BEÁGYAZOTT RENDSZEREK (C)

5. forduló

BOSCH

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

Gipsz Szabolcs szeretné sárga villogókkal felszerelni a robotfűnyírót, melyek üzem közben jelzik a veszélyt.

Úgy dönt, hogy a villogók vezérlésére egy ATtiny13-at használ, mely különböző üzemmódokban tudja vezérelni a villogókat, az üzemmódok között pedig egy digitális bemenetén kapott jel segítségével lehet váltani.

A megoldásban a következő adatlapok lesznek a segítségedre:

ATtiny13 adatlap: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2535.pdf

Application note on timers in AVRs: https://onlinedocs.microchip.com/pr/GUID-93DE33AC-A8E1-4DD9-BDA3-C76C7CB80969-en-US-2/index.html?GUID-0BED8014-6B01-47DD-A495-68F9F70287A3

Felhasznált idő: 00:00/30:00 **Elért pontszám:** 0/32

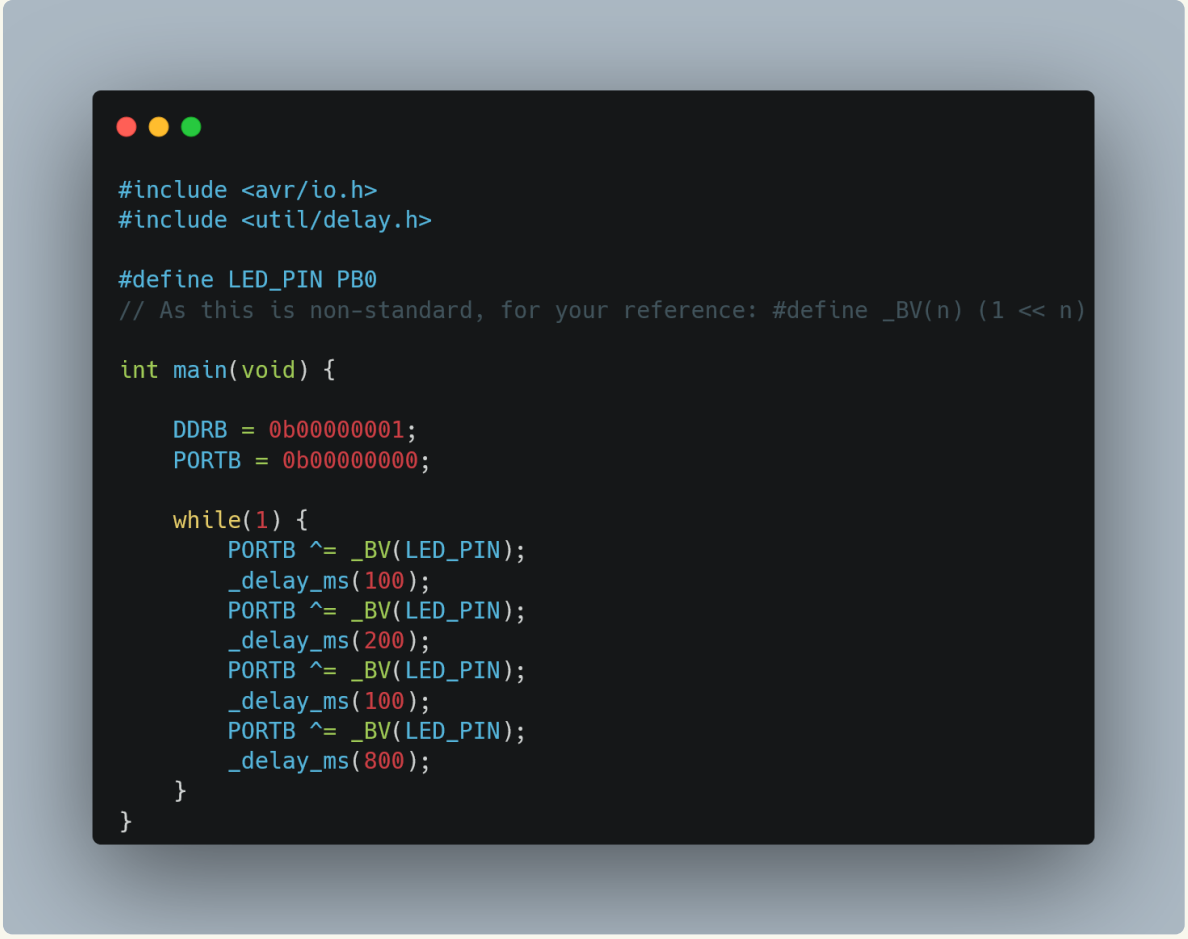
1. feladat 0/9 pont

Szabolcs szeretné a következő ütemezéssel villogtatni a villogókat: 0.1 mp be, 0.2 mp ki, 0.1 mp be, 0.8 mp ki. Az időzítés terén közelítő pontossággal is megelégszik. A villogókat a PB0 kimeneten keresztül vezérli aktív magas módban. Az ATtiny a belső 9.6 MHz-es rezonátort használja a gyárilag beállított CKDIV8 prescalerrel.

Melyik kód valósítja meg az elvárt működést?

Válaszok

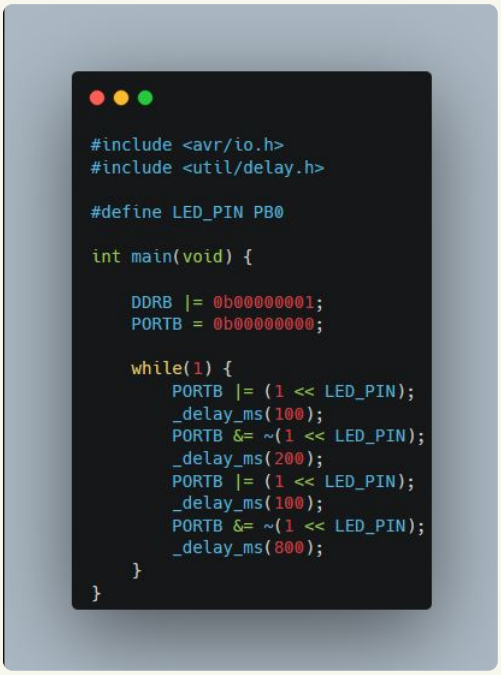
- [x] 1.válasz:



```c
#include <avr/io.h>
#include <util/delay.h>

#define LED_PIN PB0
// As this is non-standard, for your reference: #define _BV(n) (1 << n)

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    while(1) {
        PORTB ^= _BV(LED_PIN);
        _delay_ms(100);
        PORTB ^= _BV(LED_PIN);
        _delay_ms(200);
        PORTB ^= _BV(LED_PIN);
        _delay_ms(100);
        PORTB ^= _BV(LED_PIN);
        _delay_ms(800);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

2.válasz:



```c
#include <avr/io.h>
#include <util/delay.h>

#define LED_PIN PB0

int main(void) {

    DDRB |= 0b00000001;
    PORTB = 0b00000000;

    while(1) {
        PORTB |= (1 << LED_PIN);
        _delay_ms(100);
        PORTB &= ~(1 << LED_PIN);
        _delay_ms(200);
        PORTB |= (1 << LED_PIN);
        _delay_ms(100);
        PORTB &= ~(1 << LED_PIN);
        _delay_ms(800);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

3.válasz:

```c
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED_PIN PB0

int counter = 0;

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(counter == 0)
        PORTB |= (1 << LED_PIN);
    if(counter == 2)
        PORTB &= ~(1 << LED_PIN);
    if(counter == 6)
        PORTB |= (1 << LED_PIN);
    if(counter == 8)
        PORTB &= ~(1 << LED_PIN);
    if(counter++ == 22)
        counter = 0;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    while(1);
}
```

Ez a válasz helyes, de nem jelölted meg.

☐ 4.válasz:

```c
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED_PIN PB0

int counter = 0;

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(counter == 0)
        PORTB |= (1 << LED_PIN);
    if(counter == 2)
        PORTB &= ~(1 << LED_PIN);
    if(counter == 6)
        PORTB |= (1 << LED_PIN);
    if(counter == 8)
        PORTB &= ~(1 << LED_PIN);
    if(counter++ == 22)
        counter = 0;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS01)|(1<<CS00);
    TIMSK0 |= (1 << TOIE0);
    sei();

    while(1);
}
```

Magyarázat

Az első és a második megoldás is jó, a különbség abban rejlik, hogy az első megolásban XOR-t használunk, a második megoldásban pedig barátságos értékadásokkal konstans értékeket írunk a PB0-ra.

A harmadik megoldás az ATtinyban található 8 bites timert használja /256-os prescalerrel. Ez is jó megoldás, bár az 1.2 MHz-es órajelet kétszer 256-tal osztva 18.31 Hz-et kapunk, így 50 ms-os ciklusidő helyett picivel hosszabbal tudunk dolgozni, de ezt a counter értékével kompenzálva a teljes villogtatási ciklusidő 1.2 ms-ra közelíti a tervezett 1200-at, ami ilyen alkalmazás esetén elfogadható pontosság.

A negyedik megoldásban a TCCR0B CS00 és CS01 bitjeinek a konfigurálásával /64-es prescalert állítunk be, így (a kommentben is jelzett várakozásunkhoz képest) az időzítésünk nem lesz jó.

2. feladat 0/5 pont

Szabolcs szeretné az ATtiny PB1 lábán keresztül fogadott impulzussal ki- és bekapcsolni a villogást, de véletlenül benne hagyott egy hibát a kódban, ami miatt az nem működik.

Segíts neki megtalálni! Hányadik sorban van a hiba (számmal add meg!)

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_PIN PB0
#define BTN_PIN PB1

int counter = 0;
char flashEn = 1;

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(flashEn) {
        if(counter == 0)
            PORTB |= (1 << LED_PIN);
        if(counter == 2)
            PORTB &= ~(1 << LED_PIN);
        if(counter == 6)
            PORTB |= (1 << LED_PIN);
        if(counter == 8)
            PORTB &= ~(1 << LED_PIN);
        if(counter++ == 22)
            counter = 0;
    }
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    char btnPrevState = 0;

    while(1) {
        char btnState = PINB & (1 << BTN_PIN);
        if(btnState != btnPrevState) {
            if(btnState)
                flashEn ^= flashEn;
            btnPrevState = btnState;
        }
    _delay_ms(20);
    }
}
```

Válasz

A helyes válasz:

42

Magyarázat

A 42. sorban van a hiba.

Szabolcs szerette volna a villogást engedélyező változó értékét invertálni, ehhez a flashEn ^= 1; utasítást használni, de véletlenül saját magával XOR-olta a változó értékét.

3. feladat 0/18 pont

Szabolcs úgy dönt, hogy a villogókat hibajelzésre is használni fogja. Ha a robotfűnyíró olyan helyzetbe kerül, melyből nem képes önerőből kikerülni (vagy valamilyen belső hibát észlel), akkor a villogókat folyamatosan világító helyzetbe kapcsolva jelzi, hogy beavatkozásra vár.

Ehhez a meglévő kódot úgy tervezi módosítani, hogy a "ki", "folyamatos üzem" és "hibajelzés" üzemmódok között a PB1 digitális bemenetre adott legalább 0.22 mp széles impulzussal lehet váltani, valamint "folyamatos üzem" állapotból automatikusan is át tud kapcsolni "hibajelzés" állapotba, ha adott időn belül nem kap ún. heartbeat jelzést a központi vezérlőtől.

A heartbeat jelzés egy maximum 0.11 mp széles impulzus, melyet szintén a PB1 digitális bemeneten várunk. A két jelzés között eltelt idő normál üzem esetén maximum 5 mp lehet. Ha ennél hosszabb ideig nem kap heartbeat jelzést az ATtiny, akkor "folyamatos üzem" állapotból "hibajelzés" állapotba kapcsol.

Innen egy újabb heartbeat észlelése esetén visszatérhet "folyamatos üzem" állapotba, egy legalább 0.22 mp széles impulzussal pedig "ki" állapotba vezérelhető.

Figyelem! Ha az impulzusunk "végtelen széles" lenne, azt nem vesszük figyelembe, tehát szükség van a lefutó él észlelésére is a kapcsoláshoz.

Melyik kód valósítja meg helyesen az elvárt működést?

Válasz

1.válasz:

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_PIN PB0
#define BTN_PIN PB1
#define HBEAT_TMOUT 92

unsigned int counter = 0;
char flashMode = 1;    // 0 = off, 1 = normal, 2 = steady

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(flashMode == 0)
            PORTB &= ~(1 << LED_PIN);
    if(flashMode == 1) {
        if(counter == 0)
            PORTB |= (1 << LED_PIN);
        if(counter == 2)
            PORTB &= ~(1 << LED_PIN);
        if(counter == 6)
            PORTB |= (1 << LED_PIN);
        if(counter == 8)
            PORTB &= ~(1 << LED_PIN);
    }
    if(flashMode == 2)
            PORTB |= (1 << LED_PIN);
    if(counter++ == 22)
        counter = 0;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    char btnPrevState = 0;
    unsigned int pressTime = -1;
    unsigned int hbTime = 0;

    while(1) {
        char btnState = PINB & (1 << BTN_PIN);
        if(btnState != btnPrevState) {
            if(btnState)
                pressTime = counter;
            else {
                if(counter - pressTime < 2) {
                    hbTime = counter;
                    if(flashMode == 2)
                        flashMode = 1;
                }
                if(counter - pressTime > 4)
                    if(flashMode++ == 2)
                        flashMode = 0;
                    if(flashMode == 1)
```

```
                    hbTime = counter;
                }
                btnPrevState = btnState;
            }
            if(flashMode == 1 && counter - hbTime > HBEAT_TMOUT)
                flashMode = 2;
            _delay_ms(20);
        }
    }
```

2.válasz:

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_PIN PB0
#define BTN_PIN PB1
#define HBEAT_TMOUT 92

unsigned int counter = 0;
char flashMode = 1;    // 0 = off, 1 = normal, 2 = steady

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(flashMode == 0)
            PORTB &= ~(1 << LED_PIN);
    if(flashMode == 1) {
        if(counter % 22 == 0)
            PORTB |= (1 << LED_PIN);
        if(counter % 22 == 2)
            PORTB &= ~(1 << LED_PIN);
        if(counter % 22 == 6)
            PORTB |= (1 << LED_PIN);
        if(counter % 22 == 8)
            PORTB &= ~(1 << LED_PIN);
    }
    if(flashMode == 2)
            PORTB |= (1 << LED_PIN);
    counter++;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    char btnPrevState = 0;
    unsigned int pressTime = -1;
    unsigned int hbTime = 0;

    while(1) {
        char btnState = PINB & (1 << BTN_PIN);
        if(btnState != btnPrevState) {
            if(btnState)
                pressTime = counter;
            else {
                if(counter - pressTime < 2) {
                    hbTime = counter;
                    if(flashMode == 2)
                        flashMode = 1;
                }
                if(counter - pressTime > 4)
                    if(flashMode++ == 2)
                        flashMode = 0;
                    if(flashMode == 1)
                        hbTime = counter;
```

```
            }
            btnPrevState = btnState;
        }
        if(flashMode == 1 && counter - hbTime > HBEAT_TMOUT)
            flashMode = 2;
        _delay_ms(20);
    }
}
```

Ez a válasz helyes, de nem jelölted meg.

3.válasz:

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_PIN PB0
#define BTN_PIN PB1
#define HBEAT_TMOUT 92

int counter = 0, lcounter = 0;
char flashMode = 1;    // 0 = off, 1 = normal, 2 = steady

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(flashMode == 0)
            PORTB &= ~(1 << LED_PIN);
    if(flashMode == 1) {
        if(lcounter == 0)
            PORTB |= (1 << LED_PIN);
        if(lcounter == 2)
            PORTB &= ~(1 << LED_PIN);
        if(lcounter == 6)
            PORTB |= (1 << LED_PIN);
        if(lcounter == 8)
            PORTB &= ~(1 << LED_PIN);
    }
    if(flashMode == 2)
            PORTB |= (1 << LED_PIN);
    if(lcounter++ == 22)
        lcounter = 0;
    counter++;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    char btnPrevState = 0;
    int pressTime = -1;
    int hbTime = 0;

    while(1) {
        char btnState = PINB & (1 << BTN_PIN);
        if(btnState != btnPrevState) {
            if(btnState)
                pressTime = counter;
            else {
                if(counter - pressTime < 2) {
                    hbTime = counter;
                    if(flashMode == 2)
                        flashMode = 1;
                }
                if(counter - pressTime > 4)
                    if(flashMode++ == 2)
                        flashMode = 0;
```

4.válasz:

```
                    if(flashMode == 1)
                        hbTime = counter;
            }
            btnPrevState = btnState;
        }
        if(flashMode == 1 && counter - hbTime > HBEAT_TMOUT)
            flashMode = 2;
        _delay_ms(20);
    }
}
```

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_PIN PB0
#define BTN_PIN PB1
#define HBEAT_TMOUT 92

unsigned int counter = 0;
char flashMode = 1;    // 0 = off, 1 = normal, 2 = steady

// Runs once each 54.6 ms
ISR(TIM0_OVF_vect) {
    if(flashMode == 0)
            PORTB &= ~(1 << LED_PIN);
    if(flashMode == 1) {
        if(counter % 22 == 0)
            PORTB |= (1 << LED_PIN);
        if(counter % 22 == 2)
            PORTB &= ~(1 << LED_PIN);
        if(counter % 22 == 6)
            PORTB |= (1 << LED_PIN);
        if(counter % 22 == 8)
            PORTB &= ~(1 << LED_PIN);
    }
    if(flashMode == 2)
            PORTB |= (1 << LED_PIN);
    counter++;
}

int main(void) {

    DDRB = 0b00000001;
    PORTB = 0b00000000;

    TCCR0B |= (1<<CS02);
    TIMSK0 |= (1 << TOIE0);
    sei();

    char btnPrevState = 0;
    unsigned int pressTime = -1;
    unsigned int hbTime = 0;

    while(1) {
        char btnState = PINB & (1 << BTN_PIN);
        if(btnState != btnPrevState) {
            if(btnState)
                pressTime = counter;
            else {
                if(counter - pressTime < 2) {
                    hbTime = counter;
                }
                if(counter - pressTime > 4)
                    if(flashMode++ == 2)
                        flashMode = 0;
                    if(flashMode == 1)
                        hbTime = counter;
            }
            btnPrevState = btnState;
```

```
        }
        if(flashMode == 1 && counter - hbTime > HBEAT_TMOUT)
            flashMode = 2;
        if(flashMode == 2 && counter - hbTime < HBEAT_TMOUT)
            flashMode = 1;
        _delay_ms(20);
    }
}
```

Magyarázat

Az első megoldás majdnem jó: Szabolcs felhasználta a counter változót, gondolván a túlcsordulására unsigned-dá is tette, de a kódban felejtette azt a részt, ami 22-nél nullázza a számlálót.

Szabolcs a második megoldásban korrigálta a fenti hibát, a counter-t folyamatosan növeli, ami 65535 után átfordul, de ezt kezeli. a LED villogtatásnál pedig modulo 22-vel oldotta meg, hogy kellő ideig legyen a LED a villogás minden fázisában. Ez tehát a jó megoldás.

A harmadik megoldásban Szabolcs egy külön counter bevezetésével szeretné orvosolni az első megoldás problémáját, de unsigned helyett signed int-et használ, így a számláló túlcsordulásakor nem jó különbségeket fog kapni, amit összehasonlítana a beállított határokkal. Így – bár az esetek többségében jól működne a vezérlés, határhelyzetben hibázna.

A negyedik megoldásban a hibaüzemből visszakerülünk a folyamatos üzembe új hearbeat jel érkezése nélkül is, amikor a számláló körülfordulás után újra kellő közelségbe ér, így ez nem jó megoldás.