17. PWM mikrovezérlővel

Írta: Szesztay Endre

Lektorálta: Veréb Szabolcs

BEVEZETÉS

Az előző részben már megismerkedtünk a PWM lényegével. Láttuk, hogy gyakorlatilag analóg (folyamatos) feszültség vagy teljesítmény szabályozást tudtunk megvalósítani a feszültség (vagy áram) gyors kapcsolgatásával, vagyis ténylegesen analóg alkatrész felhasználása nélkül. A PWM szabályzásnak ez a tulajdonsága két esetben is nagyon hasznos:

- egyrészt ha tisztán digitális rendszerekben akarunk analóg jelet előállítani, (a tisztán digitális rendszer azt jelenti, hogy csak logikai áramköreink és jeleink vannak, amik csak bekapcsolt vagy kikapcsolt értékűek lehetnek, pl. 5V és 0V)
- másrészt ha nagyobb teljesítményt akarunk szabályozni, pl. motorvezérlésnél.

A fentiek ismeretében nem meglepő, hogy a legtöbb mikrovezérlő tartalmaz PWM vezérlő egységet. Ez egy olyan periféria (hardver elem), ami vagy egy számláló - időzítő (angolul timer / counter), kiegészítve PWM funkciókkal, vagy kifejezetten PWM jelek előállítására van kialakítva. Kicsit leegyszerűsítve ez úgy néz ki, hogy beállítjuk a frekvenciát és a kitöltési tényezőt, és a mikrovezérlő egyik lábán megjelenik a PWM jel. Ezzel akár közvetlenül meghajthatunk egy LED-et, vagy kapcsoló elemet (FET).

Fontos megjegyezni, hogy a PWM jel előállítása ezek után már a processzor (CPU) közvetlen beavatkozása nélkül történik, tehát nem veszi el az időt a többi feladattól. Erről még lesz szó a későbbiekben.

MIT TUD EGY PWM PERIFÉRIA?

Ahhoz, hogy a PWM vezérlőt használni tudjuk, ismerjük meg kicsit mélyebben a működését. Egy PWM vezérlő általában a következő funkciókat tudja, bár ezek eltérhetnek a különböző mikrovezérlőkben:

- PWM frekvencia (periódusidő) és kitöltési tényező beállítása - ezt mindenképpen tudnia kell
- megszakítás generálása minden egyes ciklus végén - ezt is mindig szokták tudni. (A megszakításról később beszélünk részletesen. Röviden arról van szó, hogy a processzor bizonyos külső jelek hatására meg tudja szakítani az éppen futó programot, és egy másik program-részletet kezd futtatni, ami után visszatér a félbehagyott programhoz.)
- további megszakítások generálása a periféria különböző eseményeinél, például a kimeneti PWM jel változásakor (0->1, vagy 1->0 átmenet)
- több PWM kimenet, amelyeknek különböző lehet a kitöltési tényezője

- kimenetek (lábak) beállítása: kiválaszthatjuk, hogy az előállított jel a mikrovezérlő melyik lábán legyen hozzáférhető, illetve hogy a belső számláló lejártakor a kimeneten 0->1 vagy 1->0 átmenet történjen-e (polaritás).
- ha több kimenet van, ezek egymáshoz képesti időzítése (egyszerre kapcsoljanak be vagy ki, vagy éppen ellentétesen működjenek)
- lehet több, teljesen független PWM vezérlő, amelyek egymással szinkronizálhatóak is
- DMA vezérlés - ez egy olyan hardver elem, ami a CPU használata nélkül teszi lehetővé adatok mozgatását a memória és a periféria (itt a PWM egység) között.
- egyéb input jelek, például egy külső hibajel le tudja tiltani a PWM működését.

Nézzünk néhány példát, hogy milyen feladatokat lehet egy mikrovezérlős PWM egységgel megvalósítani:

- állandó négyszögjel, frekvencia előállítás. Ilyenkor igazából a jel kitöltési tényezőjét nem változtatjuk, csak a frekvenciáját. Ez akkor lehet hasznos, ha például változó magasságú hangot akarunk előállítani.
- fix frekvenciájú PWM jel, aminek a kitöltési tényezőjét a programból változtatjuk. Ezzel folyamatos módon vezérelhetünk például motorokat, vagy egy LED fényerejét, de szűrés után előállíthatunk vele analóg feszültséget is.
- ciklusról ciklusra változó PWM jel. Ez már kicsit bonyolultabb, de lehetőség van arra, hogy minden egyes PWM ciklusban más legyen a kitöltési tényező. Ehhez a PWM egység által generált jel (megszakítás) segítségével szinkronizálni kell a CPU programját; így valósíthatóak meg gyors szabályzások, leginkább motorvezérléshez használják.

Természetesen még számtalan más módon is használható a PWM egység, ez csak a programozó fantáziáján múlik.

Kitekintés

A fenti két utolsó bekezdés látszólag ugyanazt mondja, de míg az egyikben a vezérlés szót használom, a másikban a szabályzást. A különbség az, hogy - ha precízek akarunk lenni - a vezérlés szót arra az esetre használjuk, amikor például úgy változtatjuk egy motor fordulatszámát, hogy beállítjuk a motorra jutó feszültséget egy előre kiszámított értékre. A fordulatszám így nyilván nem lesz pontos, de közelítőleg az fog történni, amit akarunk; a motor lassabban vagy gyorsabban forog.

A szabályzás ehhez képest egy összetett folyamat; a fenti példánál maradva, a fordulaszámot folyamatosan mérjük, és ennek alapján megfelelően módosítjuk a motorra jutó feszültséget. (Ha túl gyorsan forog a motor, akkor a feszültsége lassan csökkentjük, és fordítva.) Ezt az ismételt mérés-számítás-beavatkozás folyamatot hívjuk szabályozásnak. A szakirodalomban találkozhatunk még a “visszacsatolás” kifejezéssel is; ez tulajdonképpen ugyanezt a folyamatot jelenti, bár a hangsúly ilyenkor a jel érzékelésén, a mérésen van. Ennek az érzékelésnek az eredményét, vagyis a mért jelet visszacsatoljuk (bevezetjük) a szabályzóba a szabályzási cél megvalósítása érdekében.

A szabályozás fenti módja egyébként sokkal bonyolultabb mint az első ránézésre látszik. Ennek az az oka, hogy a szabályzás könnyen be tud gerjedni, ami azt jelenti, hogy nem áll be egy állandósult állapotba, hanem folyamatosan ingadozó kimenetet állít elő. Akit érdekel a téma, a “PI szabályozó”, vagy “PID szabályozó” szavakra érdemes keresni az interneten.

Eddig már szó volt arról, hogy a PWM vezérlő a mikrovezérlő egy perifériája. De mi is az a periféria?

Egy PWM jel előállítását például szoftverben is meg lehet valósítani; számláló ciklusokat kell létrehozni, és megfelelő időpontokban át kell váltani egy kimenet (láb) állapotát. Ez egy nagyon egyszerű feladat, viszont rengeteg idejét elvenné a CPU-nak. Ezért az ilyen feladatokra külön, specializált hardver elemeket építenek be a mikrovezérlőbe, ezeket nevezzük perifériáknak.

A perifériák önállóan működnek, nem terhelik a CPU-t. Saját belső logikával működnek; megkapják az órajelet, direkt összeköttetésben vannak a mikrovezérlő IC lábaival, a programnak csak a felprogramozást és a szükséges adatok ki-be olvasását kell elvégeznie.

A leggyakoribb ilyen perifériák:

- portok: ezek a perifériák a mikrovezérlő IC lábait teszik elérhetővé a program számára; beolvashatjuk bemenetek állapotát, illetve beállíthatjuk a kimenetek állapotát, egyszerű regisztereken keresztül.
- számlálók - időzítők: regisztereken keresztül beállítjuk a számlálás sebességét, és bármikor kiolvashatjuk a programból, hogy hol tart a számláló. Ez kiválóan alkalmas időmérésre.
- PWM: erről lesz szó részletesebben
- kommunikációs egységek, például: RS232, USB
- analóg - digitális (ADC) és digitális - analóg (DAC) átalakítók. Ezek segítségével a mikrovezérlő lábain nem csak digitális jelek, tehát például 0V vagy 5V, hanem a teljes tápfeszültség-tartományban bármilyen feszültség lehet, és ezt megmérhetjük, vagy éppen programból beállíthatjuk.
- megszakítás vezérlő. Erről a tananyagban később lesz szó.

A következő fontos kérdés, hogy hogyan tud a CPU-n futó program kommunikálni a perifériákkal, a mi esetünkben például a PWM vezérlővel. Erre a célra a perifériáknak úgynevezett regisztereik vannak, ezekbe megfelelő kódokat (számokat) írva tudjuk vezérelni a perifériánkat. A regiszterek a program szempontjából egyszerű memóriák, tehát a CPU olvashatja és általában írhatja is a bennük lévő értéket. Ennél azonban többet tudnak, mert a bennük lévő adat (az egyes bitek értéke) közvetlenül befolyásolja a hardver (a perifériákban lévő logikai áramkörök) működését.

Ne keverjük össze a perifériák regisztereit a CPU saját regisztereivel; az utóbbiak a CPU belső logikájában vannak, és közvetlenül a gépi kódú utasítások tudnak rájuk hivatkozni, ezzel most nem foglalkozunk.

A perifériák regiszterei egyszerű memória írás-olvasással érhetőek el, ugyan úgy, mint a memória, csak más címeken. A mikrovezérlő adatlapja részletesen leírja a perifériák működését, a regiszterek címeit, amin elérhetőek, és az egyes bitek jelentését (hatását) a regisztereken belül.

A mi Atmel processzorunk 8 bites, ezért a perifériák regiszterei is 8 bitesek. Vannak olyan regiszterek, ahol minden bitnek külön jelentése van, és van, ahol az egész regiszterbe egyetlen 8-bites számot kell beírni. Vannak olyan perifériák is, ahol a belső működés 16-bites, például egyes időzítőknél. Ebben az esetben az alacsony (Lo) és a magas (Hi) helyiértékű bájtokat külön kell beírni. A regiszterek kezelésére és a PWM periféria esetében a konkrét beállításokra még visszatérünk részletesen.

Kitekintés

A PWM egység képességeinél említettem, hogy megszakítást tud létrehozni. Erről későbbi részben lesz még szó részletesebben, itt csak néhány mondatban leírom a lényeget. A megszakítási mechanizmus általánosan azt jelenti, hogy a CPU egy külső jel hatására megszakítja a program futását, és egy másik programrészt (megszakítás kiszolgáló kódot, subrutint) kezd futtatni. Ez a megszakítás, amit arra használnak, hogy a CPU szinte azonnal értesüljön, így a program reagálni tudjon egy külső eseményre. Fontos, hogy a megszakítás (angolul interrupt) kiszolgálásának (a hozzá tartozó feladat elvégzésének) befejezése után a CPU képes visszaállni a félbehagyott program futtatására úgy, hogy a CPU regisztereiben az értékek változatlanok, tehát a megszakított program semmit nem vesz észre a megszakítás kiszolgálásából. A megszakítás kezelésre a CPU-kban és mikrovezérlőkben speciális utasítások és mechanizmusok vannak; egyrészt megszakításkor a CPU-nak el kell mentenie a belső regiszterek állapotát, másrészt a megszakítást kiszolgáló programrésznek tudnia kell azonosítani a megszakítás forrását.

Szinte minden periféria tud valamilyen megszakítást generálni, amin keresztül a CPU gyorsan reagálhat a külvilág eseményeire; ez lehet egy külső jel egy lábon, egy adat érkezése valamilyen kommunikációs csatornán, vagy egy időzítő lejárta. Külön periféria a megszakítás vezérlő is, amelyen beállítható, hogy mely egységek megszakítás kérelmét hajtsa végre a CPU, és milyen fontossági sorrendben.

A PWM PERIFÉRIA MŰKÖDÉSE

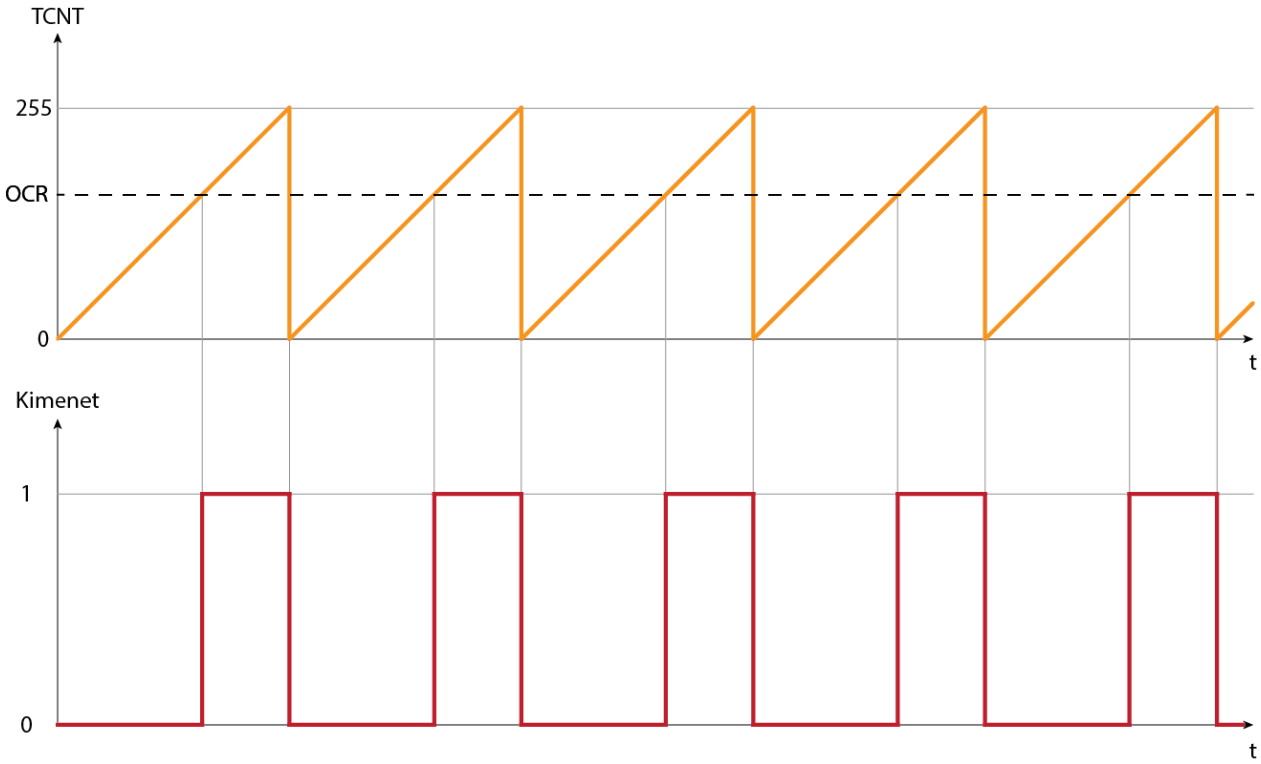
Az általunk használt Atmega16A mikrovezérlőben a időzítő-számláló egységek tudnak PWM jelet is előállítani.

SZÁMLÁLÓK (ISMÉTLÉS)

A számláló periféria lényege egy számláló regiszter. Ez egy olyan 8 vagy 16 bites tároló, ami egy külső jel hatására képes eggyel növelni vagy csökkenteni értékét. Például ha a számláló (regiszter) értéke 12, akkor az órajel hatására 13, 14… stb. értékeket fogja felvenni.

Ha a számláló 8 bites, akkor általában 255 után 0-ra változik az értéke, és a számlálás innen folytatódik tovább. A számláló regiszter értékét a CPU bármikor átírhatja és kiolvashatja, de normál PWM előállítás közben erre nincs szükség.

Ezzel a regiszterrel időzítési, időmérési és számlálási feladatokat is meg lehet valósítani; időzítést / mérést akkor, ha állandó órajelről hajtjuk a számlálót, míg ha egy kívülről jövő impulzust kötünk az órajel helyére, akkor ezeket számolja meg a számláló. Ennek megfelelően a neve angolul Timer/Counter, rövidítve TCNTn. A kis “n” betü helyén az egység száma van, tehát például a TCNT2 a második számláló periféria TCNT regiszterét jelenti.



***1.ábra** - PWM periféria működése*

A fenti ábra a PWM periféria egy tipikus működési módját ábrázolja. A miáltalunk használt ATMega16A adatlapjában ezt “Fast PWM Mode”-nak, azaz gyors PWM működésnek hívják. Ebben az üzemmódban a számláló mindig felfelé számol. A fenti ábrán felső vonal ábrázolja a TCNT regiszter értékét, látható, hogy ebben az esetben 255-ig számol, utána nullázódik, és 0-tól folytatja a számolást.

A PWM másik működési módja (a mi mikrovezérlőnknél) az úgynevezett “Phase Correct PWM Mode”, ezt magyarra talán fázistartónak lehetne fordítani. Ebben az esetben a számláló 255-nél nem nullázódik, hanem lefelé kezd számlálni, nullától meg felfelé. Ennek akkor van jelentősége, ha több PWM periféria működését kell összehangolni; ezzel az üzemmóddal itt nem foglalkozunk.

Ahhoz, hogy a fenti számláló perifériánk PWM jelet is elő tudjon állítani, szükségünk van egy nem számláló regiszterre, aminek az értékét minden pillanatban összehasonlítjuk a számláló regiszter értékével. Ezt a regisztert angolul “Output Compare Register”-nek hívják, rövidítve OCRn. A fenti ábrán ennek értékét a felső görbén a szaggatott vízszintes vonal jelzi. Ameddig a számláló értéke kisebb, mint az OCR regiszteré, addig az egység kimenete ‘0’, amikor pedig nagyobb, akkor ‘1’ (ez beállítás kérdése, működhet fordítva is).

Látható, hogy a komparálási szint, azaz az OCR regiszter értékének változtatásával a kimenet bekapcsolt és kikapcsolt időtartamának aránya változik, tehát megkaptuk a kívánt PWM jelet.

A PWM PERIFÉRIA HASZNÁLATA

LED FÉNYERŐ VÁLTOZTATÁSA

Miután nagy vonalakban megismertük a PWM jel előállításának módját, nézzük meg részletesen, hogy az ATMega16A mikrovezérlőn miket kell ehhez beállítani.

Az alábbi egyszerű program egy LED fényerejét változtatja folyamatosan, minimálisról maximálisra és vissza. Ehhez egy kb 31 kHz-es PWM jelet állít elő, aminek a kitöltési tényezője fogja meghatározni a LED fényerejét.

A program megtalálható a tananyag honlapján.

Az program működéséhez szükséges hardver-összeállítás meglehetősen egyszerű. Mindössze egyetlen LED-et kell, egy 330Ω-os ellenálláson keresztül, az ATMega16A megfelelő lába és a GND közé kötni. Ez a láb a chip 21. lába, a neve "PD7 (OC2)", részletes leírást az mikrovezérlő adatlapjában találsz. A neve arra utal, hogy ennek a lábnak két funkciója van: alapvetően a D port 7. bitje, de itt lehet kivezetni a 2-es időzítő egység OC jelét is. Az OC az "Output Compare" rövidítése, azaz itt jön ki a számláló (`TCNT`) és az `OCR2` regiszter összehasonlításának az eredménye.

Ahhoz, hogy a programunk helyesen működjön, a következő fő lépéseket kell megtennünk:

- az IO portok megfelelő beállítása, hogy az előállított PWM jel kijusson a mikrovezérlő lábán. Ezt az `IOInit()` függvény végzi
- megfelelően be kell állítani az időzítő-számláló periféria regisztereit, hogy a kívánt PWM üzemmódban működjön, ezt csinálja a `PWMInit()` függvény
- végezetül pedig be kell állítani a komparálási értéket, vagyis a PWM kitöltési tényezőjét. Ezt a `main()` függvényben futó végtelen ciklus végzi; folyamatosan, lassan változtatja a komparálási szintet, ezáltal a LED fényereje is folyamatosan változik.

A forráskódban található kommentek magyarázzák az egyes részek feladatát, de mindjárt megnézzük részletesebben is. Ha saját programot akartok írni, akkor szükséges lesz a mikrovezérlő adatlapjában a Timer/Counter egységek áttanulmányozása.

A program egyetlen fájlból áll, ennek a neve `main.c`, ez tartalmazza az összes szükséges kódot. A program futása a `main()` függvényben indul, ez hívja meg először a `IOInit()` majd a `PWMInit()` függvényeket. Az `IOInit()` egyszerű regiszter írásokat végez, a kommentekből látszik a regiszterek jelentése. A `PWMInit()` függvény az időzítő (PWM) működését állítja be. A kódba ágyazott kommentek megmutatják hogy az egyes beállítások mire vonatkoznak, a részletesebb megértésért érdemes az adatlapot áttanulmányozni.

Kitekintés

A függvényben használt `sbi()` (értsd: set bit, bit 1-be állítás) és `cbi()` (clear bit, bit nullába állítás) parancsok helyett, melyek a compat/deprecated.h fájlban definiáltak, az egyes biteket a következő "vagy" logikai kapcsolaton alapuló megoldással is beállíthatjuk:

```
TCCR2 = TCCR2 | (0<<CS22) | (0<<CS21) | (1<<CS20);
```

Ez így talán nehezebben érthető, ezért használjuk a cbi(), sbi() makrókat (a makró alatt `#define` kifejezést értünk).

Érdekes, hogy a `sbi(TCCR2, WGM20);` minden eleme makró kifejezés. Az `sbi()` egy paraméterezett makró, feloldása: `#define sbi(port, bit) (port) |= (1 << (bit))`.

A `TCCR2` is egy makró: `#define TCCR2 _SFR_IO8(0x25)`. Ennek az az érdekessége, hogy ez az `_SFR_IO8()` makrót használja, ami egy I/O címből memória címet csinál. Ez azért érdekes, mert ennek értéket is lehet adni, tehát a `TCCR2 = 0;` például egy érvényes parancs.

Természetesen a `WGM20` is egy makró, egyszerű érték definíció: `#define WGM20 6`. Ez azt fejezi ki, ami az adatlap 125. oldalán látható, hogy a WGM20 bit a 6. bit az ezt tartalmazó byte-ban.

Végezetül, a `main()` függvény további részének már csak annyi a dolga, hogy az `OCR2` regisztert folyamatosan változtatja. Ez az a regiszter (Output Compare Register), aminek az értéke minden számláláskor összehasonlításra kerül a számláló értékével, így ez határozza meg a PWM jel kitöltési tényezőjét.

```
/*
 * 17. Fejezet
 *
 * A mikrovezérlő PD7 lábára kapcsolt LED fényerejét folyamatosan változtatja,
 * a PWM kitöltési tényezőjének lineáris változtatásával.
 */

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

int main(void)
{
	//Ki- és bemenetek inicializálása
	IOInit();
	//PWM inicializálása
	PWMInit();

	//Felfelé kezdek számolni
	bool cntr_up = true;

	//Végtelen ciklus
	while (1)
	{
		//PWM kitöltés növekedik
		if (cntr_up)
		{
			//Ha növekedhet még
			if (OCR2 < PWM_DUTY_MAX)
			{
				//Növelem
				OCR2++;
			}

			//Ha elérte a maximumot
```

```
                  else
                  {
                        //Irányváltás
                        cntr_up = false;
                  }
            }

            //PWM kitöltés csökken
            else
            {
                  //Ha csökkenhet még
                  if (OCR2 > PWM_DUTY_MIN)
                  {
                        //Csökkentem
                        OCR2--;
                  }

                  //Ha elérte a minimumot
                  else
                  {
                        //Irányváltás
                        cntr_up = true;
                  }
            }

            //Késleltetés, hogy látható legyen a PWM változása
            _delay_ms(PWM_DELAY_MS);
      }

      return 0;
}

void PWMInit()
{
/*
 * A PWM frekvencia (adatlap 120.oldal) fpwm=fclkIO/(N*510),
 * ahol N az órajel osztása.
 * Ebből N=fclkIO/fpwm*1/510, vagyis 10kHz körüli
 * frekvenciához N=8e6/10e3*1/510=1.569 osztás kell.
 * A mikrokontrollerben csak 1,8,32,64,128,256,1024 osztások vannak,
 * N=1-et választva
 * fpwm=15.686kHz adódik.
 */

      //Egyszeres órajel osztás a timer2 számára
      cbi(TCCR2, CS22);
      cbi(TCCR2, CS21);
      sbi(TCCR2, CS20);
      //Ponált PWM működés, OC2 kimenet 0, ha a számláló nagyobb a küszöbnél
      sbi(TCCR2, COM21);
      cbi(TCCR2, COM20);
      //Phase correct PWM mód
```

```
        cbi(TCCR2, WGM21);
        sbi(TCCR2, WGM20);

        /* A sbi() és cbi() függvények helyett,
         * melyek a compat/deprecated.h fájlban definiáltak,
         * az egyes biteket a következő, "vagy" logikai kapcsolaton
         * alapuló megoldással is beállíthatjuk:
         * TCCR2 = TCCR2 | (0<<CS22) | (0<<CS21) | (1<<CS20);
         * Ez elég átláthatatlan, így érdemes a fenti függvényeket alkalmazni.
         *
         * A C:\Program Files (x86)\Atmel\Atmel Toolchain\AVR8 GCC
         * \Native\3.4.1061\avr8-gnu-toolchain\avr\include\compat\deprecated.h
         * fájl tartalmazza a függvények definícióját, látható, hogy a sbi() és
         * cbi() tulajdonképpen ezt a "vagy" kapcsolatos megoldást
         * valósítják meg (az elérési út a verzióktól függően változhat).
         */
}
```

További lehetőségek

A fenti kódban a fényerő változtatása az `OCR2` (8 bites) regiszter értékének átírásával történik. Ha játszani akartok a kóddal, kipróbálni saját programokat, akkor ebbe a regiszterbe kell beleírni egy 0 és 255 közötti számot. A 0 értéknél a kitöltési tényező, és így a LED fényereje nulla lesz, míg 255-nél maximális fényerőt kaptok.

Kitekintés

Ha a 0..255 tartományon kívüli értéket írtok be, az elvileg illegális utasítás (a C nyelv szabályai szerint), viszont a kód lefordul és futni is fog, a regiszter értéke viszont a beírt szám (2-es számrendszerben kifejezett) alsó 8 bitje lesz. Tehát pl a 256 beírása valójában 0 beírásának felel meg, a 257 1-nek, és így tovább.

ZÜMMER - DALLAM LEJÁTSZÁSA

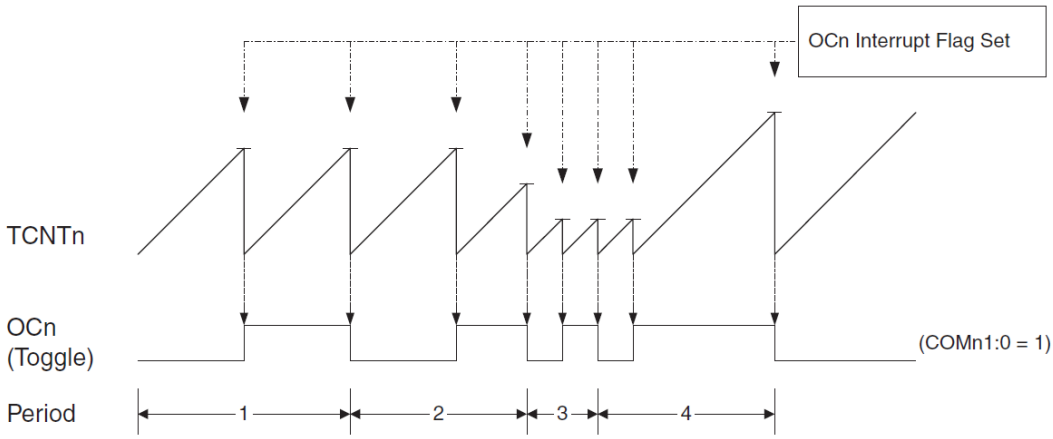
A tárhelyen elérhető egy másik példaszoftver is *KE17_2_PWM_zummer* néven, amely dallamokat képes lejátszani.

A hardver összeállítás itt sem bonyolultabb: egy hangszórót kell, szintén ellenálláson keresztül a mikrokontrollerre kötni, de most a chip 19. lábát használjuk, melynek neve PD5 (OC1A). Ennek az az oka, hogy ehhez a feladathoz az egyes számú időzítőt használjuk (16-bit Timer/Counter1). Ez, ahogy a nevéből is látszik, 16 bites, tehát sokkal finomabb felbontásban tudjuk vele a kimenet időzítését állítani.

Ennek a programnak már kicsit összetettebb a felépítése. Ez részben azért van, mert kicsit bonyolultabb a működése, részben pedig azért, mert a program egyes részei - funkciói - külön forrásfájlokban vannak. Ez, hogy az összetartozó funkciójú kódrészleteket egy külön fájlban tároljuk, általános gyakorlat. Az a célja, hogy egy fájl ne legyen áttekinthetetlenül hosszú. A kódot tehát a `main.c`, `io.c`, `pwm.c` és a `music.c` fájlok, és a hozzájuk tartozó `main.h`, `io.h`, `pwm.h` és `music.h` header fájlok tartalmazzák.

Ebben a példában az időzítő periféria az úgynevezett *"Clear Timer on Compare Match"* (CTC) módban működik. Ez azt jelenti, hogy amikor a számláló és a komparátor (OCR) regiszter értéke megegyezik, akkor nem csak a kimenet vált állapotot, hanem a számláló (TCNT) is lenullázódik, ezáltal újraindul a számlálás. Ez a működés az adatlap 74. oldalán van lerajzolva, és elmagyarázva.

Figure 14-5. CTC Mode, Timing Diagram



***2.ábra** - Timer periféria működése (ATMega16A adatlap)*

A CTC üzemmódban a keletkező jel kitöltési tényezője mindig 50%, azonban a frekvenciája az OCR értékének megfelelően változik; ez az üzemmód tehát változó frekvencia előállítására alkalmas.

A zümmer program két egyszerű dallamot játszik le a hangszórón, ehhez nincs szükség változó kitöltési tényezőre, viszont változó frekvenciára természetesen igen. A program igen egyszerűen működik, a hangokhoz tartozó OCR értékeket egy táblázatból veszi, tehát nem számolja ki a kívánt frekvenciához tartozó OCR értéket.

Ennél az időzítő/számláló egységnél az OCR értékek már nem a 0..255 tartományban vannak. Ez azért van, mert ez az egység 16 bites, ami azt jelenti, hogy a számláló regiszter (`TCR1`) és a komparáló regiszter (`OCR1A`) is 16 bitesek. Ennek megfelelően a beírható tartomány 0..65535 között van, tehát az előállított PWM (jelen esetben frekvencia) sokkal jobb felbontásban állítható.

16 bites regiszterek kezelése

Fontos, hogy a 8 bites processzoron hogyan tudjuk a 16 bites regisztereket írni-olvasni. A 16 bites számok belső ábrázolását és kezelését a fordító megoldja, de a processzor regiszterei fizikailag 8 bitesek, ezekbe nem lehet 255-nél nagyobb számot beírni. Ezt az ellentmondást úgy oldja fel a processzor felépítése, hogy például a 16 bites `OCR1A` regisztert két 8 bites regiszterként érjük el; az `OCR1AL` az alsó 8 bitet, míg az `OCR1AH` a felső 8 bitet tartalmazza. A beíráshoz tartozó kódrészlet itt látható:

```
//Felső 8 bit
OCR1AH = (note >> 8);
//Alsó 8 bit
OCR1AL = note & 0xFF;
```

Vannak azonban olyan regiszterek, amiknek az értékét egyetlen pillanatban kell kiolvasni, vagy beírni. Ilyenek a számlálók és a fenti OCR regiszterek is; ha először az egyik felét olvassuk és utána a másik felet, akkor a számláló a két olvasás között megváltozhat, és hamis értéket kapunk.

Erre a problémára a processzorban van egy külön 8 bites - rejtett - regiszter. Amikor egy 16 bites regiszter alsó 8 bitjét olvassuk vagy írjuk, akkor a felső 8 bitet ebbe a rejtett regiszterbe írja be, vagy onnan veszi ki a processzor. Ezért nagyon fontos a regiszterek hozzáférésének a sorrendje, mind íráskor, mind olvasáskor. Tehát:

Ha olvasni akarjuk az `OCR1A` regisztert, akkor először az alsó bájtot (`OCR1AL`) olvassuk ki. Ennek hatására a felső bájt értéke, pontosan ugyanabban a pillanatban, átíródik a rejtett regiszterbe. A következő olvasással a felső bájtot (`OCR1AH`) olvassuk, ekkor ez már a rejtett regiszterből jön, és ahhoz az időponthoz tartozik, amikor az alsó bájtot olvastuk ki.

Az írás pont fordítva történik. Először a felső bájtot (`OCR1AH`) írjuk, ahogy a fenti példában is látszik, ekkor ez az érték csak a rejtett regiszterbe kerül. A következő írás az alsó bájtra (`OCR1AL`) vonatkozik, és ekkor a rejtett regiszterrel együtt a teljes 16 bites érték íródik be az `OCR1A` regiszterbe egyetlen műveletben. A fent leírt működés az adatlap 87. oldalán, a 16.3-as pontban található.

További lehetőségek

A tárhelyen mellékelt forráskód kommentjei tartalmazzák a szükséges magyarázatokat a zümmer programhoz, a forráskódot itt nem részletezzük tovább. Azonban javaslom, hogy miután megértettétek a PWM alapvető működését, változtassatok a programokon, készítsétek el a saját programjaitokat, saját ötletek alapján.