data
        printf("%d: %d: %d: %d\n",timerTime.hours, \
                                                        timerTime.minutes, \
                                                        timerTime.seconds, \
                                                        timerTime.seconds100);
        //print the timer value to the console
}

void EXTI9_5_IRQHandler(void){ // for pausing timer
        if(EXTI_GetITStatus(EXTI_Line5) != RESET){
                if(timerStat==1){
                        showTime();
                        TIM_Cmd(TIM2,DISABLE); //stop timer
                        timerStat=0;
                }
                else{
                        TIM_Cmd(TIM2,ENABLE); //start timer
                        timerStat=1;
                }
                EXTI_ClearITPendingBit(EXTI_Line5);
        }
}

void EXTI4_IRQHandler(void){ // reset the timer
        if(EXTI_GetITStatus(EXTI_Line4) != RESET){
                timerTime.seconds100 = 0; // set timer struct to 0
                timerTime.seconds = 0;
                timerTime.minutes = 0;
                timerTime.hours = 0;
                TIM_Cmd(TIM2,DISABLE);
                timerStat=0;
                showTime(); //printf the current time
                EXTI_ClearITPendingBit(EXTI_Line4);
        }
}

int main(void)
{
        initJoystick();
        initLed();
        initInterrupt();
        initTimer();
        uart_init( 9600 ); // Initialize USB serial at 9600 baud

        while(1){

        }
}
```

Results:



```
tibi@tibi-t14  ~  socat stdio /dev/ttyACM0
0: 0: 0: 0

0: 0: 1: 0

0: 0: 2: 0

0: 0: 3: 0

0: 0: 4: 0

0: 0: 4: 63

0: 0: 5: 0

0: 0: 6: 0

0: 0: 6: 63

0: 0: 7: 0

0: 0: 8: 0

0: 0: 0: 0
```

On the picture the following sequence can be seen:

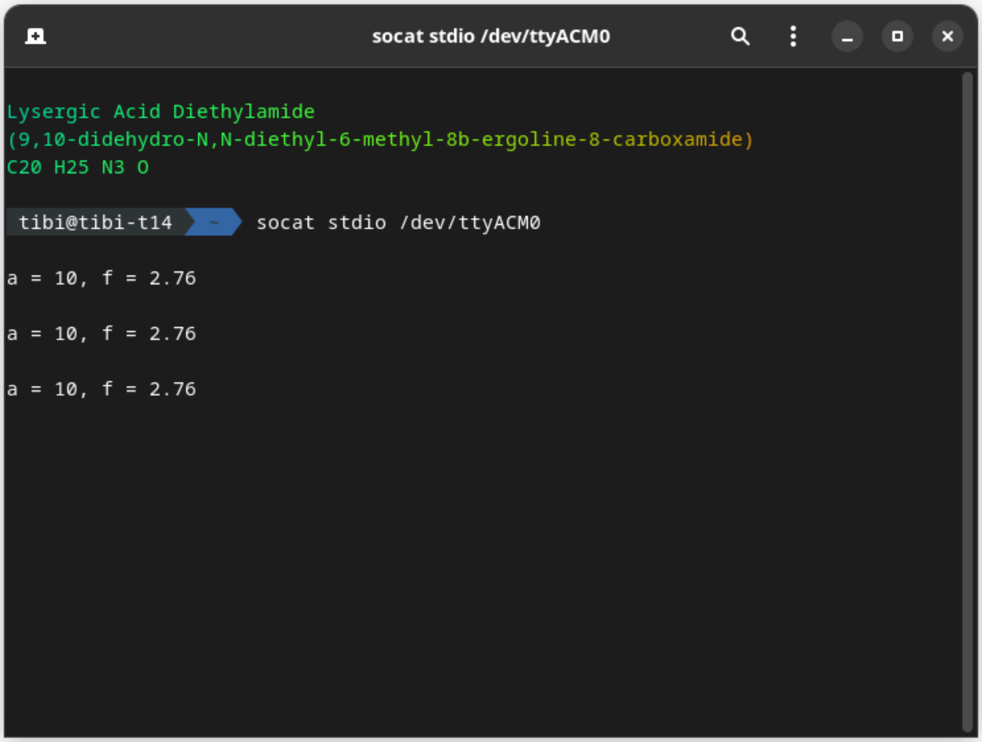
-reset timer
-start timer
-stop timer
-start timer
-stop timer
-start timer
-reset timer

5. Printing text

Source code:

```c
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"       // Input/output library for this course

int main(void)
{
        uart_init(9600); //init uart
        uint8_t a = 10;
        float f = 2.7645;
        char str[7];
        sprintf(str, "a = %2d, f = %0.2f", a, f); // format the string
        printf("%s\n",str); // print to uart console
        while(1){
        }
}
```

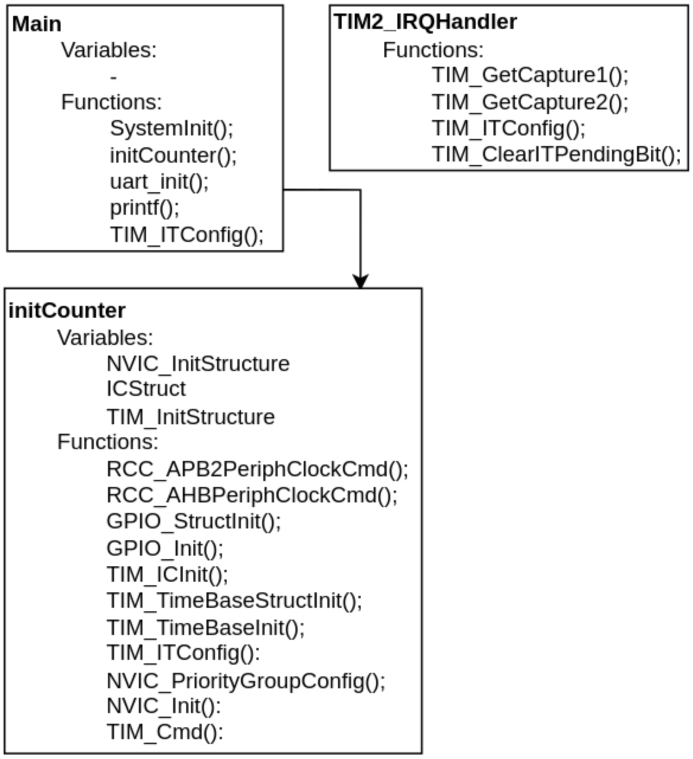
Results:



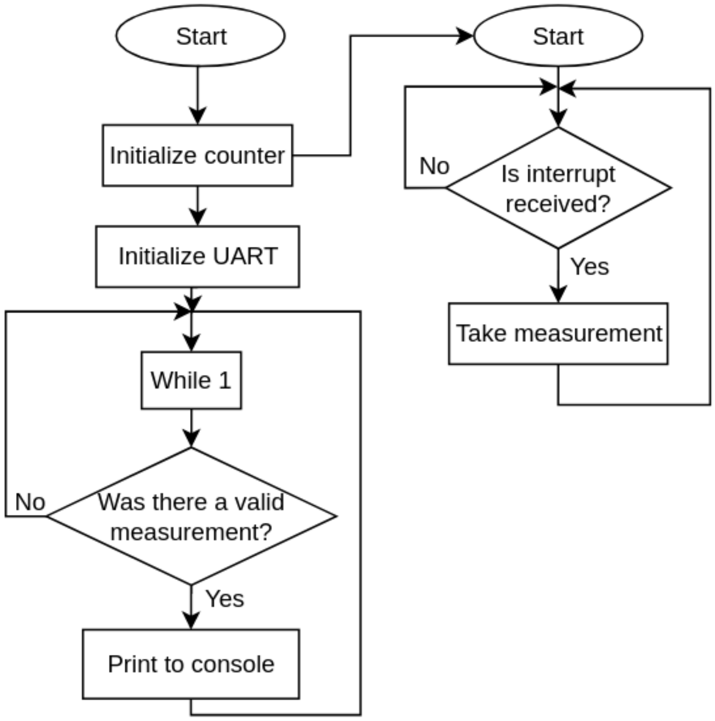
```
Lysergic Acid Diethylamide
(9,10-didehydro-N,N-diethyl-6-methyl-8b-ergoline-8-carboxamide)
C20 H25 N3 O

 tibi@tibi-t14  ~  socat stdio /dev/ttyACM0

a = 10, f = 2.76

a = 10, f = 2.76

a = 10, f = 2.76
```

6. Pulse Width Measurement

Block Diagram:



Flowchart:



Function description: *(Only the changed functions are indicated)*

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the Timer, the required variables, the interrupts and the UART port. It contains the main program loop that is continuously executed and prints the results to the UART console, but everything is handled by interrupt routines.

initCounter

Syntax `void initCounter(void);`

Parameters -

The function initializes the timer used for Pulse Width measurements.

TIM2_IRQHandler

Syntax `void TIM2_IRQHandler(void);`

Parameters -

The function is activated whenever a PWM input interrupt happens and saves the measured times for later evaluation.

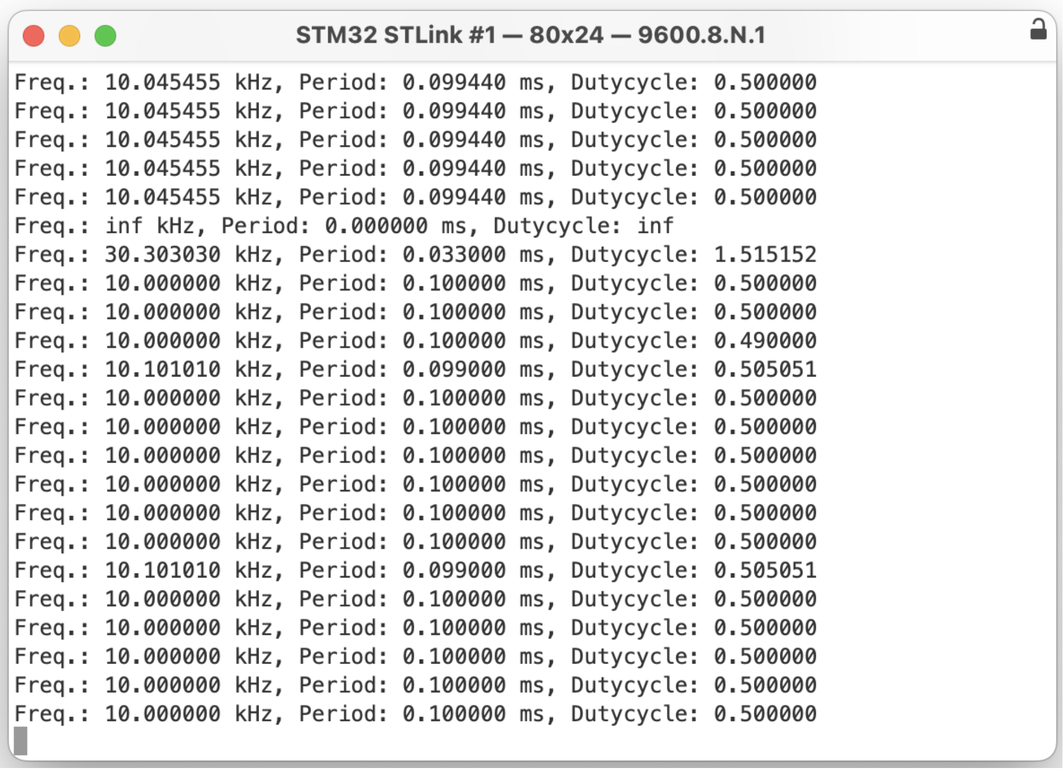
Source code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

int ICValue1 = 0;
int ICValue2 = 0;
int ICValid = 0;

void initCounter(void){
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE); //Enable clock for timer
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO Port B

      GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting pins
      GPIO_StructInit(&GPIO_InitStructAll);
      // Then set things that are not default.
      GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF;
      GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_0;
      GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN;
      GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings chosen

      GPIOA->AFR[0] |= GPIO_AF_1; //Sets pin y at port x to alternative function z
```

```
       //ICInitStruct
       TIM_ICInitTypeDef ICStruct;
       ICStruct.TIM_ICFilter = 0x0;
       ICStruct.TIM_ICPrescaler = 0x0;
       TIM_ICInit(TIM2, &ICStruct);

       // Timer
       TIM_TimeBaseInitTypeDef TIM_InitStructure;
       TIM_TimeBaseStructInit(&TIM_InitStructure);
       TIM_InitStructure.TIM_ClockDivision = 0;
       TIM_InitStructure.TIM_Period = 0xFFFF; //set the maximum period
       TIM_InitStructure.TIM_Prescaler = 63; //for 1MHz counting frequency
       TIM_TimeBaseInit(TIM2,&TIM_InitStructure);

       // Set Input Capture in TIM2 to PWM mode
       TIM2->CCMR1 = TIM_CCMR1_CC2S_1 | TIM_CCMR1_CC1S_0; //CC1 channel as input,
IC1 is mapped on TI1
       TIM2->SMCR = TIM_SMCR_TS_2 | TIM_SMCR_TS_0;        //set trigger to TI1FP1
       TIM2->SMCR |= TIM_SMCR_SMS_2;                      //slave to reset mode
       TIM2->CCER = TIM_CCER_CC2P | TIM_CCER_CC1E | TIM_CCER_CC2E; //enable capture
and compare modules

       TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
       NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0); //set priority
       NVIC_InitTypeDef NVIC_InitStructure;
       NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
       NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
       NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
       NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
       NVIC_Init(&NVIC_InitStructure); // NVIC for timer
       TIM_Cmd(TIM2,ENABLE); //enable the timer
}

void TIM2_IRQHandler(void){
       TIM_ClearITPendingBit(TIM2, TIM_IT_CC2); //clear pending bit
       ICValue1 = TIM_GetCapture1(TIM2); // save period
       ICValue2 = TIM_GetCapture2(TIM2); // save duty cycle
       ICValid = 1; // save that the measurement happened
       TIM_ITConfig(TIM2, TIM_IT_CC2, DISABLE); // disable
}

int main(void)
{
       SystemInit(); //reset the RCC clock configuration to the default
       initCounter(); //initialize the timer
       uart_init( 9600 ); // Initialize USB serial at 9600 baud
```

```
    while(1)
    {
        if (ICValid) // printf if the measurement is valid
        {
            printf("Freq.: %f kHz, Period: %f ms, Dutycycle: %f \%\n",
                    1000/(double)ICValue1,  //calculate the frequency
                                 //cast type to double for division
                                 ICValue1*1e-3,  //calculate the period
                                 (double)((double)ICValue2/(double)ICValue1));
//calculate the duty cycle
            ICValid = 0; //set to 0 for new measurement
            TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE); // enable the timer
        }
        for (int i = 0; i < 10000000; i++)
            ;
       } //delay for slower printing
}
```

Results:

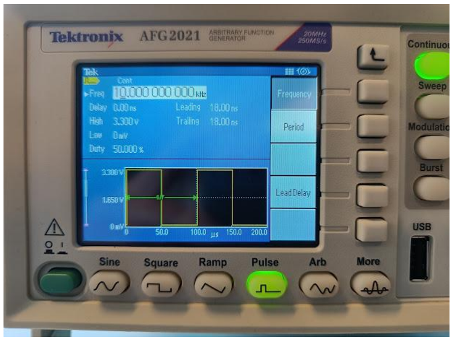


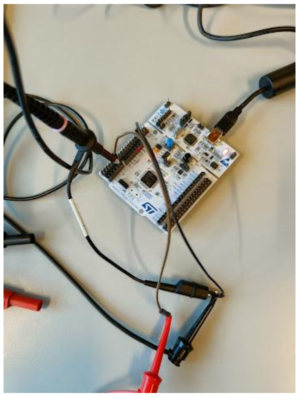
STM32 STLink #1 — 80x24 — 9600.8.N.1

```
Freq.: 10.045455 kHz, Period: 0.099440 ms, Dutycycle: 0.500000
Freq.: 10.045455 kHz, Period: 0.099440 ms, Dutycycle: 0.500000
Freq.: 10.045455 kHz, Period: 0.099440 ms, Dutycycle: 0.500000
Freq.: 10.045455 kHz, Period: 0.099440 ms, Dutycycle: 0.500000
Freq.: 10.045455 kHz, Period: 0.099440 ms, Dutycycle: 0.500000
Freq.: inf kHz, Period: 0.000000 ms, Dutycycle: inf
Freq.: 30.303030 kHz, Period: 0.033000 ms, Dutycycle: 1.515152
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.490000
Freq.: 10.101010 kHz, Period: 0.099000 ms, Dutycycle: 0.505051
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.101010 kHz, Period: 0.099000 ms, Dutycycle: 0.505051
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
Freq.: 10.000000 kHz, Period: 0.100000 ms, Dutycycle: 0.500000
```

First we let the program make some measurements with a 10kHz input signal. After these measurements we were able to calculate the error of the timer and correct the values by the percentage of the error. After modifying the values by the correct error we took some more measurements. After the correction the measured frequency, period and duty cycle was the same as what we set on the function generator.

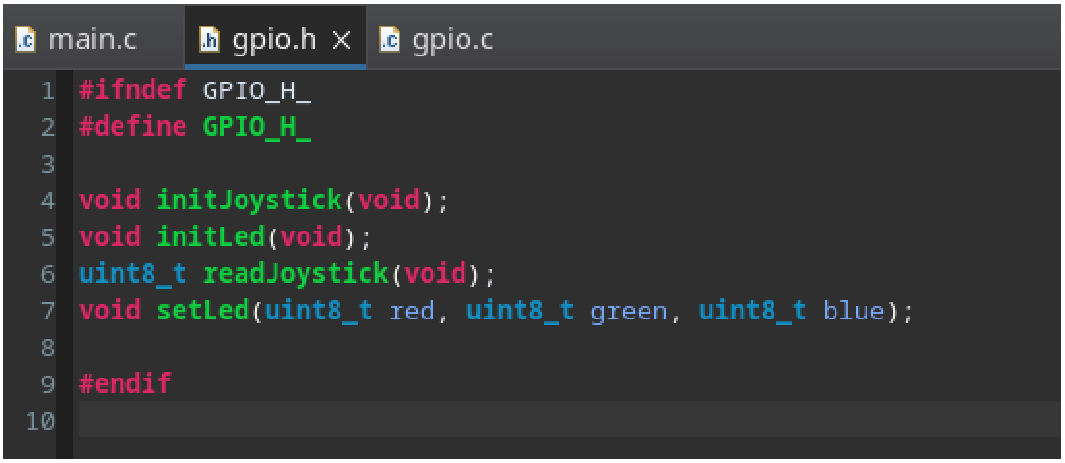
Pictures of the setup used:



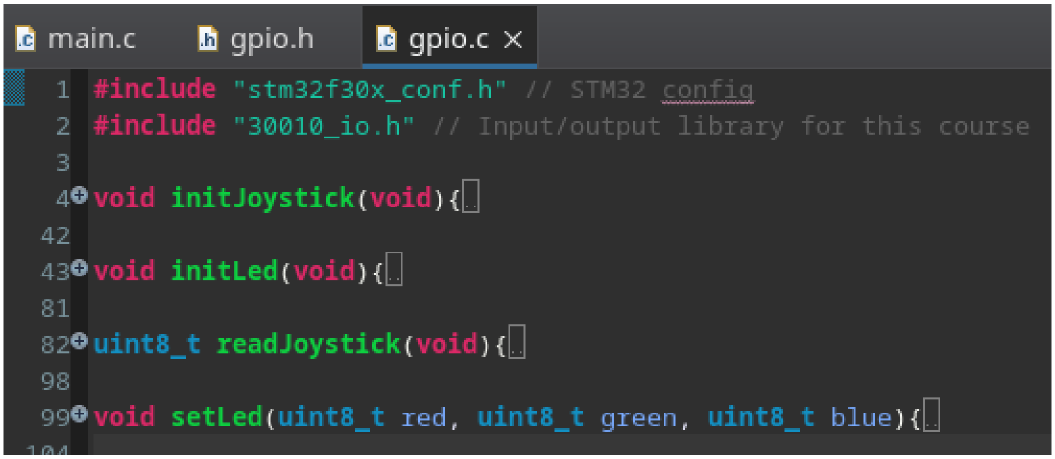


7. Application Structure

As the exercise told us to do we separated the gpio related functions to separate header and source files. The results are can be seen below:

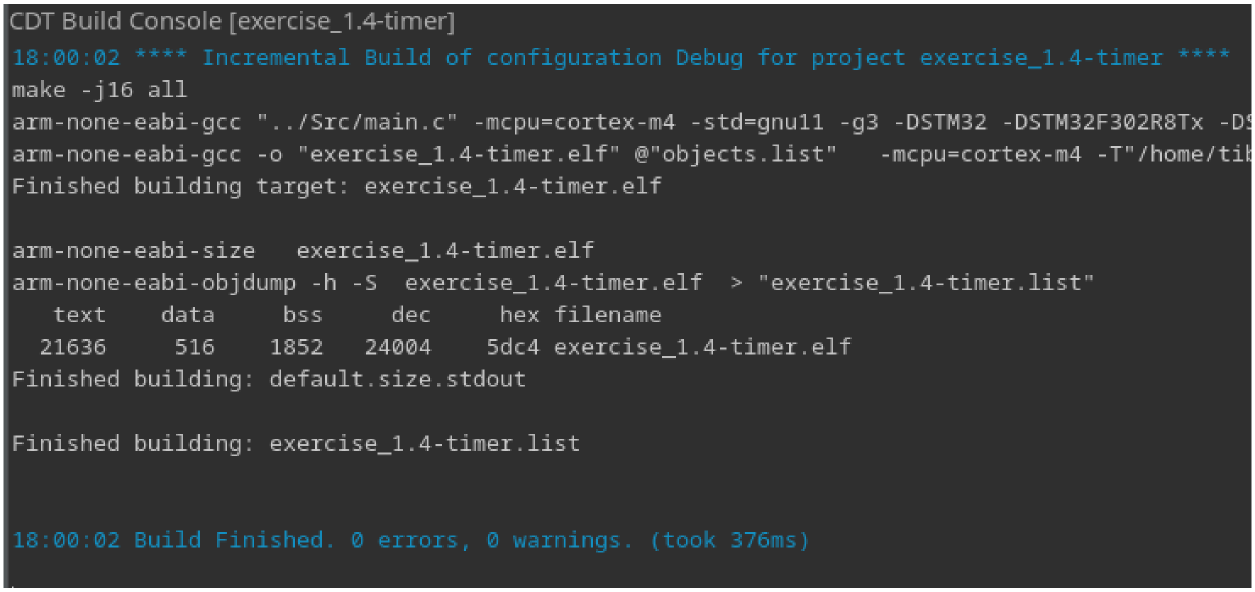


```
#ifndef GPIO_H_
#define GPIO_H_

void initJoystick(void);
void initLed(void);
uint8_t readJoystick(void);
void setLed(uint8_t red, uint8_t green, uint8_t blue);

#endif
```



```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h" // Input/output library for this course

void initJoystick(void){

void initLed(void){

uint8_t readJoystick(void){

void setLed(uint8_t red, uint8_t green, uint8_t blue){
```

After the function separation the project was successfully built and tested.


CDT Build Console [exercise_1.4-timer]

```
18:00:02 **** Incremental Build of configuration Debug for project exercise_1.4-timer ****
make -j16 all
arm-none-eabi-gcc "../Src/main.c" -mcpu=cortex-m4 -std=gnu11 -g3 -DSTM32 -DSTM32F302R8Tx -DS
arm-none-eabi-gcc -o "exercise_1.4-timer.elf" @"objects.list"   -mcpu=cortex-m4 -T"/home/ti
Finished building target: exercise_1.4-timer.elf

arm-none-eabi-size   exercise_1.4-timer.elf
arm-none-eabi-objdump -h -S  exercise_1.4-timer.elf  > "exercise_1.4-timer.list"
   text    data     bss     dec     hex filename
  21636     516    1852   24004    5dc4 exercise_1.4-timer.elf
Finished building: default.size.stdout

Finished building: exercise_1.4-timer.list


18:00:02 Build Finished. 0 errors, 0 warnings. (took 376ms)
```