**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

**Departamentul de Calculatoare și Securitate Cibernetică**



***Utilizare senzor digital vibrații***

***Platforma de dezvoltare frdm-kl25z***

Std. Sg. Maj. Acatrinei Alin

Std. Sg. Maj. Stratulat Dragoș

Std. Plt. Adj. Valeca Cristiana Maria

Grupa C114A & Grupa C114E

**București**

**2023**

Cuprins

[1. Prezentarea senzorului DFR0027 3](#_Toc123842030)

[2. Prezentarea senzorului DFR0067 4](#_Toc123842031)

[3. Scop proiect 5](#_Toc123842032)

[4. Conectare senzor – placă de dezvoltare 6](#_Toc123842033)

[5. Descriere program 7](#_Toc123842034)

[5.1. Funcția main 7](#_Toc123842035)

[5.2. Funcțiile periferice 8](#_Toc123842036)

[5.2.1. Funcțiile de UART 8](#_Toc123842037)

[5.2.2. Funcțiile de timp 10](#_Toc123842038)

[5.2.3. Inițializarea modulului GPIO 12](#_Toc123842039)

[5.2.4. Funcțiile senzorului DHT11 17](#_Toc123842040)

[5.2.5. Funcțiile senzorului de vibrație 21](#_Toc123842041)

[6. Program Python 22](#_Toc123842042)

[7. Dificultăți întâmpinate 24](#_Toc123842043)

[8. Bibliografie 29](#_Toc123842044)

# Prezentarea senzorului DFR0027

Senzorul DFR0027 este un senzor digital ce detectează vibrații. Prin activarea unei întreruperi pe falling edge, acesta acumuleaza un numar de vibratii per ciclu de polling al vibratiilor.



Figura 1. Senzorul DFR0027

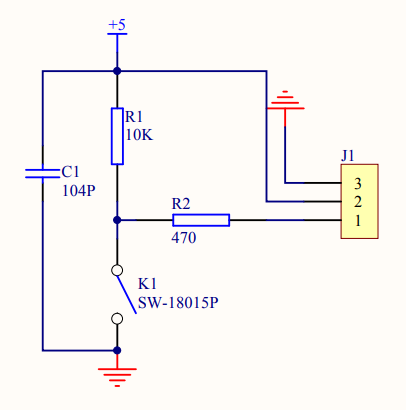


Figura 2. Circuitul senzorului DFR0027

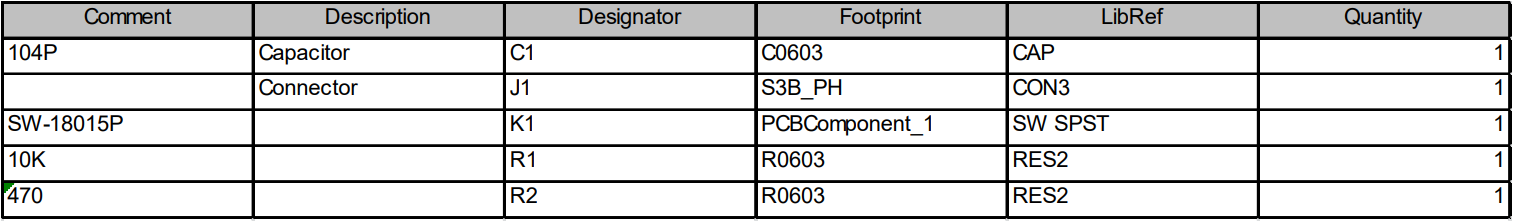


Figura 3. Componentele circuitului

Acesta este format dintr-un condensator de 104 pF și două rezistențe (una de 10KΩ, una de 470Ω. Se conectează la placa de dezvoltare prin trei conectori: unul negru (GND), unul roșu (+5V) și unul verde (OUTPUT). [1]

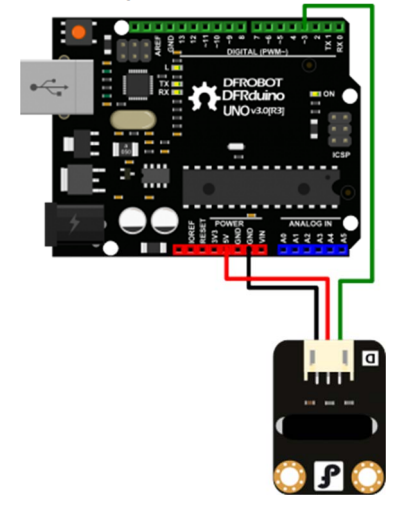


Figura 4. Exemplu conectare senzor

# Prezentarea senzorului DFR0067

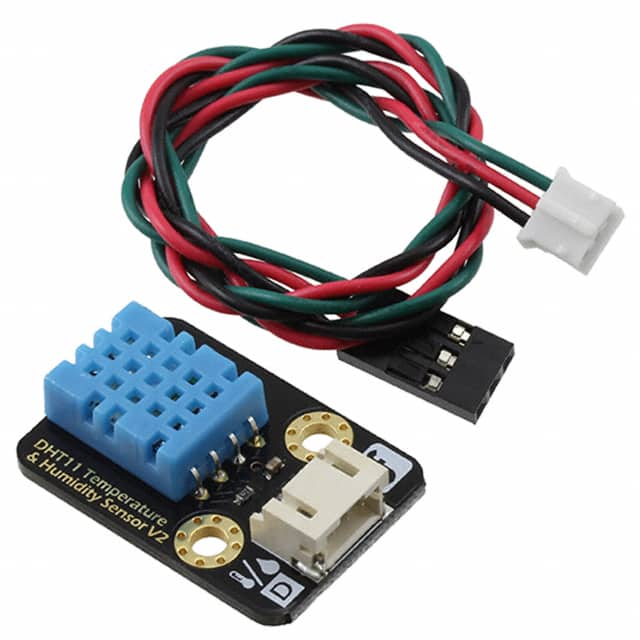


Figura 5. Senzorul DFR0067

Senzorul DFR0067 are la baza senzorul digital de temperatură si umiditate DHT11, acesta fiind montat de PCB si conectat la placa cu microcontroller prin conector JST.



Figura 6. Exemplu conectare senzor

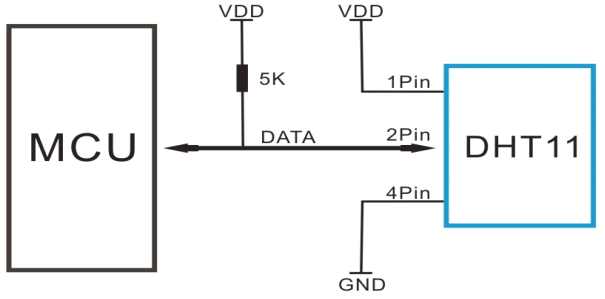


Figura 7. Circuit Senzor

# Scop proiect

Scopul acestui proiect este acela de a modifica culorile din LED-urile integrate in placă, în funcție de valorile primite de la senzorii digitali de vibrație si temperatură.

* Semnalizarea depășirii anumitor praguri de valori recepționate de către senzor, prin

intermediul led-ului RGB, e.g., R-temperatură, G-umiditate, B-vibrație.

* Afișarea unui grafic pentru fiecare informație primită de la senzori (temperatură, umiditate și vibrație).
* Un buton grafic pentru aprinderea tuturor led-urilor.

# Conectare senzor – placă de dezvoltare

Vom conecta senzorii astfel:

* Firul verde vibratie – PTD1
* Firul verde temperatura – PTA12
* Firele roșii – 3.3V
* Firele negre – GND

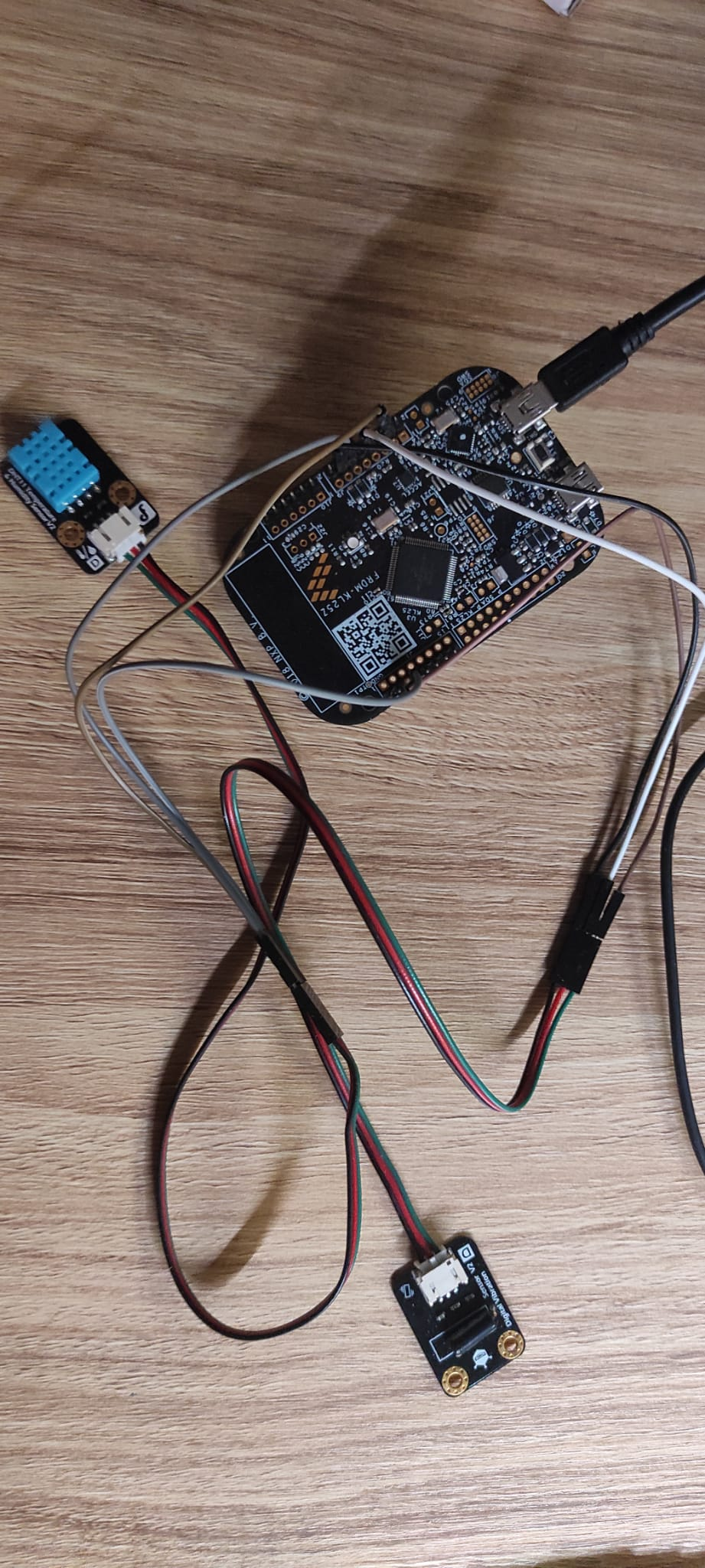


Figura 8. Conectare senzori

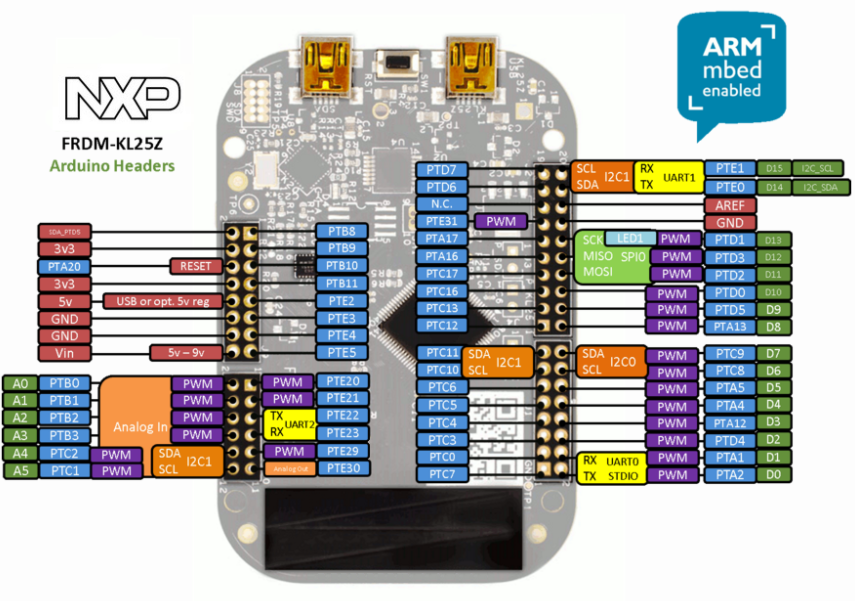


Figura 9. Pini placă dezvoltare[[1]](#footnote-1)

# Descriere program

## Funcția main

În fișierul main.c am inclus fișierele header în care sunt declarate funcții și variabile ce urmează a fi folosite: *dht.h* (functiile responsabile de senzorul de temperatura si umiditate), *uart.h* (funcțiile de transmisie si receptie seriala UART), *vibration.h* (functiile responsabile de senzorul de vibratie), *gpio.h* (functiile ce se ocupa de GPIO), timer.h (functiile de timp) și fișierul header generat de către mediul de dezvoltare Keil, *MKL25Z4.h*, specific plăcii de dezvoltare.

#include "dht.h"

#include "vibration.h"

extern double humidity;

extern double temperature;

extern uint16\_t vibrationCount;

extern uint8\_t ledState;

extern uint8\_t ledOverride;

/\* Only works for vibration, since temp. & hum. DHT11 sensor

\* doesn't pick up values properly for the moment

\*/

void decideSensorValues()

{

    if (vibrationCount < 256)

        ledState = 2;

    else if (vibrationCount < 1024)

        ledState = 3;

    else

        ledState = 1;

    vibrationCount = 0;

}

int main()

{

    UART\_Init(9600);

    Timer\_Init();

    Gpio\_Init();

    UART\_println("Start Program");

    UART\_println("------------------");

    ledOverride = 0;

    delay(500);

    for(;;) {

        printTemperature();

        printVibration();

        decideSensorValues();

        showLedStatus();

        UART\_println("------------------");

        delay(50);

    }

}

Logica principală a programului este următoarea: întâi se apelează funcțiile de inițializare a modulelor, apoi, într-un ciclu infinit, se transmit pe UART temperatura, umiditatea si numarul de vibratii acumulate de microprocesor intr un interval de timp in 50ms, se va verifica pragul atins de senzori si se vor afisa ledurile.

## Funcțiile periferice

### Funcțiile de UART

Prin apelul funcțiilor de receptie si transmisie, sunt puse la dispozitie modalitatile prin care se va receptiona cu PC-ul. În cadrul fișierului uart.c sunt create mai multe functii ce configureaza UART pentru transmisie si pentru receptie, dar si functii ajutatoare de transmisie de date mai complexe, precum un sir de caractere intreg.

#include "uart.h"

extern uint8\_t ledOverride;

uint8\_t UART\_read(void){

    //Punem in asteptare pana cand registrul de receptie nu este plin

    while(!(UART0->S1 & UART0\_S1\_RDRF\_MASK));

                return UART0->D;

}

void UART0\_IRQHandler(void) {

        char c = UART\_read();

        ledOverride = c - '0';

}

void UART\_printChar(char ch)

{

    //Punem in asteptare pana cand registrul de transmisie a datelor nu este gol

    while(!(UART0->S1 & UART0\_S1\_TDRE\_MASK)){}

                UART0->D = ch;

}

void UART\_Init(uint32\_t baud\_rate)

{

    //Setarea sursei de ceas pentru modulul UART

    SIM->SOPT2 |= SIM\_SOPT2\_UART0SRC(01);

    //Activarea semnalului de ceas pentru modulul UART

    SIM->SCGC4 |= SIM\_SCGC4\_UART0\_MASK;

    //Activarea semnalului de ceas pentru portul A

    //intrucat dorim sa folosim pinii PTA1, respectiv PTA2 pentru comunicarea UART

    SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK;

    //Fiecare pin pune la dispozitie mai multe functionalitati

    //la care avem acces prin intermediul multiplexarii

    PORTA->PCR[1] = ~PORT\_PCR\_MUX\_MASK;

    PORTA->PCR[1] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2); // Configurare RX pentru UART0

    PORTA->PCR[2] = ~PORT\_PCR\_MUX\_MASK;

    PORTA->PCR[2] = PORT\_PCR\_ISF\_MASK | PORT\_PCR\_MUX(2); // Configurare TX pentru UART0

    UART0->C2 &= ~((UART0\_C2\_RE\_MASK) | (UART0\_C2\_TE\_MASK));

    //Configurare Baud Rate

    uint32\_t osr = 15; // Over-Sampling Rate (numarul de esantioane luate per bit-time)

    //SBR - vom retine valoarea baud rate-ului calculat pe baza frecventei ceasului de sistem

    //   SBR  -     b16 b15 b14 [b13 b12 b11 b10 b09        b08 b07 b06 b05 b04 b03 b02 b01] &

    // 0x1F00 -     0       0   0    1   1   1   1   1      0   0   0   0   0   0   0   0

    //            0   0   0    b13 b12 b11 b10 b09    0   0   0   0   0   0   0   0 >> 8

    //   BDH  -   0   0   0    b13 b12 b11 b10 b09

    //   BDL  -   b08 b07 b06  b05 b04 b03 b02 b01

    uint16\_t sbr = (uint16\_t)((DEFAULT\_SYSTEM\_CLOCK)/(baud\_rate \* (osr+1)));

    uint8\_t temp = UART0->BDH & ~(UART0\_BDH\_SBR(0x1F));

    UART0->BDH = temp | UART0\_BDH\_SBR(((sbr & 0x1F00)>> 8));

    UART0->BDL = (uint8\_t)(sbr & UART\_BDL\_SBR\_MASK);

    UART0->C4 |= UART0\_C4\_OSR(osr);

    //Setare numarul de biti de date la 8 si fara bit de paritate

    UART0->C1 = 0;

    //Dezactivare intreruperi la transmisie

    UART0->C2 |= UART0\_C2\_TIE(0);

    UART0->C2 |= UART0\_C2\_TCIE(0);

    //Activare intreruperi la receptie

    UART0->C2 |= UART0\_C2\_RIE(1);

    UART0->C2 |= ((UART\_C2\_RE\_MASK) | (UART\_C2\_TE\_MASK));

    NVIC\_EnableIRQ(UART0\_IRQn);

}

void UART\_print(char\* string)

{

        int i = 0;

    while(string[i] != '\0'){

        UART\_printChar(string[i]);

        i++;

    }

}

void UART\_println(char\* string)

{

    UART\_print(string);

    UART\_printChar('\n');

    UART\_printChar('\r');

}

UART\_print, UART\_println, și UART\_printChar sunt funcții prin care se transmite date sub formă de șiruri de caractere, cu o influență din biblioteca “string.h” a C. UART\_read și UART0\_IRQHandler sunt folosite pentru recepție de caractere și o mini-procesare a datelor primit – programul se asteapta sa primeasca caractere numere digitale. Cu aceste numere se vor activa/dezactiva LED-urile.

UART\_init este configuratia care se va face inainte de folosirea modulului UART. Acesta are pașii următori:

1. Setare sursă ceas
2. Activare ceas modul UART0
3. Activare ceas Port A 1 si 2
4. Configurare multiplexare pentru recepție și transmisie
5. Configurare esantionare
6. Setare biți paritate UART
7. Dezactivare întreruperi transmisie
8. Activare întreruperi recepție

### Funcțiile de timp

În fișierul timer.c sunt definite toate funcțiile ce țin de sincronizarea datelor.

#include "timer.h"

int timerCounter;

unsigned long microCounter;

void Timer\_Init(void) {

    // Activarea semnalului de ceas pentru perifericul PIT

    SIM->SCGC6 |= SIM\_SCGC6\_PIT\_MASK;

    // Utilizarea semnalului de ceas pentru tabloul de timere

    PIT\_MCR &= ~PIT\_MCR\_MDIS\_MASK;

    // Oprirea decrementarii valorilor numaratoarelor in modul debug

    PIT->MCR |= PIT\_MCR\_FRZ\_MASK;

    // Setarea valoarea numaratorului de pe canalul 0 la o perioada de 1 microsecunda

    //PIT->CHANNEL[0].LDVAL = 0x14;

    PIT->CHANNEL[0].LDVAL = 0x17;

  // Activarea întreruperilor pe canalul 0

    PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TIE\_MASK;

    // Activarea timerului de pe canalul 0

    PIT->CHANNEL[0].TCTRL |= PIT\_TCTRL\_TEN\_MASK;

    // Activarea întreruperii mascabile si setarea prioritatiis

    NVIC\_ClearPendingIRQ(PIT\_IRQn);

    NVIC\_SetPriority(PIT\_IRQn,5);

    NVIC\_EnableIRQ(PIT\_IRQn);

}

void PIT\_IRQHandler(void) {

    if(PIT->CHANNEL[0].TFLG & PIT\_TFLG\_TIF\_MASK) {

        timerCounter++;

        microCounter++;

        PIT->CHANNEL[0].TFLG &= PIT\_TFLG\_TIF\_MASK;

    }

}

void delayMicroseconds(int x){

    timerCounter = 0;

    while(timerCounter < x){;}

}

void delay(int x){

        delayMicroseconds(1000 \* x);

}

unsigned long micros(){

    return microCounter;

}

Timer\_Init este responsabil de inițializarea semnalului de ceas a PIT:

1. Activare semnal de ceas pentru modul PIT
2. Utilizare semnal ceas pentru tablou de timere
3. Oprire numărat în modul debug
4. Setare canal 0 odată la o microsecundă
5. Activare întrerupere și timer pe canalul 0
6. Activare întrerupere

PIT\_IRQHandler va îngloba funcția de întrerupere, în aceasta se vor incrementa 2 variabile, folosite pentru funcțiile delay (translație directă a celor din biblioteca Arduino).

### Inițializarea modulului GPIO

Funcțiile de gpio.c vor include translații a bibliotecii Arduino, pentru implementarea senzorului de temperatură & umiditate.

#include "gpio.h"

uint8\_t ledState = 0;

uint8\_t ledOverride = 0;

void Gpio\_Init(void) {

    // Activation of PORT A, PORT B and PORT D signals

    SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK | SIM\_SCGC5\_PORTB\_MASK | SIM\_SCGC5\_PORTD\_MASK;

    /\* DHT11 Temperature Sensor START\*/

    // Setting the multiplex value on 1 to activate the port

    PORTA->PCR[DHT\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

    PORTA->PCR[DHT\_PIN] |= PORT\_PCR\_MUX(1);

    GPIOA\_PDDR |= (1 << DHT\_PIN);

    GPIOA\_PSOR |= (1 << DHT\_PIN);

    /\* DHT11 Temperature Sensor STOP\*/

    /\* Vibration Sensor START\*/

    // Setting the multiplex value on 1 to activate the port

    PORTD->PCR[VIB\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

    PORTD->PCR[VIB\_PIN] |= PORT\_PCR\_MUX(1);

    // Interrupt on falling edge

    PORTD->PCR[VIB\_PIN] |= PORT\_PCR\_IRQC(0x0A) | PORT\_PCR\_PE\_MASK;

    NVIC\_ClearPendingIRQ(PORTD\_IRQn);

    NVIC\_SetPriority(PORTD\_IRQn, 128);

    NVIC\_EnableIRQ(PORTD\_IRQn);

    /\* Vibration Sensor STOP\*/

    /\* LED START \*/

    // --- RED LED ---

    // Utilizare GPIO ca varianta de multiplexare

    PORTB->PCR[RED\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

    PORTB->PCR[RED\_LED\_PIN] |= PORT\_PCR\_MUX(1);

    // Configurare pin pe post de output

    GPIOB\_PDDR |= (1<<RED\_LED\_PIN);

    // Stingerea LED-ului (punerea pe 1 logic) - LED Stins pe HIGH

    GPIOB\_PSOR |= (1<<RED\_LED\_PIN);

    // --- GREEN LED ---

    // Utilizare GPIO ca varianta de multiplexare

    PORTB->PCR[GREEN\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

    PORTB->PCR[GREEN\_LED\_PIN] |= PORT\_PCR\_MUX(1);

    // Configurare pin pe post de output

    GPIOB\_PDDR |= (1<<GREEN\_LED\_PIN);

    // Stingerea LED-ului (punerea pe 1 logic) - LED Stins pe HIGH

    GPIOB\_PSOR |= (1<<GREEN\_LED\_PIN);

    // --- BLUE LED ---

        // Utilizare GPIO ca varianta de multiplexare

    PORTD->PCR[BLUE\_LED\_PIN] &= ~PORT\_PCR\_MUX\_MASK;

    PORTD->PCR[BLUE\_LED\_PIN] |= PORT\_PCR\_MUX(1);

    // Configurare pin pe post de output

    GPIOD\_PDDR |= (1<<BLUE\_LED\_PIN);

    // Stingerea LED-ului (punerea pe 1 logic) - LED Stins pe HIGH

    GPIOD\_PSOR |= (1<<BLUE\_LED\_PIN);

    /\* LED STOP \*/

}

int digitalReadA(int pin){

    delayMicroseconds(1);

    if(GPIOA\_PDIR & (1 << pin))

        return 1;

    return 0;

}

void digitalWriteA(int pin, int value){

    if(value == LOW){

        GPIOA\_PCOR |= (1 << pin);

    }

    else if (value == HIGH){

        GPIOA\_PDOR |= 1 << pin;

    }

    //delayMicroseconds(1);

}

void pinModeA(int pin, int mode){

    if (mode == OUTPUT){

        GPIOA\_PDDR |= 1 << pin;

    }

    else {

        GPIOA\_PDDR &= ~(1 << pin);

    }

}

void PORTD\_IRQHandler() {

    vibrationCount++;

    PORTD\_ISFR = (1<<VIB\_PIN);

}

char\* itoa(int value, char\* result, int base) {

    // check that the base if valid

    if (base < 2 || base > 36) { \*result = '\0'; return result; }

    char\* ptr = result, \*ptr1 = result, tmp\_char;

    int tmp\_value;

    do {

        tmp\_value = value;

        value /= base;

        \*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp\_value - value \* base)];

    } while ( value );

    // Apply negative sign

    if (tmp\_value < 0) \*ptr++ = '-';

    \*ptr-- = '\0';

    while(ptr1 < ptr) {

        tmp\_char = \*ptr;

        \*ptr--= \*ptr1;

        \*ptr1++ = tmp\_char;

    }

    return result;

}

void setLEDStatus(int LED\_PIN, int status){

    switch (LED\_PIN){

        case RED\_LED\_PIN:

            if (status == HIGH){

                GPIOB\_PCOR |= (1<<RED\_LED\_PIN);

            }

            else{

                GPIOB\_PDOR |= (1<<RED\_LED\_PIN);

            }

            break;

        case GREEN\_LED\_PIN:

            if (status == HIGH){

                GPIOB\_PCOR |= (1<<GREEN\_LED\_PIN);

            }

            else{

                GPIOB\_PDOR |= (1<<GREEN\_LED\_PIN);

            }

            break;

        case BLUE\_LED\_PIN:

            if (status == HIGH){

                GPIOD\_PCOR |= (1<<BLUE\_LED\_PIN);

            }

            else{

                GPIOD\_PDOR |= (1<<BLUE\_LED\_PIN);

            }

            break;

            default:

                break;

    }

}

void showLedStatus(void){

    switch (ledOverride)

    {

        case 0:

            break;

        case 1:

            setLEDStatus(RED\_LED\_PIN, HIGH);

            setLEDStatus(GREEN\_LED\_PIN, LOW);

            setLEDStatus(BLUE\_LED\_PIN, LOW);

            return;

        case 2:

            setLEDStatus(RED\_LED\_PIN, LOW);

            setLEDStatus(GREEN\_LED\_PIN, HIGH);

            setLEDStatus(BLUE\_LED\_PIN, LOW);

            return;

        case 3:

            setLEDStatus(RED\_LED\_PIN, LOW);

            setLEDStatus(GREEN\_LED\_PIN, LOW);

            setLEDStatus(BLUE\_LED\_PIN, HIGH);

            return;

        default:

            break;

    }

    switch (ledState)

    {

        case 0:

            break;

        case 1:

            setLEDStatus(RED\_LED\_PIN, HIGH);

            setLEDStatus(GREEN\_LED\_PIN, LOW);

            setLEDStatus(BLUE\_LED\_PIN, LOW);

            return;

        case 2:

            setLEDStatus(RED\_LED\_PIN, LOW);

            setLEDStatus(GREEN\_LED\_PIN, HIGH);

            setLEDStatus(BLUE\_LED\_PIN, LOW);

            return;

        case 3:

            setLEDStatus(RED\_LED\_PIN, LOW);

            setLEDStatus(GREEN\_LED\_PIN, LOW);

            setLEDStatus(BLUE\_LED\_PIN, HIGH);

            return;

        default:

            break;

    }

    return;

}

Gpio\_init va inițializa pinii PTA12 (senzor temperatură și umiditate) PTD1 (senzor vibrație) și pinii LED-urilor. Pentru senzorul de vibrație se va configura și întreruperea necesară pe falling-edge.

În acest fișier sunt introduse și funcții auxiliare precum itoa, showLedStatus, setLEDStatus și funcțiile translatate.

### Funcțiile senzorului DHT11

În cadrul fișierului dht.c sunt implementate read și read11 – translatate din biblioteca DHT11 Arduino. read11 este o particularizare, întrucât există și varianta de senzor DHT22 cu funcția sa specifică, read22. În acest program a fost translatată numai cea pentru senzorul folosit.

Principiul de funcționare este explicat în adâncime în capitolul “Probleme Întampinate”, însă, pe scurt, programul citește 5 octeți de la senzor pe care îi va transforma în informație utilă – în cazul acesta, temperatură și umiditate.

#include "dht.h"

#define TIMEOUT 10000

uint8\_t bits[5]; // buffer to receive data

double humidity;

double temperature;

// return values:

// DHTLIB\_OK

// DHTLIB\_ERROR\_CHECKSUM

// DHTLIB\_ERROR\_TIMEOUT

int read11(uint8\_t pin)

{

    // READ VALUES

    int rv = read(pin);

    if (rv != DHTLIB\_OK)

    {

        humidity = DHTLIB\_INVALID\_VALUE; // or is NaN prefered?

        temperature = DHTLIB\_INVALID\_VALUE;

        return rv;

    }

    // CONVERT AND STORE

    humidity = bits[0]; // bit[1] == 0;

    char result[20];

    itoa(humidity, result, 10);

    // itoa(humidity,result,2);

    result[19] = '\0';

    UART\_println(result);

    temperature = bits[2]; // bits[3] == 0;

    // TEST CHECKSUM

    uint8\_t sum = bits[0] + bits[2]; // bits[1] && bits[3] both 0

    if (bits[4] != sum)

        return DHTLIB\_ERROR\_CHECKSUM;

    return DHTLIB\_OK;

}

// return values:

// DHTLIB\_OK

// DHTLIB\_ERROR\_TIMEOUT

int read(uint8\_t pin)

{

    // INIT BUFFERVAR TO RECEIVE DATA

    uint8\_t cnt = 7;

    uint8\_t idx = 0;

    int i;

    // EMPTY BUFFER

    for (i = 0; i < 5; i++)

        bits[i] = 0;

    // REQUEST SAMPLE

    pinModeA(pin, OUTPUT);

    digitalWriteA(pin, LOW);

    delay(20);

    digitalWriteA(pin, HIGH);

    delayMicroseconds(30);

    digitalWriteA(pin, LOW);

    pinModeA(pin, INPUT);

    // GET ACKNOWLEDGE or TIMEOUT

    unsigned int loopCnt = TIMEOUT;

    while (digitalReadA(pin) == LOW)

        if (loopCnt-- == 0)

            return DHTLIB\_ERROR\_TIMEOUT;

    loopCnt = TIMEOUT;

    while (digitalReadA(pin) == HIGH)

        if (loopCnt-- == 0)

            return DHTLIB\_ERROR\_TIMEOUT;

    delayMicroseconds(170);

    // READ THE OUTPUT - 40 BITS => 5 BYTES

    for (i = 0; i < 40; i++)

    {

        loopCnt = TIMEOUT;

        int test = LOW;

        while (test == LOW)

        {

            test = digitalReadA(pin);

            if (loopCnt-- == 0)

                return DHTLIB\_ERROR\_TIMEOUT;

        }

        unsigned long t = micros();

        loopCnt = TIMEOUT;

        test = HIGH;

        while (digitalReadA(pin) == HIGH)

        {

            test = digitalReadA(pin);

            if (loopCnt-- == 0)

                return DHTLIB\_ERROR\_TIMEOUT;

        }

        while (digitalReadA(pin) == HIGH)

        {

            // digitalReadA(pin);

            if (loopCnt-- == 0)

                return DHTLIB\_ERROR\_TIMEOUT;

        }

        if ((micros() - t) > 40)

        {

            bits[idx] |= (1 << cnt);

            UART\_print("1");

        }

        else

        {

            UART\_print("0");

        }

        if (cnt == 0) // next byte?

        {

            cnt = 7;

            idx++;

        }

        else

            cnt--;

    }

    return DHTLIB\_OK;

}

void printTemperature(void)

{

    char buffer[6];

    read11(DHT\_PIN);

    int temperature\_temp = (int)temperature;

    int humidity\_temp = (int)humidity;

    itoa(temperature\_temp, buffer, 6);

    buffer[6] = '\0';

    UART\_print("T");

    UART\_println(buffer);

    itoa(humidity\_temp, buffer, 6);

    buffer[6] = '\0';

    UART\_print("H");

    UART\_println(buffer);

}

//

// END OF FILE

//

### Funcțiile senzorului de vibrație

Senzorul de vibrație nu are o configurație mare, precum senzorul anterior, întrucât este necesară numărarea pulsurilor pe care senzorul le transmite când simte mișcare, iar întreruperea de pe portul D o va număra.

#include "vibration.h"

uint16\_t vibrationCount = 0;

int readVibration(void)

{

    return vibrationCount;

}

void printVibration(void)

{

    char buffer[6];

    int temp\_vibration = readVibration();

    itoa(temp\_vibration, buffer, 6);

    buffer[6] = '\0';

    UART\_print("V");

    UART\_println(buffer);

}

# Program Python

Programul din Python este unul simplist, ce implementează o interfață in framework-ul PyQT. Pe scurt, acesta deschide o comunicație pe canal UART serială pentru a transfera datele între PC si FRDM-KL25Z. Din interfață se pot vedea datele senzorului de vibrație (codul DHT11 are probleme de funcționare) și acționa cele 4 butoane pentru a controla culoarea LED-urilor sau pentru a le elibera de override.

import serial  
from PyQt5 import QtWidgets, QtCore  
from PyQt5.QtWidgets import QPushButton  
import pyqtgraph as pg  
import sys  
import serial  
  
ser\_com = serial.Serial('COM15', 9600)  
  
  
class MainWindow(QtWidgets.QMainWindow):  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super(MainWindow, self).\_\_init\_\_(\*args, \*\*kwargs)  
 self.setWindowTitle("Microcontroller Window")  
  
 self.graphWidget = pg.PlotWidget()  
 self.setCentralWidget(self.graphWidget)  
 self.graphWidget.setBackground((245, 245, 245))  
 self.graphWidget.setTitle("\nVibration Data\n", color="black", size="25px")  
  
 styles = {'color': 'black', 'font-size': '15px'}  
 self.graphWidget.setLabel('left', 'Vibration', \*\*styles)  
 self.graphWidget.setLabel('bottom', 'Time', \*\*styles)  
 self.graphWidget.showGrid(x=True, y=True)  
 self.graphWidget.setYRange(0, 6000, padding=0)  
  
 self.x = list(range(50))  
 self.y = list(range(50))  
  
 pen = pg.mkPen(color=(0, 128, 128), width=5)  
 self.data\_line = self.graphWidget.plot(self.x, self.y, pen=pen)  
  
 self.timer = QtCore.QTimer()  
 self.timer.setInterval(1)  
 self.timer.timeout.connect(self.update\_plot\_data)  
 self.timer.start()  
 self.initUI()  
  
 def initUI(self):  
  
 self.setGeometry(100, 100, 1600, 900)  
 self.redButton = QPushButton(self)  
 self.redButton.setText("Red")  
 self.redButton.clicked.connect(self.signalRed)  
 self.redButton.move(0, 0)  
  
 self.greenButton = QPushButton(self)  
 self.greenButton.setText("Green")  
 self.greenButton.clicked.connect(self.signalGreen)  
 self.greenButton.move(100, 0)  
  
 self.blueButton = QPushButton(self)  
 self.blueButton.setText("Blue")  
 self.blueButton.clicked.connect(self.signalBlue)  
 self.redButton.move(200, 0)  
  
 self.overrideButton = QPushButton(self)  
 self.overrideButton.setText("OFF")  
 self.overrideButton.clicked.connect(self.signalOff)  
 self.overrideButton.move(300, 0)  
  
 def update\_plot\_data(self):  
 data = ser\_com.readline().decode('utf-8')  
 try:  
 for line in data.split('\n'):  
 line = line.strip('\r')  
 if not line[0] == 'V':  
 continue  
 val = line[1:]  
 if val == '':  
 floatdata = 0  
 else:  
 floatdata = float(val)  
  
 self.x = self.x[1:]  
 self.x.append(self.x[-1] + 1)  
  
 self.y = self.y[1:]  
  
 self.y.append(floatdata)  
 self.data\_line.setData(self.x, self.y)  
 except:  
 return  
  
 def signalRed(self):  
 ser\_com.write(b'1')  
  
 def signalGreen(self):  
 ser\_com.write(b'2')  
  
 def signalBlue(self):  
 ser\_com.write(b'3')  
  
 def signalOff(self):  
 ser\_com.write(b'0')  
  
  
def main():  
 ser\_com.write(b'0')  
 while True:  
 data = ser\_com.readline()  
 data = data.decode("utf-8")  
  
 # print(data)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # main()  
 app = QtWidgets.QApplication(sys.argv)  
  
 SensorWindow = MainWindow()  
 SensorWindow.show()  
 sys.exit(app.exec\_())

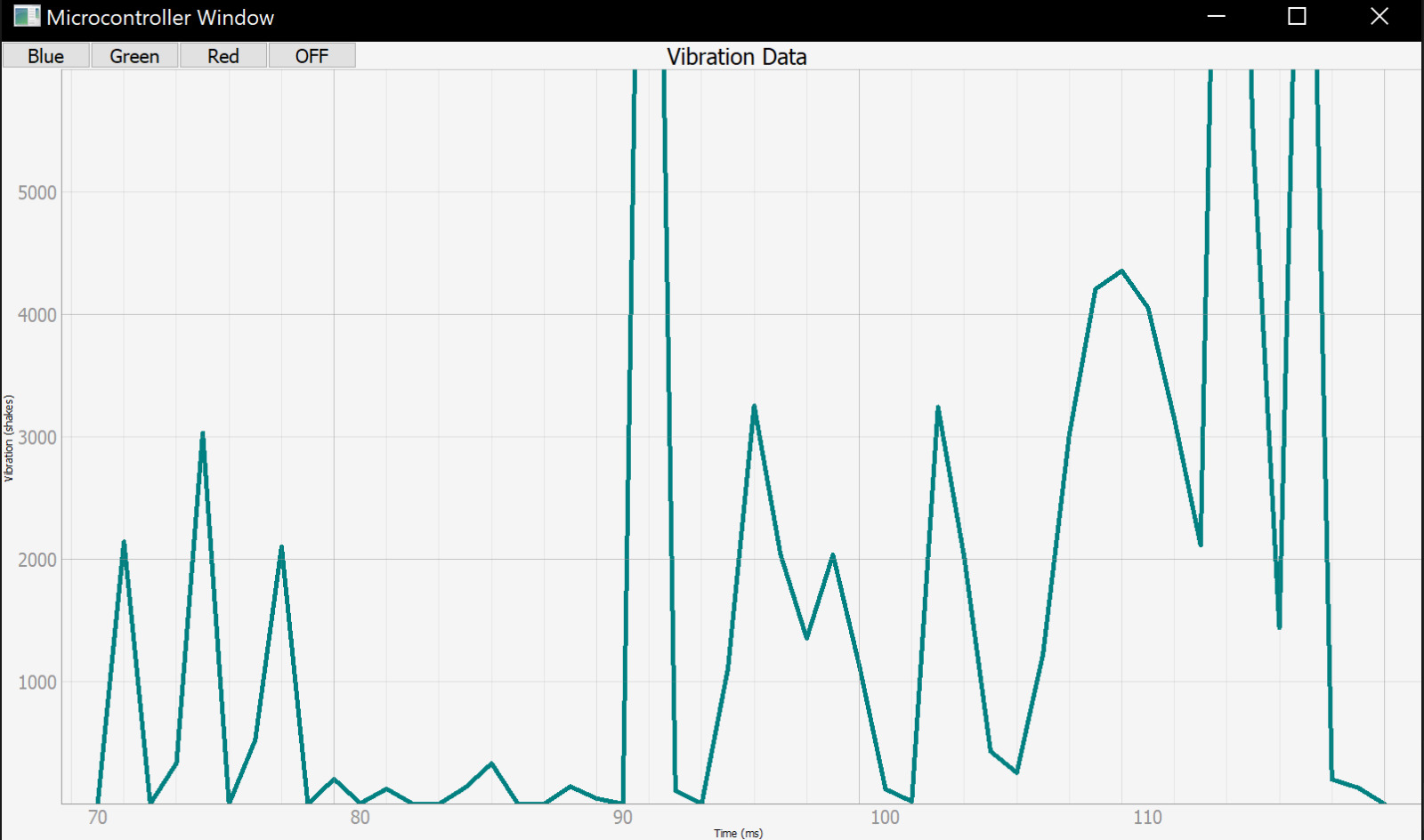


Figura 12. Output Program Python

# Dificultăți întâmpinate

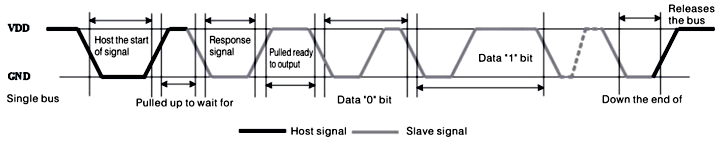
Pentru configurarea senzorului digital de temperatură si umiditate DFR0067 exista probleme in configurarea bare-bones a codului. Senzorul DFR0067 se bazeaza pe DHT11, iar acesta are nevoie de o configurare specializată: <https://components101.com/sites/default/files/component_datasheet/DHT11-Temperature-Sensor.pdf>.

Figura 13. Protocol DHT11

După cum se vede in imaginea alaturată, Microcontroller-ul trebuie sa inițieze captura unei masurători printr-un semnal specific de minim 18 ms 0 logic, urmat de 1 logic.

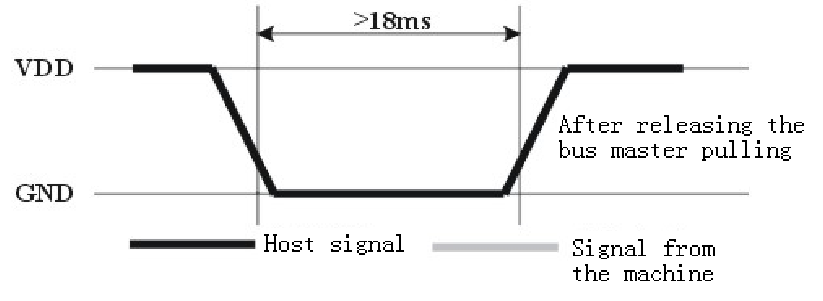


Figura 14. Inițiere captură

În urma începerii capturii, senzorul transmite date în următorul mod:

* 0 logic timp de 50 µs
* 1 logic timp de 30 µs (bit de 0) *sau* 70 µs (bit de 1) – în funcție de cât este ținut pe HIGH

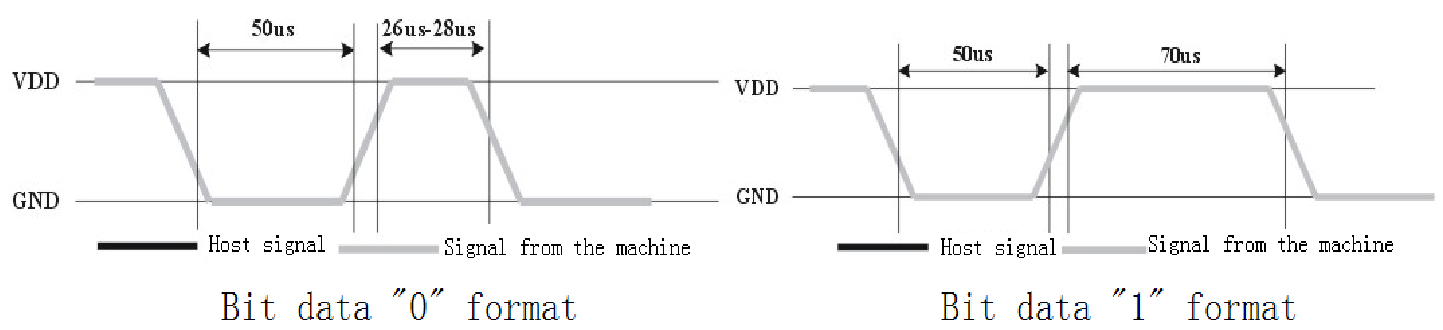


Figura 15. Primire bit de la DHT11

În programul realizat de echipa noastră, nu am reușit să capturăm complet un asemenea flow de date – pachetul este de 5 octeți, însă am realizat cu un analizor logic o captură de date de la/către senzor unde se pot vedea transmisii “sănătoase”.

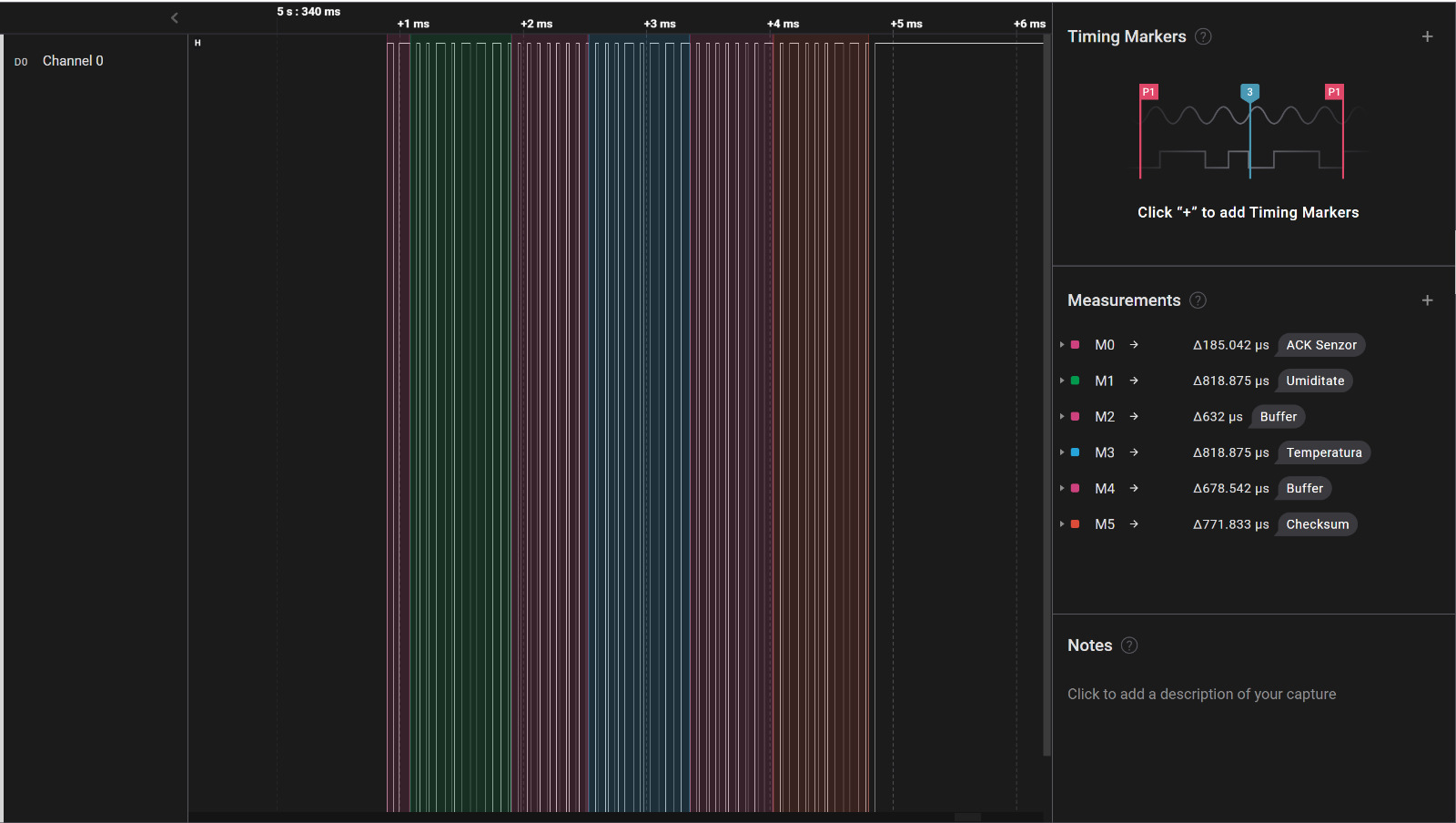


Figura 16. Captură analizor logic



Figura 17. Captură umiditate

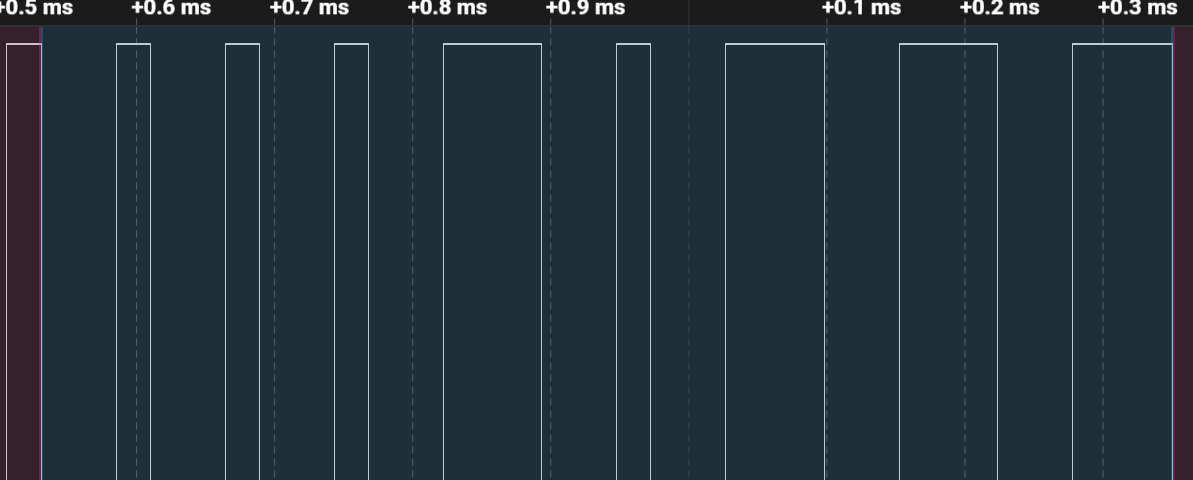


Figura 18. Captură temperatură

În pozele anterioare se pot observa oscilații de la 0 la 1 logic, in pachete de câte 8 biți. Primul segment verde și al 3-lea segment sunt de interes, întrucât segmentul 2 si 4 transmit reprezentarea pe 2 octeți a datelor, iar temperatura si umiditatea sunt sub 100, așadar încap într-un octet. Octetul 5 este un checksum, pentru verificare integritate date transmise.

Se poate observa în segmentul verde de umiditate că acesta corespunde numărului binar 0010 11102 , care, în zecimal, este 4610 – 46 % umiditate.

În cel albastru, descifrarea este 0001 01112 , iar în zecimal este 2310 – 23 de grade Celsius.

Este de menționat că această încercare de primire/transmisie biți de la/către DHT11 a fost translatată din apeluri din biblioteca DHT11 a Arduino: <https://github.com/adidax/dht11>.

Pentru a face codul sa functioneze, am folosit MbedOS de pe Keil Studio Cloud si am refacut tiparul codului in acesta, pentru a observa cum se interogheaza datele de pe senzorul DHT11.

#include "DHT.h"

#include "mbed.h"

uint16\_t humidity;

uint16\_t temperature;

int vibration\_count;

char buf\_vibration[6];

char buf\_temperature[10];

char buf\_humidity[10];

int led\_override = 0;

int led\_state = 0;

InterruptIn vibration\_pin(D2);

DigitalOut RED\_LED(LED\_RED, 1);

DigitalOut GREEN\_LED(LED\_GREEN, 1);

DigitalOut BLUE\_LED(LED\_BLUE, 1);

DHT dht(D3, DHT11);

static BufferedSerial serial\_port(USBTX, USBRX, 9600);

void vibration\_count\_func() { vibration\_count++; }

char \*getVibration() {

  sprintf(buf\_vibration, "%d", vibration\_count);

  vibration\_count = 0;

  return buf\_vibration;

}

char \*getTemperature() {

  dht.readData();

  float t = dht.ReadTemperature(CELCIUS);

  int roundT = (int)t;

  sprintf(buf\_temperature, "%d", roundT);

  return buf\_temperature;

}

char \*getHumidity() {

  dht.readData();

  float h = dht.ReadHumidity();

  int roundH = (int)h;

  sprintf(buf\_humidity, "%d", roundH);

  return buf\_humidity;

}

int main() {

  vibration\_pin.fall(&vibration\_count\_func);

  printf("Start Program\n");

  printf("------------------\n");

  ThisThread::sleep\_for(1000);

  while (1) { // wait around, interrupts will interrupt this!

    // RED\_LED = 0;

    // ThisThread::sleep\_for(250);

    // RED\_LED = 1;

    // ThisThread::sleep\_for(250);

    printf("T = %s °C\r\n", getTemperature());

    printf("H = %s %\r\n", getHumidity());

    printf("V = %s\r\n", getVibration());

    ThisThread::sleep\_for(100);

  }

}

# Bibliografie

|  |  |
| --- | --- |
| [1] | "DFR0027 Datasheet," [Online]. Available: https://www.farnell.com/datasheets/2700130.pdf. |
| [2] | Freescale Semiconductor, Inc., KL25 Sub-Family Reference Manual, 2012. |
| [3] | STM, STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual, 2017. |
| [4] | Adidax. [Online]. Available: https://github.com/adidax/dht11. |
| [5] | "DFR0067 Datasheet," [Online]. Available: https://wiki.dfrobot.com/DHT11\_Temperature\_and\_Humidity\_Sensor\_\_SKU\_\_DFR0067\_. |

1. https://os.mbed.com/platforms/KL25Z/ [↑](#footnote-ref-1)