



**KANDÓ KÁLMÁN
VILLAMOSMÉRNÖKI KAR**

Sándor Tamás – Milotai Zsolt

Beágyazott rendszerek

ÓE-KVK 2126

Budapest, 2015.

Készítette:

**Sándor Tamás, adjunktus, Óbudai Egyetem Kandó Kálmán
Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet**

Milotai Zsolt, okleveles villamosmérnök (MSC)

Közreműködött:

Bedő Sándor, villamosmérnök (BSC)

**Mészáros Dániel, műszaki ügyintéző, Óbudai Egyetem Kandó Kálmán
Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet**

Juhász Viktor, villamosmérnök (BSC)

Lektorálta:

**dr. Boráros-Bakucz András PhD, egyetemi tanársegéd, Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar, Műszertechnikai és Automatizálási
Intézet**

Tartalom

1	Előszó	7
2	A beágyazott rendszerek alapfogalma, alapvető jellemzői.....	9
2.1	Beágyazott rendszer alapfogalma	9
2.2	Valós idejű rendszerek.....	9
2.3	A kemény és a lágy valós idejű (HRT és SRT) rendszerek	10
2.4	Valós idejű rendszerek jellemzői	11
2.4.1	Válaszidő	11
2.4.2	Viselkedés csúcsterhelés esetén.....	11
2.4.3	Ütem vezérlése (control of pace)	12
2.4.4	Biztonság	12
2.4.5	Adatfájlok mérete	12
2.4.6	Redundancia	13
2.4.7	Vezérlési mód	13
2.4.8	Hibadetektálás	13
2.4.9	Hibás állapot kezelése	14
2.5	Ellenőrző kérdések.....	14
3	Mikrokontrolleres vagy mikroprocesszoros megvalósítások	16
3.1	Architektúrák	18
3.2	Multiprocesszoros rendszerek	18
3.3	Feladat kiosztás.....	18
3.3.1	Statikus feladat kiosztás	18
3.3.2	Dinamikus feladat kiosztás.....	19
3.3.3	Felépítése.....	20
3.3.4	Szorosan csatolt multiprocesszoros rendszerek.....	20
4	Mikrokontrollerek gyakorlati megvalósítása	26
4.1	32 bites rendszerek.....	26
4.1.1	Gyakorlati megvalósítás: ARM	26

4.1.2	Cortex-es ARM-ok.....	29
4.1.3	Atmel SAM3 és SAM4	35
4.1.4	A mikrovezérlő	35
4.1.5	A mikrovezérlő interfészei	35
4.1.6	A SAM3N-EK fejlesztői panel	37
4.1.7	Bevezetés az Atmel Studio 6.0 fejlesztői környezet használatába	
	39	
4.1.8	A SAM3N-re történő programfejlesztés alapvető kérdései.....	47
4.1.9	A SAM3 regisztereinek kezelése	47
4.1.10	Az interfészket működtető órajelforrás	49
4.1.11	Az IO-lábak funkcióinak multiplexálása	50
4.1.12	Általános célú IO (GPIO) kezelés	52
4.1.13	Késleltetés, időzítés.....	68
4.1.14	Megszakítások	78
4.1.15	Teszt üzemmódok	78
4.1.16	Az UART kontroller regiszterei.....	79
4.1.17	Példa az UART konfigurálására	84
4.1.18	Külső perifériák illesztése	86
4.1.19	A SAM3 mikrovezérlő ADC je	91
4.1.20	Az ADC felépítése és funkciói	91
4.1.21	Az AD konverter regiszterei.....	93
4.1.22	Az AD konverter csatornái	96
4.1.23	Megszakítások	97
4.1.24	Példa az ADC konfigurálására és használatára.....	100
4.1.25	DAC	100
4.1.26	Impulzus szélesség moduláció (PWM).....	107
4.1.27	A SAM3 időzítő és számláló egységei (TC)	118
4.1.28	Megszakítások	125

4.1.29	SPI interfész.....	137
4.2	Texas Instruments	154
5	Ipari hálózatokban alkalmazott kommunikációs hálózatok.....	154
5.1	RS485	154
5.2	CAN	156
5.2.1	Fizikai réteg	156
5.2.2	Üzenetek keretformátumai	159
6	Programozható logikai eszközök	170
6.1	SPLD	172
6.2	CPLD.....	177
6.3	FPGA.....	179
6.3.1	FPGA technológiák.....	183
6.3.2	Logikai cella	186
6.3.3	LUT	186
6.3.4	Alkalmazási területek.....	189
6.3.5	Xilinx® Series-7 FPGA-k	192
6.4	További programozható logikai eszközök.....	198
7	Fejlesztési lehetőségek	200
7.1	Fejlesztési folyamat FPGA-val.....	201
7.1.1	Logikai tervezés, rendszertervezés	202
7.1.2	A fejlesztés további lépései	211
8	Melléklet	212
8.1	számú melléklet, IO lábak funkciói	212
8.2	melléklet, Az órajeladáshoz és megszakításokhoz kapcsolódó elnevezések	214
8.3	melléklet, A SAM3N-EK és EXPBRD-t csatlakoztató összekötőpanel port kiosztása	216
9	Irodalom jegyzék, internethivatkozások.....	219

10 Ábrajegyzék	220
----------------------	-----

1 Előszó

Mi történik a háttérben abban az esetben, ha beszállunk egy autóba vagy egy liftbe, vagy esetleg egyszerűen telefonálunk? minden esetben egy beágyazott rendszer által működtetett elektronikus egységeket is tartalmazó eszközök vagy berendezést használunk. Manapság átszövik életünket azok az eszközök, amelyek beágyazott rendszereket tartalmaznak, még akkor is, ha közvetlenül nem is látjuk vagy tapasztaljuk jelenlétüket.

A jegyzet célja, hogy megfelelő elméleti alapokat összefoglalja az **Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Műszertechnikai és Automatizálási Intézet** oktatott **Beágyazott rendszerek** tárgyhoz, amellyel segítséget nyújt a harmadéves hallgatók számára az egyes specializációkban történő alkalmazás esetén. Emellett a beágyazott rendszer ismeretek hasznos alapot adhatnak a képzés során előírt **Projekt I.** és **Projekt II.** tárgyak sikeres teljesítéséhez.

A jegyzet a beágyazott rendszerek alapfogalmainak tárgyalását követően a multiprocesszoros rendszerek témaületen tárgyalja a szorosan és lazán csatolt rendszerek fajtáit, jellemzőit gyakorlati példákon keresztül bemutatva. Ezt követően a beágyazott rendszerek interfész megoldásaira tér ki, bemutatva a szabványos interfész megoldásokat.

A jegyzet épít a **Programozás II.** tárgyban bemutatott 8 bites mikrokontrolleres architektúra ismeretekre, ezeket kiegészítve részletesebb interfész ismeretekkel, kitérve a SPI és TWI interfész megoldásokra.

A 8 bites architektúrákat követően a jegyzet áttér a 32 bites architektúrák leírására, itt kiemelve az ARM (korábban **Advanced RISC Machine**, azelőtt **Acorn RISC Machine**) architektúra jelentőségét és szerepét a mai beágyazott megvalósítások esetében. Az ARM architektúrára épülő különféle gyártói megvalósítások, mint például az Atmel SAM3 és SAM4 sorozatinak részletes bemutatása, illetve a Texas Instruments saját megvalósításai.

A jegyzet zárásaként a beágyazott rendszerek egy másik gyakori

megvalósítási módját a programozható logikai hálózatok (CPLD, FPGA) általános jellemzőit, fajtáit és felépítését mutatja be.

Budapest, 2015.március 31.

Szerzők

2 A beágyazott rendszerek alapfogalma, alapvető jellemzői

A fejezetben tárgyalásra kerül a beágyazott rendszerek alapfogalma és alapvető jellemzői. A fogalmak tisztázása során törekszünk arra, hogy egyértelmesítsük azokat a későbbiekben gyakran használt kifejezéseket, amely a tématerület megkíván. Lássuk elsőként a beágyazott rendszerek fogalmát!

2.1 Beágyazott rendszer alapfogalma

A beágyazott rendszerek olyan *számítógépes eszközök*, amelyek alkalmazás-orientált célberendezésekkel, illetve komplex alkalmazói rendszerekkel szervesen *egybeépülve*, azok *autonóm* működését biztosítják, vagy segítik. [IEE Guidelines]. Közös jellemzőjük a *fizikai környezettel* való intenzív *információs kapcsolat*.

2.2 Valós idejű rendszerek

Egy rendszer valós idejűnek tekinthető abban az esetben, ha a külvilág felől érkező jelzésre **meghatározott időn belül** reagál. Ez meghatározott idejű reakció lehet rövid idejű, akár mikroszekundum nagyságrendű vagy akár néhány óra is, attól függően, hogy a rendszer, amely esetében értelmezzük, az egy nagyon gyors beavatkozást igénylő rendszer (pl. személygépjármű fékrendszer) vagy egy lényegesen lassabb rendszer (pl. hőtechnikai rendszer).

Egy másik definíciót is alkalmazhatunk, lássuk ezt is. „Azokat a rendszereket, amelyekkel szemben a környezeti, valós időskálához kötött idő-követelményeket támasztunk, valósidejű rendszereknek nevezzük. Előírhatjuk például, hogy a rendszer egy környezeti eseményre mennyi időn belül reagáljon, vagy milyen időzített akciókat hajtson végre. [...] A valósidejű rendszerek két alapvető fajtája:

- a szigorúbb feltételeket teljesítő **kemény valósidejű rendszerek** (**hard real-time**) biztosítják, hogy a kritikus munkák befejeződjenek időben,
- míg a **lágy valósidejű (soft real-time)** rendszerek csak azt garantálják, hogy a kritikus munkák prioritással futnak.”[1]

2.3 A kemény és a lágy valós idejű (HRT és SRT) rendszerek

A **kemény valós idejű** rendszer (**hard real-time system, HRT**) esetén a rendszernek meghatározott időn belül mindenképpen kell válaszolnia a környezetből érkező jelzésre, mert ha nem, akkor az katasztrófális következményekkel jár.

A **lány valós idejű** rendszer (**soft real-time system, SRT**) esetén a rendszernek nem kell meghatározott időn belül válaszolnia a környezetből érkező jelzésre, megengedett az idő túllépés is, az eredmény értékes lehet az időkorláton túl is, viszont az idővel degradálódhat is.

2.4 Valós idejű rendszerek jellemzői

Foglaljuk össze, hogy a beágyazott rendszerek esetében milyen főbb jellemzőket tudunk megemlíteni:

- Válaszidő,
- Viselkedés csúcsterhelés esetében,
- Útem vezérlése,
- Biztonság,
- Adatfájlok mérete,
- Redundancia,
- Adatintegritás,
- Vezérlési mód,
- Hibadetektálás.

Nézzük meg ezek után részletesen kifejtve az egyes fogalmakat.

2.4.1 Válaszidő

A **válaszidő (response time)** az az idő, amely idő alatt a beágyazott rendszernek reagálnia kell a környezetből érkező eseményre, vagy az az idő, amely időnként interakciót kell végrehajtania a rendszernek.

Az idő nagysága az adott környezeti technológiától függhet. Gépjárművek esetében a légzsák bekapcsolásának ütközés esetében másodperc töredéke alatt meg kell, hogy történjen, míg egy hőtechnikai szakasz irányítása esetében akár percek is eltelhetnek a válasz megérkezéséig.

2.4.2 Viselkedés csúcsterhelés esetén

Csúcsterhelésnek nevezzük (peak-load performance) azt az állapotot, amelyet a rendszer meghibásodás nélkül elvisel, illetve ezen maximális terhelés mellett a rendszer még teljesíti a számára előírt követelményeket. A rendszerek különféle módon viselkednek csúcsterhelés esetén.

A kemény valós idejű rendszerek (HRT) esetén jól definiált kell, hogy legyen ez a viselkedés. Tervezéskor biztosítani kell, hogy a beágyazott rendszer

minden szituációban az időkorláton belül teljesítse feladatát, hiszen a HRT rendszerek éppen azáltal valósítják meg a velük szemben megfogalmazott elvárásokat, hogy még a ritkán előforduló csúcsterhelések idején is jósolható módon viselkednek.

A lágy valós idejű (SRT) rendszereket átlagos teljesítmény-jellemzőkre tervezük, a ritkán előforduló csúcsterhelések következményeit - gazdaságossági megfontolásból – elviseljük azt is, hogy a beágyazott rendszer ne teljesítse az előírt követelményeket ideiglenesen.

2.4.3 Ütem vezérlése (control of pace)

A beágyazott rendszerek esetében a feladatok végrehajtását különféle módon lehet megoldani. Az ütemezett feladatok azok, amelyek meghatározott időnként végrehajtásra kerülnek, illetve valamilyen külső ütemező esemény hatására hajtódnak végre.

A kemény valós idejű (HRT) rendszernek minden körülmények között szinkronban kell lennie környezetének állapotával. A szinkron állapot fenntartását biztosíthatja belső ütemező, vagy a környezetből érkező szinkron jel.

A lágy valós idejű (SRT) rendszerek befolyásolják környezetüket, ha nem képesek eleget tenni feladatuknak, például megnövelik a végrehajtandó feladat végrehajtási idejét.

2.4.4 Biztonság

A biztonság kritikusságának mértékétől függően sokféle feladat merülhet fel tervezési időben. Autonóm hibadetektálási mechanizmusokat kell kidolgozni, amelyek valamilyen “talpra állítási” (recovery) akciót indítanak az adott alkalmazás által diktált időviszonyok mellett.

2.4.5 Adatfájlok mérete

HRT rendszerek kisméretű adatfájlokon dolgoznak, amelyek valós idejű adatbázist alkotnak. Ezek jellemzője az adatintegritás rövid idejűsége, mert az

idő műlásával az adatok jelentős része aktualitását veszíti.

Az SRT rendszerekben éppen ellenkezőleg a hosszú idejű adatintegritás fontos.

2.4.6 Redundancia

HRT rendszerek esetén ez a stratégia csak korlátozottan használható mert:

- az időkorlát tartása nehéz, mert a visszagörgetéshez szükséges idő nem, vagy nehezen jósolható,
- a környezetet befolyásoló “utasítás” nem tehető meg nem történtté,
- az ellenőrzési pontnál érvényes adatok az idő műlásával érvényüket veszítik.

SRT rendszerekben (pl. tranzakciós rendszerek) hiba esetén a számításokat “visszagörgetik” a legutolsó ellenőrzési ponthoz, amikor még biztosan helyes volt a működés és onnan kezdik a “talpra állítást”.

2.4.7 Vezérlési mód

A beágyazott rendszerek esetében alapvetően kétféle vezérlési módot tudunk értelmezni:

- eseményvezérelt (event triggered) rendszer
Az aszinkron módon érkező megszakítások kiszolgálása, illetve ebből adódóan dinamikus ütemezés szükség.
- idővezérelt (time triggered) rendszer
Minden kommunikáció, illetve feldolgozás központi időzítéshez (órához) szinkronizált módon történik.

2.4.8 Hibadetektálás

A hibadetektálás a valós idejű rendszerek esetében eltérően történhet HRT és SRT rendszerek esetében.

HRT esetében a hibadetektálás autonóm módon történik, ami azt jelenti, hogy külső beavatkozásra nincsen lehetőség. Nézzünk egy egyszerű esetet, például amikor a személygépjármű ütközik egy másik járművel. Ebben az

esetben a gépjárművezető személynek nincs esélye, hogy érzékelje az ütközést, illetve bekapcsolja a légzsákat, és ha ezt egy automatika nem tenné meg, akkor az ilyen jellegű balesetek következményei még súlyosabbak lennének.

SRT esetében lehetőség van arra, hogy emberi beavatkozás segítségével történjen meg a hiba detektálása. SRT esetében a hibadetektálásra különféle módszereket lehet alkalmazni:

- bemenetek és kimenetek állapotának visszajelzése (pl. LED-ek segítségével),
- meghibásodott rendszertől kapott hibaüzenetek,
- a berendezés meghibásodáskor vagy felvett állapotának jelzése.

Ezen visszajelzések alapján a hibadetektálást végző operátor gyorsabban beazonosíthatja a keletkezett hiba helyét és esetleg az okát is.

2.4.9 Hibás állapot kezelése

Katasztrófa megakadályozása *bénítással* (*fail-safe*) is történhet. Példaképpen a vonatok esetében, ha a biztonsági rendszer meghibásodik, akkor a szabad utat jelző jelzőlámpa rendszerében csak a piros lámpák fognak jelezni.

Másik módszert abban az esetben alkalmazzák, ha a normál működtetés már lehet hatásos, ilyenkor a katasztrófa elhárítás *extra eszközökkel* (*fail-operational*) történik. Példaképpen, ha a repülőgép robotpilótája már nem képes normál módon működtetni a járművet, akkor a pilótának át kell vennie az irányítást a gép felett, és kézi irányítással lehözni a repülőgépet.

2.5 Ellenőrző kérdések

1. Adja meg a beágyazott rendszer fogalmát!
2. Mit nevezünk valós idejű rendszernek?
3. Mit nevezünk HRT-nek?
4. Mit nevezünk SRT-nek?
5. Milyen tulajdonságokkal lehet jellemzni a valósidejű rendszereket?
6. Hogyan viselkednek csúcsterhelés esetében a valósidejű rendszerek?
7. Mi ütemvezérléseket alkalmaznak valósidejű rendszerek esetében?
8. Mekkora az adatfájlok mérete valósidejű rendszerek esetében?
9. Milyen mértékben lehet alkalmazni a redundanciát valósidejű rendszerek esetében?

10. Milyen vezérlési módokat alkalmaznak valósidejű rendszerek esetében?
11. Milyen típusú hibadetektálásra van lehetőség valósidejű rendszerek esetében?
12. Hogyan történik a hibás állapot kezelése valósidejű rendszerek esetében?

3 Mikrokontrolleres vagy mikroprocesszoros megvalósítások

A legelső számítógépek még egyszerű utasításkészletet és a programvégrehajtáshoz interpretált alkalmaztak. Majd ezt követően néhány állomás a fejlődésben:

- IBM architektúra (1950), IBM System/360,
- közvetlen hardveres feldolgozás,
- Összetett utasítások, Digital Equipment Corporation VAX (több száz utasítás, 200 címzési mód),
- Hardveres értelmezők Motorola 68000, mikroutasítások, vezérlőtárrak (control store),
- 1980, Berkeley RISC (Reduced Instruction Set Computer),
- CISC (Complex Instruction Set Computer), pl. DEC VAX,
- 486-os RISC mag és CISC mag (a kompatibilitás miatt).

Korszerű számítógépek tervezési módszerei:

- minden utasítást a hardver hajtson végre,
- Maximálni kell az utasítások kiadásának ütemét (MIPS, Millions of Instructions Per Second),
- Csak a betöltő és tároló utasítások hivatkozzanak a memóriára (memóriát kezelő utasítások lassúak),
- Sok regiszter kell.

Az órajel frekvencia növelésének korlátai miatt az utasításszintű párhuzamosság irányában mozdultak el a fejlesztések.

Párhuzamosság:

- Utasításszintű
- Processzorszintű

Többfázisú csővezeték

- Utasítás beolvasó egység
- Utasítás dekódoló egység
- Operandus beolvasó egység
- Utasítás végrehajtó egység
- Visszaíró egység

Korlátok: késleltetés, áteresztő képesség.

Szuperskalár architektúrák

- Kettős csővezeték (u pipeline, v pipeline – egyszerű egész és lebegőpontos művelet) a közös utasítás-beolvasó egységgel.
- Szuperskalár processzor 5 funkcionális (ALU1, ALU2, LOAD, STORE, lebegőpontos egység) egységgel (1987, P4).

Processzorszintű párhuzamosság

- Tömbprocesszorok (1972), adattömbök,
- Vektorprocesszorok (1974), minden összeadás egyetlen csővezeték elven működő összeadó egység.

3.1 Architektúrák

- Neumann architektúra:
 - o Adatút (data path),
 - o Aritmetikai-logikai egység (ALU, Arithmetic Logic Unit),
 - o Utasítások
 - Regiszter-memória,
 - Regiszter-regiszter,
 - o Adatút ciklus fogalma (betöltő-dekódoló-végrehajtó),
 - o Értelmező (interpreter),
- Havard architektúra.

3.2 Multiprocesszoros rendszerek

A mikroprocesszorok árának jelentős csökkenése lehetővé teszi, hogy azokat egy bonyolult rendszerben univerzális építőelemekként használjuk.

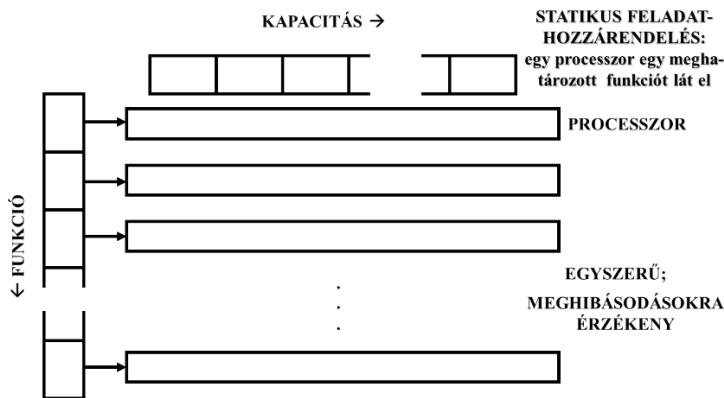
MULTIPROCESSZOROS RENDSZER: egymástól független (rész)feladatok konkurens feldolgozását végző többprocesszoros rendszer.

Két esemény konkurens, ha egyik sem tudja kauzálisan befolyásolni a másikat.

3.3 Feladat kiosztás

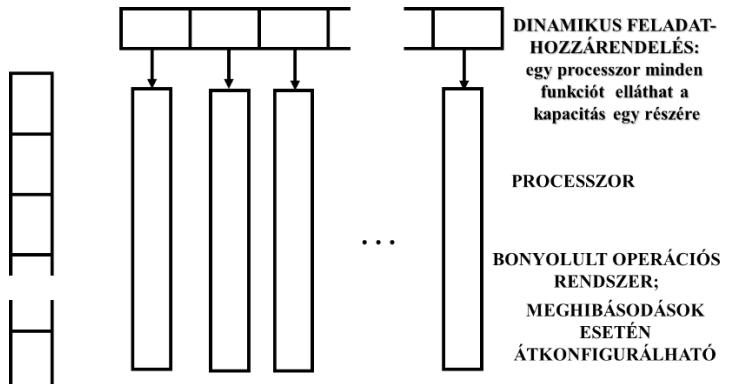
3.3.1 Statikus feladat kiosztás

STATIKUS FELADAT HOZZÁRENDELÉS: egy processzor egy meghatározott funkciót lát el.



1. ábra Statikus feladat hozzárendelés

3.3.2 Dinamikus feladat kiosztás



2. ábra Dinamikus feladat hozzárendelés

3.3.3 Felépítése

A multiprocesszoros rendszereknek két alapvető fajtáját különböztetjük meg:

LAZÁN CSATOLT RENDSZER:

- üzenet kommunikáció,
- minden processzornak saját operációs rendszere van.
- Egyszerű, lassú.

SZOROSAN CSATOLT RENDSZER:

- kommunikáció közös erőforráson keresztül,
- egyetlen operációs rendszer van.

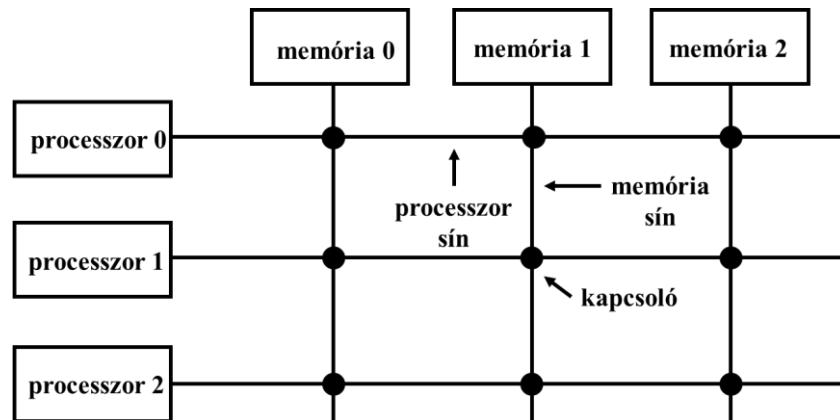
Bonyolult, gyors.

3.3.4 Szorosan csatolt multiprocesszoros rendszerek

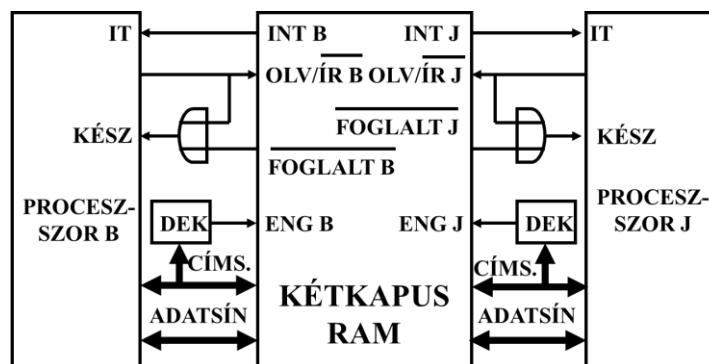
A szorosan csatolt rendszerek esetében a kommunikáció közös erőforráson keresztül valósul meg, egyetlen operációs rendszer van. Bonyolult, de gyors működésű.

Alapvető változatai:

- crossbar szervezés,
- multiport memóriára alapozott szervezés,
- rendszersínre alapozott szervezés.

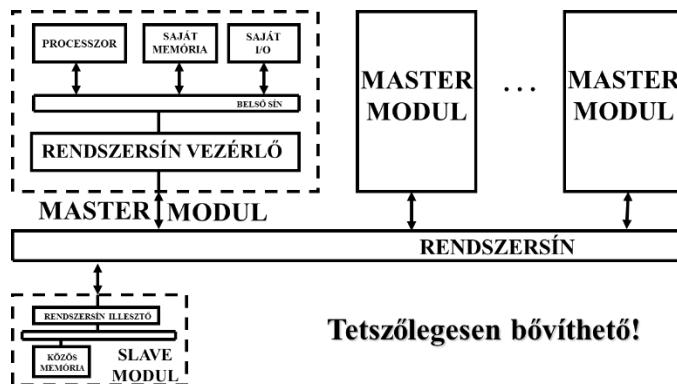


3. ábra Crossbar szervezésű szorsan csatolt multiprocesszoros rendszer



4. ábra Duálportos memória alapú szorosan csatolt multiprocesszoros rendszer

A rendszersíre szervezett szorsan csatolt multiprocesszoros rendszerek esetében Master-Slave kapcsolatban vannak az egyes modulok, és a Master modul magához ragadhatja a rendszersín vezérlését, és a közös erőforrásban információt elhelyezheti.

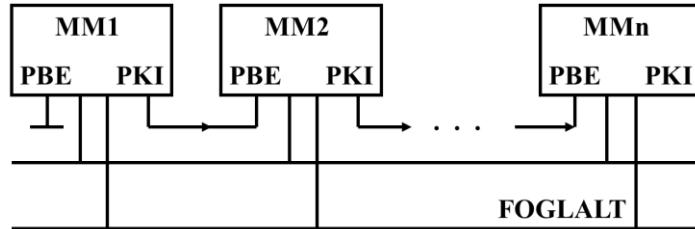


5. ábra Rendszersínre szervezett szorsan csatolt multiprocesszoros rendszer

A Slave modul nem ragadhatja magához a rendszersín vezérlését, de amikor megszólításra kerül, akkor a közös erőforráson keresztül adatokat továbbíthat. A rendszer (elvben) tetszőlegesen bővíthető és átkonfigurálható. Az egyes processzorokon futó folyamatok (a Slave modulban lévő) közös erőforráson keresztül kívánnak kommunikálni egymással. Fel kell oldani a közös erőforrás használatáért folyó versengést, de a folyamatokat futtató master moduloknak előbb hozzá kell férniük a rendszersínhez.

A Master modulok versenyeznek a rendszersínhez való hozzáférés jogáért:

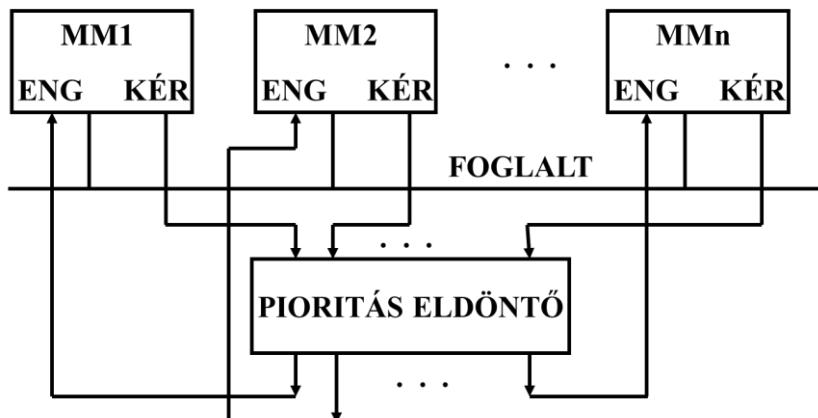
- SOROS (daisy chain), vagy
- PÁRHUZAMOS hozzáférés vezérlés.



6. ábra Master modul soros (daisy chain) rendszersín hozzáférése

A Master modul soros (daisy chain) rendszersín hozzáférésének előnye, hogy a kialakítás nagyon egyszerű. Hátránya viszont, hogy a terjedési késleltetés miatt csak nagyon kis rendszerekben alkalmazható (meg kell vární a legbaloldalibbtól is az igény végigfutását a láncon).

Másik hátránya, hogy nehéz átkonfigurálni (vezetéket kell elvágni és összekötni, huzalozott logika), de a kis rendszerek (nagy megbízhatóság) miatt erre menetközben nincs szükség. A rögzített prioritás miatt éhezés léphet fel.



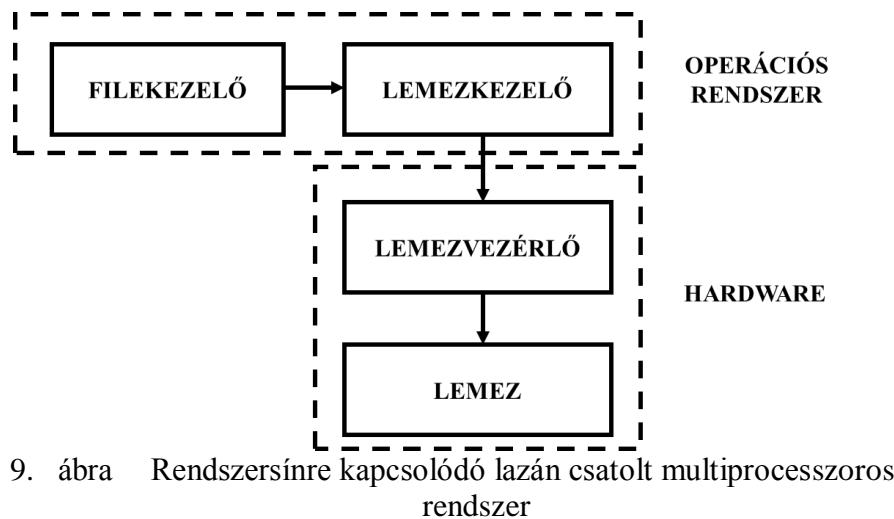
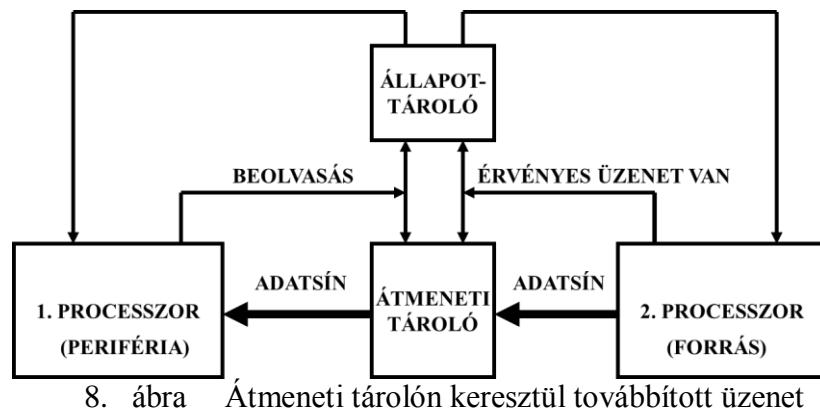
7. ábra Master modul párhuzamos rendszersín hozzáférése

A Master modul párhuzamos rendszersín hozzáférésének előnye, hogy a prioritás lehet rögzített (a prioritás eldöntő egyszerű kombinációs áramkör),

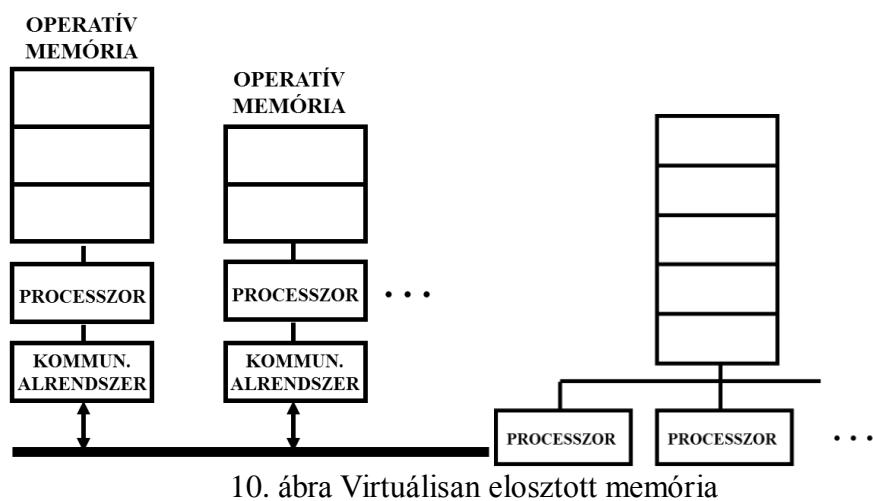
vagy változó (pl. körbenforgó - round robin - ekkor hosszabb idő alatt egyenlő esélyt kap minden modul).

A prioritás eldöntőt nagyobb rendszereknél elosztott módon valósítjuk meg (ellenkező esetben a rendszer kritikus része lenne).

3.3.4.1.1 Lazán csatolt multiprocesszoros rendszerek



A kommunikációs alrendszer a logikai pont-pont összeköttetést leképzi a fizikai átviteli közegre. A fizikai átviteli közeg korlátozott átviteli kapacitást biztosít, így torlódásvezérlés kell. minden processzor kap egy hitelértéket és akkor küldhet ki üzenetet, ha hitelértéke nagyobb, mint 0. Problémát jelent, hogy a hitel elveszésének kezelése elvileg megoldhatatlan nagy rendszerekben.



4 Mikrokontrollerek gyakorlati megvalósítása

4.1 32 bites rendszerek

4.1.1 Gyakorlati megvalósítás: ARM

Az ARM -ot egy kísérleti projektben fejlesztették ki 1983-ban az Acorn Computers Ltd.-nél. 1985-ben már megjelent az „ARM1”-nek nevezett fejlesztői környezet, majd egy évre rá forgalomba is hozták az „ARM2” nevezetű chipet. Ez a család már 26 bitet használt címzésre, 16 darab 32 bites regisztert tartalmazott, és 32 bites adatbusszal rendelkezett, ezen felül 30 ezer tranzisztor tartalmazott.

A folyamatos technológiai fejlődésnek köszönhetően további gyorsabb és újabb processzorok fejlesztése vált sürgetővé. Így megjelent az „ARM3”, amely már képes volt 25 MHz-ről működni, ellentétben az előzőekhez képest, amelyek csak 8MHz-es frekvenciával működtek. Ezek a processzorok már tartalmaztak egy egységes 4 kB Cache-t, amellyel tovább növelték a teljesítményt.

1990-ben a processzorok fejlesztését az Advanced RISC Machines Ltd. (az Apple Computer és az Acorn cég együttműködése) vette át.

A következő fontosabb család az ARM6, amely már a két cég együttműködésének köszönhetően jött létre. Ezek a chipek 32 bites címzással dolgoztak, és akár 33MHz –ről is képesek voltak működni. 1994-ben már az Apple cég „Apple Newton” nevezetű PDA-i ezeket a processzorokat használták. Bár az alapmag majdnem azonos méretű maradt a változtatások ellenére. Az „ARM2” 30000 tranzisztorral rendelkezett, az ARM6 csupán 35000 -rel. Viszont az ARM mag mellé opcionális részeket kínált, amelyekkel egy teljes CPU volt összeállítható, így alacsony költséggel nagy teljesítményt tudtak elérni.

Az ezredforduló környékén elkezdődtek a tervezések és a licensek árulása, így ennek köszönhetően további cégek (pl.: Samsung, Nintendo) jelentek meg az ARM gyártásában.

2004-ben jelent meg az ARM7, mely már akár 50MHz-ról is működött. Ezek a chippek már tartalmaztak: integrált SoC¹-t, DSP utasításokat, 3 lépéses csővezetéket, MMU²-t, MPU³-t.

Az ARM9 új család egyre inkább átvette a Harvard-architektúrát, amelyeknél már külön buszok és chace-ek voltak az utasításoknak és az adatoknak, ezzel is tovább növelte a feldolgozási sebességet.

Az órajel növelésekkel is voltak további változtatások. Bevezették a 3 fokozatú, és később az 5 fokozatú utasításokat, amely azt jelentette, hogy az ALU egyszerre akár 3 vagy 5 utasítást is le tud kezelni egy órajel alatt.

Csökkentettek a túlmelegedési problémákon is. Ezen kívül néhány ARM9 mag már tartalmazott olyan DSP utasításokat, mint pl.: a szorzás-összeadás (multiply- accumulate), melynek köszönhetően könnyebbé váltak a DSP algoritmusok leprogramozása.

Ezt követte az ARM11, majd a Cortex processzorok, tovább növelte: a memóriaterületet, maximális működési frekvenciát, DSP utasításokat és a csővezetékek (pipeline) számát. Ezen felül a Cortex-M3-as processzorok már teljesen áttértek a Harvard architektúrára, és már tartalmazzák a következőket:

- Thumb mód

¹ System on Chip

² Memory Management Unit

³ MMU- Memory Protection Unit

Sokszor a 32 bites adatbuszok nincsenek kihasználva, ezért az utasítás és az adat a kódsűrűség növelése érdekében lecsökkenthető 16 bitre ezek hossza. Hátránya ennek a megoldásnak, hogy az utasítások paraméterezhetősége lecsökken.

- **Thumb-2**

Továbbfejlesztették a Thumb módot úgy, hogy keverten találhatóak már benne 16 és 32 bit-es utasítások, és csak azoknak az utasításoknak a lecsökkentésére van lehetőség, amelyeknél ez veszteségmentesen megoldható.

- **VFP (Vector Floating Point)**

Hardveresen lehetőség nyílik a lebegőpontos számolásokra. Ezeket használják a mobiltelefonoknál, PDA-knál.

- **NEON**

A VFP továbbfejlesztése. Ez már több DSP kiegészítést tartalmaz, elősegítve a kép- és videó feldolgozási algoritmusok alkalmazását.

- **Jazelle**

Az okostelefonok és az egyéb elektronikai eszközök fejlődése váltotta ki ezt az igényt, hogy a Java kódot utasításszinten támogassa.

A tárgyhoz kapcsolódó laborokban Cortex-M3-as és Cortex M4-es processzorokkal lehet majd találkozni.

4.1.2 Cortex-es ARM-ok

4.1.2.1 Tulajdonságok, belső felépítés

A Cortex processzorok közül is megkülönböztetünk 3 típust:

A: Alkalmazásokhoz

High-End alkalmazások (okostelefonok, televíziók, stb.)

Jellemző rájuk a magas órajel, ami akár a GHz-es tartományt is elérheti. OS támogatás (MicroLinux, Android)

R: Real-Time processzorok

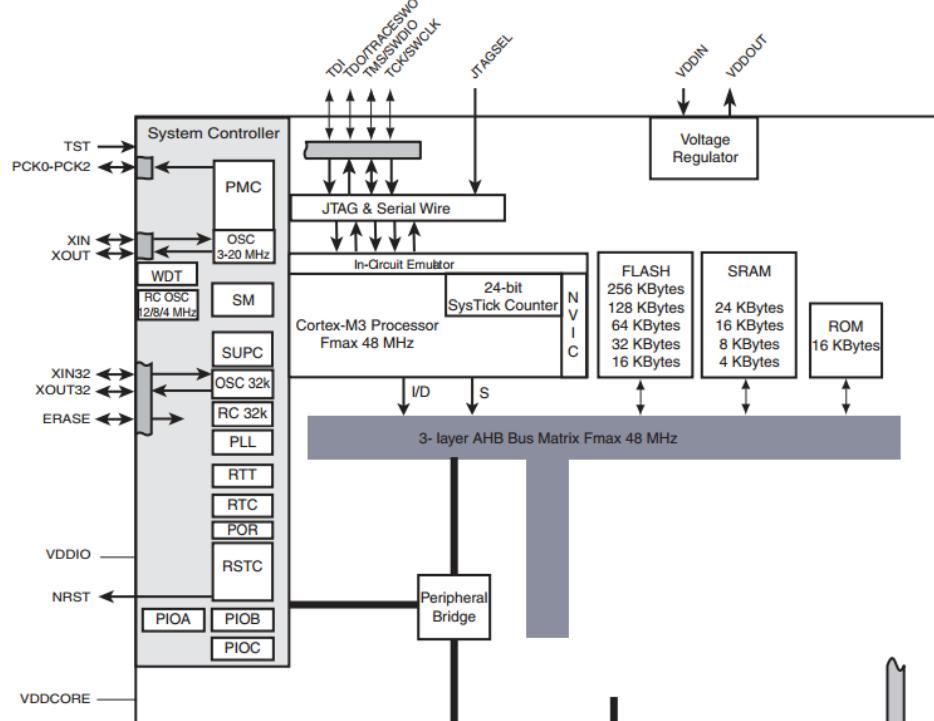
Főként ipari alkalmazásokban használják, mint például a hard Real-Time időzítések miatt (autóipar).

M: Mikrokontroller

Jellemzők:

- Alacsony ár
- Kisebb teljesítmény (maximális működési frekvencia kb. 120-180MHz)

Az Atmel cég által gyártott SAM3N4C típusú **Cortex-M3** –as processzor „alap” felépítése:

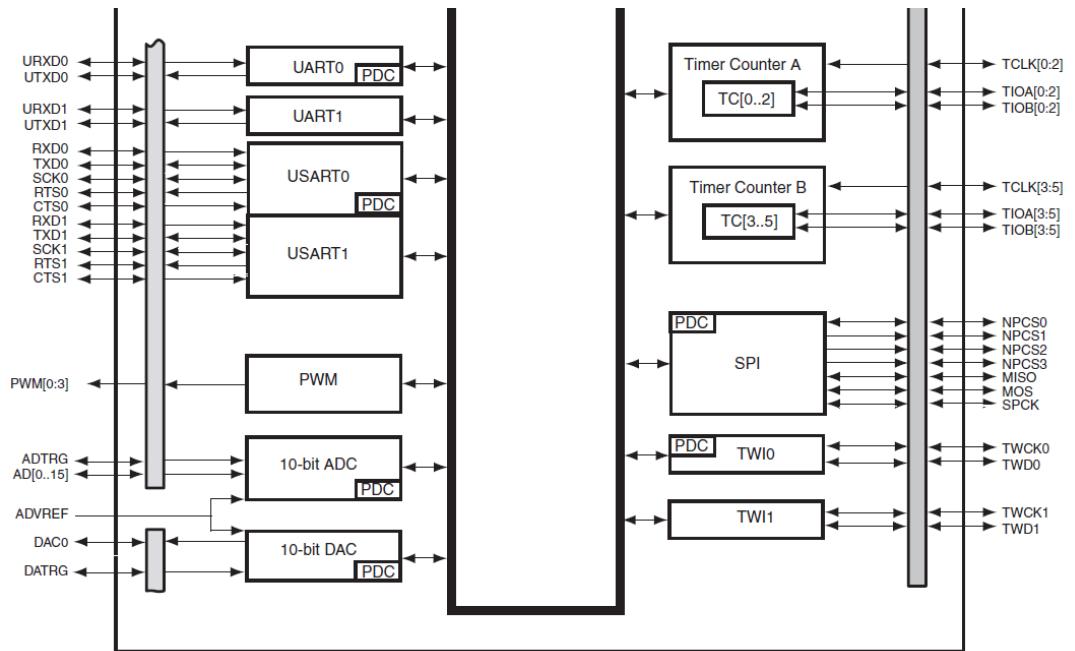


1. ábra – Cortex M3 belső felépítése

A képen látható Cortex M3 mikrokontroller a következőket tartalmazza:

- System Controller:
 - PMC (Power Management Controller) Ez az egység vezérli, szabályozza az energiagazdálkodást

- SUPC (Supply Controller) A redszer magjainak a tápfeszültség ellátásáért felel, itt valósítják meg a „Suspend” és a „Wake UP” módokat.
 - RC A mikrokontrollerben lévő belső oszcillátor, amelynek köszönhetően nem szükséges külső órajelforrás, ha nem szeretnénk pontos időzítéseket készíteni.
 - PLL (Phase Locked Loop = Fázis zárt hurok) Segítségével nagy pontosságú frekvenciákat tudunk létrehozni. Pl.: Az alap 12MHz-es külső órajel forrásunkat fel tudjuk szorozni akár 120MHz-re, és nnek következtében több számítást tud végrehajtani a mikrokontroller adott időintervallum alatt.
 - RTT (Real-time Timer) A valós idejű timer (időzítő) segítségével tudunk pontos, periodikus interrupt-okat (megszakításokat), triggereket beállítani.
 - RTC (Real Time Clock) Segítségével egyszeri beállítás után ki tudjuk olvasni belőle a pontos időt, dátumot. Még akkor is, ha a tápellátás megszakad.
- JTAG & Serial Wire – A JTAG adatvonalaikon keresztül lehet rácsatlakozni a programozóval a processzorra.
- NVIC (Nested Vectored Interrupt Controller)
- FLASH
- SRAM
- ROM (Read Only Memory)
- Belső kommunikációs buszok



2. ábra – Cortex M3 belső kommunikációs busz

SAM3N

Az Atmel cég fejlesztő board-ján egy SAM3N4C típusú mikrokontroller található, amely tartalmaz:

- 2 db UART-ot,
- Egy PLL-t (amellyel a működési órajel akár 130MHz-re is növelhető),
- 2 db TWI⁴ buszt,
- 1 db SPI buszt,
- 6 db 2 csatornás 16 bites Timer-t,
- 4 csatornás 16 bites PWM modult,
- 32 bites Real-Time Timer-t,

- 16 csatornás 10 bites ADC-t (Analog to Digital Converter).

A processzor mellett a panelon helyet kapott még:

- 2 inch –es TFT színes kijelzőt,
- Zigbee csatlakozót,
- UART csatlakozót (D-SUB9),
- MicroSD kártyafoglalatot,
- QTouch csuszkát és gombokat,
- A processzor minden lába ki van vezetve a panel szélére, további fejlesztések biztosítása érdekében.

Az Atmel cég részéről a következő fejlesztési lépés a Cortex M4-es típusú mikrokontrollerek voltak, amelyek annyiban térnek el a Cortex M3-asoktól, hogy tovább bővítették az interfészei számát és a működési frekvenciát, illetve jelfeldolgozási feladatok támogatását biztosító utasításokkal. Nézzünk ezek közül néhány jellemzőt:

- FPU (Floating Point Unit),
- A működési frekvencia elérheti a 180 MHz–et is,
- Növelték a DSP utasításokat,
- Megnövelték a megszakítások számát,
- Kiegészítették a Thumb-2 utasításkészletet.

Az FPU-nak köszönhetően azok a kódok, amelyek már tartalmaznak lebegőpontos számítást a processzor hardveresen hajtja végre, és ennek

köszönhetően a lebegőpontos számításokat tartalmazó szoftverek kisebb méretűek és gyorsabb végrehajtásúak lesznek.

A DSP utasítások: A mikrokontroller tartalmaz hardveres szorzót, amely képes már egy órajel alatt elvégezni két 32 bites regiszter összeszorzását.

Az interruptok számát megnövelték, így a fizikai interruptok száma elérheti akár a 240–et.

4.1.3 Atmel SAM3 és SAM4

Az Atmel SAM3N-EK fejlesztői panelje. A SAM3N-EK fejlesztői panel egy SAM3N4C típusú mikrovezérlő köré épül, külső perifériákkal kiegészülve.

4.1.4 A mikrovezérlő

A SAM3N4C az AT91SAM családba tartozó mikrovezérlő 32 bites ARM Cortex-M3 processzor maggal rendelkezik, tartalmaz 24 KB statikus RAM-ot, 256 KB FLASH RAM-ot, és rendelkezik számos interfésszel. A mikrovezérlő órajel frekvenciája 48 MHz, működési feszültség tartománya 1,62V - 3,6V között van.

A SAM3N4C a SAM3 eszközcsalád belépő szintű tagját képzi, viszonylag alacsony áron hozzáférhető (~5€ - 2014) és működéséhez kevés külső alkatrészt igényel, ezért ideális választás lehet olyan alkalmazásoknál, ahol a 8bites AVR-ek teljesítménye már nem elegendő.

4.1.5 A mikrovezérlő interfészei

A mikrovezélő interfészei között a legalapvetőbb az általános célú ki- és bemeneteket kezelő PIO (Paralell Input Output) párhuzamos bemenet és kimeneteket kezelő interfész.

Számos különböző megoldással rendelkezik, időzítések késleltetések megvalósítására. Egyszerűbb időzítési, késleltetési célra használhatjuk a processzor órajelről működő 24 bites SysTick időzítőt, vagy a 32768Hz-es külső órajelről működő 32 bites valós idejű időzítőt (RTT).

Összetettebb időzítési, időmérési vagy jelgenerálási feladatokra a mikrovezérlő rendelkezik két darab, három csatornás 16 bites számláló időzítő egységgel, amelyre az adatlap a Timer/Counter (TC) néven hivatkozik. Az impulzusszélesség modulált jelek előállítására pedig rendelkezésre áll a TC egységnél egyszerűbben konfigurálható négy csatornás PWM kontroller.

Az eszköz rendelkezik az analóg jelek digitalizálását végző ADC, és a digitális adatokat analóg jellé alakító DACC egységekkel. Az analóg digitális konverter 10 bites felbontású, bemenetén 16 csatornás multiplexerrel. A digitális analóg konverter (DACC) 10 bites, és egyetlen kimenet meghajtására alkalmas.

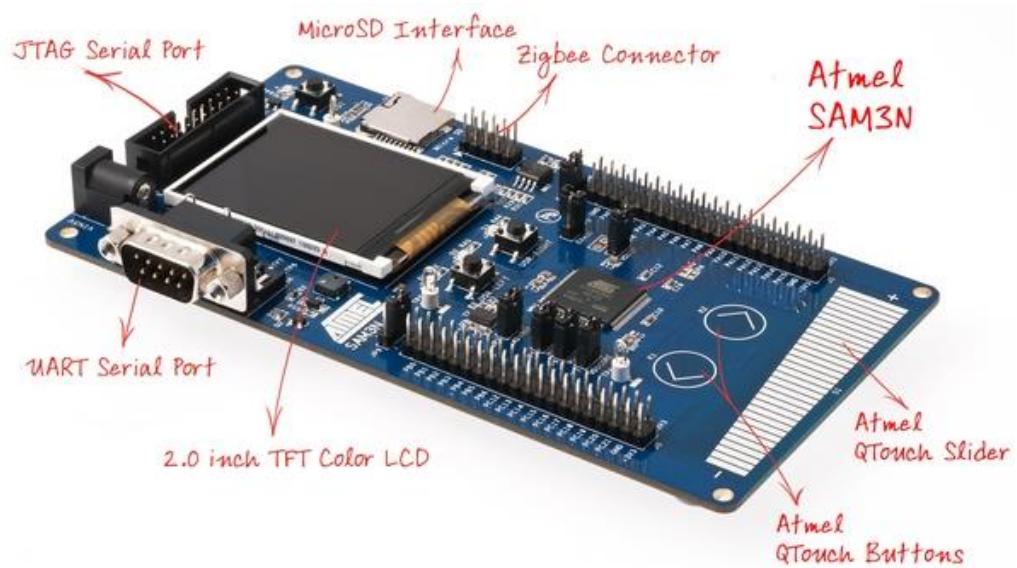
Az interfések sorából nem maradhatnak ki a különböző szinkron és aszinkron kommunikációt megvalósító megoldások sem, így rendelkezésre áll két UART, két USART, két I²C kompatibilis TWI, és egy SPI interfész.

4.1.6 A SAM3N-EK fejlesztői panel

A SAM3N-EK fejlesztői panel a központi elemét képező SAM3N4C mikrovezérlőn kívül rendelkezik még számos egyéb perifériával és bővítési lehetőséggel.

A panelen elhelyezett perifériák

A panelen találhatunk egy AT25DF321 típusú 32 Mbit méretű flash memóriát, melyet SPI buszon keresztül lehet írni és olvasni. A mikrovezérlő UART0-ás lábaira kapcsolódik egy MAX3232CSE típusú RS232 szintillesztő, amely a hozzáhuzalozott D-SUB 9-es csatlakozón keresztül lehetőséget nyújt soros kommunikáció megvalósítására. A fejlesztői panel rendelkezik egy 176x220 pixel felbontású színes grafikus LCD-vel. A grafikus LCD vezérlője ILI9225B típusú, és SPI buszon kommunikál.



1. ábra a SAM3N-EK fejlesztői panel

Az LCD háttérvilágításának vezérlése egy külön DC-DC konverter segítségével történik. A panel rendelkezik egy hangkeltésre alkalmas csipogóval, amely egy hardveres PWM kibocsátására alkalmas IO lábra lett kötve. A panelen helyet kapott egy SD-kártya foglalat, és kialakításra került több "q-touch" kapacitív érintés érzékelő is.

A panelen elhelyezésre került még két nyomógomb és három LED is, amelyek rendkívül hasznosak az általános célú IO kezelés elsajátításánál.

A mikrovezérlő 64 IO lába a panelen elhelyezett két tüskesorra ki lett vezetve, amelyek alkalmasak általános célú ki és bemenetkénti (GPIO) használatra, vagy a mikrovezérlő interfészeinek speciális céljaira.

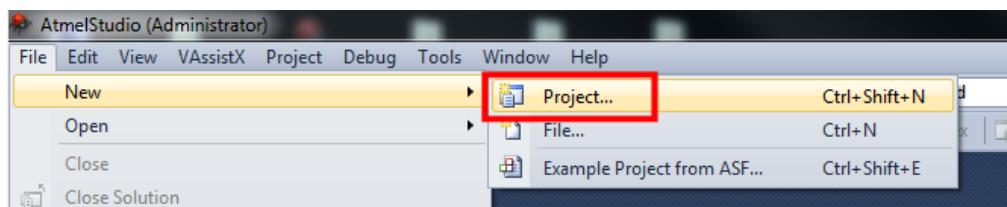
A tüskesoron keresztül a panel kiegészítésre került még további perifériákkal. A panelhez készült egy összekötő egység, amely lehetőséget ad a SAM3N-EK panel és az oktatásban használt 8 bites T-Bird kiegészítő paneljének, az EXPBRD -nek összekapcsolására.

4.1.7 Bevezetés az Atmel Studio 6.0 fejlesztői környezet használatába

Ebben a fejezetben ismertetésre kerül a fejlesztői környezet használatának és a mikrovezérlő programozásának (program letöltésének) módja. Az Atmel cég honlapjáról ingyenesen letölthető Atmel Studio 6.0 fejlesztői szoftver elindítása után, lehetőség van új projekt létrehozására, mintaprogram betöltésére az ASF (Atmel Software Framework) könyvtárból, vagy már meglévő projektek megnyitására.

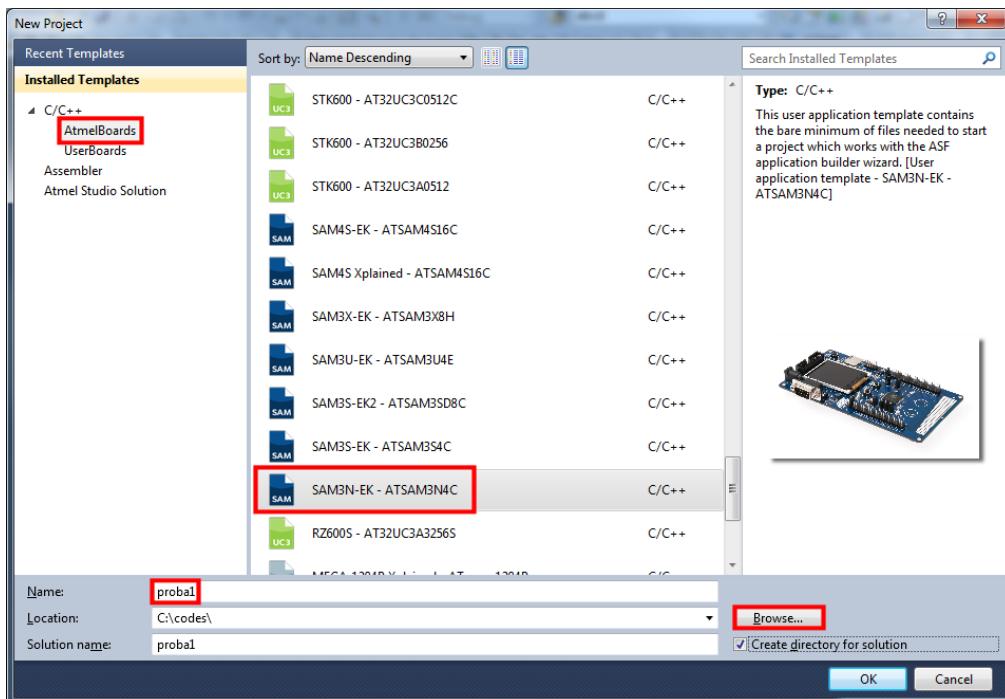
4.1.7.1 Új projekt létrehozása

Az új projekt létrehozása a File menü > New > Project... gombra kattintással történik.



2. ábra új projekt létrehozása

Ekkor megjelenik a "New Project" ablak, ahol az Installed Templates panelről ki kell választani a "C/C++" \ "AtmelBoards" lehetőséget, majd a mellette lévő listában legörgetve kiválasztani az általunk használt panel típusát (SAM3N-EK ATSAM3N4C). Lejjebb megadhatjuk a projekt nevét, és az elérési utat, ahova a projekt elmentésre kerül. Az OK gombra kattintva az új projekt létrehozása megtörténik.



3. ábra a projekt beállításai

Az új projekt létrehozása után az ablak baloldali részében megjelenik main.c fájl, amely egy forráskód sablont tartalmaz.

A forráskód sablont kitörölve kicserélhetjük azt az alábbi programra, ami a panel D3-al jelzett sárga LED-jét villogtatja.

```
#include "sam.h"
#define BOARD_MCK 48000000

int main (void)
{
    SystemInit();
    WDT->WDT_MR = WDT_MR_WDDIS;
```

```

PMC->PMC_PCER0 |= PMC_PCER0_PID11;
PIOA->PIO_OER |= PIO_PA25;
PIOA->PIO_PER |= PIO_PA25;
PIOA->PIO_PUDR |= PIO_PA25;

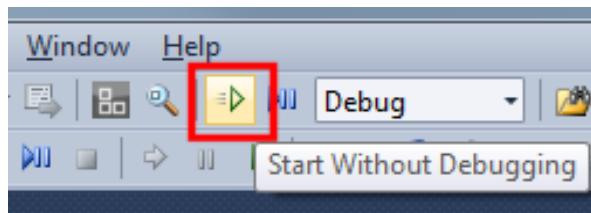
while(1)
{
    if(PIOA->PIO_PDSR&PIO_PA25)
        {PIOA->PIO_CODR |= PIO_PA25;}
        else{PIOA->PIO_SODR |= PIO_PA25;}
    for(volatile int i=0;i<10000000;i++);
}

```

4.1.7.2 A projekt lefordítása és letöltése a mikrovezérlőre

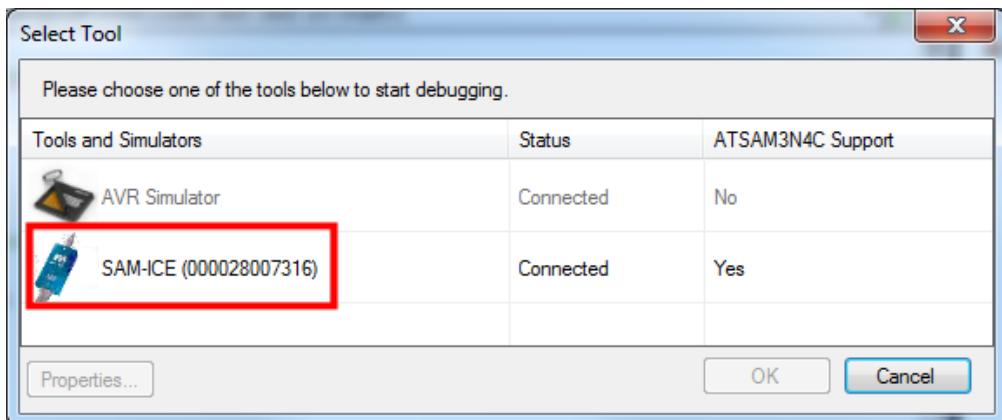
Fordítás és program letöltés egy lépésben

A beillesztett forráskód lefordítását és a mikrovezérlőre való letöltését (ráprogramozását) egy lépésben végrehajthatjuk. A "Start Without Debugging" gombra kattintva.



4. ábra fordítás és program letöltés

A mikrovezérlő programozásakor első alkalommal a felugró ablakban ki kell választani a programozó típusát.



5. ábra programozó eszköz kiválasztása

Az OK gombra kattintva megtörténik a programletöltés. A sikeres fordítás és programletöltés után a program futni kezd a mikrovezérlőn, villogatva a panel D3-al jelzett sárga LED-jét.

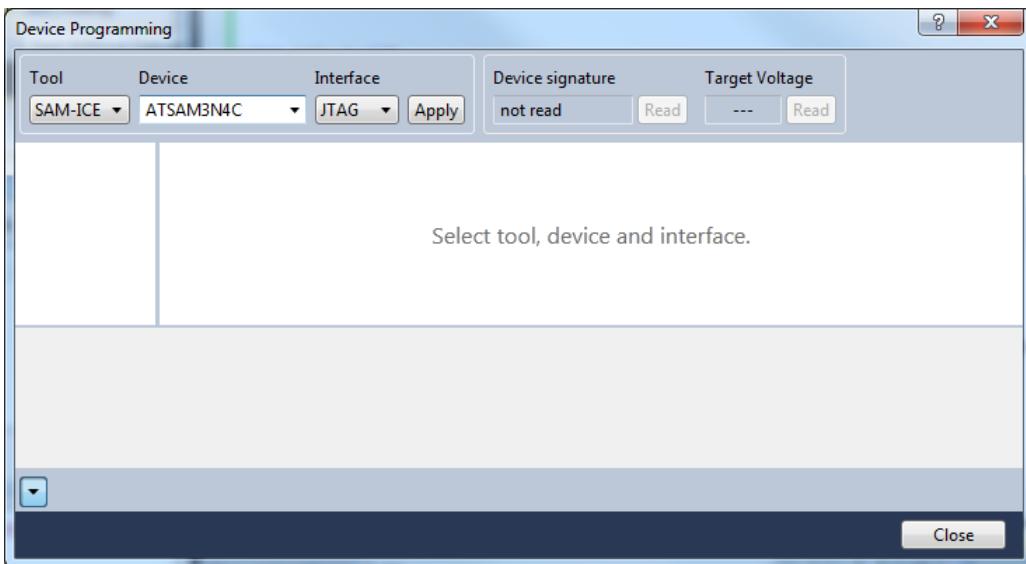
A program lefordítása

Ha csak a fordítást szeretnénk elvégezni, beillesztett forráskódot a Build menü "Build Solution" menüpontjával, vagy az F7 gomb megnyomásával fordíthatjuk.

A bináris fájl letöltése

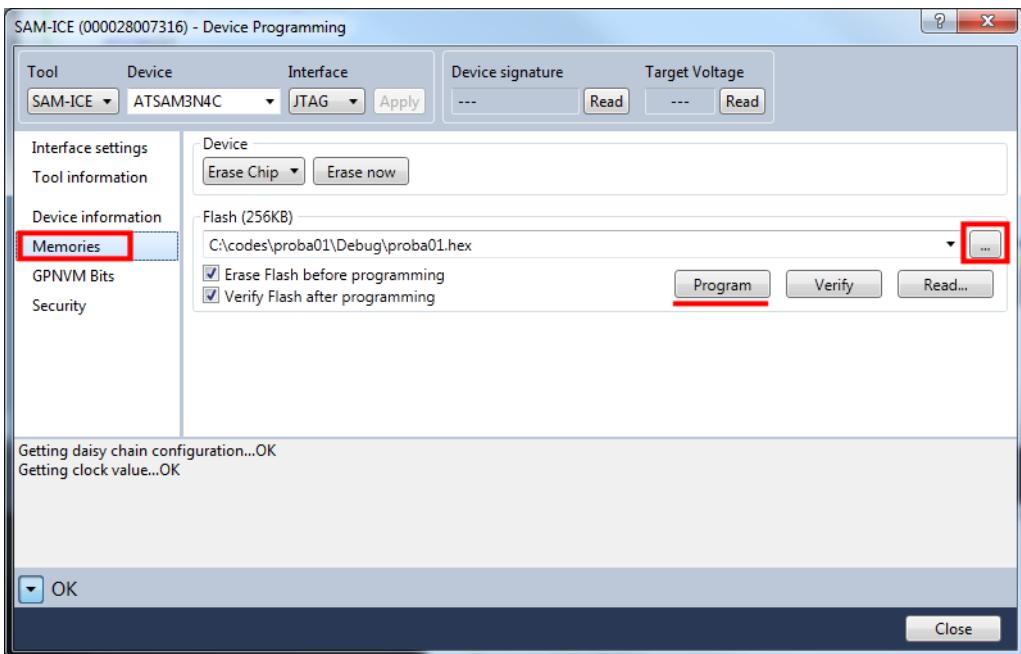
A már előzőleg lefordított projektek binárisa a mirkovezérlőre a program fordítása nélkül, külön is felprogramozható.

A Tools menü, Device Programming menüpontjára kattintva megjelenik a Device Programming ablak. A programozó típusa (SAM-ICE), a programozandó eszköz típusa (ATSAM3N4C), és az interfész (JTAG) kiválasztása után az apply gombra kell kattintani.



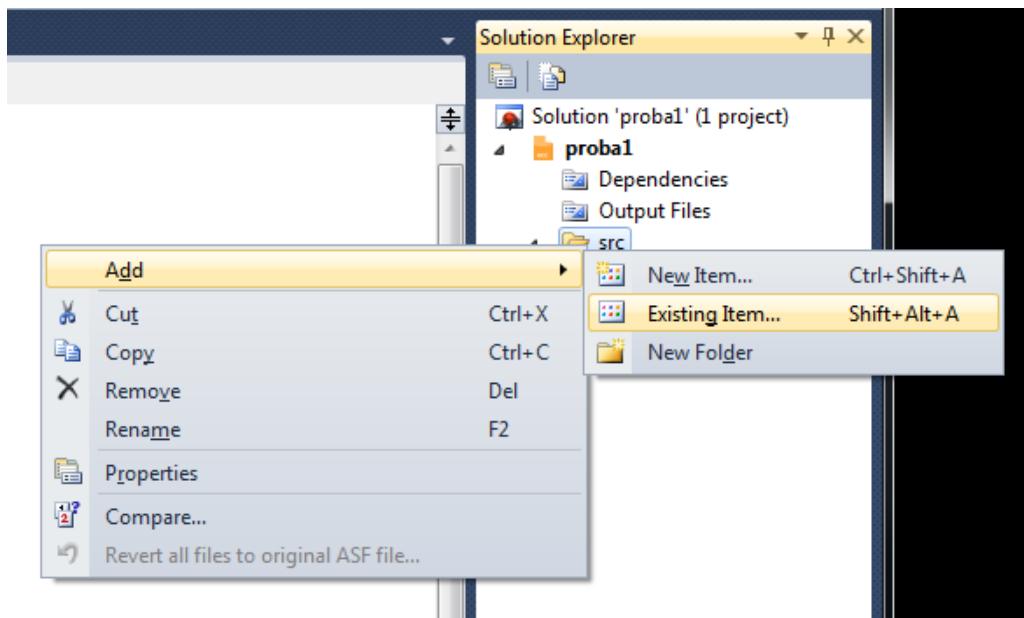
6. ábra a programozó beállítása

Ezek után a Memories menüpontra kattintva kiválaszthatjuk a mikrovezérlőre letölteni kívánt bináris fájlt. A Program gombra kattintva elvégezhető a programeltöltés. A bináris fájl az adott projekt könyvtár Debug alkonyvtárában található hex kiterjesztésű fájl.



7. ábra bináris fájl kiválasztása

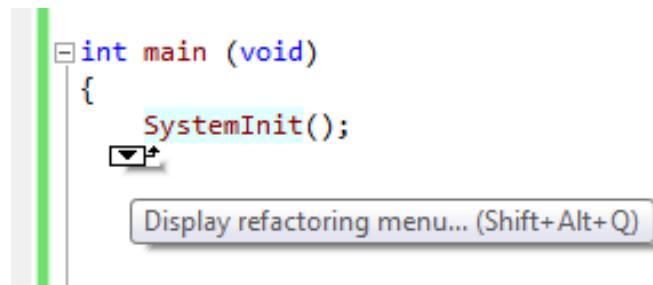
Az ablak jobb oldali részében a Solution Explorer panelen látható fa struktúrában találtatjuk a projekthez tartozó fájlokat. Ha a fa struktúrában található src mappára, az egér jobb gombjával rákattintunk, akkor az Add menüpontban a projekthez hozzáadhatjuk saját headher és C fájljainkat.



8. ábra különböző fájlok alkalmazása

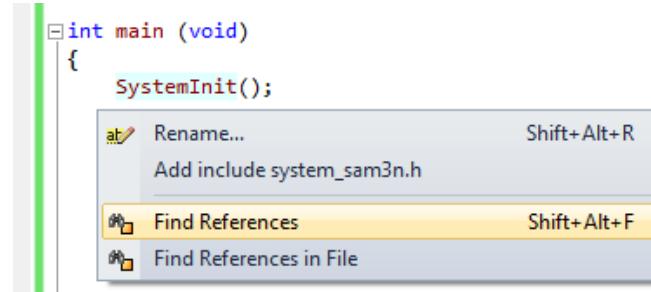
4.1.7.3 A Refactorig menü és használata

Az Atmel Studio 6.0 fejlesztői környezet felismeri, és színezéssel kiemeli a forráskód különböző elemeit. Az egeret egy adott elem (függvény, változó, struktúra, stb.) fölött húzva, egy apró nyíl jelenik meg.



9. ábra Refactoring menü

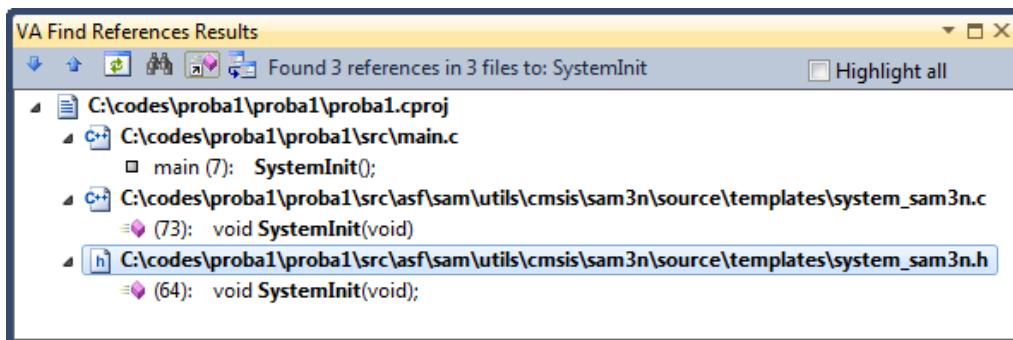
A nyílra kattintva láthatóvá válik az elemhez tartozó Refactoring menü.



10. ábra Find References

A Refactoring menüben lehetőség van az adott elem csoportos átnevezésére, vagy az elem "forrásának" visszakeresésére. Az elem "forrása" visszakereshető a Find References menüpontra való kattintással.

Ekkor a "VA Find References Results" panelen megjelenő listában látható, hogy az adott elem a projekt, mely fájljaiban található meg, így könnyen visszakereshetjük, hogy az elem hol és miként lett definiálva, és hol kerül felhasználásra. A listába kattintva az adott fájl közvetlenül is megnyitható.



11. ábra „VA Find References Results” ablak

4.1.8 A SAM3N-re történő programfejlesztés alapvető kérdései

4.1.9 A SAM3 regisztereinek kezelése

A SAM3 regisztereinek kezelése eltér a 8 bites Atmeleknél megszokottól, mivel a SAM3-nál a regiszterek száma nagymértékben növekedett, így szoftveres oldalról az összetartozó regisztereket struktúrákba szervezték. Az adott regisztereket struktúrapointerek segítségével lehet kezelní.

A nullás számú UART interfész regiszterei például az UART0 struktúrán keresztül érhetőek el.

A következő példa az UART0 interfész UART_THR nevű (adatküldés) regiszterébe történő adat írását szemlélteti.

```
UART0->UART_THR = data_out;
```

Az adat kiolvasás a fenti adat íráshoz hasonlóan, struktúra pointeres megoldással az UART0 struktúra UART_RHR (adat fogadó) regiszterén keresztül történik.

Adat kiolvasása az UART0 struktúra UART_RHR adatfogadó regiszteréből:

```
data_in = UART0->UART_RHR;
```

A SAM3 regiszterei három félre típusúak lehetnek. Egy adott regiszter lehet:

- csak írható,
- csak olvasható,
- vagy irható-olvasható.

A következő példa az UART0 interfész, küldő (TX) egységének engedélyezését mutatja be.

Az engedélyezés a control regiszter, engedélyező bitjének 1-re állításával (maszkolásával) történik.

UART0->UART_CR |= (1<<6);

Mivel az adott regiszterekhez tartozó bit pozíciók előre definiálva vannak, így a műveletet így is elvégezhetjük.

UART0->UART_CR |= UART_CR_TXEN;

A TX egység letiltása az engedélyezéshez hasonlóan történik, a megfelelő bit 1-re állításával (maszkolásával).

UART0->UART_CR |= (1<<5);

UART0->UART_CR |= UART_CR_RXDIS; // ez az előző sorral egyenértékű, de így szemléletesebb.

A mikrovezérlő bizonyos funkcióihoz több regiszter is tartozhat. Például külön regiszterekben lehet elvégezni egy adott port láb engedélyezését, letiltását vagy a hozzá tartozó állapot (engedélyezett/letiltott) lekérdezését, megfelelő bit 1-re állításával (maszkolásával).

A következő példa a PIOA (PORT A) 16. lábának engedélyezését mutatja be.

PIOA->PIO_PER |= (1<<16);

Ahol a PIOA struktúra PIO_PER (port engedélyező) regiszterének 16. bitjét 1-re állítjuk (maszkoljuk).

A port láb letiltását hasonlóképp, a letiltásért felelős regiszter megfelelő bitjének 1-re állításával (maszkolásával) érhetjük el.

PIOA->PIO_PDR |= (1<<16);

A PIOA port lábak engedélyezett vagy letiltottságának állapotát a PIOA struktúra PSR eleme (regisztere) tartalmazza, ahol a regiszter bitjeinek száma az azonos számú port lábhoz van rendelve.

A regiszter tartalmát a következőképp lehet kiolvasni:

status = PIOA->PSR;

4.1.10 Az interfészket működtető órajelforrás

A SAM3 bármely belső interfésze esetén érvényes (legyen az a PIO, Timer, UART, AD, stb.), hogy az adott egység bekapcsolásához, működtetéséhez órajelet inicializálni kell! A RESET után a mikrovezérlő perifériáinak órajel forrásai ki vannak kapcsolva. Ennek az energiafogyasztás minimalizálása érdekében van jelentősége. Az az interfész/vezérlő, amely nem kap órajelet, ki van kapcsolva, így nem is fogyaszt. A belső perifériák órajelét a Power Management Controller PMC regiszterein keresztül lehet kezelní.

Az órajelek bekapcsolása a PMC struktúra PMC_PCER0 regiszterén keresztül történik, a perifériáknak megfelelő bitek 1-re állításával (maszkolásával).

Az órajelek engedélyezésekor az adott interfész ID számával lehet hivatkozni. A tárgyalásra kerülő interfészek ID számát az 1. számú függelék tartalmazza.

A következő példa a 8-as számú interfész ID-vel rendelkező UART0 vezérlő órajelének engedélyezését mutatja:

PMC->PMC_PCER0 |= PMC_PCER0_PID8;

ahol a PMC a Power Management Controller struktúra, a PMC_PCER0 a PMC struktúra Peripheral Clock Enable regisztere, amellyel az órajel

bekapcsolása történik PMC_PCER0_PID8 a 8-as ID-vel rendelkező interfész számára az UART0 órajel engedélyező bitjének 1-re állításával.

A PMC_PCER0_PID8 a következőként van definiálva: (0x1u << 8).

Az UART vezérlő órajelének letiltása pedig a következőképpen történhet:

```
PMC->PMC_PCDR0 |= PMC_PCDR0_PID8;
```

4.1.11 Az IO-lábak funkcióinak multiplexálása

A SAM3 mikrovezérlő IO lábai több funkciót osztoznak, így egy adott IO-láb funkcionalitása lehet általános célú IO (GPIO), ekkor a láb a párhuzamos IO vezérlőhöz (PIO) van hozzárendelve, vagy a periféria multiplexeren keresztül csatlakozhat valamelyik interfészhez.

A mikrovezérlő a RESET utáni alapbeállításaként az IO lábak a párhuzamos IO vezérlőhöz (PIO) vannak hozzárendelve, ezért a multiplexeren keresztül elérhető funkciókhoz előbb át kell őket állítani az adott interfész multiplexerre.

A PB13-as láb átállítása az interfész multiplexerre:

```
PIOB->PIO_PDR|= PIO_PB13;
```

A PB13-as láb visszaállítása a párhuzamos IO vezérlőhöz:

```
PIOB->PIO_PER|= PIO_PB13;
```

A multiplexeren keresztül, minden IO láb 4 különböző belső funkcióra konfigurálható. A négy belső periféria A-tól D-ig terjedő betűkkel van jelölve, így a RESET utáni alapállapot az „A” interfész kiválasztását jelenti, mivel egy IO láb periféria multiplexerének beállításához két bitre van szükség, ezért a

beállítást két külön regiszteren is el kell végezni. Ez a két regiszter a PIO_ABCDSR[0] és a PIO_ABCDSR[1]. A két regiszter adott számú bitjei, végzik a megyegyező számú port láb multiplexerének beállítását.

Az IO lábak ABCD funkciói megtalálhatóak az 1. számú függelékben.

A PIO_ABCDSR regiszterek igazságtáblája:

PIO_ABCDSR[1]	PIO_ABCDSR[0]	A kiválasztott periféria
0	0	A
0	1	B
1	0	C
1	1	D

A következő példában a PB13-as port lábhoz tartozó mindkét multiplexer bit 1-be van állítva, így a D periféria kerül kiválasztásra.

PIOB->PIO_ABCDSR[0] |= PIO_PB13;

PIOB->PIO_ABCDSR[1] |= PIO_PB13;

Az ABCDSR regiszterek funkcióját a 12. ábra szemlélteti.

4.1.12 Általános célú IO (GPIO) kezelés

Az előző fejezetben már említésre került, hogy a SAM3 mikrovezérlő IO lábai kezelhetőek általános célú IO-ként, vagy valamelyik interfész speciális funkciójú IO lábaként.

Az általános célú IO lábak kezelése egy úgynevezett PIO (Parallel Input Output) kontrolleren keresztül történik. A SAM3nak három darab 32-bites IO portja van, amelyeket A, B és C portként különböztik meg. Mind három porthoz tartozik egy PIO kontroller, a PIO kontrollerek regisztereit a PIOA, PIOB és PIOC struktúrákon keresztül lehet kezelní. Ezeken a PIO struktúrákon keresztül tudjuk elvégezni az IO lábak konfigurálását, kiolvasását, beállítását stb.

A SAM3, mint bármely belső perifériája esetén, a PIO kontrollereknél is érvényes az, hogy az adott egység bekapcsolásához órajelet inicializálni kell! A belső perifériák órajelét a Power Management Controller PMC regiszterein keresztül lehet kezelní.

A PIOA vezérlő órajelének engedélyezése:

PMC->PMC_PCER0 |= PMC_PCER0_PID11;

A PIOA vezérlő órajelének engedélyezése után elkezdhetjük a PORT A lábának konfigurálását, a konfigurálás során érdemes figyelembe venni, hogy a hardver RESET után milyen alapértelmezett beállítások találkozhatunk, így csak azoknak a regisztereknek a beállítása szükséges, amelyeknél az alapállapot nem felel meg az igényeinknek.

A hardver RESET-et követően a portok meghajtása a PIO kontollerhez van hozzárendelve, viszont kimeneti meghajtó nincs engedélyezve, ezért a PIO vezérlő nem hajtja meg a kimeneteket. Az IO lábakhoz tartozó felhúzó ellenállás be van kapcsolva, így a lábakon tápfeszültséghez közeli érték mérhető.

4.1.12.1 IO láb konfigurálása és használata kimenetként

Abban az esetben, ha egy IO lábat szeretnénk kimenetként használni, engedélyeznünk kell a kimenet meghajtását.

Ez a port "A" (PIOA) 25-ös láb (PA25) esetében a következőképp történik:

PIOA->PIO_OER |= PIO_PA25;

A kimenetként való használatnál célszerű kikapcsolni az IO lábhoz tartozó felhúzó ellenállást.

A PA25-ös láb felhúzó ellenállásának kikapcsolása.

PIOA->PIO_PUDR |= PIO_PA25;

Ezek után beállíthatjuk a kimenet állapotát.

A PA25-ös kimenet magas szintbe állítása.

PIOA->PIO_SODR |= PIO_PA25;

A PA25 alacsony szintbe állítása.

PIOA->PIO_CODR |= PIO_PA25;

A PA25-ös kimenetre a SAM3N-EK fejlesztői panel sárga LED-je van csatlakoztatva. A LED a port láb és a tápfeszültség közé lett bekötve, ezért a kimenet alacsony szintje esetén világít.

4.1.12.2 IO láb konfigurálása és használata bemenetként

Az IO lábak állapota a kimeneti meghajtó konfigurálásától függetlenül bármikor visszaolvasható. Így SAM3 esetén csak értelmezés kérdése, hogy egy IO lábat mikor tekintjük bemenetnek.

A bemenetre kapcsolódó nyomógombok állapotának olvasásakor fontos szempont a bemeneti jelek glitch- és pergésmentesítése (prellmentesítés). Erre a célra a SAM3 rendelkezik egy programozható glitch- és prellmentesítő egységgel.

Glich és pergésmentesítő egység

A glich és pergésmentesítő egység engedélyezése:

PIOA->PIO_IFER |= PIO_PA15;

A pergésmentesítő üzemmód beállítása:

PIOA->PIO_IFSCER |= PIO_PA15;

Pergésmentesítő üzemmódban a szűrés a lassú órajel forrás SCLK alkalmazásával történik, kiegészülve egy órajel osztóval.

Az órajel osztó beállítása

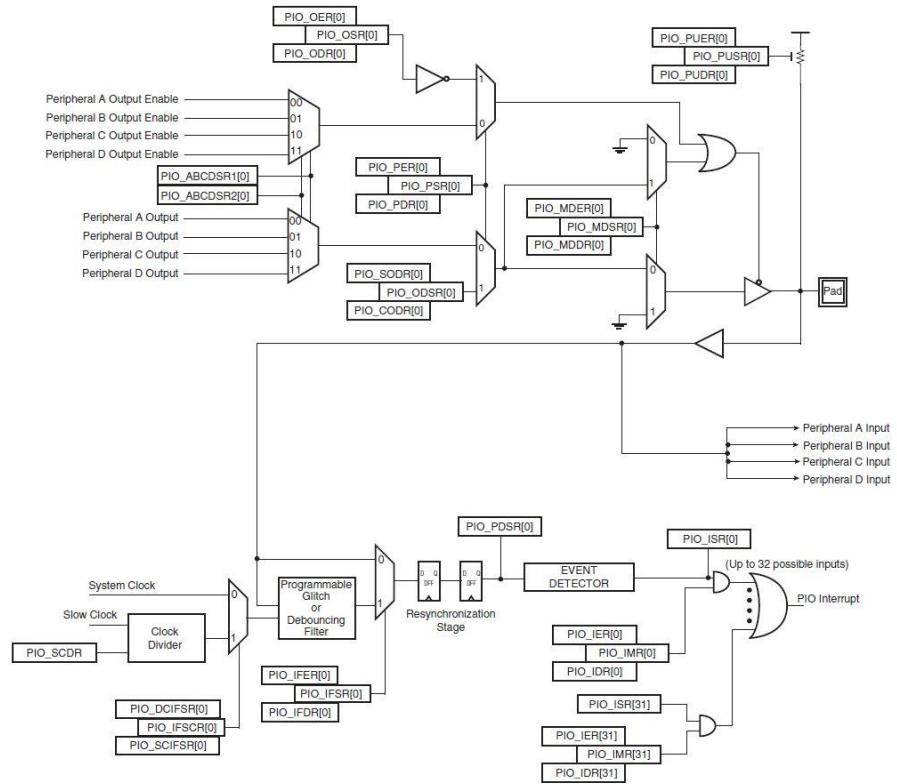
PIOA->PIO_SCDR = 200;

A fenti megoldás a leosztott órajel periódusidejének felénél rövidebb időtartamú impulzusok kiszűrésére alkalmas.

$T_{div_SCLK} = ((Osztó+1)*2)*T_{SCLK}$

Ahol T_{div_SCLK} a leosztott órajel periódusideje.

A port lábak állapota kiolvasható a porthoz tartozó PIO struktúra PIO_PDSR regiszteréből.



12. ábra a SAM3N IO lábának vezérlő logikája

Ha csak egy port láb állapotára vagyunk kíváncsiak, akkor a regiszter adott bitjeit maszkolni kell.

Ha az IO láb felhúzó ellenállása be van kapcsolva, és a rákötött nyomógomb záróérintkező, így a port lábat a földre (GND) húzza le.

Ezek alapján egy nyomógomb állapotának kiolvasása és kezelése a következőképp történhet:

```
if ((PIOA->PIO_PDSR)&PIO_PA15)
{
    //a port láb állapota magas, a gomb nincs lenyomva.
}
else
{
    //a port láb állapota alacsony, a gomb le lett nyomva.
}
```

4.1.12.3 Példaprogram IO kezelésre (LED-ek és nyomógombok)

```
// A SAM3 mikrovezérlő headher fájljának includeolása
#include "sam.h"
// A mikrovezérlő órajelének megadása
#define BOARD_MCK 48000000

int main (void)
{
    // A mikrovezérlő órajelgenerátorának és egyéb paramétereinek
    //beállítása
    SystemInit();

    // A Watchdog timer letiltása
    WDT->WDT_MR = WDT_MR_WDDIS;

    // A PIOA vezérlő órajelének engedélyezése
    PMC->PMC_PCER0 |= PMC_PCER0_PID11;

    // PA25 konfigurálása kimenetnek (sárga LED)
    // A PA25 kimeneti meghajtójának engedélyezése
    PIOA->PIO_OER |= PIO_PA25;
    // A PA25 felhúzóellenállásának kikapcsolása
    PIOA->PIO_PUDR |= PIO_PA25;
```

```

// PA23 konfigurálása kimenetnek (kék LED)
PIOA->PIO_OER |= PIO_PA23;
PIOA->PIO_PUDR |= PIO_PA23;

// PA15 konfigurálása bemenetnek (PB2 gomb)
// glich és prell mentesítés engedélyezése
PIOA->PIO_IFER |= PIO_PA15;
// prell mentesítés üzemmód bekapcsolása
PIOA->PIO_IFSCER |= PIO_PA15;

// PA16 konfigurálása bemenetnek (PB3 gomb)
PIOA->PIO_IFER |= PIO_PA16;
PIOA->PIO_IFSCER |= PIO_PA16;

// prell mentesítő órajelosztójának beállítása
// nagyobb érték, jobb prellmentesítés, de lasabb és kevésbé érzékeny
//bemenetet eredményez
PIOA->PIO_SCDR = 200;

while(1)
{
if(PIOA->PIO_PDSR&PIO_PA15) // PA15 bemenet állapotának
    //lekérdezése (PB2 gomb állapota)
{
    // a bemenet magas, a gomb nincs lenyomva
}
}

```

```

// a PIO_PA23 kimenent magas szintbe állítása (a kék LED nem
világít)
    PIOA->PIO_SODR |= PIO_PA23;
}

else
{
    // a bemenet alacsony, a gomb le van nyomva
    // a PIO_PA23 kimenent alacsony szintbe állítása (a kék LED
világít)

    PIOA->PIO_CODR |= PIO_PA23;
}

// PA16 bemenet állapotának lekérdezése (PB3 gomb állapota)
if(PIOA->PIO_PDSR&PIO_PA16) {
    // a bemenet magas, a gomb nincs lenyomva
    // a PIO_PA23 kimenent magas szintbe állítása (a sárga LED nem világít)

    PIOA->PIO_SODR |= PIO_PA25;
}

else
{
    // a bemenet alacsony, a gomb le van nyomva
    // a PIO_PA23 kimenent alacsony szintbe állítása (a sárga LED világít)

    PIOA->PIO_CODR |= PIO_PA25;
}

}
}

```

A program lefordítása és a mikrovezérlőre felprogramozása után, a BP2 nyomógomb megnyomásának hatására a kék LED, a BP3 hatására pedig a sárga LED fog világítani.

4.1.12.4 A bemeneti jelek megszakítás alapú kezelése

A SAM3 mikrovezérlő esetén lehetőség van a bemeneti jelváltozások (alacsony és magas szint, fel és lefutó él) megszakítás alapú kezelésére.

```
int main (void)
{
    SystemInit();
    WDT->WDT_MR = WDT_MR_WDDIS;

    PMC->PMC_PCER0 |= PMC_PCER0_PID11;

    PIOA->PIO_OER |= PIO_PA25; // sárga LED
    PIOA->PIO_PUDR |= PIO_PA25;
    PIOA->PIO_SODR |= PIO_PA25; // sárga LED kikapcsolása

    PIOA->PIO_OER |= PIO_PA23; // kék LED
    PIOA->PIO_PUDR |= PIO_PA23;

    PIOA->PIO_IFER |= PIO_PA15; // BP2 gomb
    PIOA->PIO_IFSCER |= PIO_PA15;

    PIOA->PIO_IFER |= PIO_PA16; // PB3 gomb
```

```

PIOA->PIO_IFSCER |= PIO_PA16;

PIOA->PIO_SCDR = 200;

// megszakítás beállítása (bemeneti jel változás)

// Kiegészítő megszakítási üzemmódok engedélyezése
PIOA->PIO_AIMER |= PIO_PA15;
// Él detektálás üzemmód
PIOA->PIO_ESR |= PIO_PA15;
// Lefutó él detektálás
PIOA->PIO_FELLSR |= PIO_PA15;

// Az IO lábhoz tartozó megszakítások engedélyezése
PIOA->PIO_IER |= PIO_PA15;

// PIOA megszakításainak engedélyezése
NVIC_EnableIRQ(PIOA_IRQn);
// Globális megszakítás engedélyezés
__enable_irq();

while(1);
}

```

```

// A PIOA vezérlő megszakításait kezelő handler
void PIOA_Handler(void)
{
    // megszakítások státusz regiszterének kiolvasása (kiolvasáskor törlődik
// a megszakítás flag)
    int us_status = PIOA->PIO_ISR;

    // Kék LED állapotának az ellenkezőre váltása
    if(PIOA->PIO_PDSR&PIO_PA23){
        PIOA->PIO_CODR |= PIO_PA23;}
    else{
        PIOA->PIO_SODR |= PIO_PA23;

    }
}

```

A PIOA handler már előzőleg a pio_handler.c fájlban definiálásra került, ezért a forító hibát jelez, így az eredeti definíciót meg kell keresni a refactoring menü használatával, és a handlert ki kell kommentezni.

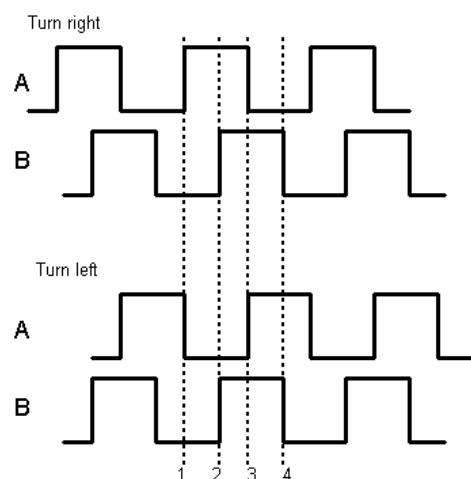
A sikeres fordítás és felprogramozás után a PB2 nyomógomb megnyomására (lefutó él) a kék LED ki-be kapcsol.

4.1.12.5 Rotary encoder kezelése

A következő program egy rotary encoder kezelésére mutat példát. Az enkóder bekötésétől és a forgatás irányától függően a kék vagy a sárga ledet kapcsolja lépéseként ki és be.

A forgókapcsolóként működő rotary encodernek három kivezetése van amelyeket A, B és C vel jelölnek. Az enkóder elfordulása lépésekben történik. Az enkóder minden lépésnél az A és a C érintkezőjét a közös pontként funkcionáló B érintkezőhöz zárja, forgásiránytól függő sorrendben. Az A és a C kivezetést a mikrovezérlő bemeneteire, a B kivezetést pedig a földpontra kötve, az enkóder elfordulásakor lefutó éleket generál a mikrovezérlő bemenetein. Az egyik lefutó él megszakítási eseményként felhasználható detektálva az elfordulást, a megszakítás handlerén belül a másik bemenet állapotát lekérdezve meghatározható az elfordulás iránya.

Az lefordulás irányától függ az enkóder A és B jeleinek fázisa, melyet az alábbi ábra szemléltet.



13.ábra A és B jelek fázisa az elfordulás irányának függvényében.

A PA26 bemeneten megjelenő lefutó él hatására megszakítási esemény jön létre, és a megszakítás handlerében a PA27-es láb állapotának megfelelően történik a forgásirány megkülönböztetése.

```
#include "sam.h"
#define BOARD_MCK 48000000 // CPU clock freq - 48MHz

void pio_config_out(Pio *port,unsigned int mask)
{
    port->PIO_OER |= mask;
    port->PIO_PER |= mask;
    port->PIO_PUDR|= mask;
}

void pio_config_in(Pio *port,unsigned int mask)
{
    port->PIO_IFER |= mask;
    port->PIO_IFSCER |= mask;
    port->PIO_SCDR = 20;
}

void init_io(void){
    //pio órajel adás (p34 p354)
    PMC->PMC_PCER0 |= PMC_PCER0_PID11;      //PIOA p34 p354
    //led config
    pio_config_out(PIOA,PIO_PA23);
    pio_config_out(PIOA,PIO_PA25);
```

```

//led kikapcsolás (active low)
PIOA->PIO_SODR |= PIO_PA25;           //Orange (Amber) led D3
PIOA->PIO_SODR |= PIO_PA23;           //Blue led D1

#define blue_toggle      if(PIOA->PIO_PDSR&PIO_PA23){PIOA-
>PIO_CODR |= PIO_PA23;}else{PIOA->PIO_SODR |= PIO_PA23;}
#define yellow_toggle   if(PIOA->PIO_PDSR&PIO_PA25){PIOA-
>PIO_CODR |= PIO_PA25;}else{PIOA->PIO_SODR |= PIO_PA25;}
}

void PIOA_Handler(void)
{
    int us_status = PIOA->PIO_ISR; // státusz regiszter olvasása

    // forgásirány megkülönböztetése
    if(!(PIOA->PIO_PDSR & PIO_PA27)) // ha a PA27 bemenet magas
//állapotban van
    {
        yellow_toggle
    }
    else // ha a PA27 bemenet alacsony állapotban van
    {
        blue_toggle
    }
}

void init_rotary()
{
    // Bemenetek beállítása

```

```

PIOA->PIO_IFER |= PIO_PA26;
PIOA->PIO_IFSCER |= PIO_PA26;
PIOA->PIO_SCDR = 20;

PIOA->PIO_IFER |= PIO_PA27;
PIOA->PIO_IFSCER |= PIO_PA27;

// Megszakítások beállítása
PIOA->PIO_AIMER |= PIO_PA26;
PIOA->PIO_ESR |= PIO_PA26;
PIOA->PIO_FELLSR |= PIO_PA26;
PIOA->PIO_IER |= PIO_PA26;
// Megaszkítások engedélyezése
NVIC_EnableIRQ(PIOA_IRQn);
__enable_irq();
}

void init(void)
{
    SystemInit();
    WDT->WDT_MR = WDT_MR_WDDIS;
    //inicializálja a panel gombjait, ledjeit
    init_io();
    //inicializálja a rotary encoderhez tartozó IO-kat és megszakítást.
    init_rotary();
}

```

```
int main(void)
{
    init();
    while(1);
}
```

4.1.13 Késleltetés, időzítés

4.1.13.1 A SAM3 valós idejű időzítője (RTT)

A SAM3 mikrovezérlő rendelkezik egy valós idejű időzítővel, amelyre az adatlap Real-time Timer (RTT) néven hivatkozik. Az RTT egy lassabb, számlálási, időzítési vagy késleltetési feladatokra felhasználható egység, amely alkalmas megszakítások generálására.

4.1.13.2 Az RTT számlálója és órajel forrása

Az RTT egység rendelkezik egy 32 bites számlálóval, melyet a mikrovezérlő SCLK-val jelölt, lassú órajel forrásról lehet meghajtani. Az SCLK órajel forrás frekvenciáját egy 32768 Herz-es külső óra kvarc határozza meg. A számlálót léptető órajel leosztható egy szoftveresen konfigurálható 16 bites lineáris előosztón keresztül. A számlálónak létezik egy RTT_VR nevű regisztere, amelyben a számláló aktuális értéke jelenik meg, és létezik egy RTT_AR nevű regisztere, amelynek az értékét szoftveresen lehet beállítani.

4.1.13.3 Az RTT egység megszakításai

Az RTT egység kétféle megszakítási eseményt képes generálni. Az egyik a számlálót léptető órajellel kapcsolatos, ez a megszakítási esemény a számláló minden egyes lépésekor létrejön. A másik az RTT_VR és az RTT_AR regiszterek értékének összehasonlításához kapcsolódik, ha a két regiszter értéke megegyezik, az egy újabb megszakítási eseményt hoz létre az az Alarm megszakítás.

4.1.13.4 Az RTT egység regiszterei és funkciói

Az RTT egység regiszterei az RTT nevű struktúra elemeiként kezelhetőek struktúrapointerek segítségével.

RTT_MR (RTT mode) regiszter

Az előosztás beállítása

Az RTT_MR regiszter RTPRES nevű bitmezőjével az előosztó osztását lehet beállítani. A regiszter nullás értéke esetén, az előosztóra a 16 bites bitmező maximális értéke kerül. Nullától különböző érték esetén pedig az órajel osztást az SCLK/RTPRES összefüggés alapján lehet meghatározni.

A megszakítások engedélyezése

Az RTT_MR regiszter ALMIEN nevű bitje a számláló, és az RTT_AR regiszter értékének egyenlőségéhez kapcsolódó megszakítást engedélyezi, ha a bit értéke 1 a megszakítás engedélyezett.

Az RTTINCIEN bit a számláló léptetéséhez kapcsolódó megszakítást engedélyezi, ha a bit értéke 1 a megszakítás engedélyezett.

Az RTT egység újraindítása

Az RTTRST bit 1-re állításával elvégezhető az RTT egység újraindítása. Az újraindítás során betöltődik az órajel-osztó értéke, és a 32 bites számláló nullázódik.

Az RTT_AR nevű, 32-bites regiszterben, a megszakítási esemény kiváltásához szükséges, számláló értékével összehasonlításra kerülő határérték állítható be.

Az RTT_VR nevű regiszter a számláló pillanat értékét tárolja.

Az RTT_SR nevű regiszterben a megszakítási események bekövetkezését jelző bitek helyezkednek el. Az ALMS jelzésű bit 1-es állapota az összehasonlítás és egyenlőséghez kapcsolódó megszakítás (az Alarm), az RTTINC bit pedig a számláló léptetéséhez kapcsolódó megszakítási esemény bekövetkezését jelzi.

4.1.13.5 Példa az RTT egység konfigurálására és használatára

Az RTT egység konfigurálása

```
// a prescaler-ben az előosztás értékét, az alarm változóban pedig az alarm  
//megszakítás szintjét (értékét) lehet beállítani  
void init_RTT(unsigned int prescaler, unsigned int alarm)  
{  
    // előosztó beállítása 32768 = 0x8000 - 1 másodperc  
    RTT->RTT_MR = ((prescaler) | RTT_MR_RTTRST);  
    // várakozás egy timer tickre,hogy a beállítás megtörténjen  
    uint32_t ul_previous_time = RTT->RTT_VR;  
    while (ul_previous_time == RTT->RTT_VR);  
    // Az „alarm” érték beállítása (ha szükséges)  
    RTT->RTT_AR = 0;  
    RTT->RTT_AR |= RTT_AR_ALMV(alarm);  
    // A megszakítások beállítása  
    NVIC_DisableIRQ(RTT_IRQn); // RTT megszakítások letiltása  
    // Az esetlegesen függőben lévő megszakítás törlése  
    NVIC_ClearPendingIRQ(RTT_IRQn);  
    NVIC_SetPriority(RTT_IRQn, 0);  
    NVIC_EnableIRQ(RTT_IRQn); // RTT megszakítások engedélyezése  
    // számláló lépetetés megszakításának engedélyezése  
    RTT->RTT_MR |= RTT_MR_RTTINCIEN;  
    __enable_irq(); // globális megszakítás engedélyezés  
}
```

Az RTT egység a konfigurálást követően a beállításnak megfelelő időközönként megszakításokat generál.

Az RTT egység megszakításainak kezelése

```
void RTT_Handler(void)
{
    int ul_status = RTT->RTT_SR;

    // ha a számláló inkrementálódik
    if((ul_status & RTT_SR_RTTINC) == RTT_SR_RTTINC)
    {
        // . . . (az elvégzendő feladatok)
    }

    // ha a számláló eléri az alarm értéket
    if ((ul_status & RTT_SR_ALMS) == RTT_SR_ALMS)
    {
        // . . . (elvégzendő feladatok)
        // újra indítja a számlálást nulláról
        init_RTT(0x1000, 4);
    }
}
```

Az alarm megszakításra önmagában egy egyszerűbb időzítés vagy késleltetés esetén nincs szükség, így a programból kihagyható, leegyszerűsítve a forráskódot.

```

void init_RTT(unsigned int prescaler, unsigned int alarm)
{
    // előosztó beállítása 32768 = 0x8000 - 1 másodperc
    RTT->RTT_MR = ((prescaler) | RTT_MR_RTTRST);
    // várakozás egy timer tickre,hogy a beállítás megtörténjen
    uint32_t ul_previous_time = RTT->RTT_VR;
    while (ul_previous_time == RTT->RTT_VR);
    // A megszakítások beállítása
    NVIC_DisableIRQ(RTT_IRQn); // RTT megszakítások letiltása
    // Az esetlegesen függőben lévő megszakítás törlése
    NVIC_ClearPendingIRQ(RTT_IRQn);
    NVIC_EnableIRQ(RTT_IRQn); // RTT megszakítások
    //engedélyezése
    // számláló lépetetés megszakításának engedélyezése
    RTT->RTT_MR |= RTT_MR_RTINCEN;
    __enable_irq(); // globális megszakítás engedélyezés
}

```

Ekkor a megszakításkezelésnél a két különböző megszakítás megkülönböztetése elhagyható.

```

void RTT_Handler(void)
{
    // az RTT_SR olvasására a regiszter nullázása miatt van szükség
    int ul_status = RTT->RTT_SR;
    // (ide kerülnek az elvégzendő feladatok)
}

```

4.1.13.6 A SAM3 UART kontrollere

IO lábak beállítása az UART használatához

A SAM3 mikrovezérlők soros kommunikáció megvalósítására, rendelkeznek két egymástól függetlenül paraméterezhető UART vezérlővel, amelyeket az adatlap a 0-ás és az 1-es számmal különböztet meg. A nullás számú UART0 vezérlő adatküldésre szolgáló lábat a TXD0, az adatfogadás céljára kijelölt lábat az RXD0 jelöléssel látták el, hasonlóképp történt ez az UART1 vezérlő esetében is, ahol a hozzáartozó adatküldés a TXD1, az adatfogadás pedig RXD1 jelölésű lábakon valósul meg. Az adatküldés és fogadás funkciókat az interfész multiplexer ABCDSR regisztereinek megfelelő beállításával az IO lábakhoz kell társítani, továbbá engedélyezni kell a belső interfészfunkciót az érintett IO lábaknál.

A baud rate meghatározása

Az UART vezérlők rendelkeznek az adatküldés célját szolgáló (TX) és az adatfogadást megvalósító (RX) egységekkel, ezek az egységek közös órajel forrásról, de eltérő órajelről üzemelnek. Az UART átviteli sebességét (Baud Rate) meghatározó órajel a mikrovezérlő fő órajelének (MCK) leosztásával állítható elő, az UART_BRGR regiszter CD bitmezőjében beállított 16 bites osztó segítségével. A küldő egységet üzemeltető (és az átviteli sebességet is meghatározó) órajel a következő összefüggés alapján számítható $MCK/(16*CD)$.

A képletben szereplő 16-szoros szorzó jelentősége ott kerül előtérbe, hogy az adatfogadó oldal a küldő órajelének 16-szorosával (MCK/CD) üzemel.

Erre a START bit lefutó élének megfelelően pontos detektálása miatt van szükség, valamint azért is, hogy az adatbitek mintavételezésénél a mintavételi időpont minél pontosabban az adatbit közepébe essen. Az adatbitek mintavételezésének időpontját azért célszerű minél inkább a bit közepéhez igazítani, mert aszinkron adatátvitelről lévén szó, a küldő és a fogadó oldal órajel forrásai különbözök, így az órajel frekvenciák is bizonyos mértékben eltérnek. Ráadásul a valós órajel nem is egyezik meg teljesen a névlegessel (ami órajel osztás során adódó maradékból, és az órajel források pontatlanságából is adódik), és ez mindenképp okoz némi eltérést a két oldal között. Ez az eltérés pedig a mintavételi időpont csúszását vonja maga után a fogadott jelhez képest. A mintavételi időpont elcsúszhat pozitív vagy negatív irányban is. A hibátlan adatfogadás érdekében fontos azt biztosítanunk, hogy a mintavételi időpont ne csússzon ki a mintavételezendő bit időintervallumából az adatkeret végéig. Így a mintavételezendő bitek közepének megcélzásával tudjuk a legnagyobb toleranciát biztosítani az eltérő frekvenciákkal szemben. Az órajel osztás képletéből a maximális baud rate MCK/16, a minimális pedig MCK/(16*65536)-nak adódik.

Az adatkeret

Az adatkeret egy alacsony szintű START bittel kezdődik, a START bitet az adatbitek követik. Az UART küldés és fogadás adatbitjeinek száma fixen nyolc. Az adatkeret végét egy STOP bit zárja. Az adatbitek és a STOP bit közé, konfigurációtól függően beékelődhet még egy paritásbit is.

A paritásbit

A paritásbitet az UART_MR regiszter PAR nevű bitmezőjében állíthatjuk be, a paritás bit lehet EVEN, ODD, SPACE vagy MARK típusú. A paritásbit SPACE beállítás esetén fixen nulla értéket vesz fel, ha MARK beállítást használunk, akkor pedig fixen 1 értéket. EVEN beállítással a paritásbit 1 értéket vesz fel, ha az adatbitek között páratlan számmal szerepel az 1-es bit, és nullát vesz fel, ha páros számmal szerepel 1-es. ODD beállítás esetén a paritásbit 1 értéket vesz fel, ha páros számú adatbit szerepel, és nullát vesz fel, ha páratlan számú 1-es.

Az RX, TX egységek engedélyezése, tiltása, resztelelése

A hardver RESET után az UART küldést és fogadást végző TX és RX egységei alapértelmezetten le vannak tiltva. Használatukhoz ezeket először engedélyezni kell. Az RX és TX engedélyezését az UART_CR regiszter TXEN és RXEN nevű bitjein keresztül lehet elvégezni, az adott bit 1-re maszkolásával. Természetesen lehetőség van az egységek letiltására is a TXDIS és RXDIS bitek segítségével. A TX és RX egységek a hardver RESET-nek megfelelő állapotba hozhatóak az RSTTX és RSTRX bitek 1-be állításával, ebben az esetben a TX és az RX működésüket azonnal félbeszakítják, az esetleges adatküldés folyamata vagy az éppen feldolgozás alatt lévő bejövő adat fogadása félbeszakad, az egységek letiltott állapotba kerülnek.

4.1.13.7 Az UART-hoz kapcsolódó státuszbitek (küldés és fogadás)

UART_SR (Status Register) tartalmazza a küldés és fogadás szempontjából nélkülözhetetlen státuszbiteket.

Adat küldés

Az adatküldés megkezdése előtt tájékozódnunk kell arról, hogy a küldő egység készen áll-e a küldendő adat fogadására. A küldő egység állapotát az UART_SR regiszter TXRDY nevű bitjének kiolvasásával ellenőrizhetjük, amennyiben a bit állapota 1, a küldő egység készen áll, ekkor a küldendő adatot az UART_THR (UART Transmit Holding Register) -be tehetjük, ezzel az adat küldése azonnal el is kezdődik. Ha az TXRDY bit állapota nulla, a küldő egység nem áll készen, ez lehet, azért mert az előző adat(ok) küldése még nem fejeződött be, vagy azért mert a küldő egység nincs engedélyezve.

Adat fogadás

Az adatfogadó egység állapotáról az UART_SR regiszter RXRDY bitjének kiolvasásával tájékozódhatunk. A bit 1 es állapota azt jelzi, hogy a fogadó egység (RX) által legalább egy adat (byte) fogadása megtörtént, és az adat készen áll a kiolvasásra. Ekkor a fogadott adatot az azt tároló UART_RHR regiszterből (UART Receiver Holding Register) kiolvashatjuk. RXRDY bit nulla állapota azt jelzi, az utolsó (UART_RHR) adatkiolvasás óta nem érkezett újabb adat, vagy a fogadó egység (RX) le van tiltva.

Hibás adatok detektálása

UART_SR (Status Register) megfelelő bitjeinek kiolvasásával visszajelzést nyerhetünk. Abban az esetben, ha hibás adatfogadás történt, vagy ha a fogadott adat kiolvasását elmulasztottuk, és a korábban fogadott adat felülíródott a következő adattal. A FRAME-bit kiolvasásával információt nyerhetünk arról, ha a fogadó egység hibás adatkeretet detektált (hiányzó stopbit). A bit 1-es értéke jelzi a hibás adatkeret detektálást. A PARE bit 1-es

állapota paritáshiba észlelését jelzi vissza. A hibadetektálás során az 1-es állapotba billent jelzőbitek állapotukat megtartják attól függetlenül, hogy a hibás után hibátlan adatfogadás is történt-e vagy sem. Az OVRE bit a fogadott adat túlfutását jelzi. Ha a bit állapota 1, valamely fogadott adat kiolvasását elmulasztottuk és az felülíródott egy újabb adatfogadás értékével. A hibadetektálást jelző PARE, FRAME és OVRE bitek természetesen lenullázódnak hardver RESET során, vagy lenullázhatóak az UART_CR regiszter RSTSTA nevű bitjének 1-re maszkolásával.

4.1.14 Megszakítások

Az UART interfész képes megszakítási események generálására is. Ezek a megszakítások kapcsolódhatnak az adatküldéséhez vagy -fogadáshoz, hibás adat detektálásához, vagy a közvetlen memória hozzáféréssel (PDC) kapcsolatos műveletekhez. Az adott megszakítások külön regiszteren keresztül engedélyezhetők vagy tilthatóak, lekérdezhető ennek állapota, vagy a megszakítási esemény bekövetkezésének ténye.

4.1.15 Teszt üzemmódok

Az UART interfész konfigurálásánál lehetőség van különböző teszt üzemmódok kiválasztására, amelyek megkönnyítik a szoftverfejlesztés során vagy a hardver elkészítésekor előforduló esetleges hibák behatárolását. A teszt üzemmódok beállításának módja és működésük az UART_MR regiszter leírásánál látható bővebben.

4.1.16 Az UART kontroller regiszterei

Az SAM3 két UART kontrollerének regiszterei a hozzájuk tartozó struktúra elemeiként érhetőek el struktúrapointerek segítségével.

A nullás számmal jelzett UART egység regiszterei az UART0 struktúrán keresztül, az 1-es számú egység regiszterei értelemszerűen az UART1 nevű struktúrán keresztül érhetőek el.

UART_CR (UART Control Register)

Az UART_CR regiszter tartalmazza az adó (RX) és vevő (TX) egység engedélyezéséhez, letiltásához, reszeteléséhez szükséges biteket. Ezen kívül az adatátviteli hibadetektálást visszajelző bitek (PARE, FRAME, OVRE) nullázásáért felelős bitet.

A RSTRX 1-re állításával az RX egység reszetelését érhetjük el, a RSTTX 1-re állításával pedig a TX egység reszetelését valósíthatjuk meg. RXEN bit a RX egység engedélyezését, az RXDIS pedig letiltását végzi. Az előzőeknek megfelelően a TXEN bit 1-re állítása a TX egység engedélyezését, TXDIS pedig letiltását végzi.

Az RSTSTA bit 1-re állításával a hibadetektálást visszajelző (PARE, FRAME, OVRE) bitek törlését (nullázását) végezhetjük el.

UART_MR (UART Mode Register)

Az UART_MR regiszter PAR (Parity Type) nevű bitmezőjének segítségével bekapsolhatjuk a paritásbitet és megadhatjuk annak típusát. A PAR (három bites) bitmező 4-es értéke esetén nincs paritásbit beállítva. A 0-ás

érték esetén a paritás EVEN típusú, 1-es érték esetén a paritás ODD, kettes érték esetén SPACE, 3-as érték esetén pedig MARK típusú lesz.

Az UART_MR regiszter CHMOD (Channel Mode) bitmezője segítségével az egyes teszt üzemmódokat lehet beállítani. Nullás érték esetén az UART interfész NORMAL módban van.

A CHMOD bitmező 1-es értéke esetén AUTOMATIC üzemmódban, ebben az esetben a küldő egység (TX) le van kapcsolva a TXD lábról, az RXD lábon beérkezett adatok közvetlenül visszakerülnek a TXD lábra (mindamellett, hogy az RXD láb jelei eljutnak a fogadó egységbe is) ez az RXD TXD lábak közvetlen összekapcsolását jelenti.

A CHMOD bitmező 2-es értéke esetén az úgynevezett LOCAL_LOOPBACK üzemmódban van, ekkor mind a küldő, fogadó egység és a hozzájuk tartozó TXD, RXD lábak közötti kapcsolat ki van iktatva. A küldő egység kimenete és a fogadó egység bemenete a mikrovezérlőn belül össze van kapcsolva. Ez az üzemmód lehetővé teszi a küldő és fogadó egységek belső szoftveres tesztelését.

A CHMOD bitmező 3-as értéke esetén az interfész REMOTE_LOOPBACK üzemmódban van. A REMOTE_LOOPBACK üzemmódban mind a küldőegység és a hozzátartozó TXD láb közti kapcsolat, minden a fogadóegység és a hozzátartozó RXD láb közti kapcsolat ki van iktatva. A TXD és RXD lábak pedig az integrált áramkörön belül közvetlenül össze vannak kötve. Ez az üzemmód lehetővé teszi az RXD és TXD IO lábak és a hozzájuk kapcsolódó vezetékek tesztelését. Beleértve a lábakhoz kapcsolt külső vezetékek és az integrált áramkörön belüli huzalozás tesztelését is.

A megszakításokkal kapcsolatos regiszterek

Az UART interfész megszakításainak engedélyezése UART_IER (UART Interrupt Enable Register), a megszakítások tiltása az UART_IDR (UART Interrupt Disable Register) megfelelő bitjeinek 1-re állításával történhet. A megszakítások engedélyezett vagy tiltott állapotának kiolvasása az UART_IMR (UART Interrupt Mask Register) azonos helyű és elnevezésű bitjeinek kiolvasásával történik (ahol az 1-es bit azt jelenti, hogy az adott megszakítás engedélyezve van, a nullás pedig hogy tiltva). Az UART0 vagy UART1-el kapcsolatos megszakítási esemény kezelésekor a vezérlés az UART0_Handler vagy az UART1_Handler, megszakítás kiszolgáló rutinnak adódik át. A megszakítási eseményeket az UART_SR státusz regiszter kiolvasásával lehet megkülönböztetni. A megszakítás engedélyezés, letiltás, és maszk regiszter bitjeinek helye és elnevezése megegyezik az UART_SR regiszter bitjeivel, ezért azok ismertetésére csak az UART_SR regiszternél kerül sor.

UART_SR (UART Status Register)

Az adatküldő és -fogadó egységének állapota, valamint az átviteli hibadetektálással kapcsolatos információk az UART_SR (UART Status Register) -ból olvashatóak ki.

A RXRDY (Receiver Ready) bit az adatfogadó egység állapotát jelzi vissza, ha a bit 1, az UART_RHR tartalmaz fogadott, de még ki nem olvasott adatot. Ha az állapota nulla az UART_RHR nem tartalmaz fogadott, de még ki nem olvasott adatot vagy a fogadó egység nincs engedélyezve.

Az TXRDY (Transmitter Ready) nevű bit 1-es állapota esetén az UART_THR nem tartalmaz olyan adatot, ami még nem lett elküldve, tehát a küldőegység készen áll az adatküldésre. Ha a bit állapota nulla, az UART_THR be beírt adat küldése még nem fejeződött be, vagy a küldő egység nincs engedélyezve.

Az ENDRX, ENDTX és a TXBUFE, RXBUFF biteknek a PDC (közvetlen memória hozzáférés) alkalmazásánál van jelentősége.

Az OVRE (Overrun Error) bit 1-es állapota azt jelzi, hogy az utóbbi adatfogadás eredménye nem lett kiolvasva az UART_RHR (adatfogadás) regiszterből, és azt egy újabb adatfogadás felülírta.

A FRAME (Framing Error) bit 1-es állapota az adatfogadás során, hibás adatkeret detektálását jelzi vissza.

A PARE (Parity Error) bit 1-es állapota paritás hiba detektálását jelzi vissza az adatfogadás során.

A fenti adatátviteli hibákat visszajelző bitek értelmezéséhez fontos megjegyezni, hogy az adott hiba típushoz tartozó bit 1-esbe billenéséhez elég az adott típusból, egyetlen hibás adatfogadás is, ezután az adott bit megtartja 1-es állapotát egészen addig, amíg az egységet le nem reszetelik, vagy az UART_CR regiszter RSTSTA bitjével a hiba visszajelző biteket ki nem nullázzák.

A TXEMPTY (Transmitter Empty) bit 1-es állapota azt jelzi, hogy az UART_THR nem tartalmaz elküldetlen adatot, és a küldő egységen nincs folyamatban lévő adatküldés.

Adat fogadás regiszter UART_RHR (UART Receiver Holding Register)

Az UART_RHR nevű adatfogadás regiszter RXCHR-rel jelölt alsó nyolc bitje tárolja a fogadóegység által fogadott adatbájtot.

Adat küldés regiszter UART_THR (UART Transmit Holding Register)

Az UART_THR nevű adatküldő regiszter TXCHR-rel jelölt alsó nyolc bitje tárolja az adatküldő egység által küldendő adatot. Az UART interfész megfelelő konfigurációja és az adatküldő egység engedélyezése után, UART_THR - TXCHR bitmezőjébe történő adatírással az adatküldés azonnal megkezdődik.

Baud Rate beállító regiszter

Az UART_BRGR (UART Baud Rate Generator Register) CD-vel jelölt 16 bites bitmezője határozza meg az UART interfész küldő- és fogadóegységeit működtető leosztott órajeleket, és ezáltal az UART interfész adatátviteli sebességét (baud rate-jét). Az UART interfész adatátviteli sebessége a mikrovezérlő órajelének (MCK) és a CD-bitmezőben beállított érték ismeretében a következő összefüggés alapján határozható meg: $MCK/(16*CD)$.

4.1.17 Példa az UART konfigurálására

```
// Az UART-hoz tartozó IO lábak konfigurálása  
PIOA->PIO_PUDR |= PIO_PA10; // kimeneti felhúzó ellenállás  
//kikapcsolása  
PIOA->PIO_PDR |= PIO_PA10; // IO láb belső perifériára állítás  
  
// Az UART órajelének bekapcsolása a PMC egységen(34,324)  
PMC->PMC_PCER0 |= PMC_PCER0_PID8;  
  
// Baud rate beállítása  
UART0->UART_BRGR = (BOARD_MCK/(baud_rate*16)); //  
//BOARD_MCK = CPU órajele  
  
UART0->UART_MR |= 4 << 9; // paritás kikapcsolása ( 515.)  
  
// A küldő egység (transmitter) engedélyezése  
UART0->UART_CR |= UART_CR_TXEN;  
  
// Az adatfogadó egység (Receiver) engedélyezése  
UART0->UART_CR |= UART_CR_RXEN;  
  
// Megszakítások beállítása  
UART0->UART_IER |= UART_IER_RXRDY; //Megszakítás  
//engedélyezés adatfogadáskor
```

```
NVIC_EnableIRQ(UART0_IRQn); // UART0 megszakításainak  
//engedélyezése  
__enable_irq(); // Globális megszakítások engedélyezése
```

Példa adat küldésre

```
// Várakozás amíg a küldő egység készen nem áll az adatküldésre  
while(!((UART0->UART_SR)&(UART_SR_TXRDY)));
```

```
UART0->UART_THR = adat_kuld; // Adat küldés
```

Példa adat fogadása

```
// Várakozás amig a fogadott adat meg nem érkezik  
while(!((UART0->UART_SR)&(UART_SR_RXRDY)));
```

```
adat_fogad = UART0->UART_RHR; // fogadott adat kiolvasása
```

Megszakítás alapú adatfogadás

```
void UART0_Handler(void)  
{  
    unsigned int status;  
    status = UART0->UART_SR; //status regiszter kiolvasása  
    if(status&UART_SR_RXRDY) adat_fogad = UART0->UART_RHR;  
    //adat fogadás  
}
```

4.1.18 Külső perifériák illesztése

4.1.18.1 A SAM3N-EK és az EXPBRD kiegészítő panel illesztése

A SAM3N-EK fejlesztői panellel kapcsolatos oktatási anyag kidolgozásakor, felmerült az igény a panel további külső perifériákkal való bővítésére.

A laborokon használt AVR alapú fejlesztői panel a T-Bird tartozéka egy kiegészítőpanel az EXPBRD, amely további perifériákkal rendelkezik.

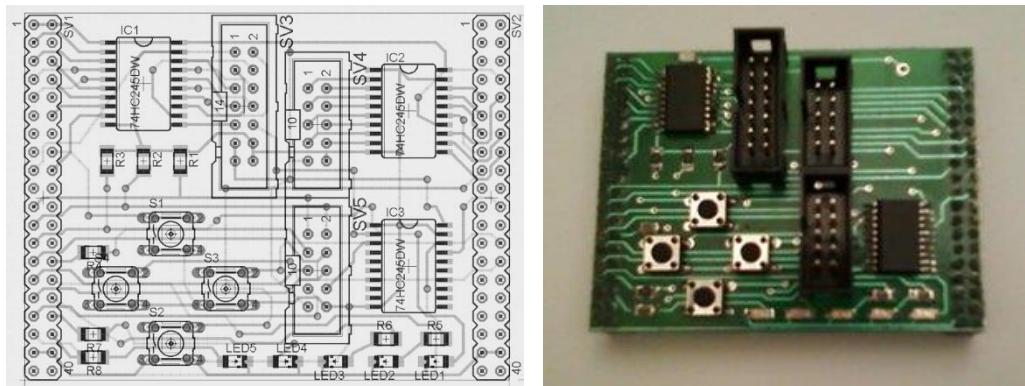
Az EXPBRD panelen található egy HD44780 típusú karakteres LCD kijelző, egy négy számjegyből álló hétszegmenses kijelző, egy 12 gombos mátrixbillentyűzet, ezen kívül egy RGB LED és egy LM35 típusú analóg hőszenzor is.

Praktikus lehetőségnek mutatkozott a SAM3N-EK-nak a már meglévő EXPBRD-el való összekötése. Elkészült a SAM3N-EK-hoz egy összekötő panel, ami lehetőséget nyújt a SAM3N-EK és az EXPBRD panelek összekötésére.

Az eszközök fizikai illesztése a SAM3N-EK GPIO expander tüskesorára tervezett összekötő panel formájában valósult meg. Az összekötő panel szalagkábel csatlakozón keresztül kapcsolódik az EXPBRD felé. A korlátozott befoglaló méret, a szabad port lábak korlátozott száma és a három szalagkábel csatlakozó megfelelő elhelyezése nagymértékben behatárolta a tervezési lehetőségeket, ráadásul szükség volt a logikai jelszintek illesztésére is, mivel a SAM3N 3.3V jelszintet használ, a T-Bird-höz tervezett EXPBRD pedig 5V-at.

A szabadon maradt port lábak kihasználására, az összekötőpanelen helyet kapott még négy darab nyomógomb és öt LED is.

A port kiosztás kialakításánál fontos tervezési szempont volt, hogy az összetartozó adatbitek lehetőleg egymás mellett legyenek (LCD data, hétszegmens data), az adott perifériához tartozó lábak minden porton legyenek, így egy periféria használatához csak egy portot kelljen inicializálni.

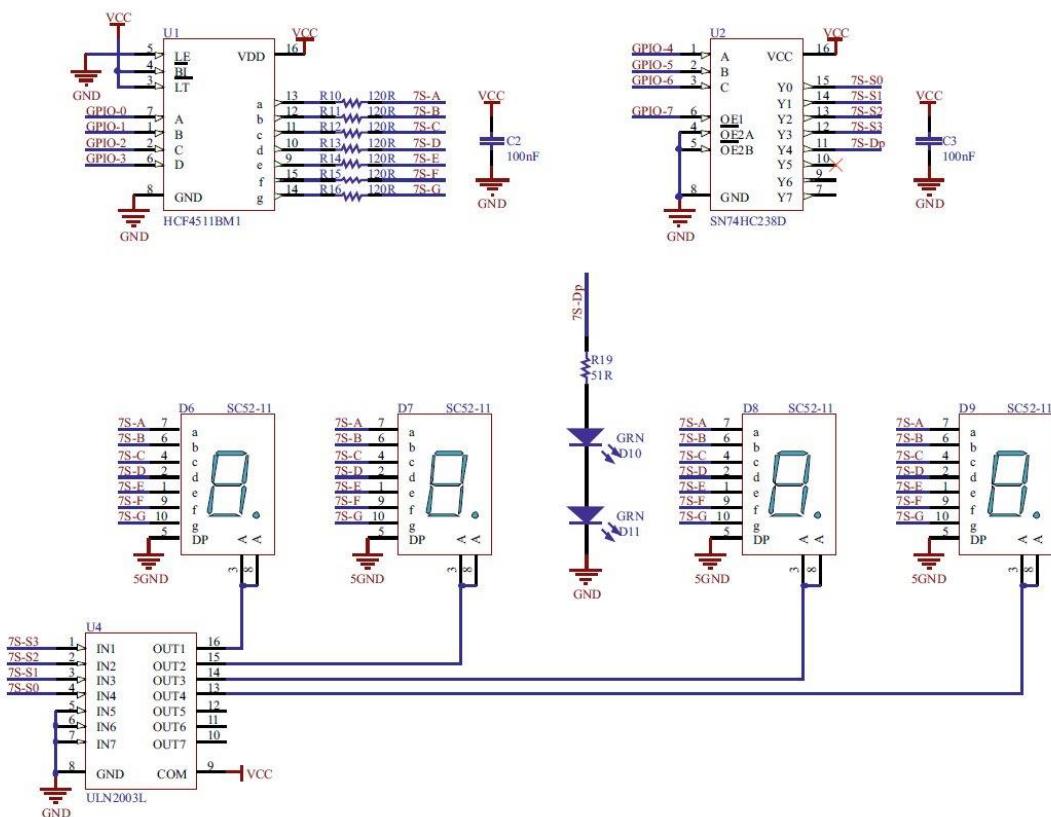


14. ábra összekötő panel

4.1.18.2 A SAM3N-EK és az EXPBRD kiegészítő panel használata

A hét szegmenses kijelző kezelése

Az EXPBRD hétszegmenses kijelzőjéhez két integrált áramkör tartozik. Egy HCF4511BM1 típusú BCD - hétszegmens dekóder a szegmensek vezérlésére, és egy SN74HC238D típusú multiplexer a számjegy kiválasztására. A szegmensek vezérlését ellátó vezetékek számjegyenként párhuzamosan vannak kapcsolva, a multiplexer működéséből adódóan egyszerre csak egy számjegyet működését lehet bekapcsolni ezért a kijelzőt a számjegyek egymás utáni gyors bekapcsolával (multiplex vezérlésével) lehet működtetni.



15. ábra a hét szegmenses kijelző lábkiosztása

A változóban tárolt számérték decimális számjegyekre bontása (BCD kódba alakítása) a következő függvényel tehető meg, ahol a num bemeneti paraméter tárolja a számértéket, és az *array pointer mutat arra a tömbre, ahova szétbontott számjegyek kerülnek.

```
void bcd(char * array, int num)
{
    char i=0;
    for(;num>0;num=num/10)
    {
        *array = num%10;
        *array++;
        i++;
    }
}
```

A hétszegmenses kijelző egyes számjegyeinek frissítésére a következő függvény nyújt megoldást, ahol a seg_data az adott szegmensre kikerülő számérték (BCD kód) a seg_num pedig a multiplexerrel kiválasztott szegmens száma (0-3).

```

void digit_frissit(unsigned char seg_data,unsigned char seg_num)
{
    // a hét szegmenseshrz tartozó kimenetek kikapcsolása
    PIOA->PIO_CODR|= (0xF << 2);
    PIOA->PIO_CODR|= (0x3 << 17);
    PIOA->PIO_CODR|= (0x3 << 20);

    //multiplexer beállítása (digit kiválasztás)
    PIOA->PIO_SODR|= ((seg_num&3)<< 17);
    PIOA->PIO_SODR|= (((seg_num&4)>> 2)<<20);

    // adat kiküldése a kiválasztott digitre
    PIOA->PIO_SODR|= (((seg_data)&(0xf)) << 2);

    //multiplexer engedélyezése
    PIOA->PIO_SODR|= (1<< 21);
}

A fenti függvény használata előtt szükséges az érintett IO lábak
kimenetként történő konfigurálása.

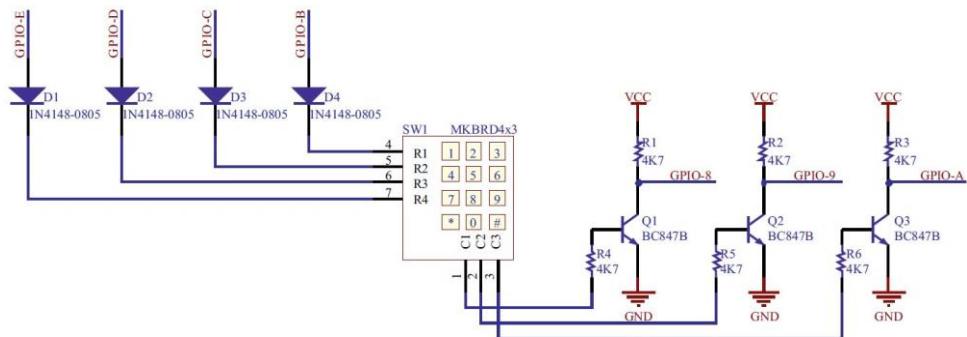
```

A mátrix billentyűzet kezelése

Az EXPBRD mátrix billentyűzete három oszlopból és négy sorból áll. A billentyűk kiolvasása soronként történik. Egy időben egy sor lehet bekapcsolva. Az sorok vezérlése a hozzájuk tartozó négy kimeneten valósítható meg. Az adott

sorokhoz tartozó gombok állapotának kiolvasása az oszlopokhoz tartozó három bemeneten történik. Ha a megvezérelt sorhoz tartozó valamelyik gomb le lett nyomva, a hozzá tartozó oszlophoz kapcsolt IO bemenet alacsony szintre kerül. Az alábbi áramköri megvalósítás megakadályozza, hogy hibás konfiguráció esetén zárlat alakulhasson ki, ami az adott IO láb meghibásodását okozná.

A sorok beállítása és az oszlopok kiolvasása közé szükséges némi késleltetést beiktatni, különben téves kiolvasás jöhet létre.



16. ábra A mátrixbillentyűzet lábkiosztása

4.1.19 A SAM3 mikrovezérlő ADC je

Az ATSAM3N sorozatú mikrovezérők mindegyike tartalmaz egy 10 bites felbontású 16 csatornás analóg digitális konvertert, amely az arra célra kijelölt analóg bemenetek feszültségértékeinek digitális átalakítását végzi.

4.1.20 Az ADC felépítése és funkciói

Az AD konverter felbontása alapvetően 10 bites, de lehetőség van arra, hogy azt 8 bites felbontással használjuk. Az A/D átalakítás sebességéért felelős leosztott órajel maximális értéke, így konverter működési sebességét is ez fogja

befolyásolni az A/D felbontás értéke, így az alacsonyabb, 8 bites felbontás kiválasztásával növelhető a mintavételi gyakoriság.

Az AD konverter egy multiplexer segítségével 16 bemeneti csatorna kezelésére alkalmas, természetesen az ADC egy időben csak egyetlen csatorna jelének feldolgozását végzi, és a 16 csatorna kezelése egymás után (vagy a konfiguráció által meghatározott sorrendben) történik.

Az AD konverzió tartománya a mikrokontroller GND lábának nulla szintjétől az ADVREF lábra kapcsolt külső referencia feszültség értékig terjed, 8 vagy 10 bites felbontásban.

Az átalakítás eredménye csatornánként külön regiszterben érhető el, ezenkívül az utolsó átalakítás eredménye megjelenik egy közös regiszterben is.

Az analóg digitális konverziót elindíthatja valamilyen külső trigger esemény az ADTRG jelzésű lábon, amennyiben szükséges a pontos mintavételi gyakoriság betartása az indítást történhet belső timer-counter segítségével, de az A/D konverter elindítható szoftveresen is az ADC Control Register START nevű bitjének 1-re állításával. Ezenkívül A/D konverter üzemelhet folytonosan is, ez az úgynévezett "free run" mód.

Az AD konverter lehetővé tesz automatikus komparálási funkciókat ("Comparison Window"), melyek alkalmasak megszakítási esemény kiváltására. Automatikus komparálás esetén az A/D konverzió eredménye összehasonlításra kerül az erre a célra fenntartott regiszter vagy regiszterek tartalmával, és konfigurációtól függően ellenőrzi, hogy az A/D konverter által szolgáltatott érték alatta vagy felette van-e a regiszterben megadott határértéknek, más beállítás estén megvizsgálja, hogy az A/D konverzió eredménye beleesik-e a regiszterekben beállított tartományba, vagy azon kívül helyezkedik el.

Az AD konverter hibátlan üzemeltetéséhez szükséges lehet bizonyos időzítések, késleltetések beállítására is. minden ADC rendelkezik egy minimális bekapcsolási idővel ("Startup Time"), és az ADC bemeneti csatornái közti átváltáskor jelentkező holtidővel ("Tracking Time"). A Startup Time-nak megfelelő késleltetés az ADC mode regiszter (ADC_MR) STARTUP-al jelzett mezőben, a Tracking Time pedig a TRACKTIM nevű mezőben állítható be.

Az ADC által szolgáltatott adatok bufferelése megoldható közvetlen memória hozzáféréssel, a "Peripheral DMA Controller" segítségével is. Továbbá lehetőség van regisztereinek írásvédelmére is, ezzel a megelőzhető, hogy egy esetleges szoftverhiba az ADC-t hibás működésre konfigurálja.

4.1.21 Az AD konverter regiszterei

Az ADC konfigurálásához és működtetéséhez szükséges regiszterek az ADC nevű struktúrában helyezkednek el, hozzáférésük struktúraponter segítségével biztosított.

Control Register

Az ADC_CR –el jelölt control regiszterben lehetőség van az ADC szoftveres reszelésére, illetve itt kapott helyet az A/D konverziót elindító, az ADC szoftveres triggerének számító START bit is. Az SWRST bit 1-re állításával megtörténik az ADC szoftver reszet, mely a hardveres reszettel megfelelő állapotba hozza az ADC-t. A szoftveres reszelés a következőképpen végezhető el:

```
ADC->ADC_CR |= ADC_CR_SWRST;
```

Az ADC ADC_CR regiszterének segítségével a START nevű bit 1-re állításával lehet szoftveresen elindítani az AD konverziót a következőképpen:

```
ADC->ADC_CR |= ADC_CR_START;  
(természetesen ezt a lépést meg kell, hogy előzze az A/D konverter  
megfelelő konfigurálása).
```

Mode Register

Az AD konverter Mode regiszterében kerül sor a működéshez szükséges
leglényegesebb konfigurációs paraméterek megadására.

A TRGEN nevű bit egy-re maszkolásával engedélyezhetjük a hardveres
triggerelés lehetőségét.

A TRGSEL bitmező segítségével választhatjuk ki, hogy melyik
hardveres trigger lehetőséget szeretnénk használni (ehhez szükséges a hardveres
trigger engedélyezése TRGEN bit segítségével). A TRGSEL bitmező értékétől
függően a következő hardveres trigger források kerülhetnek kiválasztásra. Nulla
érték esetén az ADC_TRIG0 -val jelölt külső trigger forrás lesz aktív, amelyet
az ADTRG jelölésű IO lábról lehet meghajtani, ez a PA8- as IO lab B periféria
funkciójaként érhető el, az IO lab megfelelő konfigurálása után (ehhez az
interfész multiplexer és az IO lábhoz tartozó felhúzó ellenállás megfelelő
beállítása is szükséges). A TRGSEL 1-től 3-ig terjedő értéke esetén a Timer
Counter egységekről történő triggerelés válik lehetségesre. Az 1-es érték esetén
a Timer Counter nullás csatornája (Timer Counter Channel 0), a 2-es érték esetén
az első, a 3-as érték esetén a második csatorna TIO nevű kimenete szolgáltatja a
trigger eseményt.

A LOWRES nevű bittel történik az ADC 10 vagy 8 bites felbontásának
kiválasztása. A bit egyre maszkolásával az ADC 8 bites "low resolution" módba
állítható, nullára maszkolásával pedig 10 bites felbontásba. A LOWRES bit

hardver reset utáni alapállapota nulla, így az AD konverter alapértelmezett felbontása 10 bites.

A SLEEP és az FWUP bitek az energiatakarékkossági megoldásokat takarnak. SLEEP bit 1-be állításával az AD konverziók közötti időben az ADC mag és a referenciafeszültség kikapcsolásra kerül (az ADC "sleep" módba - alvó állapotba lép). Az FWUP bit 1-be állítása megakadályozza "sleep" mód esetén a referencia feszültség kikapcsolását, így a sleep mode-ból való gyorsabb visszatérést teszi lehetővé.

A FREERUN bit segítségével engedélyezhetjük az A/D konverter "szabadon futó" üzemmódját, ekkor a csatornák analóg digitális konverziója folyamatosan zajlik, és az ADC nem vár semmilyen trigger eseményre. A FREERUN bit 1-be állítása engedélyezi, nullába állítása pedig letiltja a szabadon futó üzemmódot, a bit hardver RESET utáni alapállapota nulla.

Az ADC_MR regiszter PRESCAL bitmezőjében adható meg az A/D konverziót ütemező ADCClock nevű órajel osztójának értéke. ADCClock órajel a mikrovezérlő főórajele az "MCK" leosztásával jön létre a következő összefüggés alapján:

$$\text{ADCClock} = \text{MCK} / ((\text{PRESCAL}+1) * 2)$$

A STARTUP négy bites bitmezőjében a "Startup Time" bekapcsolási idő beállítására van lehetőség. A STARTUP-ban megadott értéknek megfelelő késleltetést az adatlapban található táblázat tartalmazza.

A TRACKTIM 4 bites bitmezőben pedig a csatorna váltások közötti késleltetés adható meg az ADCClock órajel periódusainak felszorzásával, a következő összefüggés alapján:

$$\text{késleltetés} = (\text{TRACKTIM} + 1) * \text{ADCClock}$$

Az USEQ (Use Sequence Enable) bit 1-be állításával engedélyezhetjük az ADC csatornák tetszőleges sorrendű mintavételezését.

4.1.22 Az AD konverter csatornái

Csatorna szekvencia meghatározás (Channel Sequence Register)

A mintavételezési sorrend megadására az ADC_SEQR1 és ADC_SEQR2 nevű külön regiszterek szolgálnak, ahol 4 bites mezőkben a mintavételezni kívánt ADC csatornára lehet hivatkozni a bitmezőbe beírt 0-15 terjedő értékkel, és így beállítható a mintavételezés sorrendje. A bit mezők kiértékelése az ADC_SEQR1 alsó helyiértékű bitjeivel kezdődik, majd így halad a legfelső bitmezőig, majd ugyan ez a kiértékelés megtörténik az ADC_SEQR2 nevű regiszterre is.

Fontos megjegyezni, hogy csak azok a csatornák A/D konverziója történik meg, amelyek az ADC_CHER regiszterben engedélyezve lettek.

A mintavételezési sorrend megadásánál lehetőség van az azonos számú csatornák újbóli megadására, ilyenkor ugyan azon a csatornán egymás után több mintavétel és A/D konverzió történik az ADC_SEQR1 és ADC_SEQR2 regiszterekben megadottak alapján.

Csatornák kezelése (Channel Enable, Disable Status)

Az ADC_CHER és ADC_CHDR regiszterekben történik a ADC csatornák engedélyezése és letiltása. Amennyiben az ADC_CHER regiszterben a csatornának megfelelő helyiértékű bitet 1-be állítjuk az adott ADC csatorna

engedélyezve lesz, ha pedig az ADC_CHDR regiszterben 1-re állítjuk az adott bitet, akkor pedig a csatorna letiltásra kerül.

Az ADC_CHSR Channel Status Register segítségével kiolvashatjuk a csatornák állapotát, ha a csatornának megfelelő bit értéke 0, akkor a csatorna le van tiltva, ha pedig 1, akkor engedélyezve van.

Last Converted Data

Az **ADC_LCDR** (Last Converted Data Register) az utolsóként megtörtént AD konverzió eredményét, és a hozzá tartozó csatornaszámot tárolja. A 0.-tól 11.ig terjedő LDATA nevű bitmezőben történik az eredmény tárolása, a 12.tól 15.ig terjedő CHNB bitmezőben pedig a hozzá tartozó csatorna száma (a csatorna szám csak abban az esetben jelenik meg ha az ADC_EMR regiszterben a TAG nevű bit állapotát előzőleg 1 re állították, más különben a CHNB bitmező értéke nulla).

4.1.23 Megszakítások

Az **ADC_IER** és **ADC_IDR** regiszterekben az ADC csatornákhoz tartozó megszakításokat lehet engedélyezni vagy letiltani. Az **ADC_IMR** regiszterből a megszakítások engedélyezésének vagy letiltásának állapotát lehet kiolvasni. Az **ADC_ISR** regiszter bitje pedig magát a megszakítási események bekövetkezését jelzi vissza. A természetesen megszakítások használatát engedélyezni kell a globális megszakítás vezérlőben is.

Overrun error

Az **ADC_OVER** regiszter 0-tól 15-ig terjedő bitjeiben találhatóak a csatornákhoz tartozó úgynevezett "Overrun Error" jelzőbitek. Melyeknek 1 állapot azt jelzi, hogy a adott csatorna regisztere nem lett kiolvasva és a következő AD konverzió eredményével felülíródott.

Extended Mode Register

Az **ADC_EMR** nevű "Extended Mode Register"-ben találhatóak az automatikus komparálási funkciókhoz tartozó konfigurációs bitek.

CMPMODE bitmező 0ás értéke lehetővé teszi a megszakítási esemény generálását, ha az AD konverzió eredménye a megadott határértékek (mind két érték) alatt van. 1-es érték esetén akkor hoz létre megszakítási eseményt ha a határértékek felett van. 2-es érték esetén akkor, ha a megadott határértékek között, 3-as beállítás esetén pedig akkor, ha az AD konverzió eredménye a határértékek által megszabott tartományon kívül esik.

A **CMPALL** bit 1-re maszkolása lehetővé teszi, hogy a komparálási funkció melynek üzemmódját a CMPMODE regiszterben meghatároztuk az összes csatorna értékére nézve végbenmenjen. Ha a CMPALL bit állapota 1 akkor a komparálási funkció csak egyetlen ADC csatornára hajtódik végre, és a csatorna számát a CMPSEL bitmezőben határozhatjuk meg.

A **TAG** bit az ADC_LDCR regiszterben lévő utolsó konvertálás értékéhez tartozó csatornaszám (CHNB bitmező) megjelenésének engedélyezését látja el, ez az ADC_LCDR regiszter leírásánál lett ismertetve.

Az **ADC_CWR** regiszterben a komparálási funkcióhoz tartozó komparálási értékeket lehet megadni. A 0. tol 11. bitig terjedő LOWTHRES nevű bitmezőben a komparálási tartomány alacsony küszöbértékét lehet megadni. A 16.tól a 27.ig tartó **LOWTHRES** nevű bitmezőben pedig a felső határértéket.

Az **ADC_CDR** nevű regisztertömbből olvashatjuk ki az egyes AD csatornák konverziójának eredményét, ahol az egyes csatornákat a tömb index segítségével választhatjuk ki. A tömb index a csatornáknak megfelelően nullától

15 ig terjedhet (a regiszterbe természetesen csak az engedélyezett csatornák esetén kerül adat). A kiolvasás struktúra pointer segítségével a következő módon történet meg:

```
int adc_dummy = ADC->ADC_CDR[8];
```

Órajel adás az AD konverterek

Az AD konverter működtetéséhez a regisztereinek konfigurálásán kívül természetesen szükség van még, az AD konvertort mint áramköri egységet működtető órajel bekapcsolására.

Ezt a PMC ("Power Management Controller") struktúra PMC_PCER ("Peripheral Clock Enable Register")-e segítségével tehetjük meg, az ADC-hez tartozó bit 1-be maszkolásával (az ADC "Instance ID"-je ismeretében) a következőképpen:

```
PMC->PMC_PCER0 |= PMC_PCER0_PID29;
```

IO lábak beállítása

Az ADC csatornáinak használatához szükséges az adott csatornához tartozó IO lab megfelelő konfigurálása is. Fontos belső felhúzó ellenállás lekapcsolása, ezt megtehetjük a Pio struktúra PIO_PUDR regiszterén keresztül. Ha a felhúzó ellenállás bekapcsolva marad az a bemenetet a tápfeszültségre, akkor felhúzva meghamisítja az AD konverzió eredményét. A ADC csatornák ADC_CHER regiszterben való engedélyezésekor az csatorna a hozzá tartozó bemenethez automatikusan társításra kerül, így a periféria multiplexerben való beállításra nincs szükség. Mivel hardver reset után az IO lábak alapértelmezett helyzetben bemenetként vannak konfigurálva, a felhúzó ellenállás kikapcsolása és az ADC csatorna engedélyezésén kívül nincs más teendőnk.

4.1.24 Példa az ADC konfigurálására és használatára

```
PMC->PMC_PCER0 |= PMC_PCER0_PID29; //órajel adás az ADC egységnek  
ADC->ADC_MR |= ADC_MR_LOWRES; // 8 bites felbontás  
ADC->ADC_MR |= ADC_MR_FREERUN_ON; // freerun mód bekapcsolása  
ADC->ADC_CHER |= (0x1u << 9); // a 9. ADC csatorna bekapcsolása (PA22)  
PIOA->PIO_PUDR |= PIO_PA22; // belső felhúzó ellenállás kikapcsolás a bemeneten  
ADC->ADC_CR |= ADC_CR_START; // adc elindítása  
  
while(!(ADC->ADC_ISR&ADC_ISR_EOC9)); // várakozás a konverzió végére CH9  
adc_data=ADC->ADC_CDR[9]; // konverzió értékének kiolvasása
```

4.1.25 DAC

4.1.25.1 A SAM3N D/A konvertere

A SAM3N mikrovezérlő rendelkezik egy 10 bites felbontású digitális - analóg konverterrel, melyre az adatlap a DACC rövidítéssel hivatkozik. A DACC egység lehetővé teszi 10 bites digitális adatok analóg jellé (közvetlenül feszültséggé) történő konvertálását és megjelenítését, maximálisan másodpercenként 500 ezer minta feldolgozási sebességgel (500 ksample/s). Az analóg jel a DACC egységhez társított DAC0(PB13) jelölésű IO lábon kerülhet megjelenítésre. A D/A konverter kezelését a DACC nevű struktúra eleminek segítségével lehet elvégezni.

A D/A konverzió sebességét meghatározó órajelforrás

A digitális - analóg konvertert egy a mikrovezérlő fő órajeléről (MCK) leosztott lassabb órajel működteti, így a D/A konverzió sebességét a mikrovezérlő órajele és az osztást végző lineáris osztó beállítása határozza meg. A D/A konverter adatlapban közölt maximális órajel frekvenciája 13MHz, ebből

és a mikrovezérlő 48MHz-s fő órajeléből kifolyólag a lineáris osztó értékét legkevesebb 4-re állíthatjuk, ami 12 MHz-s D/A konverter órajelt eredményez ($48\text{MHz} / 4 = 12\text{MHz}$). A lineáris osztó értékét a DACC_MR regiszter CLKDIV bitmezőjében adhatjuk meg. Az MCK és CLKDIV által meghatározott konverziós gyakoriságra az adatlap az "Internal Trigger Period" - belső trigger periódus kifejezéssel hivatkozik.

Lehetőség van a D/A konverziók késleltetésére is, ezt a DACC_MR regiszter STARTUP nevű bitezőjében lehet beállítani. Ahol a késleltetés időtartamát az MCK periódusidejének és STARTUP nevű bitmezőben beállított értéknek szorzata adja meg.

A D/A konverziók ütemezése

Az analóg jelek előállításánál szükséges lehet, a pontos „minta feldolgozási gyakoriság” betartására a kimeneti jel megfelelő frekvenciájának biztosításához, ezért lehetőség van a D/A konverzió ütemezésére (triggerelésére). Ez az ütemezés meghatározza, hogy mikor kerüljön sor a következő digitális minta, analóg jellé történő átalakítására (D/A konverzióra). Az ütemezés történhet belső vagy külső trigger jel-forrás segítségével is. Amennyiben nincs szükség a pontos hardveres ütemezésre, a D/A konverter működhet úgynevezett szabadon futó (*free run*) üzemmódban is, ekkor a DA konverter adatregiszterébe történő beírással azonnal kezdetét veszi a D/A konverzió és az analóg értéke megjelenítésre kerül. Ekkor a D/A konverziók ütemezését az egységet működtető leosztott órajel frekvenciája határozza meg.

Triggerelés engedélyezése és tiltása

A triggerelés engedélyezését a DACC_MR regiszter TRGEN bitjének 1-re maszkolásával lehetjük meg. A bit nullás állapota esetén a triggerelés le van tiltva és a D/A konverter szabadon futó (free run) módban üzemel.

Trigger jelforrás kiválasztása

A trigger jelforrás kiválasztását a DACC_MR regiszter TRGSEL bitmezőjének segítsével lehetjük meg. A bitmező nullás értéke esetén a triggerelés a DATRG (PA2) jelzésű IO láb felfutó élére történik. Ha a TRGSEL bitmező értéke 1, 2 vagy 3 a kiválasztott trigger jelforrás a Timer/Counter (TC) egység nulla, 1-es vagy 2-es csatornájának TIO kimenete.

Adatküldés a D/A konverterek

A D/A konverter 10 bites mintáit, a DACC_CDR regiszteren keresztül fogadja. A minta a regiszter nullától kilencig számoszott bitjeibe kerülhet. A fogadott minták egy négyelemű FIFO-ba töltődnek, majd a FIFO-n keresztül kerülnek feldolgozásra. Lehetőség van két darab 10 bites minta egyidejűleg történő betöltésére is, eszel meggyorsítva az adatátvitelt. A két minta egyidejű betöltésének üzemmódját a DACC_MR regiszter WORD bitjének 1-re maszkolásával lehet beállítani. Ekkor az első minta a DACC_MR regiszter 0.-tól 9.-ig terjedő bitmezőjébe a második minta a 16.-tól 25.-ig terjedő bitmezőbe kerül.

Az adatküldés ütemezése

A négyelemű FIFO állapotáról a DACC_ISR regiszter TXRDY bitjének kiolvasásával tájékozódhatunk. Ha a bit állapota 1, a FIFO készen áll az adatok fogadására, ekkor elvégezhetjük a DACC_CDR regiszterbe történő adat írást. Amennyiben a TXRDY jelzőbit állapota nulla a FIFO megtelt. A következő D/A

konverzió befejeződése után a FIFO-ban újra lesz szabad hely. Ha a jelzőbit nulla értéke ellenére adatot töltünk a DACC_CDR regiszterbe az meghamisítja a FIFO-ban tárolt adatokat és a D/A konverter hibás működéséhez vezet.

Megszakítás alapú adatküldés

A D/A konverterbe történő adatküldés ütemezése, történhet szoftveres módon a DACC_ISR regiszter TXRDY bitjének figyelésével, vagy hardveres megszakítás segítségével.

A TXRDY bithez tartozó megszakítás engedélyezése a DACC_IER tiltása pedig a DACC_IDR regiszter TXRDY nevű bitjének 1-re maszkolásával történhet. A megszakítás tiltott vagy engedélyezett állapotát a DACC_IMR regiszter TXRDY bitjének kiolvasásával tudjuk leellenőrizni.

Ha a TXRDY (ready for transmission) megszakítást engedélyezzük, a megszakítási esemény akkor jön létre, amikor a DACC_CDR regiszter újra készen áll az adatfogadásra (van hely a FIFO-ban).

A megszakítás működéséhez

TXRDY megszakítás engedélyezésén kívül engedélyeznünk kell még a DACC egységhez tartozó megszakításokat a megszakítás vezérlőben (NVIC), és szükséges még a globális megszakítások engedélyezése is.

A D/A konverter üzembe helyezéséhez az egységet engedélyezni kell. A D/A konverter működésének engedélyezése a DACC_MR regiszter DACEN bitjének 1-re maszkolásával történik, ha a bitet nullára maszkoljuk a D/A konverter letiltásra kerül.

Lehetőség van a D/A konverter szoftveres reszelésére is. A szoftveres reszelés a DACC_CR regiszter SWRST bitjének 1-re maszkolásával történik.

A szoftveres reszettelés után a D/A konverter a hardver részletek megfelelő állapotba kerül.

4.1.25.2 A D/A konverter regisztereinek összefoglalása

DACC_CR (DACC Control Register)

A DACC_CR regiszter egyetlen bitje a SWRST 1-re maszkolásával rögzíthető a D/A konverter egységet

DACC_MR (DACC Mode Register)

A DACC_MR bitjei segítségével végezhetjük el a D/A konverter konfigurálását
TRGEN bit 1-re maszkolása engedélyezi a D/A konverter triggerjel forrásainak működését, nullára maszkolása pedig letiltja azt szabadon futó üzemmódba állítva az egységet.

TRGSEL bitmező segítségével választhatjuk ki a trigger jel forrásokat.

DACEN bit 1-re maszkolásával megtörténik a D/A konverter működésének engedélyezése, nullára maszkolásával pedig letiltjuk azt.

WORD bit 1-re maszkolása a DACC_CDR regiszterbe két minta egy idejű betöltését teszi lehetővé.

STARTUP bitmezőben beállíthatjuk a D/A konverzió előtti késleltetés mértékét.

CLKDIV bitmezőben a D/A konvertert működtető órajel órajelosztását állíthatjuk be.

DACC_CDR regiszterbe (DACC Conversion Data Register) a D/A konverzióra kerülő 10 bites mintát lehet betölteni.

A következő regiszterek a D/A konverter megszakításkezeléséhez kapcsolódnak, mindegyik regiszter három bitet tartalmaz, a bitek elnevezése és helyiértéke minden regiszterben azonos, csak funkciójukban térnek el egymástól.

A **DACC_IER** regiszter bitjeivel az adott megszakítások engedélyezését, DACC_IDR bitjeivel pedig letiltását lehet elvégezni. DACC_IMR regiszterből kiolvasható az adott megszakítások engedélyezettségének vagy letiltottságának állapota, a DACC_ISR regiszter

bitjéinek 1-es állapota esetén az adott bithez tartozó megszakítási esemény bekövetkezett.

TXRDY bit 1-es állapota azt jelzi, hogy a D/A konverter adat fogadó regisztere a DACC_CDR készen áll az új minta(adat) fogadására.

A következő két bit a D/A konverter közvetlen meóriahez hozzáféréséhez tartozik ENDTX, TXBUFE.

Órajel adás és IO lábak konfigurálása

A D/A konverter mint áramköri egység üzembe helyezéséhez szükséges az azt működtető órajel forrás engedélyezése. Az órajel engedélyezését a a PMC struktúra PMC_PCER regiszterével lehetjük meg, az DACC-hez tartozó bit 1-be maszkolásával. A D/A konverter (DACC) a 30-as számú ID-vel rendelkezik. Az órajeladás a következőképpen történik

PMC->PMC_PCER0 |= PMC_PCER0_PID30;

A D/A konverter működéséhez szükséges még a hozzá kapcsolódó IO lábak megfelelő konfigurálása. A D/A konverter kimenetéhez a PB13-as láb D jelű funkciója tartozik.

Külső triggerrel forrás használata esetén a DATRG (PA2) láb bemenetként való konfigurálására is szükség van, a DATRG a C periféria a funkcióként érhető el.

4.1.25.3 Példa a D/A konverter konfigurálására és használatára

```
#include "sam.h"
#define BOARD_MCK 48000000 // CPU clock freq - 48MHz

void init_dac(void)
{
    // IO beállítás
    PMC->PMC_PCER0 |= PMC_PCER0_PID12; // órajeladás a PIOB-nek
    //IO láb beállítása
    PIOB->PIO_PUDR |= PIO_PB13; // felhúzó ellenállás kikapcsolása
```

```

PIOB->PIO_ABCDSR[0] |= PIO_PB13; // D periféria kiválasztása
PIOB->PIO_ABCDSR[1] |= PIO_PB13;
PMC->PMC_PCER0 |= PMC_PCER0_PID30; // órajeladás a DACCnak
DACC->DACC_MR =0;
DACC->DACC_MR |= DACC_MR_STARTUP(0xff); // startup time
beállítása
// internal trigger 48MHz/3000= 16KHz
DACC->DACC_MR |= DACC_MR_CLKDIV(3000u);
// D/A konverter engedélyezés
DACC->DACC_MR |= DACC_MR_DACEN;
}

void push_sample(int data)
{
    while(!((DACC->DACC_ISR)&DACC_ISR_TXRDY));
    DACC->DACC_CDR = data;
}

void init(void)
{
    SystemInit();
    WDT->WDT_MR = WDT_MR_WDDIS;
    init_dac();
}

int main (void)
{
    init();
    unsigned int i=0;
    while(1)
    {
        if(i>1023)i=0;
        push_sample(i);
        i++;
    }
}

```

4.1.26 Impulzus szélesség moduláció (PWM)

Az impulzus szélesség moduláció (**PWM**), esetünkben egy adott periódusidejű négyszögjel kitöltési tényezőjének változtatását jelenti.

A PWM jelek alkalmazására számos technikai feladat megoldásánál szükség lehet, főként villamos fogyasztók teljesítményének szabályozásánál,

ahol a jel kitöltési tényezőjével arányos a fogyasztóra kapcsolt teljesítmény és energia. Így alkalmas fényforrások, fűtőtestek vagy motorok vezérlésére, is.

PWM jeleket alkalmazhatunk olyan esetekben is ahol az impulzus szélessége valamilyen információt hordoz úgy, mint RC szervo motorok vezérlésénél a szög elfordulás mértéke. Egy PWM jel meghatározható a jel periódus idejével, kitöltési tényezőjével és amplitúdójával.

4.1.26.1 A SAM3N mikrovezérlő PWM kontrollere

A mikrovezérlő belső periférijaként működő PWM kontroller alkalmas különböző frekvenciájú és kitöltési tényezőjű négyszögjelek előállítására. A PWM kontroller négy egymástól függetlenül paraméterezhető PWM csatornát tartalmaz. A csatornák jelölése nullától háromig terjed. minden csatorna alkalmas egy kimeneti négyszögjel előállítására.

A PWM csatornák által előállított kimeneti jelalak, az adott csatornához hozzárendelt IO lábon jelenik meg. Az adott csatornához rendelhető IO lábat a **PWM0**, **PWM1**, **PWM2** és **PWM3** jelöléssel látták el. A PWM csatorna funkció, IO lábhoz rendelését, a belső periféria engedélyezésével és a periféria multiplexer megfelelő beállításával lehet elvégezni.

4.1.26.2 A PWM jel periódusideje és kitöltése

Minden csatorna rendelkezik egy saját 16 bites számlálóval (**PWM_CCNT**). Ez a számláló tulajdonképpen egy 16-bites regiszter melyet a csatoránként kiválasztott órajel inkrementál, rendelkezik egy a számlálást behatároló és ez által az előállított jel periódusidejét meghatározó 16-bites regiszterrel (**PWM_CPRD**), és az előállított négyszögjel kitöltését meghatározó 16-bites regiszterrel (**PWM_CDTY**).

Órajel források

Az adott csatorna számlálóját inkrementáló órajel, a mikrokontroller **MCK** jelölésű, fő órajelének leosztásával állítható elő.

Az órajel leosztását egy modulo-n számláló végzi, amely kimenetin a kettő hatványainak megfelelő órajel osztást tesz lehetővé. Így az osztási arány MCK/1-től MCK/1024-ig terjedhet.

Amennyiben a moduló-n számlálóval nem tudjuk megvalósítani a kívánt értéket, rendelkezésre áll még két 8-bites **lineáris osztó**. A modulo-n számláló és a lineáris osztók együttes osztásával létrehozhatóak **CLKA** és **CLKB**-vel jelölt órajelek melyeket a PWM cellák számlálóinak inkrementálásához felhasználhatunk.

A jelalakot meghatározó tényezők

A PWM csatornák konfigurálható paramétere a **polaritás**, és a **jel igazítása** ("Align").

A polaritás átállításával a PWM jel gyakorlatilag invertálható. A jelalak igazításánál két lehetőség adódik, a balra és a középre igazítás. Alapértelmezetten a jel **balra igazított**, ekkor a számláló a felső értékének elérése után, nulláról indul újra. **Középre igazított** jelalak esetén a felső érték elérésekor a számlálás iránya megfordul és a számláló visszafelé számol, majd a nulla értéket elérve a számláló ismét felfelé fog számolni, és így tovább. Ez a jel periódusidejének megkétszerződését vonja maga után. A középre igazított jelgenerálás előnye hogy a PWM csatornák fel és lefutó élei nem lapolódnak át.

Ha egy PWM csatorna periódus idejének vagy kitöltési tényezőjének értékét működés közben megváltoztatjuk, a hatás azonnal érvényesül, és ez által szabálytalan jelalak jöhet létre. Ennek a problémának a kiküszöbölésére alkalmazzák a periódusidő vagy a kitöltési tényező dupla pufferelését („double buffering”). A dupla pufferelés egy erre a célra fenntartott regiszteren keresztül jön létre úgy, hogy a regiszterbe beírt érték a PWM jel periódusának végén átíródik, konfigurációtól függően vagy a kitöltési tényező vagy a periódusidő regiszterébe. A periódusidő és a kitöltési tényező beállításánál fontos hogy a periódusidő értéke nem lehet kisebb a kitöltési tényezőnél.

A PWM csatornák működését külön-külön engedélyezhetjük vagy letilthatjuk, az erre a célra szolgáló regisztereken keresztül.

4.1.26.3 A PWM kontroller regiszterei

A PWM kontroller regiszterei a PWM nevű struktúra elemeiként érhetők el, struktúrapointer segítségével.

PWM_MR (PWM Mode Register)

A **PWM_MR** regiszter a **CLKA** és **CLKB** elnevezésű, a PWM csatornák számlálói működtetésére alkalmas órajel források osztóinak konfigurálását végzi.

A modulo-n osztó beállítása (PREA és PREB)

A **PREA** és **PREB** a modulo-n számlálóval megvalósított osztás beállítására szolgál. Ahol a PREA a CLKA hoz, PREB pedig a CLKB-hez tartozó bitmező. Az osztási arány a következő összefüggés alapján határozható meg: $MCK/(2^N)$, ahol N a PREA és PREB-ben beállított érték, és N maximális értéke 10 lehet. Így MCK/1-től MCK/1024-ig terjedő, kettő hatványaival növekvő órajel osztás adható meg.

A lineáris osztók beállításai (DIVA és DIVB)

A **DIVA** és **DIVB** bitmezők a CLKA és CLKB jelekhez tartozó lineáris osztók (számlálók) értékét tárolják. Amennyiben a DIVA és DIVB értékei nullák akkor a CLKA, CLKB órajelek ki vannak kapcsolva. Ha az értékük 1 akkor csak a PREA, PREB osztók érvényesülnek, 2-től 255-ig terjedő érték esetén $MCK/((2^{DIVA})*PREA)$ és $MCK/((2^{DIVB})*PREB)$ összefüggések alapján határozható meg az osztás.

PWM_ENA és PWM_DIS (PWM csatornák engedélyezése és tiltása)

A **PWM_ENA** és **PWM_DIS** regiszterek a PWM csatornák engedélyezését és tiltását végzik. A **PWM_ENA** 0-tól 3-ig terjedő bitjeinek 1-re maszkolása a PWM0-tól PWM3-ig terjedő csatornákat engedélyezi. A **PWM_DIS** azonos számú regisztere pedig az annak megfelelő csatornát letiltja. A **PWM_SR** nevű PWM Status regiszter bitjeiből pedig a csatornák engedélyezett vagy letiltott állapotát tudjuk kiolvasni. Amennyiben a

csatornának megfelelő bit értéke 1, a csatorna engedélyezett. Ha nulla, akkor pedig letiltott állapotban van.

Megszakításokkal kapcsolatos regiszterek

A PWM csatornák alkalmasak periódusonként megszakítást generálni. Az adott csatornához tartozó megszakítás engedélyezését **PWM_IER** letiltását **PWM_IDR** regiszter 0-tól 3-ig terjedő bitjeinek 1-re maszkolásával érhetjük el. A megszkítások engedélyezett vagy letiltott voltát **DIVA** és **DIVB** (PWM Interrupt Mask Register) nevű regiszterből olvashatjuk ki. Nulla érték esetén az adott PWM csatornához tartozó megszakítás tiltott, 1 érték esetén pedig engedélyezett állapotban van. A megszakítási események bekövetkezését a **PWM_ISR** (PWM Interrupt Status Register) nevű regiszter csatornaszámoknak megfelelő bitjei jelzik. Az adott csatornához tartozó bit 1-es állapota azt jelenti, hogy a PWM_ISR legutóbbi olvasása óta már legalább egy jelperiódus lezajlott.

A PWM csatornák üzemmódjainak beállítása (PWM_CMR)

Minden PWM csatornához tarozik egy **PWM_CMR** (PWM Channel Mode Register) nevű regiszter melyben beállíthatóak a csatorna működéséhez nélkülözhetetlen legfontosabb paraméterek. Az adott PWM csatornához tartozó regiszter a PWM_CMR tömb elemeként érhető el, a 0-tól 3-ig számozott, a PWM csatornáknak megfelelő 0-tól 3-ig terjedő tömbindexeléssel.

Órajel forrás kiválasztása CPRD (Channel Pre-scaler)

A regiszterben található **CPRD** bitmező az adott csatorna számlálóját működtető órajel forrását hivatott megadni. Nulla érték esetén a számláló inkrementálása közvetlenül az MCK-val jelzett fő órajelről történik, 2-töl 10-ig

terjedő érték esetén a modulo-n számláló által szolgáltatott óraelosztás választható, ahol az MCK/(2^{CPRD}) összefüggés adja meg az órajel osztás mértékét. Ha a CPRD mező értékét 11-re állítjuk, akkor az órajel forrás az előzőekben már ismertetett CLKA, ha a CPRD értékét 11-nek választjuk, akkor pedig a CLKB lesz. A 11-en felüli számértékeket az adatlap "reserved" - fenntartott értéknek jelzi.

A PWM jel polaritása (CPOL)

A PWM_CMR regiszter **CPOL** bitjén a csatorna PWM jelének polaritása adható meg. Nulla beállítás esetén a jelalak alacsony szintről indul, a bit 1-es értéke esetén pedig magas szintről (az előző jelalak inverze valósul meg).

A PWM jel igazítása CALG (Channel Alignment)

A **CALG** nevű bittel a PWM csatorna jelének "igazítása" adható meg. Nullás érték esetén a jelalak **baloldalra igazított** (a 8 bites AVR-ek adatlapjában „fast PWM” módként megnevezett) módban működik, amikor a csatorna számlálója a nulla értékről indulva a periódus időt meghatározó maximumig növekszik. A CALG bit 1-es értéke esetén a jelalak **középre igazított** (a 8 bites AVR-ek adatlapjában „phase correct PWM” ként megnevezett) üzemmódban működik, ebben az esetben a számláló maximum értékét elérve a számlálás iránya megfordul, nulláig csökken, majd újra növekedni kezd, ennek megfelelően a jel periódusideje a kétszeresére nő, viszont a kitöltési tényező változatlan marad.

A **PWM_CDTY** (PWM Channel Duty Cycle) nevű regiszterben adhatjuk meg az adott csatorna kitöltési tényezőjének megfelelő számértéket.

Minden PWM csatornának meg van a hozzá tartozó regisztere melyekre a PWM_CDTY[] csatornaszámnak megfelelő tömbelemeiként lehet hivatkozni.

A **PWM_CPRD** (PWM Channel Period) nevű regiszterben a csatornák PWM jelének periódusideje adható meg, ez határozza meg a csatornához tartozó számláló maximum értékét. Az adott csatornához tartozó regiszter a PWM_CPRD tömb elemeiként érhető el. Az adott csatorna periódusideje a mikrovezérlő MCK nevű órajele és az alkalmazott osztók ismeretében a következő összefüggés szerint alakul:

balra igazított jelalak esetén: $(CRPD^*DIVA)/MCK$ vagy $(CRPD^*DIVB)/MCK$,
középre igazított jelalak esetén $(2*CRPD^*DIVA)/MCK$ vagy $(2*CRPD^*DIVB)/MCK$

*A periódus idő és a kitöltési tényező dupla pufferelésű frissítése
CPD (Channel Update Period)*

A PWM_CMRx regiszter **CPD** nevű bitje meghatározza, hogy dupla pufferelésű értékkadás esetén a csatornához tartozó PWM_CUPDx regiszter értéke a periódusidőt tartalmazó PWM_CPRDx regiszter vagy a kitöltési tényezőt tartalmazó PWM_CDTYx regiszter tartalmát frissítse (értékét írja felül). A CPD bit nulla értéke esetén a kitöltési tényezőt, 1-es érték esetén a periódusidőt írja felül. A regiszterfrissítés mindenkorán a rakkövetkező jel periódus elején történik.

A **PWM_CUPD[]** (PWM Channel Update) regiszter, dupla pufferelés esetén a csatorna kitöltési tényezőjét vagy periódusidejét frissítő regiszter, melyből a periódus elején az érték átíródik, a PWM_CMRx regiszter CPD bitjének megfelelően. minden egyes PWM csatornának meg van a külön hozzá

tartozó PWM_CUPD[] regisztere, amely a csatorna számának megfelelő indexű tömbelemként érhető el.

A PWM csatornák számlálói PWM_CCNT

A PWM csatornához tartozó **PWM_CCNT[]** (PWM Channel Counter) regiszter tartalmazza a csatorna számlálóját, az adott csatornához tartozó regiszter a PWM_CCNT[] tömb elemeként érhető el. A regiszterek csak olvashatóak ezért a számláló értékét nem lehet közvetlenül megadni. A regiszter értéke nullázódik a csatorna engedélyezésekor és balra igazított jelalak esetén, amikor a számláló eléri a maximum értéket.

4.1.26.4 Gyakorlati példák a PWM jelek alkalmazására

A SAM3N-EK fejlesztői panel hangszórójának működtetése PWM jellet

```
// IO config - beeper PIO_PA1 (PWM1)
PIOA->PIO_PUDR |= PIO_PA1; // pullup off
PIOA->PIO_PDR |= PIO_PA1; // on chip peripheral
// PIO Controller A Multiplexing PA1 - PWM1 Peripheral A
//alapbeállítás

// peripheral clk on (34, 324)
PMC->PMC_PCER0 |= PMC_PCER0_PID31;           //órajel a PWM
nek

// prescaler
PWM->PWM_MR |= (0x4u << 8); // PREA = 4 Master Clock divided
by 16 p639
PWM->PWM_MR |= (0x1u << 0); // DIVA = 1 órajel bekapcs p639

PWM->PWM_CH_NUM[1].PWM_CMR |= (0xB << 0); //CPRE = 0xB CLKA
az ürajelforrása az 1. channelnek (PWM1)

PWM->PWM_CH_NUM[1].PWM_CMR |= PWM_CMR_CPOL; // CPOL = 1 The
output waveform starts at a high level p644

PWM->PWM_CH_NUM[1].PWM_CDTY = 1500; // jel kitöltése: 50%
PWM->PWM_CH_NUM[1].PWM_CPRD = 3000; // periódus hossza: a
számláló felső értéke 3000 mert 48MHz/(16*3000)=1KHz jel
frekvencia

PWM->PWM_ENA |= (0x1u << 1); // CHID1 = 1 Enable PWM output
for channel 1 p640
```

RC szervómotor vezérlése PWM jellel

```
// IO config - servo PIO_PB1 (PWM1)
PIOB->PIO_PUDR |= PIO_PB1; // pullup off
PIOB->PIO_PDR |= PIO_PB1; // on chip peripheral
// PIO Controller A Multiplexing PA1 - PWM1 Peripheral A
//alapbeállítás

// peripheral clk on (34, 324)
PMC->PMC_PCER0 |= PMC_PCER0_PID31;           //órajel a PWM
nek

// prescaler
PWM->PWM_MR |= (0x2u << 8); // PREA = 2 Master Clock divided
by 4 p639
PWM->PWM_MR |= (47u << 0); // DIVA = 1 órajel bekapcs p639

PWM->PWM_CH_NUM[1].PWM_CMR |= (0xB << 0); //CPRE = 0xB CLKA
az órajelforrása az 1. channelnek (PWM1)
PWM->PWM_CH_NUM[1].PWM_CMR &= ~PWM_CMR_CALG; //CALG marad
nulla The period is left aligned p644
PWM->PWM_CH_NUM[1].PWM_CMR |= PWM_CMR_CPOL; // CPOL = 1 The
output waveform starts at a high level p644

PWM->PWM_CH_NUM[1].PWM_CDTY = (256+(128)); // középső servo
állásról indul (kb 1.5ms impulzusszélesség)
PWM->PWM_CH_NUM[1].PWM_CPRD = 5120; // a számláló felső
értéke 5120 mert 48MHz/(4*47*5120)=50Hz jel frekvencia 20ms
periódusidő

PWM->PWM_ENA |= (0x1u << 1); // CHID1 = 1 Enable PWM output
for channel 1 p640
```

4.1.27 A SAM3 időzítő és számláló egységei (TC)

A SAM3 mikrovezérlő TC (Timer/counter) egységei több különböző funkciócsoport ellátására is alkalmasak. A mérési alkalmazásoktól, a jelgeneráláson keresztül, a Gray code előállító és "Quadrature Decoder" alkalmazásokig.

Mivel a PWM-jel generálással és a rotary encoder kezelésével külön fejezetekben foglalkozom, a Timer/Counter tárgyalásánál főleg a frekvencia, periódusidő és kitöltési tényezők méréséhez szükséges elméleti és gyakorlati kérdések tárgyalására fektetem a hangsúlyt. Így a TC egység többi alkalmazása csak vázlatosan kerül megemlítésre.

4.1.27.1 A SAM3 időzítő/számláló (Timer/counter) egységei

A SAM3 mikrovezérlő tartalmaz két időzítő - számláló (**Timer/Counter**) egységet. Ezekre az egységekre a SAM3 adatlapja röviden csak "TC" -ként hivatkozik. minden timer counter (TC) egység három timer/counter **csatornát** foglal magába. A timer counter csatornák központi eleme egy 16 bites **számláló**. A számláló léptetése történhet belső vagy külső óraelforrásról.

4.1.27.2 Üzemmódsok, órajel források, elindítás, leállítás trigerelés

A csatornák üzemmódi

A TC csatornáknak két fő üzemmódja van, a Waveform és a Capture mód. A **Capture** mód alapvetően jelek mérésére, esemény számlálásra, frekvencia, periódusidő kitöltési tényező mérésre szolgál. A **Waveform** mód lehetővé teszi különböző frekvenciájú és kitöltési tényezőjű periódikus jelek, és impulzusok előállítását, PWM jelek generálását is.

A Waveform, Capture üzemmódok kiválasztását a TC_CMR regiszter (Channel Mode Register) WAVE jelölésű bitjével lehet elvégezni.

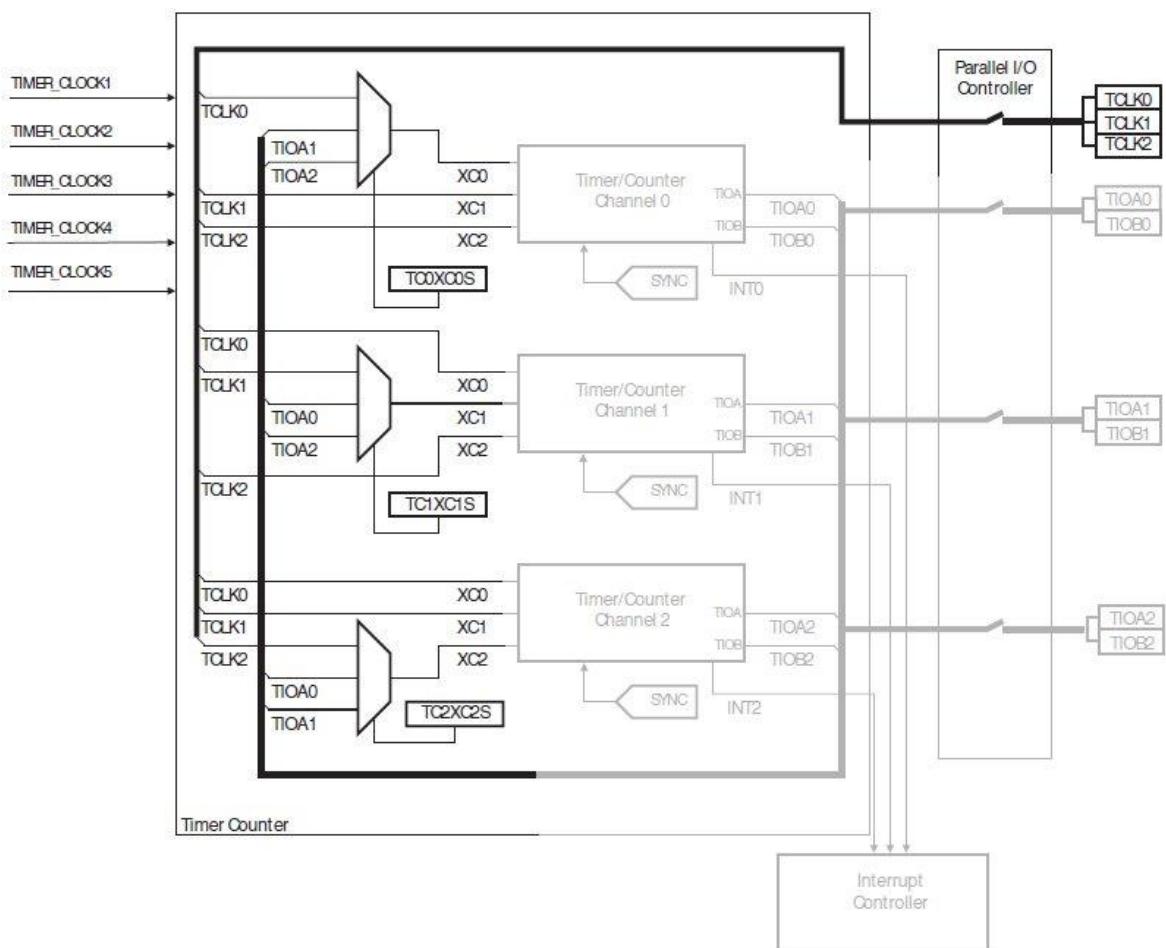
A számláló órajel forrásai

A számlálók léptetése történhet belső és külső órajel forrás felhasználásával.

Belső órajel források

A belső órajel forrás lehet a mikrovezérlő leosztott fő órajele (MCK), vagy az SCLK-val jelölt úgynevezett lassú órajel forrás. A belső órajel források jelölése TIMER_CLOCK1-től TIMER_CLOCK5-ig terjed. A TIMER_CLOCK5-el jelölt órajel forrás a lassú órajelhez (SCLK) kapcsolódik. A TIMER_CLOCK1-től TIMER_CLOCK4-ig terjedő jelöléssel ellátottak pedig a fő órajelből (MCK) leosztott órajelek. A TIMER_CLOCK1 MCK/2, TIMER_CLOCK2 MCK/8, TIMER_CLOCK3 MCK/32, a TIMER_CLOCK4 pedig MCK/128-as órajel osztást jelent.

Belső órajel források használata a számlálót alkalmassá teszi, periódus idő, kitöltési tényező vagy ezekhez hasonló jellegű mérések elvégzésére.



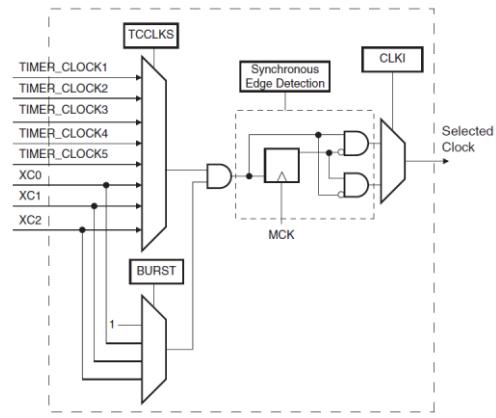
17. ábra a Timer/Counter egység blokkvázlata az órajel források kiemelésével

Külső órajel források

A számláló léptetését végezhetjük külső órajel források segítségével is. Külső órajelek használatával a számlálók alkalmassá válnak, eseményszámlálás vagy frekvenciamérési feladatokra. A külső órajeleket a csatornák **XC0**, **XC1** és **XC2** jelölésű jelvezetékeire kapcsolhatjuk. Ezek a jelvezetékek a mikrovezérlő **TIOA** és **TCLK** nevű lábaihoz vannak rendelve csatornánként eltérő összeállításban.

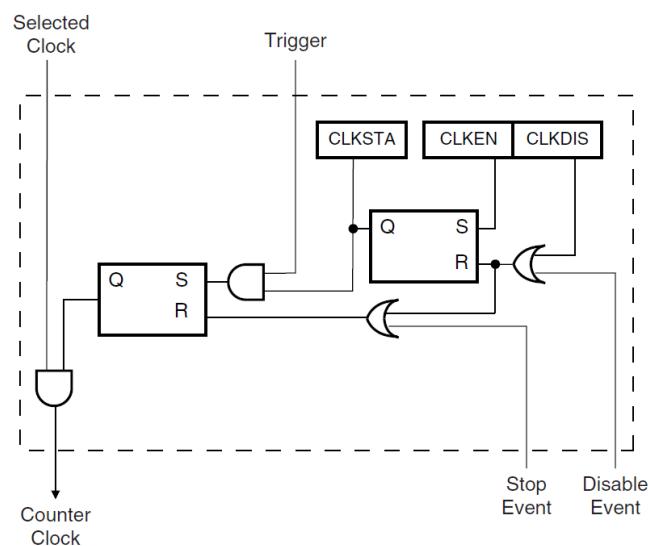
18. ábra az órajel forrás kiválasztása

Az órajel források kiválasztására a **TC_CMR[]** regiszter **TCCLKS**-el jelölt bitmezőjében van lehetőség.



A számláló elindítása és leállítása

A számláló elindítása vagy leállítása a kiválasztott órajel engedélyezésével vagy letiltásával történhet. Az órajel engedélyezés letiltás a csatornához tartozó TC_CCR[] regiszter erre a célra kijelölt bitjeinek segítségével szoftveresen történik. Viszont az órajel tényleges be és ki kapcsolását a "Trigger", "Stop Event" és "Disable Event" jelek végzik.



19. ábra a számláló elindítása, leállítása triggerrelése

A számláló elindítása

A számláló elindításához, az adott csatorna órajel regiszter engedélyezése után a trigger jelre van még szükség, ennek hatására a kiválasztott órajel megjelenik a számláló egység bemenetén.

A számláló leállítása

A számláló leállítása többféleképp is megtörténhet.

A "Stop Event"-jel segítségével a számlálás leállítható, ez után a trigger-jel a számlálót újra tudja indítani. Ekkor a trigger-jel a számlálót nullázza, és az órajelet újra elindítja. A trigger jel közvetetten származhat több különböző belső, vagy külső forrásból.

A számlálást leállíthatja a "Disable Event" is. A "Disable Event" letiltja az órajelet, így a trigger jel nem tudja újraindítani a számlálást. Ahhoz, hogy a trigger jel hatása újra érvényesülni tudjon, előbb szoftveresen engedélyezni kell az órajelet.

A számláló triggerelése

A számláló nullázását, újraindítását vagy aktuális értékének kiolvasását egy trigger jel segítségével lehet elvégezni.

A trigger jel konfigurálható, származhat külső vagy belső forrásból.

Belső trigger források

A belső trigger jelek lehetnek hardveres vagy szoftveres triggerek. A szoftveres trigger jelek közé tartozik SWTRG és a SYNC. Az SWTRG csatornánként különböző trigger jel, az adott csatornához tartozó TC_CCR[] regiszter (Channel Control Register) SWTRG-el jelölt bitjeként lehet hozzáférni. A bit 1-re maszkolásával az adott csatornán megtörténik a trigger esemény. A SYNC a timer mind három csatornájának egy idejű triggerelését végzi el, és a TC_BCR regiszter (Block Control Register) SYNC-el jelölt bitje ként lehet hozzáférni. A bit 1-re maszkolásával a timer egység minden három csatornáján egyszerre következik be a trigger esemény. Létezik egy belső hadveres trigger forrás is. Megfelelő konfigurációval a számlálóhoz tartozó "Register C"-regiszter tartalmát a számláló értékével összehasonlítva, egyenlőség esetén triggereli a számlálót (újraindítva azt).

Külső trigger források

Külső trigger forráskánt funkcionálhatnak a számlálóhoz tartozó TIOA és TIOB nevű bemenetek. A TC_CMR regiszter ABETRG bitjével választható, hogy a TIOA és TIOB bemenetek közül melyiket szeretnénk triggerként használni. A kiválasztott trigger bemenet jele egy "Edge Detector" nevű áramköri egységbe jut, melyben meghatározhatjuk, hogy a bemeneti jel felfutó, lefutó vagy esetleg mind két élre jöjjön létre trigger esemény.

A számláló pillanatnyi értékének kiolvasása

Minden számlálóhoz tartozik három különleges célú regiszter, melyek alkalmasak a számláló pillanatnyi értékének eltárolására vagy a regiszter és a számláló értékének egyenlősége esetén trigger esemény kiváltására.

Ezekre a regiszterekre az adatlap Register A, Register B és Register C névvel hivatkozik és TC_RA, TC_RB, TC_RC-vel jelöli őket.

A számláló értékeinek eltárolására a TC_RA és a TC_RB jelű regiszter alkalmas. Az érték eltárolása (Capture módban) a TIOA bemeneten lezajló esemény hatására kerülhet sor. Ez az esemény konfigurációtól függően létrjöhét felfutó él, lefutó él, vagy mindenkorának hatására.

A TC_RB regiszterbe való beírás csak akkor mehet végbe, ha a TC_RA-ba már megtörtént.

A TC_RC regiszterbe szoftveresen betöltött érték segítségével, hardveres összehasonlítást végezhetünk a TC_RC és a számláló értéke között. Ha a számláló értéke eléri a TC_RC regiszterét az trigger eseményt vált ki, ami a számláló nullását vonja maga után. Az összehasonlításhoz kapcsolódó trigger eseményt a TC_CMR[] regiszter CPCTRG bitjének 1-re maszkolásával lehet engedélyezni.

Waveform módban a TC_RA, TC_RB és TC_RC regiszterek mindegyike alkalmas összehasonlítás végzésére, és a kimeneteken megjelenő jelalakok formálására.

A kiolvasott érték felhasználása a periódusidő és kitöltési tényező mérésére:

[4.1.28 Megszakítások](#)

A timer/counter funkciókkal kapcsolatos megszakítások engedélyezése a TC_IER (Interrupt Enable Register), tiltása a TC_IDR regiszter (Interrupt Disable Register) megfelelő megszakítási eseményhez tartozó bitjeinek 1-re maszkolásával végezhető el.

A megszakítások engedélyezett vagy tiltott állapotát a TC_IMR (Interrupt Mask Register) azonos helyiértékű és elnevezésű bitjeiből lehet kiolvasni.

A TC egység QDEC (Quadrature Decoder Logic) üzemmódjához külön megszakítási események tartoznak. A TC_QIER a megszakítás engedélyezés, TC_QIDR a megszakítások tiltása, TC_QIMR a megszakítások engedélyezett vagy tiltott állapotának visszaolvasása, TC_QISR megszakítási események bekövetkezését visszajelző regiszter.

[4.1.28.1 Regiszterek kezelése](#)

A Timer/counter (TC) egységek változóit a hozzájuk tartozó struktúra elemeiként lehet kezelní, struktúrapointerek segítségével. A SAM3N mikrovezérlő két egymástól független Timer/counter (TC) egységet tartalmaz. A hozzájuk tartozó struktúrákat TC0 és TC1-el jelölik. A TC egységen belüli csatornák saját regisztereire a TC_CHANNEL[] struktúrán keresztül lehet hozzáférni, amely a TC struktúra eleme. De egyszerűbb ezt egy gyakorlati példán keresztül szemléltetni.

A következő programsorban a nullás timer egység (TC0) 0. csatornájának (TC_CHANNEL[0]), Channel Control Regiszterében (TC_CCR), órajel engedélyező bit (TC_CCR_CLKEN) 1-re maszkolása történik.

```
TC0->TC_CHANNEL[0].TC_CCR |= TC_CCR_CLKEN;  
// ugyan az a timer egység, ugyan az a regiszter csak az 1-  
es számú csatornánál  
TC0->TC_CHANNEL[1].TC_CCR |= TC_CCR_CLKEN;  
// ugyan az a 2-es csatornánál  
TC0->TC_CHANNEL[2].TC_CCR |= TC_CCR_CLKEN;  
  
// Az 1-es timer egység csatornáinál pedig így  
TC1->TC_CHANNEL[0].TC_CCR |= TC_CCR_CLKEN;  
TC1->TC_CHANNEL[1].TC_CCR |= TC_CCR_CLKEN;  
TC1->TC_CHANNEL[2].TC_CCR |= TC_CCR_CLKEN;
```

4.1.28.2 Regiszterek leírása

TC_BCR (Block Control Register)

A TC_BCR regiszter egyetlen bitje a SYNC. A SYNC bit 1-re maszkolásával, a TC egység minden három csatornájára trigger-jel juttatható egy időben.

TC_CCR (Channel Control Register)

Az adott csatornához tartozó TC_CCR regiszterben történik a csatorna órajelének, engedélyezése CLKEN bit 1-re maszkolásával, az órajel tiltása a CLKDIS bit 1-re maszkolásával, és itt kapott helyet a SWTRG bit is, melynek 1-re maszkolásával szoftveres trigger eseményt generálhatunk.

TC_CMRR regiszter kettős funkcionálisája:

A TC_CMRR (Channel Mode Register) egy funkcionálisában különlegesnek mondható regiszter, mivel a WAVE bitjének beállításától függően, a regiszter többi bitjének funkcionálása, és elnevezése is megváltozik. A WAVE bit nullára maszkolásával a csatorna (és a regiszter is) Capture móba állítható, a WAVE bit 1-re maszkolásával pedig Waveform üzemmódba.

TC_CMR (Channel Mode Register: Capture Mode)

A TC_CMR regiszter bitjei Capture mód esetén

TCCLKS (Clock Selection)

A TC_CMR regiszter TCCLKS 3 bites bitmezője az órajel forrás kiválasztására szolgál. 0-tól 4-es értéig a TIMER_CLOCK1-TIMER_CLOCK5 belső órajel források között választhatunk. Ahol a TIMER_CLOCK5 az SCLK jelölésű lassú órajel forrást jelenti (a lassú órajel forrás általában 32KHz kvarcról üzemel, ez az órajel forrás működteti a real time timer és real time clock nevű egységeket is). A TIMER_CLOCK1 - TIMER_CLOCK4-ig terjedő belső órajel források pedig a mikrovezérlő fő órajelének (MCK) leosztott órajelei. A TIMER_CLOCK1 MCK/2, a TIMER_CLOCK2 MCK/8, a TIMER_CLOCK3 MCK/32, TIMER_CLOCK4 pedig MCK/128-as órajel osztást jelent.

TCLKS	órajel forrás
0	TCLK1
1	TCLK2
2	TCLK3
3	TCLK4
4	TCLK5
5	XC0
6	XC1
7	XC2

TCCLKS bitmező 5, 6, 7 értékű beállításai az XC0, XC1 és XC2-vel jelölt külső órajel forrásokat kiválasztását jelentik.

Az XC0, XC1, XC2 külső órajel források a TCLK és TIOA, TIOB-vel jelölt IO lábakhoz vannak hozzárendelve, csatornánként eltérő kiosztással (multiplexerrel bővítve).

A külső órajel források kiosztása a 15. ábrán látható.

A **TC_CMR** regiszter CLKI (Clock Invert) bitje az órajel polaritásának kiválasztását teszi lehetővé. Nulla érték esetén, a számlálót az órajel felfutó éle inkrementálja, 1-es érték esetén pedig a lefutó éle.

A **BURST** (Burst Signal Selection) bitmező lehetőséget nyújt a kiválasztott órajel, külső jel általi bekapuzására. Nullás érték esetén a Burst funkció ki van kapcsolva, 1-es érték esetén az XC0-al jelzett jelforrás végzi a kiválasztott órajel bekapuzását, 2-es érték esetén az XC1, 3-as érték esetén pedig az XC2 jelforrás. A BURST bitmező funkcióját 16. ábrán láthatjuk.

LDBSTOP (Counter Clock Stopped with RB Loading)

Az LDBSTOP bittel konfigurálható, hogy az RB regiszterbe történő adat betöltéssel egy időben a számláló órajele leállításra kerüljön. Amennyiben a bit 1-re lett maszközva az adatbetöltéskor a számláló leállításra kerül, a számlálás a trigger esemény indíthatja újra. Az RB regiszterbe való adat betöltés csak akkor engedélyezett, ha előzőleg az RA regiszterbe már történt adat betöltés.

LDBDIS (Counter Clock Disable with RB Loading)

Az LDBDIS bittel beállítható, hogy az RB regiszterbe történő adat betöltés alkalmával a számláló órajele letiltásra kerüljön. Ha az LDBDIS bit értéke 1, az RB regiszterbe történő adatbetöltéskor a számlálót léptető órajel letiltásra kerül, ez után a trigger jel önmagában nem képes újraindítani a számlálást, előbb az órajel szoftveres engedélyezése szükséges.

ABETRG (TIOA or TIOB External Trigger Selection)

Az ABETRG bit a TIOA és TIOB külső trigger jelek közötti választást teszi lehetővé. Az ABETRG nulla értéke esetén a kiválasztott trigger jel a TIOA. Az ABETRG 1-es értéke esetén pedig a TIOB.

ETRGEDG (External Trigger Edge Selection)

Az ETRGEDG bitmező értékével azt határozzatjuk meg, hogy a kiválasztott külső trigger jelforrás mely élre jöjjön létre a trigger esemény. A bitmező nulla értéke esetén egyik élre sem, ekkor a külső trigger forrás kikapcsolnak tekinthető. 1-es érték esetén felfutó élre, 2-es érték esetén lefutó élre, a bitmező hármas értéke esetén pedig a bemenő jel minden két éle kiváltja a trigger eseményt.

CPCTRG (RC Compare Trigger Enable)

A CPCTRG bittel egy belső, hardveres triggerforrás engedélyezésére van lehetőség. Az RC jelű regiszterbe töltött érték és a számláló értékének egyenlősége esetén, az összehasonlítást végző áramköri egység trigger jelet generál. A CPCTRG bittel ennek a trigger jelnek az engedélyezését vagy letiltását tudjuk elvégezni. A CPCTRG bit 1-es értéke esetén a trigger jel engedélyezett a nullás érték esetén pedig letiltott állapotba kerül.

WAVE

A WAVE bit a Capture és Waveform üzemmódok közti átváltást teszi lehetővé. A bit nullás értéke esetén Capture, 1-es értéke esetén pedig Waveform üzemmódba kerül az adott csatorna.

LDRA (RA Loading Selection)

A LDRA két bites bitmező értéke meghatározza, hogy az RA regiszterbe való adatbetöltés (a számláló pillanatértékének betöltése) a TIOA jel mely élre történjen meg. Nullás érték esetén a betöltés nem történik meg, az adott funkció letiltottnak tekinthető. 1-es érték esetén a felfutó, 2-es érték esetén a lefutó, 3-as érték esetén pedig a TIOA jel mindenkét élre végbemegy az adatbetöltés.

LDRB (RB Loading Selection)

Az LDRB bitmező funkcionalitása megegyezik az LDRA bitmezőjével, azzal a különbséggel, hogy az RB regiszterhez kapcsolódik és az RB regiszterbe

történő adatbetöltés csak akkor engedélyezett, ha az RA regiszterbe az adatbetöltés már előzőleg megtörtént.

A TC_CMR regiszter bitjei Waveform mód esetén

TC_CMR (Channel Mode Register: Waveform Mode)

A TC_CMR regiszter bitejinek funkcionalitása Waveform mód esetén részben megegyezik a Capture módnál már tárgyaltakkal, ezért a következő részben csak az attól eltérő bitek-bitmezők kerülnek részletes tárgyalásra.

A TCCLKS, CLKI, BURST és a WAVE bitek (bitmezők) működése megegyezik a Capture módnál leírtakkal.

Az RC regiszterhez tartozó órajel leállítás és órajel letiltás

CPCSTOP (Counter Clock Stopped with RC Compare) bit segítségével az RC regiszter és a számláló értékének egyenlősége esetén lehetőség van az órajel leállítására. A bit 1-es állapota engedélyezi az órajel leállítását, nullás állapot esetén ez a funkció ki van kapcsolva.

CPDIS (Counter Clock Disable with RC Compare)

A CPDIS bit az RC és a számláló egyenlősége esetén engedélyezi az órajel letiltását. Ha a bit állapota 1, az órajel letiltás engedélyezett, ha állapota nulla, akkor a letiltás nem valósulhat meg. Az órajel letiltása után a trigger esemény önmagában nem elég az órajel újbóli elindításához, azt előbb engedélyezni kell.

EEVTEDG (External Event Edge Selection)

Az EEVTEDG bitmező, a külső trigger jelforrás, trigger eseményt kiváltó élének kiválasztására alkalmas. A bitmező 1 értéke esetén felfutóél, 2 értéke esetén lefutó él, 3 értéke esetén a fel és lefutó él is kiváltja a trigger eseményt. A bitmező nulla értéke esetén a funkció ki van kapcsolva, így semmilyen és sem vált ki trigger eseményt.

Külső trigger jel forrás Waveform módban lehet az XC0, XC1, XC2 jelforrások egyike, vagy származhat a TIOB jelű IO lábról.

EEVT (External Event Selection)

Az EEVT 3 bites bitmező szolgál a külső trigger jelforrás kiválasztására. Ha az értéke nulla az TIOB kiválasztását jelenti, 1 érték esetén XC0, 2 esetén XC1, három esetén pedig az XC2-vel jelzett bemenet szolgáltatja a trigger jelet. Az XC0, XC1 és XC2 jelek kiosztása csatornánként különböző. A jelek kiosztás 17. ábra szemlélteti.

ENETRG (External Event Trigger Enable)

Az ENETRG bit a külső trigger jelforrások engedélyezését végzi. A bit 1-es állapota esetén a külső trigger források engedélyezve vannak, nulla állapot esetén pedig letiltásra kerülnek.

WAVSEL (Waveform Selection)

A WAVESEL két bites bitmező kettős funkciót lát el, mellyel négy üzemmódot kódol. Az alsó helyiértékű bitje meghatározza a számláló számlálásának módját. Nullás érték esetén a számláló nullától felfelé számol, maximális értékénél túlcordul, és újra nullától kezd felfelé számolni. A bit 1-es állapota esetén a számláló először felfelé számol, majd elérve maximális értékét visszafelé számol egészen nulláig és így tovább. A felső helyiértékű bit, nullás értéke esetén a számláló és az RC regiszter értékének egyenlőségéhez kötött automatikus triggerelés nincs engedélyezve. A bit 1-es értékének esetén pedig engedélyezve van, így a 2 bit négy kombinációjának értékei szerint a következő négy üzemmód adódik. Ha a bitmező értéke nulla, az "UP" üzemmód, 1-es érték esetén "UPDOWN", 2 esetén "UP_RC", háromnál pedig "UPDOWN_RC" lesz beállítva.

A kimeneti jelgeneráláshoz tartozó bitmezők

A **TC_CMR** Waveform módjához tartozó következő bitmezők, a kimeneti jelek generálásának jellemzőit adják meg.

Minden csatornához két kimeneti jel tartozik, a TIOA és a TIOB.

A kimeneti jelek mindegyikére a következő utasításokat lehet kiadni, SET a jel magas szintbe kapcsolása, **CLEAR** a jel alacsony szintbe kapcsolása, a **TOGGLE** pedig a jel állapotának megváltoztatása (negálása).

Az utasítások kiadását, különböző eseményekhez lehet kötni, mint például a számláló RA, RB, RC-vel való egyenlősége, külső és szoftveres trigger események.

A következő bitmezők az adott eseményhez kapcsolódó SET, CLEAR, TOGGLE utasításokat határozza meg a TIOA és TIOB kimenetek egyikén.

A bitmezők mindegyike 2 bites, 1-es érték esetén SET, 2-es érték esetén CLEAR, 3-as érték esetén TOGGLE utasítást hajt végre a hozzá kapcsolódó esemény hatására. Nulla érték esetén a hozzá kapcsolódó esemény hatástalan az adott IO lábra.

ACPA (RA Compare Effect on TIOA)

A számláló és RA regiszter egyenlősége esetén érvénysűlő, a TIOA kimenetre vonatkozó utasítást hordozza.

ACPC (RC Compare Effect on TIOA)

Az ACPC bitmező ACPA hoz hasonló működésű, csak az esemény az RC regiszterrel való összehasonlításból adódik.

BCPB (RB Compare Effect on TIOB)

Az esemény az a számláló és az RB regiszter egyenlőségekor jön létre, a kiadott utasítás pedig a TIOB kimenetre vonatkozik.

BCPC (RC Compare Effect on TIOB)

Az esemény az a számláló és az RC regiszter egyenlőségekor jön létre, a kiadott utasítás pedig a TIOB kimenetre vonatkozik.

AEEVT (External Event Effect on TIOA)

Az AEEVT bitmezőben beállított érték a külső trigger esemény által TIOA kimenetre kiadott utasítást határozza meg.

ASWTRG (Software Trigger Effect on TIOA)

Az ASWTRG bitmezőben beállított érték a szoftveres trigger esemény által TIOA kimenetre kiadott utasítást határozza meg.

BEEVT (External Event Effect on TIOB)

A BEEVT bitmezőben beállított érték a külső trigger esemény által TIOB kimenetre kiadott utasítást határozza meg.

BSWTRG (Software Trigger Effect on TIOB)

A BSWTRG bitmezőben beállított érték a szoftveres trigger esemény által TIOB kimenetre kiadott utasítást határozza meg.

TC_CV (Counter Value Register)

A TC_CV regiszter CV-vel jelzett bitmezője, tulajdonképpen az adott csatornához tartozó 16 bites számláló, a számlálás értékét tároló regiszter. A regiszter tartalma csak olvasható.

TC_RA (Register A)

A TC_RA regiszter RA-val jelölt bitmezője, Capture üzemmódban a TC_CV számlálóból kiolvasott pillanatértéket tárolja, Waveform üzemmódban a TC_RA regiszterbe írt érték a számláló értékével összehasonlítva jelgenerálási feladatokra alkalmas (pl PWM).

TC_RB (Register B)

A TC_RB regiszter RB-val jelölt bitmezője, Capture üzemmódban a TC_CV számlálóból kiolvasott pillanatértéket tárolja, Waveform üzemmódban a TC_RB regiszterbe írt érték a számláló értékével összehasonlítva jelgenerálási feladatokra alkalmas (pl PWM).

TC_RC (Register C)

A TC_RC regiszter RC-val jelölt bitmezője, Capture üzemmódban a TC_CV számláló értékével való egyenlőség esetén trigger esemény generálására használható, Waveform üzemmódban a TC_RC regiszterbe írt érték a számláló értékével összehasonlítva jelgenerálási feladatokra alkalmas (pl PWM).

***TC_SR** státusz regiszter funkciója*

A TC_SR jelű, státusz regiszter bitjeinek kiolvasásával tájékozódhatunk az adott timer/counter cella állapotáról. A státusz regiszterben helyet foglaló bitekhez (a CLKSTA és a TIOA, TIOB kivételével) megszakítási esemény is tartozik, így a megszakításokat kezelő handlerben a státusz regiszter kiolvasásával lehet majd eldöntení, hogy mely esemény váltotta ki a megszakítást. A megszakítások engedélyezésére, és letiltására, aza arra kijelölt TC_IER és TC_IDR regiszterek azonos helyiértékű bitjei lehetőséget.

Fontos megjegyezni, hogy a TC_SR regiszter kiolvasásakor a megszakításokkal kapcsolatos bitek nullázódnak, így a megszakítások kezelésénél a regiszter kiolvasása után a megszakítási eseményeket jelző bitek törlődnek.

***TC_SR** (Status Register) bitjei*

COVFS (Counter Overflow Status)

A csatorna számlálójának túlcordulását jelző bit. Ha a bit állapota 1, a számláló legalább egyszer túlcordult., a státusz regiszter legutóbbi kiolvasása óta.

LOVRS (Load Overrun Status)

Ha a bit értéke 1, az RA vagy RB regiszter egynél többször lett feltöltve a számláló pillanatértékével, a regiszterek kiolvasása nélkül Capture módban.

Ezt jelenti a "Load Overrun" esemény. Ha a bit értéke nulla, a "Load Overrun" esemény nem következett be, vagy a csatorna Capture módban van.

CPAS (RA Compare Status)

Ha a bit értéke 1, az a számláló és az RA regiszter értéke egyenlőségének megvalósulását jelenti (Waveform mód esetén). Ha nulla, akkor az adott esemény nem következett be, vagy a csatorna Capture módban van.

A **CPBS** (RB Compare Status) és a **CPCS** (RC Compare Status) bitek funkciója a CPAS regiszterével megegyező, azzal a különbséggel, hogy az RB és RC regiszterekkel kapcsolatos eseményt jelzik.

LDRAS és **LDRBS** (RA, RB Loading Status)

Az LDRAS és LDRBS bitek 1-es értéke az RA és RB regiszterekbe való adat (számláló pillanatértékének) betöltését jelzik Capture módban. Ha a bitek értéke nulla akkor vagy nem történt adat betöltés az adott regiszternél, vagy a csatorna Waveform módban van.

ETRGS (External Trigger Status)

A bit 1-es állapota külső triggeresemény bekövetkezését jelzi vissza.

CLKSTA (Clock Enabling Status)

A CLKSTA bit nullás állapota azt jelenti, hogy a csatornához tartozó órajel ki van kapcsolva (nincs engedélyezve), a bit 1-es állapota pedig azt jelenti, hogy az csatorna órajele engedélyezve van.

MTIOA és **MTIOB** (TIOA, TIOB Mirror)

Az MTIOA és MTIOB bitek en keresztül a TIOA és TIOB jelű kimenetek állapotát olvashatjuk vissza.

Megszakítások engedélyezése, tiltása

A különböző megszakítási események engedélyezése a TC_IER megfelelő bitjeinek 1 re maszkolásával, tiltásuk pedig az azonos helyiértékű és

elnevezésű bitek TC_IDR regiszterben történő 1-re maszkolásával érhető el. A megszakításoknak az engedélyezett vagy tiltott állapotát a TC_IMR (Interrupt Mask Register) bitjeiből olvashatjuk ki.

Gray kód előállítása

A **TC_SMMR** (Stepper Motor Mode Register) regiszterben történő beállítások, csatornákhoz tartozó TIOA és TIOB kimeneteket alkalmassá teszik gray kód előállítására, így közvetetten léptetőmotorok vezérlésére.

A **Gcen** (Gray Count Enable) bit 1-re állításával engedélyezhető a gray kód előállítás, ekkor a TIOA és TIOB kimeneteket egy 2 bites gray számláló vezérli.

A **DOWN** bit segítségével a számlálás irányát adhatjuk meg, a bit nullás értéke esetén számláló felfelé, 1-es értéke esetén lefelé számol.

TC_BMR (Block Mode Register)

A TC_BMR regiszter a Quadrature Decoder üzemeltetéséhez szükséges konfigurációs lehetőségeket tartalmazza.

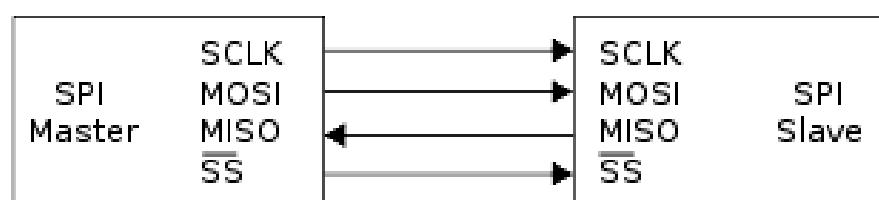
4.1.29 SPI interfész

Az SPI (Serial peripheral interfész) egy szinkron soros adatkapcsolatot megvalósító interfész, melyet elterjedten használnak beágyazott rendszerekben, kis távolságok áthalására. Tipikus felhasználásai közé tartozik az érzékelők, epromok és SD kártyák illesztése beágyazott rendszerek mikroprocesszoraihoz, de alkalmas akár mikroprocesszorok (mikro vezérlők) közti kommunikáció, (inter chip kommunikáció) megvalósítására is.

A nagyszámú SPI buszon kommunikáló eszköz és a beágyazott rendszerekben való gyakori előfordulása miatt nagy jelentősége van a beágyazott rendszerek, és automatizált gyártórendszerekkel kapcsolatos oktatásban.

4.1.29.1 Az SPI felépítése és jellemzői

"Az SPI interfész lényegét tekintve egy shift regiszter, ami sorosan követít adatokat, a másik SPI-ok felé."^[1 doc11011 p430] Az SPI busz szinkron működésű mivel az adatok küldése és fogadása egy órajel által ütemezve valósul meg. Az adat mintavételezése, léptetése konfiguraciótól függően az órajel felfutó/lefutó éleire lehet szinkronizálva. Az adat küldés és fogadás két külön vezetéken egy időben történik ezért mint adatátviteli csatorna full duplexnek tekinthető. Az SPI kommunikáció modellje úgynevezett single master multi



slave, ahol a buszon lévő egyetlen master végzi az órajel (SCLK) előállítását és az adott slave eszköz kiválasztását.

20. ábra az SPI jelei

4.1.29.2 Az SPI adatjelei adatvezetékei

MOSI: (master output slave input) a master eszköz adatkimenete és a slave eszközök adat bemeneteként szolgál melyen a master eszköz adatot küld a kiválasztott slave eszköznek.

MISO: (master input slave output) a slave eszközök adat kimenete és a master eszköz adat bemenete melyen a master által kiválasztott slave eszköz küld adatot a master eszköznek.

Adatbitek értelmezése: Az időben egymás után következő adatbitek sorában az első a legmagasabb helyértékű (MSB - most significant bit) és az utolsó a legalacsonyabb helyértékű (LSB - last significant bit). Az adat jelek esetén a magas jelszint jelenti a logikai 1-et. Általában spi busz konfigurálható paraméterei közé tartozik az adatbitek száma, mely többnyire 8-16 bit közt állítható.

4.1.29.3 Az SPI órajele

Az SPI busz órajelvezeték jelölése gyártónként, szakirodalmanként különböző lehet, az esetek nagy többségében SPCK,SCK,SCLK jelölésekkel lehet találkozni. Az spi busz órajelét a master eszköz állítja elő, rendszerint a master cpu órajelének leosztásával, melynek jellemző maximuma a cpu órajelének 1/2, 1/4-e.

Az órajel konfigurációs lehetőségei

Az órajelnek a frekvenciáján kívül általában konfigurálható paramétere még a polaritása és fázisa. Az órajel polaritása azt határozza meg, hogy az órajel a busz inaktív állapotában folytonosan alacsony vagy magas állapotban van-e. A fázisa pedig azt hogy, az órajel az inaktív állapotából kilépve az adat mintavételezését/léptetését az első vagy második ére végzi. Az órajel polaritás és fázis beállítására 1-1 bit szolgál ennek 4 kombinációja alapján spi mode-okat különböztetnek meg (mode0 - mode3) viszont az spi mode-ok meghatározása, gyártónként vagy eszközönként eltérhet, ezért célszerű ellenőrizni, hogy adott esetben ezeket az spi mode okat miként értelmezik, milyen órajel fázist és polaritást takarnak. Az spi busz adatátviteli sebességének fő meghatározó tényezője az órajele, mely az esetek többségében pár száz KHz től több 10 MHz ig terjed. Az adott esetben, maximális alkalmazható órajelnek a ki és bemenetek kapu késleltetésein túl határt szab még a nagy frekvencián tapasztalható áthallás is.

Slave eszközök kiválasztása (chip select)

A slave eszközök kiválasztása általában egy külön erre a célra fentartott vezetéken történik, amit CS(chip select) vagy SS(slave select) jelöléssel látnak el. Alapesetben minden külön slave eszközt a saját chip select vezetékkel lehet

aktiválni a chip select lábra adott alacsony jelszinttel. Egyes fejlettebb slave eszközökknél lehetőség van arra hogy, a chip select biteket az adatkeretbe ágyazzák és ilymódon az eszközök kiválasztása az adatkereten keresztül való "megcímzéssel" történik.

4.1.29.4 Az SPI interfész konfigurálása

Az ATMEL SAM3N mikro vezérlő tartalmaz egy konfigurálható hardveres SPI interfészt, amely a master és a slave-ként való működésre egyaránt alkalmas.

Az SPI interfészhez kapcsolódó konfigurációs és adat regisztereket az SPI stuktúra változóiként lehet kezelní, struktúrapointerek segítségével.

Az SPI interfész konfigurálását megelőzően, érdemes elvégeznünk az SPI buszhoz tartozó IO lábak beállítását, és mint minden más belső periféri vagy interfész esetén is, az SPI interfésnél is el kell végezni az adott áramköri egységet működtető órajel bekapcsolását. Ez az órajel az SPI interfész alapvető működéséhez szükséges és nem tévesztendő össze a SPI buszon megjelenő az adatátvitelt szinkronizáló SPCK val jelölt órajellel!

4.1.29.5 IO lábak beállítása

Az SPI buszhoz tartozó IO lábak beállítását a parallel IO controller PIO struktúráin keresztül végezhetjük el. Az SPI busz adatvezetékei (MISO, MOSI) a PORTA PA12 és PA13-as számú lábain, az SPI busz SPCK -val jelölt órajele pedig a PA14 es lábon elérhető el. A használni kívánt chip select lab(lábak) beállítása ugyancsak szükséges, a következő példában ez a NPCS2 jelölésű 2. chip select lab beállítása fog szerepelni ami konfigurálástól függően több IO lábon is hozzáférhető, a példában a PA30 port lábhoz lesz hozzárendelve.

[doc11010.pdf 388.old figure 26-3.]

IO lábak konfigurálása a belső periféria vezérlőre (I/O Line or Peripheral Function Selection)

Az SPI interfészhez kapcsolódó IO lábak alkalmazhatóak általános IO lábként "GPIO" (ilyenkor a PIO controllerhez vannak hozzárendelve) melynek az állapota szoftveresen változtatható és a kimeneteket a PIO vezérlő hajtja meg, vagy hozzárendelhetők valamely belső periféria bemenete/kimenetéhez, esetünkben az SPI interfész-hez. Hardver reset után, amennyiben nem történik konfigurálás, a GPIO lábak a PIO controllerhez vannak hozzárendelve, így általános felhasználású (GPIO) lábként működnek. Az adott porthoz tartozó IO lábak hozzárendelésének állapotát a porthoz tartozó Pio struktúra PIO_PSR regiszteréből lehet kiolvasni, esetünkben ez a PIOA->PIO_PSR struktúra pointer segítségével érhető el. A PIO_PSR regiszter 0-31 ig terjedő bitjei a PA0 tól - PA31 ig terjedő port lábak hozzárendelésének állapotát jelzik vissza, amennyiben az adott port lábhoz tartozó bit 1, a port láb a PIO controllerhez van hozzárendelve, tehát általános célú (GPIO) lábként funkcionál, ha a port lábhoz tartozó bit nulla akkor az adott IO lab valamely belső perifériához van hozzárendelve. Az IO lábak felépítését a 12. ábra szemlélteti.

Mivel hardver reset (bekapcsolás) után a az IO lábak a PIO kontrollerhez vannak hozzárendelve, ezeket szükséges átkonfigurálni a belső perifériákhoz, amit a Pio struktúra PIO_PDR (PIO Disable Register) segítségével tehetjük meg, a port lábakhoz tartozó bitek 1 re maszkolásával, a következőképpen.

```
PIOA->PIO_PDR |= PIO_PA13; // MOSI  
PIOA->PIO_PDR |= PIO_PA14; // SPCK  
PIOA->PIO_PDR |= PIO_PA30; // az NPCS2 jelölésű chip  
select láb hozzárendelése.
```

A MISO láb az SPI interface bemenetként használt lába (vezetéke), a konfigurálás itt is elvégezhető, ezzel megelőzve, hogy a PIO vezérlő esetlegesen meghajthassa a bemenetként használt lábat.

```
PIOA->PIO_PDR |= PIO_PA12; // MISO
```

A fenti regiszter bitek konfigurálása természetesen egy sorban is elvégezhető, tömören de kevésbé szemléletesen, a következőképpen:

```
PIOA->PIO_PDR |= 0x40007000u;  
vagy így:  
PIOA->PIO_PDR |= 0x40007u << 12;
```

Belső periféria funkció kiválasztása (Peripheral A or B or C or D Selection)

Az IO lábak belső perifériavezérlőre állítása a fentiekben megtörtént, de nem szabad megfeledkeznünk arról, hogy az IO lábak konfigurálásánál több különböző interfész-hez (beli perifériához) tartozó funkció kiválasztására is lehetőség nyílik. Az adott IO lábat négy különböző belső periféria vezetékeihez is hozzátársíthatjuk egy multiplexerrel keresztül, ezt a négy periféria választási lehetőséget az adatlap A B C D betűkkel jelöli, az IO lábakhoz tartozó multiplexerek beállítását (és ez által a belső periféria kiválasztását) a Pio struktúra PIO_ABCDSR1 és PIO_ABCDSR2 regiszterein keresztül lehet megtenni. Ha az adott port lábhoz tartozó bit az PIO_ABCDSR1 és

PIO_ABCDSR2 regiszterben egyaránt nulla, akkor az A jelzésű belső periféria vezeték (funkció) lesz kiválasztva, ha minden két bit 1-ben áll akkor a D jelzésű, ha a PIO_ABCDSR1 bitje 1 és az PIO_ABCDSR2 bitje 0 akkor a B jelzésű, ha a PIO_ABCDSR1 adott bitje 0 és az PIO_ABCDSR2 akkor a C jelzésű funkció lesz kiválasztva.

A MISO MOSI és SPCK vezetékek a multiplexernél az 'A' jelzésű funkcióként szerepelnek. A PIO_ABCDSR1 és PIO_ABCDSR2 regiszterek hardver reset (bekapcsolás) után nulla kezdőértéket vesznek fel, ami az 'A' jelzésű periféria (funkció) kiválasztását jelenti az összes IO lábon. Ezért a MISO MOSI és SCLK jeleknél a multiplexer alapbeállítása megfelelő. Így az konfigurációt nem igényel, viszont a PA30 láb esetén az NPCS2 (chip select) a 'B' jelzésű funkcióba került, ezért a PA30-as lábhoz tartozó multiplexert át kell állítanunk a B perifériára. Ezt a PIO_ABCDSR1 regiszter PA30-as lábhoz tartozó bitjének egybe és a PIO_ABCDSR2 regiszter PA30-as lábhoz tartozó bitjének nullába állításával tehetünk meg, a következő módon.

// a PIO_ABCDSR1 regiszter PORTA PA30 as lábához tartozó bit 1 be maszkolása.

```
PIOA->PIO_ABCDSR[0] |= PIO_PA30;
```

```
PIOA->PIO_ABCDSR[1] &= PIO_PA30;
```

a PIO_ABCDSR2 regiszter PORTA PA30-as lábához tartozó bit nullába maszkolása(mivel a regiszter bitjéinek hardver reszet utáni alapállapota nulla, ezért ezt a nullára maszkolást csak abban az esetben szükséges elvégezni, ha az adott bit előzőleg már 1-be lett állítva)

Az ABCDSR1 és ABCDSR2 regiszterek írhatóak és olvashatóak egyaránt, így a fent alkalmazott struktúrapointeres megoldással értékük lekérdezhető.

A belső periféria vezérlő beállítása, és a periféria funkció kiválasztása után szükséges lehet még a port lábakhoz tartozó felhúzó ellenállások kikapcsolása is.

Felhúzó ellenállások kikapcsolása

Mint minden IO lábhoz az SPI interfészhez kapcsolódó lábakhoz is tartozik belső felhúzó (pull-up) ellenállás ami szoftveresen konfigurálható, ezek a felhúzó ellenállások a hardver resetet követően bekapcsolva vannak, alkalmazásuk célszerű a bemenetre kötött nyomógombknál, vagy open collectoros (open draines) vonalmeghajtás esetén, viszont az SPI vezetékeknél általában úgynevezett push-pull meghajtást alkalmaznak, így a felhúzó ellenállások kikapcsolhatóak. A felhúzó ellenállások ki/bekapcsolt állapota a Pio struktúra PIO_PUSR regiszteréből lekérdezhető, amely a PORTA esetén a következőképp érhető el PIOA->PIO_PUSR. A port lábakhoz tartozó felhúzó ellenállások kikapcsolását a Pio struktúra PIO_PUDR regiszter bitjeinek 1 re maszkolásával végezhetjük el, a következő módon.

```
PIOA->PIO_PUDR |= PIO_PA13; // MOSI  
PIOA->PIO_PUDR |= PIO_PA14; // SPCK  
PIOA->PIO_PUDR |= PIO_PA30; // NPCS2  
PIOA->PIO_PUDR |= PIO_PA12; // MISO
```

Ezzel az IO lábak beállítása megtörtént, következhet az interfést működtető órajel engedélyezése.

4.1.29.6 Órajeladás a belső perifériának

Az SPI -nál mint bármely más interfész (beli periféria) esetében, az interfész mint elkülönült áramköri egység üzembe helyezéséhez (bekapcsolásához) szükség van az azt működtető órajel engedélyezésére.

Az órajelek kezelését a Power Management Controller (PMC) nevű áramköri egység látja el. Hardver reset után a belső perifériák órajele kikapcsolt állapotban van. Az órajelek engedélyezését/bekapcsolását a PMC struktúra Peripheral Clock Enable Regiszterén a PMC_PCER -en keresztül lehet elvégezni.

Egy adott periféria órajelének engedélyezésekor a perifériára a hozzá tartozó "Instance ID" számmal lehet hivatkozni (amely megtalálható az adatlap 9-1. táblázatában) az SPI interfész ID száma a 21, a fentiek alapján az SPI órajel engedélyezése a következőképpen történhet:

```
PMC->PMC_PCER0 |= PMC_PCER0_PID21;
```

4.1.29.7 Az SPI interfész paramétereinek beállítása

Az IO lábak, és a működtető órajel beállítása után elkezdhetjük magának az SPI interfésznek a beállítását.

Az SPI interfész működésért felelős regiszterekhez a SPI struktúra elemeiként lehet hozzáférni.

A regiszterek sorában az első az SPI Control Register (SPI_CR) melynek bitjeivel az SPI controller engedélyezését (SPIEN), letiltását (SPIDIS), szoftveres resetelését (SWRST) lehet elvégezni, (továbbá itt kapott helyet a LASTXFER bit is mellyel a folyamatos adatküldés esetén az utolsó byteot - az adatkeret végét lehet jelezni).

Az SPI interfész engedélyezését a konfiguráció utolsó lépéseként célszerű elvégezni amikor már az összes lényeges paraméter beállításra került és az interfész működésre kész, így erre a pontra a konfiguráció végén visszatérünk.

Az **SPI Mode Register** (SPI_MR) segítségével kiválaszthatjuk hogy az spi interfészünket master vagy slave eszközként szeretnénk használni (az MSTR bit 1 re maszkolásával az interface masterként 0-ra maszkolásával pedig slave üzemmódba kerül).

A chip select kezelésének módját ugyan ebben a regiszterben lehet beállítani: a PS bit 0 érékével "Fix" perifériakezelést, 1-be állításával pedig *variable* (változtatható) perifériakezelést tudunk megvalósítani). Fix periféria kezelés esetén a chip select kiválasztást a PCS bitek segítségével végezhetjük el. A PCS bitmezőben szereplő 16. számú bit nullára maszkolása a NPCS0 ,így a 17.bit a NPCS1, a 18. a NPCS2, a 19.bit nullára maszkolása pedig a NPCS3 -al jelölt *chip select* láb kiválasztását jelenti. Az PCS bitmezőben a legalacsonyabb helyértékű nullás bit lesz a mérvadó, felsőbb helyértékű bitek állapota érdektelen ("*don't care*"), a mind a négy PCS bit egy-be állítása tiltott kombinációnak minősül, ekkor egyetlen *chip select* láb sem kerül kiválasztásra.

Fix perifériakezelés esetén a PCSDEC (Chip Select Decode) bit egybe állításával lehetőség van az SPI_MR regiszter PCS bitjeinek közvetlen továbbítására a chip select lábakon keresztül, így a chip select lábakra kapcsolt demultiplexer segítségével 16 slave eszköz kiválasztása is megoldható.

A **PS** bit egybe maszkolásával a változtatható perifériakezelést állíthatunk be, ekkor a SPI_TDR regiszterben szereplő PCS bitek az adatkeretben kerülnek elküldésre a (LASTXFER bittel együtt), mely az ennek feldolgozására képes slave eszközök kiválasztására alkalmas.

A **DLYBCS** bitek segítségével meghatározhatjuk azt a késleltetést, amely két különböző chip select aktiválása közt kell, hogy elteljen, ezzel elejét vehetjük a chip select-ek esetleges időbeni átlapolódásának. A késleltetés értéke a DLYBCS bitek által képvisel számérték és az MCK (master clock) hánnyadosából adódik.

A **WDRBT** bit 1-be maszkolásával egy korlátozás iktatható be, miszerint az SPI interfész csak akkor indíthatja el az új adat küldését, ha a fogadott adatot

tároló regiszter (SPI_RDR) nem tartalmaz kiolvasatlan adatot, így megakadályozható az adatfogadó regiszter túlfutása.

Az **LLB** (Local Loopback Enable) bit 1-be állításával a MISO és a MOSI adatvezetékek belső összeköttetése valósítható meg master módban, így az adat küldés-fogadás közvetlenül tesztelhetővé válik.

A fentiek alapján az SPI_MR regiszterhez kapcsolódóan a következő beállításokat kell elvégezni:

Az SPI interfész beállítása master működésre:

```
SPI->SPI_MR |= SPI_MR_MSTR;
```

A chip select kezelés fix perifériás beállítása:

```
SPI->SPI_MR &= ~SPI_MR_PS;
```

A chip select lábak közvetlenül végzik a slave eszközök kiválasztását:

```
SPI->SPI_MR &= ~SPI_MR_PCSDEC;
```

(a nullára maszkolással elvégzett regiszter beállítások adott esetben nélkülözhetőek, mivel a bitek hardver reset utáni alapállapota nulla)

Az SPI_MR regiszter PCS bitjeinek beállítása, az NPCS2 chip select kiválasztásához:

```
SPI->SPI_MR |= (0xfu<<16); // PCS bitek 1 re maszkolása  
SPI->SPI_MR &= ~(0x1u<<18); // NPCS2 -nek megfelelő bit nullára maszkolása.
```

A **SPI_RDR** (Receive Data Register) nevű regiszter a SPI interfész adatfogadó regisztere, ahol a 0-tól 15-ig terjedő RD vel jelölt bitek a fogadott adatot tárolják, a PCS-vel jelölt bitek master beállítás esetén azt jelzik, hogy a chip select kimenetek milyen álapotban voltak az RD ben tárolt adat fogadásakor, így kiolvasásukkal megállapítható, hogy fogadott adat mely slave eszköztől származik.

Az **SPI_TDR** (Transmit Data Register) az SPI interfész adat küldő regisztere, ha az SPI interfész engedélyezve van, az SPI_TDR regiszterbe történő adat írása után az adatküldés azonnal elkezdődik, ekkor a benne tárolt adat átkerül a párhuzamos-soros átalakítást végző "serializer" pufferébe, és az adatküldő regiszter újra felszabadul. Az SPI_TDR regiszterben található LASTXFER bit és a PCS biteknek csak a chip select változtatható periféria kiválasztásra való konfigurálása esetén van jelentőségük, ekkor az adatkeretbe ágyazott többlet információkat tartalmazzák.

Az **SPI_SR** nevű SPI Status Regiszter fontos információkkal szolgál az SPI interfész állapotára vonatkozólag.

A regiszter **RDRF** (Receive Data Register Full) bitjének 1-es állapota azt jelzi, hogy az SPI_RDR adatfogadó regiszterben olyan új fogadott adat áll rendelkezésre ami eddig még nem került kiolvasásra.

A **TDRE** (Transmit Data Register Empty) bit 1 állapota azt jelzi, hogy a SPI_TDR adat küldés regiszter üres, és készen áll a küldendő adat fogadására. A bit nullás állapota azt jelzi hogy az adatküldés regiszter nem áll készen a következő adat fogadására, ez abban az esetben fordulhat elő ha az SPI_TDR – be már beírtuk a következő adatot de az előző adat küldése még nem fejeződött be, vagy az SPI interfész nincs engedélyezve. A TDRE bit 1 állapotának figyelése szükséges az spi-on történő adatküldés megkezdése előtt. A bit állapotának figyelése továbbá alkalmas az SPI buszon történő folyamatos adatáramlás biztosítására. Az SPI_TDR adat küldő regiszter tartalma küldés előtt átkerül a párhuzamos-soros átalakítást végző *serializer* -be ez után az SPI_TDR készen áll az új adat fogadására, így az előző adat küldésének ideje alatt a SPI_TDR regiszter a következő adattal újratölthető, biztosítva ezzel az adat

küldés folytonosságát miközben a master az adott slave eszközhöz tartozó chip selectet folytonosan aktív állapotban tartja, ennek egyes spi perifériák kezelésénél nagy jelentősége van. (Az adatküldés folytonosságát természetesen megszakítás alapon is lehet kezelni.)

A **TXEMPTY** (Transmission Registers Empty) bit 1-es állapota azt jelzi, hogy a SPI_TDR regiszter és a serializer egyaránt üresek (a bit 0-ás állapota azt jelzi, hogy a SPI_TDR vagy a serializer valamelyike nem üres - a küldés nem fejeződött be, esetleg el sem kezdődött)

Az **SPI_IER** (Interrupt Enable Register) az SPI interfész státuszbitjeihez kapcsolódó megszakítások engedélyezésére az **SPI_IDR** (Interrupt Disable Register) pedig ezek tiltására szolgáló regiszter.

Az **SPI_IMR** (Interrupt Mask Register) bitjei a különböző megszakítási események bekövetkezését jelzik, mivel az egy adott perifériához tartozó összes megszakítási esemény egy azon megszakítás handlerbe vezet, a handleren belül az interrupt mask regiszter bitjei alapján lehet eldönteni hogy a megszakítást melyik esemény váltotta ki.

Az **SPI_CSR** regiszterekben történik az SPI busz konfiguráció leglényegesebb paramétereinek megadása, mind a négy chip select lábhoz megvan a hozzá tartozó konfigurációt külön tároló SPI_SCR regiszter, így akár négy különböző beállításokat igénylő slave eszköz is használható. A négy regiszterre tömbelemekként lehet hivatkozni 0-tól 3-ig terjedő tömb indexeléssel (**SPI_CSR[0]** **SPI_CSR[1]** **SPI_CSR[2]** **SPI_CSR[3]**, melyek a NPCS0, NPCS1, NPCS2 és NPCS3 vezetékekhez tartoznak)

A **SPI_CSR** regiszter CPOL (Clock Polarity) nevű bitje, az órajel polaritásának beállítására alkalmas, ez az inaktív állapotában felvett értékét

határozza meg. Ha a CPOL bit értéke 0 , az SPCK órajel inaktív állapota alacsony logikai szint lesz, ha CPOL bit értéke 1, az SPCK inaktív állapotában magas logikai szintet vesz fel.

Az **NCPHA** bit az órajel fázisát határozza meg. Ha a bit beállítása 0 a küldött és fogadott adatok bitjei az inaktív fázisából kilépő órajel első élére változnak, és második élére történik a bit mintavételezése (ezek a polaritás beállításától függően fel és lefutó éleket jelentenek). Ha az NCPHA bit beállítása 1 az adatbitek mintavételezése az inaktív fázisából kilépő órajel első élére, a bitek változása (léptetése) pedig a második élre történik.

SPI Mode	CPOL	NCPHA	Adatbit léptetés	Adatbit mintavételezés	SPCK inaktív szint
0	0	0	lefutó élre	felfutó élre	alacsony
1	0	1	felfutó élre	lefutó élre	alacsony
2	1	0	felfutó élre	lefutó élre	magas
3	1	1	lefutó élre	felfutó élre	magas

Az órajel polaritás és fázis beállítások összefoglalása.

A **CSAAT** (Chip Select Active After Transfer) bit segítségével az adott slave eszköz folyamatosan aktív állapotban tartható az adatküldések közti időszakban, ha a CSAAT bit állapota 1 a chip select alacsony állapotban marad egészen addig, amíg egy másik slave eszköz kiválasztása meg nem történik (a chip select folyamatos aktív állapotban tartása úgy is megszakítható, ha az utolsó adatküldés előtt az SPI_CR regiszter LASTXFER bitjét 1 re maszkoljuk).

Az **SPI_CSR** regiszter BITS (Bits Per Transfer) nevű bitjei segítségével megadható az egyszerre átvinni kívánt bitek száma. Amely a BITS bitekben beállított számérték függvényében 8-tól 16 bitig változhat, ahol a hozzáartozó érték 0 tól 8 ig terjedő bináris kód határoz meg.

A **SCBR** (Serial Clock Baud Rate) bitek segítségével az MCK-ból (master órajel) való leosztással állítható be az SPI interfész órajelének frekvenciája, ahol az SCBR-ben beállított érték képviseli magát az osztót. Az SPI órajele a következőképpen alakul: SPCK = MCK/SCBR. Az órajel beállításnál fontos megjegyezni, hogy az SCBR bitek hardver resetet követően nulla értéket vesznek fel, és a SCBR nullás értéke tiltott állapotnak számít, ezért mindenkor szükséges az osztót nullától különböző értékre állítani még az első adatküldés (fogadás) megkezdése előtt.

A NPCS2 -es chip select lábhoz kapcsolódóan a következő konfigurációs beállításokat szükséges elvégezni:

Az órajel polaritásának beállítása:

```
SPI->SPI_CSR[2] &= ~SPI_CSR_CPOL;
```

Az órajel fázisának beállítása:

```
SPI->SPI_CSR[2] |= SPI_CSR_NCPHA;
```

Az órajel osztó beállítása 200KHz SPCK órajel előállításához (48Mhz/240=200KHz):

```
SPI->SPI_CSR[2] |= (240u<<8);
```

Az SPI interfész alapvető konfigurálásának végére érve, még egy lépés van hátra, engedélyeznünk kell az SPI interfész működését az SPI Control regiszterben.

```
SPI->SPI_CR |= SPI_CR_SPIEN;
```

4.1.29.8 SPI adat küldés és fogadás

Az adat küldés és fogadás az SPI_TDR regiszterbe történő adat írással elindítható, de mielőtt ezt megtennénk célszerű ellenőrizni a SPI_TDR állapotát az spi státusz regiszter (SPI_SR) TDRE bitjének ellenőrzésével. Amennyiben a

TDRE bit értéke 1, az SPI_TDR regiszter üres, így készen áll a küldeni kívánt adat fogadására.

Az alábbi példában a program addig várakozik még a TDRE bit 1 be nem vált, ez után megtörténik az adat küldés.

```
while( !(SPI->SPI_SR & SPI_SR_TDRE) );
      SPI->SPI_TDR = data;
```

Az SPI interfész működéséből adódóan az adatküldéssel egy időben adat fogadás is történik, ezért a küldés után célszerű a beérkezett adatot is kiolvasni.

Az alábbi programrészlet, addig várakozik amíg az adat meg nem érkezik az SPI_RDR regiszterbe, majd megtörténik a fogadott adat kiolvasása;

```
while( !(SPI->SPI_SR & SPI_SR_RDRF) );
      data = SPI->SPI_RDR;
```

Az SPI adat küldés-fogadást végező függvény célszerű megvalósítása:

```
unsigned char spi_send_rec(unsigned char data){
    while( !(SPI->SPI_SR & SPI_SR_TDRE)); // vár amíg az előző küldés végére
    SPI->SPI_TDR = data; // küldi az adatot

    while( !(SPI->SPI_SR & SPI_SR_RDRF)); // vár a fogadott adatra
    return SPI->SPI_RDR; // kiolvassa a fogadott adatot
}
```

A megoldás hátránya, hogy hibás konfiguráció esetén, ha például az SPI nem kap órajelet, vagy nem lesz engedélyezve, a status bitek sosem billennek a megfelelő állapotba és a program végtelenségig várakozik.

```
int spi_send_rec_timeout(unsigned char data){
    int i = 1000;
    for(i=1000; !(SPI->SPI_SR & SPI_SR_TDRE)&&(i>0);i--);
    // küldésre vár
    if(i<=0) return -1;
    SPI->SPI_TDR = data; // küldi az adatot
```

```
    for(i=1000; !(SPI->SPI_SR & SPI_SR_RDRF)&&(i>0);i--);
// fogadott adatra vár
    if(i<=0) return -1;
    return SPI->SPI_RDR; // kiolvassa a fogadott adatot
}
```

4.2 Texas Instruments

5 Ipari hálózatokban alkalmazott kommunikációs hálózatok

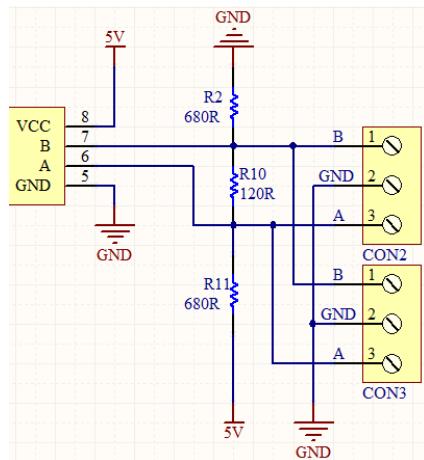
5.1 RS485

Az RS-232 interfész a CCITT nemzetközi távközlési bizottság határozta meg a soros adatátviteli interfész elektromos jellemzőit és feladatait. Ezt az adatátvitelt először számítógépek közötti kommunikációra használták, de a későbbiekben a mikrokontrollerek elterjedésének köszönhetően már minden ilyen nagyobb chip tartalmaz legalább egy soros interfész, melyet UART⁴ vagy USART⁵ –nak neveznek. A soros kommunikáció során az adatcsomagok egy START bittel kezdődnek, amely a csomag kezdetét jelzik, majd ezt követi a 7-9 -ig terjedő adatbit, amelyet a mikrokontrollereknél külön be tudunk állítani, majd következik egy paritás bit, és az üzenet egy STOP bit zárja le. Az adatátviteli sebességét bit/s–ben határozzuk meg, amely a másodpercenként átvitt bitek számát jelenti. Ezen kívül lehetőségünk nyílik beállítani a STOP bitek számát, amelynek akkor van jelentősége, hogy ha az eszközök feldolgozási sebessége nem elegendő, akkor van még egy órajel ideje feldolgozni az adatot. A mikrokontrollerekben az UART általában duplán bufferelt, amely azt jelenti, hogy a fogadott adatot rögtön áttölti egy másik bufferbe, és majd onnan történik az adat feldolgozása.

⁴ Universal asynchronous Receiver/transmitter

⁵ (Universal Synchronous Asynchronous Receiver/Transmitter

Az RS-485 buszkommunikáció alapja az UART és egy SN75176 vonali meghajtó IC segítségével. A rendszer egy kétirányú half-duplex átvitelt biztosít. A buszra fűzhető maximális adók és vevők száma 32, amelyet jelismétlők segítségével 128 résztvevőre lehet bővíteni. Nagy előnye, hogy az eszközök párhuzamosan csatlakoznak a rendszerre, ahol az első és az utolsó eszközt egy-egy 120Ω -os ellenállással lezárunk (az RX és a TX vonal közé behelyezzük a lezáró ellenállást). A kommunikációs jellezetékek „totem pole” -os kimenetűek, ami azt jelenti, hogy a vonalakat földre és tápra kell húzni egy-egy ellenállással (1.ábra).



1. ábra – Totem pole kimenet

Az RS-485 maximális adatátviteli sebessége 10m-ig 35Mbit/sec, 1200m-nél viszont már csak 100kbit/sec sebességet éri el.

5.2 CAN

A régebbi kommunikációs módok felváltására fejlesztette ki az 1980- as években a Robert Bosch Kft. a CAN6 buszt. Ezt elsősorban az autók folyamatos fejlődése váltotta ki, mivel egyre több elektronikus eszköz tartalmaztak, és ennek köszönhetően egyre bonyolultabbak lettek a kábelkorbácsok, amelyek ezeket az eszközöket összekötötték. Ezt váltotta fel ez a soros kommunikációs csatorna.

A rendszer lényege, hogy kétvezetékes csatornán helyezkednek el az eszközök, melyek minden különböző címekkel rendelkeznek. Az első CAN meghajtó chip-et 1987-ben fejlesztették ki, amely még csak a CAN 1.0 szabványt ismerte.

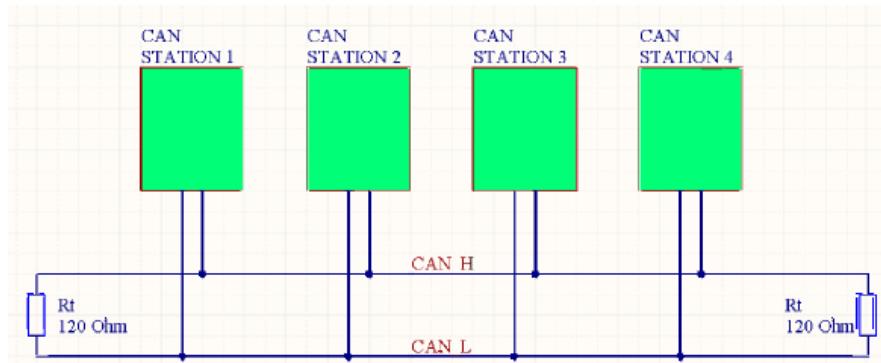
Majd a ma is használt CAN 2.0 A vagy B szabványt 1991- ben publikálta a Bosch cég. Nagymértékű elterjedését annak köszönheti, hogy nagyon magas a hibadetektálási biztonsági szintje, valamint nagy az adatátviteli sebessége (akár 1Mbit/sec). Ez elengedhetetlen is, mivel az autóiparban ezen a buszon helyezkednek el olyan eszközök, amelyek életvédelmi szempontból elengedhetetlen a megbízható működésük.

5.2.1 Fizikai réteg

A fizikai réteg felel a bitek hibamentes átviteléért. Az adatátvitel, mint említettem két vezetéken történik (2. ábra), melyeket CAN_H-nak és CAN_L-nek nevezünk. Nagy előnye, hogy itt már nem kell a földet is továbbítanunk,

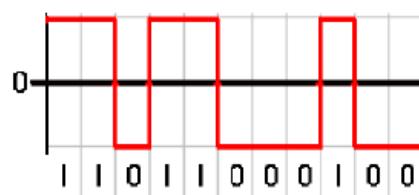
⁶ Controller Area Network

mert ezek már szimmetrikus adatvonalak, és az is, hogy könnyedén csatlakoztathatók és választhatók le az eszközök a buszról.



2. ábra – CAN eszközök összeköttetése

A CAN busz a Non return to zero⁷ [x] kódolást alkalmazza. Ennek az a szerepe, hogy csak akkor van átmenet a bitidő kezdetén, ha az előzőtől különböző bit érkezik (3. ábra). Az NRZ kódolásból, nem tudjuk kinyerni az órajelet, így a kommunikációban résztvevő eszközöknek mindenig szinkronizálóniuk kell a vételhez.

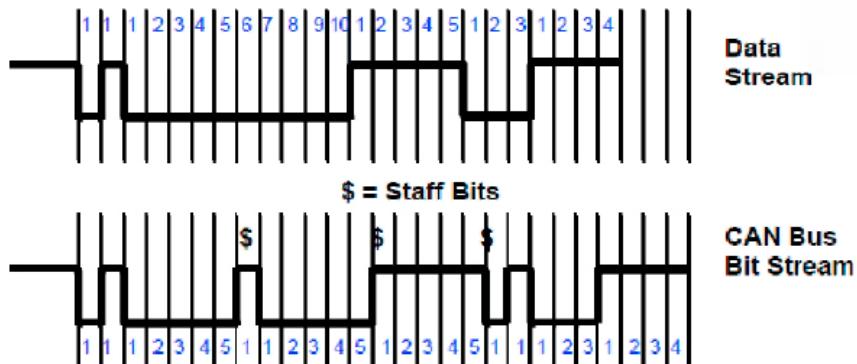


3. ábra – NRZ kódolás

⁷ Non return to zero - NRZ

A szinkronizációk két fajtája van. Az első esetben az üzenet első bitjére történik a szinkronizálás (SOF⁸). A Start Of Frame (SOF) kezdetekor egy lefutó él keletkezik, amelyre történik meg az első szinkronizáció. Ezt „Hard Syncronization”-nak nevezzük.

A második szinkronizáció előtt meg kell nézni a beszúrt bitet⁹. Abban az esetben, hogy az 5 megegyező bit követi egymást, akkor a CAN szabvány kimondja, hogy egy ellentétes bit értéket be kell szúrni az üzenetbe (4. ábra).



4. ábra – Beszúrt bitek

Lényege, hogy minden recesszív¹⁰ – domináns¹¹ bitátmenetnél a vevők hozzá tudják magukat szinkronizálni az adáshoz. Ezzel csökkentve az aszinkron kommunikációból eredő elcsúszási problémákat.

⁸ Start of frame

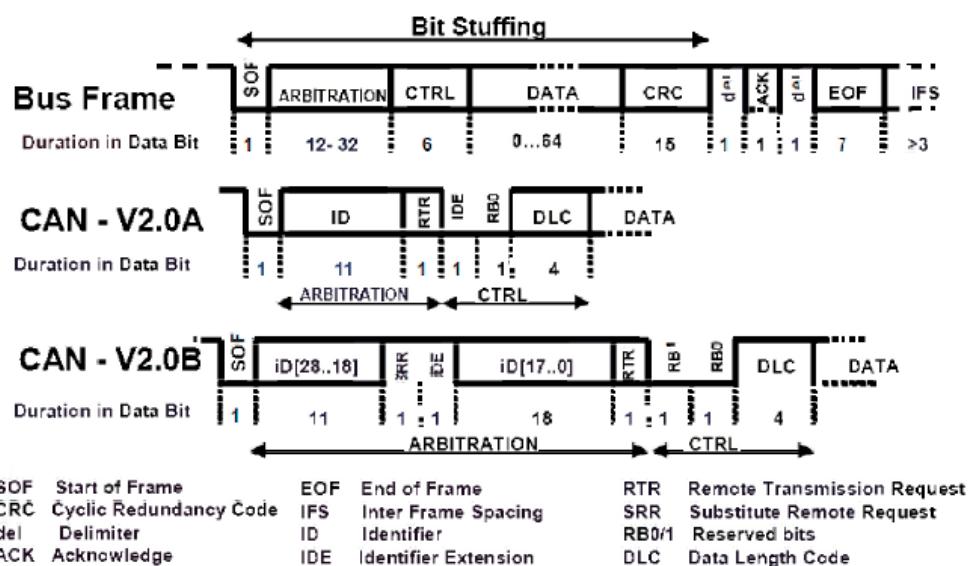
⁹ Angol elnevezés: Staff bits

¹⁰ recesszív - a CAN buszon a logikai „1” –nek felel meg

¹¹domináns - a CAN buszon a logikai „0” –nak felel meg

5.2.2 Üzenetek keretformátumai

A CAN üzenetek meghatározott keretformátumot kaptak. Az első változat a CAN 1.0 volt, de ezt ma már nem használják. A ma használatos üzenettípusok a CAN 2.0A és CAN 2.0B (5. ábra). Az eltérés a keretformátumok a között az arbitrációs¹² részben található. A CAN 2.0A formátumnál 11 bit szolgál azonosításra, amely 2048 különféle címzést tud megvalósítani, míg a CAN 2.0B-nél már 29 bit-en történik a címzés, amely pedig már 536870912 db különféle címet jelent. Ez a kibővített keretcímzést „Extended Frame” -nek is nevezik.



5. ábra – CAN keretformátumok

¹² Arbitráció: A busz használatáért történő versengés része, az üzenet elején található, ez a rész tartalmazza az üzenet címét. Az üzenet küldésekor ez a rész dönti el, hogy kinek lesz jog a adáshoz több adó esetén.

Látható hogy az atbitrációs részt leszámítva a keretek nagyon hasonlóak. Általánosan kijelenthető egy keretről, hogy a SOF (Start of Frame) bittel kezdődik. Ezt követi az arbitrációs rész, vagyis az üzenet azonosítója, majd egy RTR (Remote Transmission Request) bit következik. Ha ennek a bitnek az értéke „1” akkor adatkérés, ha pedig „0”, akkor adat küldés fog történni. A kiterjesztett keretnél lehetőségünk nyílik beállítani még az IDE (Identifier Extension) bitet, mellyel kiválaszthatjuk, hogy a standard vagy a kiterjesztett formátumú keretet szeretnénk használni.

A következő rész a CTRL field, amely tartalmazza a DLC (Data Length Code) négy bites részét, amely meghatározza, hogy az átvitt keret hány bájt adatot tartalmazzon. Ennek értéke 0 és 8 között lehet. Ezt a részt követik az adatbájtok, amelyek maximum 64 biten helyezkedhetnek.

Ezután következik egy 15 bites CRC (Cyclic Redundancy Check) mező, amely rész az üzenet hibamentes célba juttatásának ellenőrzéséért felel. Az adó állomás előállít egy 15 bites CRC kódöt, amelyet az üzenetbe illeszt, majd ezt a vevő visszakódolja, és ha megegyezik az adatbájtokkal, akkor a második ACK bitet domináns értékkel írja felül, amelyből az adó tudja, hogy az adatátvitel sikeresen megtörtént. Ha egyik vevő sem ezt a bitet domináns 0-ra, akkor ez a bit recesszív értékű marad, amely hatására küldő azt érzékeli, hogy sikertelen volt az átvitel.

Az üzenet végét egy 7 bites EOF (End Of Frame) jelzi, melynek minden értéke 1-es.

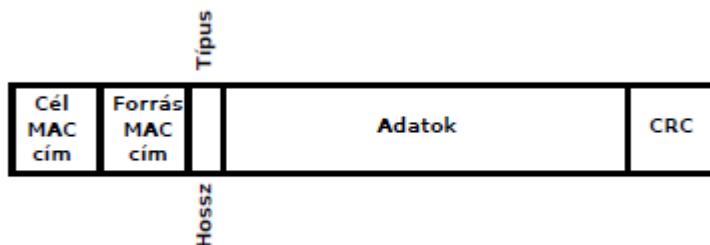
5.2.2.1.1 CAN üzenetek fajtái

- Data frame (Adat keret): Ez a csomag adatot juttat el az adótól a vevőig.
- Remote frame (Adat kérő keret): A keret lényege, hogy egy eszköz kezdeményezni tudja, hogy egy másik állomás üzenetet küldjön. Fontos, hogy az adatkérő keret azonosítójával azonos azonosítójú választ kell küldenie.
- Error frame (Hibakelező keret): Ezen a kereten keresztül tudnak visszajelzni az állomások, ha hibát észleltek.
- Overload frame (Túlcordulást jelző keret): Ha egy csomópont már nem tud több üzenetet fogatni, akkor ezzel jelez vissza az adónak, hogy késlethesse a további adat küldését.

5.2.2.2 Ipari Ethernet

Az Ethernet kommunikáció az OSI modell fizikai és adatkapcsolati rétegében helyezkedik el. A fizikai rétegen belül történik a bitek továbbítása mellett a hibakezelés és a hibadetektálás, az adatkapcsolati rétegen belül pedig a keretek összállítása és szétbontása az adó és vevő között. Ezen kívül itt történik a címzés is. A címek fizikai címek, melyeket a gyártó fixen állított be.

Ellenkezőleg az általam felhasznált IC –vel, amelynél a MAC cím szoftveresen változtatható. Az eszközök közötti kommunikáció keret felhasználásával történik, mely a 6. ábrán látható.



6. ábra –Ethernet keret

- Cél MAC: Ez tartalmazza a céleszköz fizikai címét. (általánosan 48 bit)
- Forrás MAC: A küldő fizikai címét tartalmazza. (általánosan 48 bit)
- Típus vagy hossz: Ez egy 16 bit –es rész, amely meghatározza az adatcsomag hosszát, típusát.
- Adat: A két fél közötti adatkeret tartalmazza, amely 46 – 1500 bájt lehet.
- CRC: Hibaazonosító kódrészlet

A kommunikáció a TCP/IP technológiára épül, amely a hálózati és a szállítási rétegen található. Az eszközök összeköttetésére a csillag topológiát használják, viszont ennek az volt a hátránya, hogy ha több fél azonos csatornán egy időben adást kísérelne meg, akkor adatütközés történne.

Erre fejlesztették ki a CSMA/CD protokollt, melynek lényege, hogy ha adatütközés történne, akkor az adók egy meghatározatlan ideig felfüggeszti az adást, és majd csak utána kísérlik meg újraküldést.

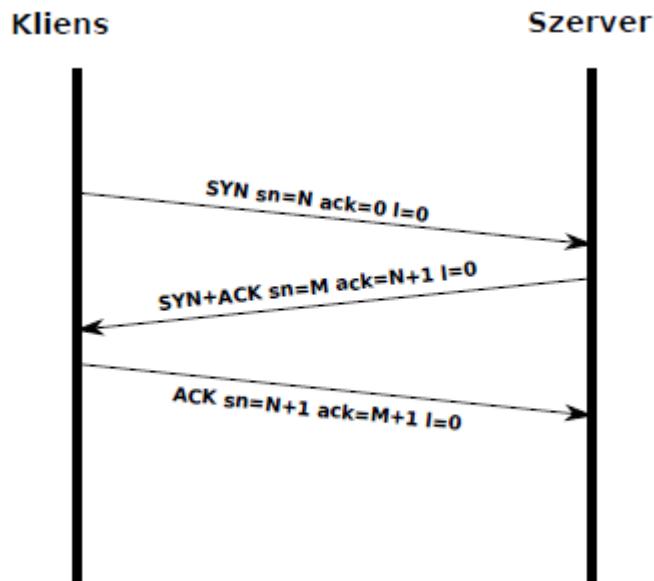
A TCP/IP egy protokollkészlet, amelyet arra fejlesztettek ki, hogy a hálózaton lévő számítógépek meg tudják osztani az erőforrásait egymással.

A TCP végzi az üzenetek széttördelését, összeállítását és datagrammok helyes sorrendjének a visszaállítását. Ugyanis minden datagramm rendelkezik egy sorszámmal, melynek köszönhetően a vevő vizsgálni tudja, hogy minden adatot helyes sorrendben kapott –e meg, illetve azt, hogy minden csomag beérkezett–e.

A TCP és az IP közötti kommunikáció a következő:

- a TCP egy datagrammot ad át az IP–nek a célba juttatandó címmel együtt.
- az IP pedig tudja, hogy az adott datagramm mely kapcsolathoz tartozik, majd azt továbbítja a megfelelő irányba.

A TCP kapcsolat kialakulását a „kliens” és a „szerver” között a következő ábra szemlélteti.



7. ábra – TCP kapcsolat kialakulása

A lépések a következők:

1. a kliens kapcsolatot kezdeményez a szerverrel

- SYN 1 (SYN bit)
- sn N (szekvencia szám)
- ack 0 (nyugtázási szám)
- I 0 (hossz)

2. a szerver nyugtázza a kérést

- SYN 1 (SYN bit)
- ACK 1 (ACK bit)
- sn M (szekvencia szám)
- ack N+1 (nyugtázási szám)
- I 0 (hossz)

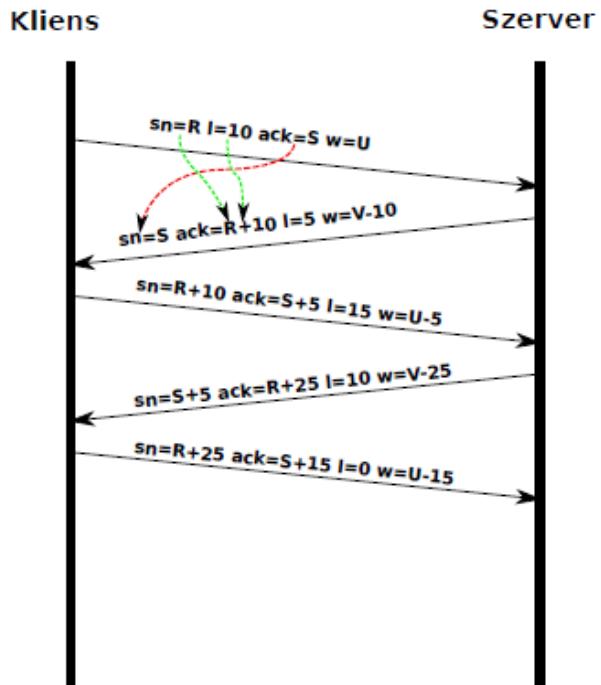
3. a kliens nyugtázza a kapcsolatfelvételt

- ACK 1 (ACK)
- sn N+1 (szekvencia szám)
- ack M+1 (nyugtázási szám)
- I 0 (hossz)

Az IP feladata létrehozni a datagramm eljuttatásához megfelelő útvonalat, hogy az eljusson a vevőhöz. A különböző átjárók és a közbülső rendszerekben történő könnyebb átjutás érdekében az IP hozzáteszi a datagrammhoz a saját fejlécét, mely a következőkből áll:

- forrás: a küldő eszköz címét tartalmazza,
- rendeltetési hely címe: a vevő oldali eszköz címét tartalmazza,
- protokoll szám: kijelöli, hogy a datagramm mely szállítási folyamathoz tartozik,
- ellenőrző szöveg: segítségével tud megbizonyosodni a vevő, hogy minden adat sikeresen célba ért.

A következő ábra egy TCP adatforgalmat reprezentál:



8. ábra – TCP adatforgalom

A kommunikáció lépései a következők:

1. a kliens adatot küld a szervernek
 - sn R (szekvencia szám)
 - ack S (nyugtázási szám)
 - w U (ablak méret)
 - I 10 (hossz)
2. szerver válaszol a kliensnek és adatot küld
 - sn S (szekvencia szám)
 - ack R+10 (nyugtázási szám)
 - w V (ablak méret)

- I 5 (hossz)

3. a kliens nyugtázza és újabb adatot küld a szervernek

- sn R+10 (szekvencia szám)
- ack S+5 (nyugtázási szám)
- w U-5 (ablak méret)
- I 15 (hossz)

Datagrammok: Egy datagrammot akkor kell széttördeni, ha az egy olyan hálózaton halad keresztül, melyen nem biztos, hogy biztonságosan célba ér. minden datagramm rendelkezik „Élettartam” mezővel, amely egy számot tartalmaz. Ez a szám minden egyes rendszeren történő áthaladáskor csökken, és ha eléri a nullát, akkor az a datagramm megsemmisül. Erre azért volt szükség, hogy ne alakuljanak ki végtelen ciklusok a kommunikációban. Végül az „Azonosítás” mező, mely ahhoz kell, hogy a vevő meg tudja állapítani a célhosszt, amely fontos szerepet tölt be, hiszen el kell dönteneni, hogy az újonnan érkezett adat mely datagrammhoz tartozik.

Az IP cím (Internet Protokoll Cím) egy egyedi hálózati azonosító, melyet az Etherneteren egymással kommunikáló eszközök azonosítására használnak. minden eszköz rendelkezik IP címmel, de egy-egy cím nem feltétlenül kötődik fix eszközhöz, mint a MAC cím.

Az IPv4 szabvány szerint az IP cím egy 32 bites egész szám, melyet négy darab egy bites számra osztottak fel, melyeket pontokkal választanak el. (a könnyebb olvashatóság miatt) pl.: 10.180.98.101

Azok az eszközök, melyek nem kapcsolódnak közvetlenül az internetre, (például belső hálózatok) azoknak nem szükséges globálisan egyedi IP címet adni. Ezeknek az eszközöknek három darab IPv4 tartomány van fenntartva.

Privát IP cím tartományok	Tartomány kezdete	Tartomány vége	Címek száma
24 bites tömb	10.0.0.0	10.255.255.255	16 777 216
20 bites tömb	172.16.0.0	172.31.255.255	1 048 576
16 bites tömb	192.168.0.0	192.168.255.255	65 536

Az IPv6 az IPv4 továbbfejlesztése. Bővítették a címzést, kijavították a hibákat, és további új szolgáltatásokat építettek bele. Ezzel az „új” internet protokollal a címzés már 128 biten történik.

Az IPv6 címeket 8 darab két bájtos csoportokra bontották pontosvesszővel elválasztva. Például: 3001 : 0db5 : 75b2 : 0000 : 0000 : 8e2a : 0268 : 8843

A cím két részből épül fel. Az első rész a hálózati előtag, a második pedig az interfész azonosító.

Az IPv6 címek három csoportra bonthatók:

- az anycast címek, melyek egy interfész csoportot azonosítanak, amelyek általában nem egy helyen találhatóak.
- az unicast címek, melyek egyedileg azonosítanak egy interfészt,

- a multicast, melynek segítségével egyszerre több interfészről is megcímhetünk egy adott csomaggal.

6 Programozható logikai eszközök

Az egyedi tervezésű, alkalmazás-specifikus IC-ktől, az ún. ASIC¹³-ektől eltérően a programozható logikai eszközök¹⁴ olyan integrált áramkörök, melyekkel a digitális technikában alkalmazott logikai kapcsolatok valósíthatóak meg és egyszer programozhatóak vagy többször újraprogramozhatóak. A programozhatóság azt jelenti, hogy az eszközben a hasznos logikai elemek között programozható összeköttetések találhatóak, melyek gyártás utáni megfelelő módosításaival kialakítható a lapkán belül a kívánt áramkör. Léteznek olyan programozható összeköttetések, melyek csak egyszer módosíthatóak¹⁵, illetve melyek a felhasználás során többször programozhatóak.

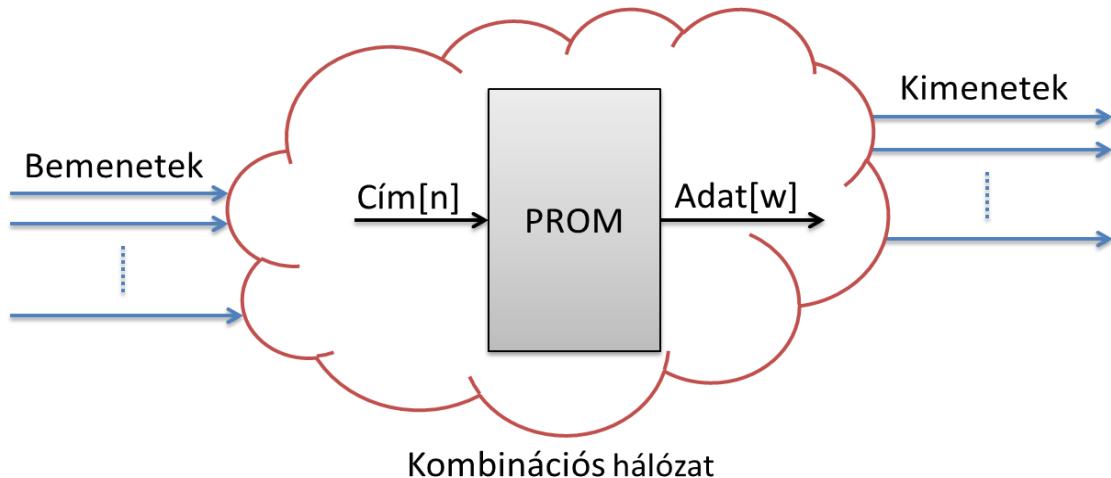
A programozható logikai eszközök megjelenése előtt is volt már lehetőség egyszerű logikai függvények programozható megvalósítására ROM, PROM, EPROM vagy EEPROM felhasználásával (1.1. ábra).

¹³ Application-Specific Integrated Circuit

¹⁴ PLD: Programmable Logic Device

¹⁵ OTP: One-Time Programmable

Ebben az esetben a memóriában el kell tárolnunk az összes lehetséges bemeneti kombinációhoz tartozó kimeneti értéket, melyeket a memória címzésével tudunk kiolvasni. Azaz a megvalósítani kívánt kombinációs hálózat bemenete a memória címvezetékeire, míg kimenete a memória kimenő adatbuszára

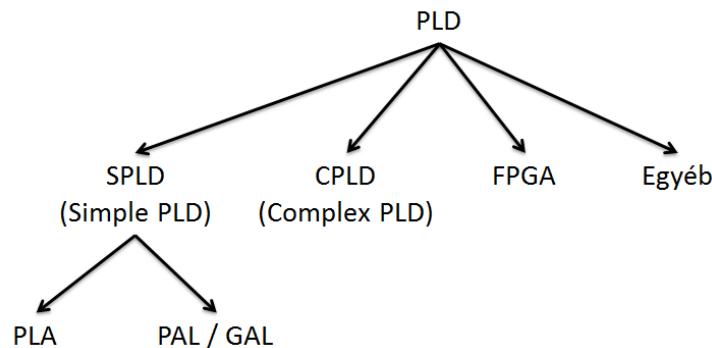


6.1. ábra: Programozható kombinációs hálózat

csatlakozhat. A memória mélysége és ezáltal a címvonal szélessége (n) szabja meg a kombinációs hálózat lehetséges független bemeneteinek számát. A memória adatszélessége (w) pedig korlátozza az egyazon bemeneti változókat használó, egymással konkurens vagy különböző bemeneti váltózók esetén nem azonos időben használható kombinációs hálózatok számát.

Az 1960-1970-es években a digitális eszközök nagy része a népszerű TTL IC-kból épült fel, viszont az új termékekkel szemben támasztott igények, illetve a kívánt funkcionalitás bővülésével a bonyolultság is oly mértékben megnőtt, hogy csak nagyon sok TTL IC felhasználásával lett megoldható egy összetett feladat. Ez nem csak az integráltságot rontja, de még a költségeket is növelheti.

Egy megoldás lehet a fentebb bemutatott memóriában implementált kombinációs hálózat, bár kissé pazarlónak tűnhet a funkcionalitáshoz tartozó szilíciumterület szempontjából és ráadásul nem is túl rugalmas. Ezek a felismerések vezettek az első programozható logikai eszközök megjelenéséhez. A programozható logikai eszközök felépítésük szerint az **Hiba! A hivatkozási forrás nem található..** ábra szerint csoportosíthatóak. Ez a csoportosítás egyúttal egy időrendi besorolás is és tükrözi az egyes eszközök bonyolultságát. A következő alfejezetekben áttekintjük ezeket a kategóriákat.



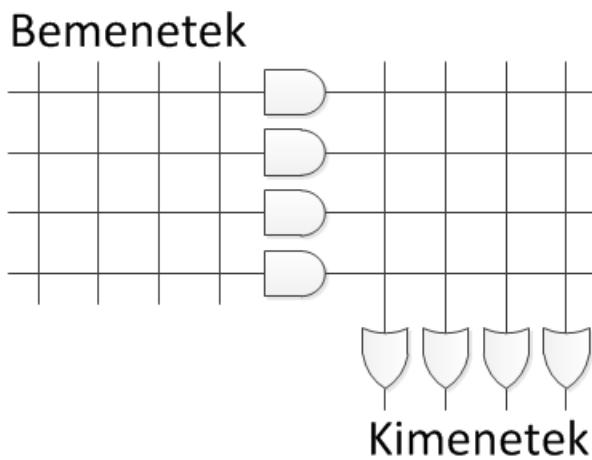
6.2. ábra: Programozható logikai eszközök csoporthozása

6.1 SPLD

Az SPLD¹⁶ egyszerű programozható logikai eszközt jelent. Kronológiai szempontból ezek az eszközök a legrégebbiek és egyben a legegyszerűbbek. Ide tartoznak a PLA (Programmable Logic Array) és a PAL (Programmable Array Logic) architektúrájú eszközök.

¹⁶ Simple Programmable Logic Device

A PLA ötletével 1975-ben Ron Cline állt elő a Signetics™ vállalatnál. Mivel kombinációs hálózatokat általában kétszintű ÉS-VAGY kapuhálózattal realizálunk, ezért kézenfekvőnek tűnik, ha a felhasználóknak egy olyan univerzális ÉS-VAGY logikai tömböt nyújtanak, ahol egyrészt a programozható összeköttetések segítségével a bemenetek az egyes ÉS kapukhoz rendelhetőek (ÉS-mátrix programozása), továbbá pedig az ÉS-kapuk és a VAGY-kapuk



6.3. ábra: PLA architektúra

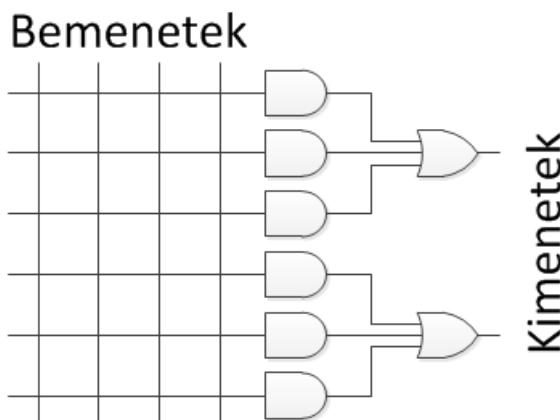
közötti összeköttetések (VAGY-mátrix programozása) is meghatározhatóak.

A PLA előnye, hogy csak a lapkán lévő ÉS-, ill. VAGY-kapuk száma korlátozza a realizálható kombinációs hálózat nagyságát, hiszen bármilyen ÉS-VAGY kombináció létrehozható. Az egy chipen megvalósítható logikai függvények száma legfeljebb annyi lehet, amennyi kimenet rendelkezésre áll. Továbbá lehetőség van a már egyszer kialakított mintermek megosztására is több logikai függvény között.

Hátránya, hogy az ÉS-, illetve VAGY-kapuk számának növelésével négyzetesen növekszik a programozható összeköttetések száma is minden két mátrix esetében, mely egyre lassabbá teszi az eszközt (egyre növekvő t_{pd} késleltetési idő).

Nem sokkal később, 1978-ban a Monolithic Memories Inc. (MMI) cégnél dolgozó John Birkner és H. T. Chua megalkották a PAL-t. Az 1.4. ábrán látható, hogy az eszköz a PLA-hoz hasonlóan úgyszintén kétszintű ÉS-VAGY kombinációs hálózat realizálását teszi lehetővé, viszont csak egyetlen programozható ÉS-mátrixszal rendelkezik, az ÉS-, illetve VAGY-kapuk közötti összeköttetések pedig fixek.

Előnye, hogy csak egyetlen programozható mátrixot tartalmaz, így az eszköz gyorsabb működésre képes, kisebb és olcsóbb is, mint a PLA, viszont hátránya,



6.4. ábra: PAL architektúra

hogy kevésbé rugalmas.

Az egyszer programozható eszközökönél a gyártás során a programozható összeköttetések minden csomópontron össze voltak kötve. A felhasználó egy speciális programozóval tudta „kiégetni” a felesleges összeköttetéseket. Nagy

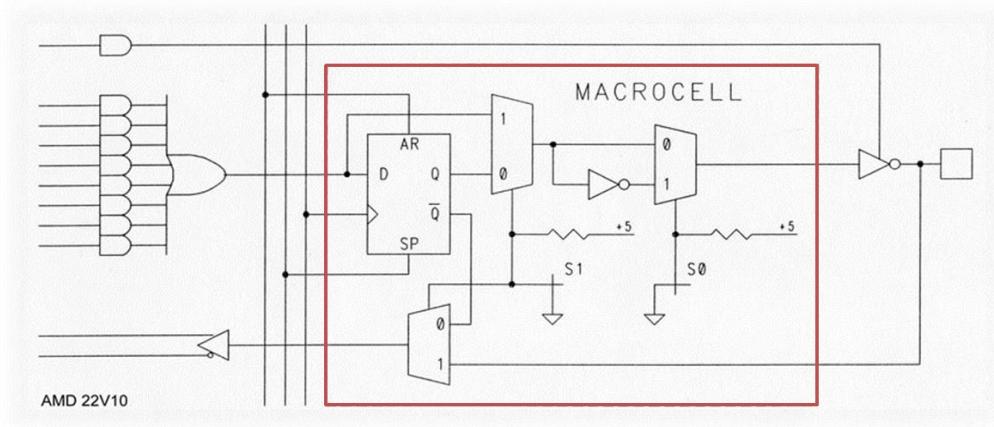
sorozatú gyártás esetén a felhasználás helyén történő programozás költsége és ideje megtakarítható volt az eszköz gyártása során igényelt maszkprogramozással. Továbbá megjelentek az EPROM-okhoz hasonló UV fénnyel törölhető változatok is, lehetővé téve az újraprogramozást.

Kezdetben a fejlesztés részeként az alkalmazás tervezőjének kellett a realizálni kívánt logikai függvényből származtatnia a programozható összeköttetések állapotát leíró bittérképet (fuse map) azaz, hogy melyik összeköttetés maradjon továbbra is fenn és melyik szünjen meg.

Ezen munka megkönnyítésére terveztek az első kezdetleges hardverleíró nyelveket¹⁷, mint a PALASM (PAL Assembler), az ABEL (Advanced Boolean Expression Language), illetve a CUPL (Compiler for Universal Programmable Logic). Ezek elsődleges célja az volt, hogy egy szöveges állományban a megfelelő nyelvi szintaxis betartásával megírt logikai kifejezésekkel a programozó eszköz által értelmezhető bittérképet generáljanak.

¹⁷ HDL: Hardware Description Language

A PAL kezdeti sikerei után 1983-ban az AMD™ olyan eszközt jelentetett meg, melynek alapja a PAL struktúra, viszont minden VAGY kapu kimenetére egy úgynevezett makrocellát integrált (1.5. ábra). minden makrocella a felhasználó által konfigurálható, azaz eldönthető, hogy egy VAGY-kapu kimenete, mint kombinációs kimenet csatlakozzon a lapka kimeneti lábához vagy pedig egy flip-flopon keresztül regisztrált kimenetet nyújtson. Ezen felül beállítható, hogy



6.5. ábra: AMD 22V10 - a makrocella megjelenése

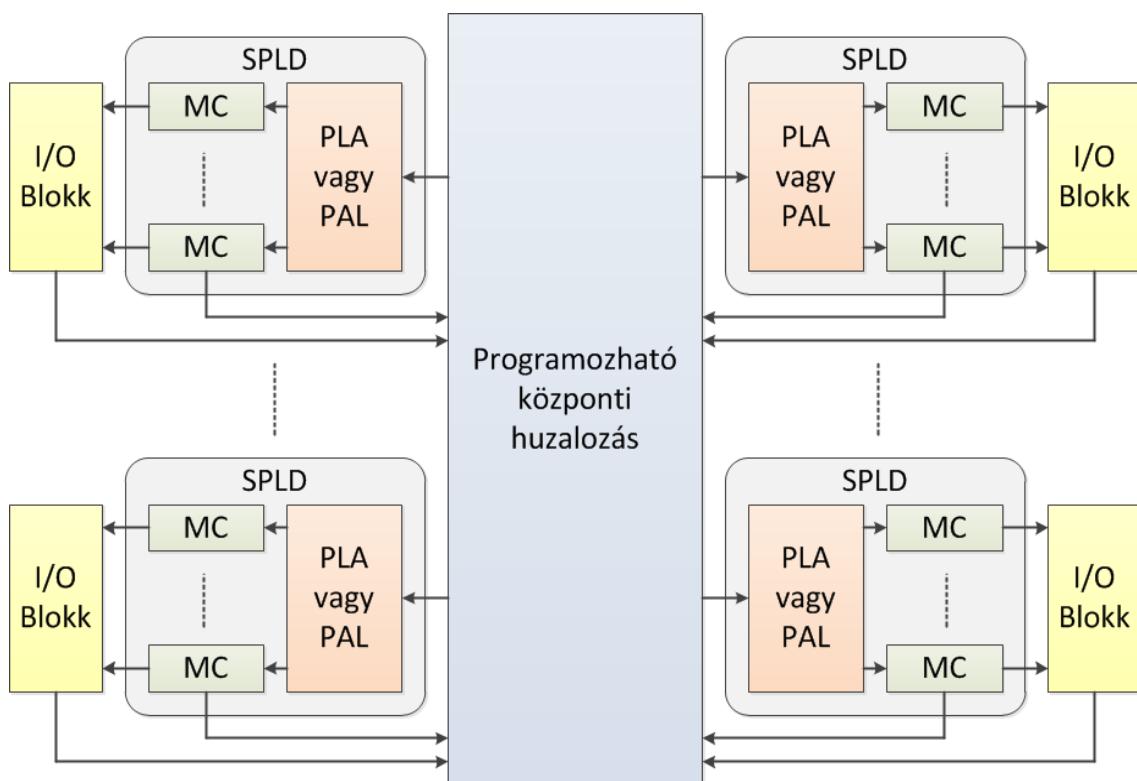
az adott kimenet aktív szintje az alacsony vagy a magas legyen.

1985-ben a Lattice Semiconductor kifejlesztette a GAL (Generic Array Logic) lapkát, mely logikailag ugyanazt a struktúrát alkalmazza, mint a népszerűvé vált PAL, viszont elektromosan törölhető és újraprogramozható, ami által a programozható logikai eszközök alkalmassá váltak prototípus-fejlesztésre.

6.2 CPLD

A CPLD¹⁸ komplex programozható logikai eszközt jelent. Bonyolultságukat tekintve az SPLD-k és az FPGA-k között helyezkednek el.

A CPLD-k általános felépítését mutatja az 1.6. ábra. Az alapkoncepció, hogy a gyártástechnológia fejlődésének köszönhetően több SPLD blokkot integráljanak egyetlen lapkára, közöttük egy központi (globális) programozható huzalozással. minden SPLD blokk tartalmaz kombinációs hálózat megvalósítását lehetővé



6.6. ábra: CPLD architektúra

¹⁸ Complex Programmable Logic Device

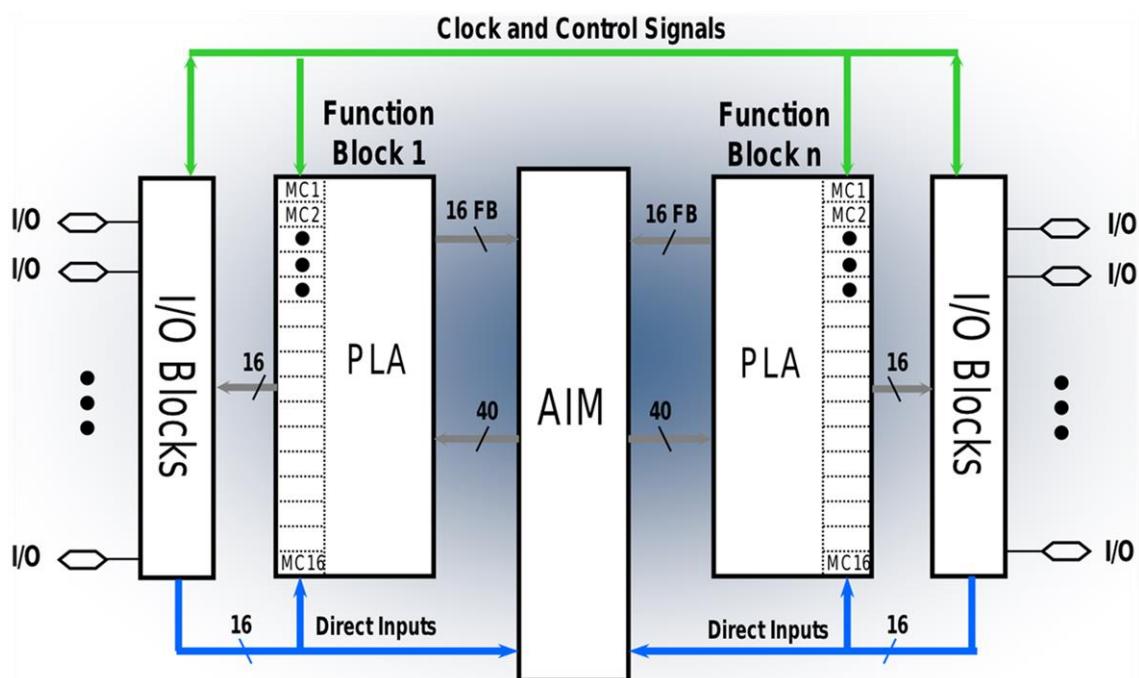
tevő PLA vagy PAL struktúrát. Ezek kimenetei (az 1.6. ábrán „MC”-vel jelzett) makrocellákhoz kapcsolódnak, melyek segítségével konfigurálható, hogy egy kombinációs kimenet regisztrált legyen-e vagy sem. Azaz a makrocellákban lévő regiszterek felhasználásával lehetőség van nem csak kombinációs, de szekvenciális hálózatok kialakítására is. Egy makrocella kimenetén megjelenő eredmény akár a lapkán belül felhasználható vagy a külvilág felé vezethető I/O (Input/Output), azaz bemeneti/kimeneti blokkon keresztül.

Az I/O blokkok segítségével a CPLD különböző szabványos jelszintekhez (pl. 5V; 3,3V; 2,5V) könnyedén illeszthető. Továbbá tipikusan lehetőséget nyújt lábanként felhúzó vagy lehúzó ellenállás bekapcsolására, egy kimenet nagyimpedanciássá tételere vagy a slew rate beállítására.

Példaként az 1.7. ábra mutatja egy Xilinx® CoolRunner-II CPLD blokkvázlatát. A programozható központi huzalozás fantázianeve AIM: Advanced Interconnect Matrix. Az SPLD blokkokat „Function Block”-nak (FB) nevezik, melyek PLA struktúrából és makrocellákból épülnek fel. A CoolRunner-II család tagjai 32 ... 512 db makrocellát tartalmaznak a választott típusról függően.

6.3 FPGA

Az FPGA (Field-Programmable Gate Array) architektúrája teljesen más koncepcióra épül, mint az eddig bemutatott SPLD és CPLD eszközök. Komplexitását tekintve egy bonyolultabb eszköz. Alapját Ross Freeman ötlete



6.7. ábra: Xilinx CoolRunner-II CPLD blokkvázlata

és szabadalma adja, aki erre építve Bernard Vonderschmitt és James Barnett társaságában megalapította a Xilinx® céget 1984-ben, amely mai napig az élvonalmi FPGA-k megalkotója és szállítója világszerte (az első FPGA-t 1985-ben szállították). Az FPGA-k architektúrájuknak folyamatos fejlesztése és a legfejlettebb gyártástechnológia alkalmazása révén mára az elsőszámú

programozható logikai eszközökkel váltak. Napjainkban az FPGA-k piacának legnagyobb résztvevői a Xilinx® és az Altera® vállalatok. Az Altera® egy 1983-ban kifejezetten újraprogramozható logikai eszközök tervezésére alapított cég, mely első FPGA-ját 1992-ben szállította. Más cégek is kínálnak FPGA-kat, mint pl. a Microsemi® vagy a Lattice Semiconductor®, de piaci részesedésük jelentősen kisebb.

Az FPGA elnevezés egy olyan eszközre utal, mely „field-programmable”, azaz a felhasználás helyén a felhasználó által programozható „gate array”, azaz a szilíciumlapkán logikai cellák egy kétdimenziós előredefiniált tömbje található. Az FPGA-k az elmúlt 30 év során az első lapkák megjelenésétől kezdve a napjainkban elterjedt korszerű architektúráig hatalmas fejlődésen estek át: sokkal komplexebbek, sokoldalúbbak és hatékonyabbak lettek, viszont az alapelvek változatlanok. Először ezeket az alapelveket fogjuk áttekinteni, majd pedig közelebbről is megvizsgálunk egy korszerű architektúrát.

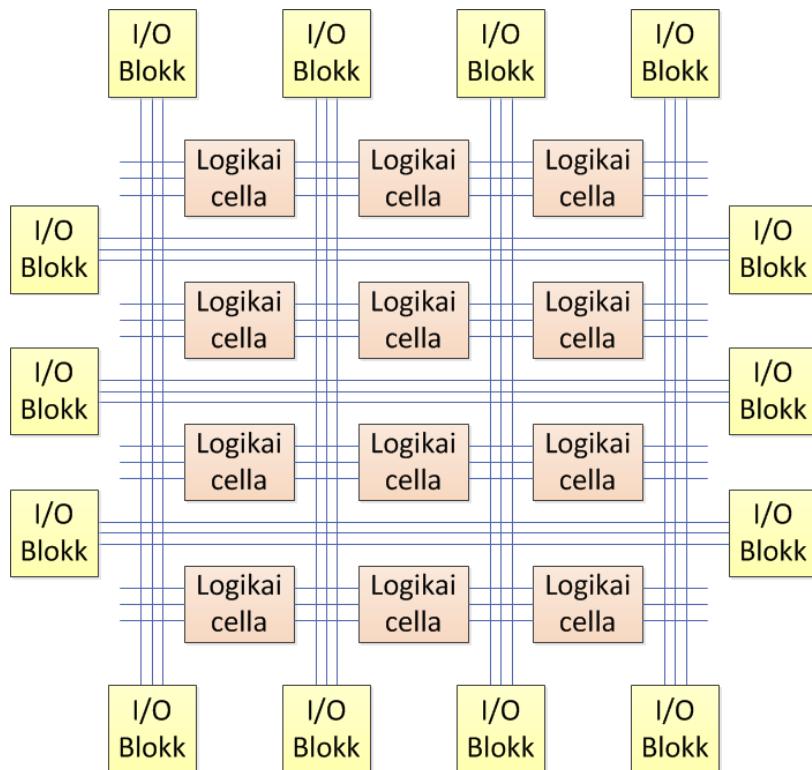
Az FPGA-k általános blokkvázlatát az 1.8. ábra mutatja. Három fő részből épülnek fel:

- általános célú logikai blokkok vagy logikai cellák,
- speciális funkciót (pl. I/O) ellátó dedikált blokkok,
- belső huzalozás, kapcsolómátrixok.

Az általános célú logikai blokkok lehetőséget nyújtanak kombinációs és szekvenciális digitális hálózatok megvalósítására. A speciális funkciót ellátó dedikált blokkok az adott speciális funkcionalitást támogatják, nem pedig általános digitális hálózat realizálására szolgálnak. Például ilyen dedikált blokk az I/O blokk, melynek speciális funkciója a már CPLD-knél megismert I/O konfiguráció. A korai FPGA-k esetén még csak az I/O blokk volt az egyetlen speciális blokk, később megjelentek további dedikált blokkok, mint például:

- órajel-menedzsment blokk főként órajel-szintézisre és kondicionálásra,
- memória blokk adatok tárolására,
- hardveres szorzó blokk,
- DSP szelet a jelfeldolgozási feladatok tipikus műveleteivel,
- memóriavezérlő külső (pl. DDRx) memória illesztésére,
- Ethernet MAC a hálózati kommunikáció támogatására,
- PCI Express blokk a rendszerszintű kommunikáció támogatására,
- processzor a bonyolultabb szekvenciális műveletsorozatok ellátására,
- multi-gigabites adó-vevők nagysebességű illesztésekhez,

- A/D konverter analóg jelek illesztéséhez és rendszermeddzs-



6.8. ábra: *FPGA általános blokkvázlat*

ment blokk.

Mint látható vannak olyan dedikált blokkok, melyek funkcionalitása megvalósítható lenne az általános célú logikai blokkok felhasználásával (például memória blokk, szorzó blokk). Viszont mivel ezek tipikusan nagyon sok alkalmazás által igényelt funkciók, ezért döntöttek úgy, hogy ne az általános logikát kelljen „pazarolni”, hanem álljanak azonnal a felhasználó rendelkezésére. Ennek előnye, hogy tipikusan kisebb helyen elfér az adott

funkcionalitás a szilíciumlapkán, illetve hogy ugyanazt a funkciót sokkal kisebb késleltetéssel, azaz gyorsabb működéssel képesek nyújtani, mint amit az általános célú logikával el tudnánk érni, hiszen a teljes funkcionalitás közvetlenül, dedikáltan, a programozható összeköttetések és azok késleltetései nélkül kerül kialakításra a lapkán. Ettől függetlenül persze, ha például vannak dedikált szorzó blokkok egy FPGA-ban, ugyanaz a szorzás funkció továbbra is megvalósítható az általános célú logika felhasználásával, bár erre leginkább olyankor kerül sor, ha már elhasználtuk a rendelkezésre álló dedikált blokkokat. A belső huzalozás és a kapcsolómátrixok felelősek az általános célú logikai blokkok, illetve a dedikált blokkok közötti programozható összeköttetések biztosításáért. Mivel a megvalósítandó feladat nagymértékben befolyásolja a kialakítandó huzalozást, azaz az egyes blokkok egyes vezetékeinek az útvonalát, ezért bonyolultabb a huzalozása, mint egy CPLD-nek, ahol csak egy központi, globális huzalozás létezik. A CPLD központi huzalozásának köszönhetően egyszerűen számolható, determinisztikus időzítéssel rendelkezik. Ezzel szemben az FPGA a bonyolult huzalozásának köszönhetően az eltérő útvonalhosszakból eredő eltérő jelkésleltetések miatt huzalozás utáni időzítés analízisre (STA: Static Timing Analysis) szorul, mellyel ellenőrizhető, hogy az egyes pontok között nem lép-e fel a megengedettnél nagyobb késleltetés, ami már befolyásolná a tervezett digitális rendszer működését.

6.3.1 FPGA technológiák

A programozható összeköttetések összességének egy lehetséges állapotát az FPGA konfigurációjának nevezzük. A tervezés során célunk az FPGA konfigurálása, programozása annak érdekében, hogy a kívánt digitális rendszer

valósuljon meg a lapka belsejében a programozható összeköttetések révén.

Az FPGA-k lehetnek egyszer programozhatóak (OTP) vagy többször programozhatóak. Önmagában ez a két lehetőség is különböző technológiákat feltételez. Az egyszer programozható FPGA-k úgynevezett „antifuse” technológiát alkalmaznak. A többször programozhatóak pedig lehetnek SRAM alapúak, SRAM alapúak belső flash memóriával, illetve flash alapúak. Ebben az alfejezetben röviden áttekintjük ezeket a technológiákat.

Antifuse FPGA-k

Az egyszer programozható FPGA-kban a programozható összeköttetések megvalósítására úgynevezett antifuse technológiát alkalmaznak. Az antifuse szó a fuse (biztosíték) ellentéte. Ahogyan egy biztosíték alapállapotában vezet és kiégethető, az antifuse pont ennek a fordítottja, azaz alapállapotban a programozható összeköttetések helyén szakadás található és a programozás során csak a kívánt helyeken létesítenek összeköttetéseket.

Ennek a technológiának az előnye a kis fogyasztás, illetve a zajjal, sugárzással szembeni ellenállóság, ami miatt akár műholdak és ūrszondák potenciális alkatrészei lehetnek. Hátránya, hogy az FPGA csak egyszer programozható, így ha módosítani szeretnénk a tervezett áramkörön, akkor másik FPGA-ra van szükségünk, tehát prototípus-fejlesztésre nem alkalmas. Az antifuse FPGA-k a piac egy szűkebb szegmensét képviselik (pl. Microsemi® Axcelerator® FPGA-k).

SRAM alapú FPGA-k

Az SRAM alapú FPGA-k esetén a programozható összeköttetések állapota SRAM cellákban kerül eltárolásra, így a tápfeszültség bekapcsolása után ezek az értékek bármikor felülírhatóak, azaz az FPGA újraprogramozható. A tápfeszültség megszűnésevel egyúttal az SRAM cellákban tárolt konfiguráció is

elvész, tehát minden bekapcsolás az FPGA konfigurálásával kezdődik. Ahhoz, hogy ez megtörténhessen, a konfigurációt valahol tárolni kell, ezért található sokszor egy külső flash memória az FPGA mellett. Az SRAM-alapú FPGA-k megfelelő rugalmasságot és a lapkán elérhető legjobb alkatrész-sűrűséget nyújtják, így napjainkban a legtöbb FPGA ezt a technológiát alkalmazza és a további alfejezetekben mi is ezen FPGA-kra fogunk koncentrálni.

Az SRAM alapú FPGA-k esetén az FPGA programozása egy konfiguráció, azaz nyers bitfolyam (bitstream) letöltését jelenti az eszközbe, mely tartalmazza az összes lehetséges programozható összeköttetés állapotát.

SRAM alapú FPGA-k belső flash memoriával

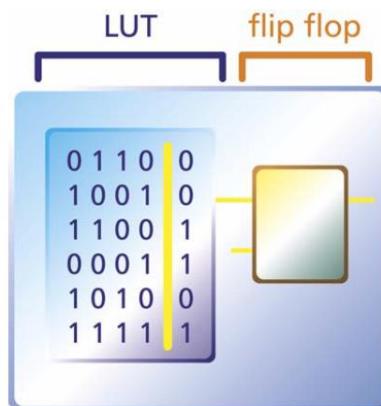
Az SRAM-alapú FPGA-k által biztosított rugalmasság a külső flash memória szükségessége miatt az integráltságot rontja. Abban az esetben, ha ez egy fontos szempont, akkor a gyártók ajánlanak olyan megoldást, ahol az FPGA tokjába integrálják a konfiguráció tárolásához szükséges flash memóriát is (pl. Xilinx® Spartan-3AN FPGA család). Tehát az FPGA továbbra is SRAM alapú, viszont a tokon belüli flash memóriából történik a konfiguráció. Ez a megoldás olyan esetekben is előtérbe kerülhet, ahol fontos szempont, hogy illetéktelenek ne tudják „lehallgatni”, lemásolni vagy esetleg visszafejteni az FPGA-ba töltött konfigurációt.

Flash alapú FPGA-k

A flash alapú FPGA-k esetén a programozható összeköttetések állapota nem SRAM, hanem flash memóriacellákban kerül tárolásra. A technológia előnye az alacsonyabb fogyasztás, a zajjal, sugárzással szembeni jobb tűrés, illetve a konfigurációt ugyanúgy nem lehet „lehallgatni”, mint az előző esetben. Ezek az FPGA-k szintén csak a piac szűkebb szegmensét képezik (pl. Microsemi® IGLOO® FPGA család).

6.3.2 Logikai cella

Az 1.8. ábrán látható logikai cellák alkotják az FPGA-k általános célú logikai egységeit, melyek egyaránt alkalmasak kombinációs és sorrendi hálózatok realizálására. Az 1.9. ábra mutatja egy logikai cella legalapvetőbb építőelemeit. Tetszőleges kombinációs hálózat implementálására a hagyományos PAL vagy PLA struktúra helyett az úgynevezett LUT (Look-Up Table) szolgál, melyet a következő alfejezetben vizsgálunk meg. Annak érdekében, hogy sorrendi hálózatokat is megvalósíthassunk, minden logikai cella tartalmaz flip-flopot.



6.9. ábra: *FPGA logikai cella*

6.3.3 LUT

A LUT (Look-Up Table) szó egy általános fogalom. Először áttekintjük, hogy általánosságban mit jelent, majd pedig megnézzük, hogy az FPGA-kra miként került alkalmazásra.

A LUT általánosságban egy olyan táblát jelent, mely célja egy pont-pont

hozzárendelés, azaz egy függvénykapcsolat megvalósítása. A táblázat címzését a független változóval végezzük és a LUT minden sorába a megvalósítani kívánt leképezés általi függő értéket helyezzük így a táblából kiolvasható a független változó értékéhez tartozó érték.

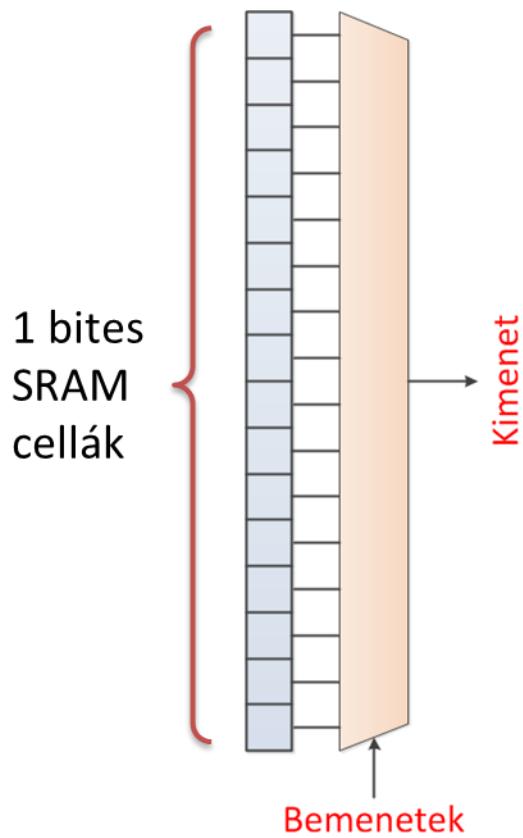
1. példa - 8 bites egész számok konstanssal való hardveres szorzása: ehhez a feladathoz építhetünk egy szorzáramkört vagy akár használhatunk LUT-ot is. Ebben az esetben a LUT a következő leképezést valósítja meg: $y \mapsto c \cdot x$, ahol c a konstans. Mivel x 8 bites egész érték, ezért összesen $2^8 = 256$ lehetséges y értéket fog tartalmazni a táblázat, melyeket előre kiszámolunk és tárolunk. A LUT 8 bites címvonalát közvetlenül az x változó címzi, így minden lehetséges x -re a LUT kimenetén megkapjuk a helyes $c \cdot x$ értéket.

2. példa - szinusz jel digitális szintézise: a szinusz függvény értékeit adott felbontással megkaphatjuk a $\sin(x)$ Taylor-sorában szereplő első néhány tag közvetlen hardveres realizálásával vagy akár használhatunk LUT-ot is. Ekkor a LUT a következő leképezést valósítja meg: $y \mapsto \sin(x)$. Tegyük fel, hogy az x egy 12 bites fixpontos törtszám. Ebben az esetben $2^{12} = 4096$ lehetséges x értékre számoljuk ki és tároljuk el a LUT-ba a $\sin(x)$ értékét. A LUT 12 bites címvonalát közvetlenül az x változó címzi, így minden lehetséges x értékre a LUT kimenetén megkapjuk a helyes $\sin(x)$ értéket.

3. példa - kombinációs hálózat realizálása: egy logikai függvénytel megadott kombinációs hálózatot megvalósíthatunk kapuáramkörökkel vagy akár használhatunk LUT-ot is. Ekkor a LUT a következő leképezést valósítja meg: $y \mapsto (A, B, C \dots)$, ahol A, B és C a logikai függvény változói. Tegyük fel, hogy 3 változós logikai függvényt szeretnénk realizálni, azaz csak A, B és C változóink vannak. Ebben az esetben a logikai függvény diszjunktív kanonikus

alakja közvetlenül megadja a $2^3 = 8$ lehetséges kombinációhoz tartozó y értéket, melyeket eltárolunk a LUT-ban. A LUT 3 bites címvonalát közvetlenül az A, B és C 1-1-1 bites változókkal címezzük, így a 3 változó minden lehetséges kombinációja esetén a LUT kimenetén megkapjuk a helyes y értéket (ami 0 vagy 1).

Vegyük észre, hogy ebben valójában semmi újdonság nincs: a kombinációs hálózat LUT-os megvalósítása tulajdonképpen szinonimája a digitális technikában tanult multiplexerrel való univerzális kombinációs hálózat



6.10. ábra: *FPGA LUT*

realizálásnak. Azaz a kombinációs hálózat bemenetei egy multiplexer címbemeneteihez kapcsolódnak, a MUX egyes adatbemeneteit a logikai függvény által előírt fix '0'-ra vagy '1'-re kötjük és ekkor bármely bemeneti kombináció esetén a MUX kimenetén megjelenik a kombinációhoz tartozó y érték.

Az FPGA-kban lévő LUT (1.10. ábra) tulajdonképpen pontosan a 3. példában vázolt módszer gyakorlati alkalmazása, azaz kombinációs hálózat multiplexerrel történő megvalósítása. Mivel biztosítani kell az újraprogramozhatóságot, ezért a MUX adatbemenetei nem fixek, hanem SRAM cellákhoz kapcsolódnak. Így az SRAM cellákban tárolt értékek határozzák meg, hogy egy LUT milyen logikai függvényt valósítson meg. Tehát SRAM alapú FPGA-k esetén nem csak a programozható összeköttetések, hanem a LUT-okban lévő SRAM cellák értékei is konfigurálásra kerülnek.

6.3.4 Alkalmazási területek

Az FPGA-k megjelenésük óta a töretlen innovációjuknak köszönhetően:

- A gyártástechnológia úttörői (az Intel® processzorokkal egyetemben)
- Egyre összetettebbek, azaz minden újabb generáció több és kifinomultabb erőforrást kínál a mérnököknek, akik így egyre nagyobb és összetettebb rendszert tervezhetnek meg velük.
- Nem csak a felhasználható erőforrások száma növekszik, hanem újabb, a kor igényeit kielégítő erőforrások is helyet kapnak

- Speciális követelményeket is kielégítő variánsokat is terveznek, mint például a hadseregnak szánt *Defense*, vagy az űrszondákra, műholdakra tervezett *Space* minősítésű FPGA-k.

Ezek mind hozzájárultak ahhoz, hogy napjainkban a legelterjedtebb programozható logikai eszközökkel váljanak. Felmerül a kérdés, hogy mikor alkalmazzunk FPGA-t? Tipikusan olyan esetekben, amikor nagy számításigényű, jól párhuzamosítható feladatot kell elvégeznünk és/vagy az integráltságot szeretnénk növelni, s nem célunk akkora kihozatalt elérni, hogy gazdaságos legyen egyedi céláramkört (ASIC) tervezni és gyártatni.

A teljesség igénye nélkül összefoglalunk néhány alkalmazási területet, ahol FPGA-kkal találkozhatunk:

- (Nagy számítási teljesítményű) jelfeldolgozási feladatok esetén digitális szűrők, digitális modulátorok, adaptív algoritmusok, gépi látás területe.
- Vezetékes kommunikáció megvalósítása, mely napjainkban tipikusan 10, 40, 100, 400 Gbit/s, ill. 1Tb/s átviteli sebességű switchelek, routerek, csomagfeldolgozó (pl. szűrő) egységeket jelent.
- Vezeték nélküli kommunikáció során a 3G / 4G / 5G mobilhálózatokban szükséges nyalábformálási és MIMO algoritmusok megvalósítása.
- Digitális műsorszórás területén video switch-ekben, valósidejű enkóderekben, dekóderekben, transzkóderekben.
- Tudományos számítások, kutatási projektekben (pl. CERN LHC, SETI).

- Orvosi képfeldolgozás és bioinformatika területén: CT, MRI, PET, mintafelismerés, fehérjemodellezés.
- HPC: High Performance Computing, azaz a nagy számítási teljesítményű számítástechnikában.
- Adatközpontokban.
- ASIC emuláció során.
- Katonai és védelmi alkalmazásokban: éjszakai látás, célkövetés, képstabilizálás, radar és sonar.
- Ūrtechnológiai alkalmazásokban: műholdak és ūrszondák.
- Ipari, illetve autóipari felhasználás tipikusan alakfelismerés, vezető nélküli autók, és kommunikációs feladatokban.
- Fogyasztói cikkek esetén: kamera, TV, Set-top Box.

Beágyazott rendszerekben tipikusan olyan esetekben alkalmazzák, amikor szükség van nagy számítási teljesítményre, és a feladat is jól párhuzamosítható, illetve az integráltságot szeretnék növelni több funkcionálitás egy lapkán való elhelyezésével.

További szempontok lehetnek még, melyeket bizonyos FPGA-k szintén kielégíthetnek:

- Speciális környezeti feltételek között való alkalmazás.
- Biztonságkritikus működés.
- Minimalizált energiafogyasztás.

Az implementáció rugalmasságát tekintve a legrugalmasabb eszközöknek tekinthetők, hiszen nem csak akár az FPGA-ban megvalósított processzoron futó szoftver, de a teljes hardver is bármikor módosítható, továbbfejleszthető.

6.3.5 Xilinx® Series-7 FPGA-k

Az alapelvek tárgyalása után közelebbről is megvizsgálunk egy napjainkban korszerű architektúrát, a Xilinx® 7. generációs eszközeit. Egy új generáció tipikusan fejlettebb gyártástechnológiát, fejlettebb, kifinomultabb, az aktuális igényeket kielégíteni igyekvő architektúrát, több felhasználható erőforrást, illetve új erőforrások megjelenését jelenti.

A gyártók megkülönböztetnek FPGA családokat (néhol további alcsaládokat), illetve családon belüli konkrét típusokat. Egy család minden tagja a családra jellemző célt szolgálja. A 7-es sorozaton belül három családot különböztet meg a Xilinx®, ezek az Artix-7, Kintex-7 és Virtex-7 FPGA-k. A 7-es sorozat nagy előnye, hogy minden család egységes architektúrával rendelkezik, így például egy Artix-7-re készített terv különösebb erőfeszítés nélkül portolható egy Kintex-7 vagy Virtex-7 családra.

Az Artix-7 család tagjai a 7-es sorozaton belüli legkisebb FPGA-k. Ez a család az alacsony költségre és a kisebb fogyasztásra fókusztál miközben alapvető feladatokhoz nyújt elegendő számítási teljesítményt. Ennek szöges ellentéte a Virtex-7 család, mely a sorozaton belüli legnagyobb FPGA-kat tartalmazza és a lehető legnagyobb számítási teljesítményre koncentrál. Itt az ár és a fogyasztás csak másodlagos tényezők. A Kintex-7 család pedig az arany középút: jelentősen nagyobb számítási teljesítményt kínál, mint az Artix-7 (de kevesebbet, mint a Virtex-7), kicsit nagyobb fogyasztásért és magasabb áráért cserébe, így ezek az FPGA-k már nagyobb rendszerek fejlesztésére is költséghatékonyan alkalmazhatóak.

A Virtex-7 családon belül három további alcsalád található:

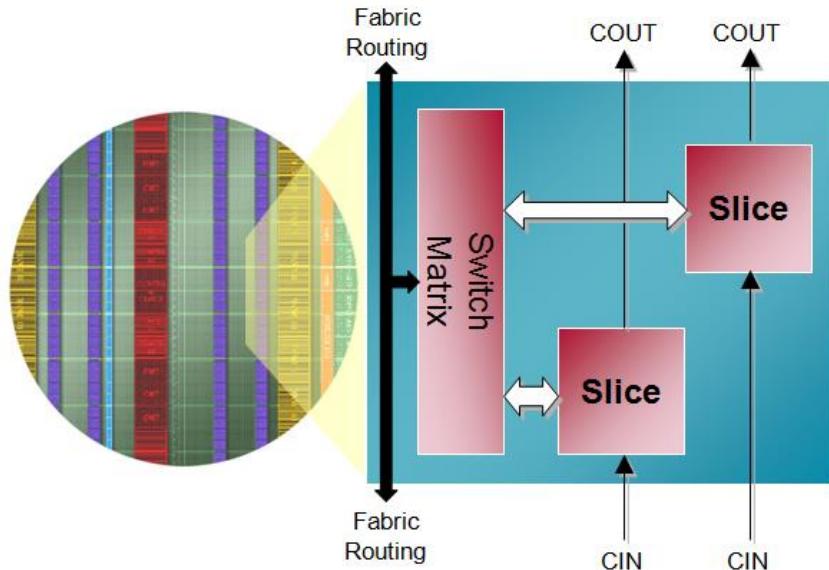
- Virtex-7 T: a legtöbb logikai erőforrás ASIC prototípus-fejlesztésekhez (2 millió logikai cella).
- Virtex-7 XT: nagyteljesítményű jelfeldolgozási feladatokra (5,3 TMAC).
- Virtex-7 HT: nagysebességű kommunikációs eszközökhöz ($n \times 100$ Gb/s, ill. 400 Gb/s protokollok).

A 7-es sorozat egyes FPGA-i minden családon, ill. alcsaládon belül a logikai cellák számában, illetve a különböző dedikált blokkok számában térnek el egymástól.

A Xilinx terminológiájában CLB-nek (Configurable Logic Block) nevezik az általános célú logikai cellákat. Egy CLB felépítését mutatja az 1.11. ábra. minden CLB két szeletet (Slice) tartalmaz, melyek kapcsolómátrixon keresztül csatlakoznak a belső huzalozáshoz. A Slice-okban lehetőség van aritmetikai áramkörök kialakítására is, a CIN (Carry IN) és a COUT (Carry OUT) a szomszédos Slice-okhoz csatlakozó átvitel-továbbító vonalak.

Kétféle Slice került kialakításra:

- Slice_L: egyszerűbb Slice kevesebb funkcionalitással
- Slice_M: teljes értékű Slice minden lehetséges funkcióval.



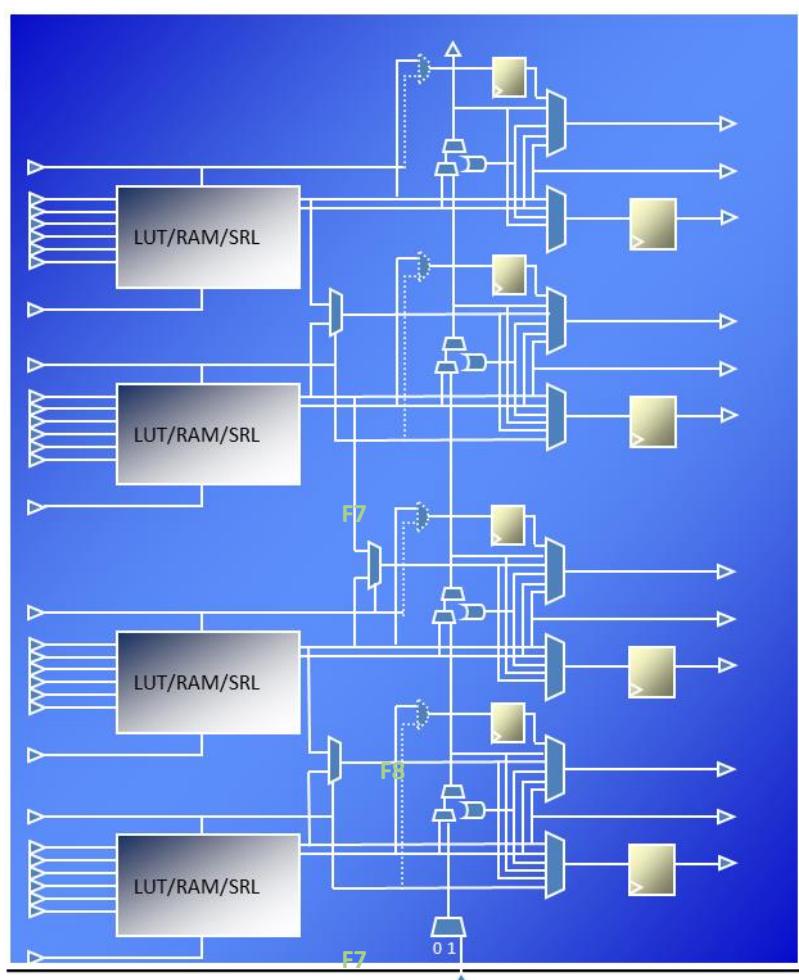
6.11. ábra: Xilinx Series-7 CLB

Nem nehéz olyan alkalmazásokat elképzelni, ahol jelentős mennyiségű kombinációs logikára van szükség, vagy akár olyat ahol sok memóriaelémre. Abban az esetben, ha a rendszerterv sok tárolóelemet tartalmaz miközben kombinációs logikát alig, az FPGA-ban lévő LUT-ok kihasználatlanul maradnának, holott minden LUT-hoz tartoznak az adott LUT konfigurációját tároló SRAM cellák, melyek ilyen esetekben felhasználhatóak lehetnének memóriaelémekként. Sőt kis módosítással nem csak egyszerű memóriaelémekként, hanem akár a szomszédos 1 bites SRAM cellák összekapcsolásával shift-regiszterként.

Ezért a Xilinx® FPGA-kban nem csak egyszerű LUT-ok találhatóak, hanem úgynevezett Function Generator-ok (FG), melyek a megvalósítani kívánt rendszer igényeinek megfelelően konfigurálhatóak és felhasználhatóak akár

LUT-ként, memóriacellaként vagy shift-regiszterként is. Az FPGA konfigurálásakor eldől minden FG esetén, hogy a három lehetséges funkció közül melyiket fogja megvalósítani. A Function Generator-ra LUT/RAM/SRL-ként is hivatkoznak, utalva ezzel a három lehetséges konfigurációra. A 7-es sorozatú FPGA-kban minden Slice_M szelet 4 db LUT/RAM/SRL egységet tartalmaz, ahogy az az 1.12. ábrán is látható. Ezért mondhatjuk, hogy ezek teljes értékű Slice-ok minden lehetséges funkcionalitással. Ezzel szemben a Slice_L szeletekben az FG-k helyett csupán LUT-ok találhatóak.

Minden LUT, ill. LUT-ként konfigurált FG 6 bites, ami azt jelenti, hogy egy LUT legfeljebb 6 bemenettel rendelkező kombinációs hálózat megvalósítására alkalmas. Egy FG-t memóriaelemként (RAM vagy ROM) konfigurálva legfeljebb $2^6 = 64$ bit tárolására nyújt lehetőséget. A Xilinx® terminológiájában annak érdekében, hogy a memóriaként konfigurált FG-eket megkülönböztessük a dedikált blokként integrált memóriáktól, különböző elnevezéssel hivatkoznak



6.12. ábra: Xilinx Series-7 Slice

rájuk. A dedikált memóriablokkokat Block RAM-nak (vagy röviden BRAM-nak) nevezzük, utalva arra, hogy dedikált blokként helyezkednek el a lapkán. Az FG-kból kialakítható memóriát Distributed RAM-nak (elosztott memóriának) nevezzük, hiszen nem egy dedikált területük van a lapkán, hanem Slice-okban, a kívánt mérettől függően akár több Slice-on átnyúlva, elosztva alakíthatóak ki. Annak érdekében, hogy ne csak legfeljebb 6 bemenetű kombinációs hálózatok legyenek megvalósíthatóak, minden Slice-on belül további multiplexerek találhatóak, melyek a LUT-ok kombinálásával segítik nagyobb hálózatok kialakítását. Az F7MUX primitívek 2 db LUT-ot fognak össze, ami által $2 \cdot 2^6 = 128$ lehetséges kombinációt fednek le, azaz 7 bemenetű kombinációs hálózatot képesek realizálni. A Slice-on belüli egyetlen F8MUX pedig a 2 db F7MUX kimeneteit fogja össze, így realizálva 8 bemenetű kombinációs hálózatot. Igény esetén további Slice-ok bevonásával még több LUT kombinálható össze nagyobb kombinációs hálózat megvalósítására.

A Slice-okban egy vertikális irányú gyors carry-továbbító lánc is helyet kapott, mely segítségével aritmetikai műveletek valósíthatóak meg hatékonyan. Azért indokolt egy ilyen lánc kialakítása, mert nagyon gyakoriak az aritmetikai műveletek: hiszen nem csak közvetlenül egy összeadó áramkör realizálásakor alkalmazhatunk aritmetikát, hanem például különböző számlálók esetén is, melyekben az inkrementálás / dekrementálás, mint aritmetikai művelet valósítható meg hatékonyan a lánc segítségével.

A tárolóelemeket tekintve minden Slice 8 db flip-flopot (FF) tartalmaz, melyek közül 4 db csak flip-flopként használható, míg a maradék 4 db konfigurálható flip-flopként avagy latch-ként való használatra.

Az általános logikát megvalósító CLB-ken kívül még számos dedikált blokkot

tartalmaznak a 7-es sorozatú FPGA-k.

6.4 További programozható logikai eszközök

Bár az FPGA vált a legnépszerűbb, legsikeresebbnek mondható programozható logikai eszközzé, ám a PLD-k fejlődése napjainkban sem állt meg. Nem csak az új FPGA generációk szárnyalják túl az előző generációk képességeit, hanem a technológia fejlődése lehetővé tette még összetettebb, egychipes rendszerek (SoC: System-on-Chip), illetve heterogén platformok megjelenését. Egychipes vagy egylapkás rendszernek olyan integrált áramkört tekintünk, ahol egy teljes számítógépes rendszer foglal helyet egy lapkán, beleértve a feldolgozó egység(ek)et (CPU, DSP), memóriát, illetve egyéb digitális és gyakran analóg, illetve rádiófrekvenciás komponenseket is. Ilyen lapkák például a Texas Instruments által szállított OMAP, illetve KeyStone SoC-k, melyek tartalmaznak egy vagy több magos alkalmazásprocesszort, és egyéb társprocesszorokat (például DSP vagy hálózati társprocesszor), memóriát, illetve bőséges perifériakészletet. Mind a Xilinx, mind az Altera szállít olyan lapkákat, melyekbe belekerült egy rugalmasan konfigurálható SoC, illetve FPGA terület. Ezelőtt is voltak próbálkozások olyan integrált áramkörök kialakítására, melyek ötvözni próbálták az FPGA technológiát valamilyen már meglévő processzorral. Már az ezredforduló táján a Xilinx dedikált blokként helyezett el az FPGA terület részeként egy vagy több népszerű PowerPC processzormagot. De más gyártók is próbálkoztak egyszerűbb megoldással, mint például az Atmel® az FPLIC (Field Programmable System Level Integrated Circuits) áramkörökkel, melyek egy lapkán tartalmaznak FPGA területet, SRAM-ot, illetve AVR mikrovezérlőt. Más irányt képviselnek a Cypress Semiconductor PSoC (Programmable System-

on-Chip) áramkörei, melyek alapgondolata, hogy a felhasználóknak egy olyan integrált lapkát kínálnak, mely tartalmaz egy teljes mikrokontrolleres alrendszeret, továbbá konfigurálható digitális és analóg perifériákat, közöttük programozható összeköttetésekkel.

Néhány évente akár egész eredeti ötletre építve fel-felbukkanak olyan új innovációk, melyektől azt reméli a tervező cég, hogy hasonló forradalmat vihetnek véghez a programozható logikai eszközök palettáján, mint amit az FPGA véghezvitt, bár eddig még ezen feltörekvésekből egy sem volt igazán sikeres.

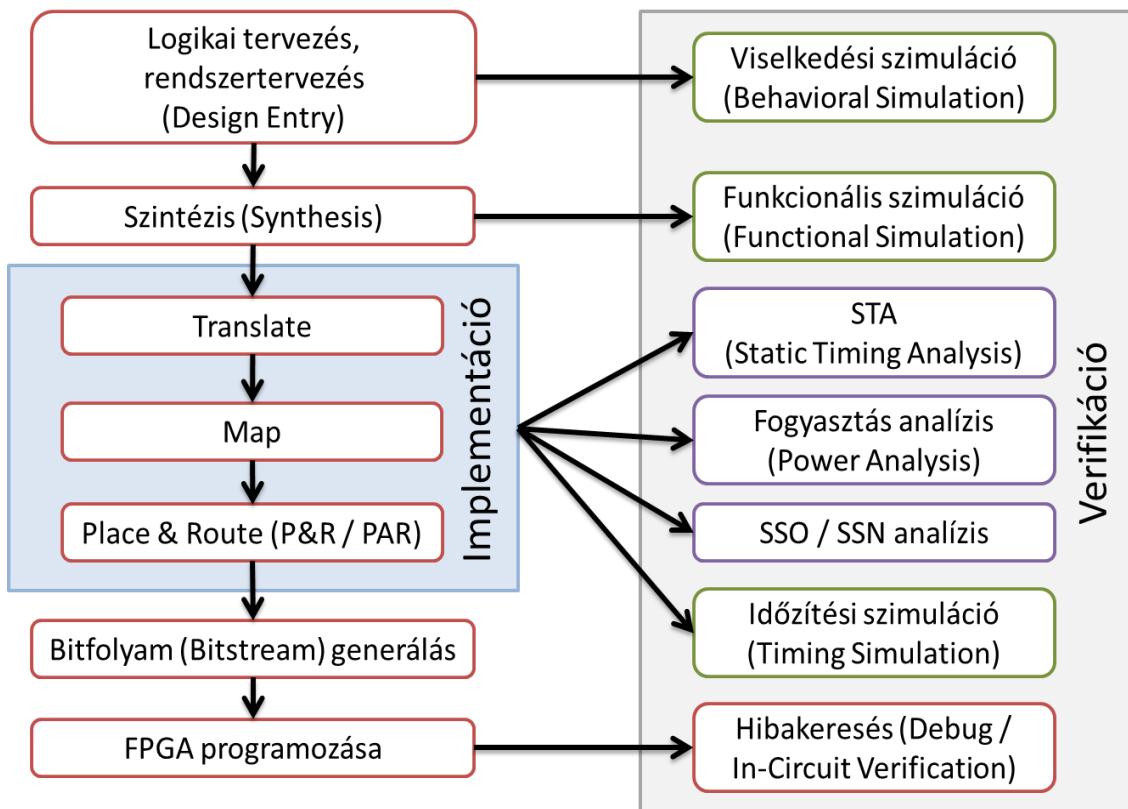
7 Fejlesztési lehetőségek

Minden rendszernek vagy részegységnek a tervezésekor, fejlesztésekor a követelmények, a költségek, a fejlesztési idő, a szükséges szakértelem, a beszállítói támogatás, stb. függvényében döntés születik arról, hogy az adott funkcionálitást milyen eszközzel valósítsák meg.

Mivel napjainkban a programozható logikai eszközök közül az FPGA tekinthető zászlóshajónak, ezért ebben a fejezetben az FPGA-kra fókuszálva áttekintjük azt a fejlesztési folyamatot, mely ezen eszközök felhasználásához szükséges. Itt kitérünk a rendszertervezés és a verifikáció lehetőségeire is.

7.1 Fejlesztési folyamat FPGA-val

A fejlesztés folyamatábráját a 2.1. ábra mutatja. Az ábra bal oldala a tervezési,



7.1. ábra: A fejlesztés folyamatábrája

implementálási lépéseket mutatja, melyek kötelező érvénnyel követik egymást. Az ábra jobb oldala az egyes tervezési, implementálási szakaszokhoz tartozó ellenőrzési (verifikációs) lehetőségeket mutatja. Ezek opcionálisak, azaz nem feltétlen szükségesek ahhoz, hogy az FPGA felprogramozható legyen, viszont nagymértékben ajánlott a használatuk, sőt nélkülözhetetlenek egy, az

elvárásoknak eleget tevő rendszer fejlesztéséhez.

7.1.1 Logikai tervezés, rendszertervezés

Egy rendszer tervezésekor az első feladatok egyike az egymástól jól elkülöníthető funkciók azonosítása, és ezek önálló tervezése, hiszen a legtöbb rendszer túl komplex ahhoz, hogy egyetlen funkcióként, modulként könnyen kezelhető legyen. Azt a stratégiát, ahol a rendszert kisebb alrendszerre bontjuk, melyeket szükség esetén addig finomítunk tovább, míg a legelembb építőkövekhez nem érkezünk, felülről lefelé tervezésnek (top-down design) nevezünk. A legkisebb modulok tervezése során csak az adott modulra kell koncentrálnunk, a rendszer többi részét figyelmen kívül hagyhatjuk. Amint a kisebb modulok elkészültek, a rendszer hierarchiájában elfoglalt helyükre kell illeszteni őket. A felülről lefelé tervezés előnye a felesleges részletek elrejtése mind az egyes modulok tervezése, mind az integráció során. A legfontosabb tervezői kérdés pedig az egyes funkciók lehető legjobb körvonalazása és az építőelemek közötti interfész precíz meghatározása.

Ezzel szemben a lentről felfelé tervezés (bottom-up design) stratégiával először a legelembb építőkövek kerülnek meghatározásra, majd ezekből építünk nagyobb egységeket mindaddig, míg végül összeáll a teljes rendszer.

A valóságban a legtöbb projekt esetében együttesen alkalmazzák mind a felülről lefelé tervezést, hiszen a rendszerkövetelmények először a legmagasabb szinten kerülnek specifikálásra, mind pedig a lentről felfelé tervezést, mert általában már vannak komponensek, melyek készen (vagy kis módosítással) rendelkezésre állnak.

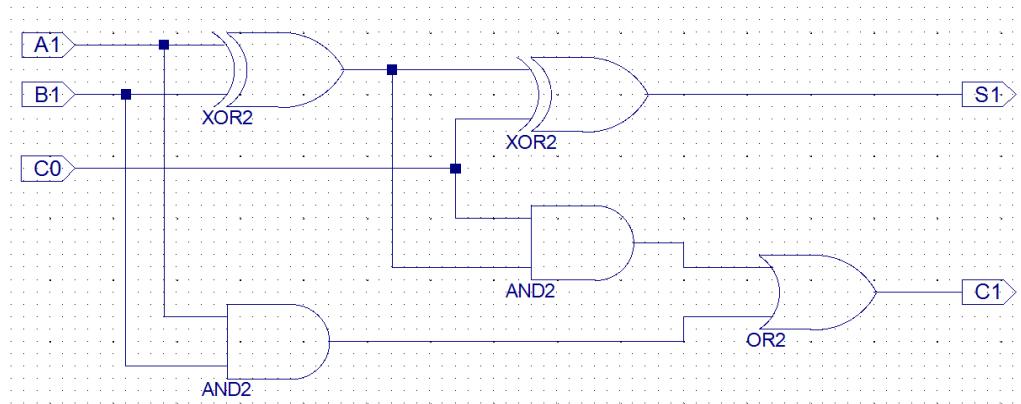
Az olyan újrahasznosítható hardvertervet, mely egy jól definiált funkciót valósít

meg és a külvilág felé jól definiált interfésszel rendelkezik, IP¹⁹ core-nak, magyarul a tervezők szellemi alkotásának nevezzük. Egy ilyen IP lehet akár egy cég licenszelhető terméke (például egy ARM processzor IP), valamelyen ingyenes és általában nyílt forrású IP vagy akár saját tervezésű. Az FPGA gyártók is számos IP-t kínálnak a tervezőknek, melyek egy része ingyenesen elérhető a fejlesztőkörnyezetekben, mások megvásárolhatóak.

Mivel egy digitális rendszer leguniverzálisabb tervezési módszere a hardverleírás használata valamelyen hardverleíró nyelv (pl. VHDL, Verilog) segítségével, ezért az IP-k is valamelyen hardverleírás formájában állnak rendelkezésre, mely lehet olvasható szöveges leírás vagy titkosított leírás, melyre az IEEE szabványt is kidolgozott (IEEE P1735). Attól függően, hogy az adott IP milyen realizációra készül, beszélhetünk hard IP-ról, illetve soft IP-ról. Hard IP esetén dedikáltan kerül kialakításra a funkció a lapkán (ilyen egy ASIC-re tervezett IP, mint például egy FPGA dedikált blokkja), soft IP esetén pedig az FPGA általános logikájának felhasználásával kerül az IP kialakításra. Abban az esetben, ha egy felhasználni kívánt IP egy előző projektből származik, IP újrafelhasználásról (IP reuse) beszélünk.

¹⁹ Intellectual Property

Egy egység tervezésére többféle lehetőség létezik. Egyszerűtől hagyományos módszert alkalmazva meg lehet rajzolni a logikai kapcsolást egy kapcsolási rajz szerkesztőben. Ennek előnye, hogy a logikai tervezést leszámítva csak a kapcsolási rajz szerkesztő ismerete szükséges hozzá, viszont hátránya, hogy nagyon rugalmatlan és a komplexitás növekedésével meglehetősen időigényessé válik a munka. A sematikus ábrán való tervezésre példaként egy 1 bites teljes



7.2. ábra: Logikai tervezés sematikus ábrán

összeadót mutat a 2.2. ábra.

Egy másik lehetőség valamelyen hardverleíró nyelv szintaktikai és szemantikai szabályainak betartásával egy egyszerű szöveges állományban hardverleírás készítése. Hátránya, hogy mindenépp legalább egy hardverleíró nyelv alapos ismerete szükséges hozzá, viszont cserébe nagyfokú rugalmasságot és a fejlesztési idő jelentős csökkenését nyerhetjük. Az előbbi 1 bites teljes összeadó

```
library ieee;
use ieee.std_logic_1164.all;

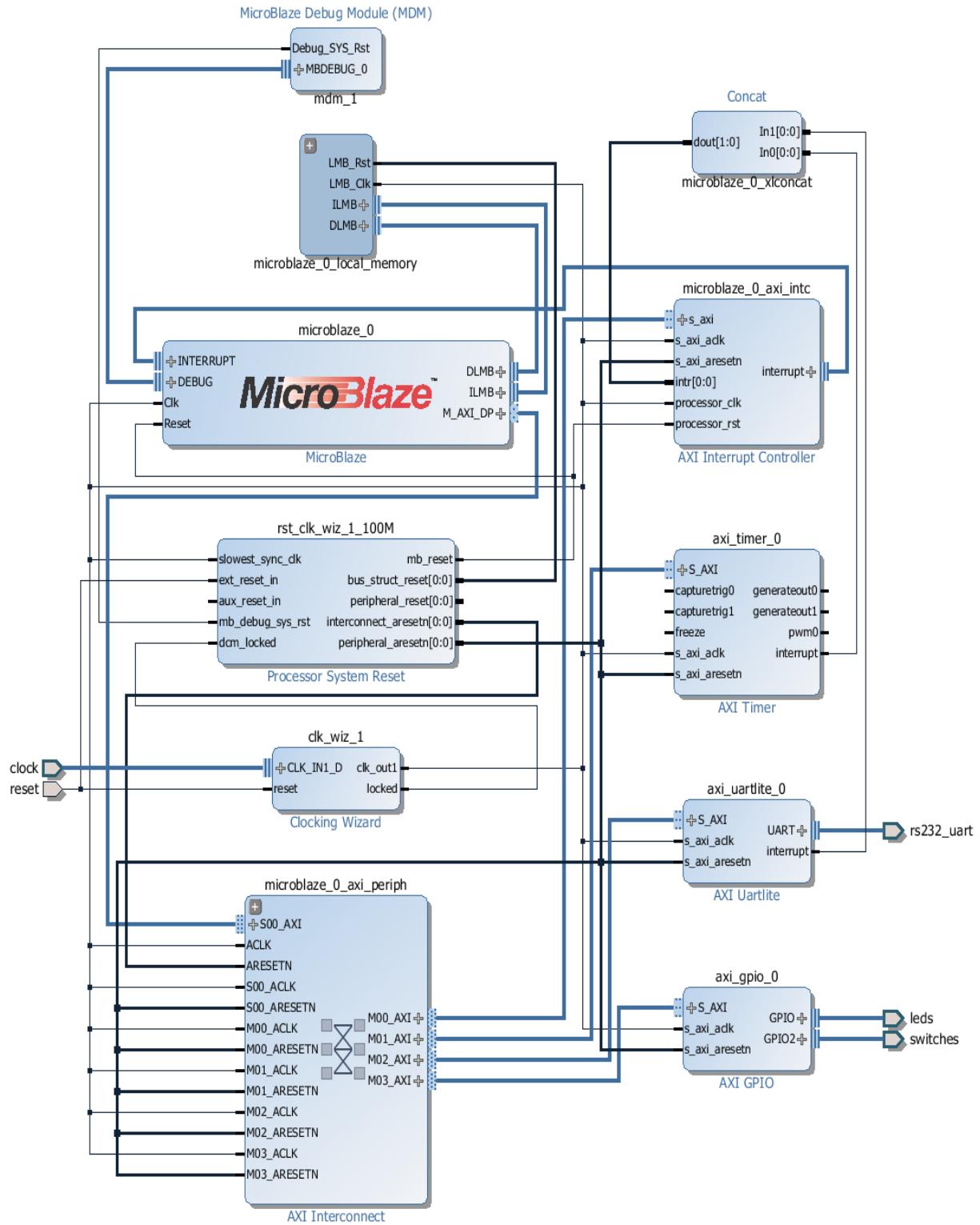
-----
entity full_add is
  port (
    a1 : in std_logic;
    b1 : in std_logic;
    c0 : in std_logic;
    s1 : out std_logic;
    c1 : out std_logic
  );
end full_add;
-----

architecture dataflow of full_add is
  signal sum_a1_b1 : std_logic;
begin
  sum_a1_b1 <= a1 xor b1;
  s1 <= sum_a1_b1 xor c0;
  c1 <= (a1 and b1) or (sum_a1_b1 and c0);
end dataflow;
```

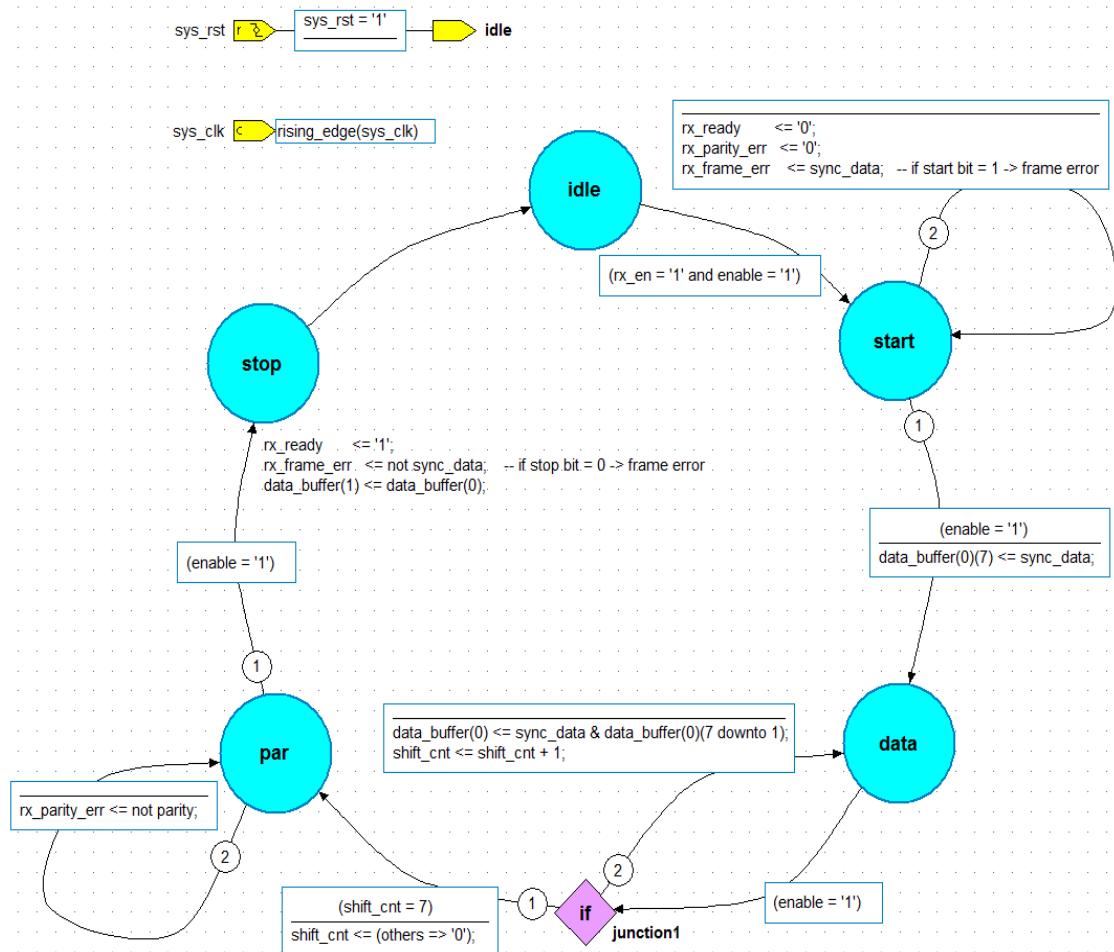
7.3. ábra: Logikai tervezés hardverleírással

VHDL nyelvű leírása látható a 2.3. ábrán.

Annak ellenére, hogy a sematikus ábrán való logikai kapcsolás rajzolása, mint tervezési módszer mára teljesen elavulttá vált, a rendszertervezés magasabb szintjein továbbra is van a sematikus ábráknak létjogosultsága. Itt olyan nagyobb komponensek, mint például egy vagy több processzor, perifériák és különböző feldolgozóegységek közötti összeköttetések jól láthatóan és könnyedén megrajzolhatóak, ami akár a dokumentáció részét is képezheti. Továbbá a



7.4. ábra: IP centrikus rendszertervezés sematikus ábrán

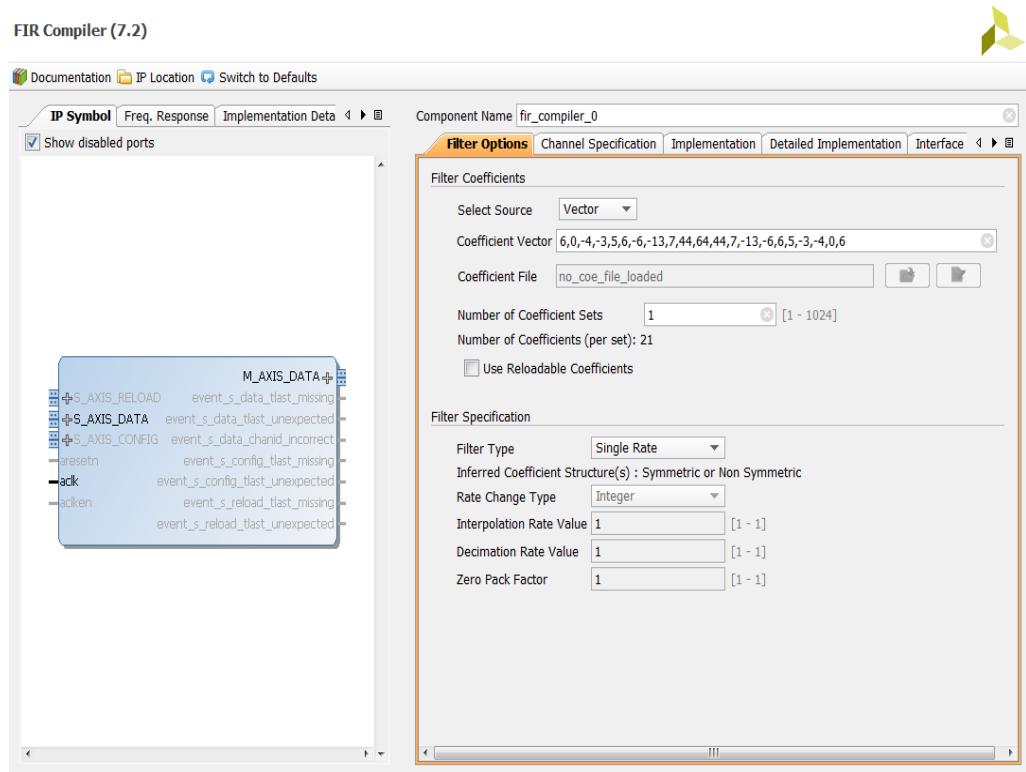


7.5. ábra: Állapotgép tervezése

ellenőrzőeszköz segíti a gyors fejlesztést (2.4. ábra).

Állapotgépek tervezésére állapotdiagram szerkesztővel is van lehetőség, ami szintén a fejlesztési időt csökkentheti. Továbbá a vizuális megjelenítés gyakran jó áttekinthetőséget nyújt és akár közvetlenül a dokumentáció részét is képezheti. A leggyakrabban használt műveletek, funkciók számára a fejlesztőkörnyezet az

igényekhez illeszkedő IP-t képes generálni néhány paraméter megfelelő beállítása alapján. A lehetőségek széles skálán mozognak az egyszerű számlálóktól és aritmetikai műveletvégzőktől az órajel-szintézisen és a digitális szűrökön, modulátorokon át a népszerű kommunikációs interfésekig. A 2.6. ábra egy digitális FIR szűrő IP konfigurációs lehetőségeinek egy kis részletét mutatja. Hasonlóan más IP is a felhasználás előtt testre szabható.



7.6. ábra: FIR szűrő IP konfigurációs felületének részlete

Különböző számítási feladatok során gyakran alkalmazunk modellek, ahol a modell és a modell paramétereinek finomhangolásával a megfelelő szimulációs környezetben kiértékelhető a működés. Például egy digitális szűrési feladat

során készíthetünk egy Matlab Simulink modellt, ahol a feladathoz igazíthatjuk a szűrő típusát, fokszámát és együtthatóit. Ilyenkor hasznos, ha a modellből közvetlenül a fejlesztőkörnyezet képes HDL kódot generálni. Ezt a módszert modellalapú fejlesztésnek nevezzük. Természetesen kézzel is megírható a hardverleírás, viszont magasszintű modell nélkül, egyszerű logikai szimulációval a legtöbb esetben elég nehézkes a helyes működés kiértékelése. Előfordulhat olyan eset, hogy már rendelkezésre áll a feladathoz szükséges egy-egy függvény C nyelven, melyet egy megelőző projektben való verifikációja alapján helyesnek vélünk és szükséges válik ugyanennek a funkciónak a hardveres implementálása. Ebben az esetben akár kézzel újraírhatjuk a függvényt egy választott hardverleíró nyelv alkalmazásával, ami egyrészt fárasztó, másrészt a hibázás lehetőségét rejti magában, ami miatt a HDL implementációt is ugyanolyan kimerítően verifikálni szükséges, mint az előzőleg megírt C nyelvű kódot. Ezt elkerülendően ma már vannak olyan szoftverek, melyekkel lehetőség van az úgynevezett magasszintű szintézis alkalmazására, ahol egy C vagy C++ nyelvű realizációnak több-kevesebb módosításával automatikusan generálható a hardverleírás.

Bármilyen tervezési módot is választunk a rendszer vagy annak részegységeinek fejlesztésére, első lépésként a tervből a fejlesztőkörnyezet úgynevezett szintetizálható hardverleírást generál, mely a fejlesztés következő lépéseinak, a szintézisnek a bemenete. A szintetizálható hardverleírás annyit jelent, hogy a HDL kód kizárálag olyan nyelvi konstrukciókat tartalmaz, melyek hardveresen implementálhatóak akár ASIC vagy FPGA technológiát alapul véve, azaz nem tartalmaz semmilyen szimulációt segítő nyelvi elemet.

7.1.2 A fejlesztés további lépései

Visszatekintve a 2.1. ábrára, a rendszertervezés, illetve logikai tervezés után a szintézis következik. A szintézis az az eljárás, ahol az ún. szintézér a HDL-ben megírt logikai tervből egy huzalozási listát (netlist) állít elő, mely egységes formában tartalmazza a felhasznált primitíveket és a közöttük lévő összeköttetéseket.

Mind a szintézishez, mind a következő implementációs lépésekhez lehetőségünk van különböző kényszerek, megkötések (constraint) megadására is, melyek akár a fejlesztőkörnyezet grafikus felületén vagy akár kézzel szöveges állományba is bevihetők. Ilyen megkötés például, hogy a logikai terv be- és kimenetei mely FPGA lábakhoz legyenek társítva. Ennek alkalmazása elengedhetetlen egy működőképes terv esetén, hiszen az FPGA-ba szükséges bizonyos jelek bevitelére, és a feldolgozás eredményének kiviteli.

A szintézis után az implementációs szakasz következik, melynek első lépéseként a szintetizált huzalozási listák egy egységgé való összefűzése történik meg (translate). Ezután az egyes FPGA primitíveket szükséges CLB-kbe, I/O blokkokba leképezni (map). Miután minden primitív elfoglalta a helyét, minden CLB-nek és dedikált blokknak a pontos helyét kell kijelölni (place) és közöttük az összeköttetéseket kialakítani (route).

A terv sikeres behuzalozása után egy FPGA-ba letölthető bitfolyamot generálunk, majd pedig egy speciális programozával az FPGA-t felprogramozzuk.

8 Melléklet

8.1 számú melléklet, IO lábak funkciói

Funkciók					Funkciók				
I/O láb	A	B	C	D	I/O láb	A	B	C	D
PA0	PWM0	TIOAO		WKUP0	PC0				
PA1	PWM1	TIOBO		WKUP1	PC1				
PA2	PWM2	SCK0	DATRG	WKUP2	PC2				
PA3	TWDO	NPCS3			PC3				
PA4	TWCK0	TCLK0		WKUP3	PC4		NPCS1		
PA5	RXDO	NPCS3		WKUP4	PC5				
PA6	TXDO	PCK0			PC6				
PA7	RTSO	PWM3			PC7		NPCS2		
PA8	CTSO	ADTRG		WKUP5	PC8		PWM0		
PA9	URXD0	NPCS1		WKUP6	PC9		PWM1		
PA10	UTXD0	NPCS2			PC10		PWM2		
PA11	NPCSO	PWM0		WKUP7	PC11		PWM3		
PA12	MISO	PWM1			PC12				AD12
PA13	MOSI	PWM2			PC13				AD10
PA14	SPCK	PWM3		WKUP8	PC14		PCK2		
PA15		TIOA1		WKUP14	PC15				AD11
PA16		TIOB1		WKUP15	PC16		PCK0		
PA17		PCK1		AD0	PC17		PCK1		
PA18		PCK2		AD1	PC18		PWM0		
PA19				AD2/WKUP9	PC19		PWM1		
PA20				AD3/WKUP10	PC20		PWM2		
PA21	RXD1	PCK1		AD8	PC21		PWM3		
PA22	TXD1	NPCS3		AD9	PC22		PWM0		
PA23	SCK1	PWM0			PC23		TIOA3		
PA24	RTS1	PWM1			PC24		TIOB3		
PA25	CTS1	PWM2			PC25		TCLK3		
PA26		TIOA2			PC26		TIOA4		
PA27		TIOB2			PC27		TIOB4		
PA28		TCLK1			PC28		TCLK4		
PA29		TCLK2			PC29		TIOA5		AD13
PA30		NPCS2		WKUP11	PC30		TIOB5		AD14
PA31	NPCS1	PCK2			PC31		TCLK5		AD15

PB0	PWM0		AD4
PB1	PWM1		AD5
PB2	URXD1	NPCS2	AD6/WKUP12
PB3	UTXD1	PCK2	AD7
PB4	TWD1	PWM2	
PB5	TWCK1		WKUP13
PB6			
PB7			
PB8			
PB9			
PB10			
PB11			
PB12			
PB13		PCK0	DAC0
PB14	NPCS1	PWM3	

8.2 melléklet, Az órajeladáshoz és megszakításokhoz kapcsolódó elnevezések

1. Interfész		órajel		megtájékoztatás	
Neve	jelölése	engedélyezés, tiltás, ID		engedélyezés, tiltás	handler
Real Time Timer	RTT	PMC_PCERO_PID 3		RTT IRQn	RTT Handler()
UART 0	UART0	PMC_PCERO_PID 8		UART0 IRQn	UART0 Handler()
UART 1	UART1	PMC_PCERO_PID 9		UART1 IRQn	UART1 Handler()
Parallel I/O Controller A	PIOA	PMC_PCERO_PID 11		PIOA IRQn	PIOA Handler()
Parallel I/O Controller B	PIOB	PMC_PCERO_PID 12		PIOB IRQn	PIOB Handler()
Parallel I/O Controller C	PIOC	PMC_PCERO_PID 13		PIOC IRQn	PIOC Handler()
Serial Peripheral Interface	SPI	PMC_PCERO_PID 21		SPI IRQn	SPI Handler()
Timer/Counter 0	TC0	PMC_PCERO_PID 23		TC0 IRQn	TC0 Handler()
Timer/Counter 1	TC1	PMC_PCERO_PID 24		TC1 IRQn	TC1 Handler()
Timer/Counter 2	TC2	PMC_PCERO_PID 25		TC2 IRQn	TC2 Handler()
Timer/Counter 3	TC3	PMC_PCERO_PID 26		TC3 IRQn	TC3 Handler()
Timer/Counter 4	TC4	PMC_PCERO_PID 27		TC4 IRQn	TC4 Handler()

Timer/Counter 5	TC5	PMC_PCERO_PID 28	TC5_IRQHandler	TC5_Handler()
Analog-to-Digital Converter	ADC	PMC_PCERO_PID 29	ADC_IRQHandler	ADC_Handler()
Digital-to-Analog Converter	DACC	PMC_PCERO_PID 30	DACC_IRQHandler	DACC_Handler()
Pulse Width Modulation	PWM	PMC_PCERO_PID 31	PWM_IRQHandler	PWM_Handler()

Példa az elnevezések használatára

```
// PIOA órajelének engedélyezése
PMC->PMC_PCER0 |= PMC_PCER0_PID11;

// PIOA megszakítások engedélyezése
NVIC_EnableIRQ(PIOA_IRQHandler);

// PIOA megszakítás handlere
void PIOA_Handler(void)
{
}
```

8.3 melléklet, A SAM3N-EK és EXPBRD-t csatlakoztató összekötőpanel port kiosztása

PORT	funkció
PA 0	
PA 1	
PA 2	7SEG data A
PA 3	7SEG data B
PA 4	7SEG data C
PA 5	7SEG data D
PA 6	gomb (alsó)
PA 7	
PA 8	
PA 9	
PA 10	
PA 11	
PA 12	
PA 13	
PA 14	
PA 15	
PA	
PA	
PA	
PA 16	
PA 17	7SEG demux A
PA 18	7SEG demux B
PA 19	
PA 20	7SEG demux C
PA 21	7SEG demux enable
PA 22	
PA 23	
PA 24	RGBLED

PORT	funkció
PB 0	LCD data 4
PB 1	LCD data 5
PB 2	LCD data 6
PB 3	LCD data 7
PB 4	
PB 5	
PB 6	
PB	
PB 7	
PB 10	LCD RS
PB 11	LCD R/W
PB 12	
PB 13	LCD EN
PB 14	
PC 6	gomb (felső)
PC 12	temp sensor LM35DZ
PC 13	
PC 14	KEYP bal
PC 15	KEYP középs
PC 16	KEYP jobb
PC 17	KEYP 1.sor
PC 18	KEYP 2.sor
PC 19	KEYP 3.sor
PC 20	KEYP 4.sor
PC 21	RGBLED
PC	
PC	
PC 22	RGBLED

PA	25	
PA	26	gomb (jobb)
PA	27	gomb (bal)
PA	28	
PA	29	
PA	30	
PA	31	

PC	23	
PC	24	
PC	25	
PC	26	
PC	27	led (jobb)
PC	28	led
PC	29	led
PC	30	led
PC	31	led (bal)

Ábrajegyzék

1.1. ábra: Programozható kombinációs hálózat	171
1.2. ábra: Programozható logikai eszközök csoportosítása	172
1.3. ábra: PLA architektúra.....	173
1.4. ábra: PAL architektúra.....	174
1.5. ábra: AMD 22V10 - a makrocella megjelenése	176
1.6. ábra: CPLD architektúra	177
1.7. ábra: Xilinx CoolRunner-II CPLD blokkvázlata.....	179
1.8. ábra: FPGA általános blokkvázlat	182
1.9. ábra: FPGA logikai cella.....	186
1.10. ábra: FPGA LUT	188
1.11. ábra: Xilinx Series-7 CLB	194
1.12. ábra: Xilinx Series-7 Slice.....	196
2.1. ábra: A fejlesztés folyamatábrája	201
2.2. ábra: Logikai tervezés sematikus ábrán	204
2.3. ábra: Logikai tervezés hardverleírással.....	205
2.4. ábra: IP centrikus rendszertervezés sematikus ábrán.....	207
2.5. ábra: Állapotgép tervezése	208
2.6. ábra: FIR szűrő IP konfigurációs felületének részlete	209

9 Irodalom jegyzék, internetes hivatkozások

Internetes hivatkozások:

1. <http://www.icinsights.com/data/articles/documents/541.pdf>
2. <http://www.icinsights.com/news/bulletins/MCU-Market-On-Migration-Path-To-32bit-And-ARMbased-Devices/>

Felhasznált szakirodalom:

3. AT91SAM ARM-based Flash MCU (ATSAM3N Series)
<http://www.atmel.com/Images/doc11011.pdf>
4. SAM3N-EK Development Board User Guide
<http://www.atmel.com/Images/doc11080.pdf>

10 Ábrajegyzék

6. 1. ábra <http://www.rlocman.ru/i/Image/2010/11/16/sam3n-ek.jpg>
7. 2-11. ábra Az Atmel Studio 6.0 fejlesztői környezetről készült képernyő mentések
8. 12. ábra AT91SAM ARM-based Flash MCU (ATSAM3N Series),
Figure 26-3. I/O Line Control Logic
9. 13. ábra <http://www.qsl.net/p/pa3ckr/bascom%20and%20avr/encoders/>
10. 14. ábra (saját munka)
11. 15, 16. ábra http://www.hestore.hu/files/tbird_exp2_sch.pdf
12. 17. ábra AT91SAM ARM-based Flash MCU (ATSAM3N Series),
Figure 31-1. Timer Counter Block Diagram
13. 18. ábra AT91SAM ARM-based Flash MCU (ATSAM3N Series),
Figure 31-3. Clock Selection
13. 19. ábra AT91SAM ARM-based Flash MCU (ATSAM3N Series),
Figure 31-4. Clock Control
15. 20. ábra http://en.wikipedia.org/wiki/File:SPI_single_slave.svg