1 Mikrokontroller perifériák programozása

1.1 Fejlesztő környezet

1.1.1 Code Composer Studio + Sysconfig

A mikrokontroller programozásához a **Code Composer Studio**-t használom, mely kifejezetten a Texas Instruments által készített mikrokontrollerekhez lett készítve, így ezeket natívan támogatja.

A Code Composer Studio ezen kívül alapértelmezetten tartalmazza a **Sysconfig** nevű konfigurációs eszközt, melynek a segítségével egy grafikus felületen tudjuk konfigurálni a perifériákat.

Help: C28x Academy

1.1.2 C2000Ware

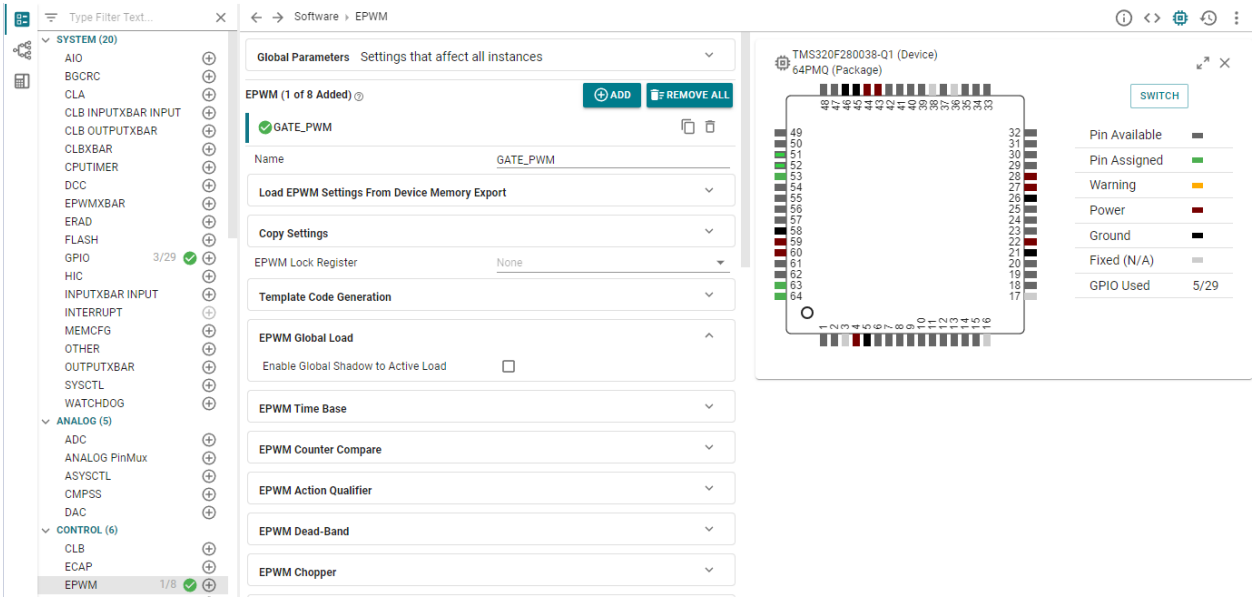
A mikrokontrollerekhez tartozó SDK[1]-t letölthetjük a C2000Ware telepítésével. Ez tartalmazza a C2000-es családba tartozó mikrokontrollerek driverjeinek könyvtárát, illetve példákat is tartalmaz a különféle mikrokontrollerekhez.

[1] SDK = Software Development Kit

1.2 Konfiguráció sysconfig segítségével

A fejlesztés során a perifériák felprogramozásához részletes leírást találunk a mikrokontroller adatlapjában, ahol láthatjuk, hogy mely regiszterek mely értékei mit jelentenek. Ezek a regiszterek a fejlesztőkörnyezetben az SDK segítségével adatstruktúrákként jelennek meg. A periféria felélesztéséhez ezeket a struktúrákat kel kitölteni a kívánt viselkedés elérésének érdekében.

Ez a folyamat azonban hosszadalmas és sok adatlapolvasást igényel. A fejlesztés felgyorsításának érdekében készült a sysconfig nevű konfigurációs eszköz, ahol egy grafikus felületen be tudjuk állítani a perifériát és ez alapján kódot generál nekünk.



1-1. Sysconfig konfigurációs interfész

Source: *C2000 SysConfig (spracx3.pdf)*

1.3 Program feltöltése a mikrokontrollerre

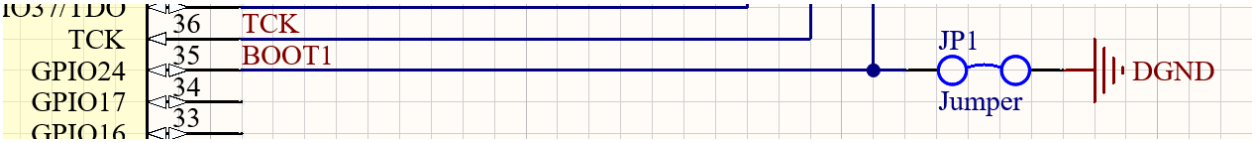
1.3.1 Boot mode kiválasztása

Én a soros kommunikáció (SCI) mellett döntöttem, mivel ehhez elég egy USB-UART átalakító, melyet olcsón és könnyedén a mikrokontroller mellé lehetett integrálni. A soros kommunikáció használatához a táblázat alapján látni lehet, hogy a **GPIO24**-et alacsonyra kell húzni, míg a **GPIO32**-őt magasra. Mivel alapértelmezetten mind a kettő GPIO értéke magas, hogy flash-ről bootoljon, így célszerűen elég csak a GPIO24-et lehúzni alacsony szintre.

Table 4-4. Device Default Boot Modes

Boot Mode	GPIO24 (Default boot mode select pin 1)	GPIO32 (Default boot mode select pin 0)
Parallel IO	0	0
SCI / Wait Boot(1)	0	1
CAN	1	0
Flash	1	1

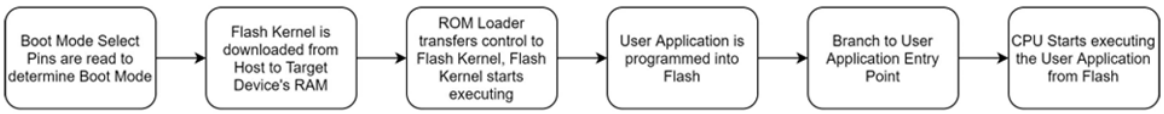
1-2. A mikrokontroller boot módja a kiválasztott pinek alapján



1-3. Boot 1 pin alacsonyra húzása jumper segítségével

1.3.2 Kód feltöltésének folyamata[1]

Amikor a mikrokontroller kommunikációs perifériáit használjuk, az nagyon egyszerűen tud a **RAM**-ba adatot, kódot tölteni. Viszont ez még nincs benne a **nem felejtő memóriában**. A két memória közötti különbséget a **flash kernel** fogja áthidalni. Ez egy olyan program részlet, melyet a kommunikáció elején a RAM-ba töltünk és a RAM-ból fogjuk futtatni. Innentől kezdve a kommunikáció többi csomagját már a Flash memóriába fogjuk tölteni. A mikrokontrollert újraindítva a „Flash” boot mode-ban már az így letöltött kódunk fog elindulni.



1-4. Program feltöltésének folyamatábrája

Az F280038-as mikrokontrollerhez a B típusú Flash Kernel fog tartozni, ami nem csak a kódot fogja betölteni a flash memóriába, de még visszajelzést is ad a felhasználó fele. Ezután vár

egy ’a’/’A’ karakterre. A kommunikáció megkezdésekor auto-baud segítségével autómatikusan kiválasztódik a legmagasabb baud rate és a letöltés megkezdődik a flash memóriába.

A flash kernelt a mikrokontrollerhez tartozó driverlib-ben fogjuk megtalálni. Itt a példák között található flash mappában megtaláljuk az SCI-hez tartozó kernel kódják, melyet lefordítva megkapjuk a <kernel>.txt fájlt.

1.3.3 Alkalmazás a program feltöltéséhez

A mikrokontrollerre egy .out típusú hex fájlt kell feltölteni. Amikor CSS-ben lefordítjuk a programunkat, a build mappában létre is fog jönni egy ilyen fájl. Ezt a fájlt a **serial_flash_programmer** nevű parancssoros alkalmazásból tudjuk feltölteni a mikrokontrollerre. Alapértelmezetten a C2000Ware SDK-ban az alábbi útvonalon lehet megtalálni:
C2000Ware_x_xx_xx_xx > utilities > flash_programmers > serial_flash_programmer

Az alkalmazás számára meg kell mondanunk minden szükséges paramétert ami szükséges a mikrokontroller programozásához:

- -d <device>: The name of the device to connect and load to: **F28003x**
- -k <file>: The file name for the CPU1 flash kernel: **flash_kernel_ex3_sci_flash_kernel.txt**
- -a <file>: The application file name to download: **UPS.out**
- -p COM<num>: Set the COM port to be used for communications: eg. **COM10**
- -b <num>: Set the baud rate for the COM port: eg. **115200**

A teljes paraméter lista így fog kinézni:
serial_flash_programmer.exe -d f28003x -k flash_kernel_ex3_sci_flash_kernel.txt -a UPS.hex -p COM10 -b 115200

1.4 PWM jel előállítása

Az ePWM, vagyis enchanced pulse width modulator (impulzus-szélesség moduláció) segítségével négyszögjeleket állíthatunk elő a buck-boost konverterünkben található MOSFET-ek vezérlésére. A mikrokontrolleren belül ezen a perifériák párban állnak rendelkezésre (A modul és B modul), melyek egymásnak az inverz jelét állítják elő. Ennek a segítségével könnyedén vezérelhetjük az átalakítónkat.

Az ePWM generátor egy számláló segítségével állítja elő a négyszögjelünket. A számláló minden órajel ciklusra megváltoztatja az értékét (ez lehet felfelé, lefelé, illetve fel-le számlálás. A számláló maximális értékét meghatározva kapjuk meg a pulzusunk periódus idejét. A kitöltési tényező meghatározásához szükségünk van egy komperálási szintre is, melynek az értéke esetén meg tudjuk változtatni a jelünket.

Az ePWM-ből képesek vagyunk különböző eseményeket is elindítani, például a periódus elérésekor, a számláló null értéke esetén, vagy valamely komperálási szint elérésekor. Ez számunkra hasznos lesz az analóg mintavételezéshez, hogy mindig a jel közepén vegyünk mintát.

Source: C2000 ePWM Developer's Guide (sprad12a.pdf)

1.4.1 ePWM modul

1.4.1.1 Frekvencia beállítása

A PWM jel előállításához szükségünk lesz a periódusidő kiszámolására is. Ennek a kiszámolásához szükségünk van az ePWM modul órajelének meghatározására. Ez jelen esetben, mivel az órajelünket nem osztjuk le, meg fog egyezni a rendszer órajelével (SYSCLK), vagyis 1**20MHz** lesz.
Ez alapján a paraméterek a következőek:

TBCLK[2] = 120MHz => $\mathbf{T_{TBCLK}}$ = **8.33ns**

Határozzuk meg a MOSFET-ek kapcsolási sebességét (100kHz):

$$\mathbf{T_{PWM}} = \frac{1}{\mathrm{F_{PWM}}} = \frac{1}{100\mathrm{kHz}} = \mathbf{10us}$$

A számláló a TBPRD[3] értékig fog számolni, így a periódusidő is ettől fog függeni. Fel vagy lefelé számolás esetén más képleteket kell figyelembe vennünk, mint fel és lefelé számolás esetén, mivel olyankor kétszer olyan hosszú a tartomány.

[2] TBCLK = time-base clock
[3] TBPRD = time-base period

Fel **vagy** lefelé számolás:

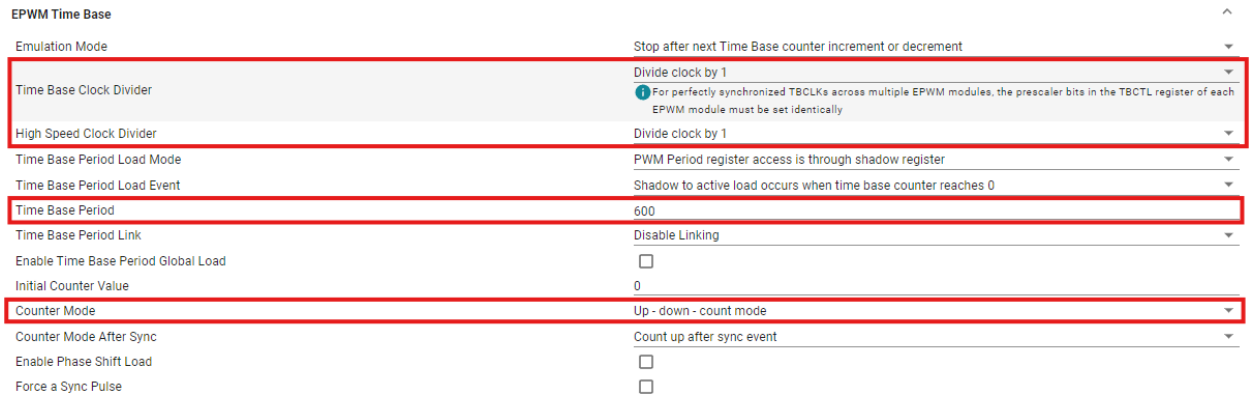
$$T_{PWM} = (TBPRD + 1)T_{TBCLK}$$

Fel **és** lefelé számolás:

$$T_{PWM} = 2 * TBPRD * T_{TBCLK}$$

A jelünket szeretnénk mintavételezni is, amit érdemes a jel közepénél megtenni. Erre érdemes háromszögjelet választani, vagyis **felfelé és lefelé** számolni, mivel ilyenkor tudunk a számláló ZRO[4] és PRD[5] értékénél mintavételezést indítani. Ez azért nagyon hasznos nekünk, mivel a CMPA érték megváltoztatása, vagyis a kitöltési tényező megváltoztatása esetén is a jel közepénél fogunk mintát venni, így azt nem kell megváltoztatni.

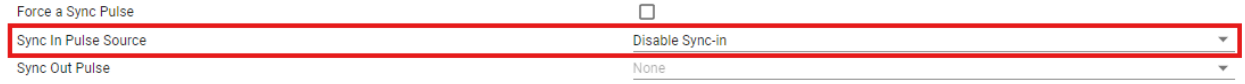
$$\mathbf{TBPRD} = \frac{T_{PWM}}{2 * T_{TBCLK}} = \frac{10us}{2 * 8.33ns} = \mathbf{600}$$



Frekvencia beállítására sysconfig segítségével.

1.4.1.2 Szinkonizáció a PWM jelek között

Mivel a mikrokontroller teljesítményelektronikák vezérlésére készült, így több beépített funkciót is tartalmaz ezek támogatására. Egyik ilyen funkció, hogy tudjuk szinkronizálni a jeleinket, ami akkor nagyon lényeges, ha a jeleink között fáziseltérés van, amit tartani is szeretnénk (például egy háromfázisú inverternél). Jelen esetben csak egy darab PWM jelünk van, így csak arra kell figyelni, hogy ne várjon szinkronizációs inputra.



[4] ZRO = Zero
[5] PRD = Period

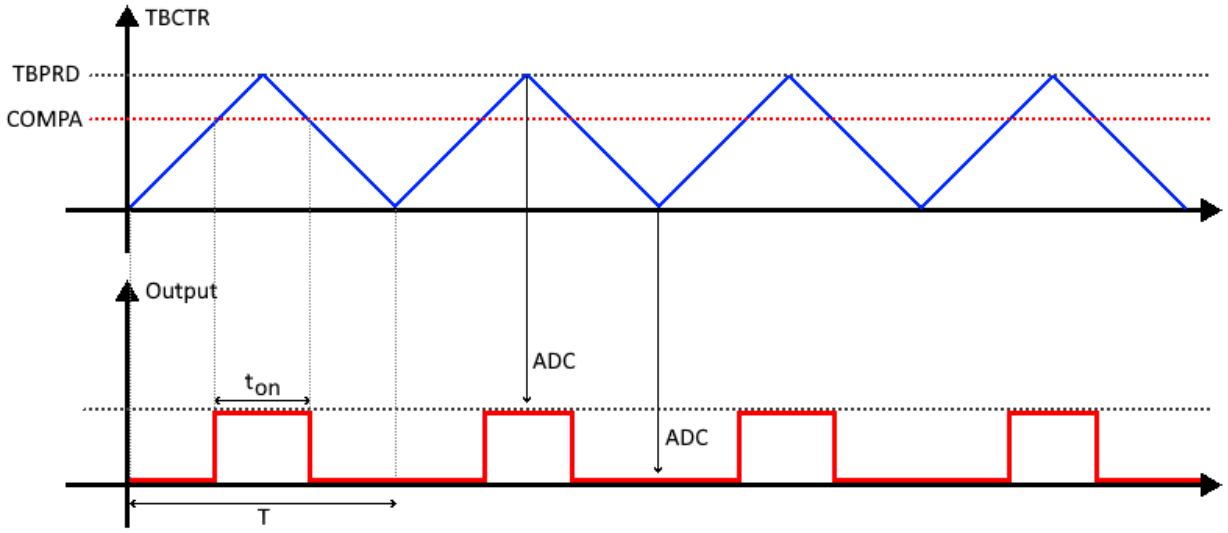
1.4.1.3 Kitöltési tényező beállítása

A kitöltési tényező (duty cycle) meghatározásához a komparátor értékét kell beállítanunk. Például ha 5V-ból 3.7V-ot szeretnénk előállítani, ahhoz 74%-os kitöltési tényezőre lesz szükségünk. Állítsuk elő a jelünket úgy, hogy alapból alacsony az értéke, de a komparátor két értéke között (vagyis amíg nagyobb a számláló értéke, mint COMPA), addig magas a PWM kimenete.

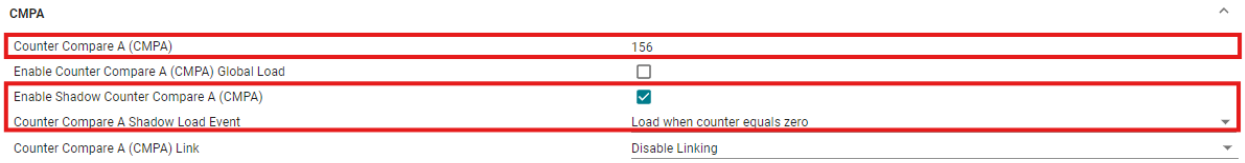
A kitöltési tényező kiszámítása:

$$D = \frac{t_{on}}{T} = 1 - \frac{COMPA}{TBPRD} \Rightarrow$$

$$\mathbf{COMPA} = (1 - D) * TBPRD = \mathbf{156}$$



PWM jel előállítása és mintavételezés

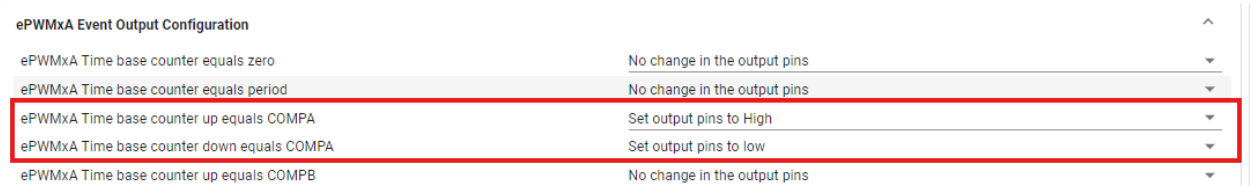


A Shadow regiszter engedélyezésére azért van szükség, hogy ha a későbbiekben majd módosítani szeretnénk CMPA értékét, hogy más legyen a kitöltési tényező, akkor azt majd itt fogjuk tudni megtenni. Azért érdemes Shadow regiszteren keresztül betölteni, mert ha menet közben állítanánk, meg van rá az esély, hogy átugorjuk a számláló értékét, ezzel a kimenetünkön kihagyva egy jelet és torzítva a PWM-et. A Shadow regisztert a számláló null értékekor fogjuk betölteni.

1.4.1.4 Action Qualifier

Ahhoz, hogy előállítsuk a PWM jelet, meg kell mondanunk a jelünknek, hogy milyen esemény esetén mit csináljon. A mi esetünkben amikor felfelé számlálunk ePWMxA-val, amikor

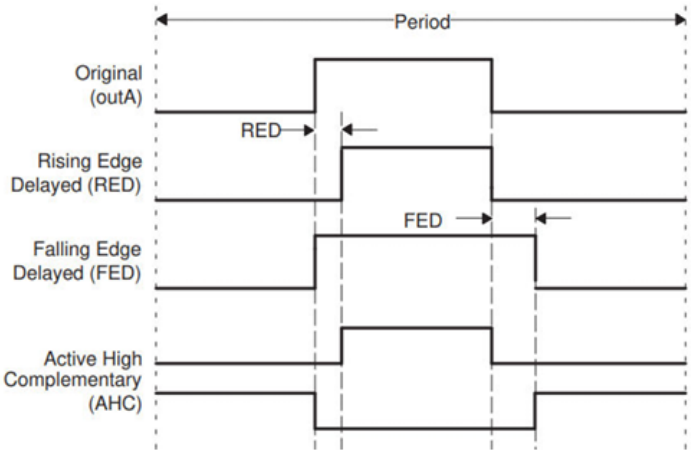
eléri a COMPA értékét, akkor legyen magas a kimeneti jel. Amikor lefelé számolva éri el COMPA értékét, akkor pedig legyen alacsony a kimeneti jel.



Ezt a beállítást csak ePWMxA esetén kell megtennünk, mivel ePWMxB-t a Deadband Submodule segítségével fogjuk előállítani, olyan módon, hogy **Active High Complementary** jel legyen, aminek a forrása ePWMxA lesz.

1.4.2 Deadband (DB) Submodule

A deadband[6] segítségével egy holtsávot határozhatunk meg az A és B ePWM modulok között. Ez azért nagyon fontos, mivel a MOSFET-nek váltás esetén kell még egy kis idő, hogy lezárjanak. Ha ezt az időt nem adjuk meg nekik, a VDD és GND vonalak között rövidzár alakul ki, mivel mindkét MOSFET nyitva van. Ezt a jelenséget **keresztezési áramnak** hívjuk.



1.4.2.1 Deadband méretezése

A MOSFET teljes lezárásához meg kell várnunk a t_f „fall time”-ot (és ennél érdemes egy kicsit több idővel számolni. Az áramkörhöz választott MOSFET esetén: t_f = 20.2ns. Ez alapján a „**dead time**”-ot ennél jóval nagyobbra választom, **100ns**-ra, hogy biztosan zárva legyen már a MOSFET.

A jeleket „**Active High Complementary**” (AHC) módban kell használni, vagyis az aktív jel előtt és után is lesz egy olyan állapot, amikor egyik jel sem aktív. Ezt úgy tudjuk elérni, hogy az aktív jel várakozik amikor felfutna az éle (RED = Rising Edge Delay), míg az invertált jel pedig az aktív jel lefutó éle esetén várakozik (FED = Falling Edge Delay).

[6] Deadband = holtidő

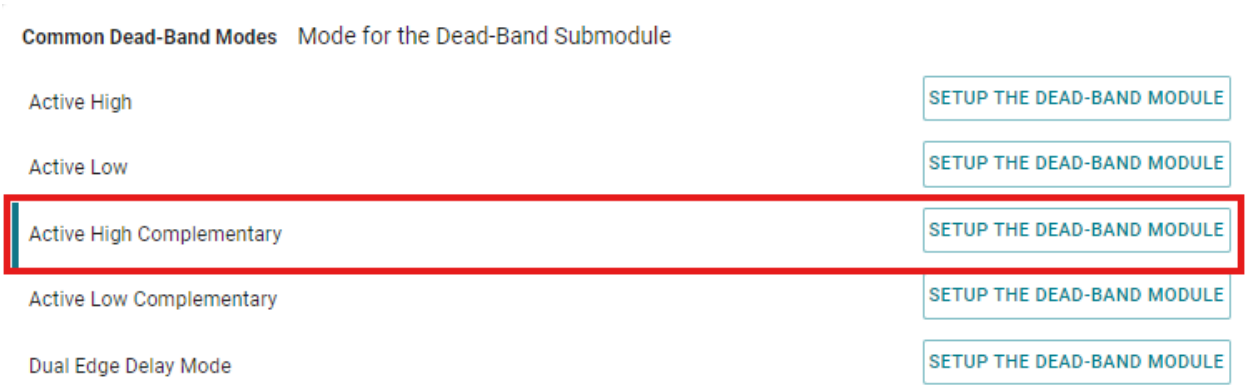
A RED, illetve FED értékeket az alábbi képletek alapján lehet kiszámolni:

$$\mathrm{RED} = \mathrm{DBRED} * \mathrm{T_{TBCLK}} \Rightarrow$$

$$\mathbf{DBRED} = \frac{\mathrm{RED}}{\mathrm{T_{TBCLK}}} = \frac{100ns}{8.33ns} \rightarrow \mathrm{DBRED} = \mathbf{12}$$

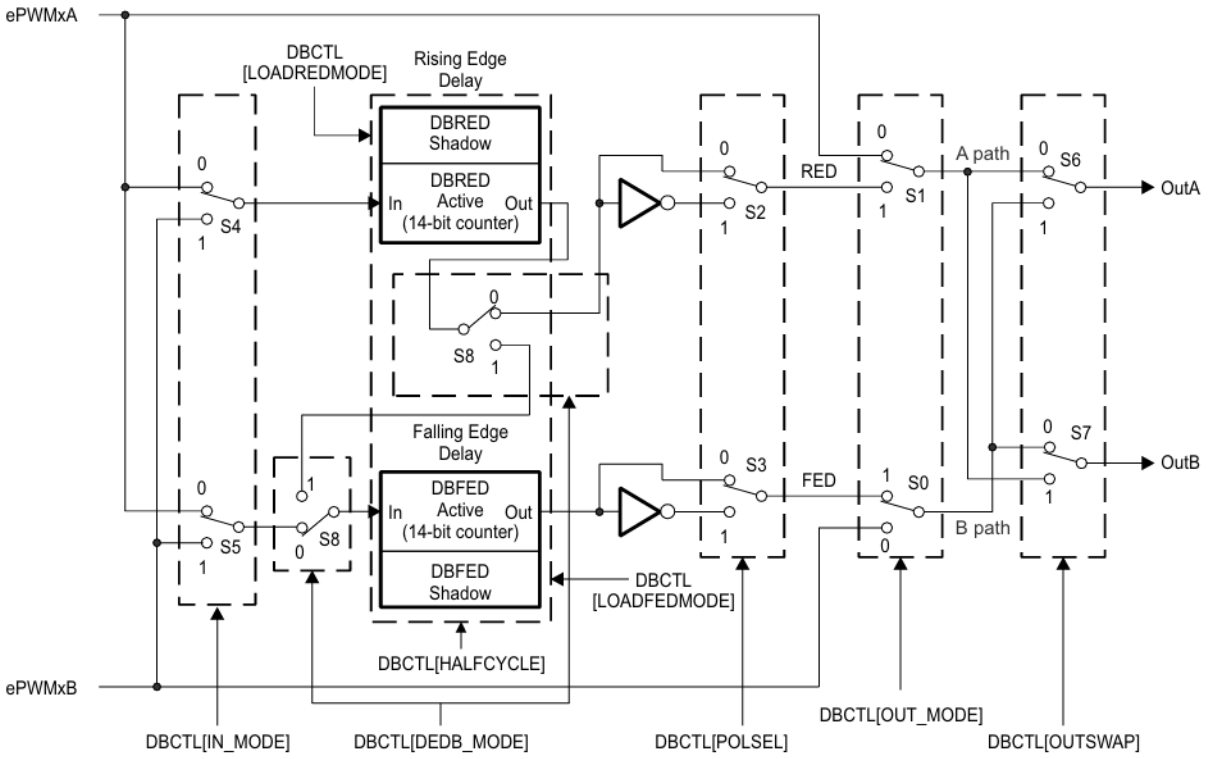
A „**falling edge delay value**” értéke meg fog egyezni a DBRED értékével.

A Dead-Band modul felkonfigurálásának megkönnyítésére be tudjuk tölteni a gyakori konfigurációkat. Itt én az Active High Complementary opciót választottam, ami kiválasztotta megfelelő opciókat, így nekem már csak a kívánt késleltetési időket kellett beírnom.



Gyakori Dead-Band konfigurációk betöltése.

Az alábbi ábrán látható, hogy a periféria konfigurációja során, a „gyorsgomb” milyen opciókat állított be (többet között, hogy melyik jel legyen a késleltetés alapja, legyen-e invertálás, stb.).



Dead-Band modul konfigurációs opciói.

EPWM Dead-Band

Common Dead-Band Modes Mode for the Dead-Band Submodule

Rising Edge Delay Input	Input signal is ePWMA
Falling Edge Delay Input	Input signal is ePWMA
Rising Edge Delay Polarity	DB polarity is not inverted
Falling Edge Delay Polarity	DB polarity is inverted
Enable Rising Edge Delay	☑
RED Shadow Load Event	Load when counter equals zero
Enable RED Shadow Mode	☐
Rising Edge Delay Value	12
Enable Falling Edge Delay	☑
FED Shadow Load Event	Load when counter equals zero
Enable FED Shadow Mode	☐
Falling Edge Delay Value	12
Swap Output for EPWMxA	☐
Swap Output for EPWMxB	☐

A beállított opciók, ahol már csak a késleltetési értékeket kell beírni.

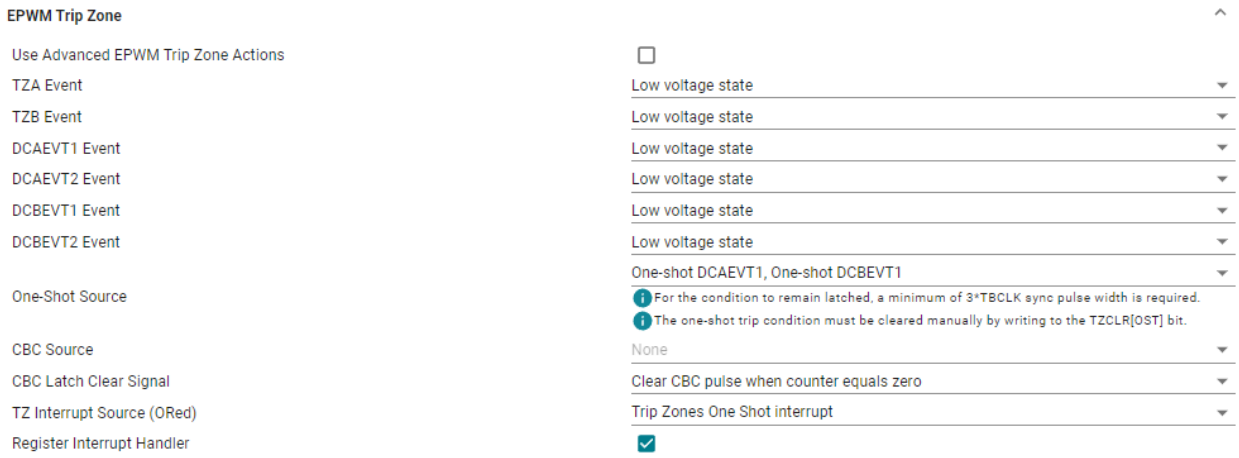
1.5 PWM hibavédelem

1.5.1 Trip-Zone (TZ) Submodule

Az ePWM esetén lehetőség van Trip-Zone-ok („leoldási zónák”) meghatározására. Ez egy olyan védelem, ahol egy hiba vagy esemény hatására a PWM modul egy meghatározott állapotba lép. A Trip-Zone aktíválására lehetőségünk van digitális komperátor használatára.

A Trip-Zone működése lehet **Cycle-By-Cycle**, ilyenkor aktiválódáskor a kimenetek a kívánt állapotban vannak, viszont ZRO vagy PRD esetén újból engedélyezve vannak. A másik fajta működés a **One-Shot**, mely csak egyszer aktiválódik, ahhoz, hogy a PWM folytassa működését szoftveresen engedélyeznünk kell.

A panelhez a **One-Shot** működést szeretnénk, hibás működés esetén nem szeretnénk, hogy az áramkör újból működésbe lépjen.

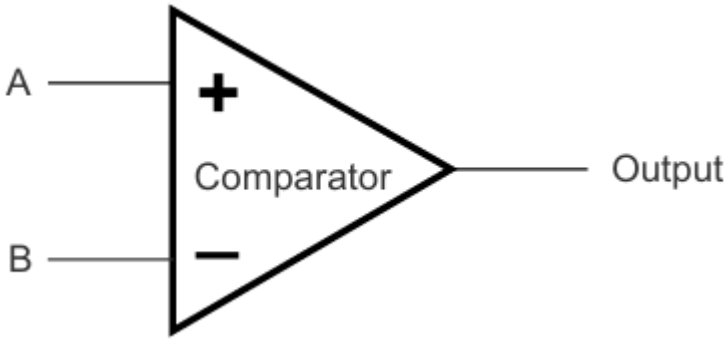


Trip-Zone aktiválódásakor alacsony értéket vesznek fel a kimenetek.

1.5.2 Digital Compare (DC) Submodule

1.5.2.1 Digitális komparátor

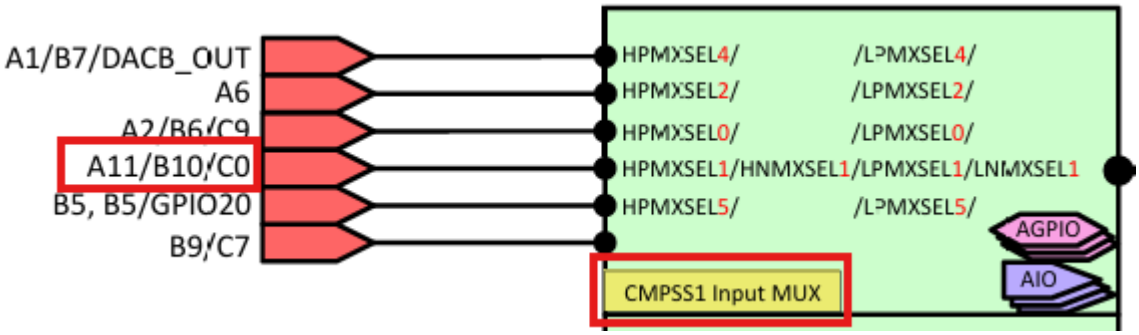
Egy digitális komperátor belső 12 bites analóg referenciáját hasonlítjuk össze egy analóg bemenettel. Az összehasonlítás eredményeképpen az EPWMxA és EPWMxB modulokat az általunk meghatározott állapotba vihetjük. Ez egy fontos védelem, melynek a segítségével hardveres alapon a hiba bekövetkeztével rögtön reagálhatunk rá, a szoftver aktuális állapotától függetlenül.



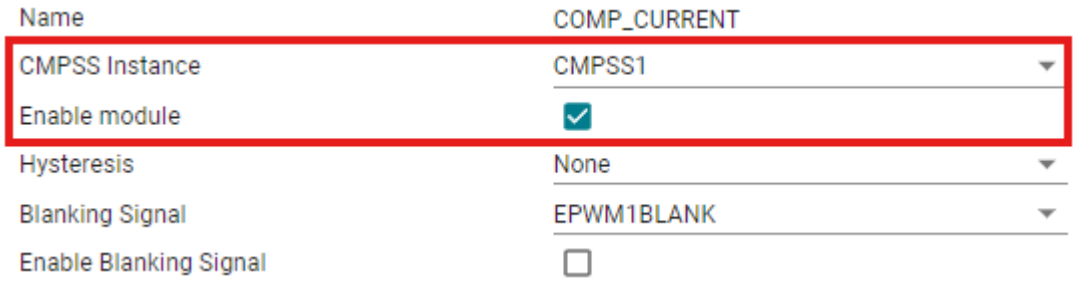
A digitális komperátor kimenete magas, ha a komperátor pozitív bemenetén nagyobb a feszültség, mint a negatív bemenetén. A védelmet úgy tudjuk megvalósítani, hogy kiszámoljuk, mely feszültség esetén van **overvoltage** vagy **undervoltage.** Ennek az értéknek megfelelő referencia feszültséget kötünk a komparátor negatív bemenetére és ha a pozitív bemenetén magasabb a feszültség, akkor a komparátor kimenete magas értéket vesz fel. Undervoltage esetén a komparátor kimenetét invertálnunk kell. A komparátor magas értéke esetén aktiválódik a **Trip-Zone**.

Egy Digital Compare submodule-ban két komparátor található, egy **High** és egy **Low**. Ezeket egy submodule használatával tudjuk ellenőrizni az over- és az undervoltage-ot.

A komparátor használatához meg kell határoznunk, hogy melyik komparátor tartozik az áram mérésére szolgáló analóg pinhez. Ezt az „Analog Subsystem Block Diagram”-ja alapján könnyedén meg tudjuk tenni:



Analog Subsystem Block Diagram



Megfelelő komparátor kiválasztása

1.5.2.2 Komparátor méretezése

Az áramkör alap állapotban normális működés esetén **6A** áramnál nem fogyaszt többet. Ennél csak akkor lehet magasabb az áramfelvétel, ha az **akkumulátor feszültsége nagyon alacsony** (ami már a kapacitásának a legvége és ilyenkor az akkumulátor használata már káros) vagy pedig ha **hiba lépett fel** az áramkörben.

Az áramkör maximálisan 8A áram felvételére van méretezve, mikor már az akkumulátor feszültsége nagyon alacsony (2.5V), melyet nem szeretnénk elérni. Így a korlátot **7A**-re méretezzük.

Az árammérő referencia feszültsége 1.65V, és a teljes tartományban (-8A … 8A) -1.6V és 1.6V között változik az árammérő feszültsége.

Linearizált képlet a feszültség kiszámolására az áram függvényében:

$$\mathbf{V_{cur+}} = \mathrm{V_{ref}} + 0.2 * \mathrm{I} = 1.65\mathrm{V} + 0.2 * 7\mathrm{A} = \mathbf{3.05V} = \mathbf{DACOUT_{+}}$$

$$\mathbf{V_{cur-}} = \mathrm{V_{ref}} + 0.2 * \mathrm{I} = 1.65\mathrm{V} - 0.2 * 7\mathrm{A} = \mathbf{0.25V} = \mathbf{DACOUT_{-}}$$

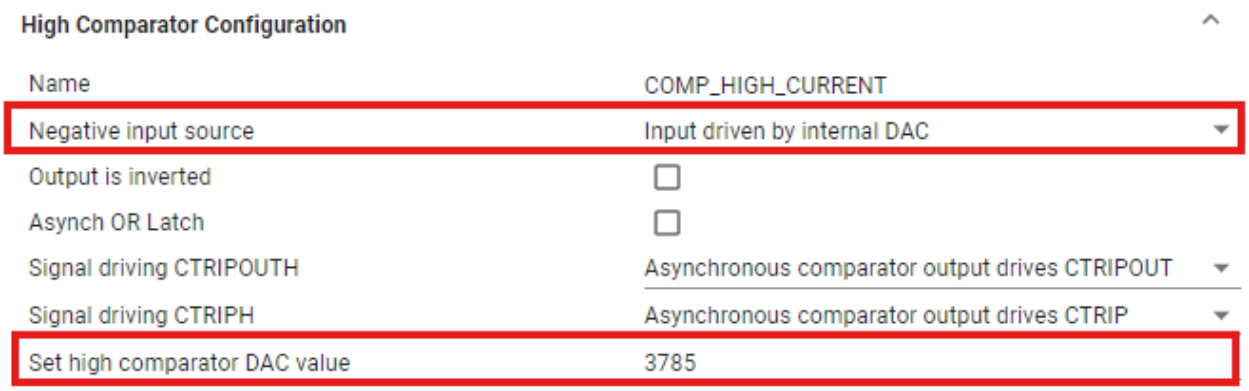
Referencia feszültség DAC[7] értéke:

$$\mathrm{DACOUT} = \frac{\mathrm{DACVALA} * \mathrm{DACREF}}{4096} \Rightarrow$$

$$\mathbf{DACVALA_{+}} = \frac{4096 * \mathrm{DACOUT}}{\mathrm{DACREF}} = \frac{4096 * 3.05\mathrm{V}}{3.3\mathrm{V}} = \mathbf{3785}$$

$$\mathbf{DACVALA_{-}} = \frac{4096 * \mathrm{DACOUT}}{\mathrm{DACREF}} = \frac{4096 * 0.25\mathrm{V}}{3.3\mathrm{V}} = \mathbf{310}$$

[7] DAC = Digital-Analog Converter



High comparator beállítása

A low komparátor esetén ügyeljünk arra, hogy a kimenet **invertálva** legyen.

Ezen kívül figyeljünk, hogy a DAC referencia feszültsége VDDA (3.3V) legyen.

VDDA referencia feszültség

1.5.2.3 Digital Filter

A digitális komparátorunknak lehetőségünk van a kimenetének a megszűrésére. Ezzel elkerülhetjük, hogy egy zaj okozta hibás mérés miatt leálljon az egész áramkör.

Egyrészt szűrve van a magas frekvenciás rajoztól, mivel csak minden *n-edik* órajelre veszünk intát. Másrészt a FIFO-ban található magas értékeknek többségben kell lenniük.

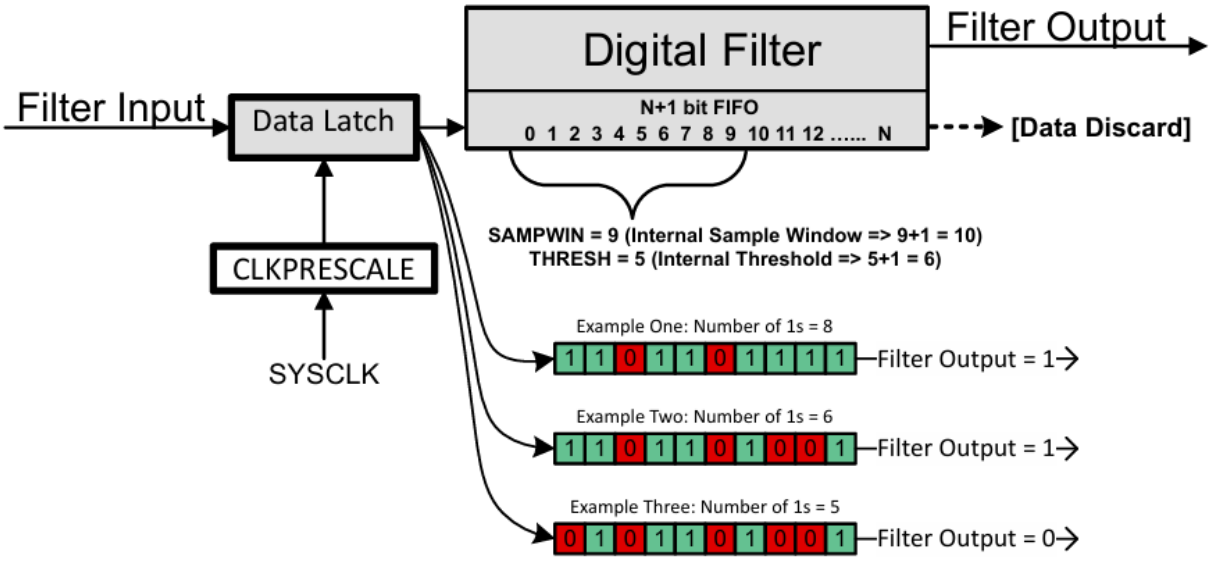


Figure 18-6. Digital Filter Behavior

A filtert kezdetben töröljük, hogy inicializálás esetén ne menjünk rögtön hibás állapotba. A prescale 0, ami azt jelenti, hogy minden órajelre mintát veszünk. Illetve a FIFO mérete 9 lesz, amiből legalább 5 darab 1-esnek kell lennie, ahhoz hogy aktiválódjon a komparátor kimenete.

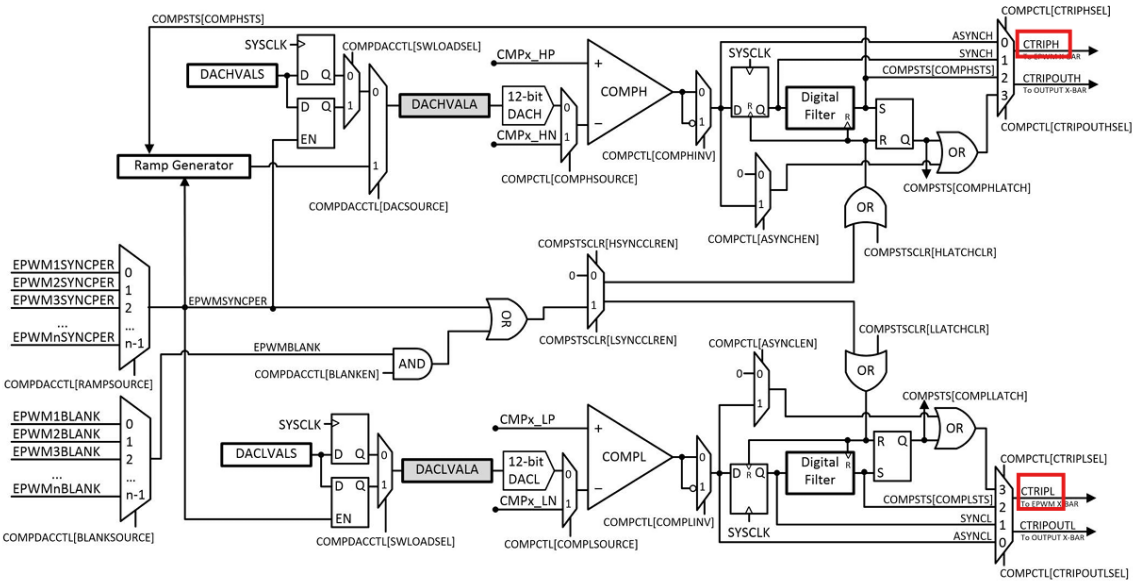
Ez azt jelenti, hogy 9/120MHz = **75ns** késleltetést visz be a rendszerbe, viszont ellenállóbb lesz a zajokkal szemben.

Digitális filter beállítása.

1.5.3 ePWM X-BAR

A digitális komparátor beállítása után szükségünk van, hogy a jelét a Trip-Zone-hoz vezessük. Mivel alapértelmezetten nincsen közvetlen összeköttetés közöttük, ezért saját magunknak kell meghatároznunk azt. Ezt az ePWM X-BAR segítségével tudjuk megtenni. Ahhoz, hogy ezt konfigurálni tudjuk meg kell határoznunk a jelünket.

A komparátor oldaláról meg kell keresnünk, annak a kimenetét.



High and Low comparator output.

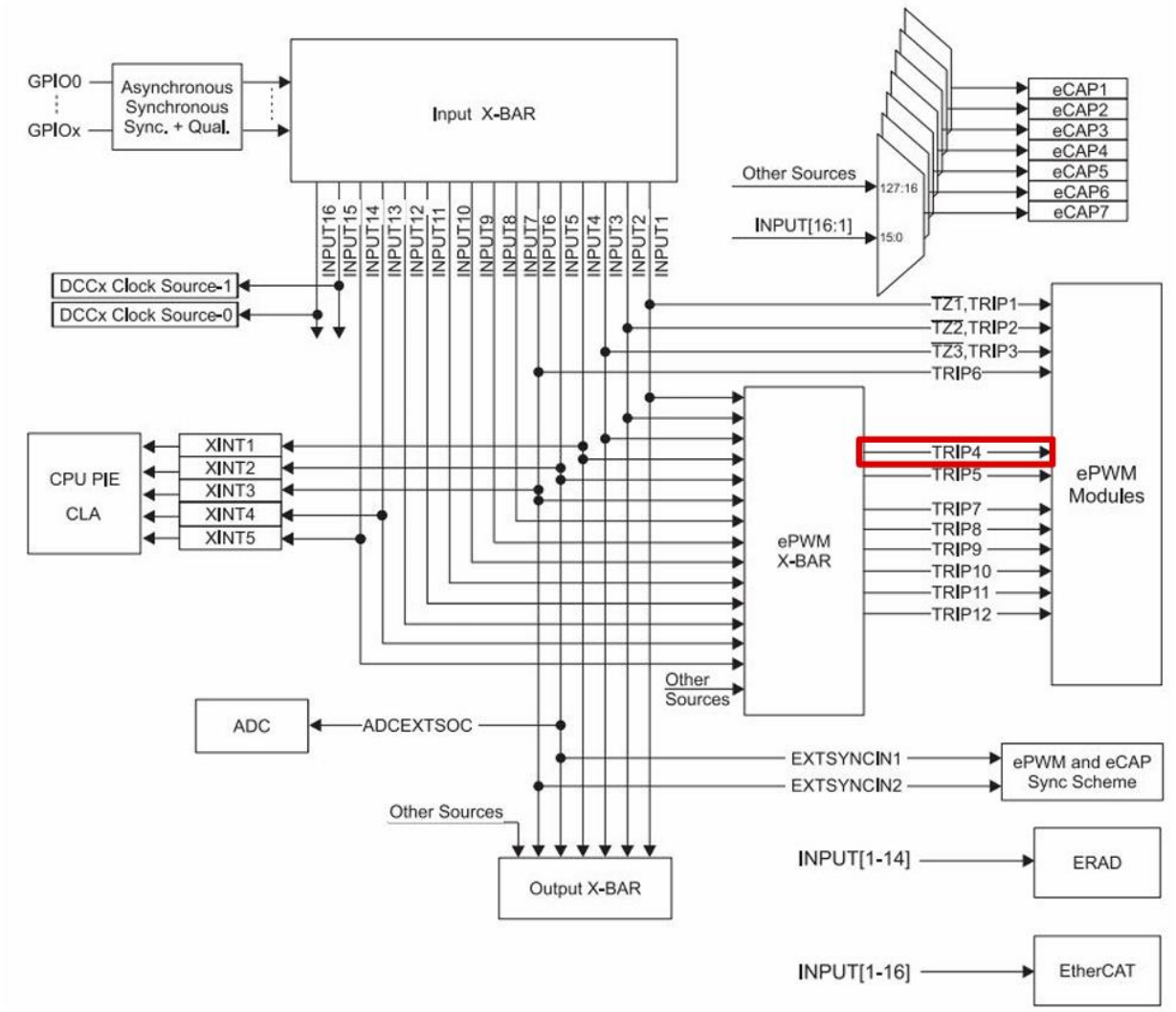
Ezt a jelet az ePWM X-BAR Mux konfigurációs táblázatban megkeresve láthatjuk, hogy mi lesz a Mux értéke. A táblázat alapján látható, hogy a high és a low komparátor kimenetét be lehet egyszerre kötni.

Table 11-3. EPWM X-BAR Mux Configuration Table

Mux	0	1	2	3
G0	CMPSS1_CTRIPH	CMPSS1_CTRIPH_OR_CTRIPL	ADCAEVT1	ECAP1_OUT
G1	CMPSS1_CTRIPL	INPUTXBAR1	CLB1_OUT12	ADCCEVT1
G2	CMPSS2_CTRIPH	CMPSS2_CTRIPH_OR_CTRIPL	ADCAEVT2	ECAP2_OUT

ePWM X-BAR Mux Configuration Table

A Trip-Zone oldaláról meg kell keresnünk a bemeneti jelet. Látható, hogy az első három Trip jel a GPIO-khoz tartozik. Így az analóg Trip jelét az ePWM X-BAR jelei közül kell választani.



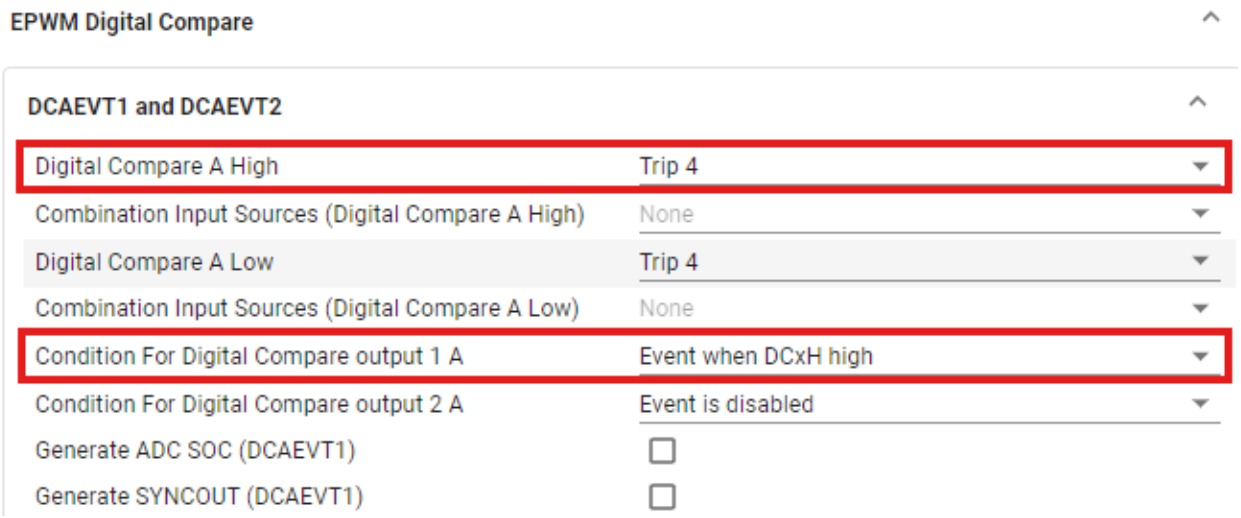
Input X-BAR Block Diagram

A diagrammok alapján konfigurált EPWMXBAR

Miután már összekötöttük a DC jelét a Trip-Zone-al, az ePWM digital compare viselkedését be tudjuk állítani. A mikrokontroller dokumentációjában olvashatjuk, hogy One-Shot eseményt csak az DCxEVT1-es event válthat ki. Mivel egy DCxEVT1 event csak egy komparátor kimenetét tudja fogadni, így a DCAEVT1 és a DCBEVT1 eventeket is használjuk.

Each $\overline{TZn}$ input can be individually configured to provide either a cycle-by-cycle or one-shot trip event for an ePWM module. DCAEVT1 and DCBEVT1 events can be configured to directly trip an ePWM module or provide a one-shot trip event to the module. Likewise, DCAEVT2 and DCBEVT2 events can also be configured to directly trip an ePWM module or provide a cycle-by-cycle trip event to the module. This configuration is determined by the TZSEL[DCAEVT1/2], TZSEL[DCBEVT1/2], TZSEL[CBCn], and TZSEL[OSHTn] control bits (where n corresponds to the trip input), respectively.

20.9.2 Operational Highlights for the Trip-Zone Submodule



Trip-Zone bemenet és kiváltó feltétel beállítása.
B event esetén a kiváltó feltétel a low komparátor.

1.6 Analóg mintavételezés

1.6.1 Analóg referencia feszültség

Az analóg mintavételezéshez szükséges, hogy megadjuk az átalakítók referencia feszültségét. A mért feszültségek a 3.3V-os tartományra vannak méretezve, így a referencia feszültséget is ennek megfelelően kell választani.

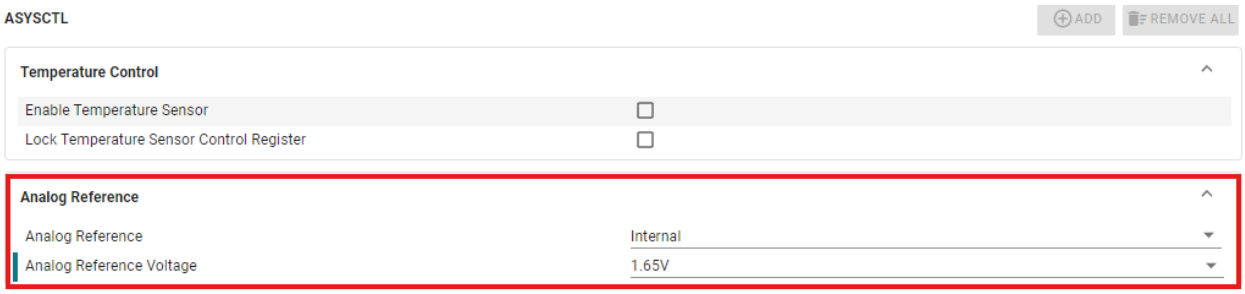
16.2.3.2 Internal Reference Mode

In internal reference mode, the device drives a voltage out onto the VREFHI pin. The VREFHI and VREFLO pins then set the ADC conversion range.

The internal reference voltage can be configured to be either 2.5V or 1.65V. When the 1.65V internal reference voltage is selected, the ADC input signal is internally divided by 2 before conversion, which effectively makes the ADC conversion range from VREFLO to 3.3V.

Internal Reference Mode

A mikrokontroller adatlapja alapján, ha belső 3.3V-os referencia feszültséget szeretnénk, akkor az 1.65V-os referenciát kell választani, mely esetén a mért jel is le van osztva 2-vel.



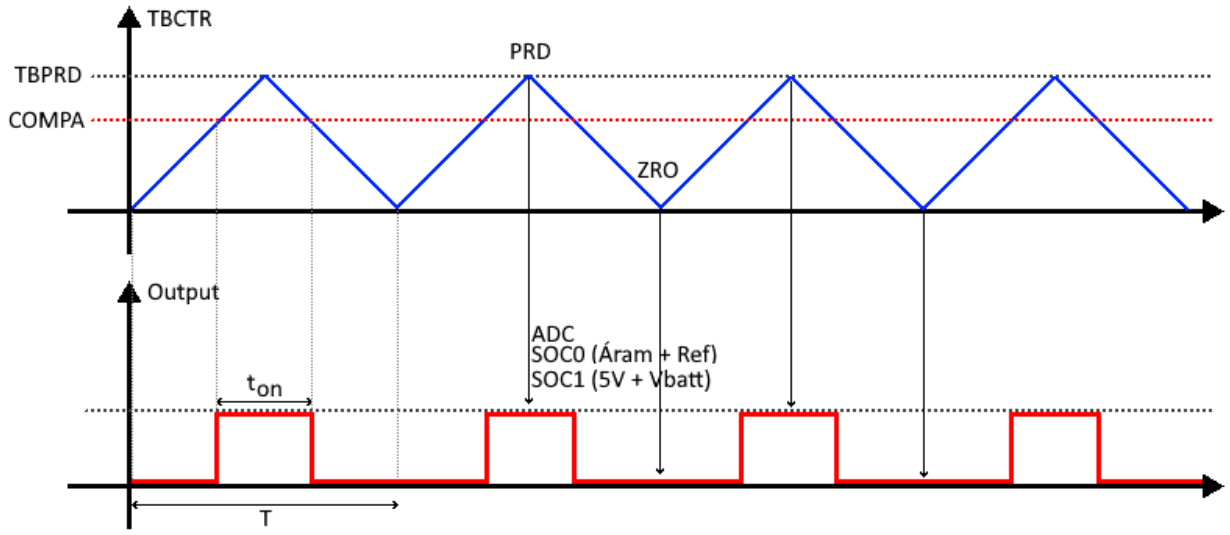
1.6.2 Event-Trigger (ET) Submodule

Az Event-Trigger modul segítségével a TB[8], CC[9] és DC[10] eseményeinek hatására megszakítást (IT) vagy analóg-digitál konverziót (ADC) tudunk indítani.

A Buck-Boost konverter megvalósításához érdemes lesz a számlálót ZRO, illetve PRD érték esetén indítani analóg-digitál konverziót, ekkor a PWM jelek közepén fogunk mintavételezni, amikor már (feltehetőleg) lefutottak a tranziensek.

A rendszeres mintavétel kulcsfontosságú az átalakító megvalósításához, hiszen ez lesz a visszacsatolásunk a szabályzást illetően.

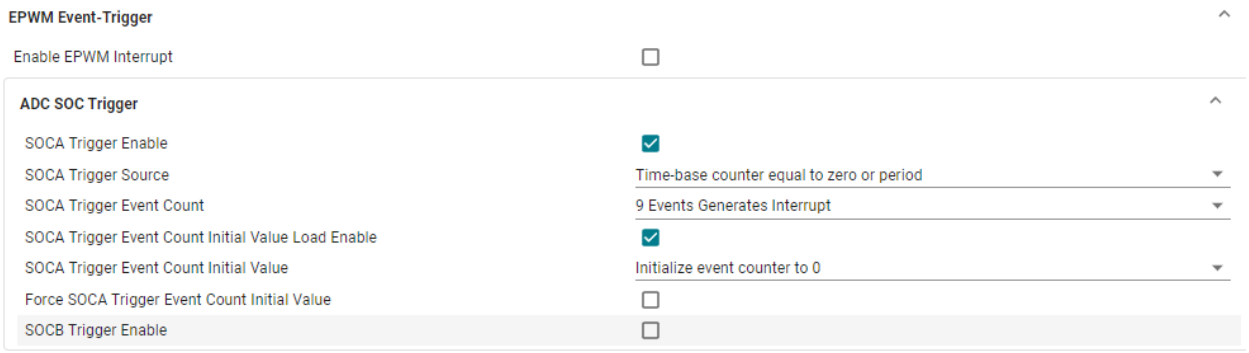
[8] TB = Time-Base submodule
[9] CC = Counter-Compare submodule
[10] DC = Digital Compare submodule



Analóg mintevételezés PRD és ZRO értékek esetén.

A SOC-okat az ePWM modul Event-Trigger submodule-jával tudjuk meghívni. Egy két féle jellel is jelezhetünk egy SOC-nak, SOCA-val és SOCB-vel. Mivel van lehetőség, hogy ZRO és PRD értékekre is jelezzen SOCA, így jelenleg nincs szükség SOCB-re.

Azért, hogy ne legyen túl sok megszakítás generálva, lehetőség van az „Event Count” megadására, amelynek segítségével csak minden n-edik trigerre fog megszakítás érkezni.



Event trigger beállítása

1.6.3 Start-Of-Operation (SOC)

1.6.3.1 SOC konfiguráció

Az Analóg-Digitál Konverterek működésének egyik alapköve a SOC rendszer. Mindegyik ADC külön SOC-okkal rendelkezik, és egy SOC-ban a következő mezők találhatóak: analóg pin, mintavételi idő, trigger feltétel, eredmény.

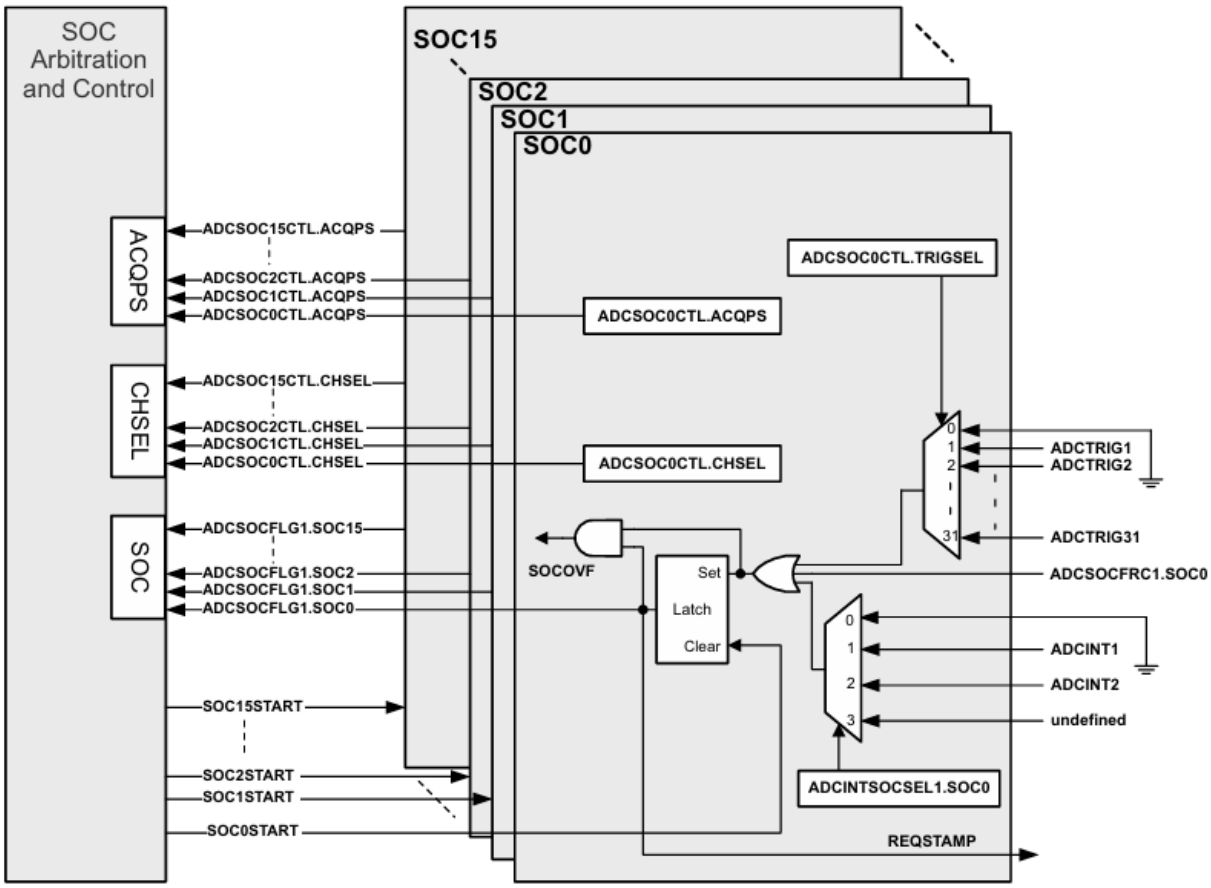
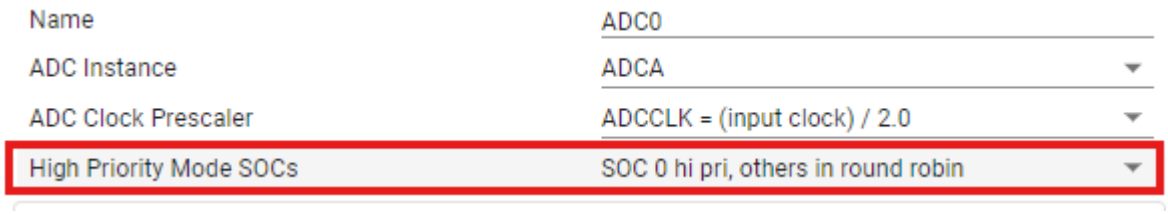


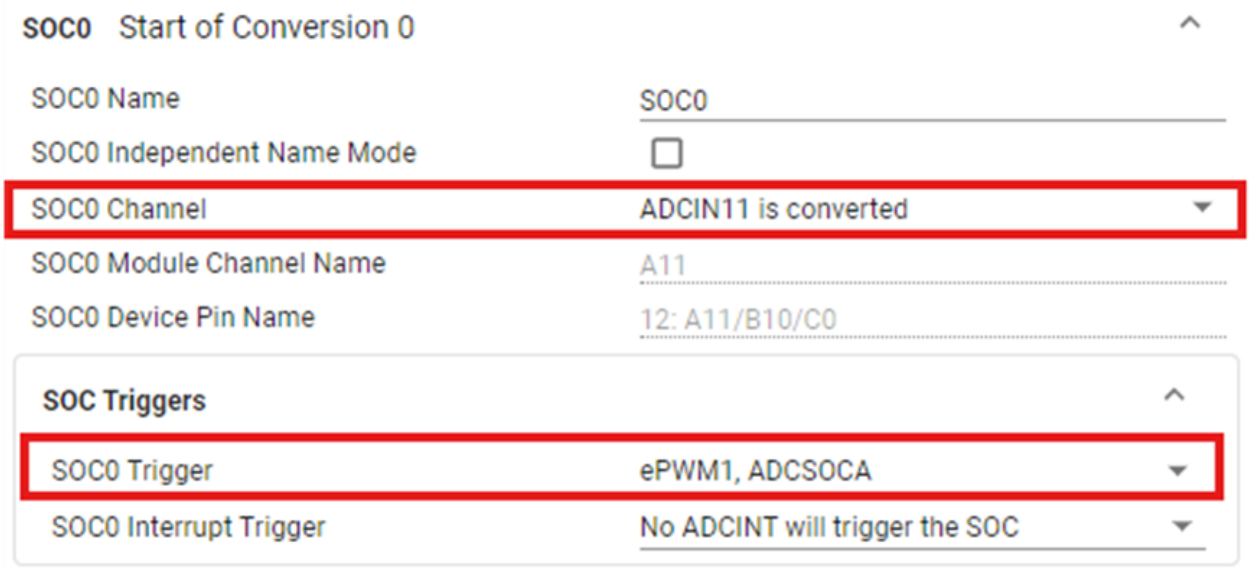
Figure 16-2. SOC Block Diagram

Mivel a mikrokontrollerben 3 darab ADC van és mi 4 különböző analóg jelet akarunk mintavételezni, így nem tudunk minden jelnek külön ADC-t adni, hanem egymás között kell felosztani őket. Ha egy ADC-nek több SOC-ot is megadunk, akkor Round-Robin alapon kerül felosztásra (de akár prioritással is lehet őket). A konverzió ideje sokkal gyorsabb, mint a PWM jel periódusideje, így ez számunkra nem lesz meghatározó, viszont azért, hogy a megszakítások hívásakor mindig készen álljanak az eredmények, így SOC0-nak adom a prioritást, míg SOC1 fogja hívni az IT-t.



SOC prioritás beállítása.

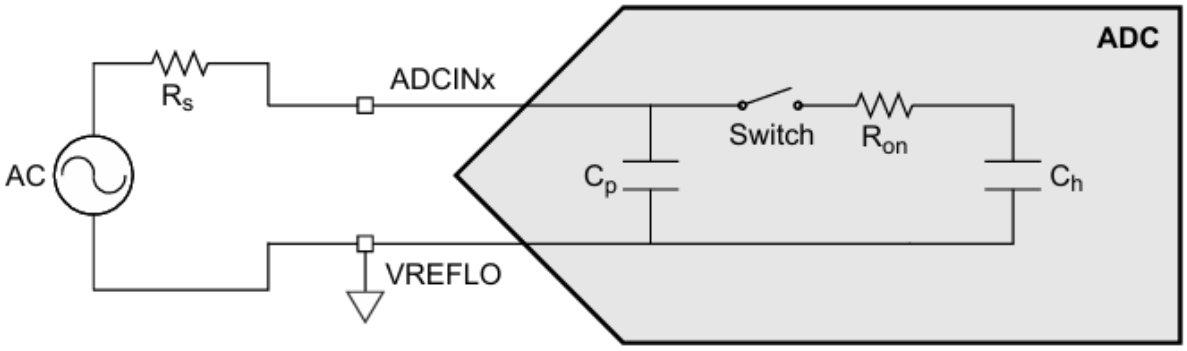
A SOC-nak megadjuk, hogy melyik analóg bemenetet mintavételezze, kiolvasáskor a SOC-ban benne lesz az eredmény. Illetve a SOC-nak meg kell adni a trigger forrását, ami az előbbiekben beállított EPWM1, SOCA forrása lesz.



Pin és trigger feltétel beállítása

1.6.3.2 Mintavételi ablak meghatározása

Az ADC működéséhez szükség van egy belső kondenzátorra, amit fel kell töltenünk. Ha túl gyorsan veszünk mintát, nem tud eléggé feltöltődni és hamis értéket fogunk kiolvasni.



ADC modellje

A mintavételhez szükség van a Ch kapacitás feltöltésére, ami a vele sorosan kapcsolt ellenállás lassít. Ugyanakkor a bemenetre kapcsolt kondenzátor segít, mivel tud töltéseket biztosítani.

An approximation of the required settling time can also be determined using an RC settling model. The time constant for the model is given by the equation:

$$\tau = (R_S + R_{on}) \times C_h + R_S \times (C_S + C_p) \quad (5)$$

And the number of time constants needed is given by the equation:

$$k = ln\left(\frac{2^n}{settling\ error}\right) - ln\left(\frac{C_S + C_P}{CH}\right) \quad (6)$$

So the total S+H time must be set to at least:

$$t = k \cdot \tau \quad (7)$$

Rs-t a bemeneten lévő ellenállás osztóból meg tudjuk határozni: Rs = 3.3k x 4.7k = 1940 Ohm

A bemeneten egy Cs = 47nF értékű kondenzátor van.

Az adatlap alapján ismerjük a többi értéket:

Ron = 860 Ohm

Ch = 7.5pF

Cp = 13pF

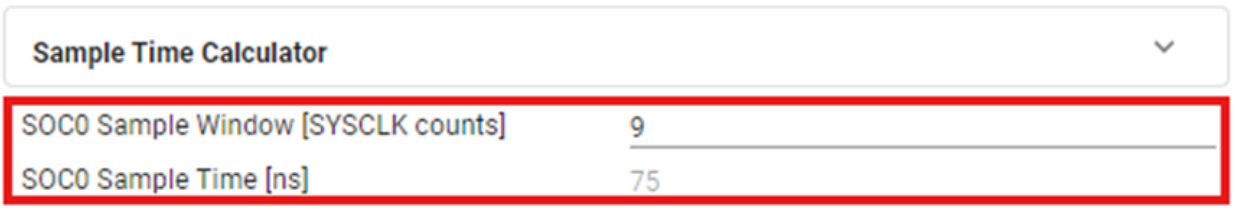
A fenti egyenletbe behelyettesítve **Tau = 60.77ns**.

A végértéket ¼ LSB-re közelítsük meg, tehát a settling error legyen 1/4.

k = 9.7 – 8.7 = 1

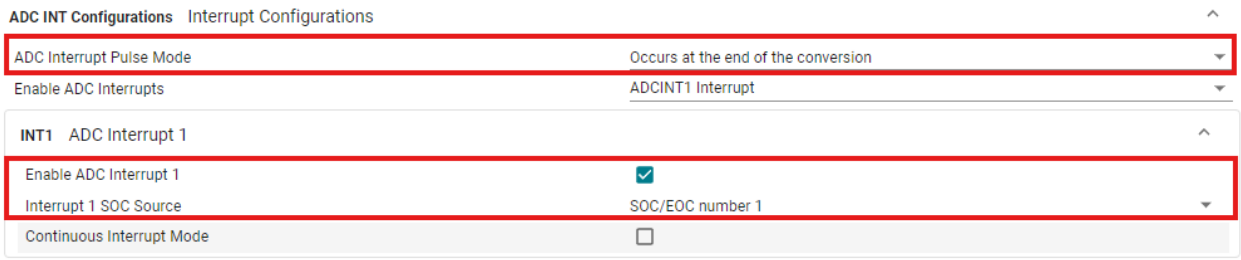
Tehát legalább 60.77ns-ot kell várni.

A sample window legkisebb értéke 9, ami 75ns-nek felel meg, így ez nekünk jó is lesz.

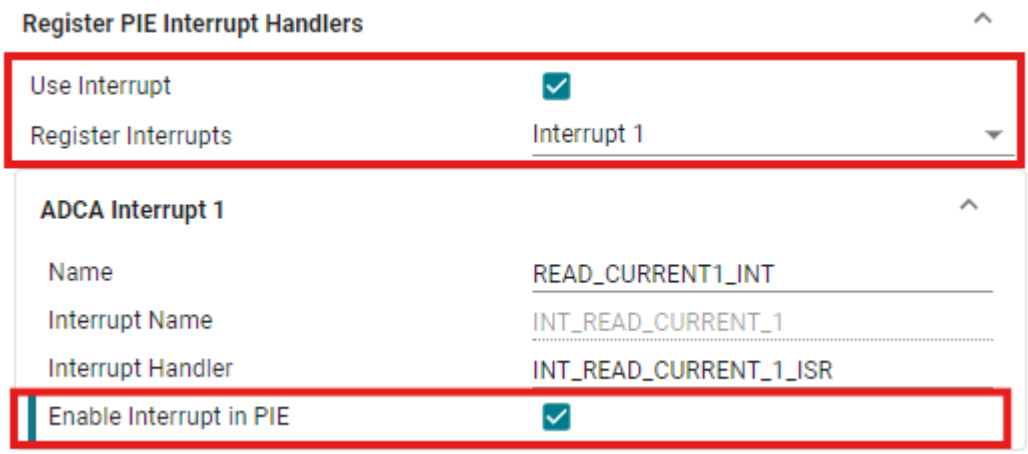


1.6.3.3 ADC megszakítás

Engedélyezzük a megszakításokat, hogy szoftverből is elérhessük. A megszakítást SOC1 indítsa, mert akkor már lefutott mind a kettő konverzió.



Beállítjuk a megszakítást.



A megszakítást engedélyezzük a megszakítás kezelőben is.

1.7 Timer

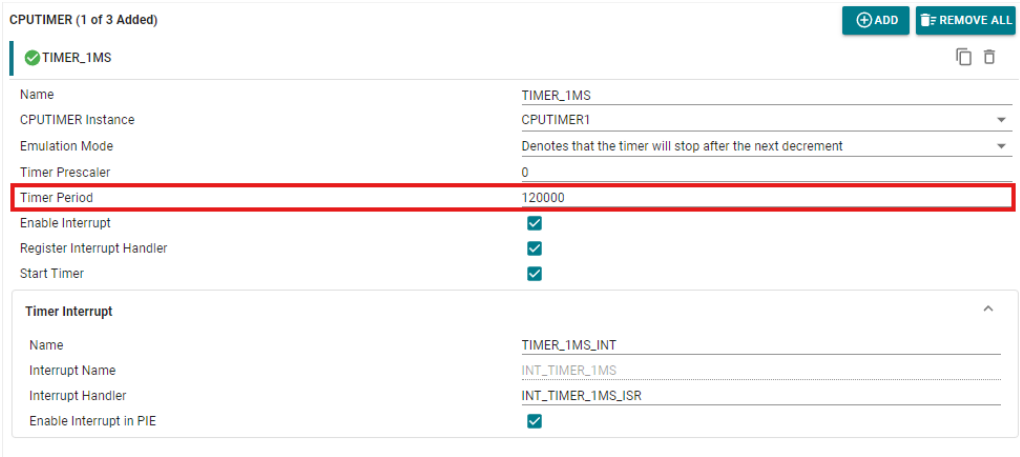
A fő ciklusban lassú függvények is lefutnak. Viszont, ha nem lenne semmi korlát rajtuk, akkor az idő nagy részében a CPU velük foglalkozna, miközben lehetséges, hogy van másik fontos feladat is, amit végre kellene hajtani.

Erre a feladatra jelent megoldást egy időzítő használata. Az időzítő segítségével meg tudjuk oldani, hogy a lassú program részletünk csak adott időközönként fusson le.

A timer modult ezen kívül, mint ahogy a nevéből is adódik, időzítésre is tudjuk használni. Az időzítő által generált megszakításokat számolva nem blokkoló módon várakozhatunk. Az időzítő beállításához meg kell adnunk, hogy hány órajelig számoljon. Jelen esetben nem lesz leosztva a CPU óra jele, így a képlet a következőképpen alakul:

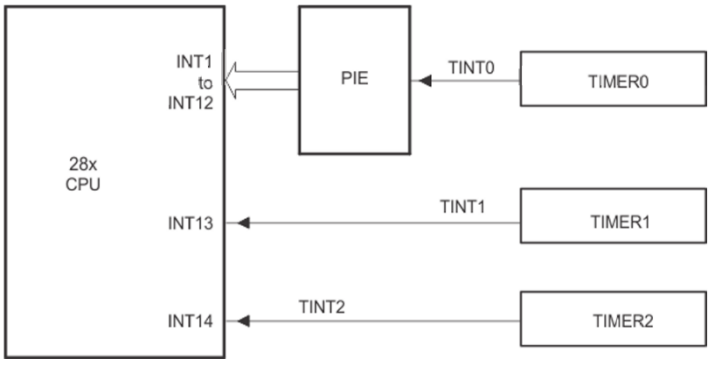
$$T_{TIM} = T_{CPU} * N \Rightarrow N = T_{TIM} * f_{CPU} = 1ms * 120MHz = 120000$$

Tehát az időzítőnek 120000 órajelig kell várnia minden megszakítás generálás között.



1-5.Időzítő beállítása

Mivel a TIMER1 megszakítása közvetlenül van bekötve a CPU-ba, így a megszakításkezelő függvényünkben már nem kell külön törölni a megszakítás flag-jét.



1-6. Időzítő megszakítások bekötése a CPU-ba

2 Referenciák

[1] *Serial Flash Programming of C2000™ Microcontrollers (sprabv4h.pdf)*

C2000™SoftwareControlledFirmwareUpdateProcess (spracn1.pdf)