# BEÁGYAZOTT RENDSZEREK (C)

5. forduló

BOSCH
Invented for life

A kategória támogatója: Robert Bosch Kft.

## Ismertető a feladatlaphoz

> Kezdj neki minél hamarabb, mert a feladatot a forduló záró időpontjáig lehet beküldeni, nem addig lehet elkezdeni!

Sok sikert!


---

Mekk Mester azt a feladatot kapta, hogy készítsen egy autólopás-érzékelőt. Ehhez a feladathoz az ESP32 mikrovezérlőt és az MPU6050-et választotta, amiben a gyorsulásmérő szenzorral detektálni szeretné az autó mozgatását.

MPU6050 register adatlap: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

MPU6050 adatlap: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

ESP IDF assembly utasítások: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ulp_instruction_set.html

ESP32 Wire API: https://www.arduino.cc/reference/en/language/functions/communication/wire/

## 1. feladat 3 pont

Mekk mester szeretné, hogy az adatok 25Hz-el jöjjenek a szenzorból. Mennyire kell ehhez beállítani a SMPLRT_DIV regiszter értékét, ha a szenzorban a DLPF engedélyezve van?

### Válasz

## 2. feladat 5 pont

Mekk Mester szeretné inicializálni a szenzort úgy, hogy a mérési tartománya legyen a gyroszkópnak ±500°/s, a gyorsulásmérőnek pedig ±4g, és az INT lába jelezze a Data Ready-t magas szinttel, amíg a mérés kiolvasásra nem kerül.

Válaszd ki a kódrészletet, ami ennek megfelelően állítja be a szenzort. (A regiszterek nevei megegyeznek az adatlapban használt rövidítésekkel.)

### Válasz

```
#include <Wire.h>

void writeRegister(uint8_t registerAddress, uint8_t value) {
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.write(value);
    Wire.endTransmission();
}

uint8_t readRegister(uint8_t registerAddress) {
    uint8_t readed;
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.endTransmission();
    Wire.requestFrom((uint8_t)mAddress, 1);
    readed = Wire.read();
    return readed;
}

bool initialize() {
    uint8_t whoIam = readRegister(WHO_AM_I);
    if (whoIam != 0x68) {
```

```
        return false;
    }

    writeRegister(PWR_MGMT_1, 0x80);
    delay(10);
    writeRegister(PWR_MGMT_1, 0x00);
    delay(100);
    writeRegister(PWR_MGMT_1, 0x01);
    writeRegister(INT_PIN_CFG, 0x32);
    writeRegister(CONFIG, 0x03);
    writeRegister(SMPLRT_DIV, SAMPLE_RATE_DIV_VALUE);
    writeRegister(INT_ENABLE, 0x01);
    writeRegister(GYRO_CONFIG, 0x08);
    writeRegister(ACCEL_CONFIG, 0x08);

    return true;
}
```

```
#include <Wire.h>

void writeRegister(uint8_t registerAddress, uint8_t value) {
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.write(value);
    Wire.endTransmission();
}

uint8_t readRegister(uint8_t registerAddress) {
    uint8_t readed;
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.endTransmission();
    Wire.requestFrom((uint8_t)mAddress, 1);
    readed = Wire.read();
    return readed;
}

bool initialize() {
    uint8_t whoIam = readRegister(WHO_AM_I);
    if (whoIam != 0x68) {
        return false;
    }

    writeRegister(PWR_MGMT_1, 0x80);
    delay(10);
    writeRegister(PWR_MGMT_1, 0x00);
```

```
    delay(100);
    writeRegister(PWR_MGMT_1, 0x01);
    writeRegister(INT_PIN_CFG, 0x32);
    writeRegister(CONFIG, 0x03);
    writeRegister(SMPLRT_DIV, SAMPLE_RATE_DIV_VALUE);
    writeRegister(INT_ENABLE, 0x01);
    writeRegister(GYRO_CONFIG, 0x01);
    writeRegister(ACCEL_CONFIG, 0x01);

    return true;
}
```



```
#include <Wire.h>

void writeRegister(uint8_t registerAddress, uint8_t value) {
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.write(value);
    Wire.endTransmission();
}

uint8_t readRegister(uint8_t registerAddress) {
    uint8_t readed;
    Wire.beginTransmission(mAddress);
    Wire.write(registerAddress);
    Wire.endTransmission();
    Wire.requestFrom((uint8_t)mAddress, 1);
    readed = Wire.read();
    return readed;
}

bool initialize() {
    uint8_t whoIam = readRegister(WHO_AM_I);
    if (whoIam != 0x68) {
        return false;
    }

    writeRegister(PWR_MGMT_1, 0x80);
    delay(10);
    writeRegister(PWR_MGMT_1, 0x00);
    delay(100);
    writeRegister(PWR_MGMT_1, 0x01);
    writeRegister(INT_PIN_CFG, 0x22);
    writeRegister(CONFIG, 0x03);
    writeRegister(SMPLRT_DIV, SAMPLE_RATE_DIV_VALUE);
    writeRegister(INT_ENABLE, 0x01);
```

```
    writeRegister(GYRO_CONFIG, 0x08);
    writeRegister(ACCEL_CONFIG, 0x08);

    return true;
}
```

## 3. feladat 20 pont

Most, hogy bekonfiguráltuk a szenzort, Mekk Mester úgy döntött, hogy a mikrovezérlőt sleep módba helyezi és az ULP processzorra bízza a szenzor olvasását, ezzel megakadályozva az autó akkumulátorának az esetleges lemerülését.

Az ULP programmal feladata az lesz, hogy az MPU6050 INT lábát figyelje és Data Ready interrupt esetén kiolvassa a gyorsulásmérő X tengelyén lévő értéket, és ha ez az érték nagyobb, mint egy definiált threshold, akkor élessze fel a fő processzort. A Szenzor INT lába a GPIO35-re van kötve.

Ha feltételezzük, hogy az RTC I2C És GPIO periféria beállításai megfelelőek, akkor melyik kód valósítja ezt meg helyesen?

## Válasz

```
#include "soc/rtc_cntl_reg.h"
#include "soc/rtc_io_reg.h"
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"
#include "soc/rtc_i2c_reg.h"

.macro read_i2c register, destination
  i2c_rd \register, 7, 0, 0
  move r1, \destination
  st r0, r1, 0
.endm

.set GPIO_INT, 5
.set ACCEL_X_OUT_H_REG, 0x3B
.set ACCEL_X_OUT_L_REG, 0x3C
.set ACCEL_TREASHOLD, 10000

  .bss
    .global temp_l
  temp_l:
  .long 0
    .global temp_h
```

```
  temp_h:
  .long 0

  .text
  .global entry
entry:
  wait  1000

read_accel:
  READ_RTC_REG(RTC_GPIO_IN_REG, RTC_GPIO_IN_NEXT_S + GPIO_INT, 1)
  and r0, r0, 1
  jumpr read_accel, 0, EQ

  read_i2c ACCEL_X_OUT_H_REG temp_h
  wait 1000
  read_i2c ACCEL_X_OUT_L_REG temp_l

  move r2, temp_h
  ld r1, r2, 0
  lsh r1, r1, 8
  and r1, r1, 0xFF00

  move r2, temp_l
  ld r0, r2, 0
  or r0, r1, r0
  and r0, r0, 0x7FFF

  jumpr wake_up, ACCEL_TREASHOLD, LT
  jump read_accel

end:
  halt

wake_up:
  READ_RTC_REG(RTC_CNTL_DIAG0_REG, 19, 1)
  and r0, r0, 1
  jump wake_up, eq

  wake
  WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
  halt
```



```
#include "soc/rtc_cntl_reg.h"
#include "soc/rtc_io_reg.h"
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"
```

```
#include "soc/rtc_i2c_reg.h"

.macro read_i2c register, destination
  i2c_rd \register, 7, 0, 0
  move r1, \destination
  st r1, r0, 0
.endm

.set GPIO_INT, 5
.set ACCEL_X_OUT_H_REG, 0x3B
.set ACCEL_X_OUT_L_REG, 0x3C
.set ACCEL_TREASHOLD, 10000

  .bss
    .global temp_l
  temp_l:
  .long 0
    .global temp_h
  temp_h:
  .long 0

  .text
  .global entry
entry:
  wait  1000

read_accel:
  READ_RTC_REG(RTC_GPIO_IN_REG, RTC_GPIO_IN_NEXT_S + GPIO_INT, 1)
  and r0, r0, 1
  jumpr read_accel, 0, EQ

  read_i2c ACCEL_X_OUT_H_REG temp_h
  wait 1000
  read_i2c ACCEL_X_OUT_L_REG temp_l

  move r2, temp_h
  ld r1, r2, 0
  lsh r1, r1, 8
  and r1, r1, 0xFF00

  move r2, temp_l
  ld r0, r2, 0
  or r0, r1, r0
  and r0, r0, 0x7FFF

  jumpr wake_up, ACCEL_TREASHOLD, LT
  jump read_accel
```

```
end:
  halt

wake_up:
  READ_RTC_REG(RTC_CNTL_DIAG0_REG, 19, 1)
  and r0, r0, 1
  jump wake_up, eq

  wake
  WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
  halt
```

```
#include "soc/rtc_cntl_reg.h"
#include "soc/rtc_io_reg.h"
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"
#include "soc/rtc_i2c_reg.h"

.macro read_i2c register, destination
  i2c_rd \register, 7, 0, 0
  move r1, \destination
  st r0, r1, 0
.endm

.set GPIO_INT, 5
.set ACCEL_X_OUT_H_REG, 0x3B
.set ACCEL_X_OUT_L_REG, 0x3C
.set ACCEL_TREASHOLD, 10000

  .bss
    .global temp_l
  temp_l:
  .long 0
    .global temp_h
  temp_h:
  .long 0

  .text
  .global entry
entry:
  wait  1000

read_accel:
  READ_RTC_REG(RTC_GPIO_IN_REG, RTC_GPIO_IN_NEXT_S + GPIO_INT, 1)
  and r0, r0, 1
  jumpr read_accel, 0, EQ
```

```
  read_i2c ACCEL_X_OUT_H_REG temp_h
  wait 1000
  read_i2c ACCEL_X_OUT_L_REG temp_l

  move r2, temp_h
  ld r1, r2, 0
  lsh r1, r1, 8
  and r1, r1, 0xFF00

  move r2, temp_l
  ld r0, r2, 0
  or r2, r1, r0
  and r2, r2, 0x7FFF

  jumpr wake_up, ACCEL_TREASHOLD, LT
  jump read_accel

end:
  halt

wake_up:
  READ_RTC_REG(RTC_CNTL_DIAG0_REG, 19, 1)
  and r0, r0, 1
  jump wake_up, eq

  wake
  WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
  halt


// D - helytelen: a GPIO_INT lábat 0 esetén kell újra olvasni
#include "soc/rtc_cntl_reg.h"
#include "soc/rtc_io_reg.h"
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"
#include "soc/rtc_i2c_reg.h"

.macro read_i2c register, destination
  i2c_rd \register, 7, 0, 0
  move r1, \destination
  st r0, r1, 0
.endm

.set GPIO_INT, 5
.set ACCEL_X_OUT_H_REG, 0x3B
.set ACCEL_X_OUT_L_REG, 0x3C
.set ACCEL_TREASHOLD, 10000
```

```
  .bss
    .global temp_l
  temp_l:
  .long 0
    .global temp_h
  temp_h:
  .long 0

  .text
  .global entry
entry:
  wait  1000

read_accel:
  READ_RTC_REG(RTC_GPIO_IN_REG, RTC_GPIO_IN_NEXT_S + GPIO_INT, 1)
  and r0, r0, 1
  jumpr read_accel, 1, EQ

  read_i2c ACCEL_X_OUT_H_REG temp_h
  wait 1000
  read_i2c ACCEL_X_OUT_L_REG temp_l

  move r2, temp_h
  ld r1, r2, 0
  lsh r1, r1, 8
  and r1, r1, 0xFF00

  move r2, temp_l
  ld r0, r2, 0
  or r0, r1, r0
  and r0, r0, 0x7FFF

  jumpr wake_up, ACCEL_TREASHOLD, LT
  jump read_accel

end:
  halt

wake_up:
  READ_RTC_REG(RTC_CNTL_DIAG0_REG, 19, 1)
  and r0, r0, 1
  jump wake_up, eq

  wake
  WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
  halt
```

```
#include "soc/rtc_cntl_reg.h"
#include "soc/rtc_io_reg.h"
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"
#include "soc/rtc_i2c_reg.h"

.macro read_i2c register, destination
  i2c_rd \register, 7, 0, 0
  move r1, \destination
  st r0, r1, 0
.endm

.set GPIO_INT, 5
.set ACCEL_X_OUT_H_REG, 0x3B
.set ACCEL_X_OUT_L_REG, 0x3C
.set ACCEL_TREASHOLD, 10000

  .bss
    .global temp_l
  temp_l:
  .long 0
    .global temp_h
  temp_h:
  .long 0

  .text
  .global entry
entry:
  wait  1000

read_accel:
  READ_RTC_REG(RTC_GPIO_IN_REG, RTC_GPIO_IN_NEXT_S + GPIO_INT, 1)
  and r0, r0, 1
  jumpr read_accel, 1, EQ

  read_i2c ACCEL_X_OUT_H_REG temp_h
  wait 1000
  read_i2c ACCEL_X_OUT_L_REG temp_l

  move r2, temp_h
  ld r1, r2, 0
  lsh r1, r1, 8
  and r1, r1, 0xFF00

  move r2, temp_l
  ld r0, r2, 0
  or r0, r1, r0
```

```
  and r0, r0, 0x7FFF

  jumpr wake_up, ACCEL_TREASHOLD, LT
  jump read_accel

end:
  halt

wake_up:
  READ_RTC_REG(RTC_CNTL_DIAG0_REG, 19, 1)
  and r0, r0, 1
  jump wake_up, eq

  wake
  WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
  halt
```

Megoldások beküldése