

BEÁGYAZOTT RENDSZEREK (C)

2. forduló



BOSCH

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

Útmutató:

- A **radio button-os kérdésekre** egy helyes válasz van.
- **Ha lejár a feladatlap ideje, a rendszer AUTOMATIKUSAN** beküldi azt az addig megjelölt válaszokkal.
- Az **adatbekérős feladatokra NEM jár részpontszám**, csak a feleletválasztósakra.
- **Badge-eket** a 4.forduló után kapsz majd először.
- Ha egyszerre több böngészőből, több ablakban vagy több eszközről megnyitod ugyanazt a feladatlapot, **nem tudjuk vállalni** az adatmentéssel kapcsolatban esetlegesen felmerülő anomáliákért a felelősséget!

+1: Azért szólunk, hogy senkit ne a végén érjen meglepetés: a játék nem tipp-mix és csapatkategória sincs! Természetesen akinek nem inge...

Jó versenyzést kívánunk!

2.forduló

Gipsz Szabolcs a javítás sikerén felbuzdulva úgy érzi, hogy nem is olyan bonyolult egy robotporszívó. Mivel viszonylag nagy kertje van, elhatározza, hogy épít egy hasonló elven működő robotfűnyírót.

Fő számítógépnek egy Raspberry Pi-t vagy Nvidia Jetson Nano-t szán, melyen a ROS operációs rendszert tervezi futtatni. Mivel ez egy Linuxot futtató "klasszikus" számítógép, hard realtime feladatokat nehezen lehet, illetve nem érdemes rábízni, így a perifériák kezelésére egy ARM Cortex-M alapú kártyát tervez használni. Ezzel a résszel kezd a projektet.

(Ajánlott a feladat indítása előtt az Mbed OS portálon regisztrálni és megismerkedni az online szerkesztőben megtekinthető példaprogramokkal. Alapszintű C++ tudásra is szükség lesz!)

A megoldásban a következő adatlapok lesznek a segítségedre:

Kinetis K64F adatlap: https://os.mbed.com/media/uploads/GregC/k64f_ds_rev6.pdf

Freedom board adatlap: <https://os.mbed.com/platforms/FRDM-K64F/>

Mbed OS dokumentáció: <https://os.mbed.com/>

HC-SR04 adatlap: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

Felhasznált idő: 00:00/40:00

Elért pontszám: 0/43

1. feladat 0/2 pont

Miért **nem érdemes** a fent említett fő számítógépekkel közvetlenül kezelni/vezérelni a szenzorokat/aktuátorokat?

Válaszok

- ☒ Nem (vagy csak nagyon korlátozottan) képesek hard real-time végrehajtásra.
Ez a válasz helyes, de nem jelölted meg.
- ☒ A megszakítások kezelése lassú és kiszámíthatatlan.
Ez a válasz helyes, de nem jelölted meg.
- ☒ Multitasking mellett nem lehet huzamosabb ideig determinisztikus jelet generálni GPIO-n.
Ez a válasz helyes, de nem jelölted meg.
- ☒ A taszkok ütemezése és a kontextusváltás nemdeterminisztikus.
Ez a válasz helyes, de nem jelölted meg.
- ☐ A klasszikus számítógépek CPU órajele általában magasabb, mint a mikrovezérlőké.

Magyarázat

Elméletben lehetséges egy többmagos processzorral rendelkező számítógép egy vagy több magját kizárni a fő operációs rendszer futtatásából és azokon „bare metal” kódot futtatni, így nem jelenthetjük ki egyértelműen, hogy ezen számítógépek alkalmatlanok hard realtime feladatok elvégzésére. Viszont, ha belegondolunk, hogy a memória és a perifériák eléréséhez osztolni kell az adatbuszokon a többi (nem hard realtime) feladatot futtató processzormaggal, akkor már megdőlt az elmélet. Amennyiben a „bare metal” futtatott kód és adat belefér az azt futtató processzormag L1 gyorsítótárába, valóban determinisztikus működést érhetünk el, de a perifériák kezelésénél még mindig bajban vagyunk. Egy olyan kódot futtatni, ami teljesen el van szeparálva a külvilágtól, lássuk be, nem sok értelme van. :)

Ha a fő operációs rendszert (leginkább Linuxot) futtató processzorokon – akár kernel, akár user space-ben – futó kódot nézzük, a lehetséges válaszok mindegyike igaz.

Linux operációs rendszer esetén bizonyos megszakítások feldolgozása több, mint 2 ms-ot is igénybe vehet.

Ha az adott számítógép támogatja is hardveresen bizonyos determinisztikus jelalakok kiküldését GPIO-ra, azok elindítása/megállítása nem lesz determinisztikusan vezérelhető.

A klasszikus számítógépek CPU órajele általában valóban magasabb, mint a mikrovezérlőké, de a kérdés szempontjából ennek nincs jelentősége.

2. feladat 0/14 pont

Szabolcs több ultrahangos távolságmérőt tervez elhelyezni a robotfűnyírón, hogy biztosan érzékelje az esetleges akadályokat és ki tudja kerülni azokat. Első körben a Freedom boarddal szeretné tesztelni a kiválasztott [HC-SR04](#) ultrahangos távolságérzékelőt.

Szabolcs szeretné periodikusan kiolvasni a szenzor által aktuálisan mért távolságot, az értéket pedig centiméterben kiíratni a kártya soros portjára. A hangsebességet 343 m/s-nak veszi.

Melyik kód valósítja meg helyesen az elvárt működést?

Válaszok

- ☐ 1.válasz:

```
#include "mbed.h"

DigitalOut trigger(D12);
InterruptIn echo(D13);
Timer tof_timer;
int32_t start_of_flight;
int32_t end_of_flight;
float time_of_flight;

void echo_rise(void) {
    tof_timer.reset();
    tof_timer.start();
    start_of_flight = tof_timer.read_us();
}

void echo_fall(void) {
    end_of_flight = tof_timer.read_us();
    tof_timer.stop();
    time_of_flight = end_of_flight - start_of_flight;
}

int main() {
    trigger = 0;
    time_of_flight = -1;

    echo.rise(echo_rise);
    echo.fall(echo_fall);

    while(1) {
        trigger = 1;
        wait_us(10);
        trigger = 0;
        thread_sleep_for(200);
        printf("%f\r\n", time_of_flight * 58.3f);
    }
}
```

☒ 2.válasz:

```

#include "mbed.h"

DigitalOut    trigger(D12);
InterruptIn   echo(D13);
Timer         tof_timer;
int32_t       start_of_flight;
int32_t       end_of_flight;
float         time_of_flight;

void echo_rise(void) {
    tof_timer.reset();
    tof_timer.start();
    start_of_flight = tof_timer.read_us();
}

void echo_fall(void) {
    end_of_flight = tof_timer.read_us();
    tof_timer.stop();
    time_of_flight = end_of_flight - start_of_flight;
}

int main() {
    trigger = 0;
    time_of_flight = -1;

    echo.rise(echo_rise);
    echo.fall(echo_fall);

    while(1) {
        trigger = 1;
        wait_us(10);
        trigger = 0;
        thread_sleep_for(200);
        printf("%f\r\n", time_of_flight * 0.0343f / 2);
    }
}

```

Ez a válasz helyes, de nem jelölted meg.

☐ 3.válasz:

```

#include "mbed.h"

class Sonar {
    DigitalOut    trigger;
    InterruptIn   echo;
    Timer         tof_timer;
    Ticker        periodic_read_ticker;
    int32_t       begin;
    int32_t       end;
    float         time_of_flight;

public:
    Sonar(PinName trigger_pin, PinName echo_pin) : trigger(trigger_pin), echo(echo_pin) {
        trigger = 0;
        time_of_flight = -1;

        echo.rise(callback(this, &Sonar::echo_rise));
        echo.fall(callback(this, &Sonar::echo_fall));
    }

    void start(void) {
        periodic_read_ticker.attach(callback(this, &Sonar::trig_gen), 0.001f);
    }

    void stop(void) {
        periodic_read_ticker.detach();
    }

    void echo_rise(void) {
        tof_timer.reset();
        tof_timer.start();
        begin = tof_timer.read_us();
    }

    void echo_fall(void) {
        end = tof_timer.read_us();
        tof_timer.stop();
        time_of_flight = end - begin;
    }

    void trig_gen(void) {
        trigger = 1;
        wait_us(10);
        trigger = 0;
    }

    float read(void) {
        return time_of_flight / 58.3f;
    }
};

int main() {
    Sonar sonar(D12, D13);
    sonar.start();

    while(1) {
        thread_sleep_for(200);
        printf("%f\r\n", sonar.read());
    }
}

```

✓ 4.válasz:

```

#include "mbed.h"

class Sonar {
    DigitalOut    trigger;
    InterruptIn   echo;
    Timer         tof_timer;
    Timeout       trigger_timeout;
    Ticker         periodic_read_ticker;
    int32_t       begin;
    int32_t       end;
    float         time_of_flight;

public:
    Sonar(PinName trigger_pin, PinName echo_pin) : trigger(trigger_pin), echo(echo_pin) {
        trigger = 0;
        time_of_flight = -1;

        echo.rise(callback(this, &Sonar::echo_rise));
        echo.fall(callback(this, &Sonar::echo_fall));
    }

    void start(void) {
        periodic_read_ticker.attach(callback(this, &Sonar::trig_gen), 0.05f);
    }

    void stop(void) {
        periodic_read_ticker.detach();
    }

    void echo_rise(void) {
        tof_timer.reset();
        tof_timer.start();
        begin = tof_timer.read_us();
    }

    void echo_fall(void) {
        end = tof_timer.read_us();
        tof_timer.stop();
        time_of_flight = end - begin;
    }

    void trigger_end(void) {
        trigger = 0;
    }

    void trig_gen(void) {
        trigger = 1;
        trigger_timeout.attach_us(callback(this, &Sonar::trigger_end), 10);
    }

    float read(void) {
        return time_of_flight / 58.3f;
    }
};

int main() {
    Sonar sonar(D12, D13);
    sonar.start();

    while(1) {
        thread_sleep_for(200);
        printf("%f\r\n", sonar.read());
    }
}

```

Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A hangsebességet 343 m/s-nak véve kiszámolhatjuk, hogy 1 us alatt 0.0343 cm-t tesz meg az ultrahang, tehát a mért impulzus us-ben mért szélességét (ami a hang kibocsájtásától a visszavert hang érzékeléséig eltelt idővel megegyező) ezzel kell szoroznunk, hogy megkapjuk az ultrahang által megtett távolságot. Másik oldalról megközelítve 1 cm megtételéhez a hangnak 29.15 us-re van szüksége, tehát a mért időt ezzel az értékkel kell osztani, hogy megkapjuk az ultrahang által megtett távolságot. Mivel a hang oda és vissza irányba is megteszi az utat, az előbb kapott értékeket még meg kell feleznünk.

Az első megoldásnál a 29.15 us duplájával osztás helyett szoroztunk, ami hibás.

A második és a negyedik megoldás helyes. A második kompaktabb, a negyedik pedig jobban illeszthető komplexebb programokba.

A harmadik megoldásban 1 ms-onként küldjük ki a trigger jelet, ami túl gyors. Ennyi idő alatt a hang 34.3 cm-t tesz meg, tehát, 17 cm-nél távolabbi objektumok detektálása esetén esély sincs a mérés befejezésére, mielőtt a következő mérést elindítanánk.

3. feladat 0/18 pont

A robotfűnyírón a tervek szerint hat darab UH érzékelő lesz majd, így Szabolcs úgy dönt, hogy ezek kezelését egy külön dedikált mikrovezérlő fogja végezni. Miután stabilan tudja kezelni a szenzort, szeretné I2C interfészen keresztül kiolvashatóvá tenni az aktuális mért értékeket. A Freedom board D14/D15-ös (SDA/SCL) lábain keresztül szeretné az I2C kapcsolatot megvalósítani.

Az egyes szenzorokat 1-6 számokkal szeretné megcímezni, a címzést pedig egy 1 byte-os master -> slave üzenet küldésével tervezi megvalósítani. Ha a nullás „szenzort” címzi az I2C master, akkor az eszköz nevét küldi vissza a slave, ASCII kódolással, lezáró null karakterrel. Ha 6-osnál nagyobb értékkel címeznénk, akkor az „N/A” szöveget ugyanígy. Egyébként 2 byte-os (uint16_t) raw formátumban adja át a vezérlő a mért értékeket, amit Little-Endian bájtsorrenddel küld.

Melyik kód valósítja meg helyesen az elvárt működést?

Válaszok

☐ 1.válasz:

```
#include <mbed.h>

#define SENSOR_COUNT 6

I2C i2c(PTE25, PTE24);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    switch(selectedSensorId) {
        case 0: i2c.write(device_name, strlen(device_name) + 1);
                break;
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6: i2c.write((char*)sensorValues[selectedSensorId-1], 2);
                break;
        default: i2c.write("N/A", 4);
    }
}

int main() {
    i2c.address(0xA0);
    while (1) {
        int i = i2c.receive();
        switch (i) {
            case i2cslave::ReadAddressed:
                respond();
                break;
            case i2cslave::WriteGeneral:
            case i2cslave::WriteAddressed:
                i2c.read(&selectedSensorId, 1);
                break;
        }
    }
}
```

☐ 2.válasz:

```

#include <mbed.h>

#define SENSOR_COUNT 6

I2CSlave i2cs(PTE24, PTE25);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    switch(selectedSensorId) {
        case 0: i2cs.write(device_name, strlen(device_name) + 1);
                break;
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6: i2cs.write((char*)sensorValues[selectedSensorId-1], 2);
                break;
        default: i2cs.write("N/A", 3);
    }
}

int main() {
    i2cs.address(0xA0);
    while (1) {
        int i = i2cs.receive();
        switch (i) {
            case I2CSlave::ReadAddressed:
                respond();
                break;
            case I2CSlave::WriteGeneral:
            case I2CSlave::WriteAddressed:
                i2cs.read(&selectedSensorId, 1);
                break;
        }
    }
}

```

☐ 3.válasz:


```

#include <mbed.h>

#define SENSOR_COUNT 6

I2CSlave i2cs(PTE24, PTE25);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    switch(selectedSensorId) {
        case 0: i2cs.write(device_name, strlen(device_name) + 1);
                break;
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6: i2cs.write((char*)sensorValues[selectedSensorId-1], 2);
                break;
        default: i2cs.write("N/A", 4);
    }
}

int main() {
    i2cs.address(0xA0);
    while (1) {
        int i = i2cs.receive();
        switch (i) {
            case I2CSlave::ReadAddressed:
                respond();
                break;
            case I2CSlave::WriteGeneral:
            case I2CSlave::WriteAddressed:
                i2cs.read(&selectedSensorId, 1);
                break;
        }
    }
}

```

☒ 4.válasz:

```

#include <mbed.h>

#define SENSOR_COUNT 6

I2CSlave i2cs(PTE25, PTE24);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    switch(selectedSensorId) {
        case 0: i2cs.write(device_name, strlen(device_name) + 1);
                break;
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6: i2cs.write((char*)sensorValues[selectedSensorId-1], 2);
                break;
        default: i2cs.write("N/A", 4);
    }
}

int main() {
    i2cs.address(0xA0);
    while (1) {
        int i = i2cs.receive();
        switch (i) {
            case I2CSlave::ReadAddressed:
                respond();
                break;
            case I2CSlave::WriteGeneral:
            case I2CSlave::WriteAddressed:
                i2cs.read(&selectedSensorId, 1);
                break;
        }
    }
}

```

Ez a válasz helyes, de nem jelölted meg.

☐ 5.válasz:

```

#include <mbed.h>

#define SENSOR_COUNT 6

I2CSlave i2cs(PTE25, PTE24);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    if(selectedSensorId == 0)
        i2cs.write(device_name, strlen(device_name) + 1);
    else if(selectedSensorId > 6)
        i2cs.write("N/A", 4);
    else
        i2cs.write((char*)sensorValues[selectedSensorId-1], 2);
}

int main() {
    i2cs.address(0xA0);
    while (1) {
        int i = i2cs.receive();
        switch (i) {
            case I2CSlave::ReadAddressed:
                respond();
                break;
            case I2CSlave::WriteGeneral:
            case I2CSlave::WriteAddressed:
                i2cs.read(selectedSensorId, 1);
                break;
        }
    }
}

```

☒ 6.válasz:

```

#include <mbed.h>

#define SENSOR_COUNT 6

I2CSlave i2cs(PTE25, PTE24);
char device_name[] = "HC-SR04!";

uint8_t selectedSensorId = 0;
uint16_t sensorValues[SENSOR_COUNT];

void respond() {
    // If sensorId 0 is selected, device name is returned, for other ids the actual sensor values
    if(selectedSensorId == 0)
        i2cs.write(device_name, strlen(device_name) + 1);
    else if(selectedSensorId > 6)
        i2cs.write("N/A", 4);
    else {
        i2cs.write(sensorValues[selectedSensorId-1] >> 8);
        i2cs.write(sensorValues[selectedSensorId-1] & 0xff);
    }
}

int main() {
    i2cs.address(0xA0);
    while (1) {
        int i = i2cs.receive();
        switch (i) {
            case I2CSlave::ReadAddressed:
                respond();
                break;
            case I2CSlave::WriteGeneral:
            case I2CSlave::WriteAddressed:
                i2cs.read(&selectedSensorId, 1);
                break;
        }
    }
}

```

Ez a válasz helyes, de nem jelölted meg.

Magyarázat

Az első megoldásnál az I2C (Master) osztályt próbáljuk használni, a célunk viszont egy I2C slave eszköz létrehozása.

A második és harmadik megoldásnál felcseréltük az SDA és SCL lábakat, valamint a másodiknál az „N/A” utáni lezáró nullát sem küldjük ki.

Az ötödik megoldásnál a `selectedSensorId` változó címe helyett annak értékét adjuk át az `i2cs.read()` függvénynek, ami rossz, ráadásul veszélyes memóriakorruptiót is eredményez(het).

A negyedik és a hatodik megoldás is jó, mivel a Kinetis K64F (és az ARM Cortex-M vezérlők úgy általában) Little-Endian bájtsorrendet használnak.

4. feladat 0/9 pont

Szabolcs már visszaadta a kölcsönkapott Nucleo-t, de nemrég kapott egy barátjától egy Freedom boardot, rajta egy 120 MHz-es ARM Cortex-M magos [Kinetis K64F](#) mikrovezérlővel. Úgy érzi, ez jó lesz kísérletezéshez. Ehhez a kártyához elérhető az ARM [Mbed OS](#) szoftverkörnyezete, amivel könnyen és gyorsan ki lehet próbálni technikai elgondolásokat. Segíts neki ismerkedni a kártyával és az Mbed OS-szel!

Melyik kód valósítja meg helyesen a kártyán lévő RGB LED másodpercenkénti villogtatását 50%-os kitöltési tényezővel?

Válaszok

☐ 1.válasz:

```
#include "mbed.h"
#include "platform/mbed_thread.h"

DigitalOut LED1;

int main() {
    while(true) {
        LED1 = !LED1;
        thread_sleep_for(500);
    }
}
```

☒ 2.válasz:

```
#include "mbed.h"

int main() {
    DigitalOut led(LED_GREEN);
    while(1) {
        led = !led;
        wait_us(500000);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

☐ 3.válasz:


```
#include "mbed.h"
#include "platform/mbed_thread.h"

int main() {
    PwmOut pwm_led(LED_BLUE);

    pwm_led.period_ms(1000);
    pwm_led.write(0.5f);

    while(1) {
        thread_sleep_for(1000);
    }
}
```

☒ 4.válasz:



```
#include "mbed.h"
#include "platform/mbed_thread.h"

DigitalOut led(LED_RED);
Ticker ticker;

void toggleLed() {
    led = !led;
}

int main()
{
    ticker.attach(&toggleLed, 0.5f);

    while(1) {
        thread_sleep_for(1000);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.



5.válasz:

```

#include "mbed.h"
#include "platform/mbed_thread.h"

DigitalOut *blinkingLED;
DigitalOut greenLED(LED_GREEN);
DigitalOut redLED(LED_RED);
DigitalIn btn(PTA4);

Thread *thread;
int blinkingLedId = 0;

void threadMain(void) {
    blinkingLED = &greenLED;

    while(true) {
        if(blinkingLedId == 1)
            blinkingLED = &redLED;
        else
            blinkingLED = &greenLED;
        *blinkingLED = !(*blinkingLED);
        ThisThread::sleep_for(500);
        *blinkingLED = !(*blinkingLED);
        ThisThread::sleep_for(500);
    }
}

int main()
{
    greenLED = 1;
    redLED.write(1);
    int btnPrevState = btn.read();

    thread = new Thread();
    thread->start(callback(threadMain));

    while(true) {
        int btnCurr = btn.read();
        if(btnCurr != btnPrevState) {
            if(btnCurr == 1) {
                blinkingLedId = (blinkingLedId+1) % 2;
            }
            btnPrevState = btnCurr;
        }
        thread_sleep_for(100);
    }
}

```

Ez a válasz helyes, de nem jelölted meg.

Az első megoldásnál nem rendeltük hozzá a LED-hez tartozó GPIO-t a DigitalOut objektumhoz, aminek igazából nincs is default konstruktora, így fordítási hibát kapunk.

A második megoldásnál ezt a hibát korrigáltuk, így ez egy jó megoldás.

A PWM-es megoldás jó lenne, de a K64F kártyán az mbed OS alapból nem támogat ilyen hosszú periódusidőt (a háttérben futó timer csak 16 bites), valamint a fedélzeti RGB LED csatornái nem támogatják a PWM-et.

A Ticker használata jó megoldás ilyen esetekben, így itt is.

A Thread-es megoldás trükkös, mert a piros és a zöld csatornáját is vezérli a fedélzeti RGB LED-nek gombnyomás függvényében, de a kiírt feladatot teljesíti.



[Legfontosabb tudnivalók](#) [Kapcsolat](#) [Versenyszabályzat](#) [Adatvédelem](#)

© 2023 Human Priority Kft.

KÉSZÍTETTE 

Megjelenés

 Világos 