NCT Ipari Elektronikai Kft
Fejlesztési osztály

Szinuszos enkóder alapú pozició meghatározás

Dokumentáció

Faragó Gyula
faragogyulii@gmail.com

Németh András
nemand1996@gmail.com

2022. január 17.

Tartalomjegyzék

1. Hardveres áttekintés **2**

1.1. Kiinduló áramkör 2

1.2. A jelfeldolgozás elmélete 3

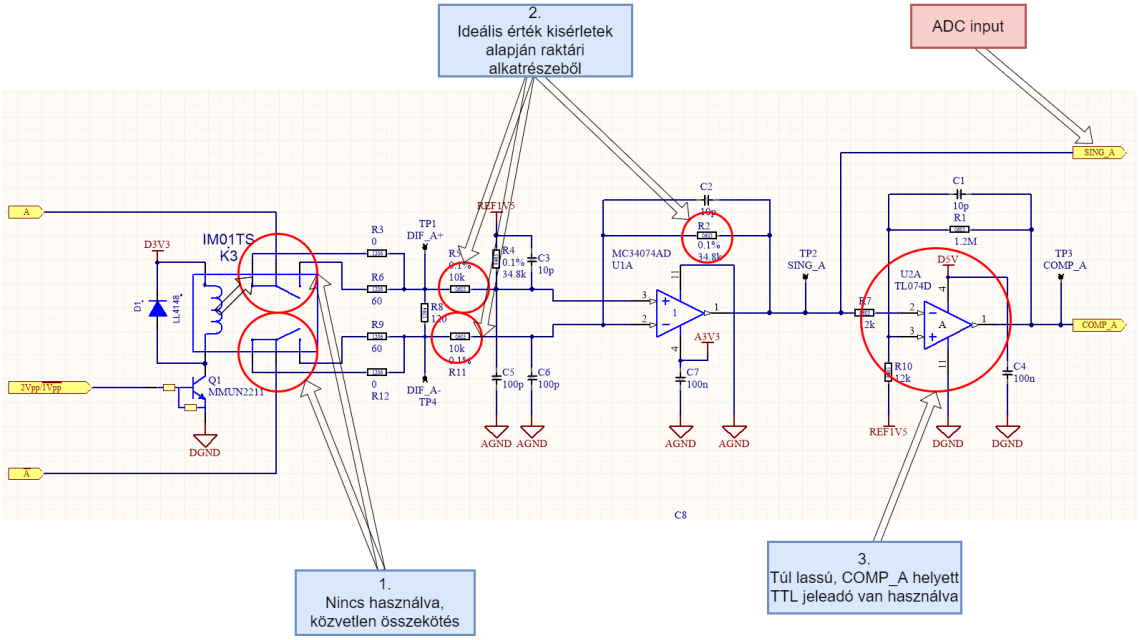
2. Szoftveres megvalósítás **5**

Irodalomjegyzék **12**

1 Hardveres áttekintés

1.1. Kiinduló áramkör

A kezdeti terv amit kaptunk az enkóderből jövő differenciált jeleket szerette volna feldolgozni de szükség volt néhány hardveres változtatásra hogy értelmezhető jeleket kapjunk. A mérések alapján talált észrevételeket és alkalmazott módosításokat az alábbi ábra mutatja (1.1).

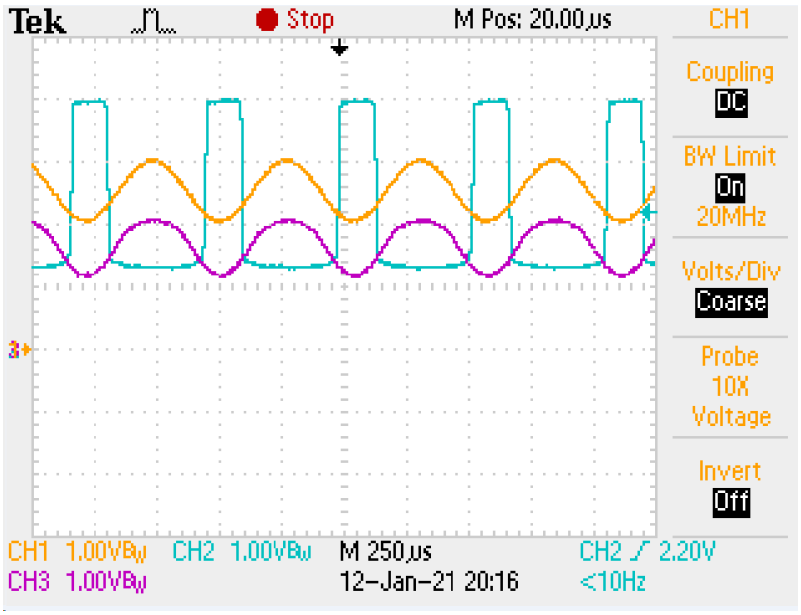


1.1. ábra. *Differenciált jel feldolgozása*

A 2-es pontban jelölt alkatrészekkel a tapasztalat alapján kísérletezni kell mivel a kapcsolásban használt értékek vagy nem erősítették eléggé a jelet vagy éppen túl erősítették ami pozícióhibát okoz. Az 1-es pontban jelölt relés kapcsolás a későbbiekben hasznos lesz, mivel azt tapasztaltuk, hogy különböző típusú enkóderek más-más kapcsolást igényeltek az optimális erősítés érdekében. A

3-as pontban jelölt áramkör eleinte jónak tűnt de nagyobb fordulatszámon ami 1000 RPM-et jelent már aligha volt használható a kapcsolás alul áteresztő jelegéből adódóan. Az áramkör célja a korábbi fokozatból származó szinuszjelből négyszögjel előállítása amit később a QEP modul tud értelmezni (1.2).

Mérésekkel és szimulációval is ellenőriztük, hogy ez a kapcsolás nagyobb fordulatszámon "túl szűk" jelet képez ami már értelmezetlen a QEP számára . A megoldás az utolsó fokozat



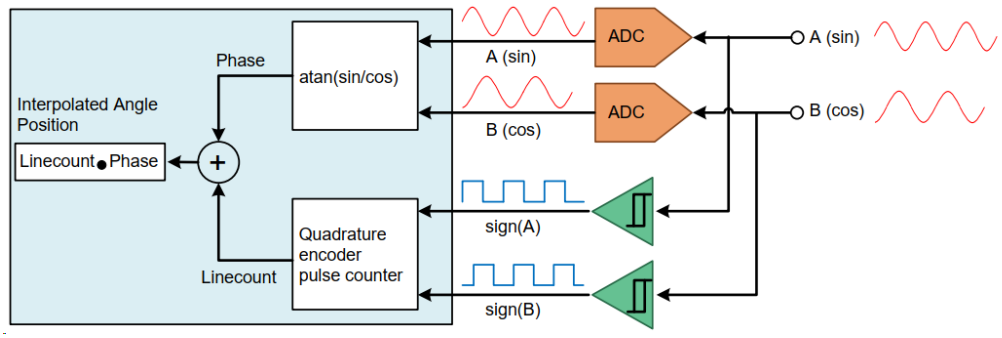
1.2. ábra. *Sárga az eredeti bemenő, lila az első fokozat utáni és kék a második fokozat utáni jel*

elhagyása volt és a DCU-n található TTL jeladó használata a későbbi pozíciószámolásban.

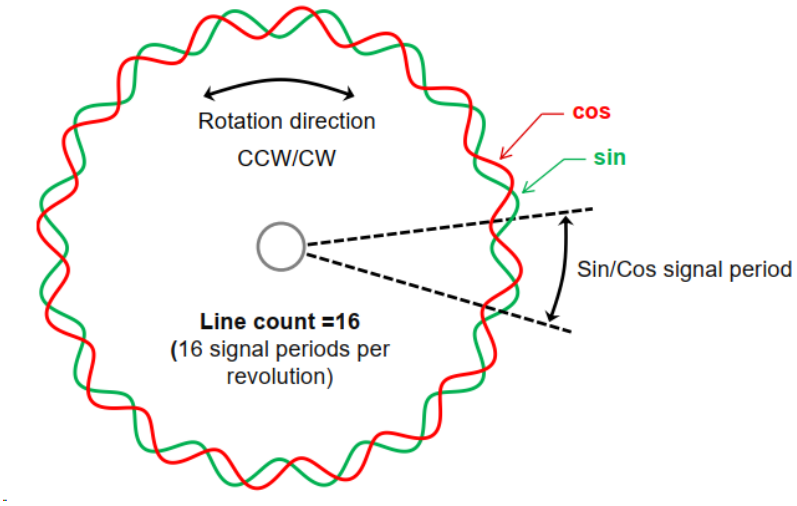
(**TODO : Használt ellenállás értékek, OP ampek, és kondik kiírása**)

1.2. A jelfeldolgozás elmélete

A jelfeldolgozást a TI TIDA-00176 application note [1] alapján végeztük ami többféle javaslatot tesz a megvalósításra. Mi a hardverből adódóan az 1.3-as fejezetben részletezett interpolációs technikát valósítottuk meg. Ez a módszer a QEP által szolgáltatott durva pozíciót tudja pontosabbá tenni két élváltás közt a szinuszos jeladóból származó jelekkel (1.3).



1.3. ábra. *Közelített jel előállítása*



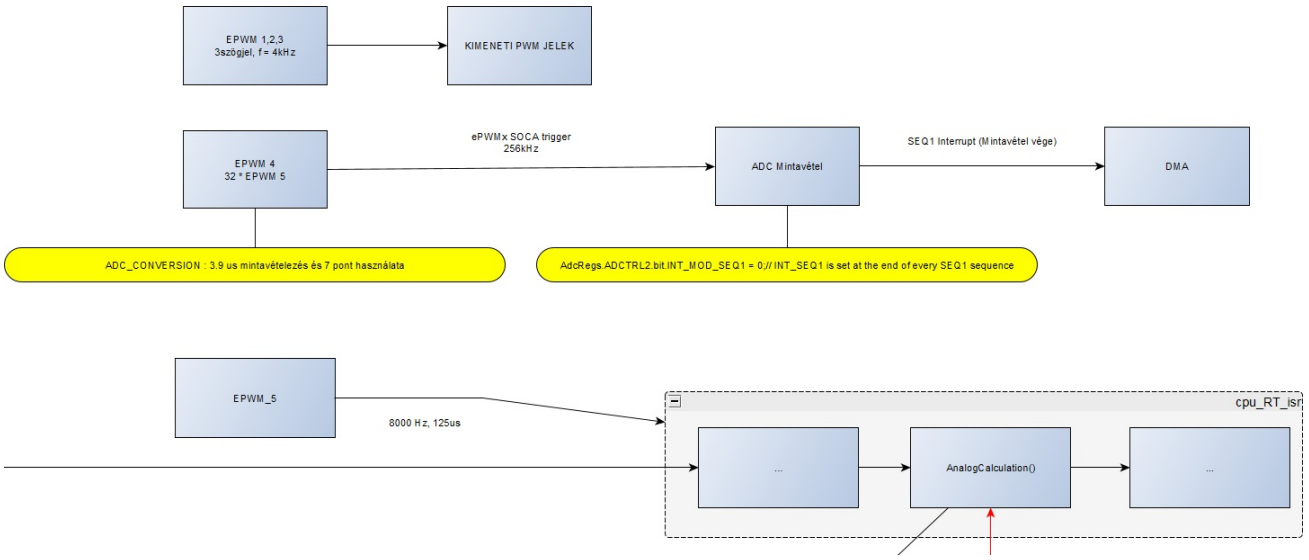
1.4. ábra. *Linecount szemléltetése $N = 16$ esetén*

Az utolsó alapfogalom a későbbi tárgyalásokhoz a linecount ami minden enkóder saját paramétere. Jelentése az egy mechanikai körbefordulás alatt mérhető szinusz jel periódusok száma (1.4).

2 Szoftveres megvalósítás

A szoftveres megvalósítással szemben támasztott alapvető kritérium, hogy ne rontson el semmit ami már korábban működött és a lehető legpontosabb pozíciót adjuk vissza egy függvényen keresztül.

A kapott szoftver három különböző módon használja az elérhető PWM csatornákat (2.1). Az első három csatorna a kimeneti meghajtó áramköröknek szolgáltatnak jeleket. A negyedik 256kHz-el mintavételezi a hajtás számára releváns jeleken. Ezek közül a jelenlegi feladat szempontjából a két szinuszt mintavételező csatorna lesz fontos (valójában 4 db van tehát a kód és a hardver két jeladót is képes üzemeltetni).

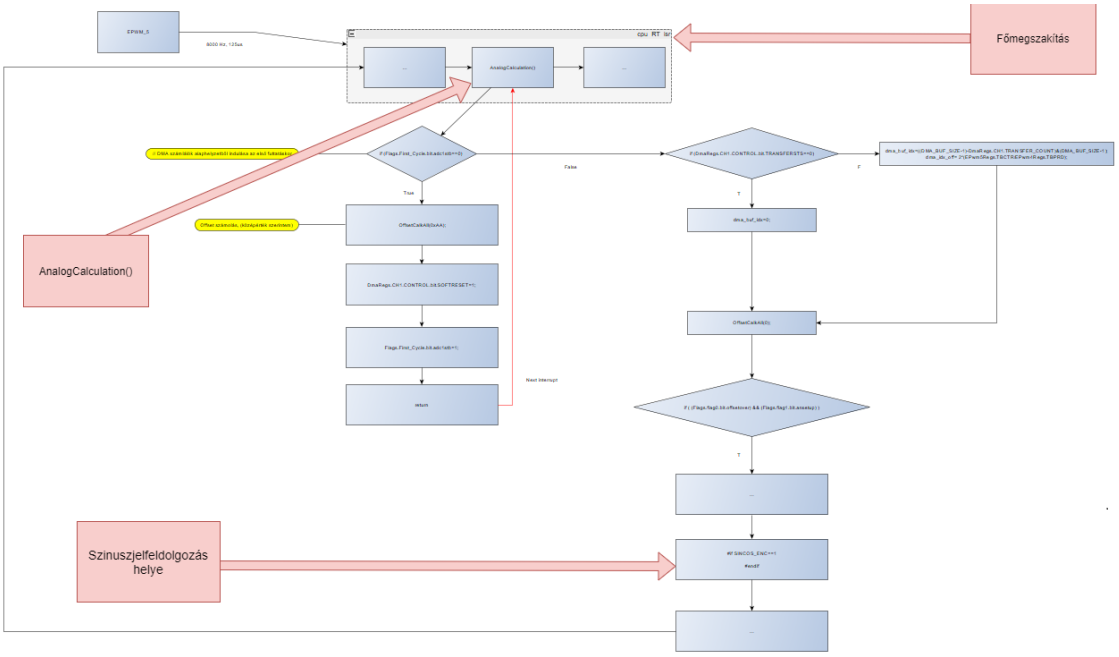


2.1. ábra. *PWM csatornák feladatai*

Mintavételezés végén a DMA automatikusan egy bufferba (*DMABuf1*) rakja a kiolvasott értékeket. A buffer 32 adatokkal teli struktúrát képes tárolni ami a főmegszakítás (*cpu_RT_isr*) 8kHz-es periódusidejéből adódik. A főmegszakításban az *AnalogCalculation()* függvényen belül kellet a szinuszos jeladó olvasásához szükséges részt elhelyezni. A korábbi tesztek során a perifériák saját beállítás alapján működtek de az első bekezdésben leírtak miatt (2) ez most nem volt lehetséges ezért a kódunkat kellett módosítani.

Az *AnalogCalculation()*-n belül a kód helyét a következő ábra mutatja(2.2). Látható, hogy a függvényen belül mélyen, több feltétel után található a kód ami a tesztelést nehezíti. Hardveres tesztek során csak úgy sikerült elérni a szinuszos kódrészlet futtatását, hogy elhitettük a hajtással, hogy minden rendben fut. Ezt a következő bitek sor-

ban történő 1-be állításával lehet elérni: *Q_ counter.I2P*, *Flags.First_ Cycle.bit.ampeeproml*, *Flags.flag0.bit.offsetover*, *Flags.flag1.bit.ansetup*.



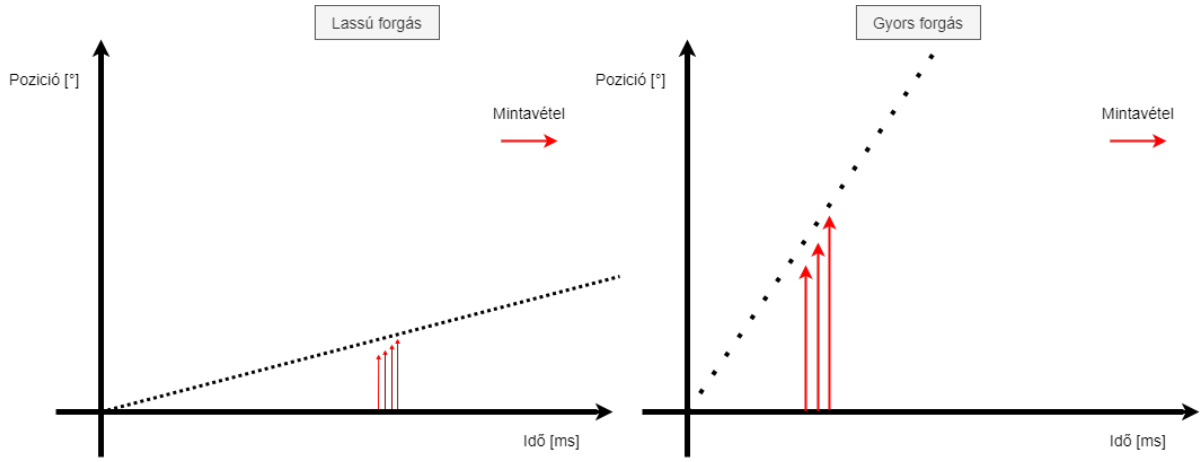
2.2. ábra. *Kód helye az főmegszakításban*

Ha megfelelő a sorrend akkor a hajtás azt hiszi minden rendben van és tesztelhető csak önmagában a jeladó. A kód a *ADCwDMA_ read.c* fájl 1093. sorában kezdődik. A függvény elején rögtön kiolvassuk a QEP számláló értéket az aktuális pozíció minél pontosabb meghatározása miatt.

```
        g_qepCounter = EQep1Regs.QPOSCNT;
```

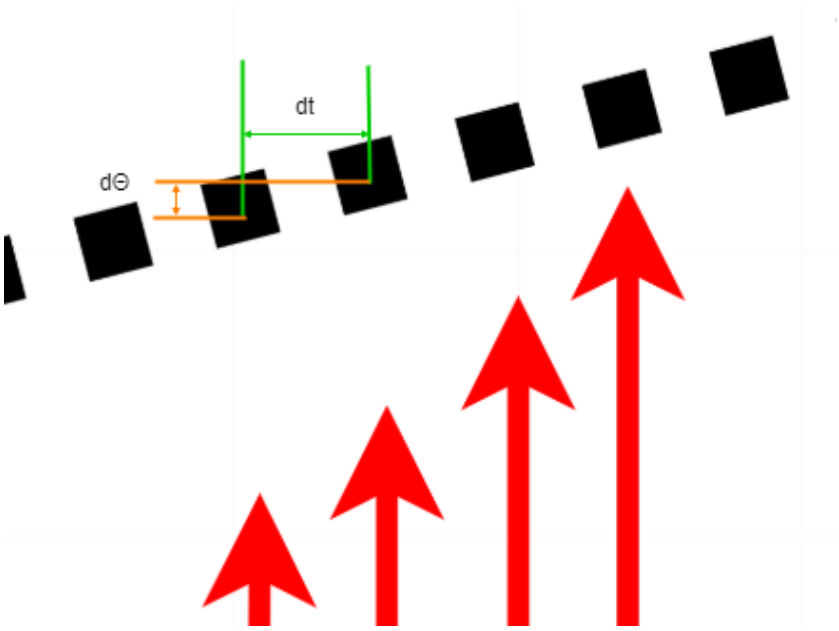
Itt rögtön két megközelítés is alkalmazható. Az egyik ami az eredeti ötlet volt, hogy DMA-n keresztül a szinusz pozíciójának lementésekkor lementjük a QEP számláló értékét is. Ebből következik, hogy amikor eljutunk a kiértékeléshez akkor pontos múltbeli pozícióval tudunk szolgálni. A probléma ezzel a megoldással, hogy nagyobb fordulatszámok esetén az aktuális pozíció és a szinuszos enkóderrel számolt pozíció közti késés növekszik. Ezt úgy lehet kivédeni, hogy a nagyobb fordulatszámon jobban működő TTL jeladót használjuk ekkor, mivel a szinuszos finom pozíciónak itt nincs akkora haszna. Ehhez viszont kellet volna egy függvény ami eldönti, hogy menyire vegyük figyelembe a szinuszos jeladót a jelenlegi fordulatszámtól függően.

Ehelyett nem használunk DMA-t hanem amikor a futás a kódhoz ér rögtön kiolvassuk az aktuális értéket. Ez természetes módon oldja meg a korábbi problémát mivel ha gyorsan forog a motor akkor a helybeli kiolvasás miatt mindig a lehető legaktuálisabb pozícióval dolgozunk (2.3).



2.3. ábra. *Gyors és lassú forgás hatása a mintavételezésre*

Látható, hogy gyors forgás esetén a DMA használatából adódó késleltetés nagyobb hibát fog okozni QEP kiolvasásakor mintha közvetlenül a futáskor olvasnánk ki. Viszont ha lassan forog akkor két pozíció kiértékelés közt (8kHz) aligha változik a QEP számláló állása ezért a finom pozíciós számolás pontos eredményt tud adni (2.4).



2.4. ábra. *Lassú forgás hatása a mintavételezésre*

Ezután megvizsgáljuk először fut-e le a pozíciókiértékelés. Ha igen akkor a *Init_ sincos_ param_ for_ calc* függvényen belül inicializálunk néhány struktúra változót ami később a számításokat fogja segíteni. Leolvassuk a QEP számláló aktuális értékét és elmentjük mint referencia pont , végül pedig tiltjuk a legközelebbi futásnál ennek a *if*-es szerkezetnek a futását.

```
if(sincos_first_run == 1)
{
        Init_sincos_param_for_calculation(); // Init encoder corrections
        sincos_qep_ref = g_qepCounter;
        sincos_first_run = 0;
}
```

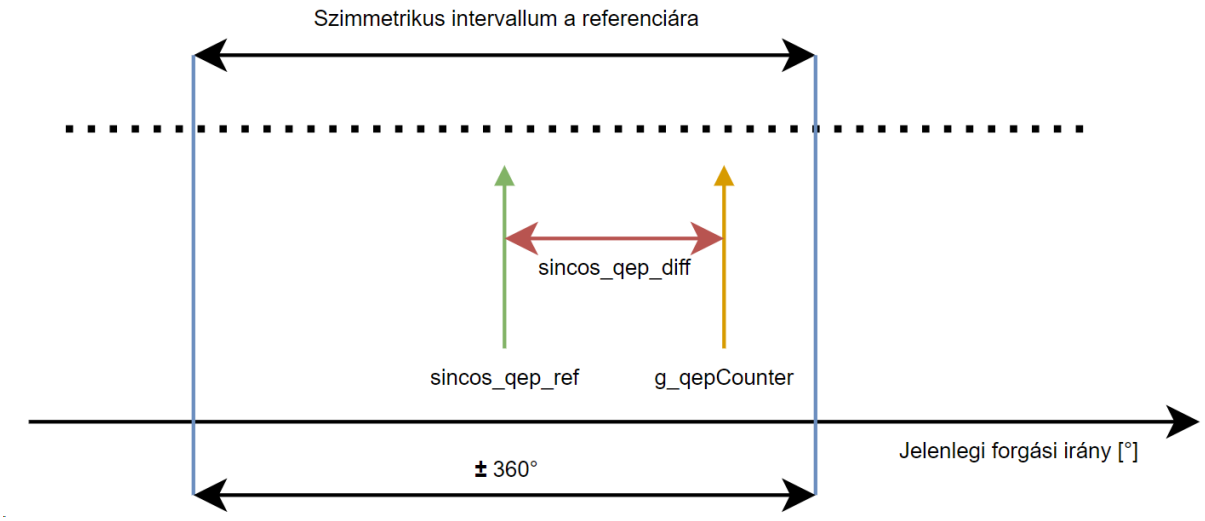
A következőkben megvizsgáljuk a korábban definiált linecount (1.4) alapján, hogy valamelyik irányban történt-e körbefordulás. Erre azért van szükség mert enélkül a függvény nem csak ±360°-ban adna vissza eredményeket.

```
sincos_qep_diff = g_qepCounter - sincos_qep_ref;

if(sincos_qep_diff > sincos_encoder_linecount)
{
        sincos_qep_ref = sincos_qep_ref + sincos_encoder_linecount;
}

if(sincos_qep_diff < -sincos_encoder_linecount)
{
        sincos_qep_ref = sincos_qep_ref - sincos_encoder_linecount;
}
```

A kód működése könnyebben megérthető a következő ábra alapján (2.5). A függvény nem számolja hányszor történt körbefordulás csak a rotor aktuális pozícióját. Pozitív forgási irány esetén ez 0 és 360° közti eredmény lesz, negatív forgási iránynál pedig 0 és -360°közti eredmény.



2.5. ábra. *Referencia módosítás bemutatása*

Ha a QEP számláló kimegy a linecount által meghatározott intervallumból akkor egyszerűen csak eltoljuk a referencia pontot így a körbefordulásokor a szög értéke nem túl 360°-on hanem nullázódik.

A következőben kiolvassuk a DMA-által olvasott utolsó pozíciót [1] és megtaláljuk a szinuszjelek középértékét az *adc_zero_crossing_find()* függvény segítségével.

```
g_AdcChanel_B = DMABuf1[31].enc_ch1cos;
g_AdcChanel_A = DMABuf1[31].enc_ch1sin;

adc_zero_crossing_find();
```

Az *adc_zero_crossing_find()* csak nyomon követi globális változókban az egyik csatorna maximum és minimum értékeit. Ha az eddig tárolt maximumnál nagyobb értéket

[1] ⟨Megvizsgálandó, hogy lehetne használni akár több értéket is és átlagot képezni⟩

olvas az ADC akkor az lesz az új maximum, minimumra hasonló a megvalósítás. Ebből a két értékből átlagot számol amit elment egy harmadik globális változóba. Erre azért van szükség mert az ADC a mérések alapján 38000- 45000 intervallumban méri az értékeket és ebből még nem eldönthető, hogy egy adott szinusz érték éppen negatív vagy pozitív félperiódusban van.

A középérték megtalálás rögtön a következő sorban az arctan kiszámolásánál lesz hasznos mivel így eldönthető, hogy egy adott érték pozitív vagy negatív ami szükséges az arctan számolás helyességéhez.

```
angles.angle_in_fixed_fine = calculate_atan();
fine_angle_correction_for_360_degree();
```

Az arctan függvény egyik problémája, hogy csak $\pm 90°$-ban ad vissza eredményeket de nekünk $\pm 360°$ kell. Ez egy korrekcióval orvosolható, de szükséges hozzá a referenciának választott csatorna előjele. Mi az A csatornát választottuk referenciának és ekkor a korrekció a következő módon valósítható meg.

```
void fine_angle_correction_for_360_degree()
{
        if (shifted_channel_A > 0)
        {
                angles.angle_in_fixed_fine = angles.angle_in_fixed_fine + (1.5707);
        }
        else
        {
                angles.angle_in_fixed_fine = angles.angle_in_fixed_fine + (4.7123);
        }
}
```

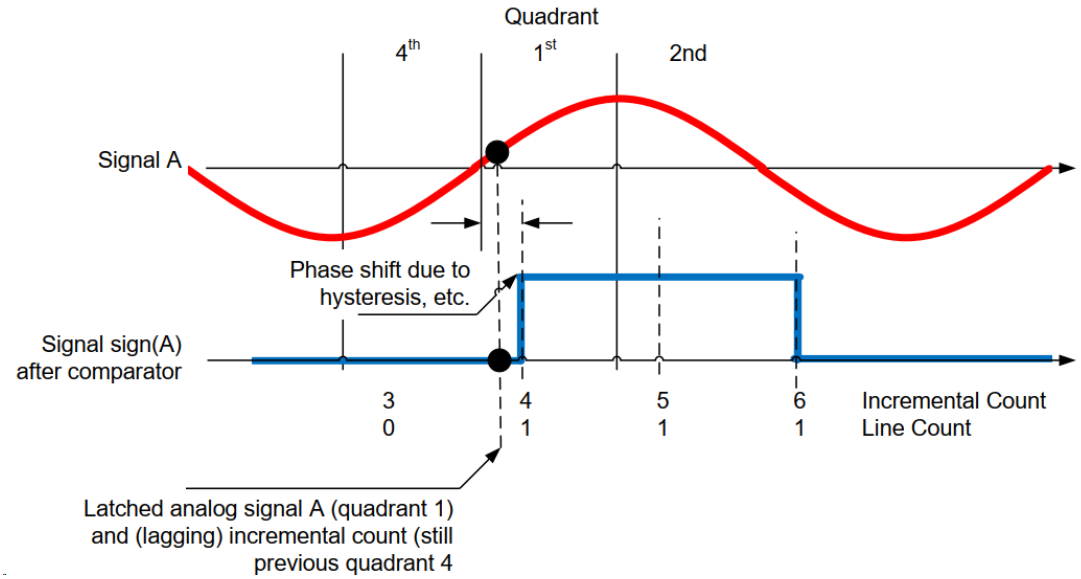
Itt a két mágikus szám a 90° és a 270° radiánban ami a kódban kommentezve is van. Ezzel az ügyes eltolással a $\pm 90°$ intervallum kibővült $\pm 360°$. Az *angles.angle_ in_fixed_fine* a függvény során végig használt finom pozíció értékét jelöli.

A következő függvény az esetleges szinkronizációs hibák ellen hasznos. [2] Ha nem egyszerre történik a QEP és az ADC érték olvasása akkor előfordulhatnak pozícióhibák a számolás során (2.6).

```
QEP_latch_error_fix_due_to_phaseshift();
```

Ez igazából csak akkor jelent problémát ha az 1. negyedből megyünk a 4.-be vagy fordítva, mivel a többen a szinuszjelek által szolgáltatott pozíció lesz a mérvadó [1](1.3.1.5) a QEP csak az egész részt adja. Ebből következik, hogy csak két esetben kell megvizsgálnunk és korrigálnunk az eltéréseket.

[2] ⟨Egyelőre még a kódban van de kérdéses, hogy nem DMA-val megvalósított QEP kiolvasás esetén nem-e okoz problémát. Bár kis fordulatszámon nem okoz elméletileg de ezt még ki kell mérni ⟩

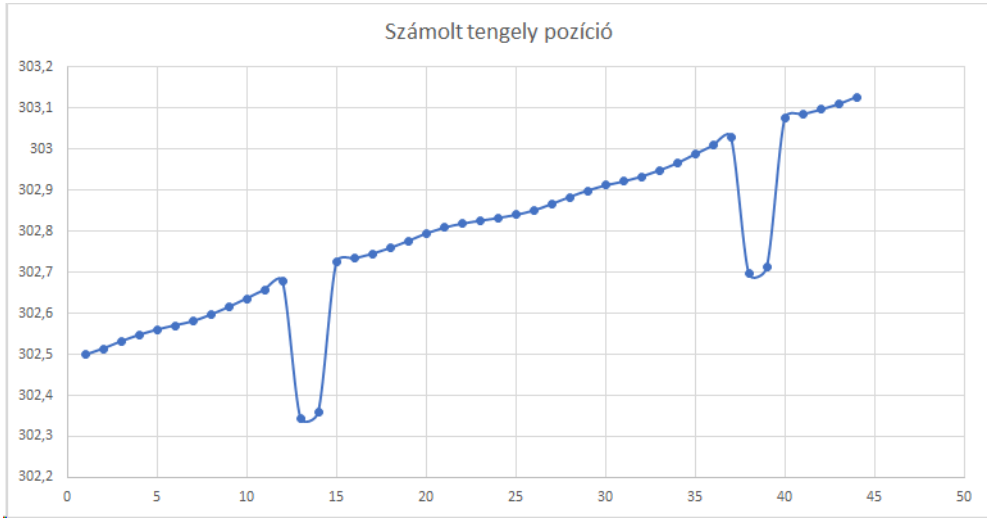


2.6. ábra. *Hibás latch-ből adodó hiba*

Első lépésben meghatározzuk a melyik negyedben vagyunk. Ezután megvizsgáljuk, hogy a QEP számláló el van-e maradva, ha igen akkor korrigálunk rajta és a helyes értékkel számolunk tovább.

```
void QEP_latch_error_fix_due_to_phaseshift()
{
        //FINE IDENTIFICATION
        angles.angle_fine_quadrant = angles.angle_in_fixed_fine/(6.283185307);
        if (angles.angle_fine_quadrant < 0.25)
        {
                if (g_qepCounter % 4 == 3)
                g_qepCounter++;
        }
        if (angles.angle_fine_quadrant > 0.75)
        {
                if (g_qepCounter % 4 == 0)
                g_qepCounter--;
        }
}
```

Ha nem használjuk ezt a korrekciós eljárást akkor az szögugráshoz vezethet amiket a mérések során is tapasztaltunk (2.7). A tesztek alapján a korrekció mindig képes kijavítani a hibás latch-ből eredő hibákat.



2.7. ábra. *Szögugrás hiba korrekció nélkül*

Az utolsó lépés a pontos szög kiszámolása. Ehhez kiszámoljuk a szög egész részét amit a QEP számláló néggyel való osztásával kapunk (*sincos_ qep_ diff » 2*). Utána hozzáadjuk a finom szög értékét ami egy 0 és 360° közti szám radiánban, így logikusan 360°-al osztjuk, hogy törtrészt kapjunk.

```
void calculate_interpolated_high_res_angle()
{
        angles.angle_in_fixed = (sincos_qep_diff >> 2)
        + ((angles.angle_in_fixed_fine)/(6.2831853));

        angles.angle_in_fixed = (angles.angle_in_fixed)
        *(g_sincos_param_for_calculation_values.heid_fine);

        angles.angle = angles.angle_in_fixed;
}
```

Itt még nem vettük figyelembe a jeladó line count-át ami hibát okoz a pozicióban ezért a következő sorban korrigáljuk ezzel az eredményt. A *g_ sincos_ param_ for_ calculation_ values.heid_ fine* egy heidenhain jeladóra kiszámolt paraméter amit úgy kapunk ha 2π-t leosztjuk a linecount értékével. Mivel ez egy magic number ami csak zavarja az olvashatóságot ezért egy struktúrába lett eltárolva.

Ezek az eltárolt értékek a *g_ sincos_ param_ for_ calculation_ values* struktúrában vannak és ki vannak számolva tamagawa jeladókra is.

```
void Init_sincos_param_for_calculation()
{
        g_sincos_param_for_calculation_values.heid_fine =  0.1757812;
        g_sincos_param_for_calculation_values.heid_coarse =  0.043945312;
        g_sincos_param_for_calculation_values.yama_fine =  0.3515625;
        g_sincos_param_for_calculation_values.yama_coarse =  0.08789062;
}
```

Utolsó lépésben elmentjük a kiszámolt értéket egy globális változóban.

Irodalomjegyzék

[1] Texas Instruments. Interface to sin/cos encoders with high-resolution position interpolation reference design. `https://www.ti.com/tool/TIDA-00176`.