21. Megszakítások a szoftverben - interruptok

Írta: Farkas Péter, Apró Csaba

Lektorálta: Veréb Szabolcs

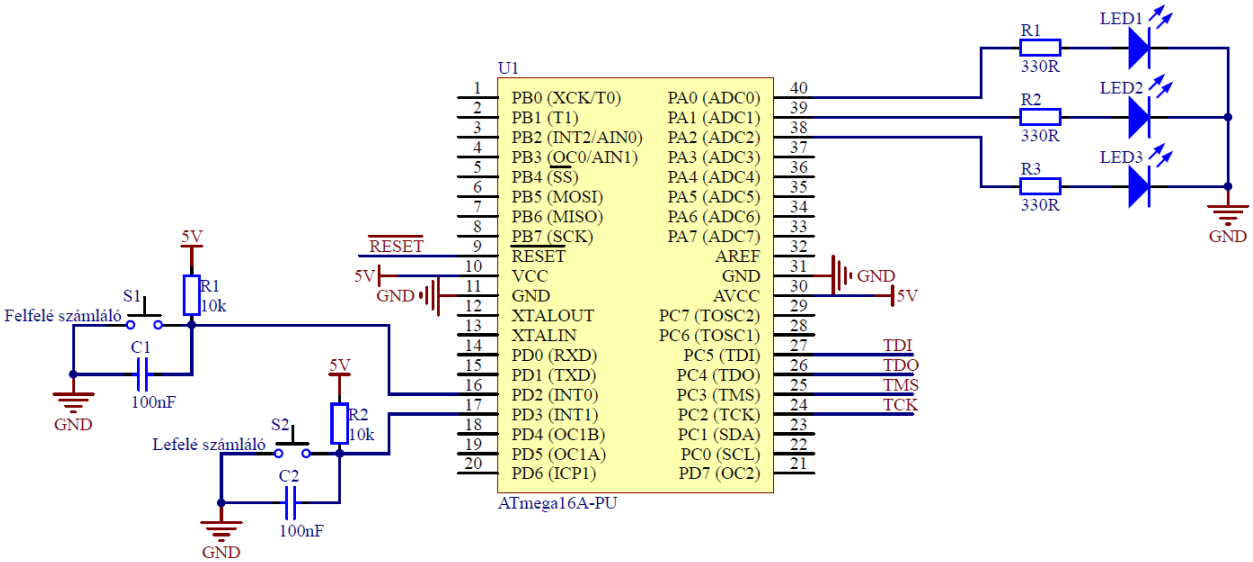
BEVEZETŐ

A minket körülvevő elektronikai eszközök, rendszerek szinte sosem önmagukban működnek, hanem a külvilágból érkező különböző impulzusokra, behatásokra reagálnak. Maguk a rendszerek és a bemeneteik lehetnek viszonylag egyszerűek, mint például egy távirányító és a gombjai, vagy rendkívül összetettek is, mint például egy utasszállító repülő robotpilótája. Bár a robotpilóta rendkívül izgalmasan hangzik, mi most mégis inkább maradnánk egy egyszerűbb példánál, annak érdekében, hogy megérthessük a mikrovezérlők egy alapvető egységének működését és megtanuljuk annak használatát.

PÉLDAÁRAMKÖR

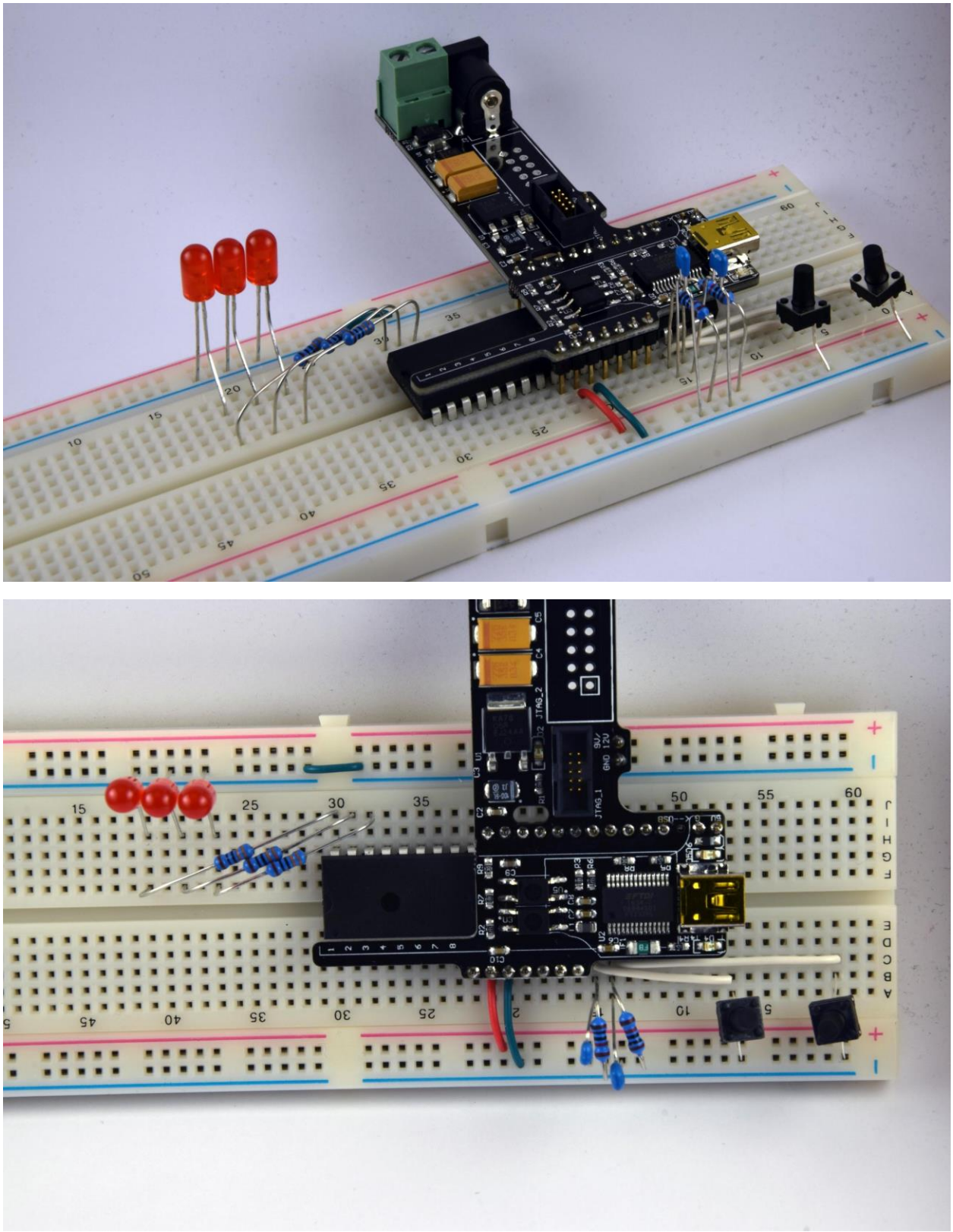
Ebben a fejezetben egy olyan programot fogunk készíteni, mely gombnyomásra számol 0 és 7 között, az aktuális értéket 3 LED segítségével jeleníti meg, bináris formában. Az egyik gomb növeli, a másik pedig csökkenti a számláló értékét.

Az ATMEGA kontrollerünk PD2 lábára fogjuk a felfelé, PD3 lábára pedig a lefelé számláló nyomógombot kötni, a “kijelzőnk”, azaz a 3 LED pedig kerüljön a PA0, PA1 és PA2 lábakra. A kapcsolási rajzot az alábbi ábra mutatja.



Ez a példány Rába Ármin kizárólagos használatú példánya.

http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.



1. ábra *- Példaáramkör kapcsolási rajza*

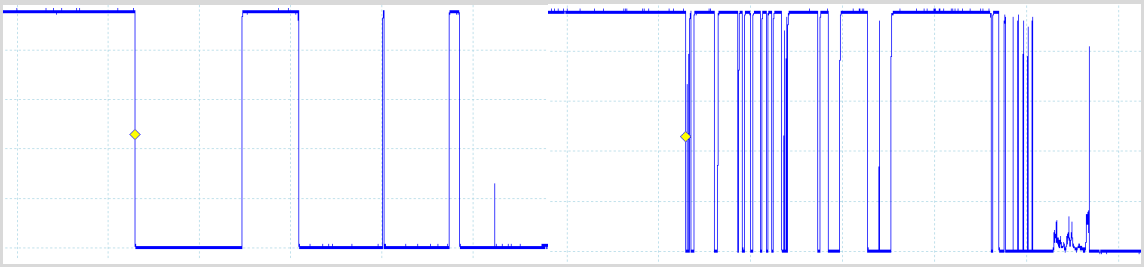
Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

A gomb és a táp között egy felhúzó ellenállás található.

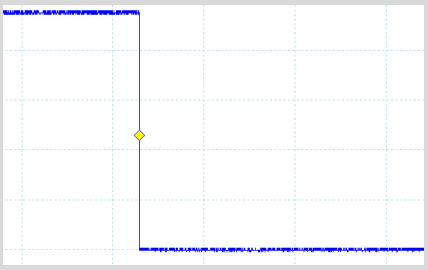
Kitekintés

A kondenzátorra a mikrokontroller lába és a föld között azért van szükség, hogy kiszűrjük a pergés (prell) nevű jelenséget. De mi az a pergés?

Egy ideális nyomógomb megnyomáskor azonnal zárja az áramkört, elengedéskor pedig nyitja. A valóságban azonban, mielőtt ténylegesen záródna a kör, a két fém érintkező többször, nagyon rövid időre összeér, majd eltávolodik egymástól, akárcsak egy földre ejtett labda pattogása, illetve annak csillapodása. Ezt mi emberek egy lámpa kapcsolójánál észre sem vesszük, egy kontroller viszont nagyon gyors, ezért a pergéseket is külön gombnyomásként értelmezheti. A bekötött (szűrő)kondenzátor ezeket a gyors impulzusokat szűri ki. Az alábbi képen oszcilloszkóppal készült felvételek láthatók egy gomb lenyomásáról felengedéséről, amin az első esetben nem használunk kondenzátort, a másikban viszont igen.



***2. ábra** - Nyomógomb pergése*



***3. ábra** - Pergésmentesítés hardveresen*

Így tehát sikerült hardveresen kiküszöbölnünk a pergést. Létezik természetesen szoftveres megoldás is a pergésmentesítésre, amivel fogunk is foglalkozni, de ne szaladjunk ennyire előre, kezdjük az elején.

HOGYAN VEGYÜK ÉSZRE A GOMBNYOMÁST?

Az egyik megoldás az, hogy a main függvényben egy végtelen ciklusban figyeljük az adott gombhoz tartozó bemenetet. Ha annak az értéke 1, akkor tudjuk, hogy valaki lenyomta a gombot, így a számláló értékét növelhetjük vagy csökkenthetjük.

Esetünkben olyan különbség jelentkezik, hogy a gombunk úgynevezett 0 aktív kapcsolásban van, ami azt jelenti, hogy a gombot nem piszkálva (nem lenyomva), a mikrovezérlő logikai magas értéket érzékel a felhúzó ellenállás miatt. A gombot vezérelve (lenyomva), az a föld felé fémes kontaktust biztosít, azaz

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

logikai 0 (0V) értékű az ellenállásból és nyomógombból álló kapcsolás kimenete. Ezt figyelembe véve a programunkban, a PIND adott bitjeinek 0 értéke jelenti azt, hogy a gombot megnyomtuk.

A korábban már használt io.c fájlban a megismert módon konfiguráljuk fel a portokat bemeneteknek és kimeneteknek, illetve kódoljuk le a választott megoldást megvalósító programot:

```
#include "../Headers/main.h"

volatile uint8_t cntr = 0;
int main(void)
{
	//IO inicializálása
	IOInit();

	//Végtelen ciklus
	while (1)
	{
		//Ha a PD2 lábra kötött gombot lenyomjuk ÉS
		// a számláló még nem érte el a maximumot
		if (!(PIND & 0x04) && (cntr < 7))
		{
			cntr++;
		}
		//Ha a PD3 lábra kötött gombot lenyomjuk ÉS
		// a számláló még nem érte el a minimumot
		else if (!(PIND &  0x08) && (cntr > 0))
		{
			cntr--;
		}
		//LED-ek kigyújtása a számlálónak megfelelően
		PORTA = cntr;
	}
	return (0);
}
```

Tehát a főfüggvényben (`main()`) a mikrovezérlő lábainak irányát és értékét beállító függvény (`IOInit()`) hívása után, a végtelen ciklusban növeljük vagy csökkentjük a számlálót, abban az esetben, ha valamelyik gombot lenyomtuk, figyelembe véve a számláló által felvehető maximális és minimális értékeket.

Az if feltételében található felkiáltójel a logikai tagadás operátor, így például a `!(PIND & 0x04)` kifejezés akkor logikai igaz, ha a PD2 láb 0 értékű, tehát pont akkor, amikor megnyomtuk a gombot. Az adott végtelen ciklus végén pedig kikerül a számláló értéke a LED-ekre.

Mit tapasztalunk? A gombok lenyomásának pillanatában vagy bekapcsol az összes LED vagy kikapcsol.

Nem pont ezt akartuk. De mi okozza ezt a furcsa működést? Gondoljunk bele: egy átlagos gombnyomás meddig tarthat? Bármilyen gyorsan is nyomjuk meg, majd engedjük fel, néhány század- vagy akár tizedmásodpercnyi ideig biztosan nyomva tartjuk. A processzorunknak mekkora az órajele? 8 Mhz. Ahhoz, hogy a main függvényben futó egyszerű kis while ciklus egyszer lefusson, egynél jóval több órajelnyi idő kell, de még így is az ezred másodperces (vagy kisebb) tartomány körül mozgunk. Tehát az alatt a

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

számunkra csak egy pillanatnak tűnő idő alatt, amíg nyomva tartjuk a gombot, a program akár több százszor is megvizsgálhatja a bemenet értékét, amit minden alkalommal 1-nek talál, így minden alkalommal növeli vagy csökkenti a számláló értékét, amíg el nem éri a számlálónk szélső értékét (0 vagy 7). Mindez olyan gyorsan történik, hogy a LED-ek ki-/bekapcsolgatása a szemünk számára összemosódik, úgyhogy mi csak a végeredményt látjuk.

Mi a megoldás? Természetesen elmenthetnénk a gomb aktuális állapotát (0 vagy 1) és csak akkor léptetnénk föl vagy lefelé a számlálót, ha az előző vizsgálatkor a bemenet értéke 0 volt, most pedig 1. (Azaz a felfutó élre, tehát a 0-1 átmenetre figyelünk). Így csak a gomb elengedésének pillanatában változna a számláló értéke.

Ezzel a módszerrel azonban van még egy jelentős probléma, ami a példánkban ugyan nem jelentkezik, de csak azért, mert a main függvényünk elég egyszerű. A való életben egy kontrollernek rengeteg feladata lehet, amit minden ciklusban el kell végeznie, és ez bizony időbe telik. Mi most nem fogunk ilyen összetett műveleteket elvégezni, de valahogy mégis jó lenne látni, hogy ennek milyen hatása lenne a kódunkra. Írjunk be a main függvénybe egy for ciklust, ami igazándiból nem csinál semmit, csak elszámol X-ig! Igaz ugyan, hogy ránézésre ennek nincs sok értelme, viszont egy biztos: jó sokáig fut, és ez nekünk most pont elég.

```
//Vegyünk fel egy kellően nagy méretű változót a main-en kívül
uint32_t i;

//Időigényes feladat imitálás valahol a while(1)-en belül
for (i=0; i<500000; i++)
{
}
```

Az új kód futtatása közben, ha nem a jó pillanatban nyomjuk meg a gombot, vagy nem tartjuk megfelelő ideig nyomva, akkor nem történik semmi. Viszont ha épp történik valami, akkor úgy tűnik, jól működik a dolog: szépen egyesével számol a megfelelő irányba.

Gondoljuk át, mi történik! A programunk az idő jelentős részében a for ciklusban számol, így csak ritkán tudja megnézni, hogy épp lenyomtuk-e a gombot. Így ha nem tartjuk nyomva elég sokáig, akkor egyszerűen lemaradhatunk az eseményről, ami semmiképpen sem nevezhető megbízható működésnek.

Ki kellene tehát találnunk egy olyan megoldást, ami csak a felfutó élt veszi észre, azt viszont bármikor, bárhol is jár épp a végrehajtás a programunkban. Szerencsére erre már mások is gondoltak előttünk, és meg is alkották a megoldást, ami nem más, mint...

AZ INTERRUPT

Az interruptot magyarul megszakításnak hívjuk, ami egy elég beszédes név.

Kinevezhetünk például dedikált bemeneteket, amiken ha impulzus érkezik vagy logikai szintváltás történik, akkor a mikrokontroller megszakítja a program futását, és meghív egy speciális, általunk írt függvényt, amiben reagálhatunk az eseményre, majd ezután pontosan onnan folytatja a program futtatását, ahol az interrupt előtt abbahagyta.

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

Ezt a funkciót external interruptnak hívjuk, a névvel jelezve azt, hogy ilyenkor a mikrovezérlő egy digitális bemenetén történt, külső hatásra kerül interrupt állapotba. A meghívott függvényt interrupt handlernek nevezzük. A handlert magyarul kezelőnek mondhatjuk, ez tehát a megszakítást lekezelő függvényünk.

Most már látjuk azt, hogy ez a lehetőség megoldást fog nyújtani a fenti nyomógombos problémánkra, de egyből sok újabb kérdésünk is felvetődhet:

- Milyen egyéb típusú interruptok léteznek még?
- Hogyan veszi észre a bemenet változását a mikrokontroller, miközben épp a főprogramban dolgozik?
- Mi történik ha a megszakításkezelő rutin futása közben valamelyik interrupt bemeneten egy, vagy több újabb interrupt-esemény jelentkezik?
    - Észrevesszük ezt?
    - Megszakítja az éppen futó interrupt-kezelő rutint is?

A kérdésekre úgy kaphatunk választ, ha legalább részben megértjük a mikrokontroller ezen részének működését.

A mikrokontroller több perifériájához is (például: digitális bemenet, timer, ADC, kommunikációs perifériák) beállíthatunk interruptokat. Az egyes perifériák hardverek, a hozzájuk tartozó konfigurációs regiszterekben beállított értékek szerint működnek, és működésük során a kimeneti regiszterekben állítanak át értékeket. A legegyszerűbb esetben a mikrokontroller adott lába azért működik digitális bemenetként, mert a hozzá tartozó konfigurációs biteket ezen cél érdekében állítottuk be. Amikor egy digitális bemenetet interrupt-bemenetként konfigurálunk, egyben azt is elérjük, hogy interrupt eseménykor az adott lábhoz tartozó interrupt bitet (egy hardveres regiszter egy bitje) 1-be állítsa. /A szakkifejezés erre a következő: 1-be állította az “interrupt flag”-et./ A lényeg, hogy ez hardveresen meg tud történni, a program futásával egy időben.

Az ATmega adatlapja tartalmaz egy “interrupt vector table” nevű táblázatot, ami összefoglalja a lehetséges interrupt-bemeneteket:

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

Table 11-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

***4. ábra** - adatlap részlet - Interrupt vektorok*

Az adatlapot tovább olvasva kiderül az is, hogy amennyiben egy időpontban több interrupt flag is 1-be van állítva, a kisebb interrupt vector numberrel rendelkezőhöz tartozó handler fog előbb lefutni. Például az INT1 és INT0 lábak esetén az INT0 élvez elsőbbséget.

Az ATmega16 mikrokontroller 1-szintű megszakításkezelővel rendelkezik, ami azt jelenti, hogy egy interrupt kezelő futását nem tudja megszakítani egy másik interrupt. Természetesen az egyes interrupt flagek bebillenhetnek, de csak az éppen futó megszakításkezelő rutin lefutása után kezeljük az esetlegesen közben beérkezett interruptokat.

Kitekintés

Kicsit összetettebb mikrokontrollereknél mi magunk állíthatjuk a különböző megszakítások prioritását, sőt prioritási szinteket állíthatunk be (többszintű megszakításkezelő). Egy ilyen rendszerben egy interruptot csak egy nála nagyobb prioritású szinten levő, tehát fontosabb interrupt szakíthat meg, mindamellett két egyszerre beérkezett megszakítás kérés közül a nagyobb prioritással rendelkező fog először lefutni. Tehát ha az interruptot meg lehet szakítani, akkor a rendszer az eredeti interrupt futását felfüggeszti, lefuttatja az új interrupt eseménykezelőjét, utána visszatér az eredeti interrupthoz, majd ha azzal is végzett, akkor tér vissza a fő programba.

Az Atmelhez hasonlóan egyszerű példánál maradva, a PIC18F család 2 szintű megszakítás rendszerrel rendelkezik. Itt, az adott perifériák megszakítás felkonfigurálásánál beállítható alacsony és magas

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

prioritási szint, így a futás közben egy magas prioritási szintű interrupt megszakíthat egy alacsony szintűt, de az azonos prioritásúak egymást nem szakítják meg.

Most már látjuk, hogy ha egy megszakítás kezelő futása közben újabb különböző típusú interruptok érkeznek, a hozzájuk tartozó flagek be fognak billenni, azonban egy adott típusú interrupt (például INT1 bemenet) előfordulásainak számát nem számolja hardver.

Az interrupt tehát egy nagyon hatékony eszköz, de tudni kell bánni vele. Legyünk óvatosak, az interrupt handler név kicsit félrevezető is lehet! Bár lehetne, de nem célszerű az esemény lekezeléséhez tartozó összes kódrészletet a handlerben megvalósítanunk. Mindig csak a legszükségesebb és legsürgősebb feladatokat végezzük el itt (például megjegyzünk egy, az interrupt forrására jellemző adatot: gomb lenyomva, hőmérséklet a kritikus 100°C fölé emelkedett, stb.), minden mást csak a fő programban, amikor valóban szükségünk lesz az eredményre (például bekapcsoljuk a ventillátort). Így minimalizálhatjuk annak az esélyét, hogy elmulasszuk egyes interrupt bemenetek jelzéseinek kezelését.

AZ INTERRUPT BEÁLLÍTÁSA

Ahhoz, hogy használni tudjuk ezt a funkciót a mikrokontrollerünknél, engedélyeznünk kell az adott periféria interrupt kérését és a mikrokontroller globális interruptját. Ezeket, a periféria vezérlő regiszter megfelelő bitjének (például `sbi()` makróval) és a globális interrupt engedélyező bit 1-be állításával (például `sei()` makróval) lehet kivitelezni.

Természetesen előfordulhat az is, hogy nem szeretnénk, hogy egy-egy főprogrambeli szakasz (például az adott rész idő és/vagy biztonságkritikus) megszakítható legyen az egyik, vagy akár az összes interrupt periféria kérése által. Ezt a megfelelő bitek 0-ba állításával tehetjük meg (`cbi()` és/vagy `cli()` makrókkal).

Így a közben beérkező interruptok csak az újraengedélyezés után kezelődnek le.

Kitekintés

Interrupt rutin futtatásakor a kontroller automatikusan módosít egyes biteket. Belépéskor globálisan tiltja a megszakításokat, törli (csak) az adott interrupthoz tartozó flaget, a handler végén pedig újra -a fordító által automatikusan beillesztett RETI utasítás hatására- globálisan engedélyezi a megszakításokat. /Kicsit talán helytelenül, de a szaknyelv a következőképp használja ezeket a kifejezéseket: globális megszakítások titása (`cli()`), és engedélyezése (`sei()`)/

A SZÁMLÁLÓS FELADATUNK MEGOLDÁSA

Az interruptok részletes megismerkedését követően térjünk vissza a számlálós feladatunkhoz.

Most már a nyomógombokat megszakításosan fogjuk kezelni, és az interrupt handlerben növelni vagy csökkenteni szeretnénk a számlálónk értékét. Az előzőekben használt ki-/bemenetek felkonfigurálását végző `IOInit()` függvényt ki kell egészítenünk a megszakítások beállításával és engedélyezésével, amit (a függvény végére elhelyezve) a következő programsorok valósítanak meg:

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

```
	//INTR0 felfutó élre, vagyis ISC00=ISC01=1 66.oldal
	sbi(MCUCR, ISC00);
	sbi(MCUCR, ISC01);
	//INTR1 felfutó élre, vagyis ISC10=ISC11=1 67.oldal
	sbi(MCUCR, ISC10);
	sbi(MCUCR, ISC11);
	//External interrupt engedélyezés 68.oldal
	sbi(GICR, INTF0);
	sbi(GICR, INTF1);
	//Global interrupt engedélyezés     9.oldal
	sbi(SREG, 7);
```

A demóprojekt main.c fájl tartalmát pedig a következőre kell módosítanunk:

```
#include "../Headers/main.h"
volatile uint8_t cntr = 0;

// + gomb megszakítása
ISR(INT0_vect)
{
	//Ha még nem érte el a maximumot a számláló, akkor növelem
	if (cntr < 7)
	{
		cntr++;
	}
}

// - gomb megszakítása
ISR(INT1_vect)
{
	//Ha még nem érte el a minimumot a számláló, akkor csökkentem
	if (cntr > 0)
	{
		cntr--;
	}
}

int main(void)
{
	//IO inicializálása
	IOInit();

	//Végtelen ciklus
	while (1)
	{
		//LED-ek kigyújtása a számlálónak megfelelően
		PORTA = cntr;
	}
	return (0);
}
```

Ez a példány Rába Ármin kizárólagos használatú példánya.

http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

Nézzük részletesen, hogyan működik: az `IOInit()` hívás hatására, a digitális lábak irányának és értékének beállítását követően, az `sbi(reg, n)` makrók az egyes mikovezérlő regiszterek bitjeit logikai 1 értékűre állítják. Az első 4 a külső megszakítások felkonfigurálását, a következő kettő a külső megszakítások engedélyezését, míg az utolsó a globális interrupt engedélyezését hajtja végre. A globális megszakítások engedélyezése és tiltása a `sei()` és `cli()` - paraméter nélküli - makrókkal is megoldható, melyek használata mellesleg jobban olvashatóvá teszi a programunkat. A main.c-ben pedig "új típusú függvények" találhatóak, melyek ISR (Interrupt Service Routine = Interrupt Handler) nevűek és `xxx_vect` paraméterrel rendelkeznek. Ezek a már fentebb megismert interrupt handler függvények, amikre csak megszakítás hatására kerül a vezérlés, a paraméterük pedig az adott megszakításra jellemző névvel kezdődik (melyek az adatlap 44. oldalán -Interrupt Vectors cím alatt- megtalálhatóak) és `_vect` toldalékkal kell végződjön. Jelen esetben ezekben a függvényekben a számlálónk értékét növeljük vagy csökkentjük, figyelembe véve a számláló megengedhető tartományát.

A main függvény végtelen ciklusában pedig csak az A port értékét beállítjuk a számláló értékére.

Mivel a nyomógomb pergésmentesítését a (szűrő)kondenzátor valósítja meg, és a külső megszakítást felfutó élre konfiguráltuk, így lenyomva valamelyik gombot még nem történik semmi, de annak elengedésekor egy 0-1 átmenet történik a mikrovezérlő lábán, ami megszakítást generál, majd a megfelelő handlerre kerül a vezérlés.

Kitekintés

Az ISR függvények, nem a megszokott működésű függvények, mivel az "ISR" a programkódban csak egy makró, amivel tudatjuk a fordítóprogrammal, hogy a függvény tartalma az interrupt handler lesz.

A fordító ezen programsoroknak megfelelő assembly utasítások végére a RETI utasítást teszi "automatikusan", ami a -hardverből- letiltott globális megszakításokat újra engedélyezi.

A fentieken kívül azt is érdemes megjegyezni, hogy az interrupt handlereket nem is tudjuk a programunkból meghívni úgy, mint az egyéb, általunk definiált függvényeket.

TIMEREK ÉS INTERRUPT

Ha eddig esetleg nem lett volna egyértelmű, akkor szeretnénk még egyszer kihangsúlyozni: az interrupt egy nagyon fontos eleme a beágyazott szoftverfejlesztésnek. Ez az oka annak, hogy a mikrokontrollerben a legtöbb periféria képes megszakítást generálni: nem csak egy-egy mikrovezérlő lábon érkezhet interrupt, de jelezheti például az analóg-digitális átalakító, hogy elkészült egy méréssel (erről bővebben a következő fejezetben), vagy ha valamilyen kommunikációs csatornán üzenet érkezik, akkor az adott modul ezt interruptban hozhatja a tudomásunkra.

A bekezdés címéből már talán sejthető, hogy általában arról is interruptban kaphatunk információt, ha az egyik timerünk "lejárt".

Egy korábbi fejezetben már volt szó a timerekről, de akkor még nem használtunk megszakítást. Nézzük most meg egy példán keresztül, hogyan is tudunk egy timert interruptos módban működtetni!

Ez a példány Rába Ármin kizárólagos használatú példánya.

http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

Beszéltünk már a pergésmentesítésről, amit egy-egy kondenzátorral, hardveresen oldottunk meg. A mostani példához távolítsuk el a kapcsolásból ezeket a kondenzátorokat, és oldjuk meg a nyomógombok pergésmentesítését szoftveresen!

Miután kivettük a kondenzátort, próbáljuk meg lenyomni, majd lassan felengedni a gombot, ekkor az esetek egy részében a számláló egyszerre többet is fog lépni az adott irányba. Ez a jelenség a -már emlegetett- nyomógombok pergése, amit a jelenlegi szoftverünk rendkívül jól láttat.

Ennek kiküszöböléséhez a következőt tesszük: a nyomógomb bemenet interruptját kezelő függvényében letiltjuk a további -azonos forrásból származó- külső megszakításokat, és elindítunk egy timert, amit előzőleg kb 100 ms-os periódusidőre állítunk. A timer lejártakor meghívott interrupt kezelő függvényben pedig újra engedélyezzük a gomb felől érkező megszakításokat.

Ezzel azt értük el, hogy a rendszer gombnyomáskor csak az első felfutó élet veszi észre, mert a pergés miatti többi felfulfáskor már nem aktív az interrupt. A beállított közel 100 ms elég nagy ahhoz, hogy a prellezés ez alatt befejeződjön, de elég kicsi ahhoz, hogy két egymás utáni, szándékos gombnyomást szinte biztosan észrevegyünk.

Hozzunk létre egy timer.c fájlt, amiben -a korábbiaknak megfelelően- konfiguráljuk fel a TIMER1 perifériát 8-as előosztással, így a `TimerInit()` függvény tartalma a következő:

```
//8x előosztás, 108. oldal
cbi(TCCR1B, CS12);
sbi(TCCR1B, CS11);
cbi(TCCR1B, CS10);
```

Mivel a kontrollerünk órajele 8 MHz, a timer órajele ennek 8-ad része. A timer 16 bites, így (8000000 / 8) / 65536 = ~15,26Hz frekvenciával, azaz 1/15,26Hz = 65,536ms-onként fog a számláló túlcsordulni. Ezt az időt fogjuk felhasználni a nyomógombok pergésének kivédésére.

Az io.c fájl tartalmát nem kell módosítani, hisz induláskor azt akarjuk, hogy a gombok felől érkező megszakítások aktívak legyenek.

A main.c fájlban mindkét external interrupt handlert ki kell egészítenünk a timer nullázásával és az overflow interrupt engedélyezésével amit a következő sorok hajtanak végre:

```
//Timer1 számláló nullázása
TCNT1 = 0;
//Timer1 overflow  interrupt engedélyezés
sbi(TIMSK, TOIE1);
```

Illetve le kell tiltani az adott external interruptot annak érdekében, hogy a gomb pergéséből adódó újabb élváltásokat ne vegye észre a kontroller:

```
//INTR0 interrupt tiltása
cbi(GICR, INTF0);
```

Ez a példány Rába Ármin kizárólagos használatú példánya.

http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

Ezen kívül létre kell hozni egy új interrupt handlert a timer túlcsordulására, amiben újra engedélyezzük az external interruptokat és töröljük a timer lejártáig esetlegesen újra bebillent external interrupt flageket, majd letiltjuk a számláló megszakítását:

```
//Timer1 túlcsordulás megszakítás
ISR(TIMER1_OVF_vect)
{
	//External interruptok engedélyezése, 67.oldal
	sbi(GICR, INTF0);
	sbi(GICR, INTF1);

	//External interrupt flagek törlése, 68.oldal
	cbi(GIFR, INTF0);
	cbi(GIFR, INTF1);

	//Timer1 overflow interrupt tiltása
	cbi(TIMSK, TOIE1);
}
```

A main függvényben így már nincs is más dolgunk, mint a `TimerInit()` és `IOInit()` függvények hívása után a végtelen ciklusban az A portra kitenni a számláló értékét.

Próbáljátok ki, hogy mi történik, ha bemásoljátok a main függvénybe az időigényes ciklusunkat.

```
//Időigényes feladat imitálás valahol a while(1)-en belül
for (i=0; i<500000; i++)
{
}
```

Ha nem is azonnal, de minden gombnyomásnak hatása van a 3 bites LED-es számlálónkra. A processzor idejének nagy részét azzal tölti, hogy a for ciklust hajtsa végre, majd frissíti a kimenetet a számláló értékével, és újra a for ciklusban fog pörögni. Azonban a megszakításaink minden egyes gombnyomásra lefutnak, így a "háttérben" a számláló értéke frissül/változik.

Kitekintés

A fenti esetben nem kritikus, hogy ~65 vagy pontosan 100 ms-ot várunk, így az egyszerűbbnek mondható számláló üzemmódot használtuk.

Ahhoz, hogy konkrétan 100 ms-ot várjunk, a timert Output Compare üzemmódban kellene használni. Ilyenkor a timer egy előre beállított érték (OCR1AH/OCR1AL és/vagy OCR1BH/OCR1BL 8 bites regiszterek) elérésekor tud megszakítást generálni.

Ezen felül egyes nyomógomboktól, illetve "gomb lenyomási jellegtől" (mennyire óvatosan/lassan van lenyomva és felengedve) függően, meglehet, hogy ez a ~65ms is kevésnek bizonyul, következésképpen több időt kell hagynunk a gombok pergésének lecsengésére.

Ennek megoldására használhatjuk a fentebb ismertetett módban a timert, vagy választhatunk nagyobb előosztást is hozzá. Ugyanakkor ezekkel a megoldásokkal, egyre több ideig hagyjuk letiltva az external

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.

interruptokat, így egyre kisebb gyakorisággal tudjuk léptetni a számlálónkat. Az optimális megoldás mindig az adott konfigurációtól függ, tekintettel arra, hogy a gombunk mennyire pereg, van-e szabad timerünk, milyen sűrű impulzusokat kell érzékelni, stb.

VÉGSZÓ

Ezzel elértük a fejezet elején kitűzött célokat, megismertük a megszakítás fogalmát és annak meglehetősen fontos szerepét. 1-1 példát néztünk external és internal interrupt használatára, amiket a könyv további részében, de azon felül, remélem, hogy hobbiprojektekben és a későbbi tanulmányok során is kamatoztatni tudja a kedves olvasó.

Ez a példány Rába Ármin kizárólagos használatú példánya.
http://kristalytisztaelektronika.hu | Minden jog fenntartva Xtalin Mérnöki Tervező Kft.