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

Precíziós óraszinkronizáció és alkalmazása vezeték nélküli szenzorhálózatban

TDK dolgozat

Készítette:

Badó Dávid
Somos Gergő
Várallyay Sámuel

Konzulens:

dr. Kovácsházy Tamás
Vörös András

2016

Tartalomjegyzék

1. fejezet

Bevezetés

Az elmúlt évek technológiai fejlődése olcsó rádiós chipek és szenzorok megjelenését tette lehetővé. Ezeket kombinálva olyan vezeték nélküli szenzorhálózatokat hozhatunk létre, amelyekkel közelről figyelhetünk meg fizikai-jelenségeket. Az ilyen szenzorokat kényelmesen használhatjuk közvetlen környezetünkben, vagy éppen kitelepíthetjük távoli, nehezen elérhető területekre. Mivel nem helyhez kötöttek, továbbá ma már alacsony fogyasztással rendelkeznek, így mozgó objektumokhoz is lehet őket rögzíteni. A szenzorhálózatok alkalmazását széles körben igénylik például egészségügyi, katonai, ipari, közlekedési vagy tudományos területeken.[1]

Ezekben az elosztott rendszerekben gyakran szükség van adatfúzióra, ahol a különböző szenzorokból érkező minták együttes feldolgozásával kapjuk meg a kívánt információt. Például egy földrengés megfigyelő rendszerben különböző szenzorok különböző időpontban érzékelik a rezgést, amiből következtetni lehet a rengéshullám irányára. Mint az előbbi példa alkalmazás is, sok más felhasználás elengedhetetlenné teszi a kauzalitási viszonyok, valamint a mintavételezés időpontjának minél pontosabb ismeretét. Ha beavatkozók is jelen vannak a rendszerben, szükség lehet azok pontos szinkronizálására, ütemezésére. Ezen okokból többnyire szükséges a precíziós óraszinkronizáció, amely biztosítja a rendszerbeli – elosztottan működő – komponensek óráinak együtt járását, amire jó példát jelentenek napjaink okos gyárai, ahol a robotok kollaboráció során készítik el a termékeket. Az elmúlt évtizedekben vezetékes hálózatokra kiforrott protokollok alakultak ki. Elsősorban Ethernet esetén, kidolgozásra került az IEEE 1588 szabvány[2], amely segítségével jóval mikroszekundum alatti hibával szinkronizálhatók az eszközök órái, megfelelő hardver támogatás esetén. Azonban vezeték nélküli esetben új problémák vetődnek fel, és a használt eszközök is nagyon különbözőek így nem lehet egy az egyben alkalmazni a vezetékes protokollokat. Vezeték nélküli esetben alacsonyabb az elérhető sávszélesség és a hálózat megbízhatatlanabb, könnyen elveszhet üzenet. Jellemzően szempont az alacsony teljesítmény felvétel hiszen izolált működés esetén véges az energiaforrás. Ennek érdekében általában kisebb a számítási kapacitás is. Az hogy mely tulajdonságokra optimalizálunk, erősen alkalmazás függő. Általánosságban elmondható hogy a fogyasztás kárára bármely tulajdonság javítható.

Munkánk során egy olyan beágyazott rendszerekben alkalmazható vezeték nélküli kommunikációt és óraszinkronizációt is támogató megoldást dolgoztunk ki, ami megközelíti az IEEE 1588 szabvány által elérhető pontosságot. Ezt a használt hardver funkcióinak újszerű felhasználásával értük el javítva az időbélyegek rögzítésének a pontosságát és egyben lehetővé téve a mérési adatok valós idejű továbbítását. Megoldásunk előnye, hogy egy olcsó, általános célú rádiós hardveren fut, amelyet könnyű alacsony fogyasztásra optimalizálni.

A dolgozat felépítése a következő. A 2. fejezetben az óraszinkronizáció jelentőségét, már megvalósított algoritmusokat és a mérési eszköz működési elvét és alaptulajdonságait mutatjuk be A 3. fejezet a hardverkörnyezetről és az óraszinkronizációhoz leginkább használt perifériákról fog szólni. A 4. fejezetben a használt kommunikációs protokollokat és a megvalósított óraszinkronizációs algoritmus mutatjuk be. Az 5. fejezetben kiválasztjuk és bemutatjuk a használni kívánt szenzort, majd az illesztés módját és a mikrokontrollerrel való kommunikációját ismertetjük. A 6. fejezetben mérésekkel prezentáljuk az óraszinkronizáció pontosságát és a választott szenzorokkal bemutatjuk fontosságát. A 7. fejezetben összefoglaljuk az eredményeket és kitekintést adunk a jövőbeli teendőkre.

2. fejezet

Háttérismeretek

Ebben a fejezetben a piacon található, általunk használt eszközök jelentőségét, működési elvét és alap tulajdonságait mutatjuk be, és az implementált protokollunk alapjául szolgáló, már meglévő szinkronizáció és kommunikáció módjait taglaljuk.

2.1. Óraszinkronizáció

Az alfejezetben bemutatjuk a szinkronizáció gyakorlati szerepét és annak felhasználási módjait, majd bővebben kifejtjük a jelenleg implementált protokollok elvi működését, alkalmazását.

2.1.1. Jelentősége

Elosztott szenzorrendszerben, a folyamatok megfigyeléséhez, szenzor adatok feldolgozását függetlenné kell tenni az egyes csomópontok információ átfutási idejétől, küldési késedelemtől. Ennek a problémának megoldására találták ki az időbélyegeket, melyek a szenzortól lekérdezett adat beérkezésének pontos idejét tartalmazza. Ezek érvényességéért a maguk csomópontok felelősek, így minden egységnek tudnia kell, hogy aktuálisan mennyi a pontos idő. Amennyiben képesek szinkronizálni az óráikat, úgy az adatok fúzióját, feldolgozását, már nem szükséges azonnal elvégezni, ugyanis az időbélyegek segítségével rekonstruálható a teljes folyamat eseménylánca.

2.1.2. Szinkronizációs protokollok és algoritmusok

Korábban kidolgozott protokollok elméleti háttere sok segítséget nyújtott a saját, vezeték nélküli verzió megalkotásához. A következőkben javarészt azokat a protokollokat mutatjuk be, melyek elősegítették a kutatást, vagy a közszférában jelentősen elterjedtek.

Óraszinkronizációs protokollok vezetékes hálózatban

Jelenleg kiforrott szabványok javarészt vezetékes hálózaton implementáltak, ám többnyire ezen protokollok elméleti alapjait használtuk fel a vezeték nélküli implementációnkhoz. Közhasználatban levő szabványok közé tartozik a Network Time Protocol (NTP), mely 1981 óta az internetre csatlakozott számítógépek óraszinkronizációjáért felelős. Máig használják, fejlesztik, milliszekundumos pontosságot tudnak elérni, közönséges hardver segítségével [10]. Az IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (egyszerűen Precision Time Protocol, továbbiakban PTP) az előbbihez hasonlóan, vezetékes hálózatokban kidolgozásra került

szabvány, amely segítségével jóval mikroszekundum alatti hibával szinkronizálhatók az eszközök órái. Ez a protokoll megkövetel bizonyos hardver támogatást[9], így egyelőre csak a CERN-ben, GSI nehéz ion kutatóközpontban és KM3NeT neutrínó teleszkópban használják.

Ezeknek a protokolloknak a szinkronizációs hálózati architektúrája eltérő. Az NTP rétegekből áll, maximum 256-ból és szerver-kliens kapcsolatot valósít meg. A legalacsonyabb (0.) szinten atomórák találhatók, melyek közvetlen összeköttetésben vannak az első rétegben található szerverekkel, így internetre közvetlenül nem csatlakoznak. Egy szinkronizálás során a kliens három vagy több, egymástól eltérő hálózaton levő – eggyel alacsonyabb rétegben található – szervert kérdez le, és mindegyikhez képest kiszámolja a saját órájának eltérését. Eztán egy algoritmussal eldönti mely három szervertől kapta a legpontosabb értékeket, és ez alapján szinkronizálódik. PTP esetében saját, Best Master Clock (BMC) algoritmussal választja ki a neki megfelelő szervert, és ennek az órájával egyezteti a sajátját. Ezt az algoritmust a továbbiakban nem részletezzük, ugyanis esetünkben egyetlen master implementált, így annak kiválasztása egyértelmű.

A szinkronizáció, a két protokoll esetében hasonló elven alapszik. A slave (vagy kliens) küld egy kérést a master-nek(vagy szervernek), majd ezután a master küld választ. Az üzenetváltás során mind a fogadás, mind a kiküldés időpontját elmentette mindkét fél, és a master azokat válaszüzenetben továbbította a slave felé. Az üzenetváltás után már meghatározható az órák közötti eltérés, és az átfutási idő is, amennyiben mindkét irányban azonosnak vesszük. Így szinkronizálni lehet az egyes slave-eket és a master-t. Az órajelet generáló oszcillátorok frekvenciája környezeti és gyártási tényezőktől is függ, így a gyakorlatban nincsen két egyenlő frekvenciájú órajel generátor. Ebből kifolyólag a szinkronizált órák közötti differencia idővel nőni fog, egyre pontatlanabbá válik a rendszer. Ennek kompenzálására dolgozták ki a PTP esetében a master által időnként küldött szinkronizáló üzenetet, mellyel a késleltetés és a saját órájának ismeretében az aktuális eltérés nagyságát meg tudja mondani bármely slave (master órájához képest).

Óraszinkronizációs protokollok vezeték nélküli hálózatban

Vezeték nélküli hálózat alkalmazása további nehézségeket okoz. Egyetlen kiküldött üzeneten négy effektus rontja a pontosságot, így a küldési idő, a csatorna hozzáférési idő, az átviteli késleltetés és a beérkezési idő. Küldési idő alatt az üzenet összeállítása és kiküldési ideje közti különbséget értjük (szabad csatorna esetén). A csatorna hozzáférési idő a csatorna foglaltsága miatt jelentkezhet, változó nagyságú, protokollonként eltérő korlátokkal rendelkezhet. Az átviteli késleltetés, az üzenet kiküldése és fogadása közt eltelt idő, a többi késleltetéshez képest relatíve kis intervallum. Az üzenet feldogozásának ideje a beérkezési idő, mely üzenettípusonként változhat. Minden késleltetés kiküszöbölése rengeteg erőforrást emészt fel, redundanciával jár. Ezért alkalmazásfüggő az egyes protokollok használata.

A világon a legszélesebb körben elterjedt vezeték nélküli hálózat a WiFi, mely Reference Broadcast Syncronisation (RBS) protokollt használ. Ez egy sajátos megoldással állt elő, mivel a protokollok többsége küldő-fogadó típusú szinkronizációt valósít meg, itt fogadó-fogadó elven működik az órák egyeztetése. Ezt úgy kell elképzelni, hogy a router – melyre a szinkronizálandó egységek kapcsolódnak – egy bizonyos, időinformáció nélküli jelet sugároz ki időnként. Ezt minden vevő érzékeli, és innentől annak beérkezési időpontja referenciaként szolgál a további szinkronizáció során. Az órák egyeztetése már csak a hálózat végpontjai között megy végbe, összehasonlítják egymás óráit és meghatározzák a relatív fázis eltolódást. Ez az időzítés így a referencia jel megérkezésének fogadási időpontjától függ. Ennek az az előnye, hogy a küldő pontatlanságát megkerülve szinkronizálja az

órákat, így hiba csak az átviteli késleltetésnél és a beérkezési időnél jelentkezik, habár az előbbi általában elhanyagolhatóan kicsi.

A Timing-sync Protocol for Sensor Networks (TPSN) egy megközelítése a vezeték nélküli óraszinkronizáció megvalósításának. Ez a protokoll vevő-fogadó típusú, ám az NTP felépítésére hasonlít. Fa struktúrát valósít meg, melyben a gyökér lehet bármely, de csak egyetlen eszköz, ez a 0. szint. Ennek kiválasztása után hálózatfelderítés segítségével keresi meg a további szinteken található eszközöket. Algoritmusa egyszerű, amiket közvetlen elér a gyökér, azok az eszközök lesznek az első szinten, a második szintűeket pedig az első szintről lehet közvetlenül elérni, és így tovább. Hasonlóan, a szinkronizációs protokolljának alapja is az NTP, ám ebben az esetben a gyökér kezdeményezi a szinkronizációt, majd a feljebbi szinten található egységek véletlen időközönként próbálják felvenni a kapcsolatot a gyökérrel, a további időinformációkat lekérni. Szinkronizálás után az első szintű egységek kezdeményeznek a második szinten található csomópontok irányába kommunikációt és így tovább, míg minden eszköz órája együtt nem jár. Ennek a protokollnak az előnye, hogy minimalizálja a küldési időt úgy, hogy a csomópontok közti túl nagy távolság áthidalása már nem okoz problémát. Habár a küldési idő késleltetése által okozott pontatlanság nincs kiküszöbölve, alacsony szintű időbélyegzés segítségével ez az algoritmus kétszer olyan precíz mint amit az RBS el tud érni.

A TPSN szinkronizációjához hasonló protokoll a Flooding Time Synchronization Protocol (FTSP). Ennek megvalósítása során igyekeztek kiküszöbölni a TSPN hátrányait, fejleszteni a pontosságot. Hasonlóan egyetlen gyökér csomóponttal rendelkezik, ám ebben az esetben minden egység ehhez csatlakozik, master - slave architektúrát kialakítva ezzel. Szinkronizáció során a gyökér elküld egy üzenetet, mely tartalmazza a globális időt, így a csomópontok a beérkezés idejéből és a kapott időbélyegből meg tudják becsülni az eltérését az óráiknak. Frekvenciakompenzáláshoz lineáris regressziót alkalmaznak, ezzel érve el nagyobb pontosságot és csökkentve a csatornafoglaltságot. Ám ez a protokoll az átviteli késést nem tudja kiküszöbölni, és a gyökér hatósugara korlátot jelent az alkalmazás fizikai méretében [14].

2.2. Mérési eszközök

A dolgozatunkban nem csak egy óraszinkronizációs protokollt fejlesztettünk, hanem egy demonstrációs rendszert ahol inerciális mérőegységek használatával prezentáljuk ennek helyes működését. Választásunk azért esett erre a fajta szenzorra, mert a mozgáskövető szenzorok használata az évek során egyre elterjedtebb és a jövőben is várható az alkalmazási területek bővülése. Olyan alkalmazási területekben van rájuk szükség, mint a navigációs rendszerek, a játékok, a virtuális és kevert valóság alkalmazások, amelyek a következő években egyre népszerűbbek lesznek. Mindenesetre felhasználásuk felhasználói felületek, sport alkalmazások és a képstabilizációs eljárások fejlesztésénél is előtérbe kerülhet. Ezeken a felhasználási területeken további elvárás lehet, hogy valósidejű adatokat kapjunk, amelyeket a lehető legjobb pontossággal tudjunk meghatározni, hogy mikor keletkezett az adat. A példaalkalmazásban egy ilyen időkritikus rendszert próbálunk modellezni, amelyben a szenzorhálózaton futó időszinkronizációval viszonylag pontosan meg tudjuk határozni a szenzoradatok kauzális viszonyait.

2.2.1. Mozgás nyomon követése

A példaalkalmazásban bizonyos mozgások nyomon követését tűztük ki célul, viszont ahhoz, hogy képesek legyünk megfigyelni őket, tudnunk kell, hogy milyen szenzorok használatával lehet ezt megtenni. Pár mozgásfajta bemutatásával megállapítjuk, hogy milyen szenzorral vagyunk képesek detektálni ezeket.

A gyorsulást, egy gyorsulásmérővel elég egyszerűen meg tudjuk határozni. Egy másik fajta mozgás a vibráció. Erre a mozgásfajtára tekinthetünk úgy mint egy olyan mozgásra, amely periodikusan, gyorsan váltakozó gyorsulásból és lassulásból áll. Az ütődés a gyorsulásnak egy egyszeri, impulzus jellegű nem periodikus fellépése. Dőlés, mint mozgásfajta egy lassabb mozgást jelent, ami így nem is a tárgy gyorsulásából hanem inkább a nehézségi gyorsulás a tengelyekhez viszonyított megváltozásáról ad információt. Az előző mozgásfajták felismerését mind meg tudjuk tenni gyorsulásmérő alkalmazásával viszont az elfordulás elemzésénél már jobban el kell gondolkodnunk, hogy milyen szenzorral tudjuk ezt a mozgásfajtát detektálni. Belátható, hogyha egy gyorsulásmérő szenzort azon tengelye körül forgatjuk, amelyen a gravitációs erő hat, akkor nem kapunk információt az elfordulásáról, ebből következően főleg az elfordulás méréséhez szükségünk van más fizikai tulajdonságot mérő szenzorra. A szögsebességmérő tengelyei mentén a szögsebességet méri, így nagyban segít, hogy a gyorsulásmérővel együtt viszonylag pontos orientációs adatokat kapjunk a szenzorok mérési adataiból. Ezekből a példákból jól látszik, hogy a gyorsulásmérő és a szögsebességmérő olyan adatokat szolgáltatnak, amelyek nagyban segíthetnek a mozgásfajták felismerésében és nyomon követésében. Ezeket az adatokat feldolgozva, fuzionálva képesek vagyunk a fizikai valóságban történt mozgásesemények detektálására és követésére [13].

2.2.2. Mozgásérzékelő szenzorok

Napjainkban MEMS technológiával készített digitális szenzorok egyre elterjedtebbek, kis méretük, kis fogyasztásuk és alacsony áruk miatt. Több ilyen szenzort magában foglaló inerciális elven működő mérőrendszert Inertal Measurement Unitnak nevezi a szakirodalom. Ezen rendszer két leggyakoribb eleme a gyorsulásmérő és a szögsebességmérő illetve még mágneses iránytűt és nyomásmérőt tartalmazhat. Az IMU-k az érzékelt fizikai mennyiségek (gyorsulás, szögsebesség, mágneses térerősség) alapján képesek egy adott tárgy mozgásadatainak saját inerciarendszerében való meghatározására, így alkalmazásuk leginkább a navigációs rendszerekben terjedt el. Ahhoz, hogy használni tudjuk őket, nem árt, hogyha működési elvükkel és alaptulajdonságaikkal tisztában vagyunk. Ezért a következőkben a projektben használt gyorsulásmérő és szögsebességmérőn kívül még egyéb a későbbiekben használható MEMS szenzorok alaptulajdonságait mutatjuk be.

MEMS Gyorsulásmérő

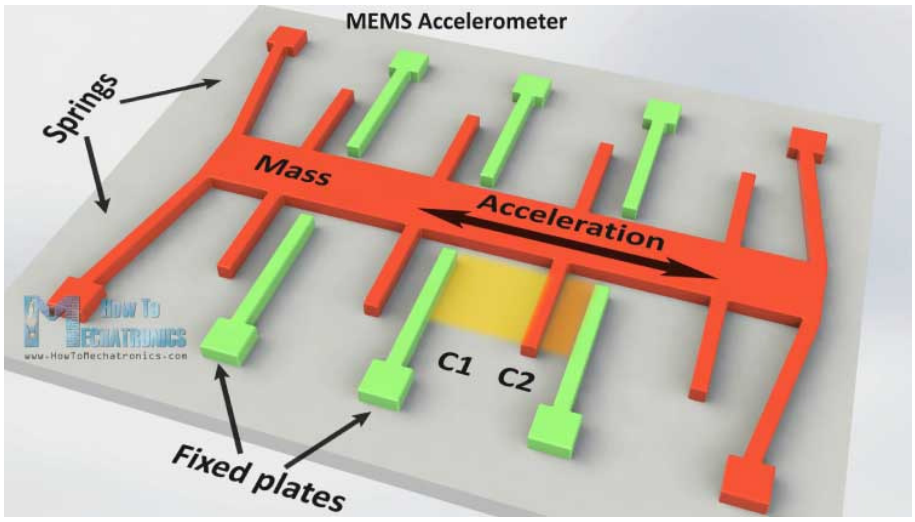
A gyorsulásmérés elve Newton második törvényén alapul.

$$\vec{F} = m \cdot \vec{a}, \tag{2.1}$$

ahol $\vec{F}$ a testre ható erő; m a test tömege; $\vec{a}$ a test gyorsulása.

A mérés történhet kapacitív, induktív, piezoelektromos és piezorezisztív módon is. A 2.1 ábrán a kapacitív elven alapuló gyorsulásmérés látható. A fellépő erő hatására a szilíciumlapkában lévő anyag alakváltozást szenved, így mérve a kapacitás változását arányosságot tudunk vonni az erő és a gyorsulás között.

A mérési és a működési elveken kívül még számos más tulajdonságokkal tudunk jellemezni egy gyorsulásmérőt. Az egyik ilyen, hogy mely irányokban tudja a szenzor érzékelni a gyorsulást. Kettő illetve három tengelyen is meg tudjuk határozni a gyorsulást hiszen több ilyen tulajdonságú szilicíumlapkát egymással merőleges irányokba helyezve több szabadságfokú gyorsulásmérőt kaphatunk. Egy másik tulajdonsága a mérési tartomány. Ezt természetesen felhasználás függő, ezért elérhetőek olyan szenzorok amelyek csak 1-2g-ig, míg más alkalmazások számára kaphatóak, amelyek több száz g méréshatárig is képes mérni. További fontos tulajdonság, hogy milyen felbontással és milyen érzékenységgel va-



2.1. ábra. MEMS gyorsulásmérő kapacitív elvű működése [11]

gyunk képesek mérni. Ezt a tulajdonságot természetesen a szenzor analóg digitális átalakítója korlátozza. A piacon elérhetőek 6 bites felbontástól egészen a 32 bites felbontásig az ezekhez tartozó érzékenységek pedig a mérési tartománytól is függenek, így típusonként nagyon változóak lehetnek.

A MEMS gyorsulásmérőnek előnyei mellett számos hátrányai vannak, amelyeket szintén figyelembe kell vennünk. A kimenetük hőmérséklet függő és zajjal terhelt. Ezek a hibákat erősítési és ofszet hibákként tudjuk modellezni, amelyek általában nem állandóak, így kompenzálásukat többször el kell végezni. Ezen kívül még állandó erősítési és nullpont hibát okozhatnak a szenzor egyéb tulajdonságai is. Előfordulhat hogy a tengelyek nem teljesen merőlegesek egymásra, így áthallás figyelhető meg a méréseik között. Az ilyen gyártási hibákat általában az adott szenzor adatlapján feltüntetik illetve gyárilag kalibrálják [12].

MEMS Szögsebességmérő

A mérési elve a Coriolis-erő és szögsebesség közötti arányosságon alapul.

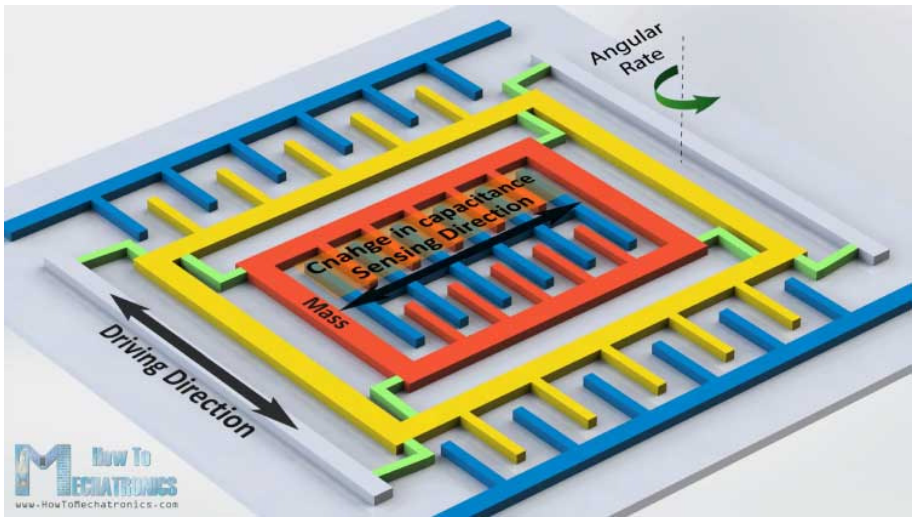
$$\vec{F_c} = -2 \cdot m \cdot \vec{\omega} \times \vec{v}, \tag{2.2}$$

ahol $\vec{F_c}$ a testre ható Coriolis-erő; m a test tömege; $\vec{\omega}$ a test szögsebessége; $\vec{v}$ a test sebessége.

A szögsebességmérés történhet kapacitív és piezorezisztív módon is. A 2.2 ábrán az induktív elven történő mérés látható. A rezgő tömeget körülvevő test elfordulás hatására a rezgéssel merőleges erőt fejt ki, amelynek következtében a rezgő tömeg elmozdul. Mérve a kapacitás megváltozását könnyen meg tudjuk határozni a szögsebességet.

Ugyanúgy, mint a gyorsulásmérőnél, hogyha a szögsebességmérőből is többet helyezünk egymásra merőleges irányokban, akkor mindhárom tengelyen meg tudjuk mérni a szögsebességet. A szögsebességmérő szenzorok méréshatárai ± 50 °/s-től ± 2000 °/s-ig terjednek. A felbontást itt is az analóg digitális átalakító határozza meg, a legkisebb a 12 bites, a legnagyobb pedig 24 bites felbontás. Az adott szenzor érzékenységét nagyban meghatározza a méréshatár és a felbontás, így minden egyes konstrukciónál különböző érzékenység figyelhető meg.

Ahogy a gyorsulásmérőknél megállapítottuk, a szögsebességmérő kimenete is hőmérséklet függő és zajjal terhelt. A legnagyobb hibát az ofszet hiba okozza, amely nem állandó,



2.2. ábra. MEMS szögsebességmérő kapacitív elvű működése [11]

így kompenzálása időnként szükséges. Ennél kisebb hibát okoz az erősítési hiba, amely a tengelyek mentén eltérő lehet, így kompenzálásához forgatni kell az eszközt. A legkisebb hibát a tengelyek ortogonalitási hibája okozza [12].

Egyéb szenzorok

A gyorsulásmérőn és a szögsebességmérőn után a leggyakrabban használt MEMS alapú szenzorok a digitális iránytű és a nyomásmérő. Ezek közül egyiket sem használjuk, így csak minimálisan térek ki a tulajdonságaikra.

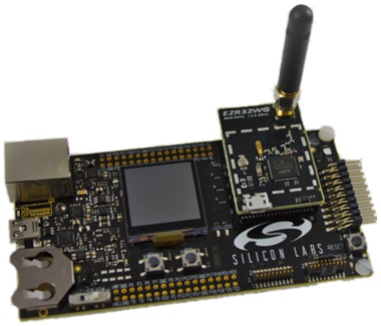
A mágneses térerősségmérő avagy digitális iránytű a Hall effektus vagy magnetorezisztív effektus alapján tudják mérni a mágneses térerősséget. Ugyanúgy mint az előző szenzoroknál, itt is lehet mindhárom tengely mentén mérni az egymásra merőleges szenzorok segítségével. Nagy hátrányuk, hogy zajjal terhelt kimenetüket nehéz kalibrálni zajos környezetben.

A nyomásmérő a membrán elmozdulását és deformációját kapacitív, induktív, piezorezisztív és piezoelektromos módon is tudja mérni, így ebből az információból le tudja vezetni a mérhető nyomást is.

3. fejezet

Hardverkörnyezet

Az időszinkronizáció megvalósításahoz és a méréshez az EZR32WG nevű rádiós csipet használtuk, ami egy MCUt és egy EZradioPro (Si446x) rádiós adóvevőt foglal magában. A fejlesztést a 3.1-es ábrán látható Silicon Labs-os WSTK6220 (Wireles Starter Kit) segítségével végezzük, ami a rádiós kártya mellett egy alaplapot is tartalmaz hibakeresési lehetőséggel, forraszthető kivezetésekkel, egyszerű perifériákkal és szenzorokkal.



3.1. ábra. WSTK6220 eszköz.

3.1. Rádiós adó-vevő

A rádiós csipet 26 MHz-es kristály oszcillátor hajtja meg, így az üzenetek beérkezésének és kiküldésének az idejét maximum kb. 40 ns-os felbontással tudjuk megállapítani. Üzeneteket rádiófrekvenciás sávban küldünk, ehhez biztosítja az eszköz a következő modulációkat: OOK-t, és négy féle FSK-t. A küldés frekvenciasávja 142-1050 MHz között változtatható, ám mi ISM sáv környékén, pontosabban az SRD860 sávban 868 MHz-en használjuk. Teszteléskor az általunk alkalmazott moduláció 2FSK, ezzel 500 kbaud-os szimbólumsebességet érhetünk el, így 500 kbit/s a maximális adatátviteli sebesség. Ez a szinkronizációnk pontosságához mérve lassú, hiszen átszámolva, egy bitet minimum két mikroszekundum alatt küldünk ki, ezzel szemben pedig nanoszekundumokban próbáljuk meghatározni a késéseket. A rádió a mikrokontrollerrel tokon belül SPI-on keresztül kommunikál, valamint megszakítást is tud generálni. A csip rendelkezik két GPIO lábbal amiket különböző funkciókra lehet beállítani. Ezeken keresztül lehet kivezetni a csomag küldés illetve fogadás jeleit, amikkel pontos időbélyegzést tudtunk megvalósítani.

3.2. Mikrokontroller

Ebben az alfejezetben a mikrokontroller főbb elemeit mutatjuk be, kiemelve az általunk használt perifériákat. Egyes elemek az óraszinkronizációban kiemelten fontos szerepet játszanak.

Mag

A mikrokontroller processzora ARM Cortex M4 és 32 kByte RAM-mal és 256 kByte flash memóriával rendelkezik.

Oszcillátorok

Egy szinkronizációs alkalmazásban az elérhető oszcillátorok ismerete nagyon fontos. A processzort négy kristályoszcillátor hajthatja meg, két alacsony 32.768 kHz-es LFXO és LFRCO vagy a maximum 28 MHz-es HFRCO és a legmagasabb frekvenciájú HFXO. Ez utóbbi órajele 48 MHz-es, így maximum 21 ns-os időbeli felbontással tudunk időt mérni, de sajnos a rádiós csip ennél lassabb. Az eszköz rendelkezik további két oszcillátorral, ám ezek a rádió működését segítik, akár küldésre vagy modulációra használja a csip.

Számlálók

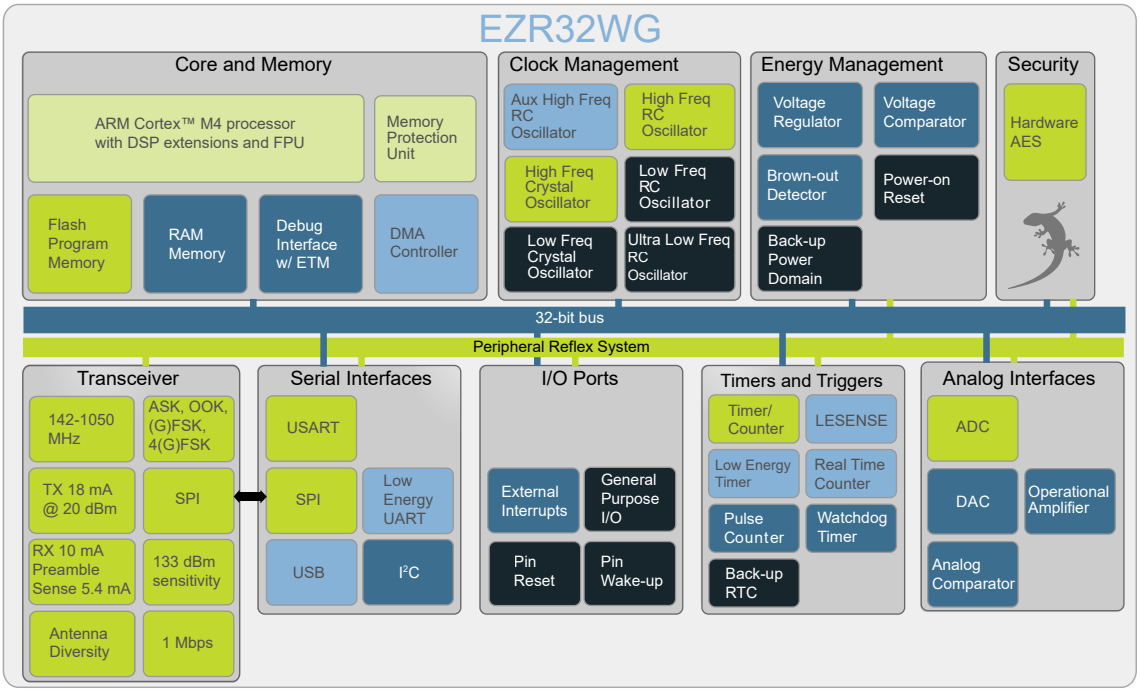
A precíz időbélyegzés alapja a számlálók azonnali, lehetőleg késleltetés nélküli lekérése. A feladat elvégzéséhez elengedhetetlen legalább egy órajel számláló egység használata. Ennek azonnali lekérdezését szoftveresen nem lehet megvalósítani, ugyanis mire az aktuális értéket kiolvassa a program, telik az idő. Ennek kiküszöböléséhez hardveres, úgynevezett *input capture* funkciót használunk, mely bemeneti változás esetén egy regiszterbe azonnal kimenti a számláló értékét, így a késleltetett kiolvasás már nem okoz gondot. A mikrokontroller rendelkezik 4 általános célú 16 bites számlálóval, melyek rendre három csatornásak. Minden csatorna szabadon konfigurálható *input capture* vagy *output compare* funkcióra. A protokollunk megvalósítása során *input capture*-t alkalmazunk két csatornán, ám mérések esetén *output compare* segítségével GPIO lábra ki tudunk vezetni PWM jelet, ennek detektálásával szemléltethető az oszcilloszkóp képén a szinkronizációnk pontossága. Számláló bemenete változtatható, lehet közvetlenül az oszcillátor, osztással vagy anélkül, de sorba is lehet kötni őket, ezzel kialakítva maximum négyszer 16 bites számlálót. Az MCU rendelkezik Real Time Counterrel (RTC) és Backup Real Time Counterrel (BURTC), melyek alacsony energiafogyasztású vagy backup módok esetén mérik az időt. Ezen kívül tartalmaz Low Energy Timert (LETIMER), mely az RTC-vel együttműködve, több komparálási értéket tud produkálni, mint az RTC önállóan.

Soros interfészek

Az EZR32WG speciális hardver architektúrával rendelkezik, ugyanis a mikroprocesszor és a rádiós csip fizikailag egy tokozású, mégis két külön álló eszköz. A közöttük zajló kommunikációt SPI-on keresztül bonyolítják le, a rádiós beállítások és üzenetek tartalma itt megy keresztül. Továbbá I^2C busszal is rendelkezik, mely a szenzorjaink illesztésében játszik szerepet, a konfiguráció és a gyűjtött adatok itt továbbítódnak. Alacsony energiaigénye miatt széles körben alkalmazott. USB és kimondottan UART modulok is találhatók a lapkán, ám ezeket nem használjuk fel a projektünkben. Továbbá egy speciális, úgynevezett LEUART egységgel is rendelkezik, amely egy alacsony energiafogyasztású UART, amely kisfrekvenciás órajellel működik.

GPIO portok

A mikrokontroller öt darab GPIO porttal rendelkezik, melyek rendre 16 kivezetést tartalmaznak. A feladata a perifériák egyes pinjeinek manipulálásának lehetővé tétele és hardveres hozzáférhetőségének elősegítése. A port regiszterein keresztül állítható egyenként a lábak funkciói. Időbélyegzés során elengedhetetlen ezeknek a használata, ugyanis egyes kivezetések konfigurálhatók, mint a számlálók Input capture bemenetei, így amikor élváltás történik a pinen, az időzítő regiszterbébe azonnal mentésre kerül a számláló aktuális értéke. Hasznos funkciója továbbá a rádiós csip úgynevezett GPIO2 és GPIO3 kivezetése, melyek a rádió konfigurációjával állíthatók, és jelenleg a protokollunkban olyan feladatot látnak el, hogy azonnal jeleznek a beérkezett vagy kiküldött üzenet detektálásakor, mely



3.2. ábra. Az eszköz blokkdiagramja.

lehetővé teszi a kiemelkedően pontos időbélyegzést. A következő felsorolás egyéb hasznos tulajdonságait ismerteti a GPIO-nak:

- Általános felhasználású, szoftverből elérhető analóg ki/bemenet
- PRS meghajtása
- Külső megszakítás bemenete
- Ébresztés – habár ezt a feladatot csak némely láb tudja ellátni

PRS

A Peripheral Reflex System (PRS) segítségével gyors és autonóm működést tudunk biztosítani a perifériák között. Ennek segítségével tehermentesíteni lehet a processzort, belső buszt és nem utolsó sorban gyorsítani lehet a rendszer működését. Azokat a perifériákat, amik reflex jeleket küldenek a PRS-en keresztül, meghajtóknak nevezzük. 12 konfigurálható összekapcsolható csatornával rendelkezik, melyek mindegyike rákötheto bármelyik meghajtóra. A reflex jelekre beállítható fel- és/vagy lefutóél detektálás, melyet a PRS detektál és értesíti a fogyasztókat a kívánt, általunk definiált módon. Protokollunkban két szabad GPIO lábat kötünk össze a számláló Input capture bemeneteivel, így gyorsítva az időbélyegzés folyamatát, üzenet küldésekor és fogadásakor.

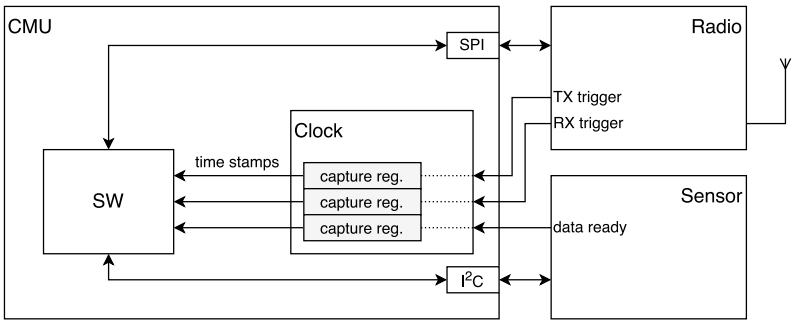
4. fejezet

Óraszinkronizáció vezeték nélküli beágyazott környezetben

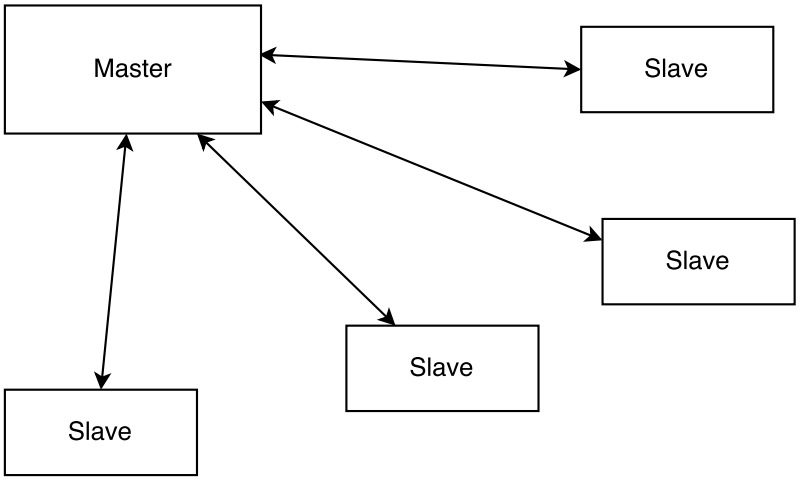
Ebben a fejezetben részletesen bemutatjuk az általunk használt óraszinkronizációs protokollt és algoritmust, a megoldás implementálása során tapasztalt problémákat, annak kiküszöbölését, javítását.

4.1. Kihívások

Óraszinkronizációs protokollunk megvalósítása során elengedhetetlenné vált, az üzenetek küldési és érkezési időpontjának pontos meghatározása. Erre a mikrokontroller és a rádió közötti lassú soros kommunikáció nem megfelelő. Így ezt megkerülve egy alternatív megoldást kellett keresnünk. Ezen kívül a csomópontok órajel generátorának névleges frekvenciája 48 MHz, ám a gyártási szórásból és az aktuális külső hatásokból származóan a valós értékek között eltérés van, ezért frekvencia hibát is kell kompenzálnunk. Továbbá, a rendelkezésünkre álló négy darab általános célú számláló rendre 16 bites. 48 MHz-es osztás nélküli órajel esetén egy számláló gyorsan túlcsordul (körülbelül 1,37 milliszekundum alatt), így a számunkra megfelelő hosszú idő mérésére alkalmatlan, ezért gondoskodnunk kellett annak kiterjesztéséről. A vezetékes infrastruktúra kiépítése nem mindig lehetséges vagy kifizetődő, ezért sok helyen szükséges a vezeték nélküli szenzor hálózatok alkalmazása. Ilyenkor egyetlen csatornán kommunikál minden egység és a csomagvesztés aránya is erőteljesen megnőhet. A rádiós csatornán előforduló adatvesztésre fel kellett készítenünk a protokollt a megbízható viselkedés érdekében. További problémát jelent az ugyan azon a frekvencián egy időben megkezdett kommunikáció, ugyanis ekkor egyik küldő fél üzenete sem feldolgozható a vevő számára. Vezeték nélküli hálózatokban a csatorna használat idejének felosztása az egyes adóvevők között, annak a csatorna aktuális foglaltságának érzékelése jelentősen megnehezíti ezen hálózatok valós idejű alkalmazását. Vezeték nélküli szenzorrendszer nagy előnye, hogy könnyen hordozható egységekből áll, sok szabadságfokú mozgó objektumokra is akadálymentesen felszerelhetők. Az ilyen típusú alkalmazáshoz a csomópontnak nem csak a kommunikációt kell vezeték nélkül lebonyolítania, hanem a tápellátást is, melyet az esetek döntő többségében elem vagy akkumulátor segítségével oldanak meg. Ekkor többnyire – általában méretkorlát miatt – csak kis kapacitású tápegységeket lehet felszerelni. Ilyenkor célszerű alacsony energiafogyasztású, az elvégzendő feladatra optimalizált egységet felszerelni. Protokollunk megoldása során törekedtünk annak egyszerűségére, a minél ritkább üzenet küldésére, a pontosság megtartása mellett.



4.1. ábra. Egy slave eszköz blokkdiagramja.



4.2. ábra. Alkalmazott master - slave architektúra.

4.2. Adatátviteli architektúra és protokoll

Ebben az alfejezetben a hálózat architekturális elrendezését és a megvalósított, felhasználó elől elrejtett kommunikációs protokoll működését mutatjuk be.

4.2.1. Hálózati elrendezés

Hálózati elrendezése az FTSN vezeték nélküli protokollhoz hasonlít. Mint a 4.2 ábrán is látható, hálózatunkban egyetlen mesteróra található, és kizárólag vele kommunikálnak slave csomópontok. Vezeték nélküli hálózat révén minden egység megkapja az összes slave üzenetét, ám ezeket a master-en kívül mindenki szükségszerűen elveti, mert számukra hasznos információt nem tartalmaz.

4.2.2. Kommunikációs protokoll

Jelenleg két kommunikációs protokollra fejlesztünk óraszinkronizációt. Egyik egy általunk írt időosztásos protokoll, amely szimbiózisban működik az óraszinkronizációnkkal, a másik a Silicon Labs által fejlesztett Connect Networking Stack (egyszeűbben Connect Stack) nevezetű absztrakt kommunikációs réteg.

Saját Protokoll

Egy master - slave architektúrájú rendszerben első lépésként azonosítani kell az egyes csomópontokat. Erre két fajta módszer létezik, lehet felprogramozáskor előre beégetett azono-

sítóval, vagy dinamikusan osztott számmal. Előbbi egy gyors, egyszerűbb megoldást kínál, ám ennek használatával a rendszer nehezen bővíthető. Amennyiben ezt a lehetőséget választjuk, számunkra egy relevánsnak tűnő megoldás, hogy egy hardveres azonosító alapján kezeljük az eszközöket. Így felprogramozás után kiderül, hogy mely egységre töltöttük fel a kódot, és ha szükséges, egy egyszerű algoritmus segítségével személyre szabott számmal tudjuk ellátni. Dinamikus azonosító osztás esetén az előre beégetett számra nincs szükség, ellenben valamilyen módon fel kell venni a kapcsolatot a master-rel. Itt szembesülünk az legfőbb problémával, ugyanis ahhoz hogy kommunikáljanak az eszközök, már szükség van protokollra, miközben mi éppen azt akarjuk beindítani. Ezért a kezdeti kapcsolatrendszer egy jóval egyszerűbb, véletlen hozzáférésű protokoll alkalmazását kényszeríti ki, melyben a slave eszközök véletlenszerű időközönként próbálnak üzenetet küldeni, kérve egy egyedi azonosítót a master egységtől. Ez egy meglehetősen komplex folyamat lehet, mivel egycsatornás vezeték nélküli hálózatot használunk. A rendszert valamennyire gyorsítani lehet, ha azon egység, mely kapott azonosítót elhallgat, és várja hogy minden csomópont csatlakozzon a hálózatra, majd a master ezt jóváhagyja. Ebben a realizációban új eszköz csatlakozása esetén nem biztosított az üzenetek ütközésének elkerülése. Remélni lehet csak, hogy a kérése hamar eljut a master-hez, és nem zavar bele a többi csomópont kommunikációjába. Az megvalósításunk során a eszköz 0 azonosítójú beégetett kódot kapja minden esetben.

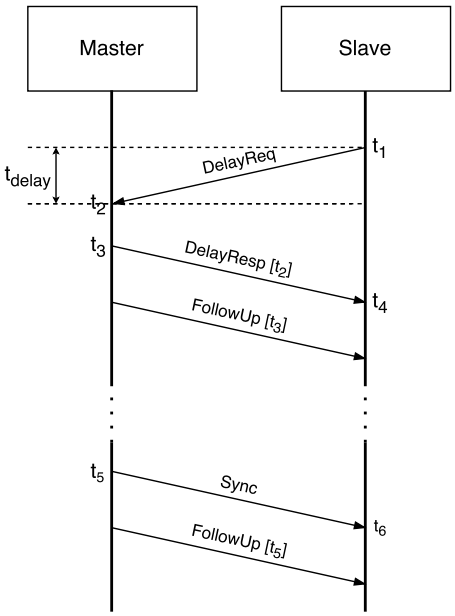
Azonosítást követően a master pontosan tudja, hogy hány slave egység kapcsolódik a rendszerhez. Az ő feladata innentől a kommunikációs protokoll létrehozásához szükséges adatok továbbítása. Amennyiben előre beégetett azonosítót használunk, az előbb említett adatokat is szintén felprogramozáskor betölthetjük a memóriába, így a bekapcsolódott eszközök már kész protokollon keresztül tudnak adatot küldeni, fogadni. Projektünk lényegi eleme az óraszinkronizációs eljárás megvalósítása volt, így kevesebb időt töltöttünk a kommunikációs protokollok tanulmányozásával. Ezért egy kézenfekvő ötlettel álltunk elő, mely időosztáson alapszik. Ebben az esetben a rendszer minden tagja kap egy idősávot, melyben üzenetet küldhet. Mivel a slave egység órája szinkronizálható – célnak megfelelő mértékben, –kizárólag a master által küldött üzenetekből, így a azoknak többnyire elég egyetlen ilyen ciklust megvárni. Ezek után az azonosítójuk alapján generált időtartományban küldhetnek szabadon üzenetet.

Connect Networking Stack

A Connect Networking Stack egy véletlen hozzáférésű csatornát használ és viszonylag egységes felületet biztosít több Silicon Labs gyártmányú rádiós mikrokontrollerhez. Egyszerűbb operációsrendszer-szerű funkciókat is tartalmaz. A magas absztrakciós szint sok mindent jelentősen megkönnyít és például az átlátható, és ezért kevesebb hibát tartalmazó kód írását. Azonban ennek az az ára, hogy az alacsonyabb szintű funkciókat körülményesebb elérni. Külön nehézség, hogy a könyvtár egy része bináris állomány, így annak pontos viselkedését nehéz megjósolni. Bár az itt megvalósított óraszinkronizációnkkal eddig nem voltak problémáink, magasabb csatorna terhelés esetén nem tudjuk garantálni a hibamentes működését, a rendszer alaposabb ismerete hiányában.

4.3. Algoritmus működése

A megvalósítandó szinkronizáló protokoll a master és slave kommunikációján alapul, a lezajlott üzenetváltás a PTP vezetékes implementációjához hasonlít. Ebben az alfejezetben mutatjuk be az implementált algoritmus részletes működését.



4.3. ábra. A szinkronizációs protokoll

4.3.1. Időbélyegzés

A protokollunk pontosságának alapja a helyes, azonnali időbélyegzés. Ez azt jelenti, hogy minden kiküldött és fogadott üzenet pontos idejét le tudjuk kérni, elmenteni. Szerencsére a rádiós chip képes érzékelni egy üzenet beérkezésének és küldésének idejét, majd jelezni azt egy külön GPIO lábon. Ezeket a kimeneteket hardveresen összekötöttük – PRS-en keresztül – az általunk használt számláló *input capture* bemenetével. Ennek hatására az időzítő azonnal kimenti egy regiszterbe az aktuális értékét, így aztán ki tudjuk olvasni szoftveresen a későbbi felhasználáshoz. Bár a rádiós csip belső működését és időzítését nem ismerjük pontosan, azt tudjuk, hogy a küldést a preamble elejétől jelzi, míg a beérkezésről a szinkronizáló szó detektálásánál szól, de ezt a konstans időt is kiküszöböli az átviteli késleltetést kompenzáló protokoll részlet.

4.3.2. Óraszinkronizációs protokoll

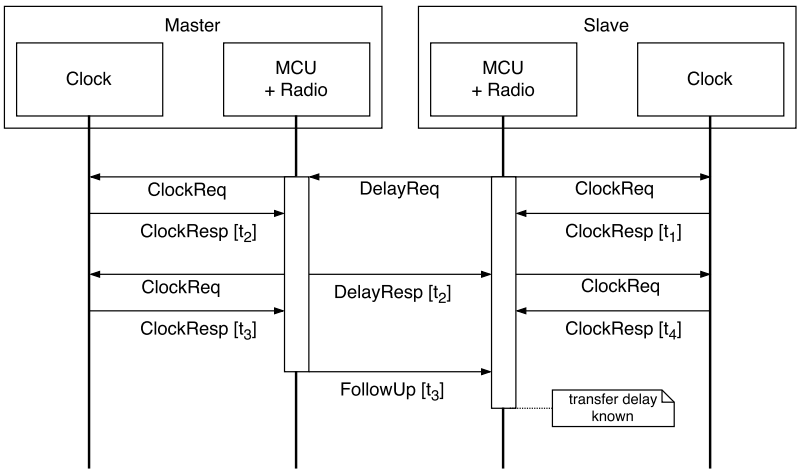
A szinkronizálás elején nem lehet garanciát adni az csomópontok számlálóinak egymáshoz viszonyított állapotáról. Ebből kifolyólag egyes eszközök bekapcsolásakor nem tudható, hogy az óráik között mekkora eltérés tapasztalható. Minden időpillanatban az aktuális eltérést a (4.1)-es egyenlet adja meg.

$$t_{\text{slave}} = t_{\text{master}} + t_{\text{offset}} \tag{4.1}$$

Feladatunk, meghatározni ezt az eltérést (t_{offset}). Ennek kiszámolásához a master küld egy szinkronizáló jelet, majd utána küldi az adás időpontját. Így az egyes slave-ek meg tudják határozni az órájuknak a master-éhez képesti eltérését a (4.2)-es képlettel. Ez még nem elég a pontos kalibrációhoz, hiszen az egyes slave-ek és a master közötti átviteli időről (t_{delay}) nem tudunk semmit.

$$t_{\text{offset}} = t_{\text{slave}_{\text{in}}} - (t_{\text{master}_{\text{out}}} + t_{\text{delay}}) \tag{4.2}$$

A késleltetés meghatározásához a slave küld egy "DelayReq" üzenetet a master-nek. A master erre az üzenet beérkezési idejével válaszol, majd még egy üzenetet küld a válasz



4.4. ábra. Az átviteli idő meghatározása

időbélyegével. Ekkor a slave minden adatot ismer a késleltetés meghatározásához, sőt, az utolsó két adat ismeretében akár össze is hangolhatja az óráját a master-évél. Ezt a folyamatot a (4.3)-as képlet valamint a 4.3 és a 4.4 ábrák szemléltetik.

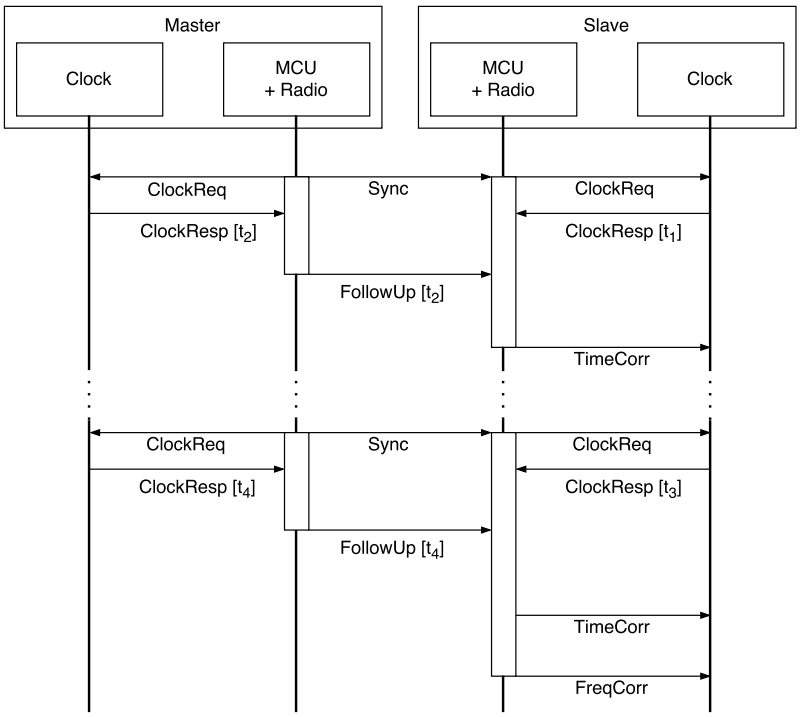
$$t_{\text{delay}} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{4.3}$$

Az eljárás feltételezi, hogy a késleltetési idő szimmetrikus, tehát a késleltetés meghatározásakor az üzenetváltások között az annak ideje nem változik. Optimális esetben egy slave egyszer kér egyeztetést a master-től, a működése elején. Ha a slave és a master közötti jelterjedési viszonyok változnak, a változás mértékétől és sebességétől valamint a szinkronizáció pontosságától függően szükség lehet az átvitel idejének többszöri meghatározására. Nekünk nem állt módunkban a jelterjedésben ilyen drasztikus változásokat előidézni úgy-hogy ezzel a problémával egyelőre nem foglalkoztunk.

4.3.3. Frekvencia korrekció

További problémát jelent, hogy az egyes slave-ek és a master az oszcillátorainak frekvenciái között szórás tapasztalható ezért, a szinkronizálást minél gyorsabban végre kell hajtani, hogy a késési idő számolásakor ne legyen szignifikáns az ofszet változása. Mivel az egyes egységek órafrekvenciái egyenként közel állandó frekvenciával járnak, idővel lineárisan növekvő eltérés figyelhető meg. Ezt frekvencia korrekció segítségével küszöböljük ki, a következőképp. Megmérjük hogy a szinkronizáló üzenetek között mekkora eltérés keletkezett és a szinkronizáció periódusát is. Ezek hányadosából meg kapjuk a relatív hibát, amivel tudunk kompenzálni.Az algoritmust a 4.5 ábra és a (4.4) mutatja be. A következő szinkronizációs fázisban, már módosítva lesz a frekvencia. Ekkor ugyanígy meghatározzuk a relatív hibát, majd a hiba hibáját elhanyagolva hozzáadjuk az előző értékhez, megkapva így megint a módosítatlan periódusidő relatív hibáját. A frekvencia hangolásának módszerére a 4.4.3 részben térünk ki.

$$\begin{aligned} T_{\text{period}} &= t_3 - t_1 \\ t_{\text{offset}} &= t_3 - (t_4 + t_{\text{delay}}) \\ \frac{t_{\text{offset}}}{T_{\text{period}}} &= \delta T(\text{relatív hiba}) \end{aligned} \tag{4.4}$$



4.5. ábra. Idő és frekvencia korrekció

4.4. Implementáció

Ebben az alfejezetben mutatjuk be az általunk használt protokollok megvalósítását, mind kommunikáció, mind pedig óraszinkronizáció kapcsán.

4.4.1. Kommunikációs protokoll

A 4.2.2 bekezdésben, eszköz azonosítás terén említett két eset közül, csak a beégetett kódú identifikációt valósítottuk meg egészében. Döntésünk meghozásakor figyelembe vettük, hogy számunkra egyelőre maximum négy csomópont áll rendelkezésre, így a bonyolultabb – azonosító osztó – verziót még nem lett volna kifizetődő alkalmazni. Minden egység rendelkezik egyedi azonosítóval, mely a gyártásának UNIX time formátumban tárolt időpontját és a telephely azonosítóját tartalmazza. Ennek felhasználásával a slave felprogramozása után lefut egy inicializáló algoritmus, mely az adott kódhoz hozzárendeli 1-től 4-ig valamelyik, előre definiált egész számot. Amennyiben master egységet kapcsolunk be, automatikusan 0 azonosítót kap, mint azt már 4.2.2 bekezdésben említettük. Így oldottuk meg, hogy a kódunk független legyen attól, hogy melyik a master és melyek a további slave-ek. Hátránya ennek a megvalósításnak, hogy a felprogramozott master egység azonosítója a slave-ek közt egy számot elfoglal a négy közül. Így azok közti időosztás továbbra is megmarad, viszont nincs eszköz ami a master egyedi kódjához tartozó időegységben adjon. Tehát a számunkra elérhető négy csomópont esetében, öt szakaszra kellett bontani a küldési periódust.

Az időosztásos kommunikáció alapja, hogy tudjuk, hogy a rendszerben hány résztvevő van, és mindegyik tag tudja, mikor van az ő időrése. Első lépésként definiálnunk kell azt az idősávot mely alatt minden egység küldhet üzenetet, azaz a küldési periódust. Esetünkben ezt az időhossz a számlálónk 8000 szeres túlcsordulásával egyenlő. Mivel az időzítőnk maximális értéke 7FFF hexa, azaz 15 biten számol, 48 MHz-es névleges frekvenciájú osztás nélküli órajelet figyelembe véve 5.46 másodperc a teljes periódus ideje, azaz nagyjából 1.1

szekundum az egy egységnek jutó küldési idő. Természetesen ezt egyetlen makró állításával lehet váltani, ám a méréseinket az itt felsorolt adatokkal végeztük.

Minden egység ugyanazzal az eljárással generálja az időosztását. Ezért pontosan tudja, hogy melyik a master sávja, melyik a sajátja, és melyek az egyéb slave-eké. Ez azért hasznos, mert az időigényesebb feladatokat, számításokat el lehet végezni abban az időintervallumban, amikor biztosan nincs bejövő üzenet – mert csak a mastertől kaphat releváns adatot –, és küldenie sem szabad.

4.4.2. Óraszinkronizációs protokoll

A szinkronizációnk megvalósításához a processzor órajelét a leggyorsabbra állítottuk – 48 MHz-re –, ennek lehető legpontosabb számlálásához a timer1 perifériát osztás nélkül inicializáltuk. Két input capture lábát (CC0, CC1) PRS-en keresztül kötöttük ki GPIO D4 és D5 pinjeire – ugyanis ezek hardveresen nincsenek kivezetve –, melyek a fejlesztőkártyán rendre a P44 és P45 lábakra vannak kötve. Utolsó – CC2 – kivezetés hardveresen a GPIO B11 pinjével van összekötve, mely a fejlesztőkártyán a P8 láb. Ezt kétféleképpen konfiguráljuk, az aktuális alkalmazás alapján. Amennyiben szenzort csatlakoztatunk az eszközhöz, input capture üzemmódban funkcionál, és a szenzor data_ready interrupt kivezetéssel kötjük össze közvetlen, így az érvényes adat időbélyegét tudjuk lementeni. Másik verzió a master esetében, vagy ha nem csatlakoztatunk szenzort, hogy PWM jelet vezetünk ki az adott lábon, így az óraszinkonizáció pontosságát tudjuk lemérni oszcilloszkóppal. A méréseink során mindkét konfigurációt alkalmaztuk, attól függően, hogy szenzoradatokat, vagy pontosságot akartunk-e mérni.A rádióval elküldhető üzenet maximális mérete 64 bájt, ugyanis ekkora FIFO-val rendelkezik. A küldő és fogadó félnek is tudnia kell, hogy mekkora az üzenet mérete, van-e CRC. Ezért, a úgy konfiguráltuk az üzenetet, hogy az első bájt a méretét adja meg a csatolt adatnak. Szerencsére ez a beállítás hardveresen támogatott megoldás, így könnyedén tudunk küldeni változó hosszú csomagokat. A rádiónak a két GPIO kivezetése a fejlesztőkártyán a P32, P33 lábakon elérhető, és az üzenetek detektálására konfiguráltuk. Ez azt jelenti, hogy küldés állapotba lépéskor, már a preamble előtt megjelenik a felfutó él az adott GPIO-n, és amint az utolsó bit is elküldésre került, visszaáll nullába. Vételi állapotba viszont a szinkronizáló szó (sync word) fogadása után kerül, ekkor vált logikai egyes szintre a láb, majd az utolsó bit beérkezte után visszaáll nullába

Üzenetek mérete az alábbi összetevőkből adódik. Az első, az üzenet adatsorának mérete(data_length), ezt követi az egyedi azonosító (id), majd az üzenet azonosítója(sequence_number), és típusa(message_type), rendre egy-egy bájton. A következő 3 bájtot az időbélyeg tölti ki, és eztán következhet az adatsor(data), melynek hosszát az üzenet mérete definiálja. Az óraszinkronizálás egy – a felhasználó elől elrejtett – háttérben futó folyamat. Minden szinkronizáló üzenet egyforma méretű, adatsor részének mérete nulla. Öt féle üzenetet definiáltunk, egyet a felhasználónak, és négy szinkronizálót. Ezeket a változókat enum segítségével alakítottuk beszédesebbé. A "SYNC", a "DELAY_RESPONSE" és a "FOLLOW_UP" típusú üzeneteket csak a master küldhet ki. A slave feladata a "DELAY_REQUEST" és a felhasználó számára definiált "DATA" üzenetek küldése.Ahhoz hogy a protokollunk megfelelően működjön, nem szabad hagyni, hogy a felhasználó az állandó szinkronizálást valamilyen, szándékán kívüli módon – például gyakori üzenetküldéssel – megakadályozza, késleltesse. Ezért a szinkronizáló üzenetek prioritást élveznek a kiküldés során. Ezt úgy érjük el, hogy egy speciális láncolt listában tároljuk az üzeneteket. Különlegessége, hogy kiküldéskor egyik végéről olvasunk és törlünk, míg a betöltés egy szinkronizáló üzenet esetén ugyanazon a végén-, felhasználó által berakott üzenet esetén pedig a másik végén történik.

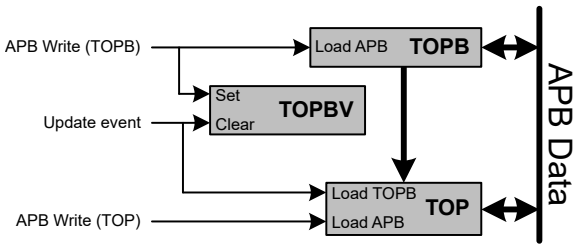
A slave az inicializálása során – amíg nincsen meg a pontos óraszinkronizáció – három állapotot érint. Kezdetben, szinkronizálatlan esetben a "Presync" fázisban van. Ebből akkor léphet tovább, ha az órája a master-hez képest kis eltéréssel jár (szám szerint a maximális tűréshatár 10000). Ezt a master által küldött "SYNC" és "FOLLOW_UP" üzenetpárossal könnyedén megteszi. Általában az első ciklussal szinkronizálja, második körben detektálja, hogy szinkronizálva van. Ezek után belép a "Get transfer delay" állapotba, ahol kiküld egy "DELAY_REQUEST" üzenetet a körülbelüli saját idősávjában. Erre választ kapva, meghatározza az átviteli késleltetést, és léphet is a végső állapotba, melynek az elnevezése "Measuring". Innentől a felhasználó használhatja az eszközt, az órák szinkronizálva vannak, a háttérben innentől csak a master küldi ciklusonként a "SYNC" és "FOLLOW_UP" üzenetpárost.

A rádiós csip nem küld interruptot, csak egy regiszterben módosítja a megfelelő bitet, ha fogadott vagy küldött üzenetet. Ezért ciklikusan meg kell hívni egy úgynevezett "ezradioPluginManager" függvényt, melyet a Silicon Labs a rádió kezelő könyvtárak részeként adott. Ennek a feladata ellenőrizni, hogy bebillentett-e bitet a rádió abban a bizonyos regiszterében, és ha igen, meghívni egy callback függvényt ez alapján. Feladatunk volt ezekbe a callbackekbe implementálni a szinkronizáló üzeneteket kezelő algoritmusainkat. Első sorban az üzenet típusa alapján kiválasztjuk, hogy mely kategóriába sorolható. Ha egyikbe se, tovább adjuk a felhasználó által definiált callbacknek, mert akkor ez nem egy szinkronizáló üzenet volt. Ezután megvizsgáljuk az egyedi azonosítóját a küldő félnek, hogy valóban nekünk szólt-e az üzenet, majd ha szükséges, ellenőrizzük az azonosító számát az üzenetnek.

4.4.3. Az óra működése

Dolgozatunk egyik kulcs eleme az óra. Bár a mikrokontrollerünk tartalmaz számlálót ez önmagában nem elég. Ahogyan a 4.1 alfejezetben is szó volt róla nem alkalmas hosszú és pontos időtartam mérésére egyszerre. A legkézenfekvőbb megoldás ennek kiküszöbölésére, hogy kiterjesztjük szoftveresen, azaz a számláló túlcsordulása esetén egy megszakítási rutinban növelünk egy értéket. A helyzetet tovább bonyolítja hogy a számláló *input capture* funkcióját használjuk, tehát ilyen alkalmakkor el kell mentenünk a szoftveres számláló bitjeit is. Ezt szintén az időzítő megszakítás kezelő rutinjában tesszük meg. A szoftveresen kiterjesztett számláló fölvet egy megkerülhetetlen problémát. Egy időpont elmentésekor a szoftveres és a hardveres biteknek konzisztensnek kell lenni. Ezt a következőképp oldottuk meg. A megszakítás kezelő függvény először ellenőrzi az állapot regisztert, amiből kideríti, mi okozta a megszakítást. Ha csak egy esemény van, egyszerű a dolgunk, lementjük az időt, vagy növeljük a szoftveres számlálót. Azonban előfordulhat hogy a túlcsordulás és a *capture* esemény olyan közel van egymáshoz, hogy egyszerre kell őket lekezelni. Ilyenkor meg kell állapítanunk, hogy melyik esemény következett be hamarabb, és annak a feladatát kell először elvégezni. Ezt úgy állapítjuk meg, hogy ellenőrizzük a *capture* regiszterét, és ha a benne lévő időpont a számláló időintervallumának első felében található, akkor a túlcsordulás volt hamarabb, ellenkező esetben a *capture*. Ez a gondolatmenet – ha gyors a megszakítás kiszolgálása – hibamentes működést eredményez.

Az óraszinkronizáció során elvárás, hogy az óra értékéhez gyorsan és pontosan tudjunk hozzáadni illetve abból kivonni. Ezt nagy számok esetén a számláló letiltásával és a kiszámolt új érték beírásával meg lehet oldani, (esetleg a folyamat időtartamával is kompenzálni kell) de ez nehézkes és megszakítások el is ronthatják. Gyakrabban előforduló probléma, hogy kisebb értékekkel kell módosítani az órát, hisz két szinkronizáló üzenet közt nem csúszik el nagyon az idő. Hardverünk számlálójának állítható a maximális értéke (top value). Belátható, ha csökkentem a a maximális értéket egy túlcsordulás erejéig, az az órához való hozzáadással egyenértékű. Ugyanígy ha növelem az kivonásként. Ám ha



4.6. ábra

kihasználom mind a 16 bitjét a számlálómnak nem tudom hová növelni a felső korlátot, ezért az alapból 7fff hexadecimális értéket állítottunk be. Ebben az esetben ha a legfölső bit is egyes, azt lekérdezéskor hozzá kell adni, a szoftveres számlálóhoz. Mindezeket a műveleteket megkönnyíti hogy egy bufferben is lehet maximális értéket beállítani, ami egyből betöltődik az előző helyére túlcsordulás esetén (4.6 ábra).

Ezzel a módszerrel kvázi a frekvenciát is lehet módosítani. Ha ismerjük az oszcillátor periódusidejének relatív hibáját (4.3.3), szorozva a számlálónk periódusidejével, megtudjuk mekkora eltérés keletkezik túlcsordulásonként ($\delta T \cdot T_{\text{overflow}} = T_{\text{err/ovf}}$). Ez valószínűleg kisebb mint egy órajel periódus, így önmagában nem elég. De ha periódusonként ezzel a számmal inkrementálunk egy változót, előbb utóbb az összegyűlt hiba meghaladja az órajel periódus idejét és tudunk vele kompenzálni. A következő kódrészlet fut le minden túlcsordulás megszakítás alkalmával:

```
int64_t high_timer_bits = 0;
float sum_of_error = 0;
float error_per_overflow;
```

```
high_timer_bits ++;
sum_of_error += error_per_overflow;
int sum_of_error_int = (int)sum_of_error;
if (sum_of_error_int)
{
        TIMER_TopSet(TIMER0, 0x7FFF+sum_of_error_int);
        TIMER_TopBufSet(TIMER0, 0x7FFF);
        sum_of_error -= sum_of_error_int;
}
```

5. fejezet

Mozgásérzékelő szenzor és időszinkronizáció integrációja

Az óraszinkronizáció működésének bemutatását egy mérőegységgel demonstráljuk, így ebben a fejezetben a szenzor kiválasztása és tulajdonságainak felvázolása után az illesztés és a kommunikáció módját mutatjuk be.

5.1. Szenzormodul kiválasztása

A 2.2. fejezetben már bemutattuk, a használni kívánt szenzorfajták alaptulajdonságait. Ezekkel az ismeretekkel és további elvárások meghatározásával több lehetőség közül kiválasztottunk egy szenzormodult.

5.1.1. Elvárások és lehetőségek

Egy szenzor kiválasztásánál nem biztos, hogy érdemes csak a 2.2.2-ben már tárgyalt alaptulajdonságok alapján dönteni. Példának okáért, esetünkben sem lehetséges, mert az alaptulajdonságokon kívül még egyéb meghatározó tényezők is fontosak a mérési eszköz kiválasztásánál. Az egyik ilyen szempont, hogy maga a szenzor képes legyen hardveresen előfeldolgozni a mért adatokat (pl.: ofszet kompenzálás) vagy tartalmazzon valamilyen digitális feldolgozó egységet. Egy másik szempont, hogy képes legyen bizonyos mozgások (pl.: érintés, szabadesés) hardveres detektálására. Viszont számunkra a lehető legfontosabb követelmény, hogy a használt szenzorok egy már kész modulon (breakout boardon) legyenek, a kiegészítő áramkörökkel együtt. Ezt az elvárást a rendelkezésre álló szenzormodulokkal kielégítettük, és hogy megbizonyosodjunk döntésünk helyességéről, módunkban állt még pár hazai beszállítótól gyorsan elérhető modellekkel együtt elvégezni az összehasonlítást.

Típus	Forrás	Gyorsulásmérő	Szögsebességmérő	Egyéb
GY-45 [4]	IQL	MM8452	-	-
GY-801 [6]	MIT	ADXL345	L3G4200D	HMC5883L (digitális iránytű), BMP085 (nyomásmérő)
GY-521 [5]	Tavir	MPU-6050		-
GY-86 [7]	Tavir	MPU-6050		HMC5883L (digitális iránytű), MS5611 (nyomásmérő)

5.1. táblázat. IMU modulok összehasonlítása

Az IncQueryLabs és a tanszék által kínált szenzor modulokon kívül még kettő hazai beszállítóktól könnyen beszerezhető modul tulajdonságait vizsgáltuk meg. Az 5.1 táblázat

a modulokon lévő szenzorokat nevezi meg, az 5.2 táblázat pedig az egyes modulokon található gyorsulásmérő és szögsebességmérő szenzorok alapvető tulajdonságait mutatja be. Az egyéb szenzorok bemutatását nem tárgyaljuk, mivel az alkalmazás szempontjából vagy felesleges a használatuk, vagy csak komplex kalibrációs eljárásokkal tudnánk használni őket.

	MM8452 [16]	ADXL34 [3]	MPU-6050 [8]
Felbontás [bit]	12 vagy 8	10..13	16
Mérési tartomány [g]	±2/4/8	±2/4/8/16	±2/4/8/16
Nemlinearitás	±2,64%	±0.5%	±0.5%
Érzékenység	±2.64%	±1%	±3%
Ofszet [mg]	±20	±150 (x,y), ±250 (z)	±50 (x,y), ±80 (z)
Mérési sebesség [Hz]	1.56..800	0.1..3200	4..1000

5.2. táblázat. Gyorsulásmérők összehasonlítása

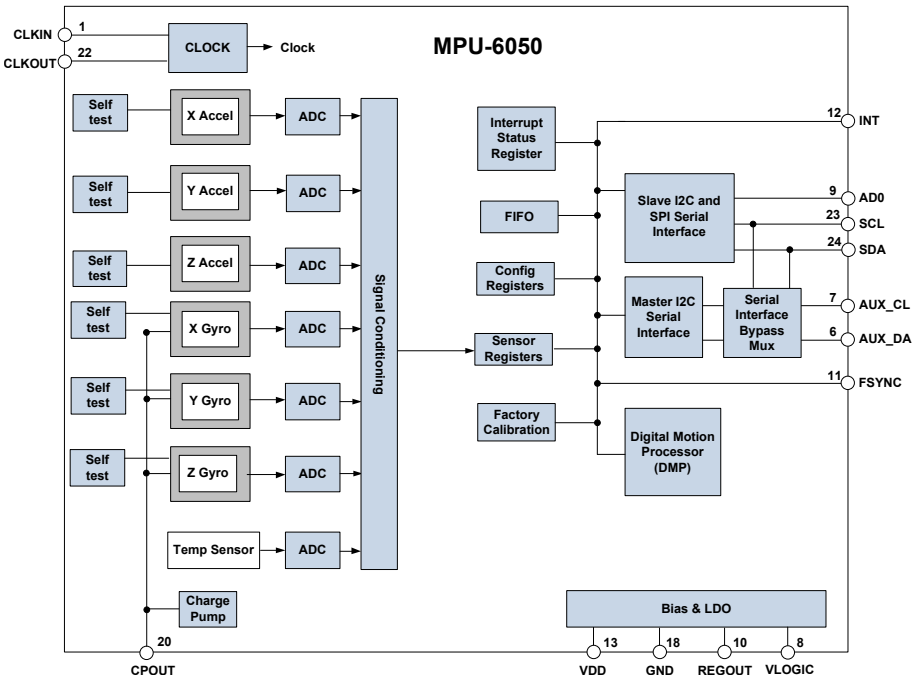
	L3G4200D [17]	MPU-6050 [8]
Felbontás [bit]	16	16
Mérési tartomány [°/s]	±250/500/2000	±250/500/1000/2000
Nemlinearitás	0.2%	0.2%
Érzékenység	±2%	±3%
Ofszet [°/s]	±75	±20
Mérési sebesség [Hz]	100..800	4..8000

5.3. táblázat. Szögsebességmérők összehasonlítása

A táblázatból látható, hogy ezek az adatok nagyjából hasonlóak, lényeges eltérés a további előnyös funkciók (érintés detektáció, előfeldolgozó egység) meglétében tapasztalható. Az összehasonlítás alapján mérlegeltünk, és úgy döntöttünk, hogy az MPU-6050-es IMU-t tartalmazó szenzormodulokat használjuk. Viszont a GY-86-os modul további szenzorjait nem tudtuk alkalmazni, mivel a digitális iránytű kalibrálásához szükséges komplex kalibrálási algoritmusaival nem sikerült kalibrálnunk ezt a szenzort.

5.1.2. Választott szenzor: MPU-6050

Ezt a szenzort alapvetően az okostelefon és táblagép gyártók számára fejlesztették ki, ahol szükség van a különböző gesztusok felismerésére. Ez egy 6 szabadságfokú eszköz, amely 3-tengelyű gyorsulásmérő, a 3-tengelyű szögsebességmérő és a Digital Motion Processzor segítségével képes a mozgás nyomon követésére Az eszköz I^2C-n keresztül képes kommunikálni. Ezen kívül rendelkezik egy kiegészítő I^2C busszal, amelyre digitális iránytűt illesztve lehetővé válik a 9 szabadságfokú MotionFusion. Erre az I^2C buszra más szenzort (pl: nyomásmérőt) is lehet illeszteni. Az alacsony fogyasztású módot elősegíti egy csipbe épített 1024 byte méretű FIFO buffer, ennek segítségével nagyobb csomagokban tudjuk az adatot olvasni. A gyorsulásmérőn és szögsebességmérőn kívül még egy beágyazott hőmérséklet szenzor is a rendelkezésünkre áll, amely elsődlegesen a hőmérsékletfüggő hibák kalibrációját segíti. A szenzorok számára rendelkezésre áll programozható alul áteresztő szűrő is. Az 5.2 és az 5.3 táblázatokban megtalálható alapadatokon kívül egyéb adatokat a adatlapban és a regisztertérképben találhatóak.



5.1. ábra. MPU-6050 blokk diagramja [8]

Szenzor konfigurációja a mérésekhez

Egy szenzor beállításánál általánosan szükséges a mérés határainak, a mintavételezés nagyságának és módjának és a felhasznált órajelnek a beállítása. Esetünkben sleep módban történik meg az órajel forrásának beállítása, illetve a gyorsulás és szögsebesség méréshatárainak megadása. Ezután átkapcsoljuk sleep módból mérési módba a szenzort, majd beállítjuk a mintavételezési frekvenciát és az megszakítás módját és működését. Természetesen mielőtt ezeket a lépéseket megtesszük, ellenőrizzük, hogy sikeresen létrejött a kapcsolat a szenzor és az adott mikroprocesszoros egység között. Ezeket a beállításokat a szenzor regisztereinek adott bitjeinek állításával tehetjük meg. A WHO_AM_I regiszter lekérdezésével megállapíthatjuk a kapcsolat létrejöttét, hogyha ez 0x68-at ad vissza, akkor meg van a kapcsolat. Ugyanis a szenzor a 0x34-es címmel csatlakozik és ennek a WHO_AM_I regiszterben egy bittel eltolva kell, hogy megjelenjen, amennyiben van kapcsolat. A sleep módot be- és kikapcsolni a PWR_MGMT_1 regiszter hetedik bitjének állításával lehet. Ugyanebben a regiszterben az utolsó 3 bit segítségével tudjuk beállítani az órajel forrását is. A méréshatárok beállítása pedig az ACCEL_CONFIG és a GYRO_CONFIG regiszterekben lehetséges, mindkét regiszterben a középső két bit határozza meg ezt, így esetünkben bitmanipulációs műveletekre is szükség volt, hogy a megfelelő biteket tudjuk változtatni. Alapesetben a szögsebességmérő 8kHz-zel, míg a gyorsulásmérő csak 1 kHz-zel tud maximálisan mintavételezni, így azért, hogy egyszerre tudjanak mintavételezni a SMPLRT_DIV regisztert 7-re állítjuk, melynek következtében $8/(1+7) = 1$kHz lesz a mintavételezési érték. A megszakítás engedélyező regiszterét (INT_STATUS) 1-be állítjuk, így minden mintavételezés esetén az adatregiszterek frissülése után megszakítás generálódik.

Problémák a nyers mérési adatokkal

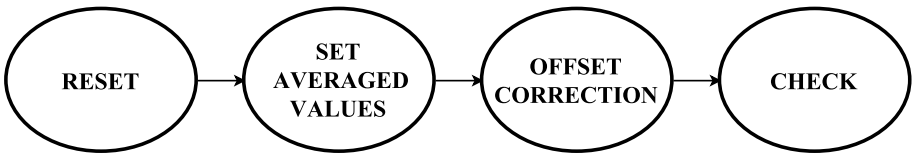
Ugyanúgy, mint a konfigurációs beállításokat, a szenzor mérési adatait is regiszterekből tudjuk kiolvasni. Mind a gyorsulás, mind a szögsebesség adatokat 16 biten ábrázolja a szenzor, így minden tengelyen 2-2 regiszter tartalmazza az értékeket, az egyik a felső nyolc bitet, a másik pedig az alsó nyolc bitet.

Az alapértelmezett felbontásokat meghagyva ($\pm$2g, $\pm$ 500 $^\circ/s$), illetve a szenzort az asztalra téve azt várjuk, hogy az x, y, z tengelyen a szögsebességmérő nullához közeli értékeket adjon, a gyorsulásmérő pedig az x, y tengelyen nullát vagy ahhoz közel eső értéket, míg a z tengelyen pedig 16384-hez közeli értéket mutasson.

A mérési adatok viszont nem az elvártaknak megfelelő értékeket kapjuk. Azoknál az értékeknél ahol nullát vártunk, többnyire valamilyen érték között ingadozik a mérési eredmény, illetve a z tengely mentén sem a várt értéket mérünk, hanem más konstans értéket. A 2.2.2 fejezetben taglalt, az egyes szenzorok tulajdonságai által okozott ofszet hiba miatt kapjuk ezeket a mérési eredményeket. A szenzorra jellemző hibák tartománya az adatlapon meg is van adva, viszont ez a használat során változhat, így ha nem használunk folyamatos kalibrációt, akkor nem árt adott időközönként megvizsgálni az ofszetek változását. Amint tudtában vagyunk ezeknek az ofszeteknek, két lehetőség elé kerülünk. Az egyik, hogy programozottan mindig ki kell vonni a mérési adatokból. A másik lehetőség, hogy az ofszet értékeket regiszterekbe mentjük, ami alapján a szenzor magától korrigálja a mérési eredményeit és így a már korrigált eredményeket menti ki az adatokat tartalmazó regiszterekbe. Esetünkben a második módszer szerint tudjuk kompenzálni az gyorsulásmérő és a szögsebességmérő méréseit. Ezeket az ofszet regisztereket is a mérési adatot tartalmazó regiszterekhez hasonlóan kezeljük. A használt szenzor pedig ezek alapján előfeldogozza a mért értékeket, így a regiszterekből kiolvasható értékek már ofszet-kompenzálva vannak.

5.1.3. Ofszet kompenzációs algoritmus

Ahhoz, hogy jó értéket tudjunk írni az ofszet regiszterekbe, ahhoz szükség lenne egy olyan kalibrációs algoritmusra, amely nyugalmi helyzetben kellő pontossággal meghatározza azokat az ofszet értékeket, amellyel a valósághoz közeli eredményt kapunk (a zajt nem fogjuk tudni teljesen kiszűrni).



5.2. ábra. Offset kalibrációs algoritmus folyamatábrája

Mivel a kezdeti fejlesztéseket egy Arduino Nanon kezdtük, ezért egy Arduinóra tervezett automatikus ofszet kalibrációs algoritmust használtunk fel [15]. Ez a kalibrációs algoritmus 4 részre bontható. Az első részében kinullázza az ofszet regisztereket. A második részben mér annyi adatot, amennyit a programozó megfelelőnek talál (ezt az értéket a mintavételi frekvencia korlátozza) majd az átlagolásuk után a kapott értékek alapján kitölti az ofszetregisztereket, a harmadik részben addig pontosítja az ofszetregiszterek értékeit, amíg a programozó által definiált határokon belülre nem kerül a mérési eredmény, amint ez bekövetkezik átlép a negyedik részbe, ahol kiíratja az ofszeteket és egy mérési eredményt, amit ezzel a beállítással mért.

Annak ellenére, hogy ez az algoritmus nem használ komplex matematikai műveleteket a hibák prezentálására és kompenzálására, használatát ehhez az alkalmazáshoz számos indokból megfelelőnek találtuk. Az egyik ilyen legfontosabb ok, hogy a kalibrációhoz nincs

szükség emberi beavatkozásra, a kalibráció a szenzor nyugalmi állapotában megtörténik. A másik, hogy sok kalibrációs algoritmussal ellentétben nincs szükség rá túl sok időre, ugyanis az algoritmus egy percen belül végrehajtódik az alapértelmezett felbontásokkal, nagyobb felbontások esetén pedig a tűréshatárok növelésével csökkenthető a kalibrációs idő. A harmadik szempont, hogy nincs szükségünk kiegészítő szoftver használatára, mivel az algoritmus nem használ komplex mátrixműveleteket. Az utolsó indok, hogy a jelenlévő erősítési és nemlineáris hibák gyárilag kompenzálva vannak, így a megváltozott erősítési hiba még kisebb hatással van jelen a mérési eredményekben.

Az ofszet kalibráción kívül az erősítési hibát is kompenzálnunk kéne. Ezt a hibát több okból sem kompenzáljuk. Az egyik, hogy egy olyan kalibrációs algoritmust akartunk megvalósítani, amelyhez nem kell emberi beavatkozás, azaz nyugalmi helyzetben is megtörténik a viszonylag pontos kalibráció. A másik, hogy a szenzormodul hardveresen gyári beállításai alapján kompenzálja az erősítési és a nemlinearitási hibát, a megváltozott erősítési hibának kompenzálására pedig nincsen hardveres megvalósítási lehetőségünk.

5.2. A szenzor illesztése a rádiós csiphez

Ahhoz, hogy látványosan szemléltetni tudjuk az óraszinkronizáció hasznát, egy szenzort választottunk ki és illesztettünk a mikroprocesszorhoz. A továbbiakban bemutatjuk, hogy mily módon történt ez az illesztés és hogy erre milyen lehetőségeket használtunk fel.

5.2.1. Használt könyvtár

Az 5.1.2 fejezetben bemutattuk, hogy a MPU-6050 DMP funkciója nagyon sok mozgást nyomon követő lehetőséget biztosít a felhasználó számára. Viszont ez sajnos a fejlesztés során nem igazán volt hasznunkra, ugyanis a szenzor adatlapja és regisztertérképe nem add semmilyen lehetőséget ezen funkciók elérésére. Továbbá ahogy már az 5.1.3 fejezetben említettük, először Arduino Nano-val végeztünk a fejlesztéseket, így a gyártó által felkínált statikus könyvtárt sem tudtuk használni. Szerencsére rendelkezésre állt Jeff Rowberg által reverse engineering módszerrel visszafejtett kód, amellyel elérhetővé vált pár funkció. Ezeket a könyvtárakat átírva C++-ból C-be, fel tudtuk használni az EZR32WG modulhoz. Ehhez a mikrokontrollerhez már használhattunk volna az Invensense statikus könyvtárát, ám ez nem volt túl jól dokumentálva és a Jeff Roberg által elérhető funkcióknál sem kínált sokkal többet, így maradtunk az eddig használt könyvtárnál.

I2Cdev

Ez a könyvtár magasabb szinten kezeli az I^2C-n keresztül történő kommunikációt. Így a függvényeinek alkalmazásával megkönnyíti az I^2C használatát. Ezen kívül a többi könyvtár is épít a függvényeire, ezért ennek megléte kiemelten fontos.

MPU6050

Olyan függvények vannak implementálva, amelyek segítségével könnyeben kezelni tudjuk a MPU-6050 regisztereit. Ennek okán könnyebb és átláthatóbb a szenzor konfigurációja, kalibrációja és adatregisztereiből való olvasás is.

MPU6050_6Axis_MotionApps20

Ebben a könyvtárban található a visszafejtett kód nagy része. Függvényeivel a előfeldolgozott és szenzorfúzionált adatokat kérhetünk le a szenzortól. Egyik hátránya, hogy maga a

kód nem túl átlátható, így gyakorlatilag semmilyen módosítást nem lehet eszközölni rajta. Viszont a mérések során nem volt szükségünk a használatára.

imu

Ez a könyvtár már nem tartozik a Jeff Rowberg könyvtárához. A bemutatott kalibrációs algoritmus és – egyes alkalmazások által hasznos – érintés detekció függvényeit tartalmazza.

main_sensor

A könyvtárban szereplő függvények megkönnyítik az eddigi könyvtárak gyors portolását és használatát. Lehetővé teszik, hogy csak ezt a fájlt implementálva elérhetőek legyenek az előbb említett könyvtárak és függvények.

type_convert

Ebben típuskonverziós függvények találhatóak, amelyek használatára a szenzoradatok rádiós küldésénél és fogadásánál van szükségünk.

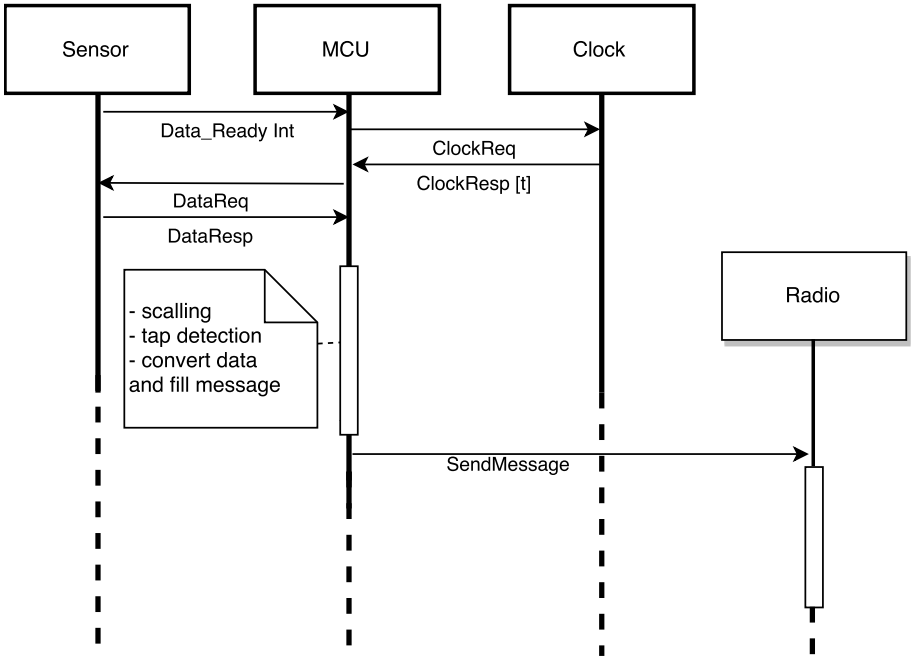
algorithms

Ez a könyvtár szenzorfúziós algoritmusok megvalósításait tartalmazza. Ezek segítségével a szenzoradatokból olyan fizikai mennyiségeket tudunk előállítani, amelyekkel pontosan le tudjuk írni a fizikai valóságot. A mérések során ezt a könyvtárat sem fogjuk használni.

5.2.2. Architektúra

A szenzor a fejlesztő lapka kiegészítő headerjén keresztül csatlakozik az EZR32WG-hez. Az mikrokontroller PD6-os pinjén az SDA, a PD7-esen pedig az SCL vonalak vannak, melyek segítségével I^2C-n keresztül kommunikál az MPU-6050 a rádiós egységet tartalmazó mikrokontrollerrel. Ezen kívül a szenzormodulhoz van kötve a 3,3 V-os táp és a föld is. A szenzor mintavételezésekor új adatok íródnak be az adatregiszterekbe, melynek hatására megszakítás generálódik az egyik kivezethető lábán. Ezt a lábat a használt számláló harmadik csatornának portjára kötöttük, így pontosan le tudjuk kérdezni az egység óráját a mintavétel bekövetkezte után. Amint ez megtörténik lekérdezzük az adatregiszterekbe található adatokat. Ezután a digitális adatot fizikai mennyiséggé alakítjuk, majd át konvertáljuk bit folyammá, hogy a rádiós egység ki tudja küldeni. Azután pedig egy Message-ként elküldjük a rádiós egységnek.

A adatbeolvasás után és a rádióhoz való kiküldés között elérhető még egy folyamat, amely feldolgozza a szenzor adatokat. Ezt az ábrán Tap Detection-ként van feltüntetve, de valójában inkább egy adott gyorsulás küszöbszint detektálására szolgál. Ezzel az algoritmussal ütés jellegű eseményeket vizsgálunk, így a beállított határszint feletti értékeket fogjuk, csak elküldeni a masterhez. Ezzel a korlátozással csökkenthetjük az adatok átküldésének nagyságát, így kevesebb sávszélességet foglalunk el.



5.3. ábra. Szenzor adatainak olvasása és bélyegzése

6. fejezet

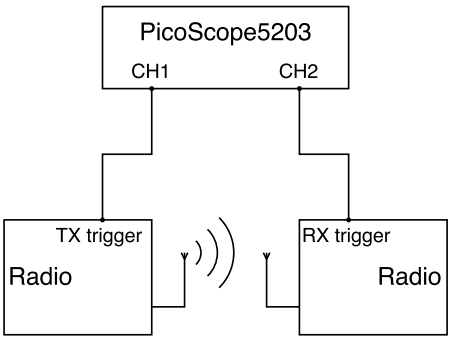
Eredmények

6.1. A szikronizáció pontossága

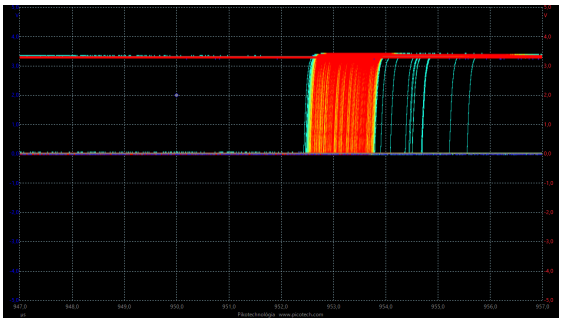
Ebben a fejezetben az óraszinkronizációnk pontosságát mutatjuk be. A méréseket egy PicoScope5203 digitális oszcilloszkóppal végeztük. Az első méréssel, a küldés és fogadás időbélyegzésének pontosságát mutatjuk be. Az oszcilloszkóp két csatornájára rákötöttük egy adó és egy vevő eszköz küldés illetve fogadás időbélyegzésre használt jelét. Az adás jel felfutó élére triggereltünk, és a vétel jel felfutó élét vizsgáltuk digitális foszfor bekapcsolásával 10 percig. A mérési elrendezést a 6.1 ábra, az eredményt a 6.2 ábra mutatja. Látható hogy az üzenet detektálása egy 1-1,5 mikroszekundumos sávban körülbelül egyenletes eloszlással történik. Az is észrevehető, hogy néhány él jócskán kívül esik ezen a szakaszon aminek meg lesznek a következményei.

A második mérés során már valóban a szinkronizáció pontosságával foglalkoztunk. Egy master és egy slave egység számlálóival PWM jelet generáltunk olyan módon, hogy túlcsordulás esetén felfutó él keletkezzen. Ekkor felfutó élek különbsége megadja az órák eltérését. Itt a master jelre triggereltük az oszcilloszkópot, és itt is a digitális foszfor beállítást használtuk. Először frekvencia korrekció nélkül mértünk, másodpercenkénti szinkronizáló jel esetén. Bár az ábrán nem látszik, a slave oszcillátora gyorsabb volt, így egyre korábban jött a túlcsordulást jelző felfutó él a master-éhez képest, amíg a szinkronizálás hatásra vissza nem került a master közelébe. Bár a foszfor csak 5 mikroszekundumnyi területet fest meg, és a master-hez képesti legnagyobb eltérés 4 μs, ez két slave között több is lehet hiszen ha egy slave oszcillátora a másik irányba tér el ugyanennyire, az máris 8 μs hiba.

A frekvencia korrekció bekapcsolása után öt mérést végeztünk különböző gyakoriságú szinkronizálással. Ezekben az esetekben megállapítható, hogy a szinkronizáció jóval sikeresebb, az eltérés nagyrészt kisebb mint 1 μs. Azonban mindegyik ábránál megfigyelhetőek

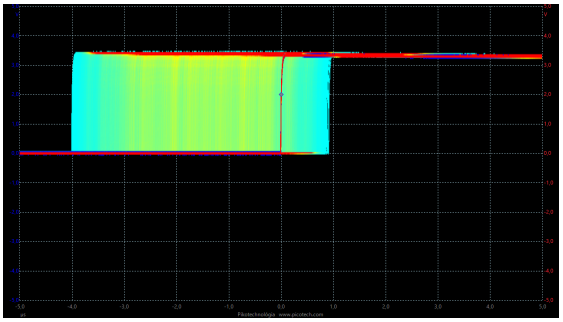


6.1. ábra. Mérési elrendezés

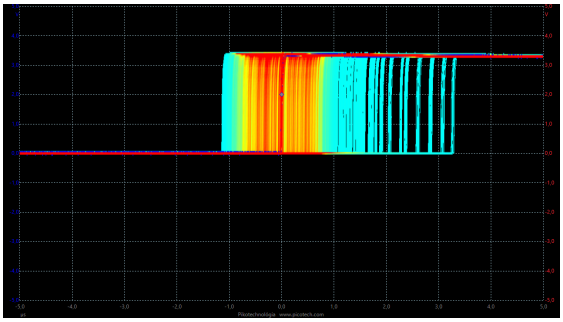


6.2. ábra. Az átviteli késleltetés ingadozása. (2 μs/osztás és 1 V/osztás)

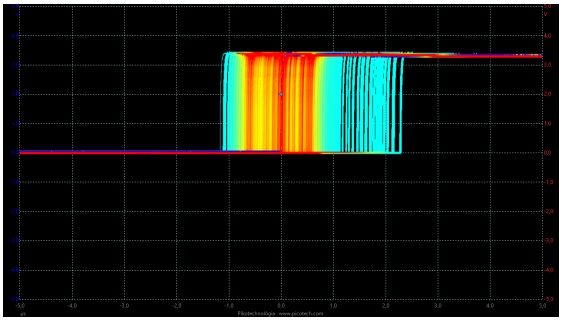
kiugró értékek, ahhol a slave jelentősen késik a master-hez képest. Ezek a kiugró értékek mérésenként 1-4 alkalommal fordultak elő és valószínűleg a késleltetés ingadozás hasonló eredményei okozzák, hiszen, ha az előre meghatározott késleltetéshez képest jóval később érkezik meg a szinkronizáló üzenet, a szabályzó azt hiheti túl gyors az órajel, és drasztikus csökkentésbe kezd. megfigyelhető, hogy ez az eltérés aközepes hosszúságó 2 másodperces szinkronizáló jelnél okozza a legdrasztikusabb hibát, valószínűleg azért, mert a lassabb esetekben az okozott hiba relatíve kisebb, gyorsabb szinkronizáció esetén pedig nincs ideje az órának nagyon elhangolódni. A mérésekről érdemes még megjegyezni, hogy az átviteli késleltetés meghatározásakor szerencsénk volt, hiszen a foszfor színezése alapján a hiba várható értéke mindig nulla körül volt.



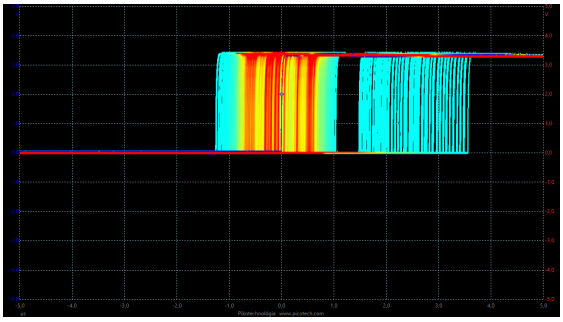
Nincs frekvencia korrekció,
a szinkronizáció periódusideje 1 s.



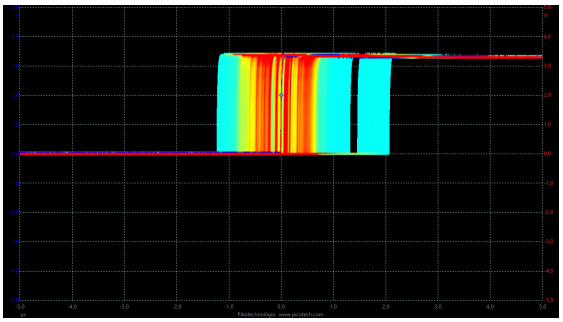
Van frekvencia korrekció,
a szinkronizáció periódusideje 0,5 s.



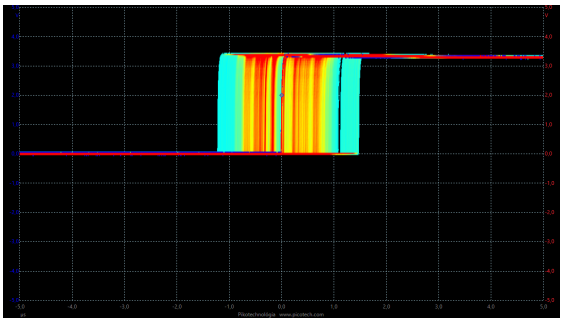
Van frekvencia korrekció,
a szinkronizáció periódusideje 1 s.



Van frekvencia korrekció,
a szinkronizáció periódusideje 2 s.



Van frekvencia korrekció,
a szinkronizáció periódusideje 4 s.



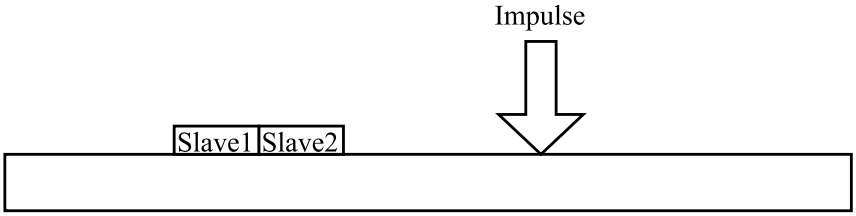
Van frekvencia korrekció,
a szinkronizáció periódusideje 8 s.

6.3. ábra. Mérési eredmények. Minden mérés 10 percig tartott.
(2 μs/osztás és 1 V/osztás)

6.2. Esettanulmány

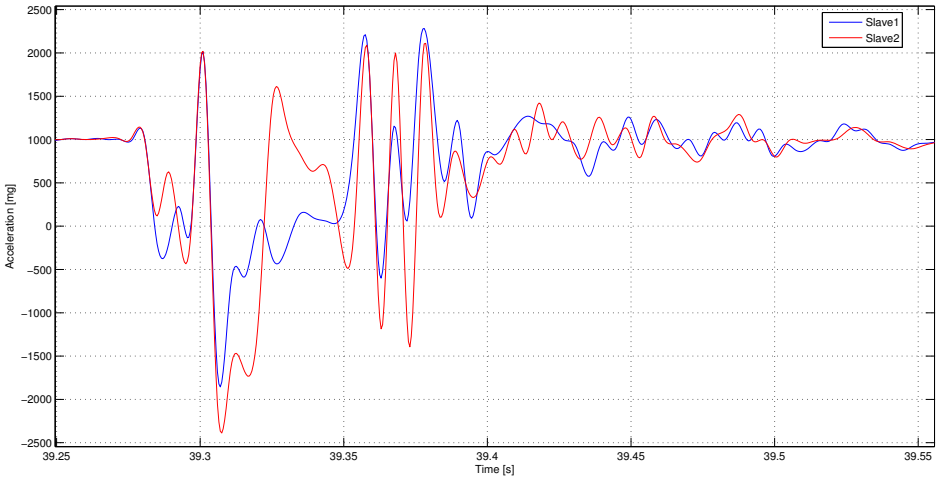
A létrehozott óraszinkronizációs protokollt a slave egységekre illesztett szenzorral rendelkező hálózatban mutatjuk be. Kettő darab slave egységre két mérési elrendezésben impulzus jellegű jellel gerjesztettük. A mintavételezések idejét pontosan meghatározva meg tudjuk mutatni, hogy az egyes slave egységek milyen időrendi és sorrendi viszonyokkal mérték az impulzust és értékelni tudjuk az óraszinkronizációt.

A mérést egy széles keskeny asztalon végeztük. Gerjesztésként pedig egy asztalra csapás funkcionált. A mérési adatokat interpoláltuk a Matlab spline metódusával. A szenzor méréshatárai és egyéb konfigurációs beállításai az 5.1.2 és az 5.1.2 fejezetben beállítottaknak felelnek meg.



6.4. ábra. Első mérési elrendezés

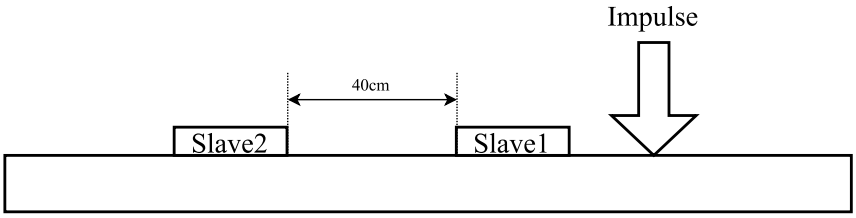
Az első mérési elrendezésben a két szenzort ugyanazon a tesztpanelre raktuk. Tőle kb. 20cm-re adtuk a gerjesztést. Azt vártuk, hogy a mérési adatok közel egyszerre jelenjenek meg közel ugyanolyan nagysággal.



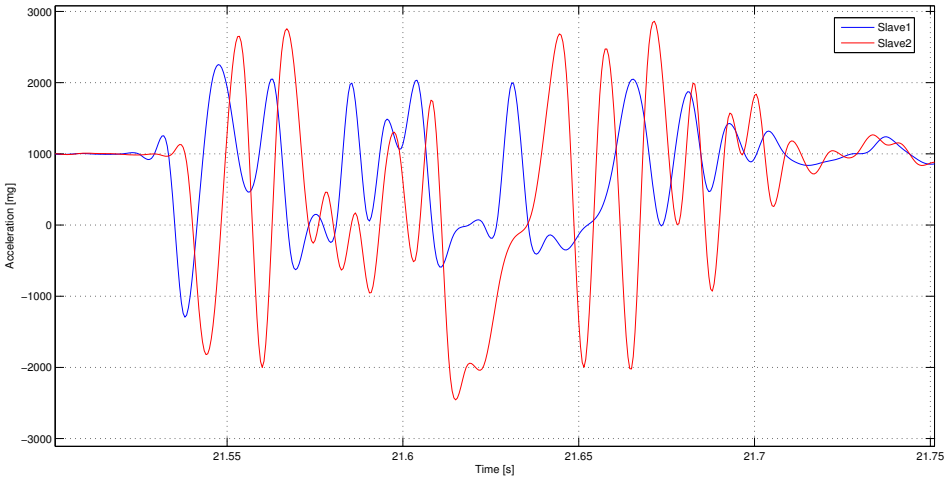
6.5. ábra. Első mérési eredmény a Z tengelyen

Ahogy a 6.5 ábrán látható, mindkét szenzor által érzékelt gyorsulás adat ugyanabban az időpontban nagyjából ugyanazzal az értékel jelenik meg. A későbbi minták viszont már nem ennyire hasonló értékeket mutatnak. Ennek az oka az asztal végéről visszaverődő rezgés illetve a gerjesztésként adott impulzus utórezgéseinek összessége.

Az második mérési elrendezésben a két szenzort egymástól kb. 40cm-re raktuk. A jobb oldali szenzortól kb. 20cm-re adtuk a gerjesztést. Azt vártuk, hogy a mérési adatok kisebb elcsúszással jelenjenek meg viszonylag eltérő nagysággal.



6.6. ábra. Második mérési elrendezés

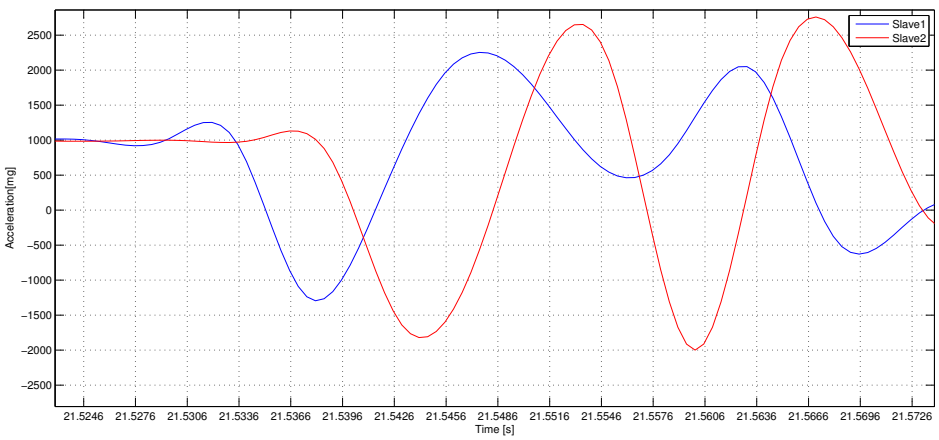


6.7. ábra. Második mérési eredmény a Z tengelyen

A 6.7 és a 6.8 ábrán látható, hogy a két szenzor által mért adat elcsúszik egymástól. Ez a két egység közötti távolság miatt van. Viszont ennek ellenére látszik, hogy jellegre hasonló jeleket érzékelnek.

Az ábrákon látszik, hogy a jelek általában túllépnek a beállított 2g-s méréshatáron. Ennek a megjelenése abból következik, hogy a méréshatár a gerjesztett jelhez képest alacsony, így a mérési eredmény sokszor eléri a maximumot. Az alkalmazott interpoláció két határérték közötti interpolált mintákat pedig a mérési tartományon kívülre számítja ki.

A mérést a későbbiekben más mérési elrendezésekkel és más mérés határokkal megvalósítva még inkább látszana, hogy miért nagyon fontos az óraszinkronizáció. Viszont, így is sikerült a szenzorok mintavételezésinek időpontját nagy pontossággal meghatározni. Egy időkritikus rendszernél ahol valósidejű adatkora van szükségünk ez a tulajdonság nagyon hasznos.



6.8. ábra. Két szenzor adatainak időbeli elcsúszása

7. fejezet

Összefoglalás

7.1. Eredmények összegzése

Munkánk célja egy olyan óraszinkronizációs protokoll kidolgozása volt, amely segítségével pontosan meg tudunk állapítani egy elosztott szenzorhálózat időzítési és kauzalitási viszonyait.

A vezetéknélküli alacsony fogyasztású rendszerek elterjedésének következtében elengedhetetlenné vált egy ilyen rendszerhez való óraszinkronizációs eljárás kidolgozása. A választott hardverkörnyezettel elértük, hogy nagy pontossággal tudjuk szinkronizálni óráinkat. Mindezt a az időbélyegek újszerű felhasználásával, szinkronizációs üzenetek küldésével és frekvencia korrekcióval valósítottuk meg.

Az óraszinkronizációs eljárást egy saját konstrukció által megvalósított és egy már létező kommunikációs protokollba is beépítettük. Majd egy mozgás nyomon követésére alkalmas szenzort választottunk és illesztettünk a rendszerhez.

Az óraszinkronizációval μs-os nagyságrendel meg tudtuk állapítani a választott szenzort tartalmazó szenzorhálozat mintavételezéseinek pontos időbeli viszonyait is.

7.2. Továbbfejlesztési lehetőségek

Természetesen még számos lehetőség van az eddigi megépített szenzorhálózat továbbfejlesztésére. A továbbiakban az alábbiakat szeretnénk megvalósítani:

- Az egyik legfontosabb továbbhaladási irány további mérési elrendezésekben való kipróbálás, amellyel akár több slave esetében jobban megfigyelhető a jelenlegi protokoll korlátai.
- Az óraszinkronizáció pontosságának további javítása is egy lehetséges irány, hiszen vezetékes eljárások már ns-os pontossággal is képesek az órák szinkronizálására.
- Szeretnénk a saját protokoll működését is javítani. Az adatküldési sebesség növelése és a dinamikus ID osztás megvalósítása is hasznos lenne a protokoll használhatóságának szempontjából.
- A használt mérőrendszer szempontjából érdekes lehet, szenzorfúziós algoritmusok megvalósítása ebben az időszinkronizációs protokollban akár a slaven, akár a masteren is.
- Távoli cél lehet egy másik mérőrendszerrel való megvalósítás egy még időkritikusabb környezetben.

Köszönetnyilvánítás

A dolgozat az IncQueryLabs Kft. és az MTA-BME Lendület Kiberfizikai Rendszerek Kutatócsoport támogatásával készült el.

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] Ian F Akyildiz – Weilian Su – Yogesh Sankarasubramaniam – Erdal Cayirci: Wireless sensor networks: a survey. *Computer networks*, 38. évf. (2002) 4. sz., 393–422. p.

[2] Richard Cochran – Cristian 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, IEEE, 116–121. p.

[3] Analog Devices: Adxl345 datasheet (2016. október 20.). http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf.

[4] Gy-45 (mma8452) gyorsulásmérő szenzor (2016. október 20.). http://shop.tavir.hu/product_info.php/szenzor-irany-seb-magas-mma8452-gyorsulasmero-szenzor-p-179?osCsid=ksvsa3oethi35r70gelo629g24.

[5] Gy-521 (mpu6050) gyroszkóp, gyorsulás szenzor (2016. október 20.). http://shop.tavir.hu/product_info.php/szenzor-irany-seb-magas-521-mpu6050-gyroszkop-gyorsulas-szenzor-p-175.

[6] Gy-80 - multi sensor board - 3 axis gyro -3 axis accelerometer - 3 axis magnetometer - barometer - thermometer (2016. október 20.). http://selfbuilt.net/shop/gy-80-inertial-management-unit.

[7] Gy-86 (iránytű, gyorsulás, gyro, nyomás) kombinált szenzor (2016. október 20.). http://shop.tavir.hu/product_info.php/nyomas-magassag-gyro-iranytu-iranytu-gyorsulas-gyro-nyomas-kombinalt-szenzor-p-564?osCsid=5g8optdid0tvnn8sf1nad0sop1.

[8] Invensense: Mpu-6050 documents (2016. október 20.). https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/.

[9] Maciej Lipiński – Tomasz Włostowski – Javier Serrano – Pablo Alvarez: White rabbit: A ptp application for robust sub-nanosecond synchronization. In *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on* (konferenciaanyag). 2011, IEEE, 25–30. p.

[10] David L. Mills: Executive summary: Computer network time synchronization (2016. október 27.). https://www.eecis.udel.edu/~mills/exec.

[11] Dejan Nedelkovski: Mems accelerometer gyroscope magnetometer (2016. október 20.). http://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/.

[12] Minha Park: Error analysis and stochastic modeling of mems based inertial sensors for land vehicle navigation applications, 2004. 04.
URL http://www.ucalgary.ca/engo_webdocs/YG/04.20194.MinhaPark.pdf.

[13] Kovács Péter: Az öt érzék hatalma: inerciális érzékelés mems-alapú alkatrészekkel (2016. október 20.). http://www.elektro-net.hu/konstruktor/2152-az-ot-erzek-hatalma-inercialis-erzekeles-mems-alapu-alkatreszekkel.

[14] Michael Roche: Time synchronization in wireless networks (2016. október 27.). http://www.cs.wustl.edu/~jain/cse574-06/ftp/time_sync/index.html.

[15] Luis Ródenas: Arduino sketch to automatically calculate mpu6050 offsets (2016. október 20.). http://www.i2cdevlib.com/forums/topic/96-arduino-sketch-to-automatically-calculate-mpu6050-offsets/.

[16] NXP Semiconductors: Mm84521q datasheet (2016. október 20.). http://cache.nxp.com/files/sensors/doc/data_sheet/MMA8451Q.pdf.

[17] STMicroelectronics: L3g4200d datasheet (2016. október 20.). http://www.st.com/content/ccc/resource/technical/document/datasheet/04/46/d6/00/be/d9/46/ae/CD00265057.pdf/files/CD00265057.pdf/jcr:content/translations/en.CD00265057.pdf.

Függelék