Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Szoftver-definiált hőmérséklet kompenzált oszcillátor fejlesztése

TDK Dolgozat

Készítette
Vozár Viktor

Konzulens
Dr. Kovácsházy Tamás

2022. november 1.

Tartalomjegyzék

Kivonat

Az elosztott rendszerek térnyerésével manapság egyre nagyobb jelentősége van az óraszinkronizációnak. Ennek egyik alapvető feltétele a pontos lokális órajel előállítása. A rezgőkvarcok hibájának legfőbb forrása a hőmérséklet ingadozása. A hiba kompenzációjára számos módszer létezik, azonban a pontosság növekedésével ezek ára drasztikusan emelkedik, továbbá mindegyik eljárásnak megvannak a maga korlátai. Ügyes megoldásokkal azonban kiválthatók a drága hardver elemek egy olcsó, kompenzálatlan oszcillátor és egy hőmérő kombinációjával.

Korábbi vizsgálataim során, a szakdolgozatomban már bemutatott mérőrendszerrel felvettem egy kristály oszcillátor karakterisztikáját, és offline szimulációkkal sikerült a frekvenciahibát egy TCXO pontosságával megbecsülnöm. Erre alapozva felmerült egy valós időben, tisztán szoftveresen működő hőmérséklet kompenzált oszcillátor megvalósításának a lehetősége, mely jelen TDK dolgozatom témáját képzi. A hálózati óraszinkronizációt, mint referenciaértéket kihasználva lehetőség nyílik a hőmérő hibajele, és a frekvenciát módosító beavatkozó jel közötti kapcsolat megteremtésére és kalibrálására.

Dolgozatomban bemutatom az elképzelt rendszer működésének elvét, illetve a felhasznált hálózati óraszinkronizációt, mely a tanszéken már megvalósításra került beágyazott rendszerekre is. Ezt követően bemutatom az egyes hardver elemek kiválasztását indoklással, majd ellenőrzőm azok működését.

A létrejövő rendszeren több kompenzációs eljárást implementáltam. Bemutatom ezek elméleti hátterét, a megvalósítás menetét, majd tesztekkel ellenőrzöm azok hatékonyságát. Az eredményeket értelmezem, összehasonlítom, és levonom a következtetéseket. Az összehasonlításokból kiderül, hogy a megoldás mennyire állja meg a helyét a piacon kapható vetélytársakkal szemben, és hogy milyen előnyt nyújt használata ezekkel szemben.

Abstract

With the widespread of distributed systems, clock synchronization is becoming increasingly important today. One of the basic conditions for this is making an accurate and precise clock signal. The main source of error in the quartz crystal is the fluctuation of temperature. There are many ways to compensate this error, but as precision increases, their cost increases dramatically, and each solution has its limitations. However, clever solutions can replace expensive hardware with a combination of an inexpensive, uncompensated oscillator and a thermometer.

In my previous investigations, I recorded the characteristics of a crystal oscillator with the measuring system already presented in my bachelor's thesis, and I was able to estimate the frequency error with the accuracy of an average TCXO using offline simulations. Based on this, the possibility of implementing a real-time, purely software-compensated oscillator arose, which is the topic of this paper. If we use clock synchronization from the network as a reference value, we can establish and calibrate the connection between the fault signal of the thermometer and the control signal that modifies the frequency.

In my dissertation I present the principle of the imagined system operation and the used network clock synchronization, which has already been implemented for embedded systems. Next, I present the selection procedure of each piece of hardware with justification, and then check their operation.

I implemented several compensation methods on the resulting system. I present their theoretical background, the implementation process, and then check their effectiveness with multiple tests. I interpret the results, compare them and then I draw conclusions. The results show how it compares to competitors on the market and what advantages it offers over them.

1. fejezet

Bevezetés

Az elosztott rendszerek térnyerésével egyre több olyan biztonságkritikus alkalmazással találkozhatunk, melyektől elvárjuk a valós idejű működést. Számos példát láthatunk ezekre a telekommunikáció, kritikus ipari folyamatok, és a járműelektronika területén. Egy több bemenetű, több kimenetű rendszer csak akkor működik jól, ha a bemeneti mérések és a beavatkozás elég nagy pontossággal ugyanabban az időpontban történik. Ellenkező esetben a jel már más, ami hibát okoz a számítások során, pl. egy szabályzó algoritmusban. Ezen kívül sok esetben szükséges, hogy egy elosztott rendszerben lévő, különböző node-okon történő események között mérjünk eltelt időt, ami szintén szükségessé teszi az órák szinkronizációját. A valós idejűség ennek megfelelően nem képzelhető el óraszinkronizáció és idő-vezérelt működés nélkül. Mindezek megvalósításához a szükséges garanciákat az RT-CPS (Real Time Cyber-Physical System) megközelítés tudja biztosítani. A garancia időbeli, tehát szükséges a közös idő, az órákat szinkronizálni kell. Ennek az egyik alapvető feltétele, hogy rendelkezzünk stabil lokális oszcillátorokkal [1].

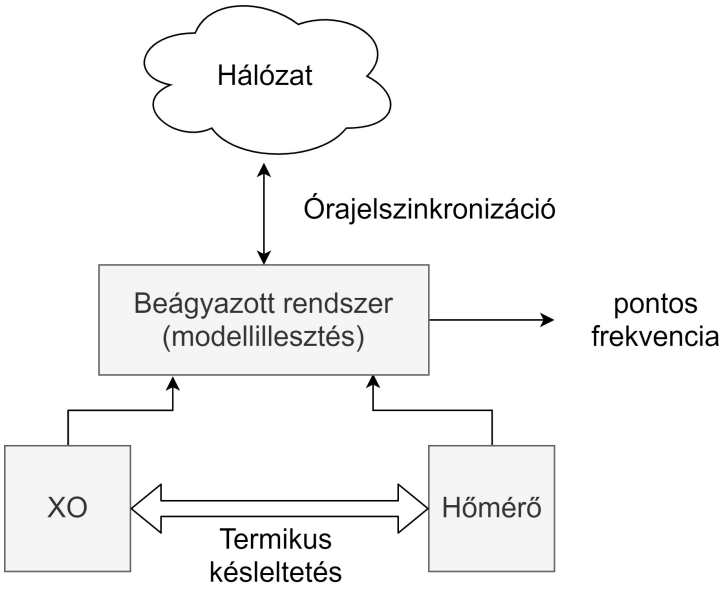
Oszcillátoraink legfőbb hibaforrása a hőmérséklet változása. Ezt okozhatja a levegő hőmérsékletének ingadozása, a légáramlás, az elektronika belső hőforrásai (pl. CPU-k, tápok), a rendszer hőmérséklet-menedzsment megoldásai (ventilátorok, hűtőbordák), és számos egyéb, alkalmazástól függő termikus folyamat. A rendszerben tehát létrejön egy zavarás, mely hatására elcsúszik a frekvencia és az idő. Az elcsúszás drift jellegű, időben változó jelenség. A hőmérséklet által okozott változások kikompenzálására számos megoldás született. Az ilyen eszközökkel szemben támasztott követelmények egyre szigorúbbak. A leginkább elterjedt frekvencia kompenzációs megoldások a TCXO[1], illetve az OCXO[2] alkalmazása. Azonban mindkét megoldásnak megvannak a maga hátrányai, illetve a pontosság növekedésével ezen eszközök ára is drasztikusan növekszik.

A BSc szakdolgozatomban megvizsgáltam a legelterjedtebb hőmérséklet kompenzációs megoldások működési elvét, bemutattam azok előnyeit és hátrányait, illetve megterveztem és implementáltam egy mérőrendszert, mely alkalmas az oszcillátorok frekvencia-hőmérséklet összefüggésének vizsgálatára. Felvettem egy tipikus

[1] Temperature Compensated Crystal Oscillator
[2] Oven Controlled Crystal Oscillator

(kompenzálatlan) kvarc oszcillátor karakterisztikáját, melyet összevetettem a fentebb említett TCXO-val, illetve OCXO-val. A mérési eredmények alapján egy offline szimulációban elkészítettem a rendszer modelljét, mellyel sikerült megbecsülni a hőmérséklet által okozott hibát egy TCXO pontosságával.

A mérési eredményekből született cikkben [2] felvetésre került egy szoftver-definiált hőmérséklet kompenzált oszcillátor (SD-TCXO) megvalósításának lehetősége, amely kihasználva az óraszinkronizáció miatt folyamatosan elérhető frekvencia referenciát és a hőmérséklet mérését, egy online identifikációs szoftver megoldással tesz javaslatot a frekvencia pontosság javítására. Jelen dolgozat fókusza ennek a javaslatnak az implementációja és vizsgálata valós környezetben. Az eljárás kulcsfontosságú részeit emeli ki az 1.1. ábrán látható blokkdiagram.



1.1. ábra. *A modellillesztést alkalmazó rendszer blokkvázlata*

A tanszéken megvalósításra került egy hálózatba kapcsolt beágyazott rendszereken elérhető óraszinkronizációs eljárás, mely *Precision Time Protokollt* (PTP) használ erre a célra [3]. Az eszköz a pontos időt Etherneten keresztül kéri le egy mesterórától. A szinkronizációs üzeneteket a mesteróra általában másodpercenként küldi multicast címzéssel. A kompenzáció tehát csak másodpercenként mehet végbe. Egyszerű topológiáknál természetesen növelhető a *Sync* üzenetek gyakorisága, azonban nagy hálózatokban a túlterhelés elkerülése végett ez a gyakoriság nem növelhető tetszés szerint. Hálózati hiba esetén továbbá a mesteróra nem elérhető. Éppen ezért az volna az ideális, ha két szinkronizáció között, illetve offline állapotban a lokális óra képes lenne minél jobban megőrizni a pontos időt, akár kisebb korrekciókat is elvégezve saját magán. Dolgozatomban bemutatom a már meglévő eljárás működését, illetve megkísérlem továbbfejleszteni az megoldás pontosságát a hőmérsékletmérési méréseimből levont következtetéseket szem előtt tartva.

A cél egy olyan valós idejű hőmérsékletkompenzációs eljárás megalkotása, amely pontossága a TCXO-k nagyságrendjébe esik, ám nem igényel költséges alkatrészeket

(gondoljunk csak bele, hogy egy átlagos TCXO 1000 Ft körül mozog, ezzel szemben egy termisztort már 10-20 Ft-ért is kaphatunk), illetve adaptivitása révén nincs kiszolgáltatva az olyan hibaforrásoknak, mint az eszköz öregedése, vagy a gyártási szórás.

1.1. Az adaptáció szintjei

Fontos beszélni a valós-idejűség lépéseiről. Megkülönböztetjük azt az esetet, amikor a csak a szabályzás történik valós időben, illetve azt, amikor a modell adaptációja is. A BSc szakdolgozatomban a modelleket egy off-line szimulációban validáltam. A következő lépés, hogy ezeket a modelleket átültetjük egy célhardverre, és megvizsgáljuk, hogy valós időben is képes-e kompenzálni a frekvenciahibát. Jelen munkám témája tehát a szimulációs eredmények alapján egy valós időben is működő szabályzót létrehozni.

Az adaptáció egy ennél magasabb szint, mely azt jelenti, hogy a szabályzó rendszer futási időben képes változtatni saját paramétereit, hiszen elképzelhető, hogy a fizikai rendszer idővel változik. Ahhoz, hogy a modell ezt a változást lekövesse, folyamatosan hangolnia kell magát a becslési hiba alapján. Ehhez az szükséges, hogy rendelkezésre álljon egy pontosnak elfogadott referenciajel, mely alapján korrigálni tudja önmagát.

Az adaptáció futhat közvetlenül az eszközön, de elképzelhető olyan megoldás is, hogy a beágyazott rendszer elküldi a gyűjtött adatokat a *felhőbe*, ahol egy nagyobb számítási kapacitással rendelkező PC végzi el a modellillesztést, majd visszaküldi a korrigált modellparamétereket az eszköznek. Jelen dolgozatban bemutatott eljárás ennek *előfutára*, ugyanis itt a modellillesztés manuálisan, több lépésben történik. Először adatot gyűjtök különböző hőmérsékleti gerjesztésekre, melyeket aztán Matlab segítségével feldolgozok, és modellt illesztek rá, végül a kapott modellt megvalósítom a beágyazott rendszeren, C nyelven.

A vizsgálataim során egy egyszerű adaptációs eljárást is sikerült letesztelnem, azonban ez még csak az offszethiba kikompenzálására alkalmas.

1.2. A dolgozat felépítése

Az elméleti áttekintésben röviden összefoglalom az oszcillátorok működésének elvét, illetve a következtetéseket, melyeket a korábbi vizsgálataim során vontam le. A TCXO alkalmazása helyett felvetek egy új, szoftver definiált megközelítést a kompenzációra, majd bemutatom az ehhez szükséges óraszinkronizációs eljárás egy már implementált változatát, a tanszéken készített *FlexPTP* algoritmust, mely beágyazott rendszereken oldja meg az óraszinkronizációt Precision Time Protocol alkalmazásával. Ezt követően kitérek az algoritmus bővítési lehetőségeire, illetve arra, hogy milyen eljárásokat érdemes alkalmazni a hőmérséklet kompenzációjára.

Az ezt követő fejezetben leírom a rendszer megvalósításának menetét. Kitérek a megfelelő mikrokontroller kiválasztásának szempontjaira, illetve megvizsgálom a jelenleg kereskedelmi forgalomban kapható hőmérő eszközöket aszerint, hogy jelen alkalmazásba melyik volna a legmegfelelőbb. Bemutatom a választott eszköz (egy termisztor) beépítésének menetét, illetve megvizsgálom a hőmérsékletmérés lehetséges hibapontjait

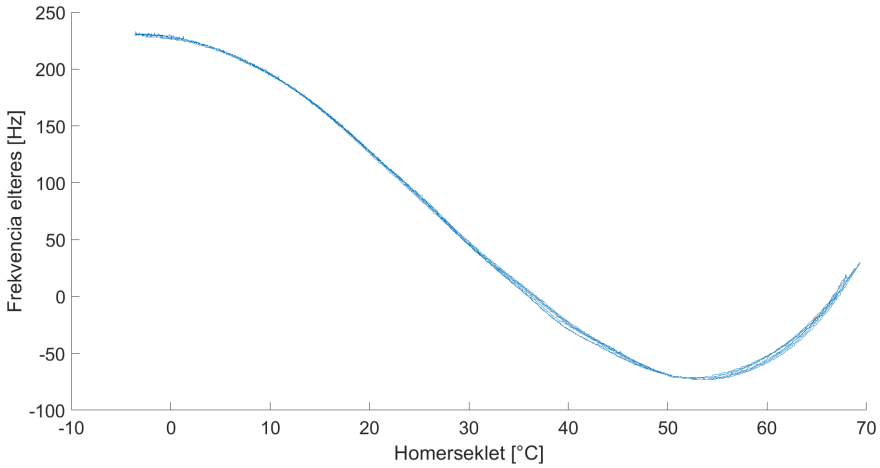
az adott alkalmazásban. A későbbi hibadetektálás végett egy infravörös hőmérőt is beépítettem az áramkörbe. A két eszköz működtetéséhez szükséges szoftver releváns részleteit is megemlítem. A termisztorra elkészítek egy zavarvizsgálatot, és megvizsgálom, hogy hogyan lehet az adott célhardverből különböző mérési praktikákkal a lehető legpontosabb mérési eredményt kihozni. Ezt követően bemutatom az általam kibővített FlexPTP algoritmus releváns részeit, mellyel az adatgyűjtést elvégeztem. A begyűjtött adatokat Matlab segítségével dolgoztam fel, illetve a modellillesztést is itt végeztem el. A kapott modell C kódját megírtam a mikrokontrolleren, melynek hatékonyságát számos teszttel ellenőriztem. Ezeket ábrák kíséretében szemléltetem. A dolgozat végén összegzem az eredményeket, valamint összehasonlítom a modellek hatékonyságát egymással, illetve a piaci vetélytársakkal. Végül megemlítem azokat a továbbfejlesztési irányokat, melyek mentén elindulva további javulás várható a modell pontosságát illetőleg.

2. fejezet

Elméleti áttekintés

2.1. Oszcillátorok működésének összefoglalása

Ezt a témát a szakdolgozatomban részletesen bemutattam [1], így a működés részleteire nem térek ki, csak röviden összefoglalom az ottani következtetéseket. A legegyszerűbb működésű kristály oszcillátorok (XO) pontossága manapság 50 – 100 ppm[1] körül mozog. Ez számos alkalmazáshoz elég, azonban a technológia fejlődésével egyre többször támad ennél nagyobb precízióra igény. Előnyük viszont, hogy áruk alacsony, illetve hogy a frekvencia-hőmérséklet karakterisztikájuk jól közelíthető egy harmadfokú függvénnyel. A 2.1. ábrán látható a szakdolgozatomban vizsgált oszcillátor kimért karakterisztikája, mely valóban egy harmadfokú polinommal írható le a legjobban.



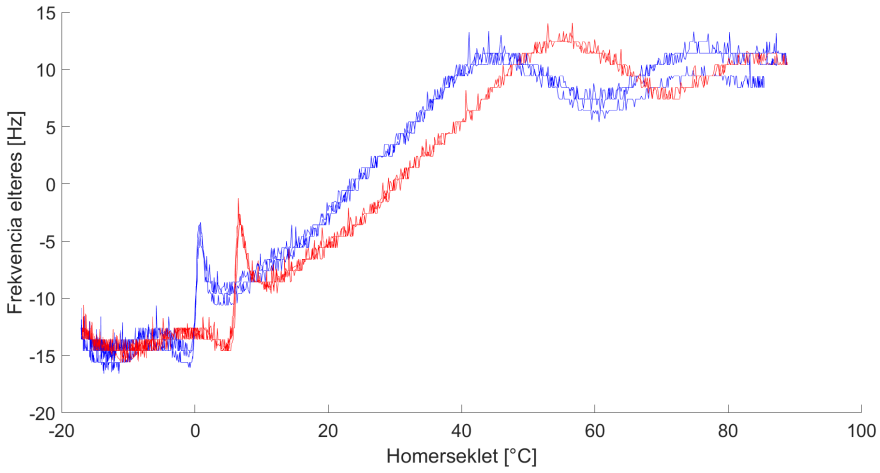
2.1. ábra. *A vizsgált kristály oszcillátor karakterisztikája [1]*

Az ábra alapján az az ötletünk támadhat, hogy mérjük meg a hőmérsékletet, és az ismert harmadfokú függvény alapján határozzuk meg a frekvenciát. Ezzel lényegében rátértünk a hőmérséklet-kompenzált kristály oszcillátorok, azaz TCXO-k vizsgálatára, hiszen ezek működése is hasonló elven történik. Ezen eszközökben egy hőmérő hibajelet állít elő, mely hatására egy beavatkozó áramkör megváltoztatja az oszcillátor valamely paraméterét úgy, hogy a frekvenciát mindig egy adott hibahatáron belül tartsa. A beavatkozás sokféle lehet.

[1]parts per million

Létezik olyan eljárás, ahol a hibajelből egy analóg szorzóáramkörrel állítják elő a megfelelő beavatkozó jelet (például a karakterisztika inverzét), mellyel egy VCO[2]-t vezérelnek [4]. Mivel a VCO frekvenciája a bemeneti feszültségtől függ, azt a hőmérséklet függvényében mindig úgy változtatják, hogy a frekvencia közel állandó maradjon. Létezik olyan megoldás is, ahol a hőmérő hibajelét egy EPROM-ban tárolt táblázat alapján megfeleltetik egy beavatkozó jelnek, mellyel a rezgőkvarc áramkörében található kondenzátor kapacitását változtatják, így hatva a kimeneti frekvenciára [5]. Ezt a megoldást szokás DTCXO-nak, azaz digitális TCXO-nak is nevezni.

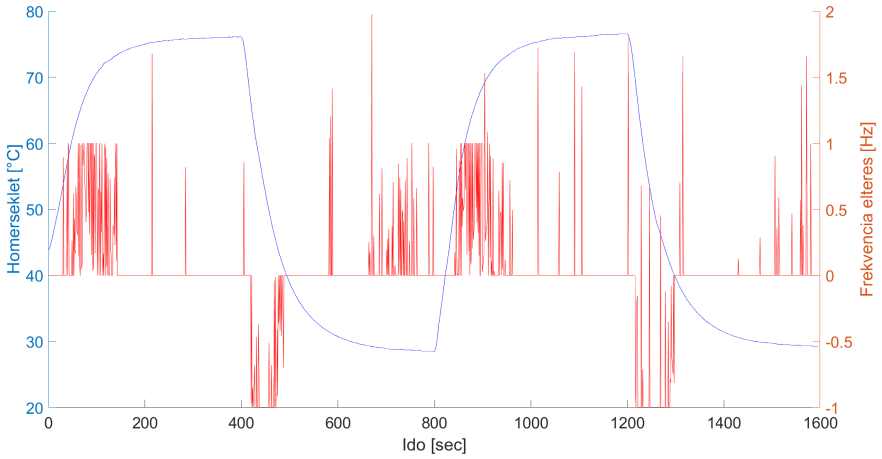
A fenti eljárások lényeges javulásokat hoznak be, a hiba 0.5 – 5 ppm környékén mozog, azonban előjönnek olyan problémák, mint a jitter, vagy fáziszaj. A TCXO belső működése nem definiált, egyedül azt garantálja, hogy a frekvencia az adott hibahatáron belül marad, azonban a beavatkozások során a karakterisztikában lépcsős ugrások, illetve hiszterézis jelenik meg. A szakdolgozatomban vizsgált TCXO f – T görbéje a 2.2. ábrán látható. Jól látszik, hogy ez a görbe már közel sem olyan szép, mint a kompenzálatlan oszcillátor esetén. Az ábrán piros szín jelzi az eszköz melegedési szakaszát, kék pedig a lehűlést. Jól látszik, hogy az f-T diagram a két irányban *nem ugyanazt az utat járja be*, azaz egy hiszterézis jelentkezik a karakterisztikában. A rendszeridentifikáció egy ilyen görbére nem, vagy csak nagyon bonyolult modellekkel lehetséges.



2.2. ábra. *A vizsgált TCXO karakterisztikája [1]*

Ennél még nagyobb pontosságot érhetünk el, ha áttérünk az OCXO, azaz az *Oven Controlled Crystal Oscillator* használatára. Ezen eszközök működésének lényege, hogy egy fűtőszállal felmelegítjük a kvarckristály egy állandó hőmérsékletre (leggyakrabban 80-100 °C közé), és ezen az értéken tartjuk működés közben. A kristályokat jellemzően úgy választják ki, hogy ezen a tartományon a karakterisztikájuk közel vízszintes legyen. Ezzel a megoldással függetlenítettük az eszközt a környezeti hőmérséklet tranzienseitől. Az eljárás oly hatékony, hogy a pontosság gyakran ppb[3] nagyságrendű. A szemléletesség végett 1 ppb hiba másodpercenként 1 nanoszekundum tévedést jelent. Ez egy év alatt is mindössze 32 milliszekundum hibát okoz.

[2]Voltage-controlled oscillator
[3]parts per billion



2.3. ábra. *Az OCXO hibája hirtelen változásokra [1]*

A 2.3. ábrán látszik, hogy az általam megmért OCXO még a legdrasztikusabb hőmérsékleti tranzienseknek is ellenállt. A jobb oldali skálán láthatjuk, hogy a legnagyobb eltérés 2 Hz értékű volt. Az oszcillátor 16.38 MHz frekvenciájú, tehát ez 120 ppb-nek felel meg. Az ábrán nincs offszet, az ugrások zajszerűek. Ráadásul a frekvenciamérő műszer kvantálási egysége 1 Hz, azaz a hiba olyan kicsi, hogy nem volt módom pontosan kimérni.

Az OCXO rendkívül magas pontossága azonban rendkívül magas árral is jár. Egy ilyen darab minimum 20-30 ezer Ft-ba kerül. A fűtőszál alkalmazása miatt egyrészt jóval nagyobb a fogyasztása, mint egy TCXO-nak, másrészt pedig a környező áramköri elemeket is melegíti, melyre a tervezés során külön figyelmet kell fordítani. Ezen felül fizikai mérete is nagyobb az átlagos órajel generátoroknál, így nem lehet akármilyen áramkörben használni.

Amennyiben ennél is magasabb pontosságra van igényünk, szóba jöhetnek a chipméretű atomórák (CSAC[4]). Ezek olyan csúcstechnológiás készülékek, melyek használata csak néhány, extrém pontosságot igénylő alkalmazásban indokolt. A pontosságnak persze ára van: ezen eszközök darabját több ezer amerikai dollárért forgalmazzák. Sajnos a dolgozatomban vizsgált módszerrel ezt a szintet nem lehet elérni, a cél *csupán* a TCXO-k pontosságának megközelítése.

Az általam vizsgált módszer lényege (az eddig bemutatottakkal ellentétben) a kompenzáció tisztán szoftveres végrehajtása. A BSc szakdolgozat során a mérési adatokból elvégeztem a rendszer modelljének offline identifikációját, mely a frekvenciát a hőmérséklet alapján 2 – 3 ppm pontossággal képes volt megbecsülni. A rendszert Hammerstein-Wiener modellel közelítettem, melynek paramétereit a Matlab System Identification Toolbox applikációjával [6] határoztam meg a mérési adatok alapján.

Ezt további mérésekkel finomítottam az Önálló laboratórium 1, MSc első féléves tárgy ideje alatt, ahol megvizsgáltam az identifikáció hatékonyságát különböző hőmérsékleti gerjesztésekre, illetve két hőmérőt használva az oszcillátor körül (MISO[5] rendszer). Ez által a hiba 1 – 2 ppm-re csökkent. Amennyiben a modellparaméterek beállítása valós

[4]Chip scale atomic clock
[5]Multiple Input Single Output

időben, közvetlenül a hardveren megtörténne, úgy egy valós alkalmazásban is elképzelhető adaptív megoldáshoz jutnánk, mely képes kiváltani a TCXO-t bármiféle költséges alkatrész használata nélkül.

A szoftveres kompenzációhoz ismerni kell a céleszközön működő hardveróra felépítését, melyet az eszköz kiválasztásának részletezésekor mutatok be. A működés lényege röviden, hogy a hardveróra két részből áll: egy frekvenciaosztásért felelős részből és magából az órából, azaz egy számláló regiszterből, ami az időt tárolja [7]. A frekvenciaosztó egy visszacsatolt összeadó, mely egy akkumlátor regiszter értékét minden oszcillátor ciklusban megnöveli egy előre megadott értékkel. A túlcsordulás jelzést felhasználhatjuk a kimeneti órajel léptetésére. Az időhibát pontosan mérve pedig a pontos órajelnek megfelelő léptetési egységet állíthatunk be.

Az akkumulátor regiszter léptetési egységének változtatásával lehetőségünk van az oszcillátor belső működésének megváltoztatása nélkül, tisztán szoftveresen végrehajtani a kompenzációt. A kvarc frekvenciája tehát adott (a környezet, pl. hőmérséklet határozza meg). A frekvenciahibát az időhibán keresztül mérjük, és a lokális óra léptetési egységének állításával semlegesítjük a hatását. A frekvenciát tehát közvetlenül nem állítjuk.

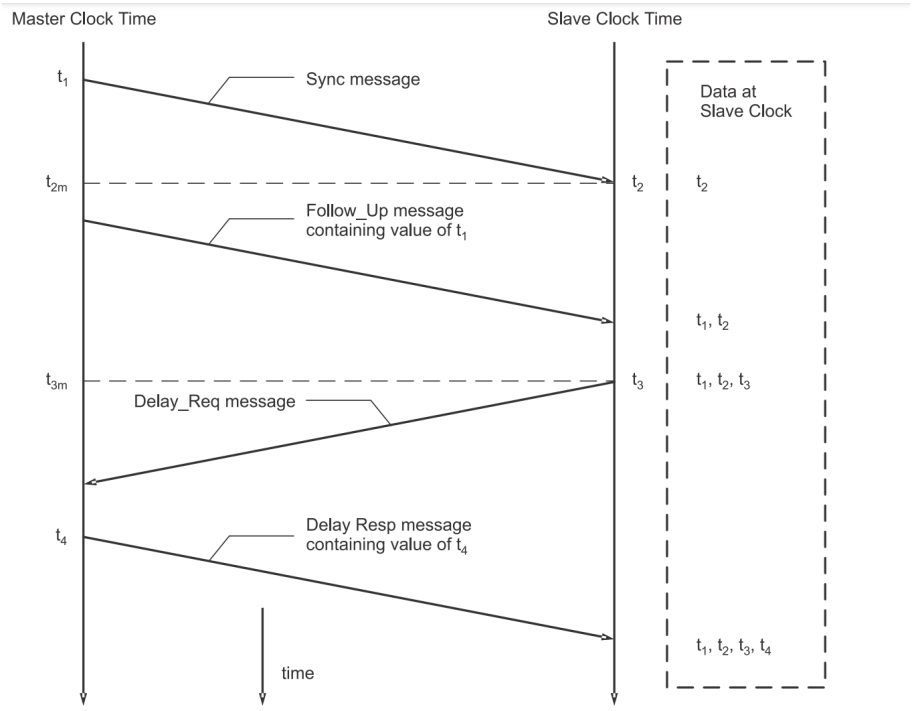
Ez azért előnyös, mert így egy alacsony jitterrel és fáziszajjal rendelkező kvarc oszcillátort is használhatunk TCXO helyett, és hasonló pontosságot érhetünk el vele.

Emellett tetszőleges szabályzó algoritmus implementálható a hőmérő jele és a léptetési egység közötti megfeleltetésre. Ráadásul tetszőleges gyakorisággal változtathatunk ezen az algoritmuson. Az eszköz tehát működhet adaptívan, így nincs kitéve például az áramkör öregedésének, mely a klasszikus TCXO egységekben a frekvencia elcsúszását eredményezi.

Az adaptív működéshez az kell, hogy az eszköz képes legyen egy referenciaforrásból megállapítani a helyes időt, és ahhoz igazítani a szabályzó algoritmust. Erre a feladatra alkalmazható egy, a tanszéken már elkészült óraszinkronizációs algoritmus, mely Precise Time Protocol (PTP) segítségével szinkronizálja a lokális órát a hálózaton elérhető mesterórához.

2.2. Óraszinkronizáció PTP-vel

Jelen bekezdés a [7] forrás alapján készült. A Precision Time Protocol (IEEE 1588) egy óraszinkronizációs protokoll. A specifikáció számos hálózattípuson megvalósítható, azonban a vizsgálataim során kizárólag Etherneten, IPv4/UDP felhasználásával alkalmaztam. Alapesetben a hálózat egy mesterórából és több slave eszközből áll. A mesteróra a pontos időt periodikus (általában másodperces gyakoriságú) multicast üzenetekkel szolgáltatja. A szinkronizáció szempontjából releváns üzenetek a 2.4. ábrán látszódnak.



2.4. ábra. *A PTP szinkronizációs üzenetei [7]*

A működés a következő: Kezdetben a slave lokális órája valamilyen értékkel eltér a master órájától ($\mathbf{\Delta t}$). A szinkronizáció célja, hogy ezt az eltérést nullára redukálja. A master először elküld egy *Sync* üzenetet saját órája szerint t_1 időpontban. Ez a slave saját órája szerint t_2 időpontban érkezik meg. Az üzenet tartalma alapesetben lényegtelen, csupán a küldés és fogadás időpontja számít. A t_1 időpont értékét a master külön, egy késleltetett *Follow-up* üzenetben küldi el. Ennek megérkezését követően a slave (ismét saját órája szerinti) t_3 időpontban elküld egy *Delay-Req* üzenetet, amely a masterhoz (master óra szerint) t_4 időpontban érkezik meg. A t_4 értékét egy *Delay-Resp* üzenetben küldi el a slave felé.

A t_1, t_2, t_3, t_4 értékek ismeretéből a slave ki tudja számítani a saját órájának eltérését. Ideális esetben a hálózat δ késleltetése szimmetrikus. Így t_2 felírható a következő módon:

$$\mathbf{t_2} = \mathbf{t_1} + \mathbf{\Delta t} + \delta \tag{2.1}$$

Azaz a fogadás időpontja nem más, mint a küldés időpontja plusz a hálózati késleltetés plusz a slave óra eltérése a mesterórától ($\mathbf{\Delta t}$).

Amikor a slave küld *Delay-Req* üzenetet, az a masterhez a feltételezés szerint ugyancsak δ késleltetéssel érkezik meg. Azonban a slave óra késése itt ellentétes előjellel jelenik meg. Ezek alapján t_4 időpontot a következő módon fejezhetjük ki:

$$\mathbf{t_4} = \mathbf{t_3} - \mathbf{\Delta t} + \delta \tag{2.2}$$

Ahol t_3 – Δt jelenti a *Delay-Req* üzenet küldésének időpontját a master órája szerint.

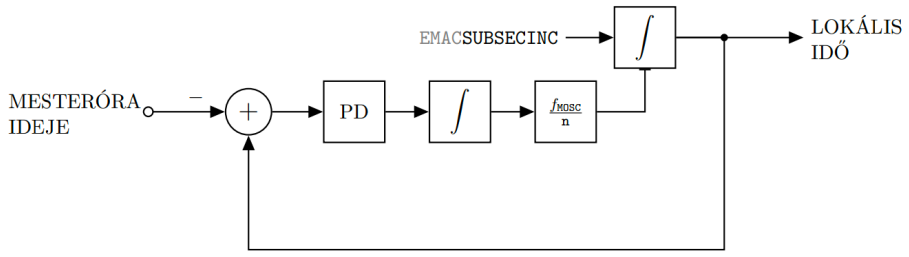
A fenti két egyenlet rendezésével az eltérés a következőképp áll elő:

$$\mathbf{\Delta t} = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{t_2} - \mathbf{t_1} - \mathbf{t_4} + \mathbf{t_3}) \tag{2.3}$$

2.3. Az óraszinkronizációs szoftver működése

A tanszéken elkészítésre került egy óraszinkronizációs algoritmus, mely a PTP protokollt kihasználva beágyazott rendszerek lokális óráját képes egy hálózati mesterórához szinkronizálni [7]. Ez a szoftver a *FlexPTP* fantázianévre hallgat.

A [7] forrásban leírtak szerint az óraszinkronizáció egy diszkrét PD szabályzóval történik. A referencia jel a mesteróra ideje, a visszacsatolt jel a lokális óra értéke. Ezen két érték alapján a hibajel (Δt) a fentebb ismertetett egyenletek alapján áll elő, mely a PD szabályzó bemenetéül szolgál. A szabályzó kimenete az *EMACTIMADD* regisztert állítja (3.1.1), melynek feladata a frekvenciaosztás meghatározása (pontosabban ez a lokális időt tároló akkumulátor regiszter léptetési egysége). Azaz ezen regiszter helyes finomhangolásával állítható a kimeneti frekvencia.



2.5. ábra. *A szinkronizációt elvégző szabályzó rendszermodellje [7]*

Mivel a Sync üzenetek másodpercenként érkeznek, így a szabályzó algoritmus is másodpercenként fut le. A hiba tovább csökkenthető, ha a hőmérő értéke alapján másodpercen belül többször hangoljuk a frekvenciaosztásért felelős regisztert. A feladatot ellátó algoritmus kiválasztása külön vizsgálatot igényel.

2.4. Szabályzó algoritmusok

Számos irányba el lehet indulni a szabályozás terén, rengeteg eljárás született, melyeknek egyedül a processzor számítási kapacitása szab határt.

2.4.1. Statikus kompenzáció

A legegyszerűbb eljárás, ha egy, a memóriában tárolt táblázat alapján megfeleltetjük egymásnak a hőmérő jelét és a léptetési egységet. Ezzel lényegében egy DTCXO működését valósítjuk meg. Az eljárás azért releváns, mert egyszerű, és nagyban növelhető vele a pontosság. A minimális hiba a hardver eszközök pontosságán (pl. hőmérő hibája),

illetve a táblázat finomhangolásán múlik. A vizsgálatokhoz hasznos lehet referenciaként. Ennél eggyel kifinomultabb megoldás, ha a memóriaigényes táblázat eltárolása helyett egy függvénnyel írjuk le az összefüggést. A 2.1. ábra alapján egy harmadfokú függvény jó kiindulás lehet. Ezt persze tovább bonyolítja, ha a hőmérő nemlineáris.

2.4.2. Dinamikus kompenzáció

A statikus modell állandó, vagy közel állandó hőmérsékleten képes jól működni. Amennyiben a hőmérsékleti tranziens gyors, a modell elbukik. Ennek elkerülésére ki kell bővíteni a modellt olyan dinamikus részekkel, melyek figyelembe veszik a változás gyorsaságát is. Ennek fizikai hátterében az áll, hogy a hőmérséklet terjedésése nem pillanatszerű. A statikus kompenzációnál ezt a terjedési időt elhanyagoljuk. A rendszerben lévő termikus jelenségeket egy dinamikus modellell írhatjuk le, melyet meghatároz az egyes elemek (oszcillátor, nyáklap, hőmérő) hőkapacitása, egymástól vett távolsága, hőszigetelő tulajdonsága stb. A termikus rendszerek jól leírhatók a villamos hálózatok tárgyalásánál használt eszközökkel, hiszen:

- a feszültség megfelel a hőmérsékleti gerjesztésnek,
- a kapacitás a hőkapacitást jelöli,
- az ellenállás a hőszigetelést/hővezetési tulajdonságot jelenti,
- az áram pedig a hőáramlást/hőenergia terjedését szemlélteti,
- valamint termikus rendszerben induktivitás nincs

A termikus tulajdonságok egy lineáris dinamikus modellel leírhatók. Ezt mi nem tesszük meg, mert az ismeretlenek száma nagyon nagy. Nem ismerjük egyik elem hőkapacitását sem, illetve a hőszigetelő tulajdonságukat sem tudjuk hosszas mérések nélkül számszerűen szemléltetni. Élhetünk azonban közelítő becslésekkel. A későbbiekben bemutatott modell dinamikus részét a modellalkotó szoftver határozza meg. Mi egyedül az átvitel fokszámát tudjuk változtatni, azonban az optimális együtthatók megkeresése automatikusan történik az identifikációra használt mérési adatok segítségével.

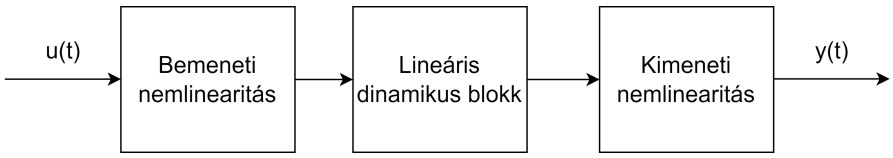
Fontos továbbá tárgyalni a hőterjedés irányáról is. Mivel 1 db hőmérőnk van, előfordulnak olyan esetek, amikor a gerjesztést a hőmérő hamarabb észleli, de olyanok is, amikor az oszcillátorhoz képest csak késleltetve érzékeli azt. Ez nagyban javítható akkor, ha több hőmérőt alkalmazunk. A rendszermodell persze bonyolultabbá válik ezzel. A vizsgálataim során 1 db hőmérőt használtam, azonban a mérések során mindig egy irányból adtam a rendszernek a hőmérsékleti gerjesztést. Ez nem optimális, azonban az itt tárgyalt eljárást már így is jól lehet szemléltetni.

Hammerstein-Wiener modell

Ezen alfejetezet elkészítéséhez a [8] forrásban írtakat használtam fel.

A nemlineáris dinamikus rendszerek egyik legáltalánosabb modellezésére az úgynevezett blokk-orientált megközelítést alkalmazzák. A modell tartalmaz lineáris

dinamikus blokkokat, és memóriamentes nemlineáris blokkokat (statikus nemlinearitás). A két legegyszerűbb eset, amikor a lineáris blokk elé (Hammerstein-modell) vagy mögé (Wiener-modell) kerül egy nemlineáris blokk. Abban az esetben, ha a lineáris dinamikus rendszer előtt és után is helyet kap egy-egy statikus nemlinearitás, Hammerstein-Wiener modellről beszélünk.



2.6. ábra. *A Hammerstein-Wiener modell blokkdiagramja [8]*

A modell képes MIMO[6] rendszereket is leírni, azonban a konkrét alkalmazásban SISO[7] rendszer leírására kell felhasználni, ahol a bemenet a hőmérséklet, a kimenet pedig a frekvencia. Gyakorlati alkalmazásban egy nyákon akár 3-4 hőmérőt is használhatunk, ekkor MISO rendszerről beszélünk. Az előzetes feltételezések alapján a Hammerstein-Wiener modell képes leírni az oszcillátor termikus-elektromos modelljét, mert a rendszer főbb jellegzetességeit magában foglalja. A lineáris blokk modellezi az oszcillátor hőmérséklete és a hőmérő által mért hőmérséklet közötti termikus késleltetést. A kimeneti nemlinearitás teremt kapcsolatot a hőmérséklet és a frekvencia között. Ez hivatott modellezni a rezgőkvarc f – T karakterisztikáját (melyet a várakozások szerint egy harmadfokú függvény ír le). A bemeneti nemlinearitásnak abban az esetben van szerepe, ha a hőmérő pontatlan, és a mért és a valós hőmérséklet között statikus nemlinearitás figyelhető meg.

A BSc szakdolgozatom során a mérési adatokból offline módon megbecsültem a modell paramétereit, mely ígéretesnek bizonyult a feladatra. Az ezt követő lépés az, hogy az illesztett modellt implementáljuk a beágyazott célhardveren, és leteszteljük annak helyességét.

2.4.3. Egyéb lehetőségek

Van, hogy az illesztett modell paramétereit működés közben, on-line kell hangolni. Ez a következő lépés - az adaptív viselkedés megvalósítása. Ez történhet közvetlenül az eszközön, azonban akkor elég erős hardverre van szükség ezen plusz feladat futtatásához. A másik lehetőség, hogy a begyűjtött adatokat elküldi a felhőbe egy nagy számítási kapacitással rendelkező eszköznek, és az már csak a kiszámított korrigált modellparamétereket küldi vissza. Az eljárásnak persze lesz valamekkora késleltetése, de a rendszer karakterisztikája jellemzően jóval lassabban változik ennél.

A fentebb bemutatott hálózati óraszinkronizáció egy PD szabályzót használ, ahol a referencia jel a mesteróra által PTP-vel küldött érték. A hőmérsékleti kompenzáció esetén sajnos nincs lehetőségünk egy ilyen szabályzó működtetésére, hiszen az az alapfeltevés, hogy nincs hálózati kapcsolat, így nincs referencia idő sem.

[6] Multiple Input Multiple Output
[7] Single Input Single Output

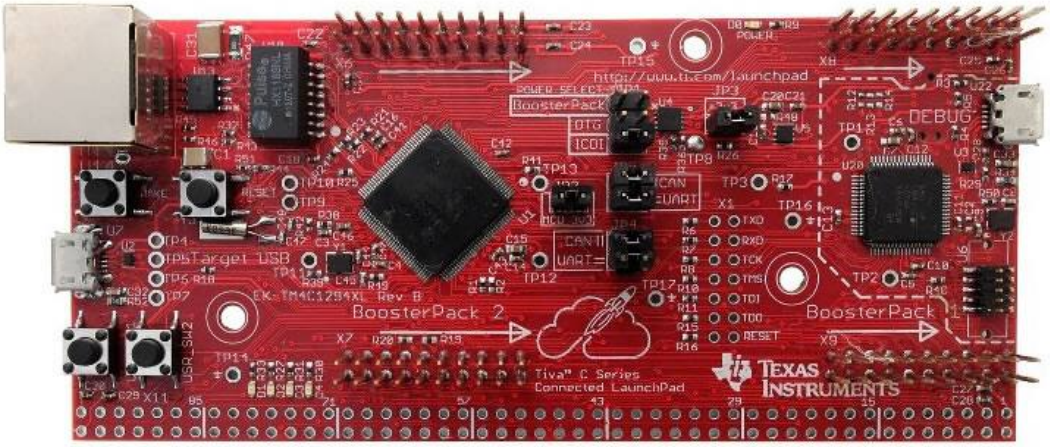
További érdekes lehetőség a neurális hálók alkalmazása, azonban ez mélyebb irodalomkutatást igényel, illetve magas erőforrás igénye miatt nem garantált a működése beágyazott eszközökön.

3. fejezet

A rendszer megvalósítása

3.1. A megfelelő mikrokontroller kiválasztása

A feladat számos ma kapható, megfizethető árú mikrokontrolleren megvalósítható. Fontos szempont az Ethernet csatlakozás, hiszen ezen keresztül működik a PTP óraszinkronizációs protokoll, illetve a PTP hardveres támogatása is. Másik fontos szempont az elérhetőség. A tanszéken 3 alkalmas mikrokontroller típus is volt a munkám kezdetén: egy BeagleBone, illetve két kártya a Texas Instrumentstől: egy TM4C1294XL (Connected LaunchPad) [9], valamint egy AM2434 [10].



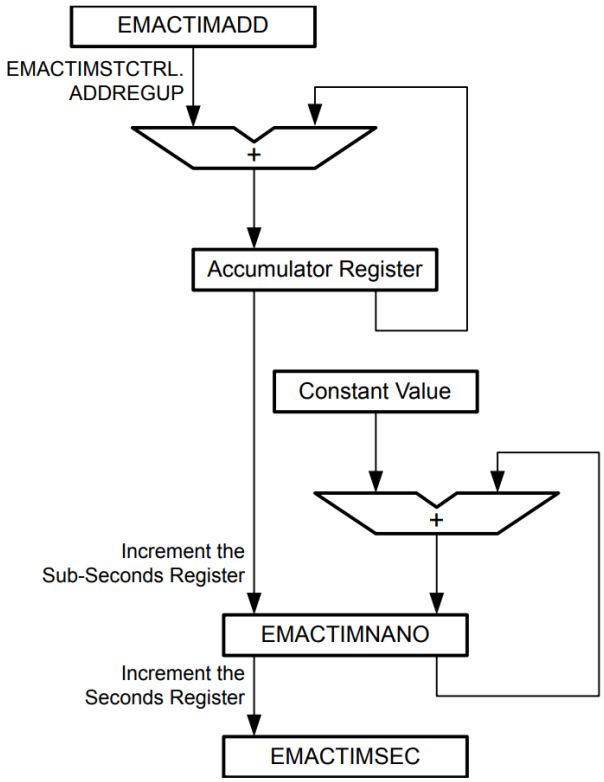
3.1. ábra. *Texas Instruments Connected LaunchPad*

A választást nagyban megkönnyítette, hogy a FlexPTP algoritmus egy Connected LaunchPad mikrokontrolleren lett implementálva, így erre esett a választásom, mivel ekkor megspórolható az idő, amit a már meglévő program új hardverre történő átültetésével töltenék.

Ez esetben a feladatom a meglévő kód funkcionalitásának kiterjesztése egy FreeRTOS taszkkal, illetve a megfelelő függvényekkel.

3.1.1. A hardveróra működése

Jelen alfejezet a [7] forrás alapján készült.



3.2. ábra. *A hardveróra blokkdiagramja [9]*

A TM4C1294XL mikrovezérlő hardverórájának blokkdiagramja a 3.2. ábrán látható. Az óra két részből áll: egy frekvenciaosztásért felelős részből, és egy számlálóból, mely az időt tárolja.

A frekvenciaosztó egy visszacsatolt összeadó, mely értéke minden egyes *Main Oscillator Ciklusban (MOSC)* növekszik egy előre meghatározott léptetési egységgel, melynek értékét az *EMACTIMADD* regiszterben tárolunk. Amennyiben az akkumulátor regiszter túlcsordul, a kimeneti órajelet léptetjük. Azaz az *EMACTIMADD* regiszter értékét változtatva a frekvenciaosztást tudjuk állítani, így a következő képlethez jutunk [7]:

$$\mathbf{f_{carry}} = \mathbf{f_{MOSC}} \cdot \frac{\mathbf{EMACTIMADD}}{\mathbf{2^{32}}} \tag{3.1}$$

Ahol f_{MOSC} a *Main Oscillator* frekvenciája.

A számláló (*EMACTISEC* és EMACTINANO) regiszterek az időt másodperc-nanoszekundum formátumban tárolják, melyek megfelelő léptetését a frekvenciaosztóból érkező jel végzi. A léptetési egységgel tudjuk megadni az óra számára, hogy milyen időközönként érkeznek a léptetési jelek (azaz a tick – nanoszekundum közötti konstans értékét adjuk itt meg). Amikor az *EMACTINANO* regiszter értéke eléri a 10^9

értéket (*0x3B9AC9FF + 1*), akkor a számláló átfordul, és ezzel együtt lépteti az *EMACTISEC* regiszter értékét is.

A FlexPTP működése során a frekvenciaosztás mértékét állítva szabályoz a mesteróra alapján. Ugyanezen regiszter értékét a hőmérsékletmérés alapján is lehetne szabályozni. Ehhez szükséges egy megfelelő pontosságú hőmérsékletmérés.

A fő oszcillátor frekvenciája (f_{MOSC}) 25 MHz, így alapesetben az *EMACTINANO* regiszter léptetési egysége 40 ns. A későbbi mérésekben azonban azt tapasztaljuk, hogy a szinkronizációs időhiba (2.3) ennél kisebb is lehet. Ennek az az oka, hogy a mesteróra finomabb felosztású. Ez azt is jelenti, hogy az idő 40 ns-os felosztása miatt egy állandó kvantálási zaj kerül a PD szabályzó hibajelére. Emiatt a szabályzó nem képes nulla hibával tartani a referencia jelet, hanem mindig ebben a kis környezetben *kering* körülötte, így a mesterórához képest hol sietni, hol késni fog. Ez különösen holdover üzemmód esetén érdekes, hiszen a kapcsolat megszakadásának pillanatában lévő eltérés megmarad, és ez adja a drift kezdeti meredekségét.

3.2. Hőmérsékletmérés implementációja

A feladat nehézsége abból adódik, hogy a mérendő objektum kis hőkapacitású. Ennek a hőmérsékletét kell gyorsan és pontosan megmérni, lehetőleg minél olcsóbb hardver elemek felhasználásával. A megfelelő hőmérő kiválasztása tehát nem triviális. A piacon kapható leggyakoribb típusok a következők:

- Termisztor (NTC[1] és PTC[2])
- IC[3] hőmérő
- Platinaszálas ellenálláshőmérő
- Infravörös hőmérséklet szenzor
- Seebeck-hőelem

A legelső szempont a kis hőkapacitás, hiszen egy robosztus hőmérő érzékenysége túl alacsony volna az oszcillátor hőmérsékleteinek kis változásaira. Ennek a kritériumnak a termisztorok és a platinaszálas ellenálláshőmérők felelnek meg. Az IC hőmérők méretükből adódóan nem alkalmasak a feladatra [11], a Seebeck-elemek használata körülményes. Az infravörös hőmérők esetén pedig nem szempont a hőkapacitás, hiszen a mérés érintésmentes, ami ez esetben igen hasznos (természetesen ezek működése sem teljesen ideális). A platinaszálas, illetve az infravörös hőmérő azonban meglehetősen drágák, így egy valódi alkalmazásban nem célszerű a használatuk. A termisztorok költsége ugyan alacsony, azonban karakterisztikájuk nemlineáris, illetve általában pontatlanabbak a másik két megoldásnál.

[1] Negative Temperature Coefficient
[2] Positive Temperature Coefficient
[3] Integrated Circuit

A nemlinearitás nem jelent akadályt, amennyiben az ellenállás-hőmérséklet karakterisztika egyértelmű. A pontatlanság viszont további vizsgálatokat igényel. Az eszköz gyártási szórása szintén nem probléma, hiszen a végső alkalmazásban a szoftver az adott hardver karakterisztikáját tanulja meg.

A mérési pontatlanság két összetevőből áll: offsetből és pillanatnyi zavarból. Az offset egy konstans érték, melyet a szoftver kikompenzál ugyanúgy, mint a gyártási szórást. A pillanatnyi zavart tekinthetjük fehér zajnak, melyet túlmintavételezéssel, illetve átlagolással lehet kompenzálni. Ezen eljárás hatékonyságát azonban mérésekkel kell igazolni.

A termisztor típusának kiválasztásakor az árat minimalizálni, a pontosságot pedig maximalizálni igyekeztem a kis hőkapacitás szem előtt tartásával. Az SMD[4] hőmérők (kis méretüknél fogva) egyértelműen előnyösebbek a THD[5] eszközöknél.

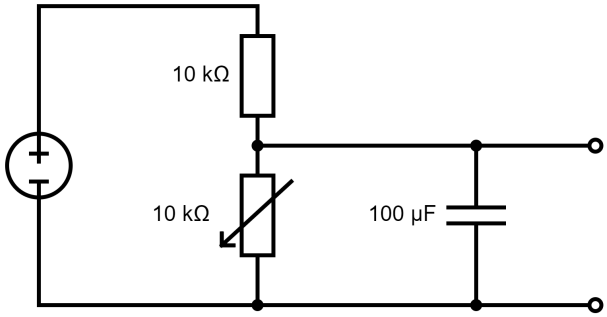
A választott termisztor: *B57232V5103F360* [12].

Az eszköz előnye, és egyben hátránya is a kis mérete. Legnagyobb kiterjedése mindössze 1 mm, ezáltal igen gyorsan felveszi a környezetének hőmérsékletét, azonban a kézi beforrasztása körülményes. A forrasztás során figyeltem arra is, hogy a használt vezetékek lehetőség szerint minél vékonyabbak legyenek, hiszen nem elhanyagolható a hőelvezetési tulajdonságuk.

3.2.1. Mérőáramkör

A mérés elve, hogy a termisztor ellenállása a hőmérséklet függvényében változik egy adott karakterisztika alapján. Az összefüggést az eszköz adatlapja tartalmazza, melyet a programon belül el lehet tárolni. Ellenállásmérésre számtalan lehetőség létezik. Az esetek döntő többségében az ellenállásmérést feszültségmérésre vezetik vissza. A két érték közötti összefüggés a mérőáramkör kialakításának sajátja. Ez a végtelenségig bonyolítható, azonban az alkalmazás költségérzékeny, így célszerű az egyszerű megvalósíthatóság és az elégséges pontosság közötti egyensúlyt megtalálni. A választásom éppen ezért egy feszültségosztóra esett. Kétségbevonhatatlanul ez az egyik legegyszerűbb áramköri elrendezés, a termisztoron kívül csak egy 10 kΩ értékű ellenállást, és egy 100 μF értékű zajszűrő kondenzátort tartalmaz. Hátránya, hogy a karakterisztika nemlineáris. Azonban az összefüggés továbbra is egyértelmű, így a hozzárendelést a program képes elkészíteni.

[4]Surface Mounted Device
[5]Through-Hole Device



3.3. ábra. *Feszültségosztó*

A mikrokontrollerben rendelkezésre áll egy 12 bites ADC[6]. Ezen periféria működési elve, hogy összehasonlítja a bemenő feszültséget a 3.3 V értékű referenciafeszültséggel, és a kettő hányadosát megszorozza a 12 biten kiadható maximális értékkel (4096-tal).

$$OUT = \frac{V_{in}}{V_{DC}} 2^{12} \tag{3.2}$$

Ha tehát a komparátor bemenetén 1.65 V van, akkor a digitalizált érték 2048 lesz. Ezt az értéket szoftverben már egyszerűen átskálázhatjuk, hogy Celsius-fok értéket kapjunk.

A mérés során tehát a következő konverziók történnek: oszcillátor hőmérséklet (analóg) → termisztor hőmérséklet → termisztor ellenállás → termisztor feszültség → ADC kimenet (digitális) → hőmérséklet (digitális). Minden egyes lépésnek lehet hibája, mely a pontosság romlásával járhat. Ezeket egyesével megvizsgálom.

- Oszcillátor hőmérséklet → termisztor hőmérséklet: Itt a legfőbb hibát a lassú hőterjedés, illetve a termisztor hőkapacitása okozhatja. A két eszközt hővezető ragasztóval rögzítettem egymáshoz, valamint a lehető legkisebb méretű ellenállást választottam ki, így jelen körülmények között ez a lépés már tovább nem javítható. (Két termisztor használata esetén, azok ügyes elhelyezésével, és értékeik átlagolásával elképzelhető a mérés pontosítása.)

- Termisztor hőmérséklet → termisztor ellenállás: Ez a karakterisztika az eszköz sajátja, az adatlapon táblázatos formában ismertetett. Gyártási szórással számolhatunk, azonban ez kiküszöbölhető. Mivel a feladat költségérzékeny, ezen a ponton sem lehet lényegesen javítani.

- Termisztor ellenállás → termisztor feszültség: Ideális esetben ez a megfeleltetés kölcsönösen egyértelmű. A hálózatban lévő induktív és kapacitív parazita elemek, illetve a pillanatnyi zavarok azonban torzíthatnak az eredményen. Mivel azonban a mérés nem tartalmaz nagyfrekvenciás elemeket, így az ezen a ponton behozott hibát elhanyagolhatónak tekintem a többi lépéshez képest.

[6] Analog Digital Converter

- Termisztor feszültség → digitalizált ADC kimenet: A mikrokontroller egy 12 bites beépített AD átalakítót tartalmaz. A pillanatnyi zavarok (például a referenciafeszültség ingadozása) nagyban befolyásolják a mérés eredményét. Az átalakítót lecserélhetnénk egy nagyobb bitszámú, pontosabb eszközre, illetve a referencia feszültség is tovább pontosítható kiegészítő áramkörök hozzáadásával, azonban a plusz költségek miatt ez kerülendő. A pillanatnyi zavarok ellen a feszültségminták átlagolásával is védekezhetünk. Ehhez (a hőmérsékletváltozás időállandójához képest) magas mintavételi frekvencia, és a zaj karakterisztikájának ismerete szükséges.

- ADC kimenet → hőmérséklet: Két digitális érték közötti konverzió nem hoz be további adatvesztést, amennyiben a szoftver jól meg van írva. A konverzió végbe mehet egy közelítő függvény alapján, vagy táblázatok megfeleltetésével is. Természetesen nem érdemes letárolni minden egyes bináris értékre az ahhoz tartozó hőmérsékletet. A programot nem érdemes már tovább bonyolítani, amennyiben a konverzió által behozott adatvesztés elhanyagolható a mérés többi pontjához képest.

Az ellenállás osztót egy nyákon valósítottam meg, mely biztosítja az összeköttetést a termisztor és a Connected LaunchPad között. Az ADC beolvasás szoftveres implementációjához a Tivaware Peripherial Driver Library függvénykönyvtárat [13] használtam fel. Az API[7]-hoz található példakódok alapján megírtam az ADC inicializációját és a beolvasást elvégző függvényeket. Fontos kiemelni, hogy a könyvtár függvényei lehetőséget nyújtanak hardver és szoftver túlmintavételezésre egyaránt.

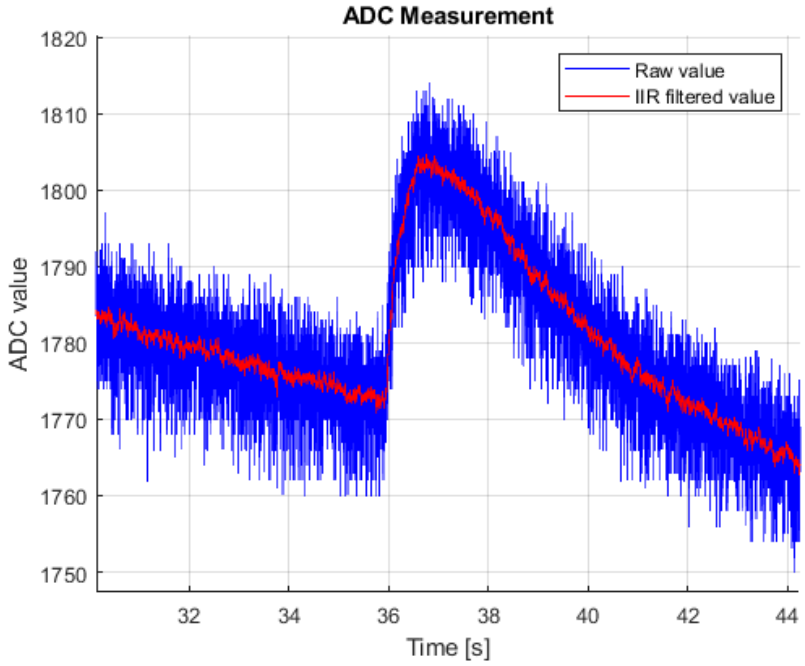
A hardveres túlmintavételezés (HW OS[8]) úgy zajlik, hogy több beolvasást végez el a periféria, melyek kiátlagolt eredményét írja bele a tároló regiszterbe. Maximálisan 64-szeres túlmintavételezést állíthatunk be ilyen úton, de ennek az az ára, hogy ennyiszer kevesebb adatot állít elő, így lassul a működése. Ez azonban még mindig bőven elégséges az adott alkalmazáshoz.

A szoftveres túlmintavételezés (SW OS[9]) lényege, hogy az eszköz az egy regisztertömbben lévő beolvasott adatokat átlagolja ki. Egy tömb maximum 8 regiszterből áll, így a szoftveres túlmintavételezésnek ez a felső határa. Lényeges különbség azonban, hogy a tömbben lévő minták érkezhetnek különböző ADC lábról is, így akár két, vagy több hőmérő átlagát is lehet képezni. Ez a jelen alkalmazásban nem releváns, azonban a jövőbeli továbbfejlesztéseknél fontos szempont lehet.

Maximálisan tehát 512-szeres túlmintavétezésre van lehetőség az API lehetőségeit teljesen kiaknázva. Ezen felül természetesen további jelfeldolgozás végezhető az adatsoron.

A 3.4. és 3.5. ábrákon a túlmintavételezés hatását figyelhetjük meg. A mérések 1 kHz frekvenciával történtek. Az y tengelyt megfigyelve jól látható, hogy 64-szeres HW OS

[7]Application Programming Interface
[8]Hardware Oversampling
[9]Software Oversampling

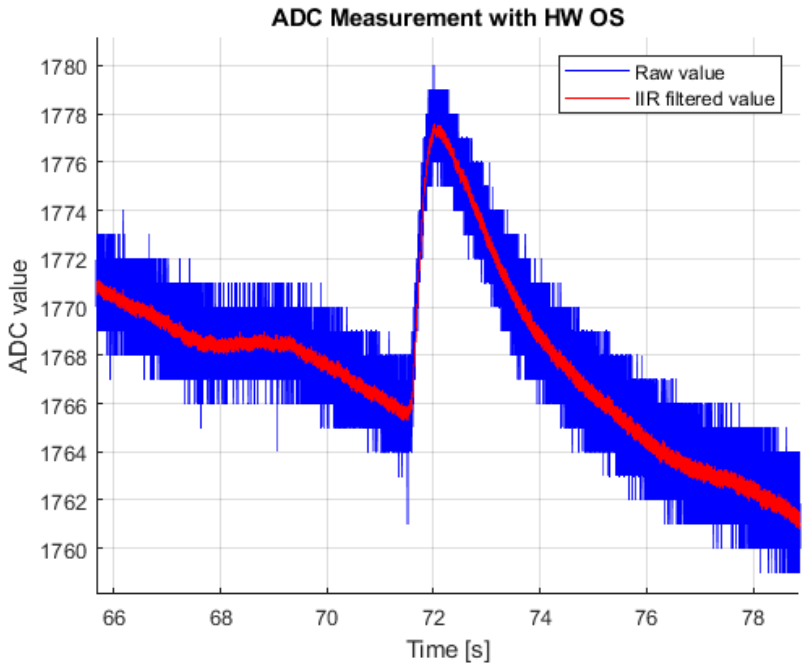


3.4. ábra. *ADC mérés túlmintavételezés nélkül*

esetén lényegesen lecsökken a jelre ráülő zaj nagysága. A 3.4. ábrán a zaj nagyjából ±15 egység értékű, míg a 3.5. ábrán ez az érték ±3 egység. Ebből is látszik, hogy nem érdemes tovább növelni a túlmintavételezés értékét, hiszen a zaj már így is a kvantálási hiba nagyságrendjébe esik.

Látható továbbá az ábrákon egy-egy piros görbe is, melyeket úgy kaptam, hogy a bejövő adatokat egy IIR[10] szűrővel ($\alpha = 0.95$) feldolgoztam. Ennek hatására a zaj további egy nagyságrenddel csökkent. A szűrő α paraméterének kiválasztásakor a zajelnyomás és a késleltetés között kell megtalálni az egyensúlyt. Ahogy az α értéke közelít 1-hez, úgy mind a zajelnyomás, mind a késleltetés növekszik. Az ábrákon látható tranzienst jól leköveti a piros görbe, nem tapasztalható lényeges késleltetés, tehát $\alpha = 0.95$ paraméter jónak bizonyult.

[10]Infinite Impulse Response



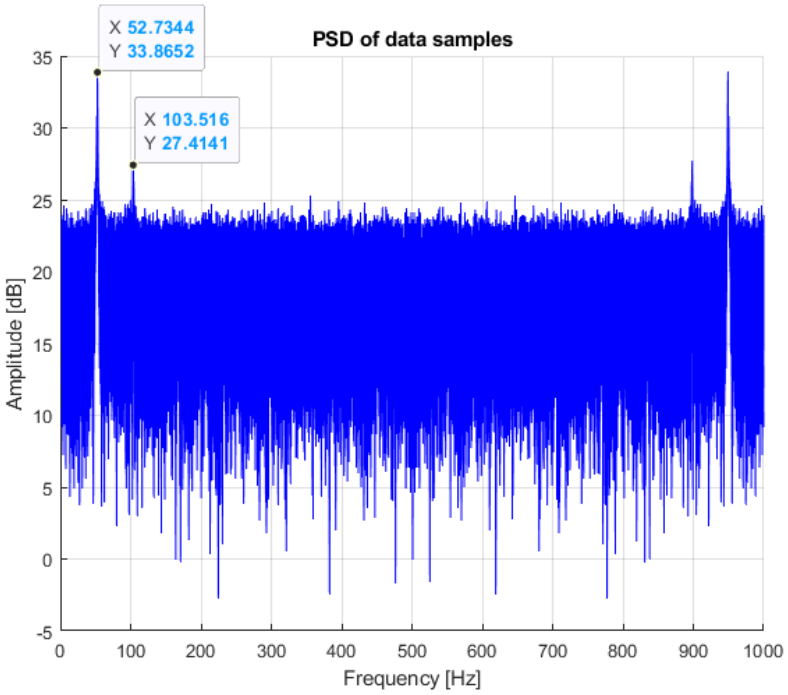
3.5. ábra. *ADC mérés 64-szeres hardveres túlmintavételezéssel*

3.2.2. Zavarvizsgálat

Méréseim során külön figyelmet szenteltem a pillanatnyi zavar analízésének. Zaj sajnos minden alkalmazásban előfordul, nem tekinthetünk el ettől. Feldolgozásukat azonban könnyíti, ha jellegüket tekintve szélessávú fehér zajnak lehet őket tekinteni. Mivel vélhetőleg a legfőbb zavarforrás a DC referenciafeszültség, mely jó közelítéssel nem determinisztikus, így jó hipotézisnek tűnik a fehér zaj feltételezése. Ezt azonban mérésekkel is ellenőriztem.

Felvettem egy kellően hosszú mintasorozatot az ADC beolvasott értékeiről 1 kHz mintavételi frekvenciával. Mérés közben a hőmérséklet állandó volt, így a hasznos jel tekinthető konstansnak. Erre ül rá a vizsgálandó zaj. A mintasorozatból kivágtam 100 db nem átlapolódó, egyenként 1024 elemből álló mintát. Ezeknek egyesével elkészítettem a spektrumát 1024 pontos FFT[11] algoritmussal. A kapott eredményeket egymásra rajzolva ábrázoltam decibel-skálán. Így adódott a 3.6. ábra. A bejövő adatokat nem normáltam le, így a zajszint nem 0 dB-nél van, azonban ez nem probléma, mivel jelen vizsgálatnál csak a relatív értékek érdekelnek.

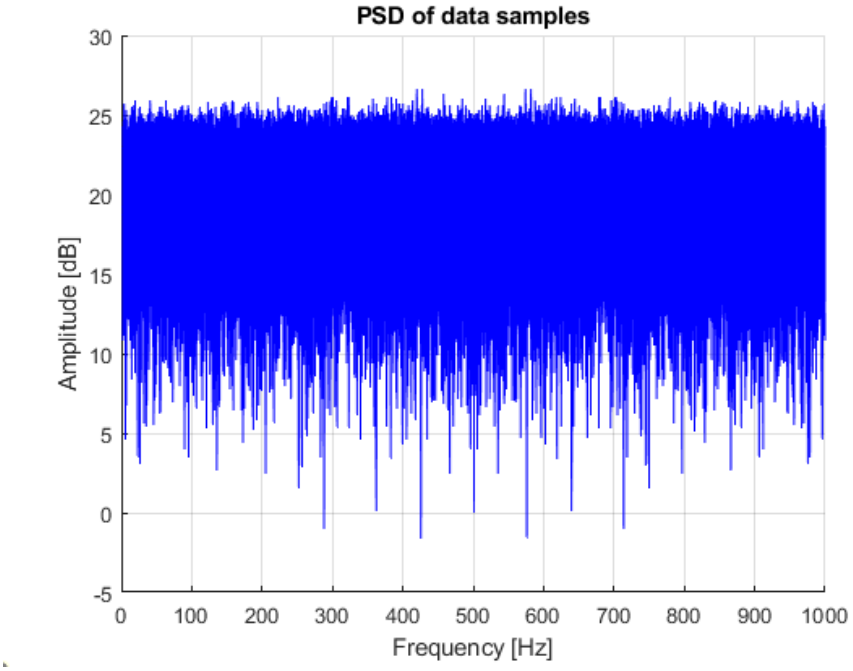
[11]Fast Fourier Transform



3.6. ábra. *A zavarjel spektruma kondenzátor nélkül*

Jól látszik, hogy nem fehér zajról van szó, hiszen közel 10 dB-lel kiemelkedik a zajszintből egy tüske 50 Hz-nél, illetve ennek felharmonikusa 100 Hz környékén. Ez azt jelenti, hogy a hálózati frekvencia valóban belekerül a mért jelbe. A kapcsolóüzemű tápegységek kimenetén leggyakrabban a hálózati frekvencián, illetve a kapcsolási frekvencián jelennek meg tüskék, illetve ezek felharmonikusain. A kapcsolási frekvencia egy nagyságrenddel nagyobb, mint a mintavételi frekvencia, így ezt nem tudjuk észlelni.

A mérést az ellenállás osztó áramköri rajzán található zavarszűrő kondenzátor nélkül végeztem. Ezt követően került a helyére a 100 µF értékű kapacitás, mellyel megismételtem a mérést. Minden egyéb paraméter és a mérési eljárás is azonos az előzőével. Az eredmény a 3.7. ábrán látszik.



3.7. ábra. *A zavarjel spektruma a kondenzátor beiktatása után*

Jól látszik, hogy eltűntek a hálózati frekvenciás komponensek, a zaj spektruma konstans. Ez kifejezetten előnyös a jelfeldolgozás során.

Készítettem két további mérést egyéb hatások vizsgálatára is. Ezekben az esetekben a kondenzátor nem volt az áramkörben.

- Először azt próbáltam ki, hogy van-e hatása, ha a laptop töltőkábelét kihúzom. A mikrokontroller ugyanis a laptop USB portjáról kapja az 5V DC tápfeszültségét, úgyhogy közvetve kapcsolatban áll a hálózattal. Ha viszont kihúzzuk a kábelt, a mikrokontroller a laptop akkumlátoráról kapja a tápfeszültséget.
- Ezt követően megnéztem a 64-szeres hardveres túlmintavételezéssel készített mintasor spektrális képét

A próbálkozások nem értek el sikert, a kondenzátor nélkül a tüskék benne maradtak a spektrumban. Ez várható volt, hiszen a zavar nem csak a galvanikus összeköttetésen keresztül képes az AD átalakítóig eljutni, hiszen a labort minden irányban átszövik a hálózati kábelek, így ezek induktív és kapacitív módon hatást gyakorolnak minden közelben lévő vezetőre. A túlmintavételezés szintén nem segít, hiszen 64-szer annyi mintában is ugyanúgy megtalálható a zaj. Az ezekről a mérésekről készült ábrák megtalálhatók a függelékben (F.1.1. és F.1.2.).

3.3. Kontrollmérés infravörös hőmérővel

A szóba jövő hőmérők tárgyalásakor megállapítottam, hogy az infravörös hőmérő előnyös, hiszen nem releváns a hőkapacitása, lévén hogy más hőmérsékletmérési eljárást alkalmaz. Magas ára miatt azonban egy célalkalmazásban nem elképzelhető, viszont a kutatás során hasznos lehet a termisztor eredményével való összevetésre. Ezen okokból kifolyólag a mérőáramkörömön egy ilyen típusú eszköz is helyet kapott.

A feladatra a Melexis *MLX90614* infravörös hőmérőjéhez készült *Gy906* breakout board modult használtam [14].

Az eszköz I2C kompatibilis, az adatok digitális formában kiolvashatók belőle. A Connected LaunchPad szabad lábain több I2C kivezetés is található, illetve a TivaWare library tartalmaz I2C függvénykönyvtárat is a mikrokontrollerhez. Az illesztés ezek ellenére azonban korántsem volt magától értetődő. Az API hiányos dokumentációja miatt sajnos számos probléma felmerült, melyek kijavítása sok időt felemésztett, sokszor próbálgatást igényelt.

Az interneten számos példakód fellelhető a hőmérő Arduinohoz való illesztéséhez [15]. Ez jó kiindulás a kommunikáció megértéséhez, azonban a kódot a saját mikrokontrolleremre is meg kellett valósítanom. Ehhez hardverspecifikus példákat kerestem, azonban a TivaWare I2C könyvtárhoz található egyszerű példakódok többnyire 1 bájt beolvasását, illetve kiírását valósítják meg ([13] és [16]), ellenben az infrahőmérőhöz 3 bájtnyi burst típusú adatbeolvasás szükséges. Ehhez más parancsokat kellett alkalmazni, melyek leírása nem egyértelmű. A példakódok közül több elévült, illetve a TI[12] egy más mikrokontrolleréhez készült, így számos rejtett hiba is előjött a fejlesztés során.

A legfőbb hiba az időzítésben rejlett. Az API biztosít egy *I2CMasterBusy()* nevű függvényt, amely azt jelzi, hogy a master egység elfoglalt-e vagy sem. Két adatbeolvasás között azonban további késleltetéseket kellett közbeiktatni, különben érvénytelen adatot olvas be a program. Az API függvény tehát nem biztosította a megfelelő funkcionalitást, melyre csak próbálgatások útján sikerült rájönnöm.

A beolvasás megvalósítását követően elhelyeztem az eszközt a kiegészítő nyáklapomon, melyen az ellenállás osztó is helyet kapott. Úgy forrasztottam be, hogy a Connected LaunchPad kristály oszcillátora felé nézzen, attól nagyjából 0.5 cm távolságban. Az eszköz nagyjából 90°-os szögben lát, ebből a tartományból gyűjt be adatot. Ebből adódóan nem csak a kristálykvarc hőmérsékletét, hanem a környező áramkörökét is méri. Feltételezhetjük azonban, hogy ezek nagyjából azonos hőmérsékleten vannak, hiszen az infrahőmérőnek csak kontroll szerepe van.

Az infravörös hőmérőre is végeztem zavarteszteket, melyek eredményeképp megállapítható, hogy az eszköz hibája nagyjából 0.5 °C. A mérésekről készült ábrákat a tömörség megőrzése végett a függelékben közlöm. (F.2.1. és F.2.2.)

[12] Texas Instruments

3.4. Mérési adatok gyűjtése és feldolgozása

A hőmérők üzembe helyezése után a következő lépés a funkcionalitás beleintegrálása a FlexPTP projektbe. Mivel a rendszer FreeRTOS operációs rendszer alatt fut, célszerű egy külön taszkot létrehozni, mely megméri a hőmérsékletet, kiszámítja a beavatkozó jelet, és beállítja az akkumulátor regiszter léptetési egységét (*EMACTIMADD*). A hőmérséklet és a léptetési egység közötti összefüggés meghatározásához azonban először mérési adatokat kell gyűjteni. Ehhez valamilyen hőmérsékleti gerjesztést kell adni a rendszernek. Az ideális hőprofilról a szakirodalom széleskörű tájékoztatást nyújt [17], továbbá a szakdolgozat során [1] is szereztem tapasztalatot ebben a témában. A statikus karakterisztika felvételéhez a legjobb, ha minél lassabban melegítjük vagy hűtjük a rendszert. Hiszen ha hirtelen melegítjük fel az eszközt, akkor nagy hőmérsékleti gradiensek jelennek meg az oszcillátor és a termisztor között, mely még a legkisebb hőkapacitású hőmérők esetén is jelentős hibát tud okozni. A lassú hőprofilt úgy érhetjük el, hogy a méréseket kellő óvatossággal végezzük, illetve ha elkerülhetetlenek a hirtelen változások, akkor az adatfeldolgozásnál ezeket a részeket kiszűrjük a vizsgált mintákból.

Dinamikus modellillesztésnél azonban fontos, hogy legyenek gyorsabb változások is, hiszen ekkor a rendszer termikus tehetetlenségét is igyekszünk számításba venni. A System Identification Toolbox akkor adja a legjobb eredményt, ha az identifikációs adatsoron többször (8-10-szer legalább) megismétlődnek a hirtelen ugrások. Itt azonban előjön egy újabb hibalehetőség: nem mindegy, hogy a hőmérsékleti gerjesztés milyen irányból jön. Ha az oszcillátor felől érkezik, akkor a termisztor lemarad hozzá képest, ha pedig ellenkező irányból, akkor viszont hamarabb érzékeli az új hőmérsékletet, mint ahogy az kifejtené a hatását. Ezzel sajnos együtt kell élnünk. A probléma úgy volna orvosolható, ha legalább kettő (de akár több) hőmérővel közrefognánk a kvarcot. Ekkor már úgy is pontosabb becslést kapunk, ha a hőmérsékletek átlagát vesszük alapul. A megoldás tovább javítható, ha nem csak az átlagot tartjuk meg, hanem kétbemenetű rendszert identifikálunk (MISO rendszer), vagy a hőmérsékleti gradienst is figyelembe vesszük. Erről az Önálló laboratórium 1 keretei között már végeztem méréseket. A tapasztalatok igazolják az állítást, miszerint 2 hőmérővel pontosabb eredmények érhetők el. Jelen rendszerben ez továbbfejlesztési lehetőségként adott. A 3.2.1-ben bemutatott Software Oversampling erre igen kézenfekvő, hiszen API szintű megoldást nyújt több ADC-ből érkező jelek átlagolására.

A laborban lévő hőlégfúvóval igyekeztem mindezen megfontolások alapján jó mérési adatokat előállítani. A hevítés sebességét úgy tudtam szabályozni, hogy különböző távolságokból fújtam a mikrokontrollert. A lehűlés menete többnyire adott: a nagyjából változatlan szobahőmérsékletre exponenciális jelleggel áll vissza. Hűtésre nem volt módom.

A mérési adatokat úgy állítottam elő, hogy a mikrokontrollert többször egymás után felhevítettem szobahőmérsékletről 50-55 °C-ra, majd megvártam, míg lassan kihűl.

Látható, hogy a módszer korántsem tökéletes. A jó minta kulcsfontosságú a jó modell elkészítésében. Ideális esetben egy hőkamrában, tetszőleges hőprofilok előállításával

végezném el a mérést, a jelenleginél jóval szélesebb tartományon. A bemutatott elv azonban már ezzel a módszerrel is jól reprezentálható. Továbbfejlesztési lehetőségként adott a hőkamrával való mérés. A vizsgálat menete abban az esetben is megegyezik az itt részletezettekkel, így könnyedén átültethető.

Az adatgyűjtéshez létrehoztam egy új taszkot, mely inicializálja mind a termisztort, mind az infravörös hőmérőt, majd pedig ciklikusan beolvassa ezek értékét. Az adatgyűjtés ideje alatt ezt a gyakoriságot 1 milliszekundumnak választottam. (Természetesen egy valós alkalmazásban nem feltétlen szükséges ilyen gyakran beolvasni a termisztor értékét.)

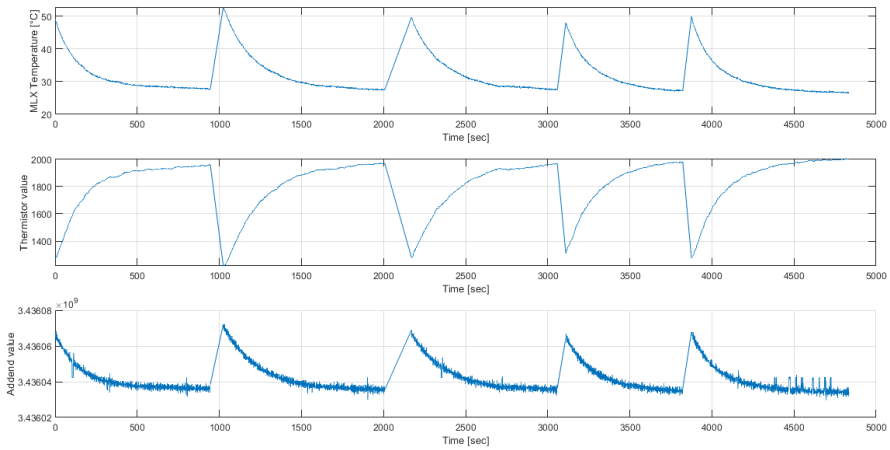
Az adatgyűjtés menete a következő: érintetlenül hagyjuk a FlexPTP által implementált szinkronizációs eljárást, így a korrekció továbbra is lefut minden másodpercben. Amennyiben változik az oszcillátor hőmérséklete, akkor az *EMACTIMADD* regiszter (továbbiakban *addend*) értékét is automatikusan korrigálja a szabályozási kör. Mindeközben folyamatosan figyeljük a termisztor értékét. Ha tehát kívülről megfelelő ütemben változtatjuk a hőmérsékletet, és elég adatot gyűjtünk, minden egyes hőmérséklethez megkapjuk a helyes akkumulátor regiszter értéket. (Ez persze első lépésben egy statikus hozzárendelést ad, elhanyagolva a termodinamikai időállandókat és a hőterjedést, de kezdetnek ez is igen jó.)

A FlexPTP soros porton vagy telneten keresztül küldi a következő adatokat:

- mesteróra ideje ns pontossággal (PTP-ből jövő T1 és T4 értékek),
- lokális óra eltérése (ns-ban és tickben is),
- addend értéke a szabályozás után.

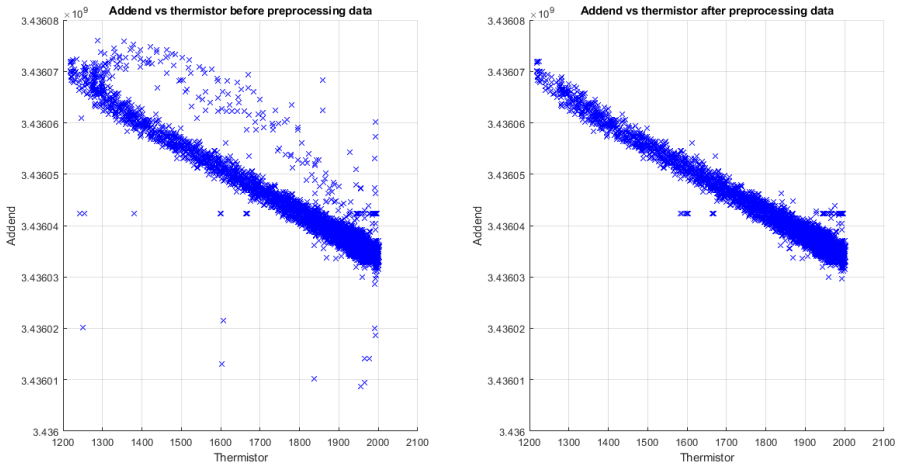
Az adatküldés minden egyes korrekció után (azaz másodperces gyakorisággal) lefut. Ehhez az adatsorhoz elég hozzáfűzni a termisztor és az infravörös hőmérő értékét, és máris rendelkezésre állnak a mérési adatok. Mivel a logolás másodpercenként történik, a hőmérséklet beolvasás pedig milliszekundumonként, ezért fontos meghatározni, hogy pontosan milyen módszer szerint tömörítjük ezred részére az elküldeni kívánt adatot. A 3.2.1-ben ismertetett módszerek alapján egy IIR szűrőt használtam $\alpha = 0.99$ értékkel. Ez már önmagában elég jól kiszűri az esetleges zajokat, így a HW túlmintavételezésre nincsen szükség. A szoftver úgy működik, hogy 1 kHz gyakorisággal olvassa be a termisztor értékét, melyet átlagol az IIR szűrővel. Ezzel párhuzamosan fut a korrekciót végző taszk másodperces gyakorisággal. (Valójában egy végrehajtó egység van, úgyhogy felváltva futnak a taszkok, de az ütemezés már a FreeRTOS dolga.) A korrekció elvégzését követően lekérjük az aktuális átlagolt termisztor értéket, és elküldjük a PC számára.

A mintákat Matlab segítségével vizsgáltam meg. Első megjelenítésre feltűnt, hogy az addend regiszter értékeiben hatalmas, pillanatszerű ugrások jelentek meg. A kiugró értékeket az okozta, hogy bekerült egy hiba a szabályzóba, mely a későbbiekben javítva lett. Ezeket még az ábrázolás előtt kiszűrtem.



3.8. ábra. *A mérési adatok időfüggvényei*

Az így kapott addend értékekben továbbra is látható némi zaj (3.8. ábra, 3. grafikon), azonban ez jóval elhanyagolhatóbb, mint a nyers bejövő adatsoron. Ezt követően azt kell megvizsgálni, hogy az addend és a termisztor értéke hogyan korrelál egymással. Ehhez a bemeneti adatsornak csak azon részeit vettem alapul, ahol a hőmérséklet lassan visszahűl 55 °C-ról szobahőmérsékletre. Erre azért volt szükség, mert a hirtelen melegítés hatására hamis összefüggéseket kapunk a késleltetések miatt.



3.9. ábra. *Az addend és termisztor értékpárok előfeldolgozás előtt és után*

A 3.9. ábrán az addend látható a termisztor értékének függvényében, előfeldolgozás előtt és után. Megfigyelhető, hogy az előfeldolgozás nagyban letisztította a grafikont. A jobb oldali grafikon ugyan még mindig zajos, de már felhasználható a korreláció megállapításához. Fontos megemlíteni, hogy azzal, hogy az adatsor dinamikus, hirtelen változó részeit kiszűrjük, csak a statikus karakterisztikára kaphatunk közelítő összefüggést. Ez kiindulási pontként hasznos, azonban ezt egy TCXO is meg tudja csinálni, így a későbbiekben külön vizsgálat tárgya lesz a dinamikus tartomány is.

Az *addend* beolvasásakor többször is előfordultak egyértelműen fals adatok, melyek több

nagyságrenddel meghaladták a bejövő adatok átlagos szórását. Gyakran 10^9 nagyságrendű ugrások figyelhetők meg 1-1 órajel ciklus erejéig, holott az átlagos szórás 10^4 nagyságrendű. Ennek oka a szabályzó kódjában elejtett hiba volt, mely csak később lett korrigálva. A mérési adatokból ezeket ki kellett szűrni. Erre több módszer is adott:

- Adatok manuális átvizsgálása egyesével: tekintve, hogy néhány adatsor mérete meghaladja a 15000 sort, ez a módszer önmagában kerülendő. Azonban ha a kiugró adatok között mintát vélünk felfedezni, akkor pl. a Notepad++ keresés és behelyettesítés funkciójával (Ctrl + H) nagyban gyorsítható a folyamat.
- Adatfeldolgozás scriptekkel: lényegesen kényelmesebb, és reprodukálhatóbb. Számos szűrőfeltétel megadható, melyekkel gyakran jobb szűrést érhetünk el, mint a manuális módszerrel.
  - Limit értékek megadása: megadunk egy felső és egy alsó korlátot, és azokat az adatokat szűrjük, melyek e tartományon kívül esnek
  - Meredekség vizsgálata: amennyiben az előző értéktől vett eltérés abszolút értéke egy adott határ fölött van, úgy kiszűrjük a mintát. Oda kell figyelni, hogy a szűrőfeltételnek beállított meredekség nem lehet kisebb, mint a hirtelen melegítés hatására felfutó adatsor meredeksége, különben azokat is kiszűrjük.
  - Mozgóátlaggal való összehasonlítás: amennyiben az előző n minta átlagától egy megadott értéknél nagyobb mértékben tér el, úgy kiszűrjük a mintát.

A kiszűrt minták helyére többféle értéket is behelyettesíthetünk:

- az egész mintasor számtani átlagát (nem tanácsos, mivel ekkor minden kiszűrt érték helyére egy konstans szám kerül),
- az előző mintát,
- az előző és a következő minta számtani átlagát (lineáris interpoláció),
- a mozgó átlagot (lehet FIR vagy IIR),
- a környező minták egyéb, tetszés szerinti átlagolásából adódó értéket.

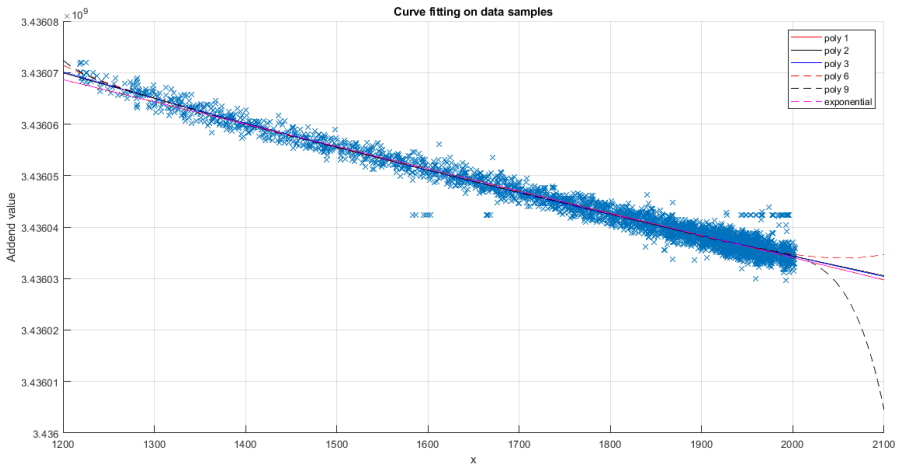
A tapasztalat azt mutatja, hogy az a legjobb eljárás, ha a környező elemek valamilyen módon vett súlyozott átlagával helyettesítjük a hibás mintákat. A szűrőfeltételeknek pedig egy optimális kombinációja, ha először lazább limit értékeket megadva kiszűrjük a nagyon eltérő elemeket, majd ezt követően a mozgóátlagtól való eltérés mértékének függvényében végzünk szűrést.

3.5. Modellillesztés

3.5.1. Statikus nemlinearitás

A kapott adathalmazra illeszteni kell egy modellt, amely jól becsli a termisztor alapján az addend értékét. Első, és legegyszerűbb módszer az, hogy ha elhanyagoljuk a rendszer dinamikáját, és egy statikus összefüggést keresünk a két változó között. Ez lényegében egy függvényillesztési feladat a 3.9. ábra jobb oldali grafikonján látható ponthalmazra. A korábbi vizsgálatokból [1] ismert, hogy a hőmérséklet és a frekvencia között statikus esetben egy harmadfokú függvény írja le a kapcsolatot. Itt azonban a termisztort és az addendet vizsgáljuk. Az addend és a frekvencia között egy lineáris kapcsolat van, így ez nem bonyolítja meg a vizsgálatot, azonban a termisztor karakterisztikája exponenciális jellegű. Természetesen az adatlapban [12] feltüntetett táblázat alapján közelítő becslések végezhetők a pontos függvényre, de ez túlbonyolítaná a modellt. Szem előtt kell tartani azt is, hogy a közelítés egy beágyazott mikrokontrolleren fog lefutni, így lényeges az egyszerűség.

A görbeillesztéshez a *Matlab Curve Fitting Toolbox* függvényeit használtam. A *fit* parancsnak megadjuk a be- és kimeneti adatsort, illetve hogy milyen függvényt szeretnénk illeszteni (pl. polinomiális vagy exponenciális), és visszakapjuk a ponthalmazra legjobban illeszkedő ilyen típusú függvény együtthatóit. A vizsgálataim során kipróbáltam a polinomiális függvényeket (elsőtől kilencedik fokig), illetve az exponenciális függvényt is. A 3.10. ábrán láthatóak a lényegesebb eredmények.



3.10. ábra. *A görbeillesztés eredménye*

Azt, hogy milyen jól illeszkedik egy függvény, a goodnessOfFit panaccsal ellenőrizhetjük le. Ez nem csinál mást, minthogy kiszámolja a ponthalmaz görbére vett négyzetes középhibáját. Ezt minden egyes illesztésre kiszámoltam:

A táblázat alapján elmondható, hogy lényeges különbség nincs az egyes becslések pontossága között. Ennek az az oka, hogy a vizsgált szakasz meglehetősen egyenes, így az

fitType	NRMSE[13]	fitType	NRMSE
poly1	0.1867	poly6	0.1836
poly2	0.1843	poly7	0.1836
poly3	0.1843	poly8	0.1835
poly4	0.1837	poly9	0.1834
poly5	0.1836	exp	0.1867

3.1. táblázat. *Az illesztett görbék illeszkedése számszerűen*

alacsonyfokú polinomok is jól teljesítenek. A 3.10. ábrát vizsgálva feltűnik, hogy a magasabb fokszámú becslések erősen elhajlanak a vizsgált szakasz képzeletbeli meghosszabbításától, úgyhogy ezeket kizártam a vizsgálatból. Mivel a rendszer modellezéséhez minimum harmadfokú görbe kell (2.1.), így itt is a harmadfokú polinomot választottam ki további elemzésre. Az összefüggést leíró képlet numerikus értékekkel a következőre adódott:

$$\mathbf{addend}(\mathbf{x}) = \mathbf{p_1}\cdot\mathbf{x^3} + \mathbf{p_2}\cdot\mathbf{x^2} + \mathbf{p_3}\cdot\mathbf{x} + \mathbf{p_4} \tag{3.3}$$

$$\mathbf{p_1} = \mathbf{3.976}\cdot\mathbf{10^{-5}}$$
$$\mathbf{p_2} = \mathbf{-0.1851}$$
$$\mathbf{p_3} = \mathbf{238.442}$$
$$\mathbf{p_4} = \mathbf{3.4360}\cdot\mathbf{10^{9}}$$

A termisztor x értéke 10^3 nagyságrendű. A fenti képletből adódik tehát, hogy a harmadfokú tag 10^4, a másodfokú 10^5, míg az elsőfokú szintén 10^5 nagyságrendű lesz. Ebből is látszik, hogy a harmadfokú tag kis súllyal szerepel az összefüggésben. Azonban ez a fokszám még nem okoz túl nagy számítási overheadet, de nyitva hagyja a lehetőséget a paraméterek pontosítására, illetve egybevág a korábbi tapasztalatokkal. Azt, hogy egy tágabb tartományon milyen görbe írja le jól a rendszert, csak széleskörűbb mérésekkel lehetne kideríteni.

3.5.2. Hammerstein-Wiener modell

A BSc szakdolgozatomban végzett vizsgálatok tapasztalata az, hogy a rendszert a Hammerstein-Wiener modellel lehet a legjobban közelíteni. Ennek a működési elvét a 2.4.2 fejezetben részletezem, illetve további információ található róla a [1, 8] forrásokban.

A modell paramétereinek becsléséhez a *Matlab System Identification Toolbox* applikációját használtam. A modellillesztéshez elegendően hosszú adatsort készítettem, melyben több melegítési és lehűlési ciklus is szerepel. Ellentétben a statikus vizsgálatokkal, itt kifejezetten fontos, hogy a hirtelen változások benne maradjanak az adatsorban. A kiugró adatok kiszűrése természetesen itt sem hagyható el. A rendszer egybemenetű, egykimenetű (SISO[14]), a diszkrét mintavételi idő 1 másodperc (hiszen a

[14]Single Input Single Output

mikrokontoller csak másodpercenként, a PTP üzenetek hatására küld log adatokat).

Az applikáció számos modelltípust felajánl, melyeket érdemes ellenőrzésképp végigpróbálgatni. A Hammerstein-Wiener modell blokkjainak paramétereivel is érdemes eljátszani, és kiválasztani a legpontosabb eredményt. A pontosságot abban mérjük, hogy az identifikációs adatsor alapján számított modell mennyire jól becsli a verifikációs adatsor bemenete alapján annak kimenetét. Fontos, hogy az identifikációs és verifikációs adatok függetlenek legyenek, azaz két különböző mérésből származzanak. Előfordulhat ugyanis, hogy a modell „rátanul" egy adott mérés valamely hibájára, és ha ugyanazon adatsoron ellenőrizzük, akkor ez nem tűnik fel.

A legpontosabb eredményt úgy kaptam, ha a bemeneti nemlinearitást elhagytam (innentől fogva nevezhetjük Wiener-modellnek a rendszert), a kimeneti nemlinearitás egy harmadfokú polinom, a középső dinamikus blokk számlálójának és nevezőjének fokszáma 2 illetve 3.

Ebben az esetben is végigpróbáltam a kimeneti nemlinearitást több, különböző fokszámmal, és hasonló eredményeket kaptam, mint a statikus vizsgálatok során. Az első- illetve másodfokú polinomokat használó modellek csak kis mértékben maradnak alul. Magas fokszám esetén azonban torzul a modell kimenete, illetve a mikrokontrolleren való kiszámítás is komplikáltabbá válik, úgyhogy itt is a harmadfokú megoldást választottam aranyközépút gyanánt.

A dinamikus blokk a Matlabban egy *idpoly* típusú struktúra írja le [6]. Általános képlete a következő:

$$A(q)\cdot y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t) \tag{3.4}$$

Az egyenletben y(t) a kimenet (esetünkben az addend), u(t) a gerjesztő jel (termisztor), e(t) a zajt jelöli, az A, B, C, D és F pedig a *time-shift* operátorral kifejezett polinomok. Az egyenlet azonban nagyban egyszerűsödik. Esetünkben a zaj nulla, így a jobb oldali összeg második tagja teljesen eltűnik. Továbbá a modellillesztés kimenetében F(q) = 1, így az egyenlet tovább egyszerűsödik.

$$A(q)\cdot y(t) = B(q)u(t) \tag{3.5}$$

Az egyenletet kibonthatjuk, illetve diszkrét időbe áttérünk, így a következő képlethez jutunk:

$$a_0 \cdot y(k) + a_1 \cdot y(k-1) + ... = b_0 \cdot u(k) + b_1 \cdot u(k-1) + ...$$

$$y(k) = \frac{b_0}{a_0} \cdot u(k) + \frac{b_1}{a_0} \cdot u(k-1) + ... - \frac{a_1}{a_0} \cdot y(k-1) - \frac{a_2}{a_0} \cdot y(k-2) - ... \tag{3.6}$$

Az akutális y kiszámításához szükségesek a korábbi értékek, így azokat el kell tárolni. A modellt megvalósító C kódrészletnek tehát memóriával kell rendelkeznie. Ezen felül fontos, hogy az új y(k) kiszámítása mindig csak olyan gyakran hívódhat meg, amekkora diszkrét lépésközzel dolgoztunk Matlabban a modellillesztés során. Esetünkben ez 1 másodperc, hiszen másodpercenként jöttek adatok a mikrokontrollerből.

Ez azt jelenti, hogy mivel 1 másodperces diszkrét mintavételi idővel vettük a mintákat, így csak ugyanilyen mintavételi idejű modellt tud készíteni az applikáció, ami azt jelenti, hogy a mikrokontrolleren is csak másodperces gyakorisággal futtatható a kiértékelés. A végső cél azonban éppen az, hogy két Sync üzenet között, másodpercenként akár többször is lefusson a korrekció. Ehhez meg kell növelni a mintavételi időt. Nem szükséges azonban a mikrokontroller működésének bárminemű átírása. Elég, ha a System Identification Toolbox által felajánlott *Resample...* opcióval újramintavételezzük az adatsort többszörös mintavételi frekvenciával. Ekkor a szoftver a meglévő adatok közé interpolál extra regisztrátumokat. Az új mintákon elvégezve a modellillesztést, az új, megnövelt mintavételi frekvenciának megfelelő időállandókat kapunk.

A modellillesztés eredményei (1 másodperces lépésközzel) számszerűen a következőképp alakultak:

- Dinamikus blokk:

$$\begin{aligned} y(k) =& u(k) + 0.9985 \cdot u(k-1) \\ & - 1.5068 \cdot y(k-1) - 0.8086 \cdot y(k-2) - 0.3013 \cdot y(k-3) \end{aligned}$$

  Ezen blokk y(k) kimenete adja a következő blokk bemenetét.

- Statikus nemlinearitás:

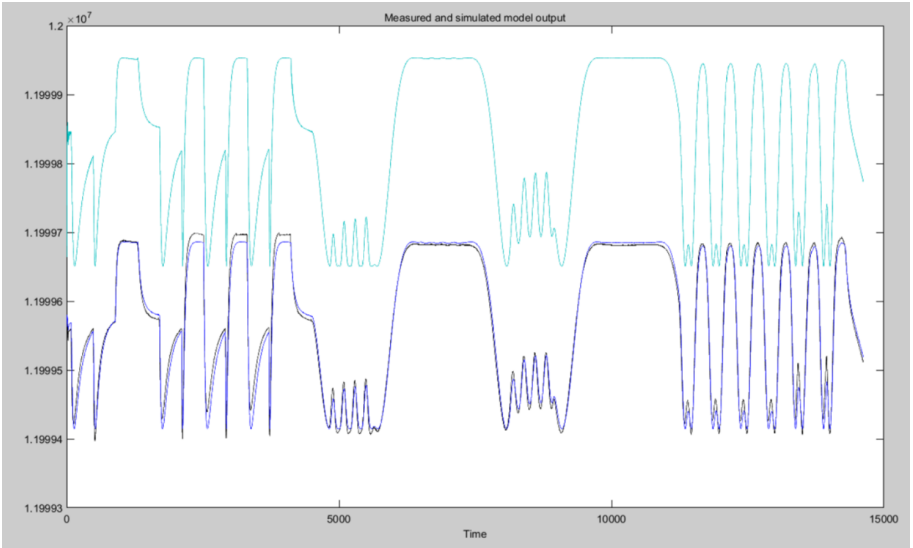
$$addend(x) = p_1 \cdot x^3 + p_2 \cdot x^2 + p_3 \cdot x + p_4$$

$$\begin{aligned} p_1 &= 6.3344 \cdot 10^{-5} \\ p_2 &= -0.17288 \\ p_3 &= 76.3226 \\ p_4 &= 3.4361 \cdot 10^{9} \end{aligned}$$

3.5.3. Adaptivitás

Számos esetben előfordulhatnak kisebb eltérések a modell becslése és a PTP eredménye között. A rezgőkvarcokon megfigyelhető az öregedés jelensége. Ez annyit jelent, hogy ugyanolyan hőmérsékleten, ugyanolyan körülmények között, de több hónapnyi eltéréssel vizsgálva egy adott oszcillátort, nem ugyanaz lesz a frekvenciája. Nem beszélve azokról az esetekről, amikor nem *ugyanolyanok* a körülmények. A modellünket a hőmérséklet változását vizsgálva készítettük, azonban a frekvenciát ezen kívül más környezeti paraméterek is befolyásolhatják. Továbbá a modellillesztésnek is lehetnek kisebb-nagyobb hibái. Ezekből kiindulva előfordulhat tehát, hogy a modell jellegre jól ugyan jól közelíti a valóságot, de közbeiktat egy offszetet a helyes értékhez képest, így a lokális óra egyenletes sebességgel elcsúszik (drift) a mesterórához képest.

Az öregedés jelenségére az MSc Önálló laboratórium 1 alatt lettem figyelmes. A modellillesztés során elővettem egy fél évvel korábban készített modellt. Ellenőrzés végett elkészítettem ennek a modellnek a másolatát. A megadott paraméterek, illetve a blokkok típusa azonos volt, viszont az identifikációra már egy frissen vett adatsort használtam. Ezt követően egy verifikációs adatsoron lefuttattam mind a régi, mind az új modellt, melynek eredményét a 3.11. prezentálja. Egy közel 5 órás mérés során különböző hőmérsékleti gerjesztéseket adtam a rendszerre. Az ábrán feketével látszik a valós frekvencia alakulása a gerjesztések hatására. Sötétkékkel az új, cián színnel pedig a régi modell kimenete látható. Az új modell kifejezetten pontosan becsüli meg a frekvenciát a hőmérséklet alapján (eltekintve a széleken lévő kisebb eltérésektől), a régi modell azonban el van csúszva frekvenciában felfelé. Jól látható, hogy a két becslés egymás „hasonmása" egy offszettől eltekintve. A fél évvel korábbi méréseken még jónak számító modell most átlagosan 240 Hz hibát hozott. Egy 12 MHz frekvenciájú oszcillátor esetén ez 20 ppm hibának felel meg. Az adaptív működés ezt a hibát képes kiszűrni.



3.11. ábra. *A kvarc öregedésének hatása*

Ezen javíthatunk, ha a modell által nem vizsgált környezeti tényezők hatását egyetlen konstanssal fejezünk ki, melyet hívjunk *korrekciós konstansnak*. Ezen értéket a szinkronizációból és a modell alapú becslésből adódó addend értékek különbségéből számítjuk.

A korrekciós konstans egy nulladfokú adaptív paraméter. Az adatptivitást kiterjeszthetnénk magasabb fokú paraméterekkel is, hiszen annál pontosabban tudjuk illeszteni a modellt, hogyha annak minél több szabadsági foka van. Azonban már egy elsőfokú igazításhoz is viszonylag széles hőmérsékleti tartomány kell, és kis zavarású mérések. Ha nagy zajban próbáljuk a meredekséget javítani, azzal ellentétes hatást érhetünk el.

Vannak olyan alkalmazási környezetek, ahol ez szóba jöhet. Például egy gépjárműben található oszcillátort jóval szélsőségesebb hőmérsékleti hatások fognak érni élete során, mint egy szerverteremben található berendezést. Az előbbinek így széles hőtartományon állnak majd rendelkezésére adatok, míg az utóbbinak többnyire csak egy ponton, illetve annak 1-2 °C-os környezetében. Mivel az általam vizsgált tartomány is aránylag szűk, így be kell érni a nulladfokú adaptivitással.

3.6. Modellek validációja

A kapott modellek működését úgy igazolhatjuk, ha implementáljuk azokat a mikrokontrolleren, és megnézzük, hogy valós időben hogyan teljesítenek. A teljesítmény mérésére számos módszerrel próbálkozhatunk. Legfontosabb mind közül azonban mégis az, hogy a lokális és a mesteróra közötti abszolút eltérés időfüggvénye hogyan alakul. Ennek megállapításához olyan mérést kell végezni, ami elégséges pontossággal egyszerre mintavételezi a lokális és mesterórát, majd kiszámítja ezek különbségét. Ezt kellő gyakorisággal (pl. másodpercenként) elvégezve megkaphatjuk az eltérés időfüggvényét, melyet deriválva előáll a *drift* értéke. **Az egész hőmérsékletkompenzációs eljárás lényege az, hogy a drift minél jobban közelítsen a nullához.**

Szerencsére az eljárás a FlexPTP-ben már megvalósult, és a különbség logolásra is kerül, így ezzel külön nem kell foglalkozni. Elég annyit változtatni az eredeti kódon, hogy a szinkronizáció alapján működő PD szabályzót lecserélem a saját, hőmérsékletkompenzációs becslőmre, és az ellenőrzéshez szükséges adatok már elő is álltak soros porton. Ezek Matlabban közvetlenül feldolgozhatók.

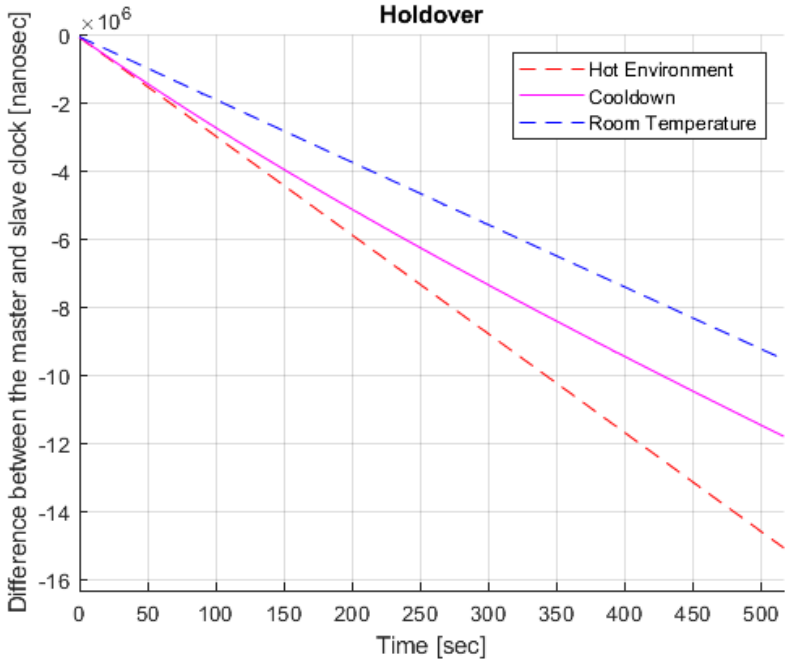
A drift számítása mellett árulkodó lehet az is, ha a PTP szinkronizációból származó addendet összehasonlítjuk a modell alapú korrekció eredményével, így ezt a tulajdonságot is vizsgálom (pl. időbeli alakulás, addendek különbségének átlaga, átlagának különbsége).

3.6.1. Kontroll adatok

Az összehasonlításhoz méréseket készítettem arról az esetről, amikor nem működik a szinkronizáció. Ekkor a mikrokontoller indulásakor egy fordítási időben fixált értékre inicializálódik az addend regiszter értéke. Ekkor a drift jelentős. Ezt szemlélteti a 3.12. ábra. Kékkel látható az óra elcsúszása szobahőmérsékleten, pirosssal 55 °C fokon.

Rózsaszínnel az látszik, amikor az óra 55 °C-on kapcsol be, majd visszahűl szobahőmérsékletre. (A kezdeti elcsúszást nullára állítottam be.) Látszik, hogy a rózsaszín görbe meredeksége kezdetben a piroséval egyezik meg, majd ahogy lassan kihűl a rendszer, egyre inkább a kék meredekségét veszi fel.

A drift értékét úgy kapjuk meg, ha kiszámítjuk a görbék meredekségét. A piros görbe esetén ez nagyjából 29000 ns/s, míg a kéknél 18500 ns/s. A rózsaszíné pedig ezen kettő érték között változik. Hasonló értékeket kaphatunk akkor, ha vesszük minden egyes görbe diszkrét deriváltjának átlagát. Ez a kék, rózsaszín és piros görbéknél rendre 18289, 19818 és 28988. Ez 18, 19, illetve 29 ppm hibát jelent. Ez napi 1.5 – 2.5 másodpercnyi pontatlanság, mely éves szinten 9-15 percnyi hiba.

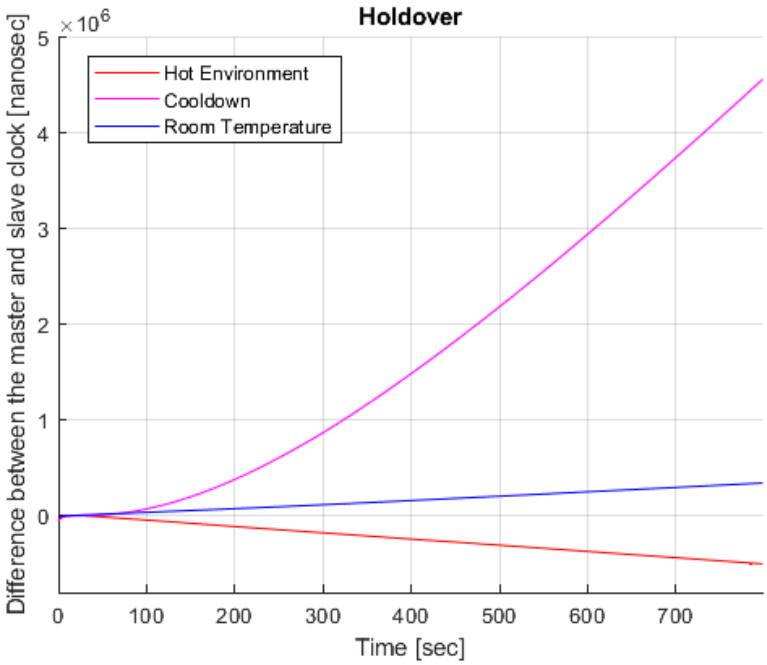


3.12. ábra. *A drift szinkronizáció nélkül*

Ennél lényegesen jobb eredményt kapunk, ha a holdover üzemmódot vizsgáljuk. Ilyenkor a szinkronizáció működik egy darabig, majd valamilyen hálózati hiba miatt nem jön több sync üzenet, így a legutoljára beállított addend értékkel dolgozik az eszköz. Ha ezt követően megváltozik a hőmérséklet, természetesen akkor is nagy driftekre számíthatunk. A holdover vizsgálatról készült eredményeket szemlélteti a 3.13. ábra. A színek jelentése azonos a 3.12. ábránál használtakkal. Megfigyelhető, hogy a görbék kezdeti szakasza még vízszintes: itt még működik a szinkronizáció.

A drift értékeket úgy számoltam ki, hogy lehagytam a kezdeti, laposabb szakaszt, hiszen itt még részben érvényesül a szinkronizáció hatása. A kék, rózsaszín és piros görbékre ez rendre 455 ns/s, 7409 ns/s, illetve -650 ns/s. Ezek az értékek jóval kisebbek az előző esethez képest. Ez várható volt, hiszen itt már egy egyensúlyi állapotból indulunk ki, így kezdetben az addend legalább nagyjából jó. A kék görbe meglepően jó eredményt hozott, hibája a mérés időtartama alatt mindössze 0.45 ppm. Ez még éves szinten is mindösszesen 0.24 másodperc eltérést okozna. Noha nem várhatjuk el, hogy 1 éven keresztül ilyen stabil

értéken maradjon az oszcillátor. A rózsaszín görbéből ez rögtön látható, hiszen azt mutatja, amikor az eszköz 55 °C-on szinkronizált, majd elvesztette a hálózati kapcsolatot, és lassan visszahűlt szobahőmérsékletre. Ekkor a hiba is jóval nagyobb, 7.4 ppm. Hosszabb mérésnél ez várhatóan még magasabb értékű lenne.

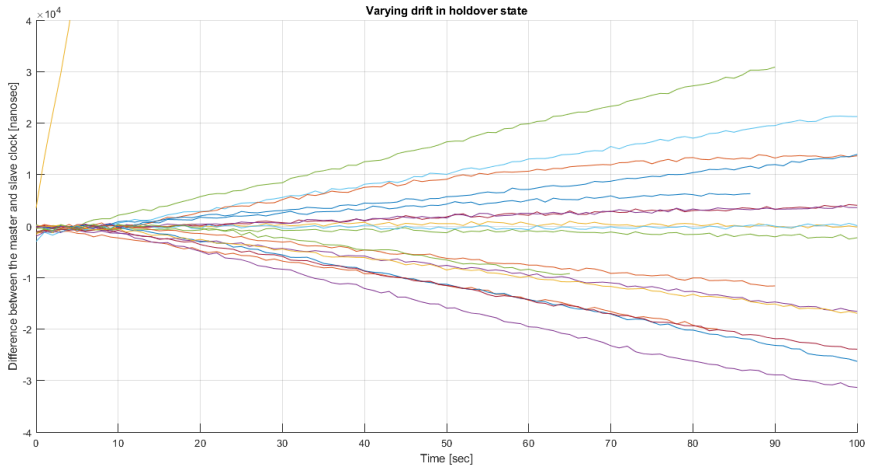


3.13. ábra. *Drift holdoverben*

Érdemes megvizsgálni azt is, hogy állandó hőmérsékleten mennyi a drift szórása. Mivel a lokális óra léptetési egysége 40 ns (3.1.1), így kvantálási zaj jelenik meg az időhiba értékében. Mivel ez szolgál a PD szabályzó bemenetéül, így a zaj ráterhelődik az addend állítására is. Ez azt jelenti, hogy a szinkronizáció hatására az addend pillanatnyi értéke cikázik a helyes érték körül. A zaj várható értéke természetesen nulla, hiszen ha egyik irányba kileng az addend, az éppen olyan irányú időeltérést eredményez, hogy a megváltozott hibajel alapján a szabályzó visszahúzza a jó érték felé.

Holdover üzemmódban azonban az történik, hogy a legutolsó szinkronizáció alapján számított addend értéket tartjuk mindaddig, amíg nem kapunk új sync üzenetet. Mivel azonban a szabályzó a mérési zaj, és az oszcillátor driftje miatt folyamatosan próbál kompenzálni, az addend is folyamatosan változik. Ez pedig azt jelenti, hogy a drift is mindig véletlenszerűen fog alakulni valamilyen valószínűségi eloszlás alapján.

A pontos sűrűségfüggvény kiszámításához rengeteg holdover mérés szükséges, és nem is ez a célunk, csupán egy közelítő képet szeretnénk kapni arról, hogy a drift milyen tartományon ingadozik. Ennek megállapítására több holdover mérést végeztem, melynek lényege, hogy bekapcsolás után hagytam elég időt a mikrokontrollernek, hogy szinkronizáljon a mesterórára (30 másodperc több mint elég), majd ezt követően egy időzítő kikapcsolja a kártyán az addend állítását. A sync üzenetek továbbra is érkeznek, így az időhibát másodperces felbontással nyomon tudjuk követni. Így kaptam a 3.14. ábrát. Mérések közben az eszközt igyekeztem állandó hőmérsékleten tartani, azonban



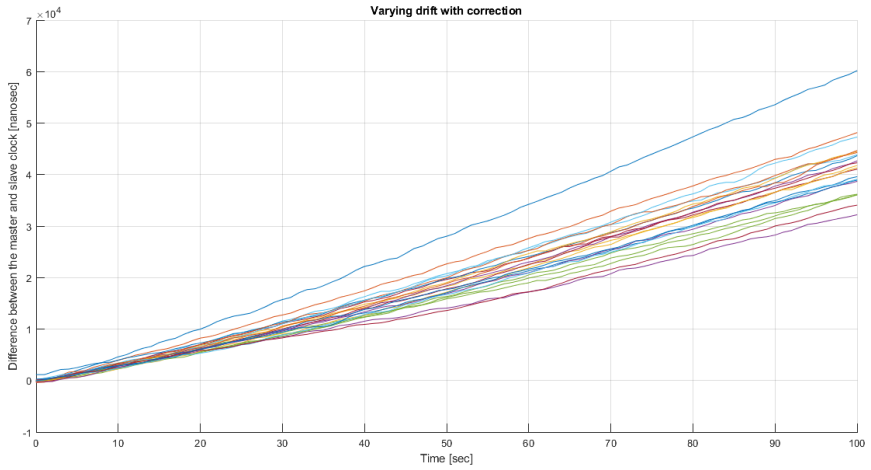
3.14. ábra. *A drift szórása holdover üzemmódban*

látható, hogy ez nem mindig sikerült, hiszen van olyan görbe, aminek a meredeksége valamelyest változik.

A kapott eredmények igazolják a feltételezést, miszerint a drift zajos. A grafikonon látható görbék zömének meredeksége ± 400 ns/s (0.4 ppm) tartományon belülre esik. Látható az ábrán azonban egy sárga színű egyenes is, melynek meredeksége drasztikusan eltér a többitől. Ezen mérés meredeksége nagyjából 9000 ns/s (9 ppm). Mivel a mérés során egyszer fordult elő, feltételezhető, hogy a szabályzó pillanatnyi hibája okozta a kiugró értéket. Ennek megállapításához jóval több mérésre van szükség. A tapasztalatok viszont azt mutatják, hogy a PD szabályzó kimenete valóban zajos (Pl. 3.8. ábra 3. grafikonja).

Az összehasonlítás végett megvizsgáltam a drift szórását a 3.5.2 fejezetben részletezett dinamikus modellel is. A hőmérséklet ugyanúgy állandó értékű volt ezen mérések során is. Ennek az eredményeit a 3.15. ábra mutatja. A görbék meredekségei 300 és 600 ns/s között mozognak. Előző esetben a görbék driftje egy 800 ns/s széles tartományon terült szét, míg itt ez a szélesség csak 300 ns/s. Látszik, hogy a szórás közel harmadára csökkent. Ezzel könnyebb dolgozni. Pl. a nulladfokú adaptív tagot úgy hangoljuk, hogy -450 ns/s eltérést hozzon be. Így a driftet ez esetben ±150 ns/s pontossággal minimalizálnánk.

Az is igaz, hogy bekapcsolt korrekcióval nem tapasztaltam olyan kiugró drifteket, mint nélküle, de a mérések száma olyan kicsi (20-30 db), hogy ebből még nem vonhatunk le következtetéseket a drift eloszlására.



3.15. ábra. *A drift szórása dinamikus korrekcióval*

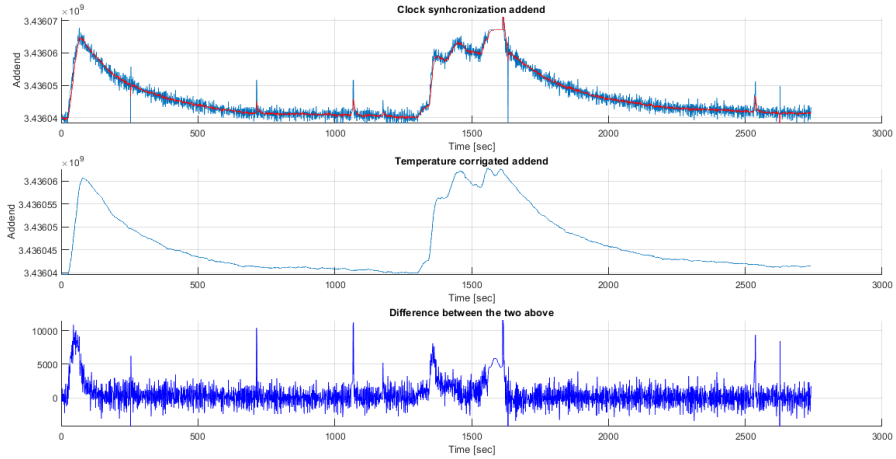
3.6.2. Statikus nemlinearitás

A feladat egy harmadfokú függvény megírása, mely a termisztor értékéből megadja, hogy mire állítsuk be az addendet. A függvénynek nincs állapota, egyszerűen a bemenet alapján kiszámolja a kimenetet. A hőmérsékletmérést elvégző taszkból, a beolvasást követően rögtön meghívható, nincs külön megkötés rá (ezzel szemben a Wiener-modellnél láthatjuk majd, hogy annak meghatározott időállandója van, így nem lehet bármilyen gyakorisággal futtatni). A feladat nem bonyolult, azonban oda kell figyelni arra, hogy a Matlab függvényilleszéséből kijövő együtthatók a 32 biten ábrázolható legnagyobb szám nagyságrendjébe esnek. Ezen felül a (3.3) képletben összeadandó tagok között több nagyságrendnyi különbség van, így a lebegőpontos számábrázolás kerülendő, hiszen pontosságvesztés léphet fel. Ezeket szem előtt tartva 64 bites, fix pontos számokkal dolgoztam.

Az ellenőrző mérés során (hasonlóan az eddigiekhez) felmelegítettem a nyákot nagyjából 55 – 60 °C-ra, majd megvártam, míg visszahűl szobahőmérsékletre.

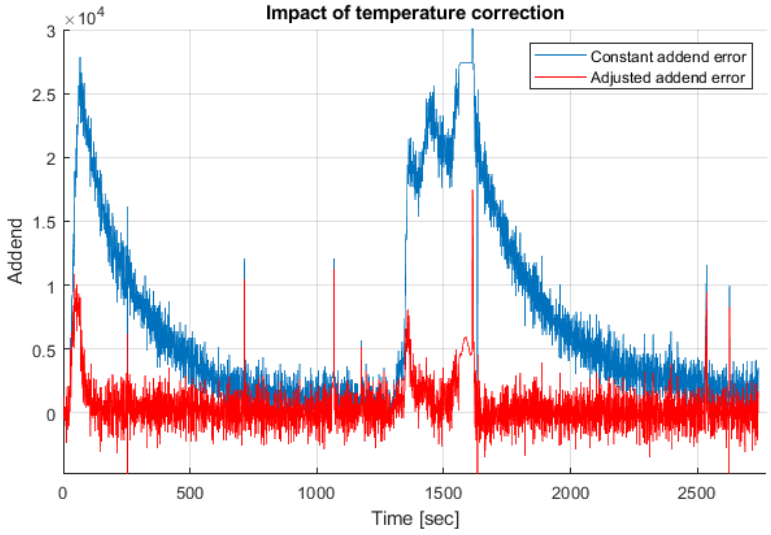
A 3.16. ábrán látható a modellillesztés eredménye. A legfelső ábrán látszik az addend tényleges értéke, melyet továbbra is a FlexPTP állít másodpercenként. A második ábrán látható az általam írt függvény kimenete a termisztor pillanatnyi értéke alapján. A harmadik ábrán pedig ezen kettő közti különbség látható. A pillanatnyi zajoktól eltekintve elmondható, hogy jellegre igen jól követi a modell a valóságot.

A korrekció hatását a 3.17. ábra szemlélteti. Kékkel látható az addend hibája akkor, ha konstans értéken állna (pl. hálózati hiba, holdover üzemmód), pirossal pedig a korrekció utáni hiba látható. A hiba itt a szinkronizált addendtől való eltérés idődiagramját jelenti. A hőmérsékleti gerjesztés azonos a 3.16. ábrán bemutatott mérésével. Abban az esetben, amikor a hőmérsékletváltozás lassú (lehűléskor), a modell hibája kicsi. A probléma a hirtelen felfutó éleknél van. Ez várható volt, hiszen a függvény illesztésekor megtisztítottuk a mintákat a gyors változásoktól, mert csak a statikus



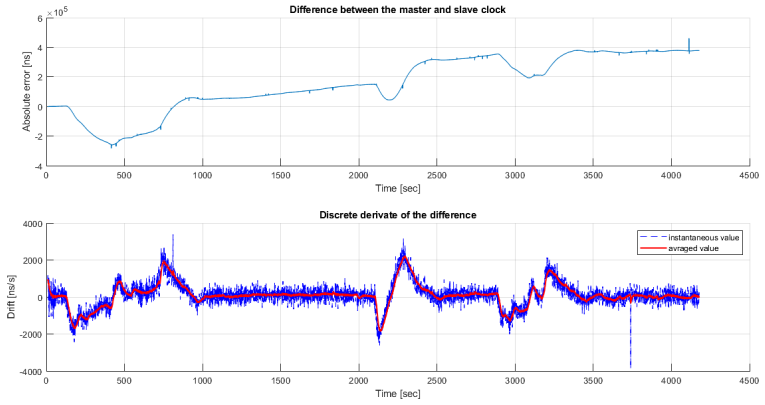
3.16. ábra. *a statikus nemlinearitás beiktatásának eredménye*

karakterisztikára voltunk kíváncsiak (3.9. ábra). Viszont még ezzel együtt is kijelenthetjük, hogy a modell *elősegíti* az óra pontosabbá tételét.

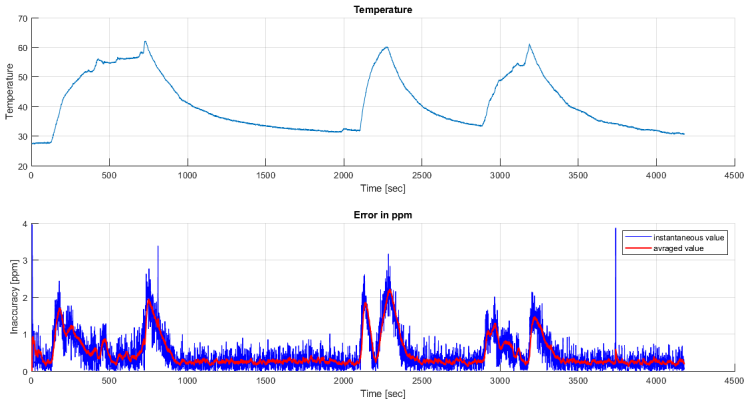


3.17. ábra. *A statikus korrekció hatása az addendra*

A tényleges használhatóságot azonban a drift vizsgálatával tudjuk megmondani. Ennek szemléltetésére ábrázoltam a lokális és a mesteróra eltérését (abszolút hiba), illetve ennek diszkrét idejű deriváltját (drift). Az eredményt a 3.18. és 3.19. ábra mutatja. A mérés során alkalmazott hőmérsékleti gerjesztés a 3.19. ábra felső grafikonján látszik. Az abszolút hiba többnyire egyenletes meredekséggel kúszik felfele, kivéve, amikor gyors hőmérsékletváltozás éri a rendszert. Ilyenkor a drift 2000 ns/s feletti értékeket is felvesz (>2 ppm). A lassú szakaszokban viszont többnyire 1000 ns/s (1 ppm) alatt marad az értéke.



3.18. ábra. *A statikus nemlinearitás abszolút hibája, illetve driftje*

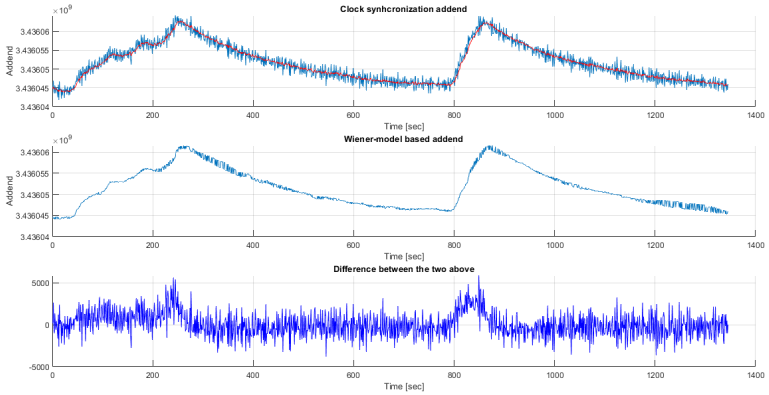


3.19. ábra. *A hőmérsékleti gerjesztés, és a drift abszolút értéke*

A 3.19. ábra alsó grafikonján a drift abszolút értéke, illetve pirossal annak átlaga látható. Ez az a görbe, ami a legjobban reprezentálja a vizsgált kompenzációs eljárás hatékonyságát. Az elvárásoknak megfelelően a kompenzáció rosszul teljesít a hirtelen melegedő/lehűlő szakaszokon, azonban meglepően jó eredményt mutat a lassú szakaszokban.

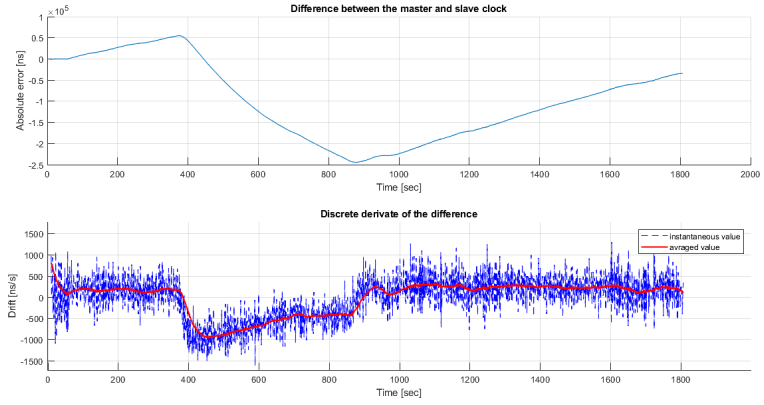
3.6.3. Hammerstein-Wiener modell

A modell két blokkból áll: egy lineáris dinamikus részből, illetve egy statikus nemlinearitásból. Az utóbbi megvalósítása megegyezik a 3.6.2 fejezetben leírtakkal, csak az együtthatók valamelyest eltérnek. A becsült addend jóságát a 3.20. ábra mutatja. Hasonlóan a 3.16. ábrához, a felső görbe mutatja a szinkronizációból eredő addendet,a középső a Wiener-modell becsült kimenete, az alsó pedig ennek a kettőnek a különbsége. Ha összevetjük az itt látottakat a 3.16. ábrával, némi javulást láthatunk, hiszen a különbség grafikonja itt 5000 környékén tetőzik, míg statikus esetben ez felmegy 10000-ig is. Ez egy jó jel, de az igazi javulást a drift vizsgálatával deríthetjük ki.



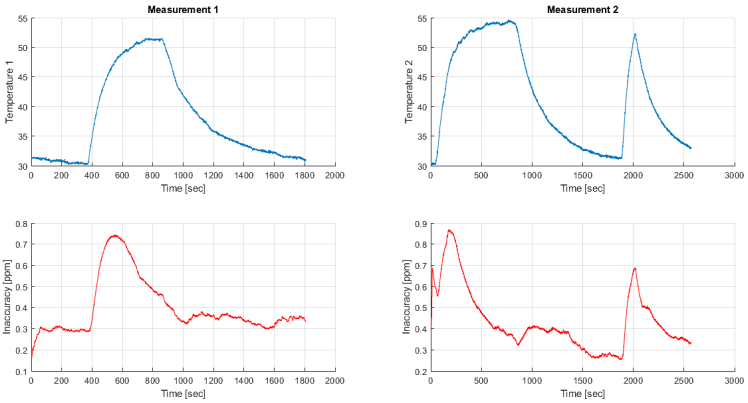
3.20. ábra. *A Wiener-modell kimenetének vizsgálata*

A driftet úgy kapjuk, hogy vesszük a lokális és a mesteróra közötti eltérés időfüggvényét, majd diszkrét időben deriváljuk azt. Ez a hiba meredeksége, mely azt adja meg, hogy másodpercenként hány nanoszekundumot téved az óra, így mértékegysége ns/s. Mivel a drift zajos, így értemes kiátlagolni a kapott eredményt. A számítást szemlélteti a 3.21. ábra. Ehhez már egy újabb mérést használtam fel, mely hosszabb ideig tartott, illetve a rendszer jelentősebb hőmérsékleti hatásoknak volt közben kitéve. A felső görbe az abszolút eltérést mutatja, az alsó pedig ennek deriváltját, illetve deriváltjának átlagát. Ha az abszolút hiba legmeredekebb szakaszát vizsgáljuk (400 – 800 sec között), az erre számított drift értéke nagyjából 610 ns/s. Ez 0.61 ppm pontosságot jelent, mely naponta 50 miliszekundum, évente pedig 19 másodpercnyi hibát okozna.



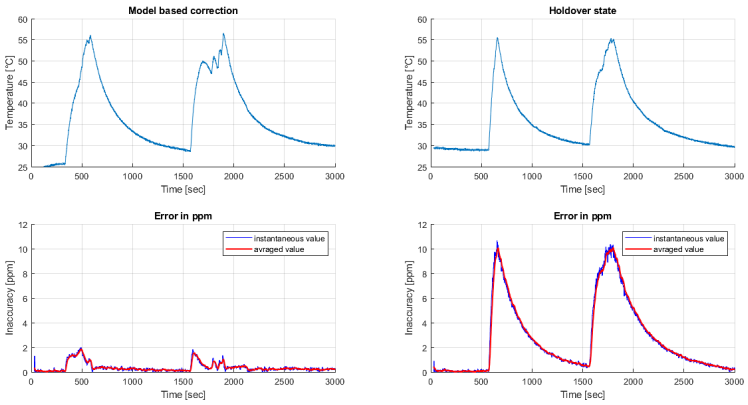
3.21. ábra. *A drift számítása*

Az átlagos driftnél beszédesebb az átlagos drift abszolút értéke, mert ez megadja a rendszer pontosságát. Ezt szemléltettem a 3.22. ábra. 1 ppm 1000 ns/s-nek felel meg. Két mérést végeztem, melyek hőmérsékleti gerjesztése a felső ábrákon látszik, alattuk pedig a pontosság ppm-ben. Látható, hogy a modell olyan jól vezérli az addendet, hogy a még a meredek hőmérsékletváltozások közepette is 1 ppm alatt tartja a hibát.



3.22. ábra. *A drift értéke a hőmérséklet változására*

A modell jelentőségét jobban szemlélteti a 3.23. ábra. Itt az első oszlopban egy modell alapú korrekció, a második oszlopban pedig egy holdover mérés eredményei láthatók. A két mérés során igyekeztem hasonló hőmérsékleti gerjesztéseket adni mindkét üzemmódban. Látható, hogy a korrekció a hibát mindig 2 ppm alatt tartja. Korrekció nélkül a hiba elszáll.



3.23. ábra. *A hiba korrekcióval és anélkül*

3.6.4. Korrekciós konstans

A korrekciós konstans számítása úgy zajlik, hogy vesszük a szinkronizációból adódó addend átlagát (pl. IIR filterrel számítva), és kivonjuk belőle a modellből kijövő becsült addend értékét. Ezzel megkapjuk a modell átlagos offszetjét. A korrekcióban ezt hozzáadjuk a beállítandó addend értékéhez, így pontosabb eredményt kapunk.

A korrekciós konstans megjelenésével adaptívvá tettük a rendszerünket. Az adaptivitás nyilvánvalóan csak hálózati kapcsolat mellett tud lefutni. Holdover üzemmódban tehát ez az érték nem frissül, hiszen nincs referencia jel, amihez hangolni lehetne.

A módszer a kvarcok öregedésénél tárgyalt hatások kiküszöbölését eredményezi. A hatását bemutatni ennélfogva nem egyszerű, hiszen jól identifikált modellek esetén a kezdeti offszethiba nullához közeli.

3.6.5. Eredmények összehasonlítása

A mérési eredményeket, illetve az adatlapokból kigyűjtött információkat az alábbi táblázatban tűntettem fel.

	átlagos drift [ns/s]	hiba [ppm]	becsült napi eltérés [sec]	becsült éves eltérés [sec]	*ár**	megjegyzés
Korrekció nélkül	20000-30000	20-30	1.7-2.6	630-946 (10-15 perc)	-	-
Holdover (állandó, ill. változó hőmérséklet)	500-15000	0.5-15	1.3	473 (7.9 perc)	-	az addend zaja befolyásolja
Statikus nemlinearitás	1000-2000	1-2	0.86-0.173	31.5-63	termisztor ára (7 Ft-tól)	adaptivitás
Wiener-modell	500-1000	0.5-1	0.043-0.086	15.8-31.5	termisztor ára (7 Ft-tól)	adaptivitás
$TCXO^*$	100-10000	0.1-10	0.0086-0.86	3.15-315	100-30000 Ft	jitter és öregedés jelensége
$OCXO^*$	$<$120	$<$0.12	$<$0.001	$<$3.8	$>$10000 Ft	nagy fogyasztás és fizikai méret

*: a farnell.hu webáruház alapján

4. fejezet

Az eredmények összegzése

4.1. Összefoglalás

A munkám során elkezdtem a szoftver-definiált hőmérséklet kompenzált oszcillátor megalkotásának menetét. Megismerkedtem a PTP-vel, illetve az ezt alkalmazó hálózati óraszinkronizációval, mely alkalmazható beágyazott rendszerekre.

Elkészítettem a szabályzóhoz szükséges hardver feltételeket. Megvizsgáltam a hőmérsékletmérés lehetőségeit, és implementáltam egy termisztoros, illetve egy infravörös megoldást a következő módon:

- kiválasztottam a megfelelő mikrokontrollert, illetve termisztort,
- készítettem egy kiegészítő áramkört,
- elvégeztem az áramkör élesztését,
- zavarvizsgálatot végeztem a termisztoron.

Ezt követően kibővítettem a szinkronizációt elvégző FlexPTP szoftvert a hőmérséklet mérésével, illetve annak logolásával. A laborban adatokat gyűjtöttem, melyeket Matlabban feldolgoztam. A termisztorról beolvasott digitális érték és a lokális időt tároló akkumlátor regiszter léptetési egysége (addend) közötti összefüggésre modellt alkottam. Több különböző modellt megvizsgáltam. Először egy statikus nemlinearitással, később pedig egy Hammerstein-Wiener modellel foglalkoztam.

A kapott modelleket megvalósítottam a mikrokontrolleren, C nyelven. A kód helyes működését leteszteltem. Összehasonlítottam a szinkronizációból eredő, illetve a saját modelljeim által számított addend értékek közötti hasonlóságokat. Ezt követően az óra hibáját legjobban szemléltető drift hatását néztem meg. A mérések alapján kiszámítottam a drift értékét akkor, amikor nincs szinkronizáció, holdover üzemmódban, statikus nemlinearitást használva, illetve a Hammerstein-Wiener modell alapján működő korrekció esetén is. A méréseket ábrákkal szemléltettem, illetve az eredményeket táblázatosan összefoglaltam.

Az eredmények láttán elmondható, hogy a dolgozatom elején kitűzött célt sikerült elérni, ugyanis sikerült egy egyszerű, kompenzálatlan kristály oszcillátor hibáját 1 ppm

alá csökkenteni egyetlen darab olcsó termisztor, a PTP alapú szinkronizáció, és egy megfelelő algoritmus segítségével. A rendszer nem csupán eléri egy átlagos TCXO pontosságát, hanem ezen felül kiküszöböli az eszköz öregedéséből adódó hibákat is.

A mérési eredmények jól szemléltetik, hogy rengeteg új lehetőséget nyit meg az órajel szoftveres kompenzációja, melyek flexibilitási előnyt biztosítanak egy hagyományos TCXO-val szemben.

4.2. Továbbfejlesztési lehetőségek

A mérések során egy hőlégfúvót használtam a hőmérsékleti gerjesztéshez. Ez közel sem ideális, hiszen nem érhetünk el vele bármilyen hőmérsékletet, illetve a tranziensek lefutása is viszonylag kötött. Feltételezhetően a modell pontossága tovább javítható volna abban az esetben, ha a minta, amin identifikálunk, nagyobb hőmérsékleti tartományt ölel fel, illetve különböző meredekségű tranzienseket is felvesz. Hideg környezetben továbbá egyáltalán nem volt lehetőségem vizsgálatokat végezni. Egy hőkamra megoldást nyújthat ezekre a problémákra, azonban ennek használatára sajnos nem nyílt lehetőségem.

Továbbra is kérdés a különböző irányból érkező hőmérsékleti tranziensek hatása. Egy hőmérőt használva nem tudjuk ezeket kiküszöbölni, hiszen ha az oszcillátor felől érkezik a tranziens, akkor a hőmérő lemarad a kvarc hőmérsékletéhez képest, ha az ellenkező irányból érkezik a változás, akkor pedig fordítva. Erre megoldást jelenthet kettő vagy több hőmérő használata, mellyel körbevesszük az órajel generátort. A műszerek által mért értékeknek használhatjuk az átlagát, de alkothatunk több bemenetű, egy kimenetű (MISO) modellt is.

A rendszer valós időben végzi el a kompenzációt, így már egy valós alkalmazásban is elképzelhető a használata. A modellillesztés azonban még mindig offline történik, ami kényelmetlenné teszi az implementációt. Először be kell gyűjteni az adatokat, azokat valamilyen formában ki kell küldeni a *„Cloud”*-ba, egy külső eszközön fel kell dolgozni (pl. Matlabbal), majd a kapott együtthatókat vissza kell juttatni a beágyazott eszközre. Sokkal kényelmesebb volna, ha a modellillesztés lokálisan, a mikrokontrolleren futna le. Ez felveti persze azt a kérdést, hogy van-e elég számítási kapacitása a kártyának.

A megfelelő modell kiválasztásakor felmerült a neurális hálók használata is. Ez a kutatási terület mély és szerteágazó, jelentős előismereteket igényel. Azonban ugyanúgy, mint a lokális modellillesztésnél, itt is felvetődik a kérdés, hogy egy beágyazott eszközön vajon van-e elég számítási kapacitás ennek futtatására.

Köszönetnyilvánítás

Szeretnék köszönetet mondani Wiesner Andrásnak az óraszinkronizációs algoritmusának ismertetéséért, és segítőkész hozzáállásáért.

Szeretném továbbá megköszönni a Méréstechnika és Információs Rendszerek Tanszék munkatársainak a laboratóriumi munkáimhoz nyújtott segítséget, tanácsokat, ötleteket.

Külön köszönöm konzulensem, Dr. Kovácsházy Tamás közreműködését, aki az elmúlt 6 szemeszterben nyújtott kimagasló mentori tevékenységével és iránymutatásával segített a projektfeladataim megvalósításában. Köszönöm továbbá szemléletformáló munkáját, mellyel a mérnöki gondolkodásmódot igyekezett átadni.

Ábrák jegyzéke

Irodalomjegyzék

[1] Vozár Viktor. Oszcillátorok hőmérséklet-frekvencia összefüggésének a vizsgálata és modellezése. `https://diplomaterv.vik.bme.hu/hu/Theses/Oszcillatorok-homersekletfrekvencia`. Hozzáférés: 2022.10.31.

[2] Viktor Vozár and Tamás Kovácsházy. Self-learnning of the dynamic, non-linear model of frequency-temperature characteristic of oscillators for improved clock synchronization. In *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6. IEEE, 2021.

[3] András Wiesner and Tamás Kovácsházy. Portable, ptp-based clock synchronization implementation for microcontroller-based systems and its performance evaluation. In *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6. IEEE, 2021.

[4] Kenji Nemoto and K-I Sato. A 2.5 ppm fully integrated cmos analog tcxo. In *Proceedings of the 2001 IEEE International Frequncy Control Symposium and PDA Exhibition (Cat. No. 01CH37218)*, pages 740–743. IEEE, 2001.

[5] Se-Joong Lee, Jin-Ho Han, Seung-Ho Hank, Joe-Ho Lee, Jung-Su Kim, Min-Kyu Je, and Hoi-Jun Yoo. One chip-low power digital-tcxo with sub-ppm accuracy. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 17–20. IEEE, 2000.

[6] Lennart Ljung. System identification toolbox. *The Matlab user's guide*, 1988.

[7] András Wiesner. Óraszinkronizáció és adatgyűjtés mikrokontrolleres beágyazott rendszerekkel. `https://tdk.bme.hu/VIK/halo3/Oraszinkronizacio-es-adatgyujtes`. Hozzáférés: 2022.10.31.

[8] Adrian Wills, Thomas B Schön, Lennart Ljung, and Brett Ninness. Identification of hammerstein–wiener models. *Automatica*, 49(1):70–81, 2013.

[9] Tiva™ tm4c1294ncpdt microcontroller data sheet, 2007. `https://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf`, Hozzáférés: 2022.10.31.

[10] Am2434 quad-core arm® cortex-r5f based mcu with industrial communications and security up to 800 mhz. https://www.ti.com/product/AM2434. Hozzáférés: 2022.10.31.

[11] Tmp23x-q1 automotive grade, high-accuracy analog output temperature sensors. https://www.ti.com/lit/ds/symlink/tmp235-q1.pdf?ts=1653217945016, 2019. Hozzáférés: 2022.10.31.

[12] Ntc thermistors for temperature measurement. https://product.tdk.com/system/files/dam/doc/product/sensor/ntc/chip-ntc-thermistor/data_sheet/50/db/ntc/ntc_smd_automotive_series_0402.pdf. Hozzáférés: 2022.10.31.

[13] Tivaware™ peripheral driver library. https://software-dl.ti.com/tiva-c/SW-TM4C/latest/exports/SW-TM4C-DRL-UG-2.1.4.178.pdf. Hozzáférés: 2022.10.31.

[14] Single and dual zone infrared thermometer datasheet, mlx90614 family. https://www.hestore.hu/prod_getfile.php?id=13182, 2019. Hozzáférés: 2022.10.31.

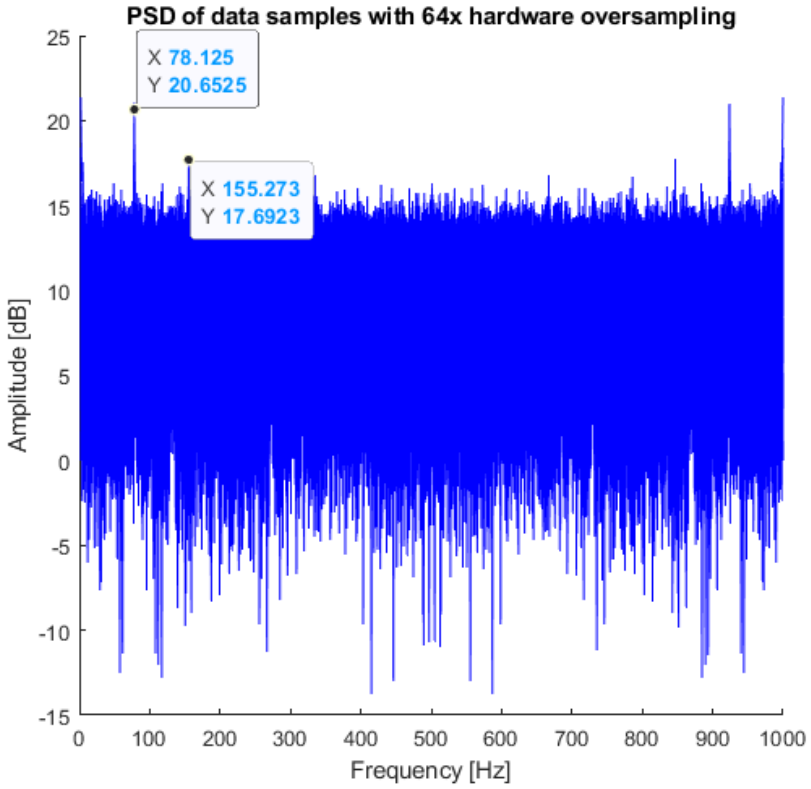
[15] Adafruit mlx90614 library. https://github.com/adafruit/Adafruit-MLX90614-Library, 2022. Hozzáférés: 2022.10.31.

[16] Tivaware™ examples user's guide. http://mercury.pr.erau.edu/~siewerts/cec320/documents/Manuals/SDK-PDL/SW-TM4C-EXAMPLES-UG-2.1.4.178.pdf, 2010. Hozzáférés: 2022.10.31.

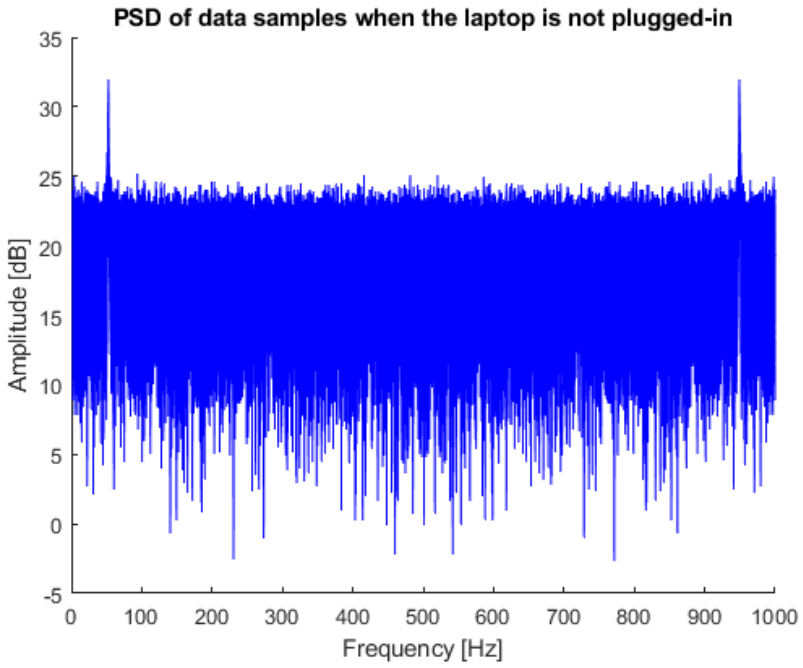
[17] ISO 16750-4: 2010. Road vehicles—environmental conditions and testing for electrical and electronic equipment—part 4: Climatic loads, 2010.

Függelék

F.1. Zavarvizsgálat

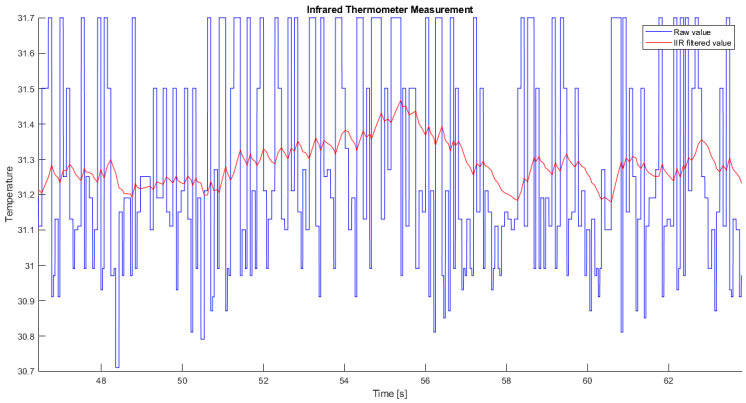


F.1.1. ábra. *A túlmintavételezett adatsor spektrumképe*

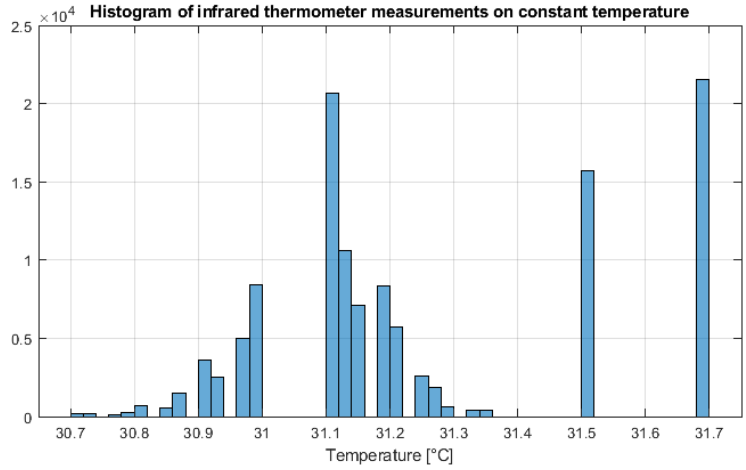


F.1.2. ábra. *a mintasor spektruma, kihúzott töltőkábellel*

F.2. Infravörös hőmérő pontosságának vizsgálata



F.2.1. ábra. *Az infravörös hőmérő zaja állandó hőmérsékleten*



F.2.2. ábra. *Az infravörös hőmérő mérési eredményeinek hisztogramja*