Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Óraszinkronizáció és adatgyűjtés mikrokontrolleres beágyazott rendszerekkel

TDK DOLGOZAT

Készítette

Wiesner András

Konzulens

dr. Kovácsházy Tamás

2020. október 28.

Tartalomjegyzék

Kivonat

Világunkban, az élet minden területén – ha nem is tudatosan – mérőrendszerek vesznek körül minket, melyek többé-kevésbé periodikusan, sokszor szinkron mintavételezik a környezetet (gondoljunk csak az okoseszközökre), sőt egyes elosztott, hálózatba kapcsolt mérőrendszerek egyenesen megkövetelik a szinkronizált működést. Hálózatba kapcsolt beágyazott mérőrendszerek esetén a sokféle szinkronizációs lehetőségből kiemelkedik Ethernet szabványos óraszinkronizációs megoldása az (IEEE 1588) Precision Time Protocol (PTP), mely nem csak komplex, hanem egyszerű, mikrokontrolleres rendszereken is implementálható.

Jelen dolgozatban a PTP beágyazott eszközökben való alkalmazhatóságának lehetőségét és szerepét mutatom be egy konkrét, Texas Instruments TM4C1294 ARM Cortex-M4F-magos – PTP-támogatással rendelkező beépített Ethernet-vezérlővel bíró – mikrovezérlőre fejlesztett *saját implementáción* keresztül. Részletesen bemutatom a hardverválasztás és a szabályozótervezés kérdéskörét, majd a kész rendszerrel végzett méréseket összehasonlítom a már létező, jól működő LinuxPTP-megvalósítás szinkronizációs képességeivel.

Abstract

In our everyday life we're transparently surrounded by measurement systems sampling our physical environment periodically, often as nodes of synchronized measurement systems (eg.: IoT-s, smart devices etc.). Numerous networked embedded measurement systems require precise synchronization to make them producing consistent a coherent measurement results. Many ways synchronization may happen in distributed systems, but the Ethernet-standard IEEE 1588 Precision Time Protocol using the same network as sychronization and data transfer medium is one of the greatest options for networked systems, implemented not only in complex, but in simple, MCU-based systems.

In this paper I introduce the fitness of PTP in embedded systems using my implementation developed onto the Texas Instruments TM4C1294 Coretx-M4F core network-enabled PTP-capable MCU. In details I present the aspects of choosing the right hardware and the tuning of the controller through several measurements, and lastly I compare the performance to the well-known LinuxPTP implementation.

1. fejezet

Bevezető

Napjainkban a különféle tudományos kutatások egyre összetettebb mérési eljárásokat, eszközöket igényelnek, melyek több fizikai mennyiség *egyidejű*, *szinkronizált* mintavételezését követelik meg egy kis térrészen belül, vagy akár egymástól egészen távol eső helyeken. Az ilyen és ehhez hasonló igényekre kínál megoldást az elosztott, flexibilis mérőrendszerek használata, melyeket egy közös hálózatra (pl.: Ethernet, CAN stb.) csatlakoztatott, szinkronizált mérőegységek – node-ok – rendszereként lehet elképzelni.

A szinkronizációt bár sokféleképpen meg lehet valósítani, ideértve a legegyszerűbb (pl.: szinkronjel közvetlen bekötése különálló vezetékekkel) módszereket is, de különösképpen a topológiát és a fizikai kiterjedést tekintve nagy hálózatok esetén célszerű a szinkronizációt – valamilyen ügyes algoritmus alkalmazásával – az adattovábbító hálózaton keresztül megvalósítani.

Bátran állíthatjuk, hogy a legtöbb vezetékes, nagy teljesítményű elosztott (beágyazott) rendszer Ethernet-hálózaton keresztül képes kommunikálni, ami a világon egyben a végberendezések számát és az átáramló adatmennyiséget tekintve a legelterjedtebb a nagy sebességű adatkommunikációs hálózatok között, mely komoly általános és egyre növekvő ipari jelentőséggel bír.

Az Ethernet szabványos óraszinkronizációs megoldása az (IEEE 1588) Precision Time Protocol [9] (röv.: PTP). A legtöbb ma létező implementáció vagy nyíltan nem hozzáférhető (nem open-source), vagy bár sok nagyon pontos [1], de túl komplex, ezért elosztott mérőrendszerekben nem alkalmazható, vagy nem biztosít kellően nagyon fokú szinkronizációt, holott nagy igény mutatkozik egyszerű PTP-képes eszközökre (pl.: beágyazott Linux-rendszerekre, melyek rendszerszinten szinkronizálhatók [11]; nagy pontossággal szinkronizálható mikrokontrolleres beágyazott rendszerekre), melyekkel mint „építőkockákkal”

lehetőség nyílna alapvetően kis bonyolultságú, de mégis a szinkron mérést megkövetelő elosztott mérőrendszerek *egyszerű* fejlesztésére. Kutatások igazolják, hogy megfelelő hardverelemekkel és implementációval a szinkronizációs hiba még beágyazott rendszerekben is akár 10ns alá [13] vagy tovább csökkenthető[8], lehetővé téve több tíz megahertzes vagy nagyobb sávszélességű jelek pontos szinkron mintavételezését.

A teljes rendszer a PTP-algoritmus működéséből kifolyólag nemlineáris, nem időinvariáns, így a jól ismert módszerekkel lineáris szabályozót csak nagyon nehezen lehet tervezni hozzá[13]. A kutatás során az alkalmazott szabályozók beállási és követési tulajdonságait visszamérve vontam le következtetéseket a szabályozók és a teljes rendszer tulajdonságaira nézve[10], majd az általánosan jól működő, beágyazott Linux-rendszerek is alkalmazott LinuxPTP-vel[7] végzett mérésekkel összevetettem a saját eredményeimet.

2. fejezet

Óraszinkronizáció PTP-vel

A PTP alkalmazásával Ethernet-hálózatokon könnyedén elérhető a hálózatra kapcsolódó eszközök igen pontos, egymáshoz képest 100ns-nál kisebb hibájú szinkronizációja. Ebben a fejezetben részletesen bemutatom az óraszinkronizáció algoritmusát.

2.1. A szinkronizáció algoritmusa

A szabványnak többféle verziója használatos, melyek nem teljesen kompatibilisek egymással, jelen írásban az általam implementált *IEEE 1588 v2* méréstechnikában alkalmazott [9] szabvány-profilt (UDP-kommunikáció IPv4 felevett, másodpercenként egy sync üzenet) mutatom be az slave-eszközök oldaláról. Az implementációm kizárólag slave-ként tud működni.

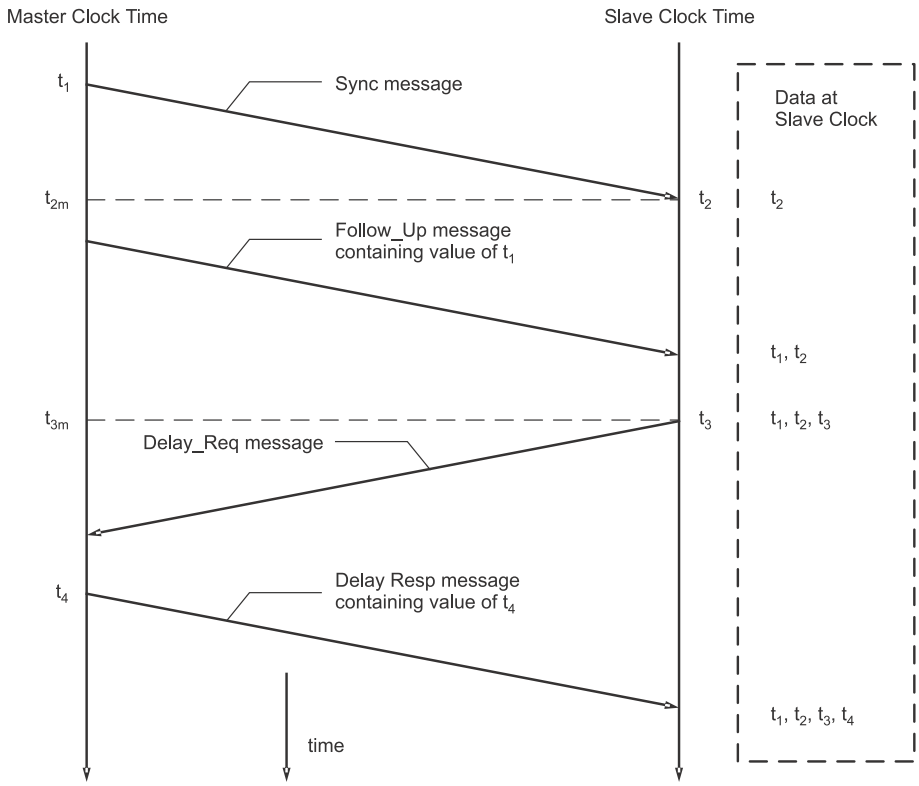
A protokoll vagy UDP-csomagokon keresztül IP-felett vagy multicast Ethernet-csomagokkal képes működni, a csomagok „hasznos” üzenettartalma mindkét esetben megegyező struktúrájú.

Legegyszerűbb esetben a hálózatra egy mesteróra (master eszköz) és valahány slave eszköz kapcsolódik. A mesteróra szolgáltatja a pontos idő-üzeneteket periodikus multicast-üzenetekben, amire a slave eszközök szinkronizálni tudnak.

2.1.1. Csomagok

A PTP-szinkronizáció az alábbi ötféle üzenetből áll össze:

- **Announce**: A mesterórák kommunikációjához, a Best-Master-Clock algoritmus futtatásához szükséges, a slave-órák szinkronizációjában nincs szerepe. Továbbiakban nem tárgyalom.



2.1. ábra. A PTP algoritmusa[6]

- **Sync**: A mesteróra küldi periodikusan multicast csomagként, a slave rögzíti a saját órája szerint a csomag érkezésének idejét. Általában az üzenet pontos időinformációt nem hordoz, vagy lényegtelen.

- **Follow_Up**: Mivel a mesteróra a sync üzenet kiküldésének t_1 idejét (általában[1]) csak annak kiküldése után ismeri meg, ezért ezt egy további üzenetben ki kell küldeni a szinkronizálódó slave eszközök számára. Ebben az üzenetben kapják meg a slave-ek az általuk t_2 időpillanatban fogadott üzenet kiküldésének pontos idejét. Ez az üzenet (bizonyos határokon belül) mindegy, hogy mennyit késik, mert nem az érkezési vagy a küldési ideje alapján, hanem a benne tárolt információt felhasználva történik meg a slave eszközökön az óra korrekciója. Ezt a szinkronizációs eljárást, mivel a szinkronizáció két lépésben történik meg, *two-step* szinkronizációnak nevezzük.

- **Delay_Req**: A slave eszközök küldik a mesterórának, miközben rögzítik a kiküldés t_3 időpontját a saját órájuk szerint. Az üzenetet a mesteróra a mesteróra ideje szerintit_4 időpontban kapja meg.

[1] Létezik single-step szinkronizáció is, ilyenkor a sync-üzenet tartalmazza a kiküldési időbélyeget.

- **Delay_Resp**: A mesteróra küldi a Delay_Req üzenetre válaszul, továbbítva benne a kérés érkezésének t_4 időpontját.

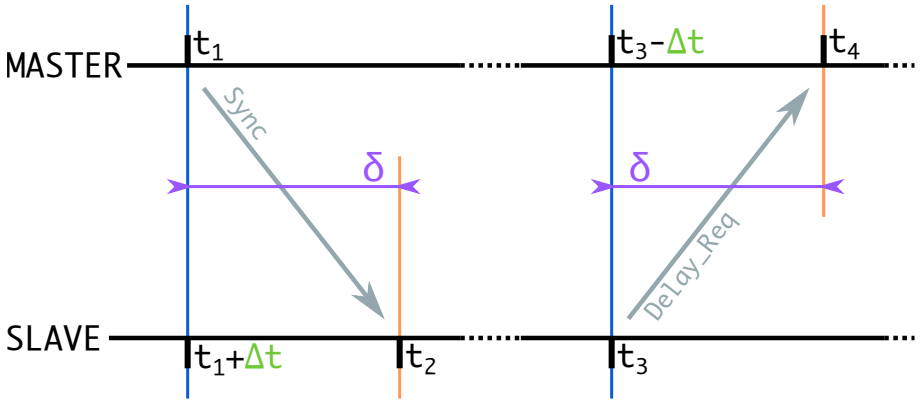
2.1.2. Hibaszámítás

A hibaszámítás eredményéül a master és slave órájának különbségét kívánjuk megkapni a slave óra egységében. A számítás során ki kell küszöbölni a „vezeték” (hálózati infrastruktúra) késleltetését.

A teljes hibaszámítás összefüggésének levezetéséhez induljunk ki a legegyszerűbb esetből, amikor nincs vezetékkésleltetés, az üzenetek a hálózaton nulla idő alatt áthaladnak, pusztán a két – master és slave – óra között van eltérés. Ilyen esetben – felhasználva a 2.1 jelöléseit – a két óra közötti *pillanatnyi* különbség a következőképp számolható:

$$\Delta t = t_2 - t_1, \tag{2.1}$$

hiszen a slave számára t_2 épp a master órája szerinti t_1 pillanat, mivel feltételeztük, hogy nincs késleltetés a hálózaton. Más szavakkal megfogalmazva: Δt megmutatja, hogy a slave órája mennyivel jár előrébb, mint a masteré.



2.2. ábra. PTP-szinkronizáció időbeli lezajlása

A valóságos hálózatokon természetesen nem végtelen gyorsan haladnak át a csomagok, így az előző összefüggésben szereplő mennyiségeket újra kell értelmezni: t_2 időpillanatba a mérés során belekerül a hálózat késleltetése is, t_1 értelmezése változatlan, hisz a master óráját a saját órája szerint mérünk („azonos koordináta-rendszerben” mérünk). A összefüggés a következőképp módosul:

$$\Delta t + \delta = t_2 - t_1, \tag{2.2}$$

ahol δ a hálózat késleltetése. δ az eddigi ismeretek alapján nem kiküszöbölhető. Tekintsük az eddig fel nem használt t_3 és t_4 időpillanatokat! Valójában a két időpillanat különbsége nagyon hasonló értelmű, mint t_2 és t_1 különbsége, csak fordítva: t_4 épp t_3 slave órája szerint mért idő a master órája szerint, megnövelve a hálózat késleltetésével. Különbségük a t_2 és t_1 különbségéhez képest épp fordított értelmű, itt azt vizsgáljuk, hogy a master órája mennyivel jár előrébb, mint a slave-é. Tegyük fel, hogy a hálózat szimmetrikus[12], mindkét irányban ugyanakkora késleltetéssel bír, valamint a `sync`-üzenet kiküldése óta elhanyagolható mértékben csúszott el a slave órája a masterhez képest, ekkor az ismert jelölésekkel:

$$-\Delta t + \delta = t_4 - t_3. \tag{2.3}$$

A két egyenletet *kivonva* egymásból kiejthető a hálózat késleltetése:

$$\underbrace{t_2 - t_1}_{\text{I.}} - (\underbrace{t_4 - t_3}_{\text{II.}}) = 2\Delta t, \tag{2.4}$$

ezért az órákra és a hálózatra megfogalmazott feltételezésekkel a két óra késése:

$$\boldsymbol{\Delta t = \frac{1}{2}\left(t_2 - t_1 - t_4 + t_3\right).} \tag{2.5}$$

2.1.3. A hálózat késleltetése

Bár a szinkronizációhoz szorosan nem kapcsolódik, mégis érdekes lehet tudni, hogy mennyi a hálózat késleltetése. A levezetett összefüggésekből ez is meghatározható a két egyenlet *összeadásával*:

$$t_2 - t_1 + t_4 - t_3 = 2\delta, \tag{2.6}$$

így:

$$\boldsymbol{\delta = \frac{1}{2}\left(t_2 - t_1 + t_4 - t_3\right).} \tag{2.7}$$

2.1.4. Óraszinkronizációt támogató hálózat

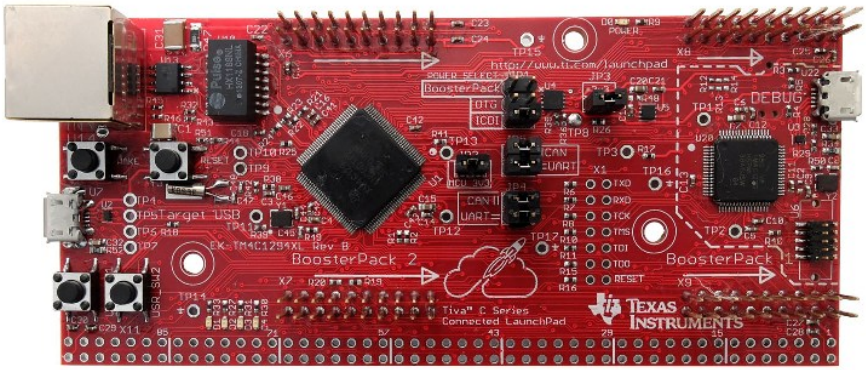
Igazán pontos szinkronizáció eléréséhez nem elég csak a végpontoknak, hanem a hálózatot alkotó berendezéseknek is PTP-kompatibilisnek (pl.: [4]) kell lenniük, fel kell ismerjék a

PTP-csomagokat és azokat fel kell tudják dolgozni. Minden PTP-csomag rendelkezik egy `correction` mezővel, amit a hálózat tölt(het) ki a csomagok továbbítása során: a hálózat a mesteróra és a slave eszköz között „átlátszóvá" tehető, ha minden hálózati eszköz annyival növeli a `correction` mező értékét, amennyi időt a csomag benne eltöltött, később a slave eszközben a mező értékét kivonva a mért késleltetésből egy sokkal pontosabb szinkronizációt érhetünk el. (Ideális esetben, ha minden hálózati eszköz tökéletesen pontosan tudja mérni a saját késleltetését, a végezetül számolt késleltetés valóban pusztán a vezetékek késleltetése lesz.)

Természetesen a szinkronizáció PTP-t nem támogató eszközökből felépített hálózaton is működik, hisz a kommunikáció teljesen szabványos multicast-csomagokon[9] keresztül történik –, csak a szinkronizációs hiba lesz nagyobb az ugyanazon eszközökkel „ideális" hálózatban elérhetőhöz képest. Gyakran csak ilyen „nem-ideális" hálózat áll rendelkezésre, így fontos, hogy a szinkronizáció, ha nagyobb hibával is, de ilyen körülmények között is tudjon működni, illetve sokszor előfordul, hogy az így elérhető pontosság is már elegendő az alkalmazás szempontjából.

3. fejezet

Hardverválasztás



3.1. ábra. Texas Instruments TM4C1294 mikrovezérlővel szerelt Connected LaunchPad

A hardverválasztás során fontosnak tartottam, hogy az minél egyszerűbb felépítésű, minél könnyebben hozzáférhető legyen, mégis még maradjon az eszköznek szabad kapacitása további feladatok ellátására. Választásom a könnyen hozzáférhető, már régebb óta piacon levő, a gyártó részéről jó szoftveres támogatással és dokumentáltsággal bíró TI Tiva TM4C1294 mikrovezérlőre esett, amihez könnyen beszerezhető az EK-TM4C1294XL fejlesztőkártya. A fejlesztőkártyán az Ethernet-illesztés mellett bőséges további szabad csatlakozási lehetőség áll rendelkezésre a később fejlesztések prototípus-tesztelése számára.

A mikrovezérlő amellett, hogy integrált Ethernet-vezérlővel rendelkezik, mely hardveresen képes az Ethernet-csomagok küldési vagy fogadási időbélyeggel történő ellátására, egy igen gyors Cortex-M4F processzorral és elegendően sok program- és RAM-memóriával bír a további feladatok kiszolgálására.

Röviden összefoglalva a mikrovezérlő feladat szempontjából fontos paraméterei az alábbiak[6]:

- **CPU**: ARM Cortex-M4F, max. 120MHz órajelfrekvencia
- S**RAM**: 256kB, közel 2GB/s átviteli sebességgel 120MHz-en
- **Ethernet**: beépített 10/100M vezérlő, hardveres **IEEE 1588 PTP** támogatással

3.1. Hardveróra

A pontos szinkronizáció elengedhetetlen eleme a hardveróra, melyet az Ethernet-vezérlő elér. A csomagok fogadásakor és küldésekor a modul ezt az órát mintavételezi, ennek pillanatnyi értékéből képzi a csomagok időbélyegét. Részletesen később, a szabályozásról szóló fejezetben fogom bemutatni felépítését.

3.2. Megjegyzések a fejlesztőkártyához

A fejlesztőkártya a jól kialakított bővítőlehetőségeinek köszönhetően hatékonyan illeszthető bármely más rendszerhez, egyedül a kártya kapcsolóüzemű tápegysége által keltett nagyfrekvenciás zavaroktól kell megvédeni a kártyához kapcsolódó (analóg) elektronikát.

3.3. Szoftverelemek

A kártya egy FreeRTOS alatt futó, megfelelően konfigurált, speciális driverre felépített lwIP-porttal képes a PTP-csomagok fogadására és azok szoftveres feldolgozásának elősegítésére. Mindkét alkalmazott szoftver rengeteg további szolgáltatással bír (pl.: az lwIP-hoz létezik kész webszerver modul is), így a további funkciók rendszerbe építését nagymértékben elősegítik.

Az szoftverben FreeRTOS alatt a PTP egy nagy prioritású taszkban van megvalósítva, így a rendszeren futó többi taszktól, illetve bármilyen későbbi fejlesztéstől elszigetelten, azokkal párhuzamosan fut.

Az Ethernet-vezérlő „gyári” driverét úgy módosítottam, hogy minden kimenő és bejövő csomagot ellásson időbélyeggel, illetve, hogy fogadja a multicast-üzeneteket.

A nagy erőforrás igényű Berkeley Socket API helyett az implementációt a kis overheaddel rendelkező – ezáltal kevesebb szolgáltatást is nyújtó – callback API-ra építve írtam meg.

Az PTP-csomagok mezőihez illesztve az időkezelést másodperc, nanosecundum egységekkel valósítottam meg, ahol csak lehetett egész-aritmetikát alkalmazva, ügyelve a lebegőpontos számok nem lineáris számábrázolásából fakadó esetleges precízióvesztés elkerülésére. (Természetesen az egész-aritmetika műveletei gyorsabbak is, mint a lebegőpontos műveletek, de a lebegőpontos feldolgozóegység meglétét és sebességét tekintve nem okozott volna szignifikáns sebességcsökkenést használatuk.)

Az ellenőrizhetőség és követhetőség megkönnyítésére a programba számos hibakeresési funkciót beépítettem (pl.: állapotgép követése, belső állapotot tükröző változók kiküldése stb.).

A szoftver hardverfüggő részeit a TI TivaWare[5] könyvtárcsomagjára építettem, így a gyártó hasonló, megfelelő tulajdonságokkal (pl.: beépített Ethernet-vezérlő, PTP-támogatás stb.) rendelkező mikrovezérlőire is fordítható a projekt. A PTP-t megvalósító programrészlet teljes mértékben platformfüggetlen, pár sor átírásával bármely más alkalmas mikrovezérlőre átvihető, pusztán a FreeRTOS szolgáltatásaira támaszkodik.

4. fejezet

Hardveróra és szabályozása

A fejezetben bemutatom a kiválasztott TM4C1294 mikrovezérlő Ethernet-vezérlőjébe épített hardveróra felépítését, részletesen írok az óra szabályzási kérdéseiről, a szabályzási algoritmusokról, a pontosság kérdéséről, mérésekkel alátámasztva.

4.1. A hardveróra felépítése

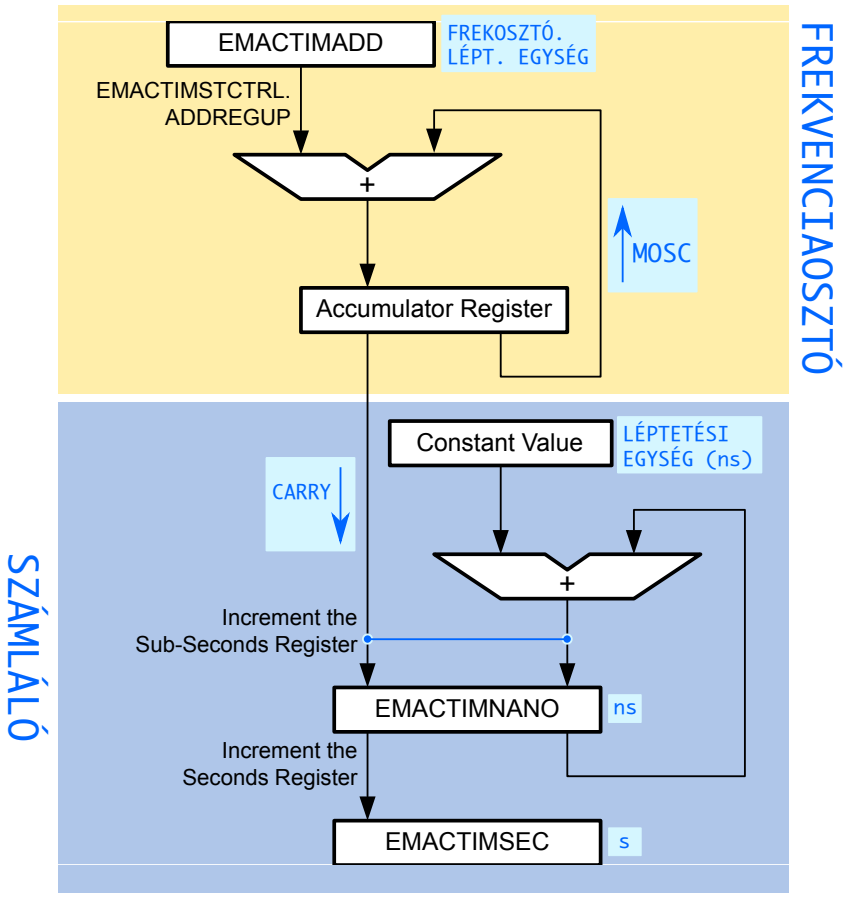
Az hardveróra a 4.1 ábra látható két részből áll: egy frekvenciaosztásért felelős részből és magából a számlálóból – órából, ami az időt tárolja. A működés megértéséhez vegyük sorra az egységeket!

Az óra minden regisztere 32-bit széles, a könnyebb érthetőség miatt továbbiakban külön nem jelzem!

4.1.1. Frekvenciaosztó

A frekvenciaosztó voltaképpen egy visszacsatolt összeadó, mely értéke minden egyes `Main Oscillator`-ciklusban (`MOSC`) az `EMACTIMADD` mennyiséggel, mint léptetési egységgel növekszik. Az akkumulátor-regiszter túlcsordulás (átfordulás) jelzését fogjuk az óra léptetésére használni, mely jelzési gyakoriságát – *frekvenciáját* – a léptetési egység nagyságának beállításával lehet a következő összefüggés alapján beállítani:

$$f_{\text{carry}} = f_{\texttt{MOSC}} \frac{\texttt{EMACTIMADD}}{2^{32}}, \tag{4.1}$$



4.1. ábra. Hardveróra felépítése[6]

ahol $f_{\texttt{MOSC}}$ a `Main Oscillator` frekvenciája, ami az Ethernet-modul által megkövetelt 25MHz[1]. Célszerű minél nagyobb frekvenciát választani, hisz ezzel a gyakorisággal fog előrelépni az óra, más szóval az óra felbontása a frekvenciaosztóból kijövő jelzés periódusideje lesz. A jelzési frekvencia maximális értéke 20MHz lehet, amivel $\frac{1}{f_{\text{carry}}} = 50\text{ns}$ felbontás érhető el. (A minimális frekvencia kb. 4MHz, de a gyártó ajánlja legalább 5MHz frekvenciájú f_{carry} alkalmazását.) Az implementációmban a könnyű hangolhatóság érdekében $f_{\text{carry}} = 12.5\text{MHz}$ (épp `MOSC` frekvenciájának fele) frekvenciát alkalmaztam, mellyel 80ns-os órafelbontás érhető el.

4.1.2. Számláló

Az óra egy, az időt másodperc-nanosecundum formátumban tároló kettős számláló, mely léptetését a frekvenciaosztóból érkező jel végzi. A léptetőjel hatására nanosecundomot tároló számláló a léptetési egységgel növekszik. A léptetési egységgel adjuk meg az óra számára, hogy milyen időközönként – milyen *periódusidővel* – érkeznek a léptetési jelek,

[1] Ez a megkötés csak bekapcsolt Ethernet-modul esetén érvényes, egyébként lehet ettől eltérő érték is.

más szóval, ezzel adjuk meg tick → nanosecundum skálafaktort, azaz az óra felbontását. Amikor EMACTIMNANO regiszter értéke eléri a 10^9-t (0x3B9AC9FF + 1) – azaz eltelik egy másodperc –, a számláló átfordul és ezzel együtt lépteti EMACTIMSEC regiszter értékét. (A nanosecundum számláló körülfordulása átállítható „bináris" üzemmódba is, ilyenkor a számláló 0x7FFFFFFF érték után fordul át, természetesen, ebben az esetben a számláló felbontása nem egy nanosecundum, de az óra felbontása így sem növelhető, mivel az a léptetési egységtől függ, ami legpontosabb esetben sem lehet 50ns-nál kevesebb.)

4.2. Időbélyegek

Az Ethernet-vezérlő – pontosabban a MAC[2] – az időbélyegeket ezen óra csomagfogadás vagy -küldés pillanatában történő mintavételezésével nyeri. Fontos, hogy az óra és a MAC-modul a chipen különböző (aszinkron) clock domainben található, így a mintavételezés kérelmét és magát a mintát a tartományok között mozgatni kell, ami a csomag haladási irányától (küldés vagy fogadás) függően 2-5 MOSC-ciklust vesz igénybe, ami véletlen hibaként hozzáadódik a mért időhöz. Ez az implementációban használt hardverkonfigurációt ($f_{\texttt{MOSC}} = 25\text{MHz}$, $f_{\text{carry}} = 12.5\text{MHz}$) tekintve a szinkronizációs hibát *nagy valószínűséggel*[3] legrosszabb esetben 200ns értékre korlátozza alulról.

4.3. Szinkronjel-kimenet

Általában alkalmazások megkövetelik valamilyen szinkronjel kihozatalát a PTP-slave órából, ezt a célt szolgálja a n-PPS, azaz n-Pulse-per-Second kimenet. A hardveróra a kimeneten egy másodperc alatt előre beállított számú pulzust ad ki saját magát használva időalapnak. Sokszor n értéke 1, azaz csak másodpercenként egy pulzust ad ki az óra, de lehetséges akár több tíz kilohertzes frekvenciájú n-PPS-jel beállítása is. (Természetesen a beállított érték névleges, *tökéletesen pontos* óra esetében egyezik a valós mérhető frekvencia a beállítottal.)

[2] Media Access Controller

[3] Előfordulhat, hogy két különböző, más forrásból származó véletlen hiba semlegesíti egymást.

4.4. A hardveróra szabályzása

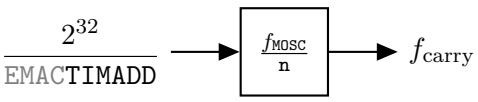
Egy lineáris, időinvariáns – LTI – szakaszhoz jól ismert, könnyen tervezhető és hangolható szabályozási struktúrák léteznek (pl.: PID, állapotteres szabályozó stb.), de nem LTI-rendszerek esetén a szakasz struktúrájának meghatározása, identifikációja is nehéz feladat lehet.

A master-slave időkülönbség mérésének pontatlanságától – mint zajtól – eltekintve alapvetően két komponensből tevődik össze a szinkronizáció hibája:

- a **master és slave órája közötti rendszeres hibából**, ami a slave órájának *egyszeri* módosításával kompenzálható – az óra „beállítható”,
- a **slave oszcillátorának csúszásából** (driftjéből), ami az óra egyszeri „jó” beállításával **nem kompenzálható**, a drift kompenzálásához szabályozóra van szükség. Matematikailag az oszcillátor konstans frekvenciaeltérése az mért idő valós idő szerinti deriváltjának 1-től való eltérésével modellezhető, ezért a kompenzáláshoz olyan szabályozót kell tervezni, ami a slave belső órájának frekvenciát képes hangolni a master-slave időkülönbség mint egyetlen egzaktul mérhető hibajellemző alapján.

A szabályozó megtervezéséhez feltétlen ismerni kell a hardveróra modelljét, ha ugyan az csak nem-időinvariáns, nemlineáris rendszerrel modellezhető, PTP-algoritmus sajátosságai és az óra struktúrája miatt. Vizsgáljuk meg, hogy az óra két nagy része, a frekvenciaosztó és a számláló milyen blokkokkal modellezhető!

4.4.1. Az frekvenciaosztó modellje



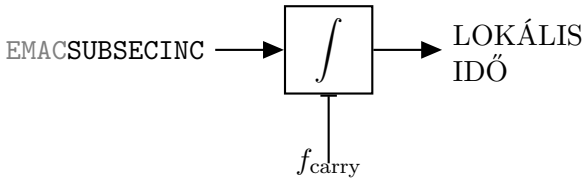
4.2. ábra. Az frekvenciaosztó blokkvázlata

Az osztó egy olyan frekvenciaosztó-blokkal modellezhető, aminek a frekvenciabemenete rögzített $f_{\texttt{MOSC}}$ (= 25MHz), és az osztásaránya (n) változtatható, mely az EMACTIMADD regiszter értékének beállításával az ábrán jelöl módon, ezért:

$$f_{\text{carry}} = \frac{f_{\texttt{MOSC}}}{\frac{2^{32}}{\texttt{EMACTIMADD}}} = f_{\texttt{MOSC}} \frac{\texttt{EMACTIMADD}}{2^{32}}, \tag{4.2}$$

ami épp a korábban bemutatott viselkedés. A blokk a frekvenciaosztás miatt nem lineáris.

4.4.2. Az számláló modellje



4.3. ábra. A számláló blokkvázlata

A számlálót felépítő két időmérő regiszter (EMACTIMNANO és EMACTIMSEC) külön-külön, de némi absztrakcióval együttesen egy diszkrét idejű integrátorral modellezhető. Míg lineáris rendszerekben az integrációs idő állandó, a szabályozó egyik paramétere, és az integrandus, az integrátor bemenő jele változik, addig jelenesetben épp ellenkezőleg, fordítva használjuk az integrátort: az integrációs időt hangoljuk és az integrandus konstans. Kevésbé matematikai nyelven megfogalmazva: a blokkban akkumulált érték – tehát a mért idő – a léptetési értékkel növekszik a hangolható f_{carry} minden periódusában. A blokk ilyenformán se nem lineáris, se nem időinvariáns.

4.4.3. Szakasz- és szabályozóstruktúra

Míg lineáris rendszerek esetében a szabályozó tervezése során törekedni kell a minél jobb (pl.: gyors beállás, kis túllövés stb.) szabályozók tervezésére, addig a vizsgált nem-LTI PTP-óra esetében sok tervezési lépés végre sem hajtható (pl.: nem ejthetők ki pólusok, hiszen az integrátor hangolásával azok mozognak), így olyan *robosztus*, *egyszerű* szabályozó tervezésére kell törekedni, ami *valamilyen szempontból* „jónak” mondható.

Az óra, mint a szabályozni kívánt szakasz egy kétszeresen integráló rendszerrel modellezhető, hisz a frekvenciaosztó valójában egy $\frac{1}{f_{\text{MOSC}}} = T_{\text{MOSC}}$ integrálási idejű, 2^{32}-nál túlcsorduló diszkrét idejű integrátorral közelíthető, aminek a kimenete a számláló integrátorának integrálási idejét határozza meg.

Korábban láttuk, hogy olyan szabályozót kell tervezni a rendszerhez, ami időhiba bemenettel rendelkezik, de az időhiba deriváltjával arányos oszcillátorhibát tudja kompenzálni a két integrátoros szakaszon. A szabályozás megvalósításához diszkrét PD-struktúrát választottam, ami könnyen hangolható, robosztus, és a deriváló taggal az erősen integráló szakasz jól kezelhető. A szabályozó után a frekvenciaosztó elé egy integrátor is bekerült, mely a program megvalósítását könnyíti meg, valójában az integrátor a frekvenciaosztó

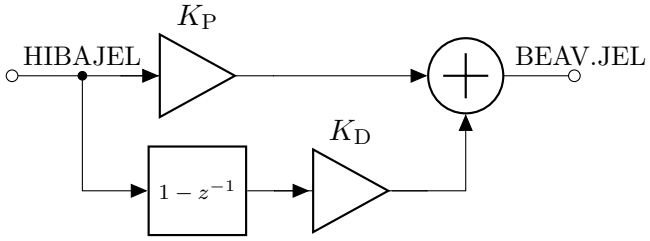
EMACTIMADD regisztere, mely természetes módon nulla frekvencia és időhibánál sem lehet nulla értékű.

4.4.3.1. Szabályozó felépítése

A szabályozó egy némileg módosított PD-szabályozó, diszkrét átviteli függvénye a következő:

$$H(z) = K_\mathrm{P} + K_\mathrm{D}(1 - z^{-1}) \tag{4.3}$$

A PD-szabályozót kiegészítettem egy olyan – nemlineáris – kódrészlettel, mely 1 másodpercnél nagyobb abszolút értékű hibajel esetén egy ciklus erejéig „kiköti" a PD-szabályozót és az órát max. ±1s pontossággal a mesterórához igazítja, ezzel elkerülve a túlcsordulást és gyorsítva a beállást a PD-szabályozóban.



4.4. ábra. Szabályozó blokkvázlata

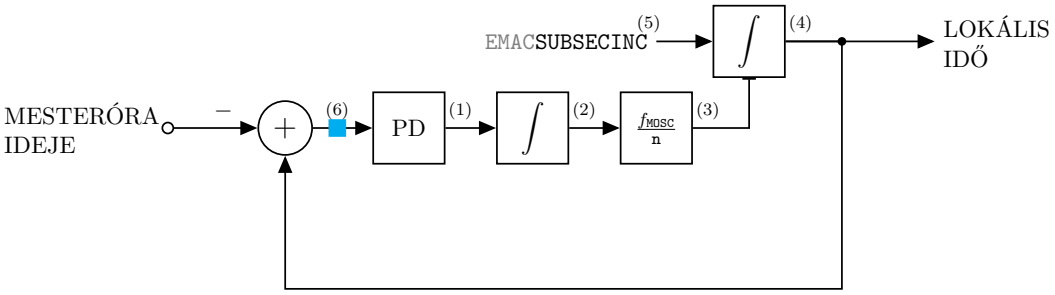
Fontos megjegyezni, hogyha nem ugrik a mesteróra (nem lesz a két óra különbsége 1 másodpercnél nagyobb), akkor a frekvencia korrekciója során bár az idő „sebessége" változik, de szigorúan monoton nő az értéke, nem „ugrik vissza a múltba".

4.4.4. Teljes rendszermodell

A teljes rendszermodell – összeállítva a korábban ismertetett elemekből – a 4.5 ábrán látható.

Ránézésre úgy tűnhet, hogy a szabályozási kör pozitívan van visszacsatolva, valójában a hibajel későbbi értelmezését teszi könnyebbé az előjelek ezen megválasztása, ugyanis az időkülönbség (hibajel) ilyenformán az időhibát a slave szemszögéből jellemzi: ha a hibajel *negatív* előjelű, akkor a slave órája *késik*, ha a hiba *pozitív*, akkor pedig siet a mesterórához képest. A negatív visszacsatoláshoz szükséges előjelváltást a szabályozó végzi el.

A visszacsatolás valójában korántsem ilyen tisztán valósul meg, ahogy az ábra mutatja, természetesen logikailag és matematikailag az ábra helyes, de magát a hibajelet



4.5. ábra. Rendszermodell-blokkvázlat. Jelölések: (1) PD-szabályozó, (2) EMACTIMADD regiszter, mint integrátor, (3) frekvenciaosztó, (4) a számláló integrátora, (5) a számláló léptetési egysége, (6) hibajel

(a 4.5 ábrán (6)-tal jelölve) nem közvetlenül a két óra összevetésével kapjuk, hanem a PTP-algoritmus bemutatása során korábban levezetett összefüggéssel (2.5 egyenlet) a hibát közvetlenül határozzuk meg, majd ezt használjuk a további számításokban.

4.4.5. A szakasz identifikációja

Bár a szakasz nemlineáris és nem időinvariáns, puszta kíváncsiságból tettem egy kísérletet az identifikációjára ötödfokú ARX-rendszerként. Ugyan a számításból matematikailag értelmes eredményt kaptam, a szabályozó hangolásához nem tudtam hatékonyan felhasználni azt. A tömörség megőrzése érdekében az eredményeket jelen dolgozatban nem közöltem.

4.4.6. Diszkrét mintavételezési idő

Egy diszkrét LTI-rendszerben a mintavételezési idő rögzített, az idő folyamán nem változik. Ez a követelmény jelen rendszer esetén nem tartható, még akkor sem a tökéletesen pontos mind a master, mind a slave órája, mivel a PTP-szabvány megköveteli a slave által küldött vezetékkésleltetés-mérő (`Delay_Req`) üzenet kiküldésének időben véletlenszerű eltolását, szabályozni pedig csak a mastertől, az üzenetre kapott (`Delay_Resp`) válasz értelmezése után lehet, hiszen akkor áll rendelkezésre minden információ a hibajel kiszámításához. (Az eltolást azért kell beépíteni a rendszerbe, hogy kiterjedt hálózatok esetén a nagy számú slave eszköz ne egyszerre árassza el kérésekkel a mesterórát.)

Implementációban a véletlen késleltetés értékét a $0 - 500$ms tartományból egyenletes eloszlással generálom, így várható értéke 250ms. A mesterórák *általában* egy másodpercenként küldik ki a pontos időt, így átlagosan két „mintavételi pont” között 1250ms telik el.

A megvalósításban szándékosan nem vettem figyelembe a mintavételi periódusok hosszának lengését, a szabályozó paramétereit úgy kerestem, hogy ezen kompenzáció nélkül is képes legyen a szakaszt megfelelően irányítani.

4.4.7. A szabályozó hangolása

A szabályozó paramétereit heurisztikusan hangoltam különféle célokat – gyorsabb beállás, jobb stabilitást, pontosabb végső beállást – kitűzve.

4.4.8. Kitekintés: a LinuxPTP szabályozója

A LinuxPTP alapértelmezetten PI-szabályozót alkalmaz[3] – mely paraméterei igény szerint hangolhatók –, de lecserélhető más típusú szabályozókra (`linreg, ntpshm, nullf`), ha arra lenne igény (pl.: ha több folyamatnak is hozzá kell férni a referenciaórához, vagy ha szinkron Ethernet-hálózatot (SyncE) használunk).

4.4.9. Kitekintés: egylépéses szabályozás

Elviekben meg lehet valósítani az egylépéses szabályozást egy nemlineáris szabályozóval[6], ha a rendszer késleltetései bizonyos, szigorú követelményeknek megfelelnek (a rendszer teljesen szimmetrikus, a késleltetés ugyanannyi minden csomag esetében stb.), általában ezen feltételek nem teljesülnek, így a jóval bonyolultabb felépítésű szabályozó is csak több lépésben tud konvergálni.

5. fejezet

Mérési eredmények

Ebben a fejezetben bemutatom a különböző szabályozóbeállításokkal elért szinkronizációs pontosságot, mind master-slave, mind slave-slave viszonylatban. A mérési eredmények bemutatása előtt ismertetem a mérési elrendezéseket, a mérések menetét.

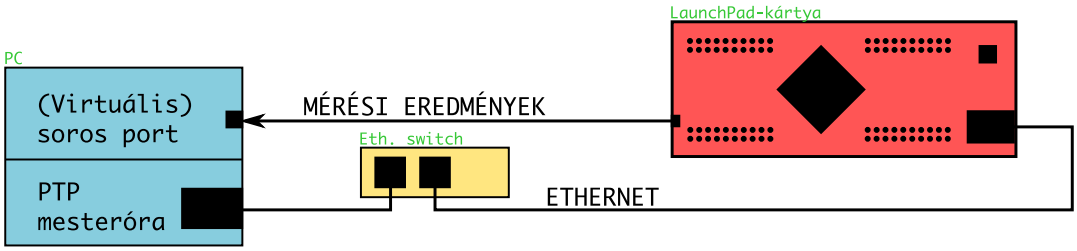
5.1. Mérőeszközök

A mérések során a következő hardveres és szoftveres eszközöket vettem igénybe:

- **Intel 82576** gigabites hálózati vezérlő, hardveres PTP-támogatással bír és beépített PTP-hardverórával rendelkezik. A hálózati kártya a számítógépemben **LinuxPTP**-vel vezérelve a hálózat felé **mesteróraként** működött.
- **HP54645D** digitális oszcilloszkóp, mely 16-bitnyi 400MHz-en mintavételezett digitális bemenettel rendelkezik. Az oszcilloszkóp RS-232 porton keresztül programozható és vezérelhető, amit a mérések Octave-ből történő automatizálása során ki is használtam.
- Egy közönséges 10/100 megabites **Ethernet-switch**, mely nem rendelkezik PTP-támogatással. A szabályozó hangolása során nagyon fontos szempontnak tartottam, hogy a paraméterek úgy legyenek meghatározva, hogy a slave-óra nem ideális körülmények között is jól teljesítsen, ezért a méréseket is így hajtottam végre.
- Kettő **Connected LaunchPad**-kártya, melyek a slave-órákat képezték.

5.2. Mérési metódusok

5.2.1. Master-slave időhiba



5.1. ábra. Mérési elrendezés

Ezen mérések során a szabályozók beállási tulajdonságait (gyorsaság, pontosság) mértem egy mesterórával és egy slave-vel. Az időkülönbséget, azaz a hibajel nagyságát a master-slave a hibajel aktuális, „mintavételi pillanatbeli" értékének soros porton keresztül történő kiolvasásával határoztam meg. Pusztán bemutató jelleggel a slave által minden periódusban kiküldött adatcsomag az alábbi struktúrájú:

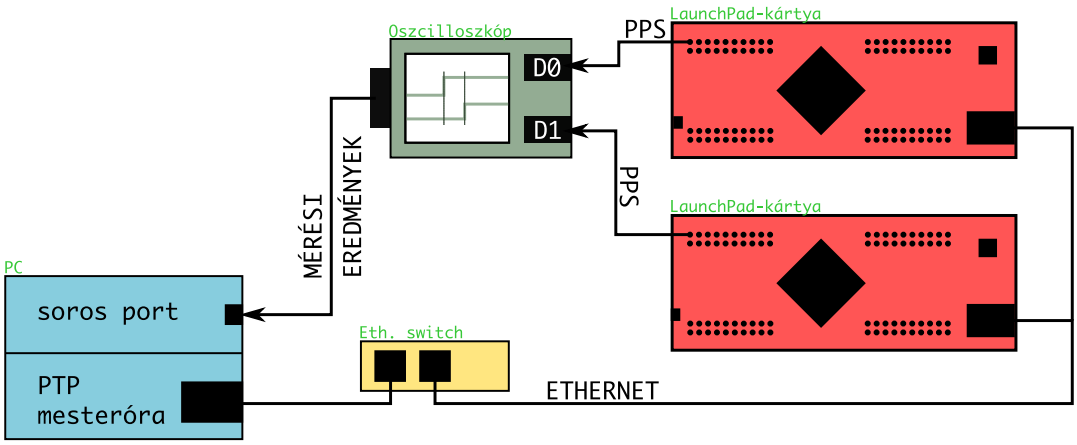
```
master time: 1603556932
dT: 0.-8440
ticks: -105
0x80003700
```

Az adatcsomagban `master time` mezőben foglal helyet a mintavételi pillanatbeli mesteróra-idő, `dT` az abszolút master-slave időkülönbség `másodperc.nanosecundum` formátumban, `ticks` pedig szintén az abszolút időkülönbség a hardveróra f_{carry} frekvenciájú órájának periódusaiban (tick) mérve. Az utolsó hexadecimális szám a frekvenciaosztó léptetési egysége, azaz `EMACTIMADD` regiszter értéke.

Az adatgyűjtés során általában 200-250 mintát vettem (a minták között kb. 1250ms idő telt el), majd annak jellemző részeit – tranziensek, állandósult állapot – ábrázoltam.

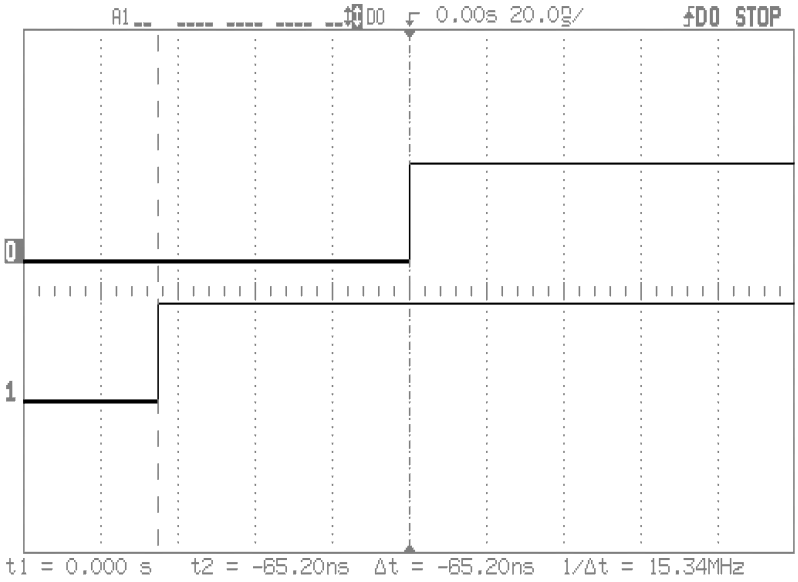
5.2.2. Slave-slave időhiba

A mérések során a két slave óra egymáshoz viszonyított hibáját, azaz az órák együttfutásának hibáját vizsgáltam. Mindkét slave-óra PPS-kimenetét 16384Hz-re programoztam, hogy a mesterórától kapott két üzenet között is bőven legyen lehetőség mérésre. A két különálló slave-óra PPS-kimenetét az oszcilloszkópom digitális bemenetére kötve az időkülönbség mérésére azonnal alkalmas, egyszerű jelformát kaptam. A méréseket automatizáltan, egy Matlab-script segítségével végeztem, mely automatikusan, az oszcilloszkópot vezérelve el-



5.2. ábra. Mérési elrendezés

végezte a kívánt mennyiségű minta begyűjtését, majd hisztogramon ábrázolta a mért értékeket. (Az oszcilloszkópból kiolvasott adatsorok 500 mintát tartalmaznak, 4ns időbeli felbontással.) Szemléltetésképpen az 5.2 ábrán egy mérés jelalakjának oszcilloszkóp-képe látható, melyről a felfutó élek időpontjának különbségéből egyszerűen számítható a két óra eltérése. A mérések során a hisztogramokat 1000 mintából rajzoltam, egy mérés az adattovábbítással együtt kb. 0.5s hosszúságú, így a teljes mérés hossza kb. 8 perc 20 másodperc, mely már elegendően nagy időintervallum, hogy a rendszer hosszútávú működéséről helyes következtetéseket lehessen levonni.



5.3. ábra. Slave-slave óra együttfutás-mérés egy mintájának oszcilloszkóp képe

5.3. Mérési eredmények

Mindkét mérést elvégeztem minden egyes szabályozóparaméter kombinációban, az eredmények értékelése a mérések ábrái után található.

A master-slave időhiba mérését kettő kiindulási állapotból is elvégeztem. Először a rendszert, ezáltal a szabályozót „normál" nagyságú hibajelnek tettem ki: a szabályozóalgoritmus tárgyalásakor bemutatott módon a szabályozó 1 másodpercnél nagyobb abszolút értékű hibajel esetén ugrasztja a hardverórát, a következő korrekciós időpontban a hiba a lokális óra két mintavételi pont közötti csúszásából, illetve az órabeállítási parancs érvényre jutása és az időpont fogadása közötti különbségből fog állni, ami egy egyébként működőképes rendszert tekintve nem adódik túl nagy értékűnek, általában max. 100ns nagyságrendű. Az idődiagramokon ezeket a méréseket narancssárga színnel jelöltem.

Másodszor a szabályozót a lehetséges maximális hibajel nagyságrendjébe eső, azaz 10^8ns nagyságú időhibájú helyzetből indítottam. A beállás ez esetben természetesen jóval lassabb, hiszen az első méréshez képest *hat nagyságrenddel* nagyobb hibajelet kell ledolgoznia a szabályozónak. A grafikonokon ezeket a méréseket kékkel jelöltem.

Jelen precíziós alkalmazásban a nem használhatjuk a szabályozástechnikában elfogadott konvenciót, mely szerint a rendszer a végérték 2%-os környezetében már az állandósult állapotát elértnek tekinthető, hisz ez a ténylegesen elérni kívánt pontosságnál (pl.: egy nagy zavarimpulzusra adott válasz esetén) nagyságrendekkel durvább állapotot jelent. A mérések során úgy tekintettem a rendszerre, hogy az elérte az állandósult állapotát, ha a végérték körüli lengése kisebb, mint 500ns. A rendszer állandósult állapotbeli viselkedését, mivel a hibajel nagysága ilyenkor nagyságrendekkel kisebb, mint a tranziensekre adott válaszokban külön grafikonon ábrázoltam.

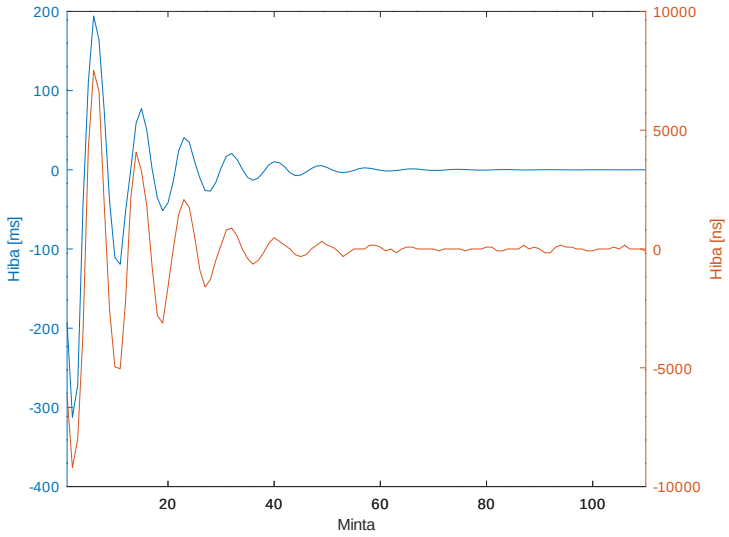
Az állandósult állapotbeli stabilitás jól jellemezhető a hibajel, mint valószínűségi változó sűrűségfüggvényével, illetve annak statisztikai paramétereivel, a várható értékkel és a szórással. A sűrűségfüggvény jelen mérések esetén nem ismert, viszont a paraméterek számításokkal a diszkrét mintákból (közelítőleg) meghatározhatók, illetve a sűrűségfüggvény közelíthető a mintákból rajzolt hisztogrammal. Hisztogramon ábrázoltam az állandósult állapot 100 elemét, illetve az elemeket felhasználva meghatároztam a statisztikai jellemzőket.

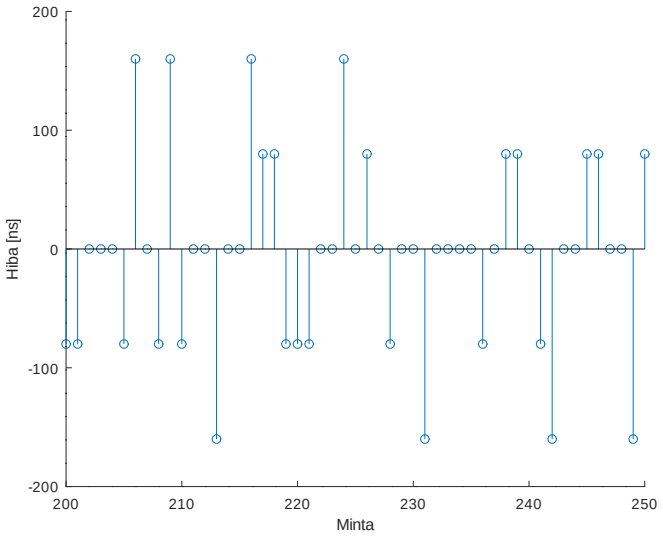
A slave-slave együttfutás mérésekor a hisztogram felrajzolása mellett a későbbi könnyebb összehasonlítás érdekében szintén kiszámoltam a statisztikai jellemzőket.

A mérések során a slave-órák (mérőkártyák) egymáshoz közel helyezkedtek el, azonos termikus és zavarkörülményeket biztosítandó.

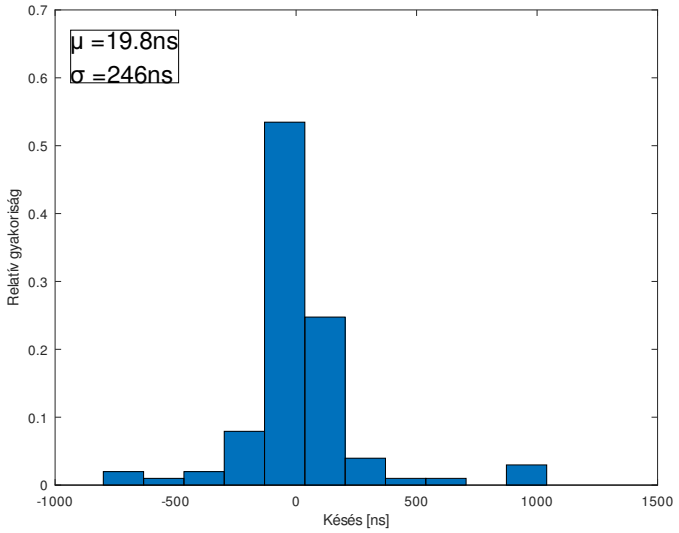
5.3.1. Részletes mérési eredmények

A mérési eredményeket a következő oldalakon mutatom be.

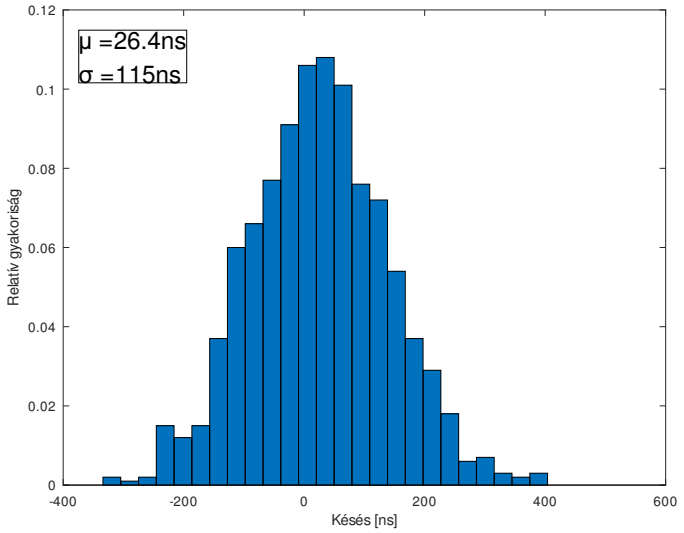




(a) Master-slave időhiba

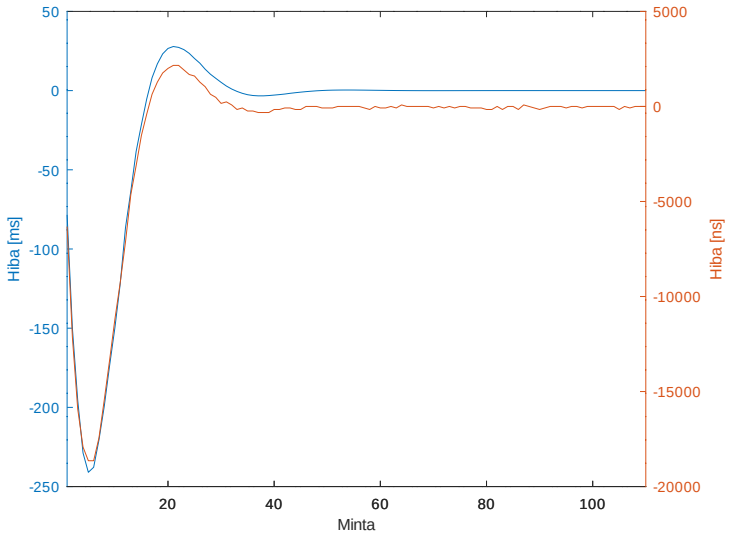


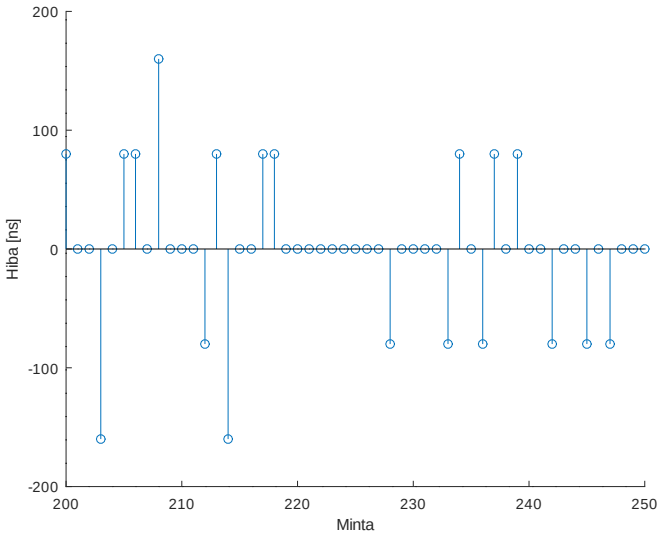
(b) Master-slave időhiba hisztogramja állandósult állapotban



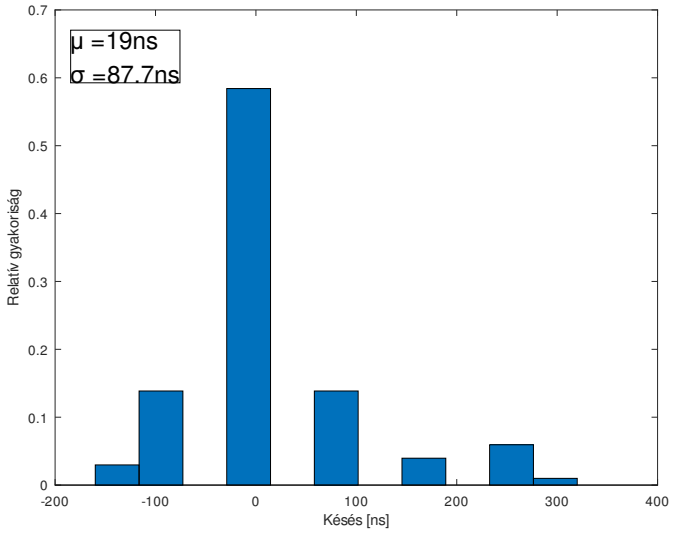
(c) Slave-slave időhiba hisztogram

5.4. ábra. $K_\mathrm{p} = 1.0, K_\mathrm{d} = 0.5$, kezdeti beállítások a rendszer első élesztésekor.

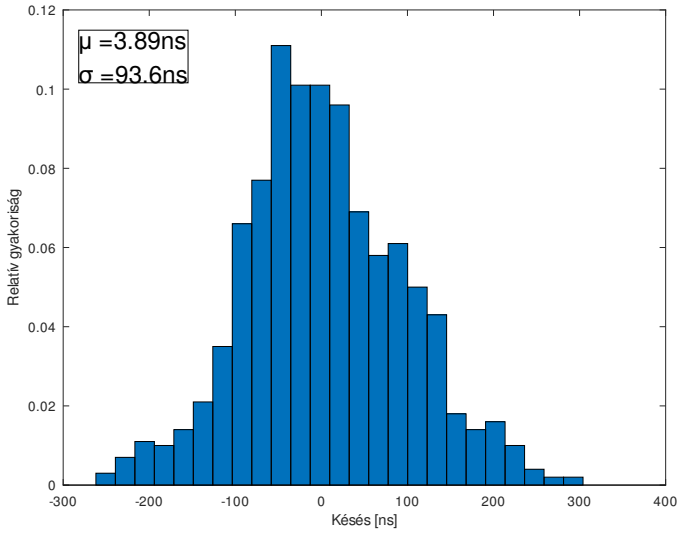




(a) Master-slave időhiba

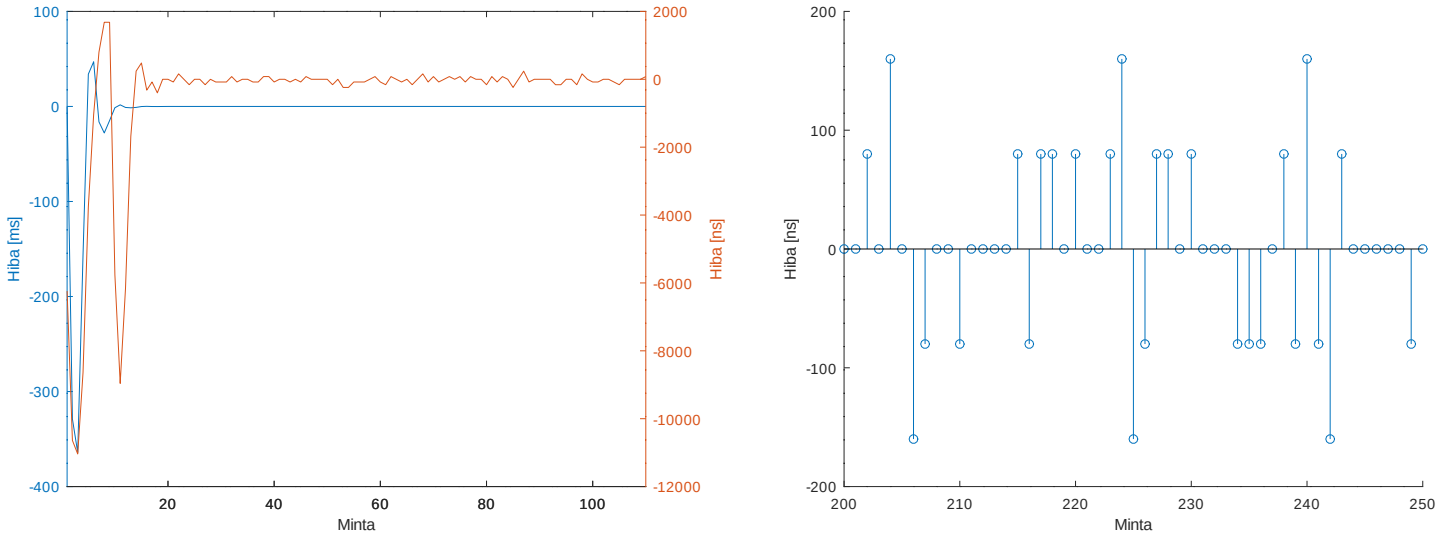


(b) Master-slave időhiba hisztogramja állandósult állapotban

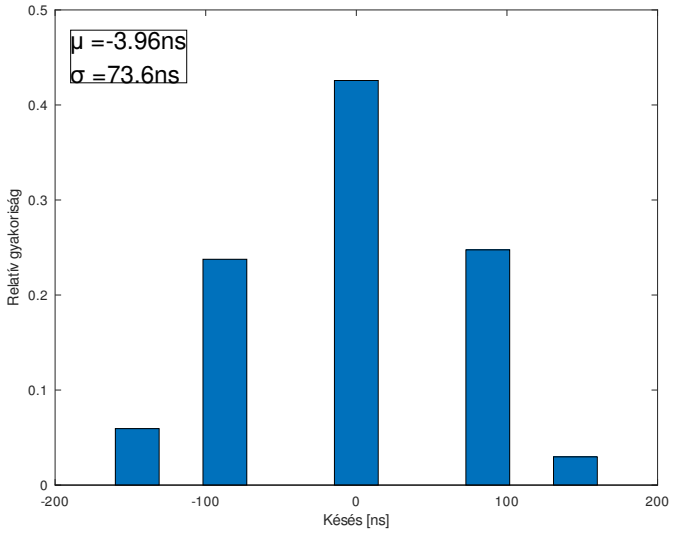


(c) Slave-slave időhiba hisztogram

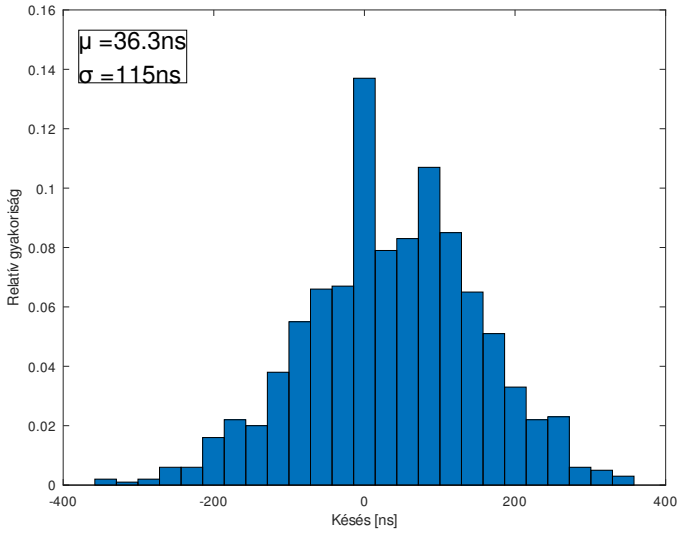
5.5. ábra. $K_\mathrm{p} = 0.1, K_\mathrm{d} = 0.5$



(a) Master-slave időhiba

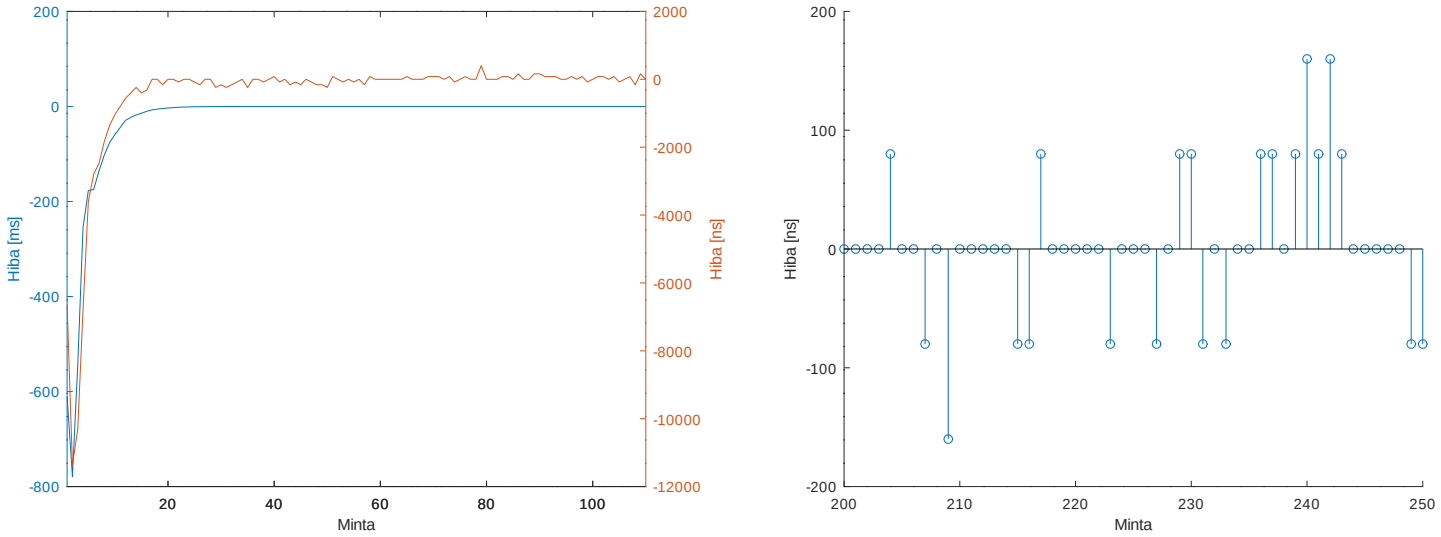


(b) Master-slave időhiba hisztogramja állandósult állapotban

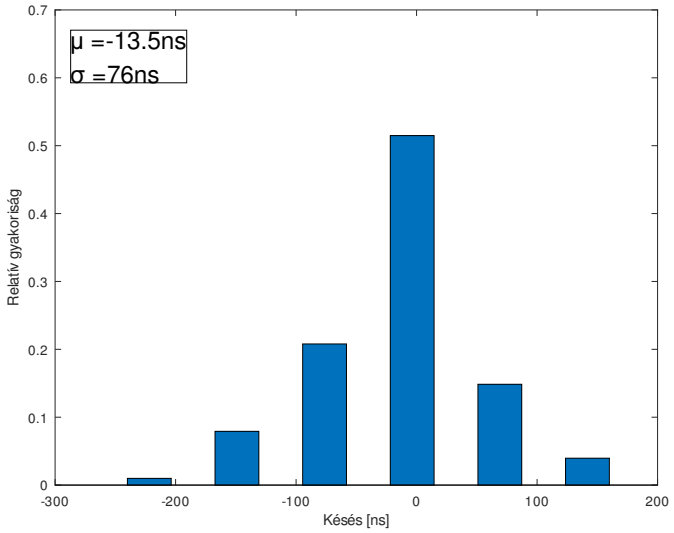


(c) Slave-slave időhiba hisztogram

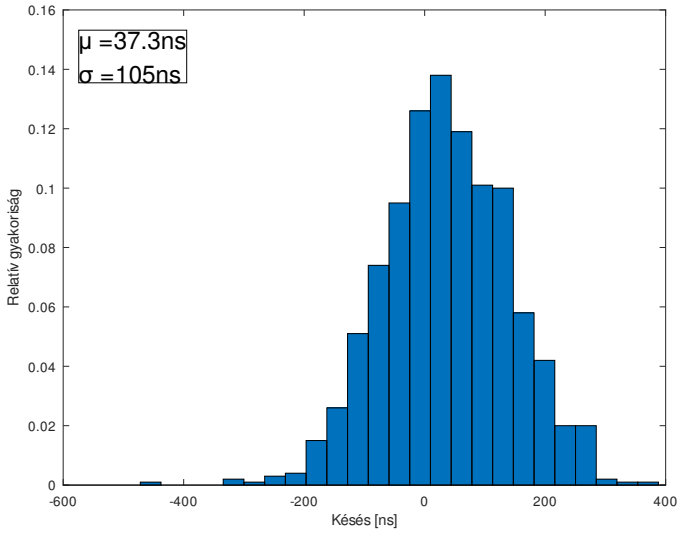
5.6. ábra. $K_\mathrm{p} = 1.0, K_\mathrm{d} = 2.0$



(a) Master-slave időhiba

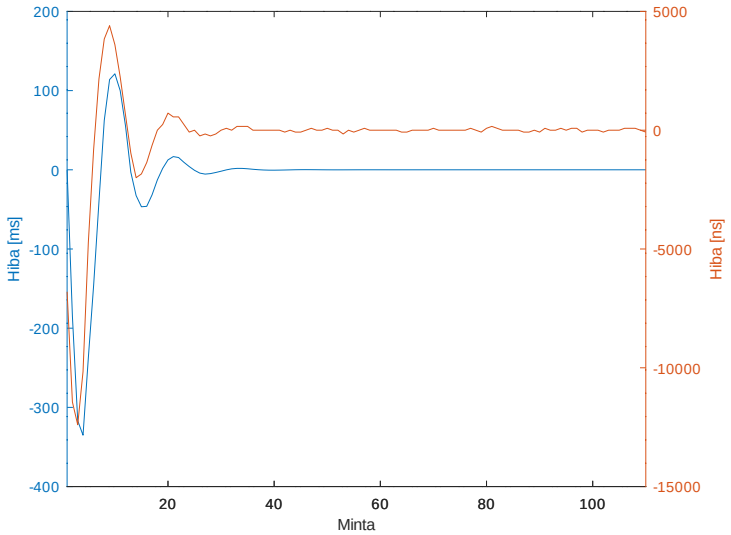


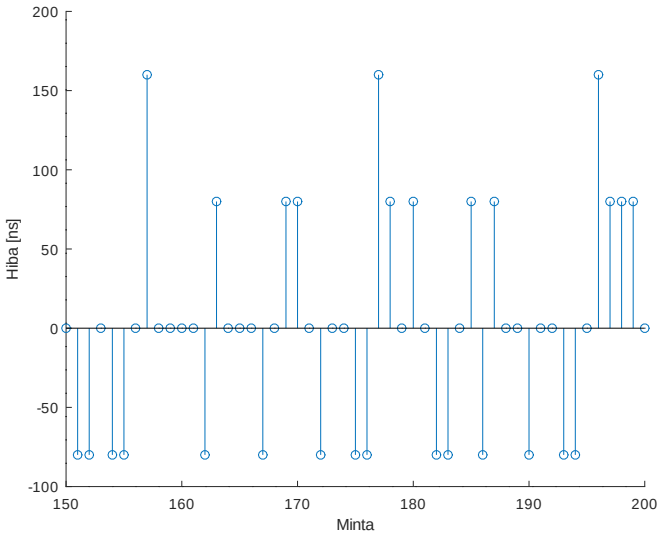
(b) Master-slave időhiba hisztogramja állandósult állapotban



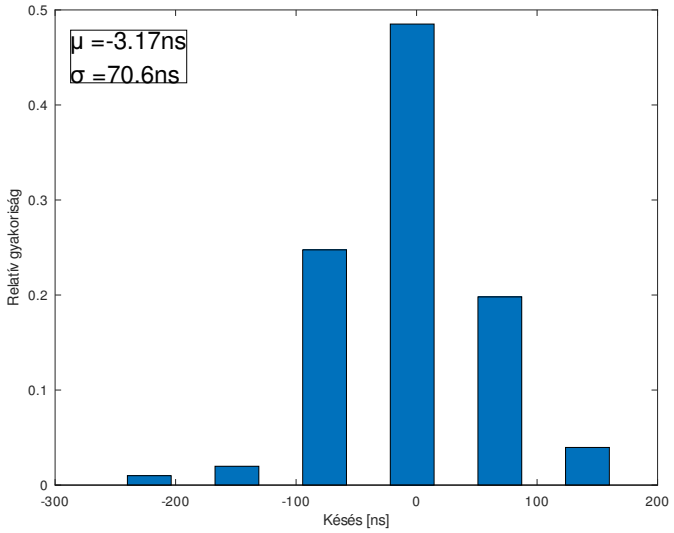
(c) Slave-slave időhiba hisztogram

5.7. ábra. $K_\mathrm{p} = 0.5, K_\mathrm{d} = 2.0$

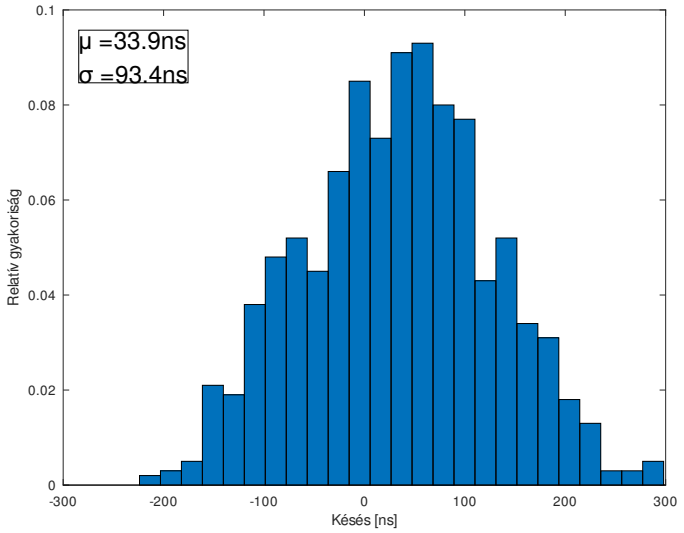




(a) Master-slave időhiba

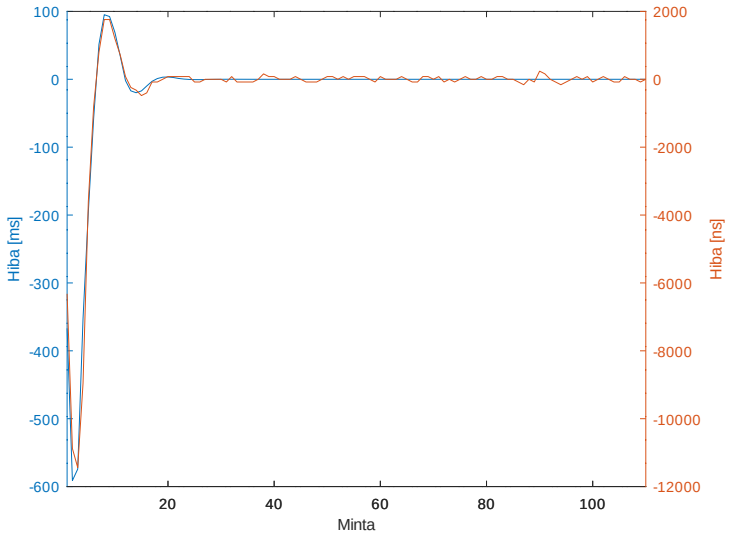


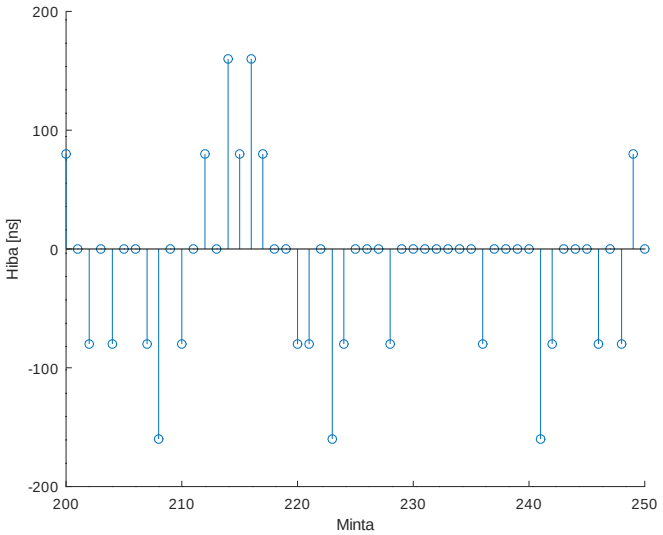
(b) Master-slave időhiba hisztogramja állandósult állapotban



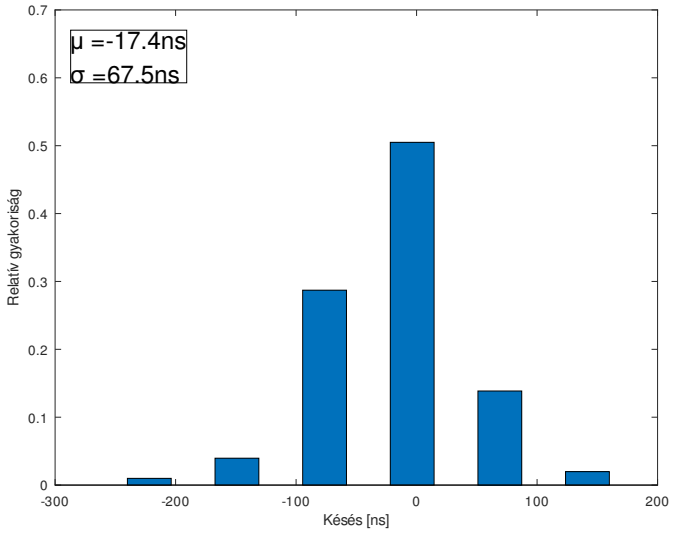
(c) Slave-slave időhiba hisztogram

5.8. ábra. $K_\mathrm{p} = 0.5, K_\mathrm{d} = 0.7$

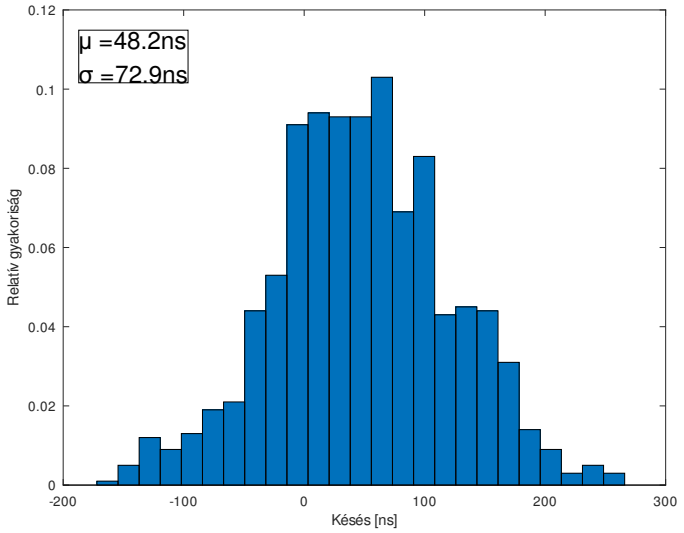




(a) Master-slave időhiba



(b) Master-slave időhiba hisztogramja állandósult állapotban



(c) Slave-slave időhiba hisztogram

5.9. ábra. $K_\mathrm{p} = 0.5, K_\mathrm{d} = 1.0$

5.4. Eredmények értékelése

	PARAMÉTEREK		MASTER-SLAVE JELL.			SLAVE-SLAVE JELL.		
#	K_p	K_d	μ_{ms}[ns]	σ_{ms}[ns]	**beállási idő [lépés]**	μ_{ss}[ns]	σ_{ss}[ns]	**mérések**
1	1.0	0.5	19.8	246	≈ 60	26.4	115	5.4 ábra
2	0.1	0.5	19	87.7	≈ 50	28.19	112.89	5.5 ábra
3	1.0	2.0	-3.96	73.6	≈ 20	36.3	115	5.6 ábra
4	0.5	2.0	-13.5	76	≈ 20	37.3	105	5.7 ábra
5	0.5	0.7	-3.17	70.6	≈ 40	33.9	93.4	5.8 ábra
6	0.5	1.0	-17.4	67.5	≈ 20	48.2	72.9	5.9 ábra

5.1. táblázat. Mérési eredmények összefoglalása, adott kategória szerinti „legjobb” érték kiemelve

A mérési eredmények is azt sugallják, amit a grafikonokon is láthattunk: nem lehet egyértelműen kinevezni egy beállítást sem „legjobbnak”, optimálisnak. (Mivel egy nemlineáris rendszert irányít egy lineáris szabályozó, ezért ez egyáltalán nem meglepő, „optimális” szabályozó tervezésére nem is törekedtem.) Az kapott eredmények alapján három különféle célt fogalmazhatunk meg, ami szerint „elég jóvá” tehető a rendszer viselkedése:

- a szabályozót hangolhatjuk a rendszer gyorsítására, **minél kisebb beállási idő elérésére**. Ilyenkor úgy kell megválasztani az együtthatókat, hogy a rendszer gyorsuljon, de stabil maradjon még nagyon nagy hibajelek esetén is. K_d növelésével tudjuk gyorsítani a rendszert, K_p megfelelő beállításával pedig a stabilitást kell megőrizni. A legfőbb nehézséget rendszer lineáris paramétereinek (pólusainak) az irányítás hatására való megváltozása okozza, a megváltozott rendszernek is stabilnak kell maradnia. A vizsgált paraméterkombinációk közül a leggyorsabb beállást talán a **3**-as szabályozó biztosítja, viszont a rendszer állandósult állapotbeli slave-slave együttfutás szórása igen nagy, hiszen a szabályozó nagy lépésekben hangolja az órát. A **4**-es rendszerben megfigyelhető, hogy a szabályozó nem hagyja lengeni a rendszert, a rendszer kvázi egytárolós tagként viselkedik, mely magától értetődően szebbé teszi a rendszerbe bejutó tranziensek lecsengését.

- A szabályozót hangolhatjuk továbbá az átlagos master-slave hiba (μ_{ms}) minimalizálására, azaz arra, hogy *átlagban* a slave minél jobban kövesse a mesterórát. Ilyen szabályozók alkalmazásával automatikusan a slave órák együttfutása is javul, hiszen a minden slave ugyanazt a mesterórát követi nagy pontossággal. Ilyen szempontból az **5**-ös szabályozó a „legjobb" a kombinációk közül, viszont a beállási idő ezzel a szabályozóval a leggyorsabb rendszerhez képest kétszeres.

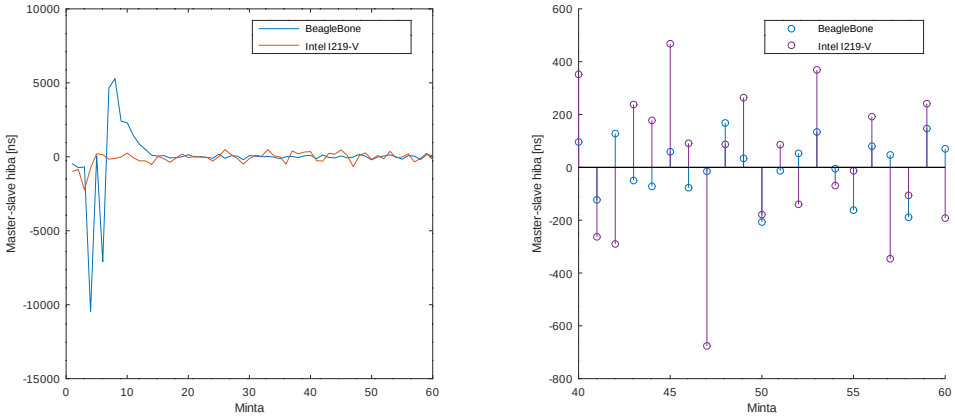
- A szabályozóval javíthatjuk kifejezetten az órák együttfutását. A táblázatból jól látszik, hogy egy, a mesterórát pontosan követő szabályozó egyáltalán nem biztos, hogy az együttfutásra nézve hasonlóan jó tulajdonságokkal bír és fordítva, de még az együttfutás szórása (σ_{ss}) sem feltétlen hozható kapcsolatba az együttfutás átlagos hibájával (μ_{ss}), hisz épp a legnagyobb szórású rendszernek (**1**) a legjobb ezen tulajdonsága. Az átlagos együttfutási ofszethiba minimalizálása helyett minimalizálható az együttfutás szórása is.

A master-slave időhiba szórását (σ_{ms}) tekintve az **1**-es rendszert leszámítva minden rendszer egy nagyságrendben van, az eltérések minimálisak.

Eddig önmagukban, abszolút skálán vizsgáltam az eredményeket, nem vettem figyelembe sem a hardver (például clock-domain crossing-ból fakadó) korlátait, sem nem mértem hozzá már meglévő, és általánosan jónak elfogadott PTP-implementációkhoz a rendszerem tulajdonságait.

Viszonyítási alapul a bizonyítottam jól működő LinuxPTP-t választottam, mely követési és statisztikai tulajdonságait két különböző (PTP-t hardveresen támogató) eszközön lemértem, egy számítógépen egy Intel I219-V hálózati kártyával és egy BeagleBone Black fejlesztőkártyán. Mindkét eszközön a tranzienseket beleszámolva 60 mintát vettem (kb. másodpercenként egyet, a `Sync` üzenetek ütemeznek), majd az állandósult állapot 40 értékéből meghatároztam a statisztikai jellemzőket. (A minták a LinuxPTP visszajelzéseiből származnak, nem PPS jelek mérésével nyertem őket.) Az idődiagramok az 5.10 ábrán láthatók. Mindkét eszköz nagyjából 15-20 lépésből áll be, a BeagleBone esetében jól megfigyelhető a tranziens. A számított statisztikai paraméterek az alábbiak:

	MASTER-SLAVE JELL.	
	μ_{ms}[ns]	σ_{ms}[ns]
BeagleBone	2.951	100.48
Intel I219-V	23.024	273.14



5.10. ábra. LinuxPTP követési jellemzői BeagleBone-on és Intel I219-V hálózati kártyán mérve

Az eredmények alapján jól látszik, hogy a BeagleBone valóban az adott feladatra valamilyen szinten fel van készítve, míg a másik esetben pusztán tényleg csak minimális hardveres támogatásról van szó. (Természetesen ettől még abszolút skálán mérve az I219-V is nagyon pontos!)

Ezek alapján bátran kimondható, hogy a LaunchPad-re készített implementációm megfelelő szabályozóbeállításokkal képes a LinuxPTP tulajdonságaival megegyező pontossági szintet elérni vagy akár – a master-slave szórás tekintetében meg is haladni –, beállási sebességben pedig közel ugyanúgy teljesíteni, ami egy, a LinuxPTP számára rendelkezésre álló erőforrásokhoz képest jóval egyszerűbb rendszeren véleményem szerint komoly teljesítmény.

5.5. Összegzés

Az implementációm, a kezdeti elvárásoknak megfelelően, mérésekkel alátámasztva valóban képes a pontos óraszinkronizációra, mégpedig egy egyszerű és alacsony költségű hardveren futtatva, a LinuxPTP-jéhez hasonló, eredményeket felmutatva PTP-t nem támogató hálózati környezetben egy jól megtervezett robosztus szabályozóval. A szinkronizációs pontosság akár megahertzes frekvenciájú mintavételező rendszerek szinkronizációjához is elegendő.

A mérések során kiderült, hogy mind a mesterórához, mind egymáshoz képest pontosan szinkronizálhatók a slave-órák, ami egy kiterjedt mérőrendszer létrehozásának alapfeltétele. A rendszer a szoftveres struktúrája miatt könnyen fejleszthető.

5.6. Fejlesztési, felhasználási lehetőségek

A rendszer egy elosztott, beágyazott mérőrendszerbe (pl.: elosztott oszcilloszkópok), mint „szinkronizációs építőkocka” beépíthető, mely az óraszinkronizáció mellett egyben az adattovábbítást is képes lekezelni.

Mivel szoftveresen a rendszer meglehetősen hardverfüggetlen, ezért bármely más, PTP-t támogató mikrokontrolleres rendszerre könnyen portolható. (Ilyenek például a másik ismert gyártó, az STMicroelectronics STM32 sorozatába tartozó egyes Cortex-M4F és újabb mikrovezérlői, melyekhez bőségesen állnak rendelkezésre fejlesztőkártyák.)

A szoftver könnyen kiegészíthető egy monitorozó rendszerrel (pl.: webes felület), mely segítségével a szinkronizáltság állapota távolról bármikor lekérdezhető.

5.6.1. Oszcillátorhiba mérése

A szinkronizáció során szinkron állapotban mérhetővé válik az oszcillátor hibája, hisz épp annak hibáját kompenzáljuk a hardveróra hangolásával. Az így nyert adattal hosszú távon mérhetővé válik az oszcillátor hőmérsékletfüggése és egyéb, a környezet megváltozása okozta zavar.

Köszönetnyilvánítás

A dolgozatban ismertetett eredmények a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar Balatonfüredi Hallgatói Kutatócsoport szakmai közössége keretében jöttek létre a régió gazdasági fejlődésének elősegítése érdekében. Az eredmények létrehozása során figyelembe vettük a balatonfüredi központú Rendszertudományi Innovációs Klaszter által megfogalmazott célkitűzéseket, valamint a párhuzamosan megvalósuló EFOP 4.2.1-16-2017-00021 pályázat támogatásával elnyert „BME Balatonfüredi Tudáscentrum" térségfejlesztési terveit.

Irodalomjegyzék

[1] *AN-1728 IEEE 1588 Precision Time Protocol Time Synchronization Performance.* https://www.ti.com/lit/an/snla098a/snla098a.pdf.

[2] *ICSS PTP 1588 Developer Guide.* https://processors.wiki.ti.com/index.php/ICSS_PTP_1588_Developer_Guide.

[3] *LinuxPTP - clock servo.* https://www.mankier.com/8/ptp4l.

[4] *Precision Time Protocol Software Configuration Guide for IE 4000, IE 4010, and IE 5000 Switches.* https://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie4000/software/release/15-2_4_e/b_ptp_ie4k.html.

[5] *TivaWare™ Peripheral Driver Library.* http://software-dl.ti.com/tiva-c/SW-TM4C/latest/exports/SW-TM4C-DRL-UG-2.1.4.178.pdf.

[6] *Tiva™ TM4C1294NCPDT Microcontroller Data Sheet.* http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf.

[7] R. Cochran – C. Marinescu: Design and implementation of a ptp clock infrastructure for the linux kernel. In *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication* (konferenciaanyag). 2010, 116–121. p.

[8] Grzegorz Daniluk – Tomasz Wlostowski: White rabbit: Sub-nanosecond synchronization for embedded systems. In *Proceedings of the 43rd Annual Precise Time and Time Interval Systems and Applications Meeting* (konferenciaanyag). 2011, 45–60. p.

[9] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008. July., 1–300. p.

[10] T. Kovácsházy: Towards a quantization based accuracy and precision characterization of packet-based time synchronization. In *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (konferenciaanyag). 2016, 1–6. p.

[11] T. Kovácsházy – T. Tusori – D. Vincze: Prototype implementation and performance of time-based distributed scheduling on linux for real-time cyber-physical systems. In *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)* (konferenciaanyag). 2018, 1–6. p.

[12] Paul Krzyzanowski: Precision time protocol - clock synchronization that computes latency and offset. `https://www.cs.rutgers.edu/~pxk/417/notes/ptp.html`.

[13] K. Lao – G. Yan: Implementation and analysis of ieee 1588 ptp daemon based on embedded system. In *2020 39th Chinese Control Conference (CCC)* (konferenciaanyag). 2020, 4377–4382. p.