MŰEGYETEM 1782

DIPLOMATERV-FELADAT

Marton Dániel (S4IBL3)
szigorló villamosmérnök hallgató részére

Programozható szenzorszimulátor fejlesztése

A modern gépjárművek biztonságtechnikai és kényelmi funkcióinak megvalósításában, környezetvédelmi jellemzőinek javításában stb. egyre jelentősebb szerepet kapnak a számítástechnikai megoldások. Ma egy prémium személyautóban közel száz elektronikus vezérlőegység (ECU) található, melyek számos, a környezet fizikai jellemzőit mérő szenzorral vannak kapcsolatban.

A vezérlőegységek teszteléséhez szükség van speciális eszközökre, melyek a jármű szenzorjait helyettesítik annak érdekében, hogy könnyen befolyásolható és megismételhető bemenetekkel gerjesszék a tesztelendő eszközt.

A szenzorok többféle módon kapcsolódhatnak a vezérlőegységekhez (analóg, digitális, különböző terepbuszok), de jelenleg a speciális, soros digitális interfészek a dominánsak.

A jelölt feladata egy szenzorszimulátor eszköz elkészítése, ami speciálisan az elektronikus kormányrendszerekben jellemző nyomatékszenzort helyettesíti a tesztkörnyezetben.

Részletesebben, a következő részfeladatokat kell megvalósítani:

- Az eszközzel kapcsolatos követelmények összegyűjtése
- A szenzor által használt kommunikációs protokoll (SENT, SPC) megismerése
- Az eszköz rendszertervének elkészítése
- A hardver részletes tesztelése, az elkészült eszköz illesztése
- Az alapfunkciók (szenzorjelek küldése) megvalósítása
- Hálózati kommunikáció megvalósítása az eszköz és a tesztrendszer többi eleme között
- Egyszerű jelalak-programozás megvalósítása (előre meghatározott jelalak megadása, amit az eszköz visszajátszik)
- Az elkészült eszköz tesztelése a célkörnyezetben.

Tanszéki konzulens: Dr. Sujbert László, docens
Külső konzulens: Dr. Balogh András (ThyssenKrupp Presta Hungary Kft.)

Budapest, 2016. március 19.

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Méréstechnika és Információs Rendszerek Tanszék

Dr. Dabóczi Tamás
tanszékvezető

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

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057. Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

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

Programozható szenzorszimulátor fejlesztése

Diplomaterv

Készítette
Marton Dániel

Konzulens
dr. Balogh András, dr. Sujbert László

2016. december 16.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Marton Dániel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. december 16.

Marton Dániel
hallgató

Kivonat

Jelen dokumentum egy autóipar által használ **SENT** és **SPC** protokollokkal kommunikáló konfigurálható szenzoremulátor leírását tartalmazza. Az emulátor feladata egy tesztkörnyezetben a különféle szenzorok helyettesítése, vezérelhetően működésének szimulálása. Az emulációt egy Texas Instruments **TivaC** mikrokontroller végzi, konfigurációja pedig történhet hálózaton keresztül, Etherneten. A komplexitásból adódóan az eszköz a **TiRTOS** valósidejű operációs rendszert futtat.

Az eszköz négy csatornás, így biztonságkritikus, mint például autóipari rendszerekben előforduló redundáns szenzorkonfigurációk jeleit szimulálni képes. Lévén alapvető funkciója a tesztelés megkönnyítése, így hálózaton vezérlehető, ezáltal automatizált tesztrendszer részét képezheti. Az ilyen alkalmazás további megkönnyítésére több hozzáférési szinten üzemeltethető. Alacsony szinten digitális jelgenerátorként adott granualitású digitális jel visszajátszása lehetséges. Középszinten a protokollok építőelemi paraméterezhetők, változtathatók, így különböző hibák beillesztése lehetséges, a protokollok fizika rétegjeleinek ismerete nélkül. Magas szinten a mérendő fizikai mennyiség leképezését nyújta felhasználó által definiált konfiguráció alapján **SENT/SPC** jellé.

A szenzorok és érzékelők rövid áttekintése után az implementálandó protokollokat bemutatva, mind hardveresen, mind szoftveresen az emulátor működését leírom, tervezési megfontolásokat ismertetem. A vezérléshez használt kommunikációs protokollt és kódolást bemutatom, majd a segítségével összeállított kommunikáció üzeneteit kifejtem.

Abstract

This document contains the descrition sensor emulator, utilizing **SENT** and **SPC** communication protocols, which among other fields of applications are used by the automotive industry. The purpose of the emulator to emulate various sensors in a test environment, while providing means to control the operational parameters of the aforementioned sensors. The emulation is done by a Texas Instruments **TivaC** microcontroller unit, while the configuration is done over the network on Ethernet. Due to its complexity the system runs **TiRTOS** realtime operating system.

The device is 4 channeled, so the simulation of redundant, safety critical sensor configurations (like the ones used by the automotive industry) is possible. The device is controllable over the network, since its basic function is to unburden the test process, making it part of an automated test-environment. The facilitate this kind of application the device can be operated on various access levels. A low level access operates it as a digital signal generator, synthesizing a digital signal. Mid level access enables building the protocoll signals from their parametrized building blocks, thus fault injection is possible, without the knowledge of the physical layer signals. High level access allows the mapping of real-world measurable quantities to a **SENT**/**SPC** signal, according to a user defined sensor configuration.

After a short introduction to sensors and transducers, I overview the protocols needed to be implemented. The subsequent chapters describe the emulator hardware and software, while explaining the various design considerations. Lastly the presentation of the communication protocol and encoding takes place, which is used to control the device, while presenting the used control messages.

Bevezető

Különféle szenzorok és érzékelők, mégha néha rejtve is alkalmazott eszközeink alap építőelemeit képzik. Szinte kivétel nélkül minden beágyazott valamilyen szinten használ szenzorokat a külvilágból információ nyerésére. A digitális technológia fejlődésével és az integráltság növekedésével egyre több korábban külön kezelt mérési és jelfeldolgozási feladat került integrálásra a szenzorokba, önálló **System-on-Chip** rendszereket képezve. Ez természetesen új szabványokat is eredményezett. Gyakran az alkalmazás megkönnyítése végett az ilyen rendszerek digitális kommunikációval közlik a mérési eredményeiket, státuszukat, illetve hasonlatosan konfigurálhatók.

A **SENT** és **SPC** két ilyen, elsősorban az autóipar által használt szenzoroknál alkalmazott protokoll. Kormányrendszer nyomatékérzékelésénél használt szenzorok is alkalmazzák őket. Egy tesztkörnyezetben hasznos ha a szenzor, akár hibás viselkedését emulálhatjuk.

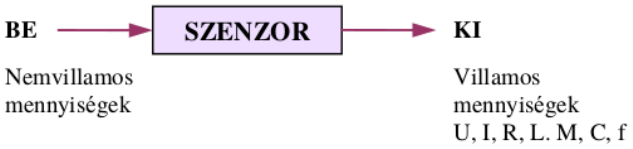
Diplomatervemben egy ezeket a protokollokat implementáló eszközt kívánok bemutatni, mely valós szenzorok alkalmazását helyettesítve, konfigurálhatóan képes a választott szenzor emulációját elvégezni, ezáltal a szenzort helyettesítve egy komplex rendszer tesztelésénél felhasználható.

1. fejezet

Szenzorok

1.1. Szenzorok és jelátalakítók

A *szenzorok* és *jelátalakítók(transducer, transzduktor)*, vagy együttes nevükön *érzékelők* feladata, hogy a környező fizikai világ tulajdonságairól, paramétereiről jeleket szolgáltassanak. A tulajdonság lehet fizikai, kémiai, biológia stb. jellegű, míg a kimeneti jel szintúgy (jellemzően azért fizikai kimeneti jelekkel foglalkozunk) . Szenzorokról beszélünk mikor a kimeneti jel egy elektromos mennyiség (tipikusan feszültség- illetve áramjelek) és átalakítókról míg más. Szakirodalomtól függően, a transzduktorokat szokás energiaátalakítóként is értelmezni mely egyfajta energiát konvertál másfélévé (pl.: lézerdióda is felfogható transzduktorként, elektromos energiát átalakítva koherens fénnyé). Továbbá előfordul a szenzoroknak, mint stimulusra reagáló eszközökként történő definiálása is.



1.1. ábra. *Általános szenzor*

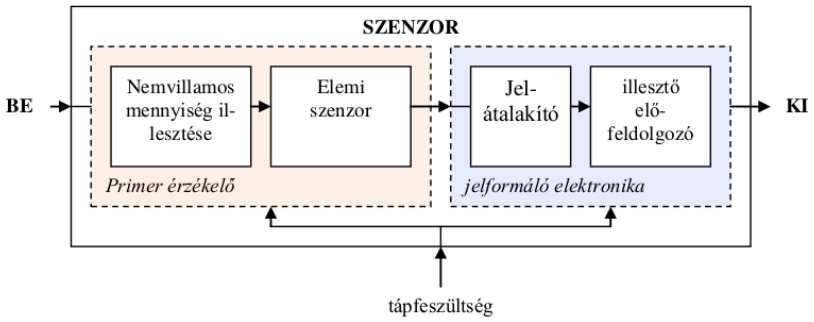
Mindkét esetben fontos jellemzőjük, hogy a bemeneti, a mérendő jel és a kimenet egymásnak kölcsönösen megfeleltethető.

Felépítését tekintve a szenzorok nem mások mint elemi érzékelők és jelformáló elektronikák összeintegrálása. Az elemi érzékelő különböző fizikai elvek hatások, törvények alapján méri a környezet jellemzőjét és alakítja villamos mennyiséggé. Ilyen elvek lehetnek:

- ellenállás-változás (megnyúlás, hőmérséklet, elmozdulás)
- induktivitás-változás (elmozdulás, erő)
- kapacitásváltozás (elmozdulás, anyagösszetétel, szint)
- termoelektromos feszültség (hőmérséklet)
- mágneses tér változása

- stb...

A mérendő mennyiséget természetesen az primer érzékelőnek és használt elvnek megfelelően illeszteni kell a bemeneten. A primer érzékelőcella jele továbbá a legritkább esetben használható fel direkt így jelkondicionálás szükséges a kívánt jellemzők és határok beállítására. Digitális szenzoroknál ezen felül szükséges a kvantálás és analóg-digitális átalakítás[6][20][29].



1.2. ábra. *Általános szenzor felépítése*

1.2. Statikus tulajdonságok

A fentebb említett megfeleltetés a kimenet és bementi jelek között a szenzor *statikus karakterisztikája*:

$$y = f(x) \tag{1.1}$$

Ez alapján felírható a bemenet Δx megváltozásának hatására a kimenet Δy változása, vagyis az elsőrendű deriváltja, ami szenzor *érzékenysége*:

$$S = \lim_{\Delta x \to 0} = \frac{\Delta x}{\Delta y} = \frac{dx}{dy} \tag{1.2}$$

Ennek megfelelően a szenzor *méréstartománya* definiálható. Az a tartomány ahol az előírt érzékenységet teljesíti a szenzor:

$$[x_{min}; x_{max}] \; ha \; x_{min} < x < x_{max} \Rightarrow S(x) > S_{min} \tag{1.3}$$

[6]

1.3. Dinamikus tulajdonságok

Változó bemenetek hatására történő viselkedés leírása. A különböző energiatárolási mechanizmusok (pl.: inercia, elektromos illetve hőkapacitás) hatására az ideális nulladrendű átviteltől eltérő viselkedést tapasztalhatunk. A szenzor előélete hatással van a kimenetére. Ennek megfelelően mint dinamikus rendszer átviteli függvényével leírható. Amennyiben számítása nehézkes, akár a különböző bemeneti gerjesztésekkel (egységugrás, szinusz stb...)

stimulálva mérhető. Dinamikus rendszerként így a sávszélesség, egyik fő jellemzője, mely leírja a kimenet milyen sebességű, ütemű változásaira képes reagálni megfelelően.
[6]

1.4. Valós szenzorok paraméterei

Valós, alkalmazott szenzornál a fenti jellemzőkön kívül továbbiak vannak definiálva, a működés pontosabb leírása érdekében. Ezek egy része hibamennyiség, egy kimért szenzorkarakteriszitika és az elméleti érzékenység karakterisztika eltérését jellemzik. Mivel gyártónként elérések lehetnek egyes hiba jellemzők mérési módjában és megadásában, így érdemes lehet az adott értékhez tartozó a gyártól által használt eljárást és a számítás módját áttekinteni.

A hibák eredetüket tekintve lehetnek **rendszeres** és **véletlenek**. Megadási módjukat tekintve pedig **abszolút** és **relatívek**. Relatív hibák megadásakor a hibát a végérték %-ban adják meg, de előfordulhat a mérési tartomány meghatározott részére vonatkoztatott relatív hiba. Előfordul továbbá a hibasáv ami a mérendő mennyiség egész tartományára érvényes maximális hiba, illetve ennek a végkitérésre vonatkoztatott százalékos értéke, az **osztálypontosság**.

Linearitás megadja, egy lineáris szenzor karakterisztikája esetén a ideális egyenestől való eltérést, míg a **monotonitás** a függő kimeneti változó kvalitatív viselkedését a független hatására. Az **offszet** megadja jellemzően a mért értékekre illesztett egyenes és valós egyenes eltérését az origóban.

Bár ideális esetben egy kimeneti értékhez egy bemeneti érték tartozik, ez nem minden esetben lehetséges különféle okokból (pl.: ferromágneses anyagok anyagi tulajdonságai). Egyazon bemenethez tartozó két különböző kimenet leírására a **hiszterézis** szolgál.

Az **érzékenység** mellet megjelenik a **kereszt-érzékenység**, vagyis érzékenység nem csak a mérendő mennyiségre, hanem tőle független, ortogonális környezeti jellemzőkre is.

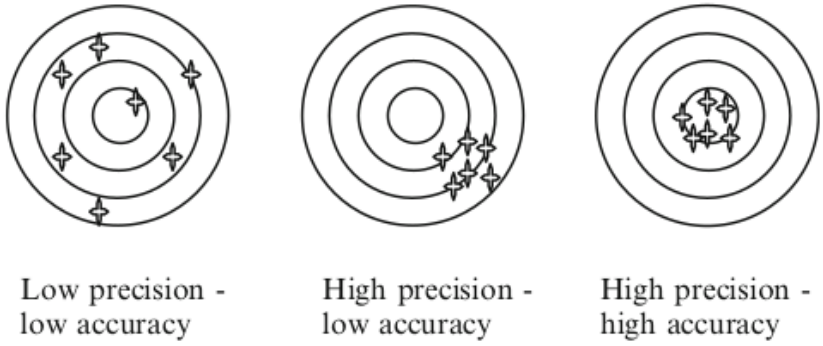
Sajnos a magyar nyelvben az angol **precision** és **accuracy** szavakat tipikusan a pontosság szinonímájaként kezeljük, így a továbbiakban **pontosság(accuracy)** és **precizitás(precision)**-ként hivatkozok rájuk.

A **pontosság(accuracy)** megadja kimenet eltérését a valós mért értéktől, vagyis abszolút értelemben véve milyen pontos az eszközünk. Mérése standardhoz hasonlítással történhet.

$$pontossag = 1 - \frac{x_{hiba}}{x_{valos}} = 1 - \frac{|x_{mert} - x_{valos}|}{x_{valos}} \tag{1.4}$$

A **precizitás(precision)** viszont a megismételhetőség, reprodukálhatóság mértéke. Vagyis a mérést változatan feltételek mellet megismételve mennyire szóródnak a mért értékeink, tehát statisztikai jellegű paraméter, ami a szórással arányos.

$$precizitas = \frac{merestartomany}{\sigma} \tag{1.5}$$



1.3. ábra. *Pontosság és precizitás*

A **felbontás(resolution)** a statikus érzékenységből származik, megadja a statikus karakterisztikát milyen lépésközökkel tudjuk olvasni. Egy valós szenzoron méréssel felvett karakterisztikából a mérési tartományban lévő ugrások számának reciprokával fejezhető ki. Digitális szenzoroknál tipikusan a kvantálás befolyásolja, 1 bitet mekkora változásnak feleltetünk meg.

Küszöbértékkel(threshold, MDS, minimum-detectable-signal) adjuk meg hogy a mérendő mennyiség mekkora megváltozása szükséges az érzékeléséhez.

Dinamikus jellemzők leírásánál a már ismertet átviteli függvény gyakorlatiasan kezelhető paramétereinek megadása a jellemző. A **beállási idő** megadja hogy egységugrás bemeneti gerjesztés hatására mennyi idő szükséges a kimeneti jelnek a $100 \pm \delta$ sávba érkezéséhez és ottmaradásához. A δ szokásos értéke 5 illetve 2%, ez alapján beszélhetünk 5 és 2%-os beállási időről. A szenzor tehetetlenségéből fakadó időkésleltetés jellemzésére használják ezen kívül a **félértékidőt** és az **időállandót** is. Digitális szenzorok esetében ennek megfeleltethető a **válaszidő**, vagyis lekérdezése milyen gyorsan képes az eszköz válaszolni. A **túllövés** a szenzor csillapítási viszonyiról ad információt.

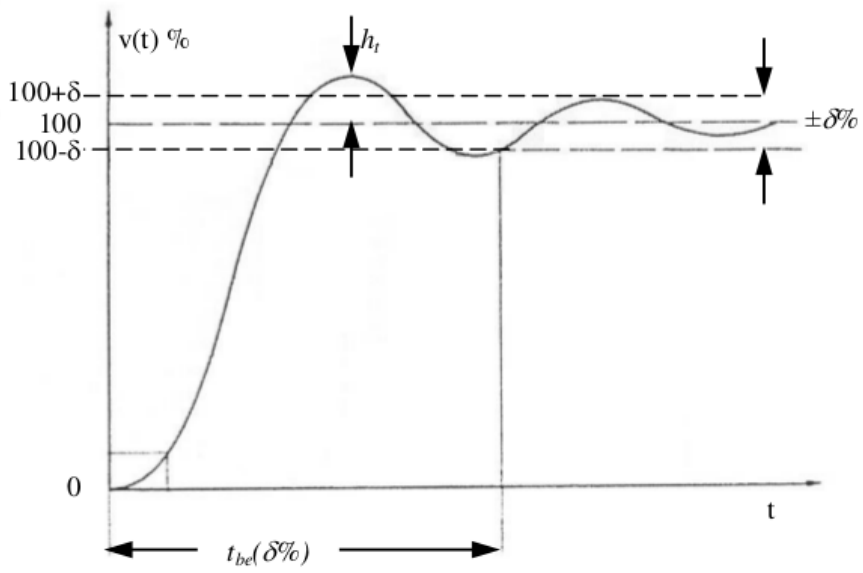
Driftnek nevezzük amikor változatlan bemenet mellet a kimenet fokozatosan változni kezd. A drift tehát a mérendő mennyiségtől független nemkívánatos változás. Rendszer hibaként kategorizálható, amit interferáló paraméterek okoznak mint szennyeződés, mechanikai és hőmérsékleti instabilitás, az eszköz öregedése stb... [6][29]

1.5. Szenzorok csoportosítása kimenet alapján

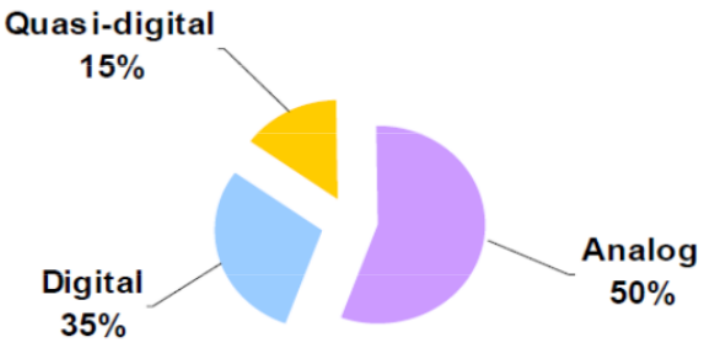
A szenzorok csoportosításának egy lehetséges módja a kimenet alapján történik. Bár egyre inkább terjedőben vannak a különféle digitális szenzorok, bizonyos gyakori alkalmazásokhoz továbbra is elegendő egyszerűbb analóg érzékelők használata[29][4].

1.5.1. Analóg kimeneti jelű szenzorok

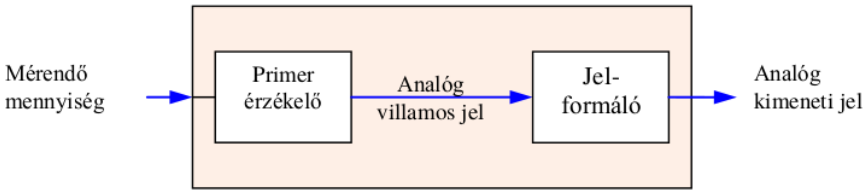
Egyszerű elemi cella és analóg jelkondicionáló elektronika kapcsolata. Kimeneti jele tipikusan feszültség vagy áramjel (távadó). Ilyen szenzorok pl.: ellenálláshőmérők, erőmérő cellák, nyúlásmérő bélyeges nyomás és elmozdulásmérés[29].



1.4. ábra. *Analóg szenzor ugrásválasza, dinamikus viselkedése*



1.5. ábra. *International Frequency Sensor Association (IFSA) tanulmánya alapján az alkalmazott szenzorok, 2011*

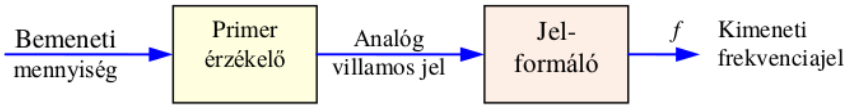


1.6. ábra. *Analóg jelű szenzor*

1.5.2. Frekvencia kimenetű szenzorok

Másnéven pszedudó-digitális szenzorok. A kimeneti jelfrekvencia hordozza az információt. Lehetnek közvetlen és közvetett átalakításúak. A mérés paramétereitől és környezetétől függ hogy a jelkondicionálás elhagyható-e. Ilyen szenzorok lehetnek: turbinás áramlásmérő, rezgőkvarcos nyomás és hőmérsékletmérés. Közvetett átalakítású szenzorok lehetnek például különböző kapacitív érzékelők[29].

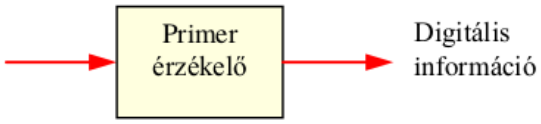
1.7. ábra. *Közvetlen átalakítású frekvenciakimenetű analóg szenzor*



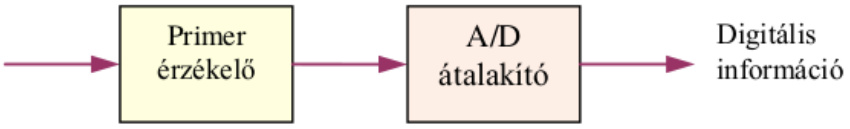
1.8. ábra. *Közvetett átalakítású frekvenciakimenetű analóg szenzor*

1.5.3. Digitális kimenetű szenzorok

Digitálisan kódolt információt továbbítanak. Szintén lehetnek közvetlen és közvetett átalakításúak. Közvetlen átalakítású például a kódtárcsás abszolút elmozdulás érzékelő, inkrementális adó. Közvetett a legtöbb digitális kimenetű, "intelligens" szenzor[29].



1.9. ábra. *Közvetlen átalakítású digitális szenzor*

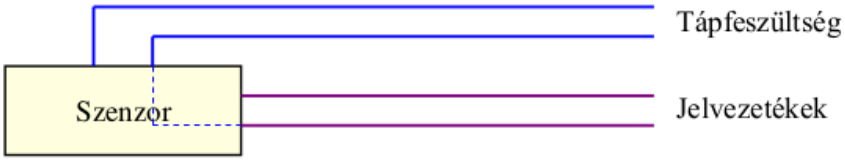


1.10. ábra. *Közvetett átalakítású digitális szenzor*

1.6. Szenzorok csoportosítása vezetékezés alapján

1.6.1. 4 vezetékes

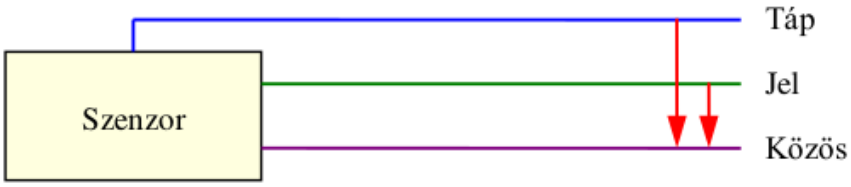
Független táp és jelvezetékekkel tápzavarok hatása csökkenthető, pontosabb analóg jelet biztosít. Hátránya a vezetékezés költsége[29].



1.11. ábra. *4 vezetékes szenzor*

1.6.2. 3 vezetékes

Jel és táp nem független, kapcsolt a referencián keresztül, ennek zavarai a jelre hatnak. Vezetékezés költsége csökken[29].



1.12. ábra. *3 vezetékes szenzor*

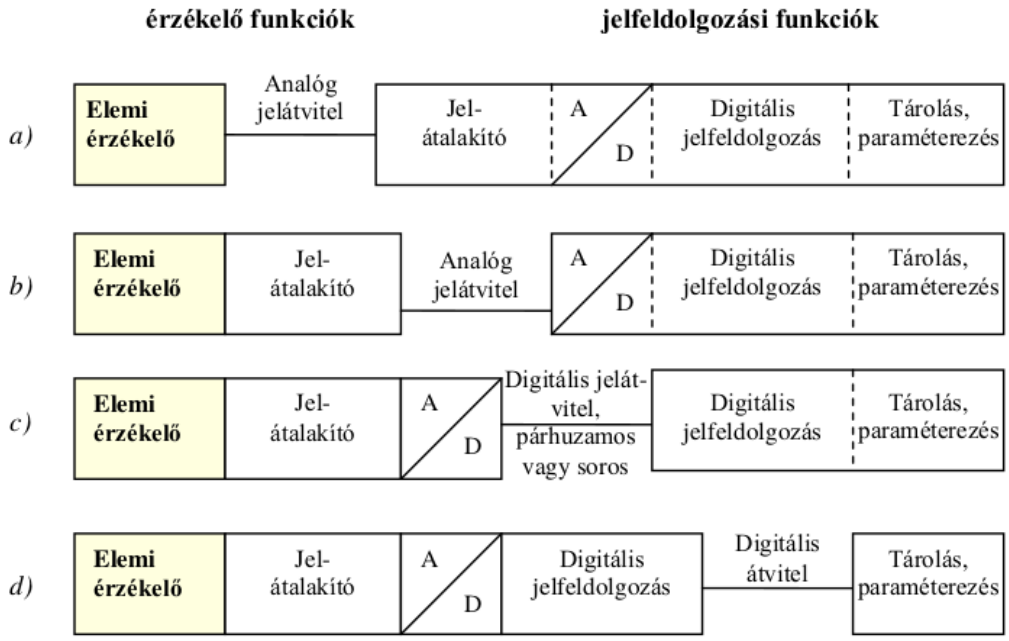
1.6.3. 2 vezetékes

Jel és táp nem független, ugyanazokon a vezetékeken futnak, az információ szuperponálódik a tápjelre. Vezetékezés költsége tovább csökken[29].

1.13. ábra. *2 vezetékes szenzor*

1.7. Funkciómegosztás

A nagyobbfokú integráltságú IC-k terjedésével a szenzorok is intelligensebbé váltak, különböző jelfeldolgozási, konfigurálhatósági és kommunikációs lehetőségeket kezdetek el biztosítani. Ennek megfelelően a korábban külön az adatgyűjtő és vezérlő rendszerben implementált feladatok fokozatosan áttevődtek a szenzorba fokozatosan a feldolgozási lánc egy újabb elemét integrálva. Emiatt a modern szenzorok viszonylag könnyen interfészelhetők különböző rendszerekkel feltéve hogy az egyes szintek átvitele szabványosan van megvalósítva. Mikroprocesszor integrálásával System-on-Chip (**SoC**) és System-in-Package (**SiP**) elrendezésben komplex megoldáscsomagot tudnak digitális rendszerek számára biztosítani. Processzormagnak hála önteszt, validáció és adaptációra is lehetőséget tudnak biztosítani. Tipikusan ezeket szokták intelligens-szenzoroknak titulálni[29][4].



1.14. ábra. *Funkciómegosztás*

2. fejezet

Szenzorok digitális kommunikációja

A digitális szenzorok elterjedésével igény nyílt mind gyártók, mind felhasználók felől egységes protokollok és interfészek kialakítására. A meglévő különböző soros ($I^2C, SPI stb...$) protokollok használhatók voltak de sokszor nem biztosítottak egyes alkalmazási területeknek igényeit kielégítő megoldásokat (hibatűrés, elektromágneses kompatibilitás). [11][17]

2.1. SENT protokoll

Az autóipar igényeit hivatott kielégíteni a **SAE J2716** számú szabványa mely a **Single Edge Nibble Transmission (SENT)** protokoll leírását tartalmazza.

Célja egy a korábban alkalmazott 10 bites A/D-s és PWM-es megoldások helyett egy magasabb felbontású költséghatékony alternatíva adása **elektromos vezérlőegység (ECU)** és szenzor kommunikációjára, mely CAN és LIN-nél egyszerűbb.

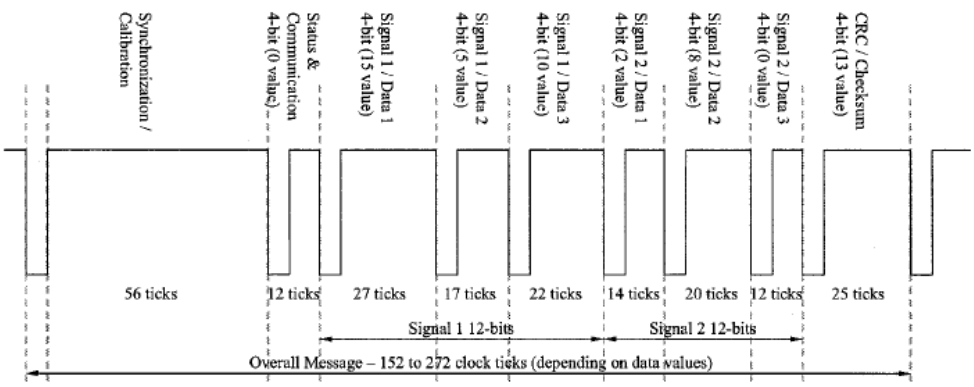
Az implementáció megköveteli, hogy a szenzor "okos" legyen mikroprocesszort tartalmazzon, vagy neki megfelelő alkalmazásspecifikus logikai IC-t (**ASIC**). [17][31][8][15][19][9][18]

A protokoll főbb jellemzői:

- A kommunikáció egyirányú, a szenzor mint küldőtől, az **ECU** mint fogadó felé. A információt pulzussorozatok kódolják mint a lefutó élek között időtartam.
- Vezérlő/koordinálójel a vevőtől nem szükséges.
- Az implementáció bizonyos részletei a gyártóra vannak bízva. Ez flexibilitást biztosít, de egy általános **SENT** protokoll megvalósító eszköz fejlesztését nehezíti.
- Az átvitel a vevőtől független, nem szükséges szinkronizáció, ez nincs is biztosítva.
- Az átvitel ideje változó, függ a küldött üzenet adattartalmától, és az adószenzor órajelbizonytalanságától.
- A üzenet pulzussorrendje fix minden adó számára.
- Az adó számára ±20%-os órajeleltérés megengedett.
- Az időzítés alapegysége a tick, egy adóspecifikus órajel 3 és 10 mikrosecundum között.

2.1.1. SENT üzenet

A SENT üzenet ütemezésének alapegysége a **tick**. Az üzenet részei ennek megfelelően tickhosszokban adottak[17][31][18].



2.1. ábra. *SENT üzenet*

Kalibrációs/Szinkronizációs pulzus

Szerepe hogy a vevő a tick-időt megmérhesse, ennek megfelelően az üzenet többi részét értelmezni tudja. Ennek megfelelően hossza fix, 56 tick-nyi. Szabvány szerint minimálisan 4 ticknyi (maximális nem definiált) ideig a vonalat alacsonyra kell az adónak hajtania, a hátra lévő időre magasra. A vevőnek ezt kötelező megmérnie, az adó számára biztosított nagy órajeleltérés miatt[17][31][18].

Adat pulzusok

Az adat pulzusok neve **nibble**, egységes tulajdonságokkal rendelkeznek, változó hosszúak. Minimális idejük 12 tick, maximális 27. Minimálisan 4 ticknyi (maximális nem definiált) ideig a vonalat alacsonyra kell az adónak hajtania. Az adatot a nibblehossz tartalmazza. 1 nibble 4 bitnyi információt kódol lineárisan. Tehát a minimális 12 tick hosszú nibble az 0x0, míg a maximális 27 hosszú az 0xF.

A nibbleök száma alkalmazásonként változó, de egy adott kódolási sémán belől fix. Például 2 darab 12 bites érték átvitele 6 adat nibble-t eredményez. Bájtsorrend nem definiált[17][31][18].

Státusz/Kommunikációs pulzus

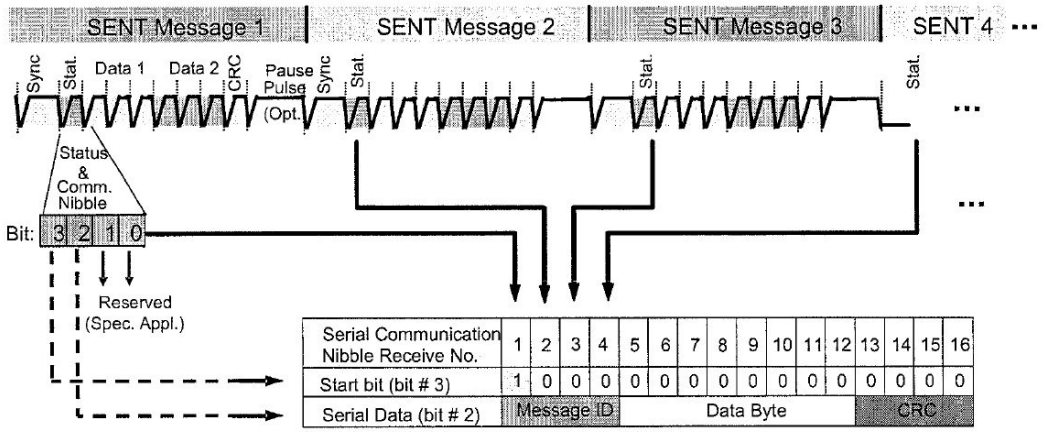
Az adat nibble-öknek megfelelő nibble speciális jelentéssel az egyes biteken. 0 és 1-es pozíciójú bitek gyártóspecifikus implementációt tesznek lehetővé (hibajelzés, méréstartomány-jelzés). 3-as pozíciójú bit soros üzenet kezdetét jelzi a 2-es pozíciójú biten (lassú csatorna, több SENT üzenetbe ágyazott 16 bites soros üzenet)

A beágyazott 16 bites üzenet 4 bit azonosítóval, 8 adatbittel és 4 bites ellenőrző összeggel rendelkezik. Az azonosítók és hozzájuk tartozó adatbitek értelmezése gyártóspecifikus.

Bit Number	Bit Function
0	Reserved for specific application
1	Reserved for specific application
2	Serial Data message bits
3	Message start = 1, otherwise = 0

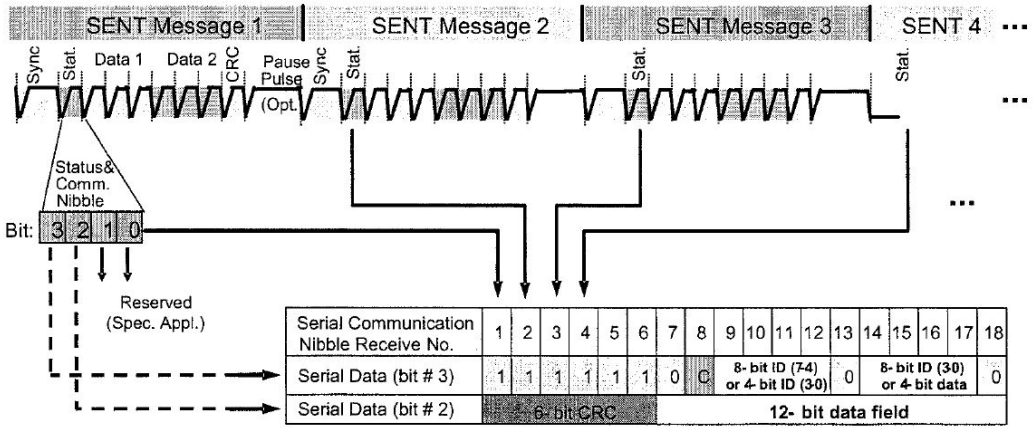
2.2. ábra. *Státusznibble tartalma*

Az ellenőrző összeg számítása a 2.1.1 alapján történik. A teljes 16 SENT üzeneten keresztüli soros adatnak sikeresen kell megérkezni, különben elvetésre kerül. A bitek sorrendje MSB.[17][31][18]



2.3. ábra. *Soros üzenet lassú csatornán*

Enchanced Státusz/Kommunikációs pulzus

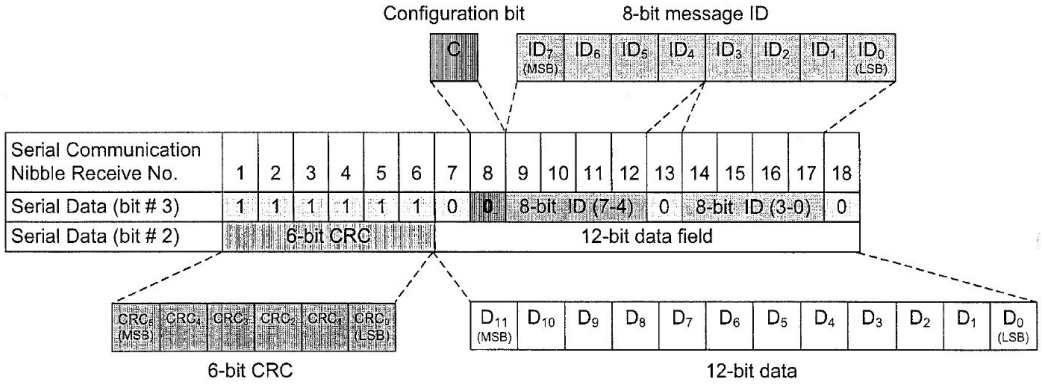


2.4. ábra. *Enchanced soros üzenet lassú csatornán*

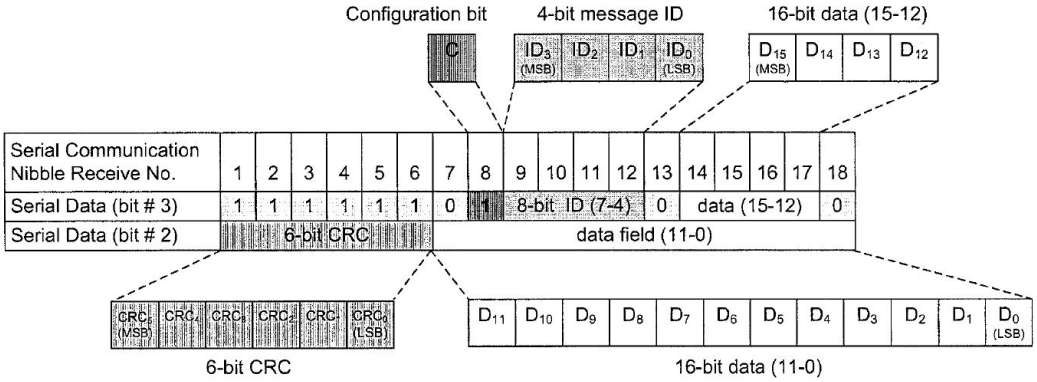
A 2010-es reviziója a szabványnak egy opcionális fejlesztett enchanced státusz/kommunikációs pulzust hozott be. Ez a pulzus továbbra is egy üzenetenként egy nibbleként kerül továbbításra, de 18 egymást követő **SENT** üzenet során. Az üzenet kezdetét a nibble 3-

dik bitjén a 01111110 sorozat jelzi, szemben a korábbi egyszerű startbittel. Ezt követi a konfigurációs bit, mellyel kétféle konfigurációt állíthatunk: [18]

- C = 0, 8 bites azonosító, 12 bites adat
- C = 1, 4 bites azonosító, 16 bites adat



2.5. ábra. *Enchanced soros üzenet lassú csatornán, C = 0*



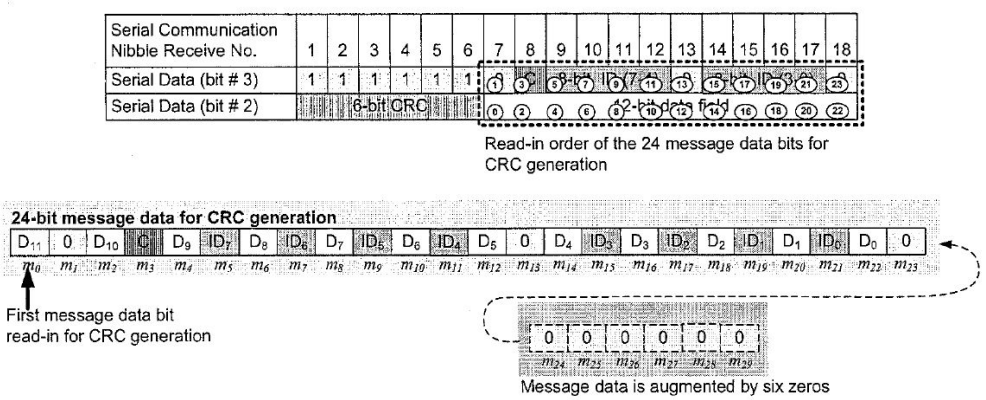
2.6. ábra. *Enchanced soros üzenet lassú csatornán, C = 1*

Az alkalmazott **CRC** is látható módon eltér, lévén 6 bites. A státusznibble 2-dik és 3-dik bitjeiből képződik, a 7-18 as keretből (tehát a CRC előbb megérezik mint a hozzátartozó adat). A generátorpolinóm a $x^6+x^4+x^3+1$ az $0b010101 = 0x15 = 21$-es seed-el. Összesen 24 hasznos, az üzenetben is szereplő bitből számolódik az ellenőrzőösszeg kiegészítve hat nullával. $m = [m_0, m_1, \ldots m_{23}, 0, 0, 0, 0, 0, 0]$ [18]

Ellenőrző-összeg pulzus

Az ellenőrzés 4 bites **CRC**-vel történik, mind a beágyazott soros üzenetekre a lassú csatornán, mind az adatpulzusokra. Adatpulzusok **CRC** számításakor a Státus/Kommunikációs nibble tartalma nem kerül felhasználásra. A generátorpolinóm a $x^4 + x^3 + x^2 + x$ az $0b0101 = 5$-es seed-el.

Ez a következő hibák detektákását teszi lehetővé:



2.7. ábra. *Enchanced soros üzenet, CRC számítás*

- Minden egybit hiba.
- Minden páratlan számú hiba.
- Minden 4 vagy rövidebb hosszúságú burstös hiba.
- 5 hosszúságú burstös hibák 87.5%-a.
- 5 nél hosszabb burstös hibák 93.75%-a.

A legveszélyesebbek azon hibák melyek az élváltás eltolódását eredményezik. Ehhez két szomszédos nibble egyikében a hosszabbodik (rövidül), míg a rákövetkező nibble ezzel ellentétesen változik. A használt polinóm nem minden hibát detektál 3, 5, 7, 11 tick hosszú eltolódásoknál. A viszonylag magas frissítési frekvenciát tekintve ez elfogadható. A nem detektálható bitkombinációkat a következő tábláza tartalmazza[17][31] [18]:

Pause pulzus

Amennyiben az szenzor az adatot nem folytonosan szolgáltatja, hanem fix időközönként, ekvidisztánsan, a két egymást követő üzenet közötti pulzus felfogható pause pulzusnak. Ez szükséges hogy a CRC nibble végét detektálni lehessen, hiszen ehhez élváltás szükséges. Az ilyen pulzusok az üzeneteket fix tickszámúra, ezáltal fix hosszúságúra egészítik ki.

A szabványba a 2010-es revízió során került bele. Követelmény feléje hogy minimálisan 12 tick hosszú (egy 0 adattartalmú nibblenek megfelelő), maximálisan 768 (3 * 256) tick hosszúságú legyen. A választott teljes hossz értelemszerűen a hasznos üzenet minimális és maximális hosszától függ (pl.: 6 adatibble-t tartalmazó üzenethez minimálisan 282 tick hosszú, ekkor a pause pulzus 12-128 tick között változik). Folytonos üzenetküldés során ez nem szükséges, sőt az elérhető sávszélességet csökkenti. [17][31][18]

Fogadott üzenet diagnosztikája

Bármely következő hiba esetén a fogadott üzenet elvetésre kerül, és a fogadó azonnal új szinkronizációs pulzust és ezzel új üzenetet kezd keresni[17][18]:

Nibble 1 Error Pattern		Nibble 2 Error Pattern	
8	1000	1	1
13	1101	2	10
5	101	3	11
7	111	4	100
15	1111	5	101
10	1010	6	110
2	10	7	111
14	1110	8	1000
6	110	9	1001
3	11	10	1010
11	1011	11	1011
9	1001	12	1100
1	1	13	1101
4	100	14	1110
12	1100	15	1111

2.8. ábra. *Nem detektált eltolódások bitkonfiguráció*

- Kalibrációs pulzus megsérti a $56 \pm 25\%$ tick-időnyi határokat. (Bár ez a határ magasabb az órajel 20%-os toleranciája, csak az órajelfeltételeket kielégítő eszközök kvalifikálta SENT-re)
- Az előírt sémától eltérő számú nibble-ek 2 szinkronizációs pulzus között
- Ellenőrző-összeg hiba
- Egymást követő kalibrációs pulzusok 1.5625%-ot meghaladó eltérése
- Logikai 0-15 tartományon kívüleső nibbleök

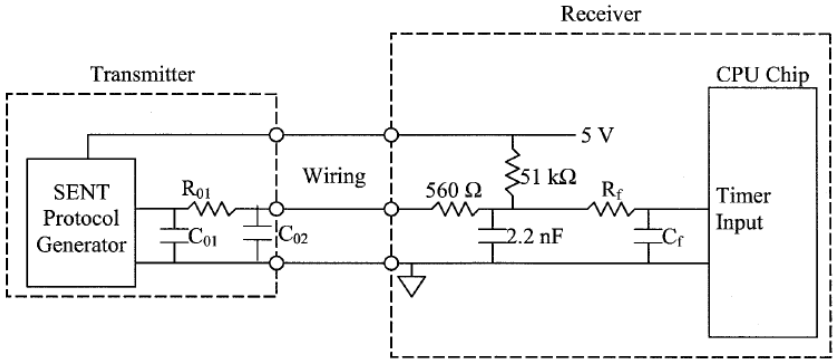
2.1.2. SENT fizikai rétege

Interfészelését tekintve a SENT fizikai rétege a 3-vezetékes kategóriába sorolható, 5V-os maximálisan 20 mA es jelekkel. Az előírt elektromágneses kompatibilitás biztosítására a kimeneten jelformálás van alkalmazva (kontrollált felfutás és aluláteresztő szűréssel a nagyfrekvenciás komponensek eltávolítása, "élkerekítés").

A küldő eszköznek feladata hogy az előírt passzív fogadó terhelést megfelelően meghajtsa, az előírt fel és lefutási időknek megfelelően az előírt jelszinteken, áramokkal:

A fogadóeszköznek kell a vonal felhúzását, egy szabályozható küszöbű digitális vevőt és időmérés megvalósítását biztosítani dekódoláshoz. Az előírt fizikai paraméterek:

A fogadó feladata továbbá a szenzor tápellátásának biztosítása, a következő paraméterekkel: [17][31][18]



2.9. ábra. *SENT interfész áramkör*

Parameter			Limits Min	Limits Max	Units	Test Conditions / Definitions
a.	V_{OL}	=		0.5	V	Low state Voltage @ 0.1 mA DC load current
b.	V_{OH}	=	4.1		V	High state Voltage @ 0.1 mA DC load current
c.	I_{SUP}	=		20	mA	Average current consumption from Receiver supply over one message
d.	$I_{SUP-RIPPLE}$	=		9.0	mA	Variation in supply current consumption
e.	T_{FALL}	=		6.5	µs	From 3.8 V to 1.1 V on wire
f.	T_{RISE}	=		18	µs	From 1.1V to 3.8V on wire
g.	ΔT_{FALL}	=		0.1	µsSec	Edge to edge jitter with static environment for any pulse period
h.	T_{STABLE}	=	6		µs	Signal stabilization time below V_{IL} (low signal) or above V_{IH} (high signal)

2.10. ábra. *SENT követelmények küldőre*

Parameter			Limits Min	Limits Max	Units	Test Conditions / Definitions
a.	$R_{PULL-UP}$	=	47000	55000	Ohms	Input pull-up resistance
b.	C_{INPUT}	=		0.1	nF	Parasitic input capacitance
c.	V_{IL}	=	1.39		V	Input low state threshold range
d.	$V_{IL-DELTA}$	=	-50	50	mV	Maximum input low state threshold variation over 1 millisecond interval with supply Voltage constant
e.	V_{IH}	=		3.8	V	Input high state threshold range
f.	Tau1	=	0.74	1.73	µs	Input filter time constant – first stage
g.	Tau2	=	0.6	1.4	µs	Input filter time constant – second stage
h.	V_{HYST}	=	0.3		V	Input electrical hysteresis

2.11. ábra. *SENT követelmények fogadóra*

Parameter			Limits Min	Limits Max	Units	Test Conditions / Definitions
a.	V_{OUT}	=	4.85	5.15	V	Supply Voltage
b.	I_{OUT}	=	20		mA	Available current for Transmitting device
c.	C_{LOAD}	=		10	µF	Supply to return load capacitance in transmitter
d.	$V_{GND-OFFSET}$	=		0.02	V	Signal return ground offset in Receiving device
e.	$V_{3.3SUPPLY}$	=	3.234	3.366	V	Internal 3.3 V supply (if needed by Receiver)

2.12. ábra. *SENT tápkövetelmények*

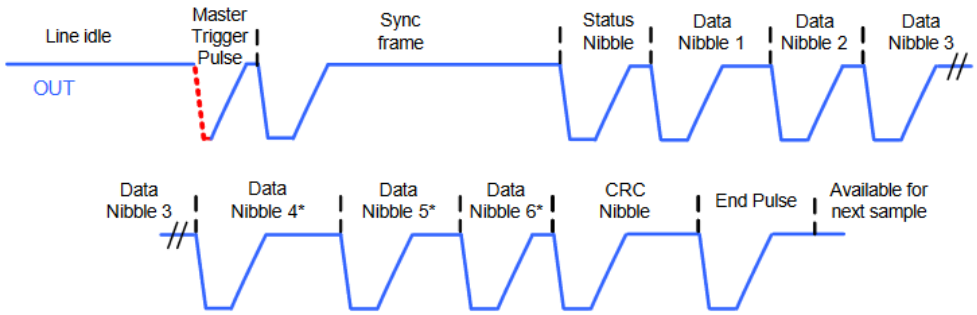
2.2. SPC protokoll

A **Short PWM Code (SPC)** protokoll felfogható a **SENT** egyfajta funkcionális kibővítésének. SPC segítségével kétirányú, fél-duplex kommunikáció és szinkronizáció valósítható meg. Fontos megjegyezni, hogy míg a **SENT**-et szabvány definiálja addig az **SPC**-t nem, így bár szinte teljesen hűen igyekszik követni a **SENT** előírásai, gyártónként eltérések lehetnek tőle[7][12][9].

2.2.1. SPC üzenet

Az küldött **SPC** üzenet teljes mértékben megyegyezik a **SENT** során definiáltal, az eltérés a küldés módjában van. Míg a **SENT** egyirányú kommunikáció, a szenzor folyamatosan szolgáltatja az adatot, függetlenül a fogadótól, addig az **SPC** során a fogadó kezdeményezi az adatátvitelt az adatvonal lehúzásával. (Ennek egyik további hatása hogy interframe/-pause pulzus mindig kerül az SPC üzenetbe)

Ilyen módon az **SPC** felfogható egy triggerelt **SENT** üzenetnek. A triggerelt működésből fakadóak több szenzor is köthető egy vonalra, hiszen az vonalmeghajtást a master engedélyezi, kezdeményezi. A triggerpulzus hossza változó lehet és információt hordoz. Ez gyártóspecifikus, szenzorfüggő (jellemzően range-választás, vagy szenzorválasztáshoz azonosító). [7][9]

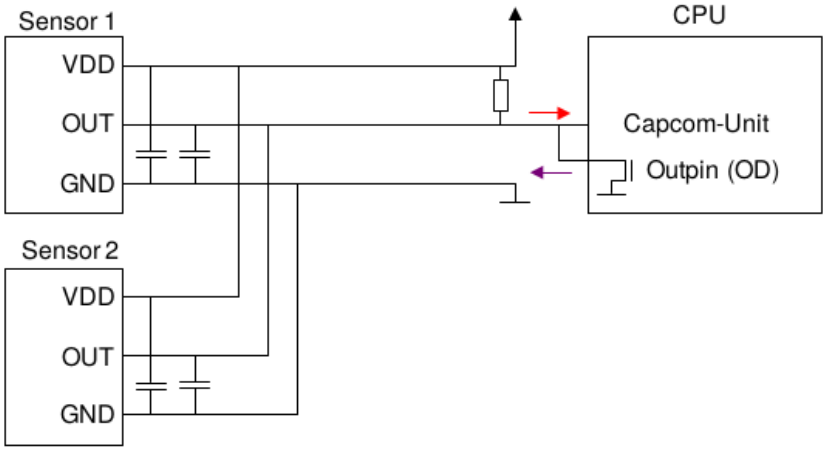


2.13. ábra. *SPC üzenet*

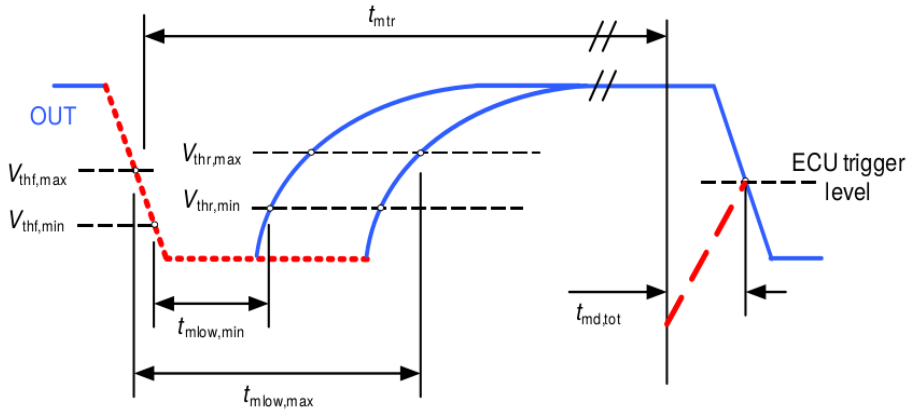
A mivel az **SPC** nem szigorúan szabványos ezért a Státusz/Komminikációs (2.1.1) nibble tartalma is eltéréseket mutathat a **SENT**-hez képest, illetve a **CRC** számítás módja is eltérhet, gyártónként (és el is tér).

2.2.2. SPC fizikai rétege

A **SENT**-el való kompatibilitás végett a gyártók igyekeznek a **SENT** által definiált fizikai követelményeket betartani, sokszor ugyanaz a szenzor **SENT** és **SPC** módra is képes. Amivel kiegészítésre szorul a SENT szabvány az a triggerpulzus definiciója. Mivel a vonal fizikai paraméterei már definiáltak, így csak a pulzus időzítési viszonyait kell megadni (hossz, komparálási küszöb, fel-lefutás), ez viszont már gyártóspecifikus. [7][9]



2.14. ábra. *Több SPC szenzor egy vonalon*



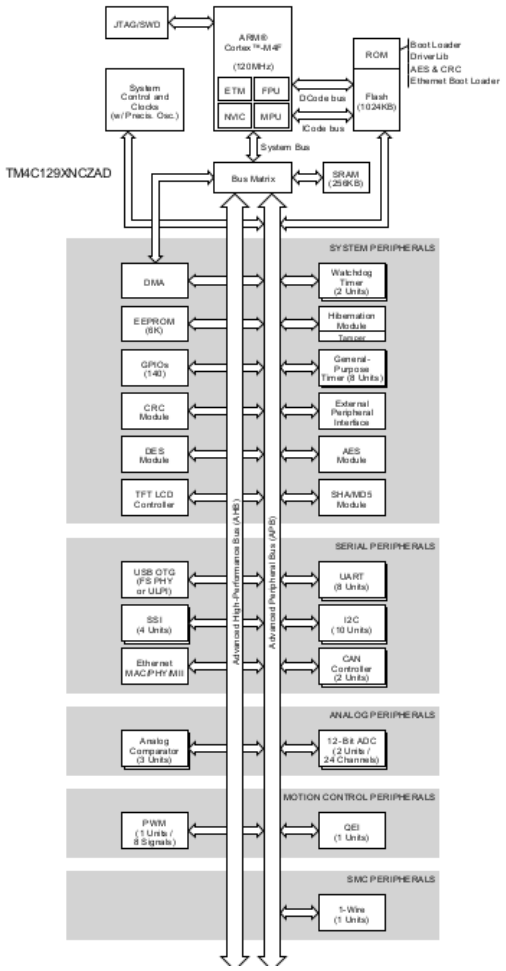
2.15. ábra. *SPC triggerpulzus jellemző paraméterei*

3. fejezet

Felhasznált eszközök

3.1. Mikrokontroller

A fejlesztés egy Texas Instruments **TivaC** családú mikrokontrolleren történt (**TM4C129XNCZAD**), a TI által készített fejlesztői kártyán. Ez egy Cortex-M4F magú, viszonylag bő periféria-készlettel rendelkező kontroller. [24]



3.1. ábra. *A használt mikrokontroller*

A fontosabb alkalmazott perifériak a hatékony digitális jelszintézishez és a vezérlő kommunikációhoz a timerek, soros kimeneti vonalak (**QSSI**), direkt memóriahozzáférés vezérlő (**uDMA** kontroller) és ethernet. Ezek közül mindet tartalmazza.[24]

A digitális jelszintézis megvalósításához a perifériakészet alapján több lehetőség is kínálkozott, de a későbbi az alkalmazásnak megfelően hibainjekció támogatására igyekeztem tetszőleges jelalakszintézist biztosítani 4 csatornára, nem csak az implementált protokollnak megfelelő jelalakok előállítását.

Tick-Timer alapú bitbangelt GPIO

Timert tickidőzítésre felhúzva általános IO pin állítgatása, egy memóriában tárolt mintareprezentáció alapján. A megoldás hátránya hogy gyakran kerül timerinterrupt végrehajtásra, hiszen a tickidő kicsik lehetnek ($3\mu s$). Operációs rendszerrel eggyütt használva így különösen nagy overheadet eredményez, míg az oprendszer az interruptok könyvelését elvégzi. A minimális interruptszolgáltatást (reentrancia tiltás stb...) beállítva az operációs rendszeren is alig volt képes teljesíteni az időzítési követelményeket. A tick interruptkezelését az operációs rendszer felügyelet alól kivéve valószínűleg lehetséges, de ez erősen ellenjavalott lévén az operációs rendszer időzítésit így befolyásolom, ami valósidejű oprendszernél rossz ötlet. Ezt a megoldást elvetettem.

Élváltás-Timer alapú bitbangelt GPIO

Az szintetizált digitális jel élváltásaira működő timerek által általános IO pin vezérlés. Tetszőleges jelalak szintézis továbbra is nehézkes, a timereken folytonosan újra kell számolni az élváltásokat. Gyakori, szélsőséges esetben tickenkénti váltáskor továbbra is fennáll a korábban ismertetett hardvermegszakítások korlátozó hatása.

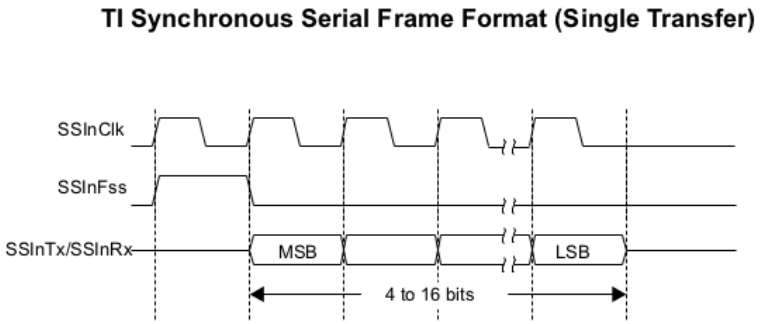
PWM alapú jelszintézis

A korábbi megoldás dedikált perifériához rendelése. Sajnos a PWM modul felépítéséből adódóan nem ideális ilyen jellegő üzemre, folytonos újrakonfigurálása/tiltása/engedélyezése lenne szükséges.

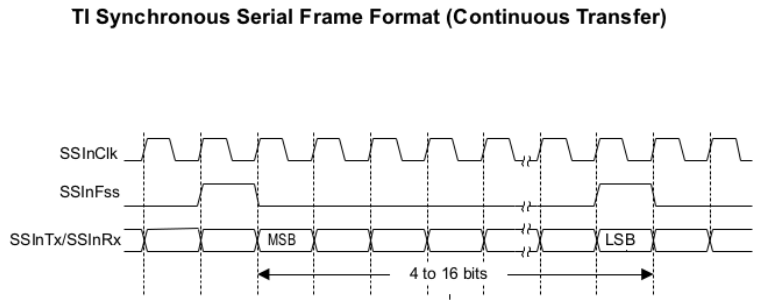
QSSI perifériával direkt memória hozzáféréssel memóriakép soros átvitele

DMA használatával a korábbi problémák gyökerét adó gyakori interruptok kiküszöbölhetők, de ehhez olyan periféria szükséges mely képes önállóan DMA kéréseket generálni és bufferel rendelkezik. A **QSSI (Quad Single Serial Interface)** az ilyen.

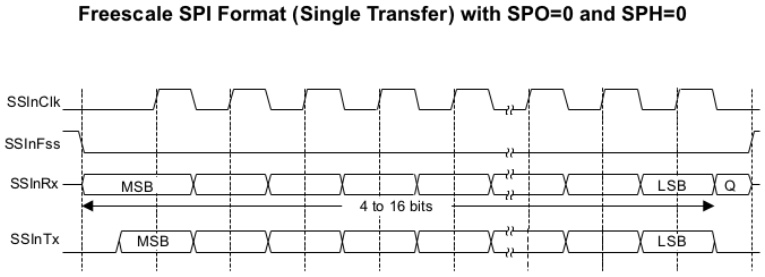
Ez a periféria 16 mély, konfigurálható 8/16 bit széles FIFO-va rendelkezik, melyből az adatokat mint SPI interfész 1-4 bit szélesen soros-párhuzamos átalakitással a kimenetére shifteli, saját bitclockjának megfelelően. Standard FIFO és End-of-Transmission interruptokkal rendelkezik. Bár szinkron interfész, ami **SPI**-t (illetve a Texas és Freescale SSI-át) valósítja meg, megfelelő konfiguráció mellet folytonos jelszintézisre használható.



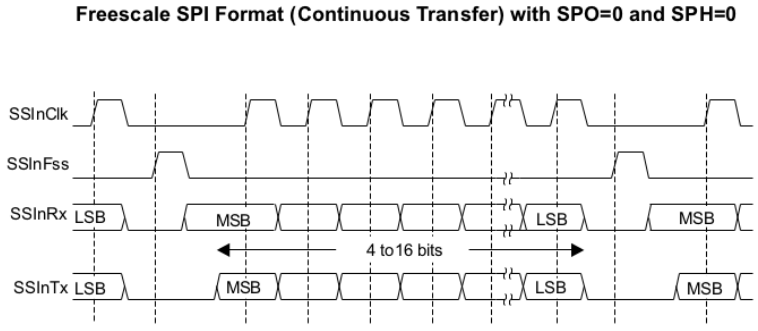
3.2. ábra. *TI szinkron soros üzenet*



3.3. ábra. *TI szinkron soros folytonos üzenet*



3.4. ábra. *Freescale szinkron soros üzenet*



3.5. ábra. *Freescale szinkron soros folytonos üzenet*

Látható hogy Texas **SSI**-ként konfigurálva, keretszinkron és bitclock jelek elhagyásával egy bit-tick megfeleltetésű adatokkal konfigurálható perifériát kapunk. FIFO nem üres

jelekre **DMA** kérést generálva, egy megfelelően konfigurált uDMA vezérlővel a szintetizálandó minta kisöpörhető a kimenetre.[24]

Sajnos a 4 **SENT** csatornának egymástól függetlennek kell lennie, mivel a **SENT** üzenetek hossza változó így a quad módban használva a **SENT** framek közötti szüneteket mindig a leghosszabb **SENT** üzenet határozná meg, ami elfogadhatatlan.

Csatornánkénti SSI perifériaával direkt memória hozzáféréssel memóriakép soros átvitele

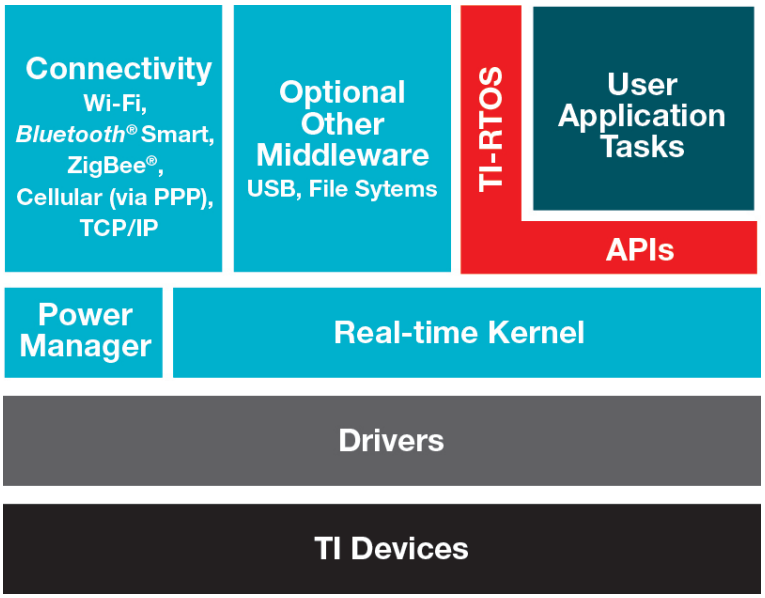
Az előző megoldást továbbvíve quad mód helyett 1 bit szélesen használva mind a 4 **SSI** perifériát a csatornák függetleníthetők egymástól. Ez azzal jár hogy mind a 4 csatorna külön generál magának **DMA** kéréseket és a **DMA** vége interruptokat külön kell kezelni. Előnye hogy a FIFO-t 16 bit szélesen is használhatóvá válik a Transmit és Recieve FIFO összekapcsolásával, így a generált **DMA** kérések száma némileg csökkenthető. Sajnos ezzel a kontroller minden **SSI** perifériáját felhasználjuk, így esetleges külső flash használata esetén **SPI**-os nem alkalmazható. A dolog egy további szépséghibája hogy az alkalmazott FIFO méret megszabja az átvivendő, jelet leíró memóriatartalom granualitását (modulo FIFOszélesség). Ez folytonos (frameszünet nélküli) **SENT** üzeneteknél okoz egyedül problémát, de szoftveresen kiküszöbölhető. (A legtöbb emulálni kívánt szenzor nem ilyen.)

3.2. Beágyazott operációs rendszer

A mikrokontrollerhez javasolt Texas Instruments féle **TIRTOS** valósidejű operációs rendszer. **FreeRTOS**-os megoldást elvetettem mert a **TIRTOS** támogatása a kontrollerhez jobb, továbbá a Code Composer Studio (**CCS**) fejleszői környezetbe jól integrált[28][26].

A **TiRTOS** teljes valósidejű operációsrendszer szolgáltatást kíván nyújtani, ütemezővel, middeware komponensekkel, driverkönyvtárakkal. Főbb komponensei:

- **SysBIOS**: a valósidejű kernel, valósidejű ütemezéssel, szinkronizációval, szálkezeléssel és hardver-absztrakcióval, konfigurációs eszközökkel. Amennyiben a konfigurációs felületét nemhasználjuk `xdc` scriptfájlokkal konfigurálható
- **UIA (Unified Instrumentation Interface)**: Loggolás és futásidejű adatgyűjtés biztosítása.
- **NDK (Network Developer's Kit)**: TI processzorok hálozati stackje
- **Network Services**: **NDK**-ra épülő alkalmazásszintű protokollok biztosítása (**DHCP**, **SNTP**, **HTTP**...). A **DHCP** használt belőle.
- **Kontrollerspecifkus driverkönyvtárak**: Mind felhasználó és operációs rendszer számára hardwarekezelő könyvtárak. Operációs rendszer csak egy részét használja.
- **XDCtools**: az oprendszer konfigurációs `xdc` scriptek összeállítását segítő eszközök, az IDE-be integrált.



3.6. ábra. *TiRTOS felépítése*

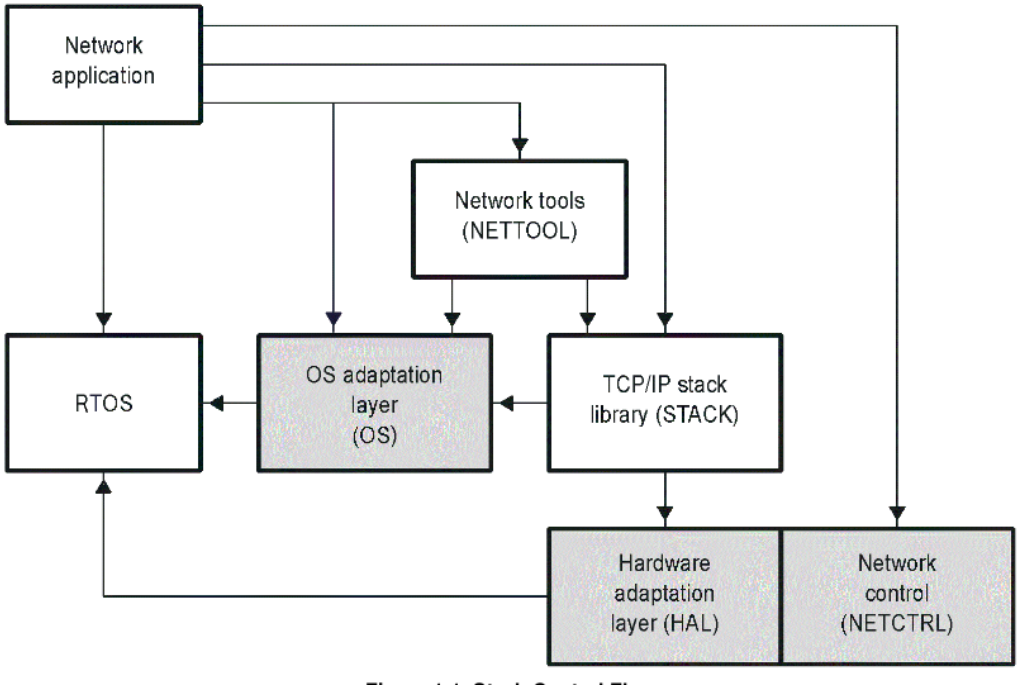
3.3. Hálózati stack

A felhasznált hálózati stack a **NDK**, a magasabb szintű szolgáltatások közül a **DHCP** van használva. Az **NDK** teljes TCP/IP környezet képes biztosítani kis memóriafelhasználás mellet.

Maga a stack el van választva az operációs rendszertől egy OS-absztrakciós rétegen keresztül, így akár más operációs rendszerekre is portolható, ezen réteg implementálásával.[27].

Főbb komponensei, könyvtárai:

- **STACK**: a fő **TCP/IP** stack implementációja, socket-rétegtől a tetején, Ethernet és PPP rétegig az alján.
- **NETTOOLS**: socket alapú hálózati szolgáltatások
- **OS**: operációs renszerhívások absztakciója, specifikus oprendszerhívásokká (jelen esetben **SysBIOS** hivásokká)
- **HAL**: hardverabsztrakciós réteg, **NDK** által használt perifériák kezeléséhez (timer, ethernet)
- **NETCTRL**: network control könyvtár, **NDK** és eszközmeghajtók inicializálása, ütemezlsi és hálózati események interészelése a megfelő callback és hook függvényekhez. A minimális változata használt.



3.7. ábra. *NDK felépítése*

3.4. Kommunikációs modell eszközei

A hálózati kommunikációs üzenetek leírása. Célom a jelszintézishez szükséges csatorna és szenzorkonfigurációs struktúrák és objektumok soros küldésének lehetővététele volt, valamiféle önleíró formátumban, lehetőleg minden absztrakciós szintre (nyers/raw jelalak, nativ adattag/nibble alapú + szenzor leíró, mért fizikai egység leíró + adattartalom + szenzorleíró). Ehhez a megfelelő adastruktúrák leírása és továbbítása szükséges szerializált formában.

Az ilyen jellegű megoldások (**XML**, **JSON**) beágyazott környezetben egy ekkora kontrolleren túl számításigényesek, tipikusan dinamikus memóriafoglalást igényelnek. A választásom emiatt az **ASN.1 (Abstract Syntax Notation One)**-ra esett. Ez egy az **ISO**, **IEC** és **ITU-T** közös szabványa mely egy absztrakt szintaxist definiál a küldendő információ alapján, de méretében nem korlátozza. A konkrét soros kódolt adatreprezentációt a különböző kódolási szabályai adják meg. Ebből a **BER(Basic Encoding Rules)**-t használtam fel.

Nagy előnye hogy sokféle fordító létezik hozzá, beágyazott környezethez is, melyek egy a kommunikált üzenetek leírója alapján a kódoló, dekódoló kódokat generálni képesek.

Fordítóként a mikrokontroller kódjának generálsához az **asn1scc** fordítót választottam. Ez a fordító nyílt forrású, **LGPL** licenszelésű, biztonságkritkus alkalmazásokhoz készült, az Európai Űrügynökség (**ESA**) által. Tiszta, dinamikus memóriát nem használó **C** kód generálására képes csupán **ASN** leíró fájlokból. A **TASTE** csomag része, de a csomag többi eleme nincs használva.

A PC oldali, vezérléshez használt java **API** számára az üzenetosztályok generálásához a **jASN1** fordított választottam. Ez szintén **LGPL** licenszű, szintén az **ESA** által is használt.

Az **ASN1** és fordítók bővebb leírását az ASN1 fejezet 6.1 tartalmazza. [10][5][30][16]

4. fejezet

Szoftver

4.1. Áttekintés

Az egyes funkciók áttekintése előtt a szoftver felépítését architektúráját a könnyebb megérthetőség miatt áttekinteném. A feladat alapvetően egy digitális jel szintézise több csatornán, amit a 3 -nek megfelelően a **QSSI** perifériákkal valósítok meg, egy a memórában tárolt mintasorozat **DMA**-zásával a periféria kimeneti bufferébe. Ennek megfelelően szükséges mindenképp egy alacsonyszintő perifériakezelő réteg (Ennek jó részét a **TivaWare** library-k szolgáltatják). Továbbá szükséges egy vezérlési réteg mely a kimenet és átvitelek állapota alapján a megfelelő időzítésekkel a perifériákat engedélyezi, **DMA** átviteleket indítja, megfelelő konfigurációk alapján. A megfelelő konfigurációkra feltétel hogy hálózaton küldhetők fogadhatók legyenek így egy hálózati réteg is mindenképp szükséges (ezt az **NDK** szolgáltatja).

A megfelelő konfigurácókat le kell, erre szolgál az esetünkben az **ASN1**. Az **ASN1** leíróból **API** kód generálható mind a kontroller **C** kódjához, mind a hálózaton vezérlő **Java API**-hoz. Az előbbihez a **ASN1SCC(Tastee)**, míg az utóbbihoz a **jASN(openMUC)** fordítókat használtam, mindkét eszköz **ESA**-s, nyíltforrású **LGPL** licenszű, ennek minden előnyével és hátrányával.

Kérdés milyen konfigurációkat alkalmazunk. Memóriaminta alapú jelszintézis lévén a legelemibb lehetőség egy adott minta megadása, viszont ez nem semmi szenzorspecifikus, a szenzor által használlt kommunikációs protokollnak megfelelő információt így külsőleg, manuálisan adjuk meg. A továbbiakban erre módra mint alacsony szintű/mintalapú jelszintézisre/vezérlése utalok.

Mivel az **SPC** és **SENT** protokoll nagyon hasonló, elemeiben szinte azonos elemekből épül fel így az adott elemeknek megfelelő alacsonyszintű mintát generáló kód készíthető és ezekből egy folytonos minta építhető (szinkronpulzus, soros üzenet, adattag, ellenőrzőösszeg, lezárópulzus stb..). A kivezérelendő jel ilyenszerű, építőblokkokból történő felépítésére a továbbiakban mint középszintü/blokkalapú szintézisre/vezérlésre utalok.

Az egyes blokkok külön kezelésének előnye hogy különféle hibák a blokkparaméterek változtatásával egyszerűbben bevihetők a rendszerbe (pl **CRC** hiba, szinkronpulzus hosszá-

nak megváltoztatása). Az ilyen jellegő leírás gyakorlatilag az jel építőelemeinek felsorolása a hozzájuk tartozó paraméterekkel. Ez a leírás viszonylag hosszú és sokszor szükségtelen mikor csak a szenzor normál működésekor egy adott gerjesztésre történő válaszát kívánjuk szimulálni. Ezenfelül ez a leírás nem állít fel kapcsolatot a valós világban mért fizikai jellemző és a hozzátartozó digitális reprezentáció között.

Emiatt van szükség a mintegy a szenzor transzfer-karakterisztikáját leírva egy magaszintű rétegre. Ezen a réteg operálva valós mennyiségeket adhatunk meg vezérléskor (szög, nyomaték), ha konfigurációban előre jelezzük miképp és melyik építőblokkra kívánjuk őket leképezni (tipikusan az adattagra, transzfer-karakterisztika alapján, vagy függvénnyel leírva, vagy lookup-táblát használva).

A konfigurációk leírásán kívül a vezérlőüzenetek is leírhatók **ASN1**-el. Ezek talán nem túl meglepő módon a csatornák engedélyezére/tiltására, konfiguráció feltöltésre, belső memórába mentésre, lookup-tábla feltöltésére vannak.

Felmerülhet továbbá az igény a bemeneti paraméterek automatikus változtatására erre egy külön task indítható távolról (`AutoChange`), ami megadott minta alapján periodikusan változtatja a szimulálandó mért mennyiségeket. (pl.: szögjel körbefordulásának lejátszása egy rolloveres sweepel)

A kód részletesebb leírása a mellékelt **Doxygen**-es dokumentációban található.

4.2. Adatmodell és működés

Az működés áttekintése alapján az adatmodell lehírható. Minden ping-pong bufferelt, egy aktív használt és egy éppen előállított bitmintát tartalmazóval. Az aktív buffereket jegyezni kell. Továbbá mivel egy bit jelzi egy tick értékét és a **SENT**/**SPC** protokollok üzenetei változó tickhosszúságúak és nem maradék nyolccal oszthatók, hogy a bitminta egész bájttöbbszörös legyen, így az bufferhez tartozó valós mintahosszat is tárolni kell. Ez többek között az is jelenti hogy a folytonos (**SENT**) módban mintafolytonosság miatt a legutolsó nem teljes bájtot érdemes tárolni és bufferváltáskor maszkolva a megfelelő pozícióból folytatni a beállítását.

A szenzorleíró struktúra tartalmazza egy **SENT**/**SPC** szenzor jellemző paramétereit mint:

- szenzor neve karakteresen
- mód (**SENT** vagy **SPC**)
- nibblekezdetet jelző alacsony tick-ek száma
- elfogadott minimális és maximális triggerpulzus **SPC** esetén
- trigger után az üzenet késleltetése **SPC** esetén
- kimeneti meghajtásának típusa (**high-z, push-pull, open-collector, open-emitter**)

Szükséges természetesen a csatorna megadás hogy a szenzort melyikhez kívánsuk rendelni. Ez a kevesebb üzenetváltás érdekében. A üzenetekben csatornaleírás minden esetben bitwise, tehát a helyiérték adja meg a csatorna számát hogy a konfiguráció rá vonatkozik-e. Ez a magasabbszintű működés üzeneteire is vonatkozik így szinkron, egy üzenettel állíthatók a csatornák értékei.

A közép/blokk szintű hozzáférés esetén szükséges a jel blokkjainak leírása szekvenciálisan. A `conv_functions.h/c` tartalmazza a használható blokkgenerátor függvényeket. Annak érdekében hogy bővíthető legyen új blokkgenerátor-függvényekkel amennyiben igény merülne fel rá és futásidőben változtatható legyen a blokkgenerátorok hívási lánca, a kommunikációs üzenetben karakteresen leírtak a generátorok, kezdeti paraméterrel. Futás közben ezt a karakteres leírást keressük egy listában, ami kulcs-érték párként a hozzá tartozó hívandó blokkgenerátor függvénypointerét tartalmazza. Új blokkgenerátor hozzáadása ennek megfelelően egy szignatúrát kielégítő függvény írásából, valamint a kulcs-értékpárjának a listára történő felvételéből áll.

A magas szintű leírók és az automata értékváltoztató taszk által hívandó függvények is hasonlóan tártoltak és bővíthetőek, a hívandó függvények szignatúrája tér el (mivel ezek jóval általánosabb függvények lehetnek emiatt változó hosszúságú paraméterlista adható nekik)

A különböző szintek ki és bemenetei és paraméterei csatornával indexelt tömbökben tároltak.

A buffervezérlő taszkok blokkolt állapotból három esemény mozdíthatja ki:

- engedélyezés/tiltás
- konfiguráció érkezése
- mintaküldés befejeződött, új minta előállítható

A mintaelőállítás ezekután a hozzáférési szint ellenőrzéséből, a szinthez tartózó vezérlőüzenet mailboxának ellenőrzéséből, bufferpointer váltásból, majd a legmagasabb szintről indulva a konverziós függvények egymásutáni meghívásából áll. **SENT** esetén az utolsó nem teljes bájtja a mintának tárolásra kerül. Végül az új buffer hosszáról és helyéről szintén egy mail készül amivel a **DMA** konfigurálható, a folyamatba lévőnek befejeződése esetén.

A vezérlőüzenetek mailboxát a **TCP/UDP** feldolgozó taszkok töltik, a fogadott üzenetek dekódolása után. Mivel **ASN1**-el leírtak a struktúrák, így a konfiguráció a csatornainformáció feldolgozása után módosítás nélkül postázható az adott csatornák mailboxaiba.

Tárolva van továbbá fixen az **EEPROM**-ba történő konfigurációmentéshez/olvasáshoz tartozó slotok címe, valamint a transzferkarakterisztika lookup-táblája, valamint a csatornával indexelve az Taskok és **IPC** handle-ök pointerei.

4.3. Hardver absztrakció

A fejlesztés egy **DK-TM4C129X** development kittel indult, ami egy Tiva C sorozatú **TM4C129XNCZAD** mikrokontrollert tartalmaz ami a megcélzott **TM4C129ENCPDT**-hez nagymértékben hasonló, mind pinout mind perifériakészlet tekintetében (**LCD** kontrollerrel rendelkezik többek között, valamint ballgrid-es tokozású quad flatpack ellenében). Az szükségszerű későbbi portolás (előbb a kisebb **TM4C129E Launchpadre**, majd a végleges kártyára), valamint általános best practice jegyében a kezdetektől igyekeztem egy hardver absztrakciós réteget kialakítani.

Ennek megfelelően a periféria driverek a `hal.c/hal.h` kódjának keresztül érhetők el a magasabb szintű funkciók által. A gyorsabb futás érdekében ezek a függvények szinte kivétel nélkül inline-ok. Ebbe a rétegbe wrappolhatjok a megfelelő függvényekbe a gyártói perifériameghajtókat. További előnye hogy ahol az adott periféria driver az igényeknek nem megfelelő vagy nem tökéletesen illeszkedik úgy akár újraírhatjuk, vagy akár más perifériákat használhatunk a funkció ellátására amíg a függvényszignatúrákat tartjuk. Jelen esetben a **QSSI** periféria például nem a gyártó által szándékolt funkciójában van így használva. Minden használt harveres funkció megjelenik itt.

Esetleges portoláskor ha nincs kézenfekvő megoldás az egyes, nem létfontosságú funkcióra természetesen üres függvénycsonk használható, esetlegesen hibaüzenet küldhető a kommunikációs stacknek, hogy továbbítsa a felhasználónak. Az interruptfüggvényeket az operációs rendszer konfigurációs fájlában regisztrálni kell hiszen ütemezés szempontjából fontosak, elkapásukról is az operációs rendszer gondoskodik. Amennyiben az adott funkcióhoz nem tartozna interrupt úgy a funkcióján szerinti esemény bekövetkezésekor ezt a jelzést generálni kell az opeációs rendszer számámára.

Hardverabsztrakciós réteg a következő funkciókat absztrahálja:

- pinsetting: kezdeti pininicializálás a funkcióknak megfelelően
- ssi_setup: jel granualitásának megadása, tickhossz alapján, esetünkben ez az QSSI periféria konfigurálását jelenti a megfelelő bitrátára
- ssi_channel_setup: egy csatornán jel granualitásának megadása, tickhossz alapján, esetünkben ez az QSSI periféria konfigurálását jelenti a megfelelő bitrátára
- drive_set: meghajtás választása: **high-z, push-pull, open-collector, open-emitter**
- eeprom_setup: nemfelejtő memória inicializás, konfiguráció illetve lookuptábla tárolásához
- eeprom_write: nemfelejtő memória írás, konfiguráció illetve lookuptábla tárolásához
- eeprom_write: nemfelejtő memória olvasás, konfiguráció illetve lookuptábla betöltéséhez

- fpu_enable: lebegőpontos egység inicializálás, engedélyezés
- irq_ssi_clr: átvitel végén hívodik, mind **SENT** mind **SPC** vége jelzések/interruptok letiltása, jelen esetben az átvitelhez tartozó, **DMA** és **SSI** ready/fifo-empty jelzések törlése
- irq_ssi_dma_enable: **SENT** esetén következő minta előkészítésére felszólítás, esetünkben megfelelel a **DMA** befejeződésének jelzése, mivel a kimeneti perifériabufferbe bekerült az összes minta.
- irq_ssi_fifoempty_enable: **SPC** esetén következő minta előkészítésére felszólítás, esetünkben megfelelel a kimeneti-buffer üres jelzésének
- irq_tim: visszaadja két meghívása között eltelt időt, triggerpulzushossz méréséhez
- debug_pin: debuggoláshoz állítható pin
- led_invert: működést jelző LED villogtatásához
- read_mac: egyedi **MAC**-cím beállítása, jelen esetben egy **I2C**-s Node Identity **EEPROM** szolgáltatja
- ssi_dma_transfer: minta küldése vonalra, jelen esetbe egy **DMA** átvitel, mintatranszfer a kimeneti bufferbe
- ssi_channel_disable: csatorna letiltása
- ssi_channel_enable: csatorna engedélyezése
- timer_channel_setup: triggerpulzus elfogásának beállítása, jelen esetben ez egy timer capture üzemmódja generált interrupt
- timer_channel_disable: triggerpulzus elfogásának letiltása, jelen esetben ez egy timer capture üzemmódja generált interrupt tiltása
- timer_periodic_enable: periódikus interrupt engedélyezés az automata bemeneti paraméter állító tasknak
- timer_periodic_disable: periódikus interrupt tiltás az automata bemeneti paraméter állító tasknak
- timer_periodic_irqclr: automata bemeneti paraméter állító task futásának jelzése, jelen esetben ez törli a hozzátartozó interruptflaget

4.4. Operációs rendszerkonfiguráció

A használt **TiRTOS** konfigurálása a projektben található `app.cfg` configfájllal történik. A csomag minden eleme ezen fájlal konfigurálható, ideértve a használt **SYS/BIOS** kernelt, az **NDK** hálózati stacket és **UIA**-t a loggoláshoz. A fájl kézzel megírható, példák alapján módosítható, de a **CCS** fejlesztőkörnyezetet használva egy GUI is rendelkezésre áll a módosításhoz. Az alábbi modulok kerültek használatra. [28][26][22]

4.4.1. SYS/BIOS, System, SysMin

Maga a kernel, értelemszerüen kell használni hogy oprendszer rendelkezésre álljon. Custom módban használtam. Taszkok, szoftver és hardverinterruptok mind engedélyezve, számukra viszonylag bőséges stacket-hagyva (`8k`). Órajel a maximumra választva (120Mhz). Futásidejű ütemezett objektumok létrehozása engedélyezett (szükséges többekközött az `AutoChange` task és **TCP** protokoll miatt, hogy fogadáskor új taskban futhasson a kapcsolat kezelése). [28][26]

A célfunkció megvalósításához szükséges taszkok statikusan induláskor jönnek létre, hiszen mindenképp szükség van rájuk, és így futásidőben nem szükséges erőforrásfoglalás számukra. A **SysMin** a minimális System modul, `printf()` támogatás belső bufferbe (**SysStandard** a standard `printf()`-et használja ami lassabb). A buffer fejlesztőkörnyezetből olvasható, vagy flusholható a standard kimenetre. [28][26]

4.4.2. Log, LoggerBuff

Cirkuláris memóriába loggolás, régi rekordok felülírása (opcionálisan a loggolás lehagyható ha a buffer feltelik). A futásid javítsára csak az **Error** és **Warning**ok vannak engedélyezve, de debuggoláshoz akár futásidőben is engedélyezhetők a különféle információs események, függvénybelépési logok. [28]

4.4.3. Hwi, Swi, Task

Hardverinterrupt és szoftverinterrupt. Taszkokkal ellentétben egyszeri lefutású önálló stackkel nem rendelkező függvények. A stack ismert értékre inicializálódik, hibakereséshez. Beágyazott interruptok engedélyezettek, hiszen szükségeses hogy akár egy hosszabb hálózati stack interruptot megszakíthassunk. **16 prioritás szint** mindháromnál. Használt hardverinterruptok a használt **SSI** perifériák kimenő bufferének üresjelzése és capture események az **SPC** triggert figyelő pineken. Statikusan inicializált taszkok az egyes csatornákat buffereinek beállítását és kiküldését vezérlő taszkok, egy alive/idle taszk legkisebb prioritással éhezés figyelésére (ez vezérli a visszajelző ledet), valamint az **EEPROM**-ba mentést/olvasást kezelő taszk. [28][26]

4.4.4. Event, Mailbox, Semaphore

Használt **IPC** mechanizmusok. **MailBox**okkal történik a kapott vezérlőüzenetek átadása a feldolgozó taszkok számára, valamint az aktuális kiküldendő bufferek leírása. **Szemafor**ok jelzik a buffert vezérlő/mintával töltő taszkok számára mikor szükséges új minták generálása. Értelemszerűen ez szükséges új paraméter és konfigurációs üzenet fogadásakor is, ezt az **Event**ek teszik meg összefogva és/vagy kapcsolatba különböző **IPC** mechanizmusokat.[28][22]

4.4.5. IP, Global, TCP, UDP, Icmp, DhcpClient

NDK networkstack részei, globális beállítások, mint a network taszkok stackméretei és prioritásai (prioritásban a jelgeneráláshoz használt interruptok alá vannak helyezve), hálózati réteg beállításai (**TTL** = 64, timeout, ip-forwarding tiltott, maximum 2 egyidejű kapcsolat, címszerzés automatikusan **DHCP** kliensként, **ICMP** engedélyezés pingeléshez), szállítási rétegprotokollok (**TCP** és **UDP** bufferméretek). Felmerülhet az igény több eszköz nevének megkülönböztetésére, ez a configfájl `Ip.hostName` sorának cserélésével tehető meg újrafordítással, vagy a networkstack indulási hookjának megadásával és elkapásával futásidőben is beállítható. [28][27]

4.5. Ütemezett objektumok

Az operációs rendszer által ütemezett objektumok a `schedueled.h/c`-ben találhatók. Könnyebb megkülönböztetés miatt a függvényeik a `taskFx`, `hwiFx`, `swiFx` prefixekkel vannak ellátva. A mindenképp szükségesek közülük statikusan inicializáláskor létrehozottak (pl.: bufferek/kimenet vezérlése), míg a igény szerint szükségesek futásidőben jönnek létre (**TCP** kapcsolatok feldolgozása) Az `auxFx` prefixű függvények segédfüggvények, az ütemezett objektumok által hívottak, a struktúráltabb kód érdekében.[28][22]

4.5.1. Taszkok

Az alábbi taszkok kerülnek futásra a rendszerben:

taskFx_BufferControll

Feladata egy csatorna kezelése, buffereinek előállítása, a küldendő buffer postázása. A működés során ismertetett (4.2) események mozdíthatják ki blokkolt állapotból.

taskFx_SaveLoad

Mentés illetve töltésre felszólító üzenet (**StoreTableMsg**) hozza ki blokkolt állapotból. Az üzentben adott slot/slokra menti illetve tölti az **EEPROM**-ba/ból a konfigurációt.

taskFX_AutoChange

Dinamikusan létrejövő taszk, **AutoChangeEnable** üzenet hozza létre. Automatikus bemeneti paraméter módosításra szolgál az `AutoChangeCfg` üzenetben megadott függcény alapján, egy belső periodikus timert konfigurálva és használva. Megszüntetése az `AutoChangeEnable}` üzenet hamis értékére történik.

taskFX_Alive

Legalacsonyabb prioritású taszk, periodikusan villogtatja a visszajelző **LED**-et, esetleges futásidejű hibákra, éhezésre figyelmezteti a felhasználót.

taskFx_UDP

UDP datagrammokat feldolgozó és küldő taszk. Dinamikusan jön létre az **NDK** hookjaként hívótt `swiFx_NetTaskCreate` hozza létre. **BSD**-s socket interfészt használ, nyitnak egy filedescriptor sessiont, inicializálják a socket struktúrát, majd bindolja a descriptorhoz. Multiplexált **IO**-t használ, egyszerre tud küldésre és fogadásra várni. `select()` helyett `poll()` van használva mert várt descriptorok és eseményeik leíró struktúráját nem roncsolja szét az esemény bekövetkezésekor. `poll()`-ozott leírok és eseményeik leíró tömbje kitöltése kerül és a poll végtelen ciklusba meghívódik, hiszen blokkol amíg nincs esemény. `POLLIN` események üzenetfogadások, ezek az üzenet **BER** dekódolásával járnak a tag-jük alapján, majd a megfelelő taszk mailboxába továbbításra kerülnek. `POLLOUT` események a `POLLIN` üzenetek nyugtázása. Out of band data nincs kezelve.

taskFx_TCPListen

TCP szerver, bejövő kapcsolatokat figyelő taszk. Dinamikusan jön létre az **NDK** hookjaként hívótt `swiFx_NetTaskCreate` hozza létre. **BSD**-s socket interfészt használ. Megnyitja a listening socket létrehozásához szükséges filedescriptor sessiont, létrehozza és kitölti a socketstruktúrát, bindolja a descriptorhoz, beállítja a socketet listeningre, (`SO_KEEPALIVE` opciót használva), majd fogadja a beérkező kapcsolatokat, egyszerre maximum kettőt. Kapcsolat érkezésekor a feldolgozó taszkot indít neki (`taskFx_TCPnoPoll`).

taskFx_TCPnoPoll

TCP üzenetek feldolgozó taszkja. A paraméterként kapott socketen üzeneteket fogad, a socket kezeléséhez megnyitja a saját filedescriptor sessionjét. A socketen ezután blokkolva (`recv()`) próbál fogadni, fogadás estén az üzenet tagje alapján **BER** dekódolni és nyugtát küldeni, majd a dekódolt üzenetet a megfelelő mailboxba továbbítani. `poll()` nincs használva mert a kapcsolat bontásának észlelése nem működött vele, lévén a `recv()` soha nem tért vissza 0-val féig nyitott socketet jelezve, így a ciklusból kilépés, socket lezárás és task megsemmisítés nem volt lehetséges.

4.5.2. Szoftvermegszakítások

Tipikusan egyszeri lefutású, szoftveresen triggerelhető függvények. Elsősorban a hálózatkezelő taszkokat létrehozásáért és megsemmisítéséért felelnek. Bár nevükben és működésükben tekintve mint szoftvermegszakításként szerpelnek, valójában inkább segédfüggvények, mert a különböző **NDK** hookfüggvényeiként vannak használva, így nem mint az **RTOS** által jegyzett szoftveres megszakítások működnek, hanem **NDK** taszkok kontextusában hívódnak. [28][22]

swiFx_NetTaskCreate

NDK hook a **TCP** szerver taszk és az **UDP** taszk létrehozására.

swiFx_UDPTaskCreate

NDK hook az **UDP** taszk létrehozására.

swiFx_UDPshutdown

UDP taszk befejezése, filedescriptor sessionjének lezárása.

swiFx_TCPTaskCreate

NDK hook **TCP** szerver taszk létrehozására.

swiFx_TCPshutdown

TCP szerver és kiszolgáló taszk befejezése, filedescriptor sessionjének lezárása.

4.5.3. Hardvermegszakítások

Elsősorban a folytonos hardverbufferelt minta fenntartásáért felelnek a kimeneten és triggerpulzuok észleléséért a capturepineken.

hwiFx_SSI

Az **SSI** megszakítások kezelője, tehát a mintaküldés végét jelzi. Törli a megszakításflageket, folytonos **SENT** esetén elveszi a mailboxból a következő mintabuffer üzenetét és újrakonfigurálja vele a **DMA** átvitelt. Végül a buffervezérlő taszknak az új minta kérését jelző szemafort beállítja. Triggerelt **SPC** esetén újraengedélyezi a triggerfigyelést, ami SPC üzenetküldés alatt le le van tiltva hiszen a vonalon a saját jelváltásinkra nem kívánunk triggerelni.

hwiFx_tim

A vonal capturetimereinek megszakítása. Megszakítás esetén ellenőrzi a pulzushosszt, és amennyiben megfelelő elindítja az **SPC** jel küldését a buffer mailbox elvételével és **DMA** konfigurálásával, majd letiltja önmagát hogy az általunk küldött jelre ne triggerelhessünk. Végül a buffervezérlő taszknak az új minta kérését jelző szemafort beállítja.

4.5.4. Segédfüggvények

Segédfüggvények különbőző funkciókra hogy a taszkok törzsei tömörebbek, áttekinthetőbbek legyenek.

auxFx_BufferControll

A BufferControll taszkok hívják, hogy ne kelljen csatornánként minden taszkba megismételni. Adott csatornára előállítja a módnak megfelelő buffert a konfiguráció alapján és posztolja a leíróját a megfelelő mailboxba. A függvény nem reentráns hiszen az őt hívó taszkok azonos prioritásszinten vann így nem lehetséges hogy megszakítsák egymást.

auxFx_channel_enable

Adott csatorna engedélyezése. Konfigurálja az tickidőt, a kiment típusát, és engedélyezi az üzemmódnak megfelelő megszakítást.

auxFx_channel_disable

Adott csatorna tiltása. Kimenetet nagyimpedanciás állapotba teszi, letiltja a **DMA** és **SSI** kimeneti bufferének megszakításait.

auxFx_EnableEvent

`Enable`-üzenet kezelése. Engedélyező/tiltó kérésnek megfelelően a megfelelő csatornára a az engedélyező/tiltó segédfüggvényt hívja.

auxFx_CalculateEvent

Új minta kérést jelző szemafor esetén hívódik, megfelelő csatornára az bufferszámító segédfüggvényt hívja.

auxFx_ConfigEvent

Konfigurációs üzenet esetén hívódik, elveszi a konfigurációs üzenetet a mailboxból, beállítja a konverziósfüggvényeket és a default paramétereket rájuk, engedélyezi a csatornát ha szükséges.

auxFx_AutoChangeTaskCreate

Az automata paraméterváltaztató task létrehozására segédfüggvény. `AutoChangeEnable` üzenet igaz értéke esetén létrehozza a taskot amennyiben van beállított `AutoChange` konfiguráció.

auxFx_AutoChangeTaskShutdown

Az automata paraméterváltaztató task megszüntetésére segédfüggvény. `AutoChangeEnable` üzenet hamis értéke esetén megszünteti a taszkot, a beállított konfigurációt viszont megtartja, így nem kell újraküldeni.

auxFx_init_structs

Belső változók, csatornaleírók, konfigurációk, paraméterek inicializálása ésszerű alapértelmezett értékekre.

auxFx_save_channel_settings

`SaveLoadMsg` alapján adott slotra mentése egy csatornakonfigurációnak az **EEPROM**ba.

auxFx_load_channel_settings

`SaveLoadMsg` alapján adott **EEPROM** slotból konfiguráció beállítása a megadott csatornákra.

4.6. Szinkronizációs mechanizmusok, objektumok

Minden egyes csatorna rendelkezik számára szükséges üzeneteknek megfelelő mailboxokkal. A hálózati taskok az üzenetdekódolás után, a csatornainformációt lefosztva és ellenőrizve a megfelelő csatornák mailboxaiba posztolják az üzeneteket. Mivel az **ASN1** leírja a felhasznált struktúrákat így a posztolás direktbe lehetséges, további feldolgozás nem szükséges. Az új minta előállítását kérvényező szemaforokat a hardveres megszakításfüggvények posztolják. Ahol szükséges több szinkronizációs objektumra várakozás ott az adott objektumokat és/vagy kapcsolattal az **event**ek fogják össze (`BufferControll`). A csatornánkénti szinkronizációs objektumhandlerek címei a könnyebb kezelhetőség érdekében, csatornaazonosítóval indexelhető tömbbe vannak összefogva. [22]

4.6.1. Szemaforok

semAutoTimer

Az `AutoChange` taszk futásának engedélyezése.

semBufferControll

A `BufferControll` taszkok számára új minta generálás engedélyezése

4.6.2. MailBoxok

mBoxChannelCfg

ASN1 csatornakonfigurációs üzenet mailboxa.

mBoxEnable

ASN1 csatornaengedélyező üzenet mailboxa.

mBoxHValues

ASN1 magasszintű hozzáférés esetén adatüzenet mailboxa.

mBoxOutBuffAddr

Kiküldendő buffer címét és hosszát tartalmazó mailbox.

mBoxRaw

ASN1 alacsony szintú hozzáférés bitmintáit tartalmazó üzenet mailbox.

mBoxValues

ASN1 középszintű hozzáférés blokkjait tartalmazó üzenet mailbox.

mBoxSaveLoad

ASN1 Konfiguráció mentés/töltést kérő üzenet mailbox.

4.6.3. Eventek

eventHandlerContolMsg

`BufferControll` taszk futásának engedélyezése, $VAGY$ kapcsolatba hozott `semBufferControll`, `mBoxEnable` és `mBoxChannelCfg`.

4.7. Prioritás hozzárendelés

Az **RTOS** 16 prioritásszintre van konfigurálva taszkok tekintetében. A preempció engedélyezett.

Interruptok azonos prioritásúak, kellően gyors futásúak, így az interrupt nestinggel járó overhead elkerülhető. Egyidejű megjelenésük kis valószínűségű, hiszen **SENT** esetén a a

triggercapture interruptok nem engedélyezettek, és minimális megszakíthatatlan **DMA**-zott adatmennyiség adott, így a kimeneti perifériabufferek egyszerre nem ürülhetnek ki és a **DMA**-k párhuzamosan nem fejeződhetnek be hiszen egyszerre egy lehet aktív a buszon. **SPC** esetén a triggercapture interruptok letiltásra kerülnek a minta küldésének befejeződséig, tehát az egyidejüség itt sem probléma.

A taszkok között az `Alive` a legalacsonyabb prioritású (1). Őt követik a hálózati taszkok, mint az **NDK** különböző prioritású taszkjai (3,5,7,9). A **TCP** és **UDP** üzeneteket feldolgozó taszkok szintén alacsony prioritással szerepelnek (2). A ráció emögött hogy a kimeneti stabil jelszintézis fontosabb mint esetleges új üzenet fogadása, feldolgozása, ami így maximum később jut érvényre. A `SaveLoad` és `AutoChange` taszk szintén alacsony prioritású emiatt (4).

A magas prioritású taszkok (13) a mintagenerálásért felelős taszkok, hiszen a friss kimeneti minta biztosítása elsődleges, leginkább **SENT** esetén ahol a folyamatosan streamelt jel megkövetelt. [28]

4.7.1. Adatfolyam DMAval streamelve kontra megszakítással vezérleve

Felmerülhet a kérdés hogy miért nincs a **DMA** vezérlő hardveresen ping-pong/stream módban használva, hiszen ez **SENT** esetén a folytonos kimeneti jelet biztosítaná és a **DMA** periféria lehetőséget biztosít rá. Való igaz ez **SENT** esetén kézenfekvő megoldás lenne, viszont az **SPC** implementációt elnehezíteni. **SPC** esetén az alkalmazott pingpongbufferek nagyságától függene a triggerre a minimális válaszidő, hiszen egy már felprogramozott, megkezdett **DMA** átvitelt nem tudunk megszakítani, így kisebb buffereket kéne alkalmazni, ami viszont gyakoribb megszakításokkal járna, overheadet okozva. Emiatt egyszerűbb egy előre beállított mintára **DMA**-t kiadni trigger esetén. A **SENT** által megkövetlet folytonos minta továbbra is biztosítható, mindössze arról kell gondoskodni, hogy a kimeneti periféria buffere sose ürülhessen ki. Ez könnyen megtehető ha ez esetben nem a buffer üres interruptot használjuk új átvitel kezdeményezésére, hanem a **DMA** vége jelzést, hiszen ekkor a hardverbuffer még egy minimális $3\mu s$ tick esetén $192\mu s$-nyi jelet tárol (egy **SENT** üzenet pause opció nélkül ekkor $462 - 810\mu s$ hosszúságú lehet).

4.8. Konverziós függvények, AutoChange fügvények

A tipikusan **SENT**/**SPC** protokolloknál aszabvány által specifikált jelalak előállításához szükséges a különböző hozzáférési szintehez használt konverziós függvényekről gondoskodtam. Ezek egyirányúak hiszen a szintézishez nincs szükség a magasabb szintre lépéshez, így a lefutási idejük gyorsítható.

Az `AutoChange` függvények is felfoghatók konverziós függvényeknek melyek külső beavatkozás nélkül az időskálán képezik le a szomszédos értékeket. Megadási módjuk is hasonló emiatt.

4.8.1. Magas-középszint konverziók

A cél a egy fizikai jelérték leképezése a szenzor által érzékelt jelértékre, így megadva a szenzor transzferkarakterisztikáját.

no_spec

Nem specifikált, nem történik magas-közép konverzió, üres függvénycsonk. Szerepe hogy hibás konfigurációkor a konverziós pointertömbbe beállítható így a hibás konfigurációs paraméter könnyebben elkülöníthető, illetve ettől még konfiguráció futtatható, nem okoz futásidejű hibát.

direct

Direktkonverziója a magasszintű értéknek a kontroller által belsőleg magaszintű jelérték-változójára. Esetünkben ez egy egyszerű double castolás. Hasznos ha ha a transzferkarakterisztikát előzőleg saját magunk kívánjuk leképezni, de szertnénk az egyszerűbb magasszintű üzenetekkel állítani a szintetizált jelet.

linear

Lineaáris transzfer karakterisztika. Paraméterei a meredekség és offszet. Valóságban legritkább esetekben jellemző egy szenzorra, de egyszerű müködés ellenőrzéséhez megfelelő, ha az egyenes jól illeszthető és csak funkcionálisan érdekel a működés, nagy pontosság nem szükséges.

lookup

Előzőleg eltárolt lookuptábla használta. A lookuptáblába a transzferkarakterisztika töréspontos közelítése kerül, egyenletes granualitással, megadva a kezdeti és végpontját a független változónak, a granualitást és az értékek tömbjét. A pontosabb közelítés érdekében a megadott pontok közötti értékekre lineárisan van közelítve konverziókor.

4.8.2. Közép-alacsonyszint konverziók

A cél a megfelelő bitminta létrehozása a memóriában ami a kimeneti perifériába DMA-zható. A használt protokollok esetén ezek tipikusan 12-16 bites érték és jel további építőelemei (szinkronpulzus, késleltetlés, státusznibble).

no_change

Nem specifikált, nem történik közép-alacsony konverzió, üres függvénycsonk. Szerepe hogy hibás konfigurációkor a konverziós pointertömbbe beállítható így a hibás konfigurációs paraméter könnyebben elkülöníthető, illetve ettől még konfiguráció futtatható, nem okoz futásidejű hibát.

delay

Késleltetés. Konstans magasszint (a vonalak alapértelmezett szintje) a kimenetre megadott tickideig.

pause

Pause pulzus. A vonatkozó **SAE** szabvány reviziójában bekerült jel. Ekvidisztáns mintákra biztosít lehetőséget. A kimeneti jelek egy, paraméterbe adott hosszúságúra kiegészítése egy extra nibble-el.

sync

A vonatkozó **SAE** szabvány által definiált 56 tickidejű szinkronpulzus.

data_4b

Legegyszerűbb 4 bites nibble. Alap adattag **SENT/SPC** protokolloknál. A hosszabb adatok reprezentációjára is ez használható.

data_8b_MSN

8 bites adat 2 nibbleben, nagyobb helyiértékű nibble először.

data_8b_LSN

8 bites adat 2 nibbleben, kisebb helyiértékű nibble először.

data_12b_MSN

12 bites adat 3 nibbleben, nagyobb helyiértékű nibble először.

data_12b_LSN

12 bites adat 3 nibbleben, kisebb helyiértékű nibble először.

data_12b_MSN_s

12 bites adathoz tartozó **secure** formátum által használt nibble. Gyakorlatilag a legnagyobb helyiértékekhez tartozó 4 bites nibble inverzének tárolása a későbbi beillesztéshez.

data_16b_MSN

16 bites adat 4 nibbleben, nagyobb helyiértékű nibble először.

data_16b_LSN

16 bites adat 4 nibbleben, kisebb helyiértékű nibble először.

data_16b_LSN_i

16 bites adat 4 nibbleben invertálva, kisebb helyiértékű nibble először. Hasznos szenzorpároknál ahol az egyik szenzor az invertált értéket küldi így ugyanazzal az üzenettel állíthatók.

data_singsec

Single Secure formátumhoz a legnagyobb helyiértékű nibble inverzének beillesztése.

stat_sae

A **SAE** által definiált státusznibble beillesztése, soros üzenettel, ellenőrzőösszeg számítással. Paramétere tartalmazza a soros üzenetet. `MSB(15:12): Message ID (11:4):Serial message (3:2):DontCare (1:0):AppSpecificData`

stat_sae_nocrc

A **SAE** által definiált státusznibble beillesztése, soros üzenettel, ellenőrzőösszeg számítás nélkül. Hasznos manuálisan elrontott ellenőrzőöszegű üzeneteke generálásához. Paramétere tartalmazza a soros üzenetet és használanó **CRC** értéket. `MSB(17:16): Message ID (15:8):Serial message (7:4):CRC (3:2):DontCare (1:0):AppSpecificData`

stat_sae_e12

A **SAE** által definiált **enchanced** státusznibble beillesztése 12 bites soros üzenettel, ellenőrzőösszeg számítás nélkül. Hasznos manuálisan elrontott ellenőrzőöszegű üzeneteke generálásához. Paramétere tartalmazza a soros üzenetet és használanó **CRC** értéket. `MSB(31:24): Message ID (23:12):Serial message (11:10):DontCare (9:4):CRC (3:2):DontCare (1:0):AppSpecificData LSB`

stat_sae_e12c

A **SAE** által definiált **enchanced** státusznibble beillesztése 12 bites soros üzenettel, ellenőrzőösszeg számítással. Paramétere tartalmazza a soros üzenetet. `MSB(31:24): Message ID (23:12):Serial message (11:10):DontCare (9:4):CRC (3:2):DontCare (1:0):AppSpecificData LSB`

stat_sae_e16

A **SAE** által definiált **enchanced** státusznibble beillesztése 16 bites soros üzenettel, ellenőrzőösszeg számítás nélkül. Hasznos manuálisan elrontott ellenőrzőöszegű üzeneteke generálásához. Paramétere tartalmazza a soros üzenetet és használanó **CRC** értéket. `MSB(31:28): Message ID (27:12):Serial message (11:10):DontCare (9:4):CRC (3:2):DontCare (1:0):AppSpecificData LSB`

stat_sae_e16c

A **SAE** által definiált **enchanced** státusznibble beillesztése 16 bites soros üzenettel, ellenőrzőösszeg számítással. Paramétere tartalmazza a soros üzenetet. `MSB(31:28): Message ID (27:12):Serial message (11:10):DontCare (9:4):CRC (3:2):DontCare (1:0):AppSpecificData LSB`

stat_TLE4998S

Elsősorban **Infineon** szenzorok által használt státusznibble ami a **SAE** által előírtnak nem felel meg. Soros üzenetet nem tartalamaz, csak státusz és range információt.

crc_sae_leg

Revizió előtti szabvány által előírt legacy CRC számítás és beillesztés.

crc_sae

SAE J2716 által előírt CRC számítás és beillesztés.

crc_infineon

Infineon szenzorok által használt CRC számítás és beillesztés.

counter_4b

4 bites számláló, paraméterben adott irányba számol. Adott csatornára minden egymást követő beillesztésére inkrementálódik/dekrementálódik.

counter_8b_MSN

8 bites számláló, legmagasabb helyiérték először, paraméterben adott irányba számol. Adott csatornára minden egymást követő beillesztésére inkrementálódik/dekrementálódik. **Single Secure** formátum alkalmazza.

counter_8b_MSN

8 bites számláló, legalacsonyabb helyiérték először, paraméterben adott irányba számol. Adott csatornára minden egymást követő beillesztésére inkrementálódik/dekrementálódik. **Single Secure** formátum alkalmazza.

4.8.3. AutoChange konverziók

no_auto

Nincs automata bemenetváltoztatás, üres függvénycsonk. Szerepe hogy hibás konfigurációkor vagy konfiguráció hiányakor és AutoChange engedélyezésekor a konfiguráció futtatható, nem okoz futásidejű hibát.

sweep_ro

Paraméterekben adott értékek között *sweep*elés megadott *step*pel, megadott időközönként. Határértékeken átfordul. Leírható vele pl.: egy szögszenzor körbefordulásakor leadott értékei.

sweep_b2b

Paraméterekben adott értékek között *sweep*elés megadott *step*pel, megadott időközönként. Határértékeken irányt vált, *back to back* sweepel. Adott tartomány értékei vizsgálhatók vele.

5. fejezet

Hardver

5.1. Áttekintés

Az eszköz egy főkártyából és négy kimeneti kártyából áll. A főkártyán helyezkedik el a mikrokontroller és a működéshez szükséges hardveres eszközök mint:

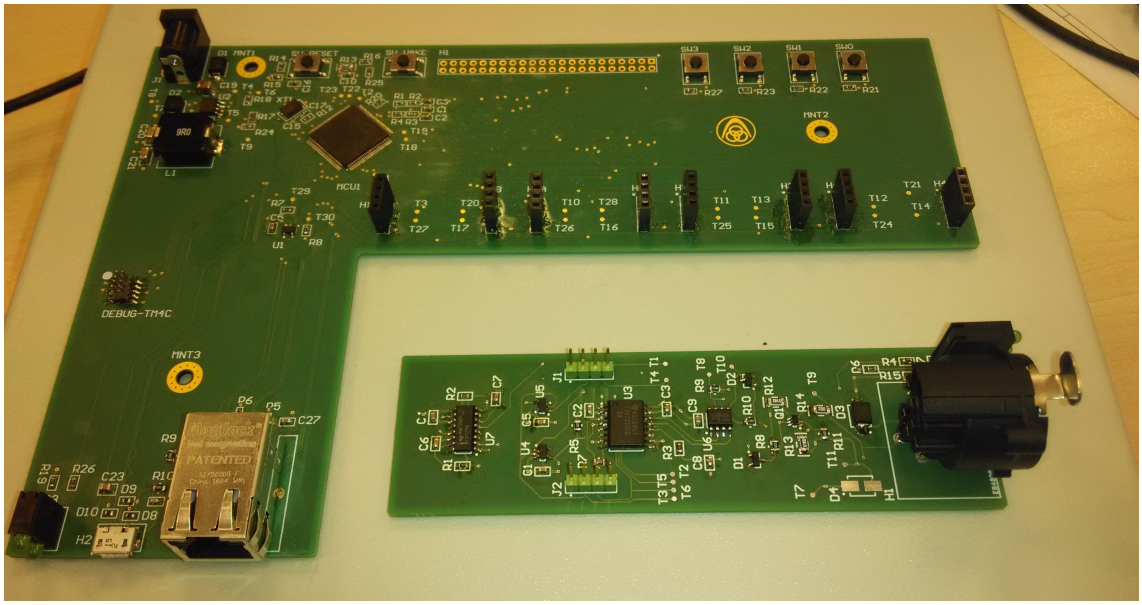
- Texas Instruments **TM4C129ENCPDT** mikrokontroller
- Analog Devices ADP2302, step-down konverter tápellátáshoz
- Microchip Node-Identity I2C **EEPROM**, **MAC** cím szolgáltatására
- MagJackes **Ethernet** csatlakozó, link-state és activity jelzéssel
- **Mini-B USB** csatlakozás
- Visszajelző **LED**, mikrokontorlleről és tápellátásról
- 4 db felhasználói gomb
- **Reset** és **Wake** gombok
- **EPI** perifériabusz kivezetés headerre esteleges bővítéshez (**LCD**)
- Kimeneti kártyák headerjei

A kimeneti kártyák a következőket tartamazzák:

- **XLR** csatlakozó illeszéshez
- Texas Instruments **ISO7331** 2/1 csatornás izolátor galvanikus leválasztáshoz
- NXP **PMBT3946YPN** tranzisztorpár a kimenet meghajtására
- Microchip **TC4427**, nagysebsségű MOSFET meghajtó a kimenethez
- Visszajelző **LED**-ek, **ECU** oldali tápellátásról, kimeneti tranzisztorpár meghajtásáról

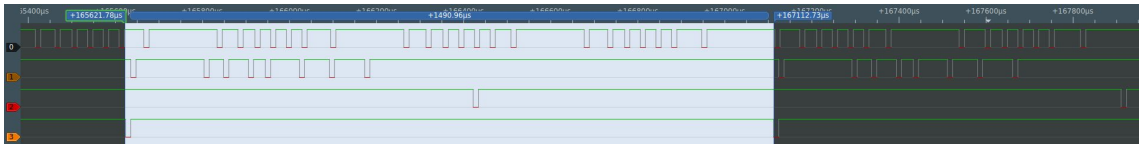
- Monostabil multivibrátor a tranzisztorpár meghajtásának, adatfolyam visszajelzéséhez
- Header a főkártya vezérlőjeleihez

A hardverrel szemben kívánalom volt hogy standard 1 unitos rackben elhelyezhető legyen (kettő egymás mellett), emiatt a csatlakozók, a tápellátás kivételével mind a frontpanel felé nézve kerültek elhelyezésre. A kártya mélyégét tekintve nem tölti ki a rack mélységét, mert szükségtelen, így akár közösen is táplálhatók. A frontpanelen továbbá **USB** kivezetés található, esetleges későbbi implementáció miatt. Az external peripheral interface (**EPI**) buszt kivezettem, esetleges kiegészítő modul ide helyezhető el. Pl.: **LCD** kijelző a **TivaWare** grafikus libraryjének használatával (ehhez az ott definiált driver implementásása szükséges, ezt megtettem **ILI9341**-es chiphez, de a grafikus felület nem lett összeállítva).



5.1. ábra. *A kész eszköz, főkártya és kimeneti fokozat*

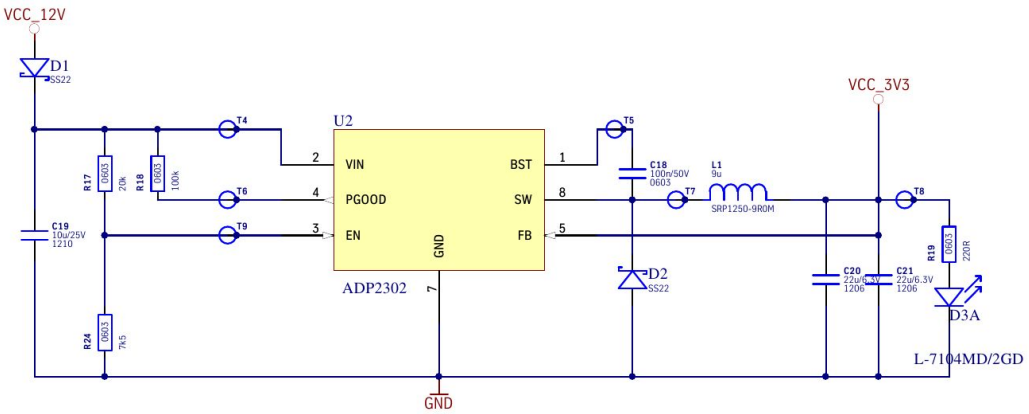
5.2. ábra. *A kész eszköz, összeállítva*



5.3. ábra. *A főkártyán előállított logikai **SENT** és **SPC** jelek, Channel 0: folytonos **SENT**, Channel 1: **SPC**, 3-as csatorna használva triggerelésre, jelölve, Channel 2,3: triggerpulzus generálás*

5.2. Tápellátás

Tápellátásról az Analog Devices **ADP2302**-al megvalósított kapcsolóüzemű step-down DC-DC táp gondoskodik. Bemeneti feszülsége $5 - 18V$ között mozoghat, fix $3.3V$-ra konvertál. $700kHz$-es kapcsolási frekvenciájána hála kis tekercsekkel és kondenzátorokkal használható. Túlfeszültség és túláram elleni védelemmel rendelkezik. [2]



5.4. ábra. *Tápellátás*

5.2.1. Méretezés

Catch-dióda választás

$$I_{d(atlag)} = (1 - \frac{V_{ki} + V_d}{V_{be} + V_d}) * I_{terhel(max)} \tag{5.1}$$

A dióda Schottky mert így kisebb a nyitóirányú feszültségesés, és nagyobb kapcsolási sebességekkel üzemeltethetők. Az átlagos áram így $1.392A$-re adódik a tervezett maximális terhelőáramnál ($2A$). A záróirányú letörési feszültsége is az előírt maximális bemeneti feszültség felett van ($20V$).

Tekercs választás

$$L = \frac{V_{be} - V_{ki}}{0.3 * I_{terhel(max)} * f_{switch}} * \frac{V_{ki} + V_d}{V_{be} + V_d} \tag{5.2}$$

A számítás alapján $6.3\mu H$-s tekercs megfelelő lenne, de annak érdekében hogy a akár $18V$-os tápfeszültségen is működhessen az eszköz a választott tekercset megnöveltem $9\mu H$-re. Ennek egy járulékos hatása hogy az áram hullámosságát csökkenti ($0.421A$ és $0.478A$ a két esetre).

$$\Delta I_{ripple} = \frac{V_{be} - V_{ki}}{L * f_{switch}} * \frac{V_{ki} + V_d}{V_{be} + V_d} \tag{5.3}$$

A csúcsáram amit a tekercsnek el kell viselni így maximálisan 2.478. Ezt a választott tekercs bőségesen kielégíti.

$$I_{peak} = I_{terhel(max)} + \frac{\Delta I_{ripple}}{2} \tag{5.4}$$

Kimeneti kapacitás választás

A kimenet feszültséghullámosságát 1%-ra méretezve $3\mu F$-os kimeneti kapacitás kellene.

$$\Delta V_{ripple} = \Delta I_{ripple} * (\frac{1}{8 * f_{switch} * C_{ki}} + ESR_{(C_{out})}) \tag{5.5}$$

A választott kondenzátoroknak kis soros ellenállása (ESR) és induktivitása (ESL) van, hiszen kisméretű kerámiakondenzátorok. Mivel az adatlap minimálisan 2 db $22\mu F$-os kapacitást javasol és $V_{out} < 5V$, és ez a komponens direktbe befolyásolja a hurokstabilitást, emiatt az adatlap alapján javasolt kondenzátorokat használtam.

Bemeneti kapacitás választás

A bemeneti szűrőkapacitásnak el kell viselnie a maximális bemeneti feszültséget és a maximális effektív áramot és hullámosságot. A maximásian elviselendő áramhullámosság $I_{terheles}/2$ lehet. Maximális effektív áram:

$$I_{be(eff)} = I_{terhel(max)} * \sqrt{D * (1 - D)} \tag{5.6}$$

Ahol D a kiöltési tényező, ami a ki/be meneti feszültégarányokkal számolható

$$D = \frac{V_{ki} + V_d}{V_{be} + V_d} \tag{5.7}$$

A választott $10\mu F$-es X7R-es kondenzátor ezen kitételeknek megfelel.

Rezisztív feszültségosztó. engedélyezés

A visszacsatolás feszültségosztója nem szükséges, lévén a fix $3.3V$ kimenetű chip van használva. A preciziós engedélyezés használva van, a bemeneti feszültség lassú növekedéséből származó problémák kiküszöbölésére (*undervoltage lockout*).

$$V_{startup} = (\frac{1.2V}{R_{bottom}} + 1.2\mu A) * R_{top} + 1.2V \tag{5.8}$$

A működés így csak stabil $4.5V$-os táp beállásakor kezdődhet.

5.2.2. NYÁK elhelyezés

A nyákon a modul a bemeneti tápcsatlakozó közelében helyezkedik el. A nagyáramú vonalak vastagabbra vannak választva és a lehetőségekhez képest rövidre fogva. Az átvezető viák mind föld és tápréteg felé többszörösek jobb vezetést biztosítva. A visszacsatoló hurok szintén rövidre van húzva. Hőelvezetés miatt a földréteg felé termálpad van nyitva az **IC** alatt.

5.3. Kimenet illesztése

A kimeneti fokozat külön kártyán került elhelyezésre, így az eszköz modulárisabb, kimenet könnyeben megváltoztatható, más csatlakozók használhatók, míg a vezérlőjeleket megfelelően kezeli.

A kártyának a következő jelekel dolgozik:

- **Enable_High**: a kimeneti félhíd felső ágának engedélyezés
- **Enable_Low**: a kimeneti félhíd alsó ágának engedélyezés
- **Sensor_Signal**: a kivezérlendő jel
- **SPC_trigger**: a triggerjel **SPC**-s esethez
- **GND**: föld
- **VCC_3V3**: táp

A fenti jelek 2 db 3-es headerön helyezkednek el, mindegyiken szerepeltetve a tápot és földet is.

5.3.1. Bemeneti jelátalakítás

A bemeneti **Sensor_Signal** jelet igyekeztem a szabvány szerinti logikai jellel megfeleltethetővé tenni, hogy a logigai és fizikia kimenet összehasonlítható legyen, jelformálás hatása vizsgálható, mérhető legyen akár egyszerűen oszcilloszkóppal. Emiatt a kimeneti tranzisztorpár meghajtásához invertálni kellett jelet, hiszen épp ellentétesen nyitnának tőle a félhíd tranzisztorai. Ehhez két egyszerű háromállapotú buffer-invertert használtam (Texas Instruments **SN74LVC1G240**). Az engedélyezőjeleknek felhasználtam az alsó-felső ág engedélyezőjeleit, így tiltás esetén az inverter után elhelyezett fel és lehúzóellenállások biztosíthatják a megfelelő ág zártságát.

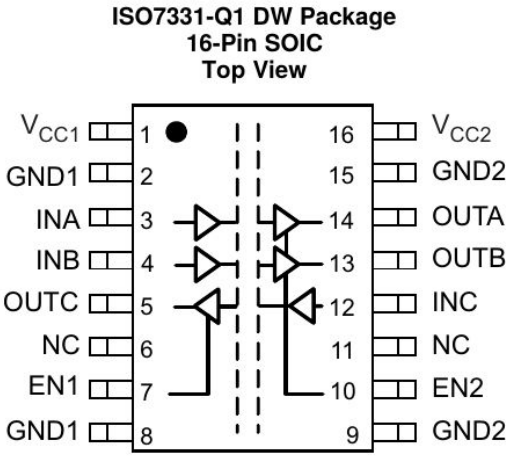
A visszajelzéshez hogy adatfolyam esetén a **LED** villanása érzékelhető legyen egy monostabil multivibrátort helyeztem el, jelváltás ezt triggereli hogy millisecig meghajthassa az adott ág visszajelző **LED**jét.

A hídág engedélyezőjelekkel választható a meghajtás típusa.

- **High Z**: Mindkét engedélyezőjel tiltott. Mindkét tranzisztor zárt.
- **Push-pull**: Mindkét engedélyezőjel engedélyezett. A tranzisztorpárok szinkronban ellentétesen kapcsolnak.
- **Active-High**: Felső ág open-kollektoros, alsó ág zárt. Szabvány szerint a lehúzó ellenállásról a vevőnek kell gondoskodnia.
- **Active-High**: Alsó ág open-kollektoros, felső ág zárt. Szabvány szerint a felhúzó ellenállásról a vevőnek kell gondoskodnia

5.3.2. Galvanikus leválasztás

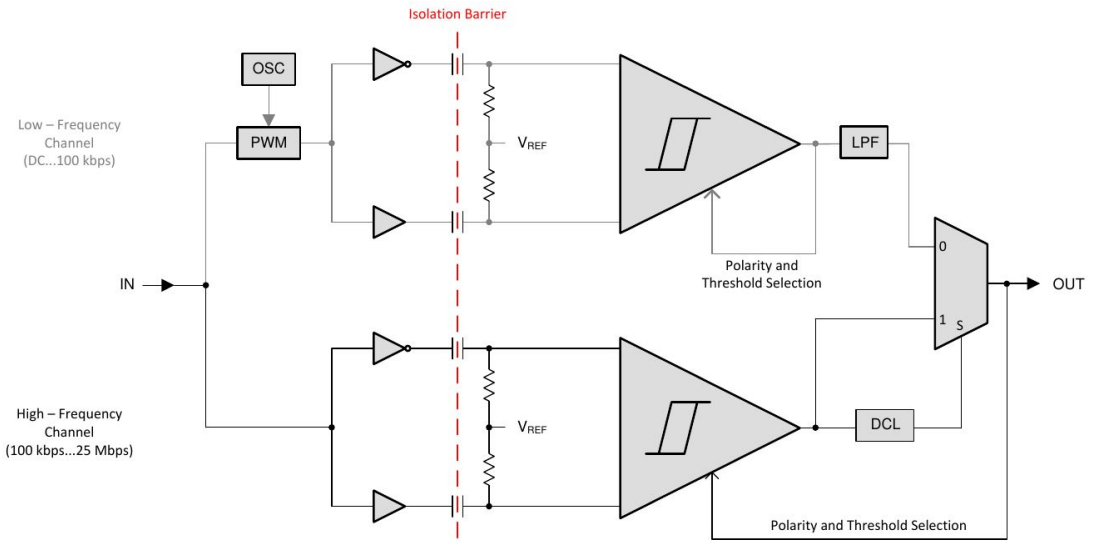
Mivel specifikáció szerint a szenzort az **ECU** táplálja, így kimeneti fokozat külön táptartományon van. A galvanikus leválasztásró a Texas Instruments **ISO7331CQDWR-Q1**-es chipje gondoskodik. Ez 2/1 csatornás konfigurációban van használva, 2 csatorna jele a hídágak meghajtása, míg a visszairányú vonal a kimenet visszavezetése. Az IC 3.3V és 5V-os jelszinteket is kezelni tud, 3.3V-on van használva a vezérlő felől, míg az **ECU** felől 5V-os tápot kap, a szintkonverziót elvégzi.



5.5. ábra. *A használt izolátor*

Mindkét tartományhoz tartozó oldal engedélyezőjelei a tartományhoz tartozó táppal fixen engedélyezve vannak. Ez megtehető mert **R5** és **R7**, fel és lehúzó ellenállás gondoskodik a félhíd nem meghajtott esetben nagyimpedanciás állapotba viteléről. [25]

Működését tekintve kapacitív izolátor. A beérkező *single-ended* jelet differenciálissá bontja, majd egy soros kapacitáson keresztül, kihasználva hogy a nagyfrekvenciás komponenseket átereszti, a túloldali komparátorra teszi ami dönt róla. Alacsony frekvenciák esetén a jelek átviteléhez így moduláció szükséges (**PWM**), erről a belső oszcillátor gondoskodik. Vétel esetén értelemszerűen egy aluláteresztő szűrővel demodulálni kell. Az **IC** automotive minősítésű. [25]



5.6. ábra. *A használt izolátor működési blokkdiagrammja*

5.3.3. Kimenet meghajtása

A kimenet meghajtásáról a **PMBT3946YPN** tranzisztorpár gondoskodik félhídba rendezve. Az **ECU** által táplált. A kellő sebességű jelváltások biztosítása érdekében Microchip

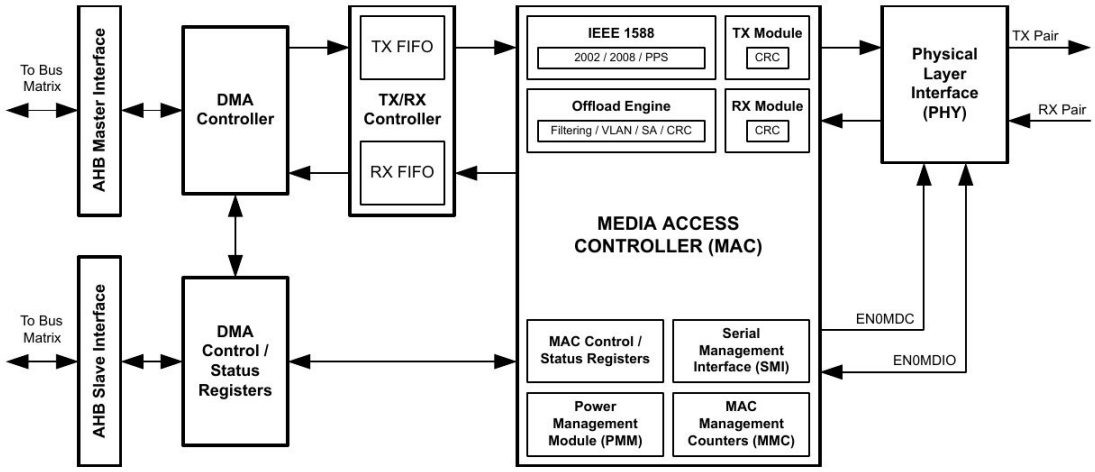
TC4427 MOSFET meghajtó gondoskodik. A kimenetek késleltetése egyeztetett. A bemenetek $300mV$-os hiszterézissel rendelkeznek, ez glitch és zavarszűrést lát el. A gatek töltéskiürítését tovább gyorsítja az ellenállásokkal párhuzamosan elhelyezett *RF Schottky* diódák, amik "kikapcsolják" a velük párhuzamos ellenállásokat váltáskor. Emiatt igen kis nyitófeszültséggel és kapacitással rendelkeznek ($V_f = 340mV, C = 1pF$).

A kimenet tápellátásáról a harmadik visszajelző **LED** ad információt, hiszen erről az **ECU** gondoskodik. A helytelen tápbekötés ellen soros dióda véd. Amennyiben az ezen történő feszültségesés túl nagy lenne (szabvány szerint), úgy elhagyható, nullohmos jumperellenállással helyettesíthető (ekkor védelem gyanánt **D4** behelyezésével fordított polaritás esetén rövidzár képezhető, a vevőre bízva ennek érzékelését, kezelését).

5.3.4. NYÁK elhelyezés

A kimeneti fokozat külön kártyán helyezkedik el, ennek oka hogy így modulárisabb, módosíthatóbb cserélhetőbb, másfajta csatlakozók használhatók. A csatlakozók jelen esetben **XLR**-ek. A két táptartomány az izolátorchip alatt került elválasztásra, ez a főkártya kontúrvonalára fekszik. Emiatt a tesztpontokat a könnyebb elérhetőség miatt kihoztam jobban. A kártyát a 2 header és a frontoldalon a csatlakozón elhelyezett csavarok rögzítik. A kártya négyrétegű, jel-táp-föld-jel konfigurációban. Alkatrészek csak egyoldalt helyezkednek el.

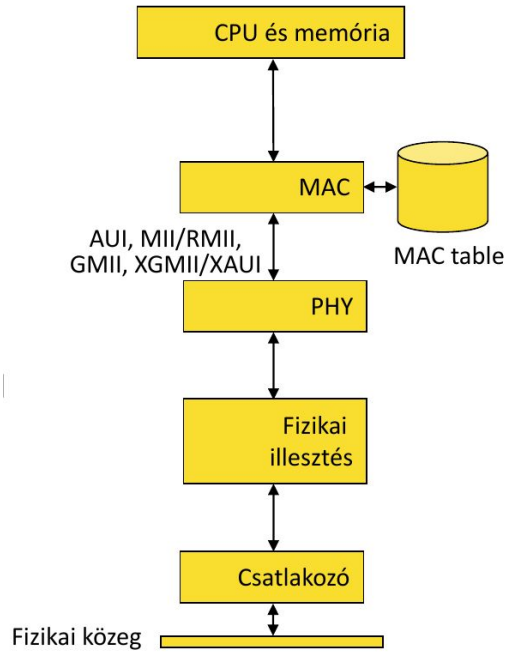
5.4. Ethernet



5.7. ábra. *Integrált **Ethernet** periféria blokkdiagrammja*

A használt **TM4C129ENCPD 10BASE-T/100BASE-T** Ethernet kontrollerel rendelkezik. Integrált **PHY** rétegget tartalmaz, így nem szükséged diszkrét fizikai rétegvezérlő chip alkalmazása standard sodort érpárokon. Ez direkt módon interfacelve van a közeghozzáférési (media access controll, **MAC**) réteghez. Fontos megjegyezni hogy amennyiben a fentiekkel élni kívánunk $25MHz$-s órajelezést kell biztosítanunk, a rétegeknek. Külvilég fele így az interfész leegyszerűsödik mindössze a küldő fogadó differenciális jelpárokból

(**Tx**/**Rx**), státuszjelző vonalakból (**ENAxLEDx**) és a referenciát beállító **RBIAS** ellenálásból áll.

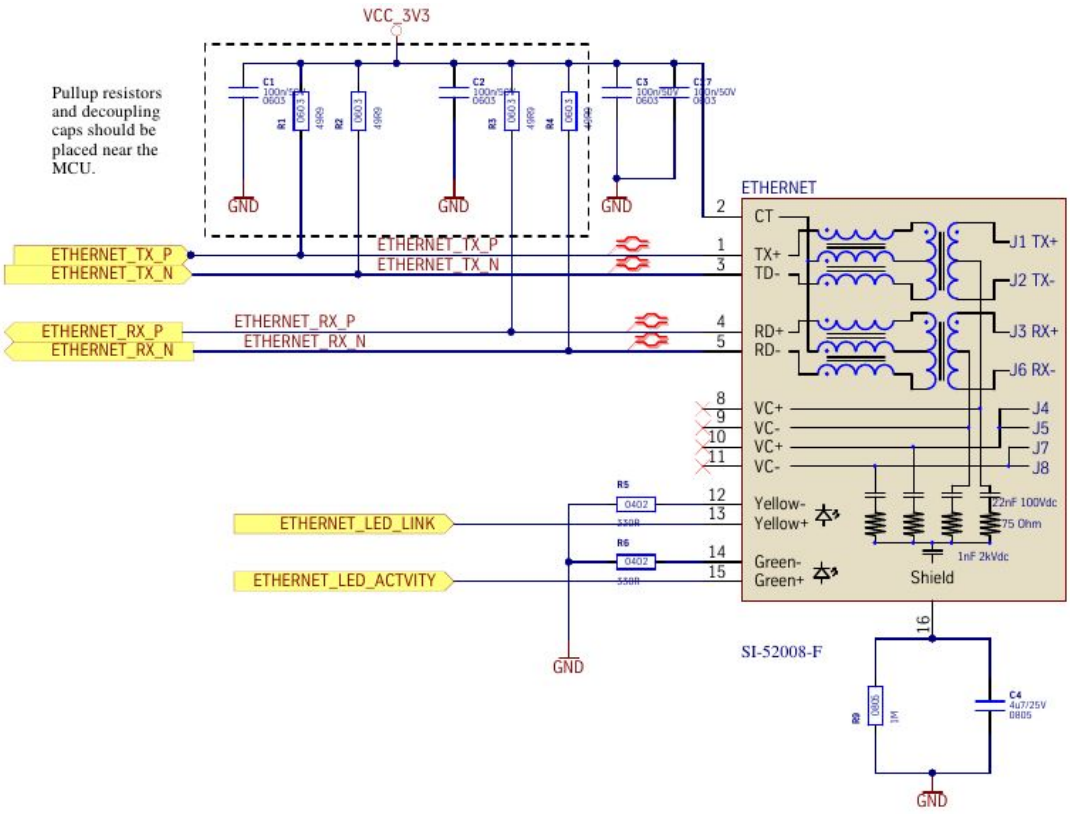


5.8. ábra. *Ethernet rétegek, [21]*

A **PHY** feladata a megfelelő vezértlő és adatjelek biztos a használt közegre. Mivel a szabvány megköveteli ezért képes automatikus *opcióegyeztetés*re, 4 protokollt támogat (**10/100**, **Full/Halfduplex** tetszőleges kombinációban). Automata *crossover* elvégzésére is képes (**Auto-MDIX**). Amennyiben a csatlakoztatott eszköz is autocrossoverképes úgy **IEE 802.3** szerinti véletelen algoritmus alapján történik a váltás.

A közeghozzáférési réteg feladata az átvitel vezérlése, és ethernet keretek összeállítása. Ez többek között magában foglalja küldéskor a preambulum és start-of-frame genrálást, flow-controlt, időbélyeg generálást, a keret **CRC**-jének kitöltését, pause keretek generálását. Küldés esetén a feladata a preambulum, fejlécmezők lefejése, dekódolása, nekik megfelelően a keretek bufferelése és státuszok tárolása, ellőnőrzőösszeg ellenőrzés és keretszűrés.

A fizikai réteg feladata a megfelelő jelszintek és időzítések biztosítása a kimeneti differenciális jelvezetékekre. Mivel a szabványos ethernet kábelekkel viszonylag nagytávolságú összeköttetések valósíthatók meg így a szabvány galvanikus leválasztást ír elő. A jelek differenciálisan, mágnesesen csatoltak. Ennek megfelelően szükséges impulzustranszformátorok használata. A használt transzformátor jelen esetben az **RJ45**-ös csatlakózóba van integrálva. Négy ellenállás terminálja a vonalakat, csatolásmenetesítésről pedig négy kondenzátor gondoskodik. Ezek választása a [23] alapján történt. Elhelyezésük a kontrollerhez közel, minimalizálva a zajhatásokat.[1][3][23]



5.9. ábra. *Ethernet vonalak diagrammja*

Az egyedi **MAC** címről a Microchip **24AA02E48** I^2C-s Node Identity soros **EEPROM** gondoskodik, ebben törölhetetlenül, **EUI-48**-as azonosítók vannak, bootprocessz során ezek felovasottak és **MAC** címként használtak. A memória másra nem használt, hiszen a belső a kontroller belső memóriája elegendő, gyorsabban elérhető.[14]

5.4.1. NYÁK elhelyezés

A nagysebességű vonalak elhelyezése a PCB-n körültekintést igényel, jelintegritás, elektromágneses interferencia és reflexiók szempontjából. A differenciális jelvezetékek hosszegyeztetettek. Az irányelveknek megfelelően a *Tx/Rx* párok külön vezetettek. Bár a layerváltás nem javasolt, de szükséges volt így a váltást azonos hosszpozícióban valósítottam meg. A hullámimpedancia 50 Ohm, a differenciális impedancia 100 Ohm-ra van egyeztetve a vonalközzel és szélességgel. A számítás egyszerűsítéséhez a **Saturn PCB toolkit**-et használtam. A csatlakozó alól eltávolítottam a táp-föld rétegeket.[23]

5.5. USB

Az **USB** vonala szintén nagysebességűek ezért az Ethernethez hasonló megfontolásokkal éltem az elrendezésüknél. Az elérni kívánt hullámimpedancia ebben a esetben 45 Ohm, míg a differenciális 90 Ohm. A kapcsolás egyszerű deviceként történő üzemeltetést tesz lehetővé,

host illetve **OTG**-módot nem (ehhez tápbiztosítás lenne szükséges). Csatlakozója micro, B-típusú ennek megfelelően.[23][13]

5.5.1. NYÁK elhelyezés

A csatlakozó a frontpanelen helyezkedi el, az **RJ45**-össel vonalban. ESD tranziensek elleni védelemként a vonalakon varisztorok vannak elhelyezve, továbbá a tápvonal szűrt.[13]

5.6. Órajelezés

Az integrált Ethernet **PHY** üzemeltetéséhez $25MHz$-es órajel szükséges. Emiatt ezt választottam a főkristálynak, és belső **PLL**-el szorzom fel a kívánt $120Mhz$-re, amivel a maximális működési sebessége biztosítható az **MCU**-nak. Az elrendezés *Pierce*-oszcillátoros. A $32.768kHz$-es realtime/hibernációs órajel nem használt. A terhelő kapacitások számítása a következő képlet alapján történt:

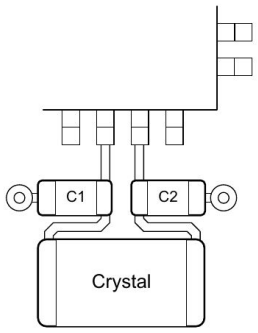
$$C_L = \frac{C_1 * C_2}{C_1 + C_2} + C_{szort} \tag{5.9}$$

Ahol a C_L a kristálygyártó által specifikált kapacitás. C_szrt a PCB elrendezés, trackszélesség és a mikrokontroller $OSCx$ pinjeinek függvénye tipikusan $2 - 4pF$. Sarkított esetben mérhető az adott pineken. Mivel C_1-et és C_2-őt azonosra választottam így a képlet egszerűsödik[23][24]

$$C_{1,2} = (C_L - 3pF) * 2 \tag{5.10}$$

5.6.1. NYÁK elhelyezés

A **PCB**-n az oszcillátor számára földsziget van képezve. Ennek célja hogy minimalizálja a földrétegen megjelenő zavarokat. A kondenzátorok a és a kristály a lehetőségekhez mérten a pinekhez közel kerültk elhelyezésre. Az **MCU** tápszűrő kondenzátorai a hátoldalon a hozzájuk tartozó pinek alatt vannak elhelyezve. Más kristály használatához a soros ellenálláson teljesítmény disszipálható, így a határérték tartható, ennek pad van képezve. Amennyiben ez nem szükséges úgy nullohmos jumperellenállás használható.[23]



5.10. ábra. *Javasolt oszcillátorelrendezés*

5.7. Lokális vezérlés

Lokális vezérléshez 4 négy nyomógomb került felhelyezésre, ezek aktív alacsonyak. Az eszköz rendelkezik továbbá **RESET** és **WAKE** gombokkal. Esetleges egyéb eszköz illesztéséhez, például kiejlző illesztéséhez az **EPI** perifériabuszt kivezettem egy headerre. Ez egy 8/16/32 bites párhuzamos busz. Üzemeltethető **SDRAM** módban, külső memória illesztése esetén (ekkor a nagysebességű vonalak illesztésére gondot kell fordítani). Használható továbbá **host-bus** módban, aszinkronként *strobe* jelekkel (8051 jellegű buszvezérlőjelek). Általános célúként használva az órajelezés, órajel engedélyezés, keretjelek, irás/olvasás *strobe*-ok konfigurálhatók.

A **TM4C129ENCPD** a családjának nagyobb tagjával ellentétben nem rendelkezik integrált **LCD** vezérlővel. Az **EPI** buszra így párhuzamos elérést támogató **LCD** vezérlő illeszthető (sajnos a soros vezérlőkhöz jellegzetesen használt **SSI** perifériák mind el vannak használva a jelgeneráláshoz). **ILI9341**-es vezérlő portolása megtörtént a **TivaWare** grafikus könyvtárához ehhez az elrendezéshez, de maga a grafikus felület nem lett összeállítva. [24]

6. fejezet

Kommunikáció

6.1. ASN.1

Az **ASN.1** nyelv különböző alkalmazások közötti üzenetek leírására szolgál, mint ilyen magasszintű üzenetleírási formákkal rendelkezik. Alkalmas adattípusok formális leírására amellyel a leírt adattípusok átalakíthatók bitfolyammá és továbbíthatók, ezáltal heterogén rendszerek közötti kommunikációt tesz lehetővé. Közös szabványa az International Organization for Standardization (**ISO**), International Electrotechnical Commission (**IEC**), és International Telecommunication Union Telecommunication Standardization Sector **ITU-T** szervezeteknek. Legutolsó revíziója az **X.680**.

Az ASN.1 maga egy **absztrakt** szintakszist definiál az üzenetekre, melyek nem korlátozzák az bitfolyammá kódolás módját, a konkrét reperezentáció létrehozását. Konkrét szintakszisnak nevezzük a küldeni kívánt adatreprezentációkat egy adott programozási nyelven. Ezek figyelembeveszik az adott nyelv szabályait és a gép architektúráját. Az **ASN.1** ezzel szemben absztrakt, nyelvfüggetlen, saját adattípusokkal és típuskonstruktorokból áll, melyekkel az alap adatípusokból komplex típusok állíthatók elő.

Szükséges továbbá a kommunikáció felépítéséhez egy **átviteli** szintakszis is. Ez gyakorlatilag az absztrakt szintakszis által leírt elem reprezentációját ada meg az átvitel során. Az átvitel továbbra is absztrakt szintakszistól függ, az átviteli szintakszis csak megadja hogyan továbbítjuk/hozzuk létre a bájtfolyamot belőle. Ez leggyakrabban különböző kódolási/dekódolási szabályokat alapján történik:

- Basic Encoding Rules **BER**
- Canonical Encoding Rules **CER**
- Distinguished Encoding Rules **DER**
- XML Encoding Rules **XER**
- Canonical XML Encoding Rules **CXER**
- Extended XML Encoding Rules **EXER**

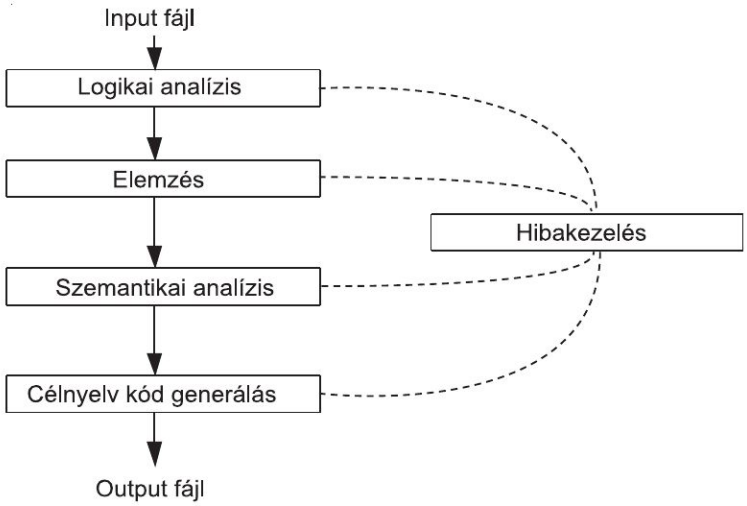
- Packed Encoding Rules **PER**
- Unaligned Packed Encoding Rules **uPER**
- Canonical Packed Encoding Rules **CPER**
- Generic String Encoding Rules **GSER**

Ezek közül általam a basic encoding rules, **BER** került alkalmazásra.

Az **ASN.1** a kódolási szabályokkal együtt így egy architektúra és az implementáció nyelvétől független strukturált adatátvitelt tesz lehetővé. **OSI** modelbe helyezve leginkább a prezentációs/megjelenítési rétegben helyezkedik el, hiszen feladata hogy az adatok a végfelhasználó által használható formában rendelkezésre álljanak. [10][5][16]

6.2. ASN.1 fordítók

A fordító általonsságban egy olyan eszköz, mely beolvas egy adott forrásnyelven íródott programot és egy másik, célnyelvnek megfelelő szintakszisú és architektúrának megfelelő ábrázolású formára hozza. Forrásfájlunk így az **ASN.1** szintakszisú leíró, míg kimenetünk a célnyelv (esetünkben **C** és **Java**) forrásfájlai amik tartalmazzás a választott kódolásnak megfelelő enkódoló/dekódoló kódot és az adatstruktúrák konkrét reprezentációját.



6.1. ábra. *Fordítás lépései*

Fordítókból ennek megfelelően sokféle van a piacon, különféle célnyelvekre változó komplexitással és licenszeléssel. Alapvetően opensource fordítót kívántam használni, a működés jobb megismerése miatt. Ennek megfelelően elsősorban **GPL/LGPL** licenszűek között néztem körül. Sajnos nem találtam olyat mely a számomra kontrolleren futtatható **C** és vezérlő PC-n futó **Java** kódot egyaránt generált volna így két különböző eszközt használtam ezekre. [5][16]

6.2.1. asn1scc/Tastee

A választott **C** fordítóval szembeni követelmény volt hogy mikrokontrolleren futtatható legyen a kód, minnél inkább valósidejűen. Emiatt nem kívántam olyan fordítót használni mely dinamikusan memóriafoglalást használ (a legtöbb ilyen, hiszen a **BER** kódolás lehetőséget nyújt akár végtelen egymásbaágyazására a struktúráknak)

A választásom végül as Európai Űrügynökség **ESA**, Tastee csomagjában használt **asn1scc** fordítóra esett. Ez a fenti problémákat kolátozásokkal hidalja át. A forrásfájl fordításakor nem engedi olyan leírásból kód generálását ami ilyen konkrét szintakszisokra lehetőséget biztosítana.

A megkötések a következők:

- `SEQUENCE OF` és `SET OF` típusoknak `SIZE` megkötéssel kell rendelkeznie
- `OCTET STRING` és `BIT STRING` típusoknak `SIZE` megkötéssel kell rendelkeznie
- `IA5String` és `NumericString` és általánosságban a stringtípusoknak `SIZE` megkötéssel kell rendelkeznie
- bővíthető `CHOICE`, `SEQUENCE` és enumerációt nem tartalmazhat

A fenti megkötésekkel élve, minden konkrét létrehozandó struktúra mérete ismert fordításidőbe. A fordító továbbá nem támogatja az **ASN.1** pár fejlettebb feature-ét (parametrizáció, Information Class Object), de ezeket nem is kívántam használni. Az egyszerűség jegyébe az **ASN.1** leírásba csak a váltandó üzeneteket vettem bele, azokat is alkalmazásspecifikus **TAG**-ekkel azonosítva az egyértelműség miatt.

Az **asn1scc** továbbá nem képes **BER** enkódolására az összetett struktúráknak önmagától (**XER** és **uPER** az elsősorban támogatott). A *StringTemplate* library segítségével viszont ez bővíthető, mivel az alap adattípusok **BER** kódoló/dekódoló függvényeivel rendelkezik. Mivel ezt a toolt nem ismertem és viszonylag nem túl sok függvényhez kellet összeállítani alaptípusokból a kódoló függyényeket, ezt így inkább manuálisan tettem meg (`asn1_ber.c/h`).

További hiányosság volt még a lebegőpontos számok kezelésénél a `WORD_SIZE` definíció változtatása esetén, hogy a lebegőpontos átalakítás függvényei nem vették figyelembe, ez is javítva lett.

A leírófájl és generált kódok és struktúrák a kontroller kódjának **ASN1** könyvtárában találhatók.

`https://essr.esa.int/project/asn1scc-asn-1-space-certifiable-compiler`

6.2.2. jASN1/openMUC

A **Java** oldali kódgeneráláshoz a szintén **ESA** által is használt, **Fraunhofer Institute openMUC** projektében lévő **jASN1** fordítót használtam. Ez **BER** és **DER** kódolásra képes volt, és a mikrokontrolleres oldal által támasztott extra elvárások itt meddőek.

Kisebb hibája volt hogy az **IEEE 754** szerinti lebegőpontos szám ábrázolása esetén 2 pozitív egész hatványai esetén az exponenst rosszul dekódolta, ezt javítottam.

A generált **Java** osztályok a PC oldali **API** *generated* packagében találhatók. Mivel a valós értékek feldolgozása módosítva lett ezért az aggregált **jar** fileok helyett mellékeltem a módosított forrásfájlokat.

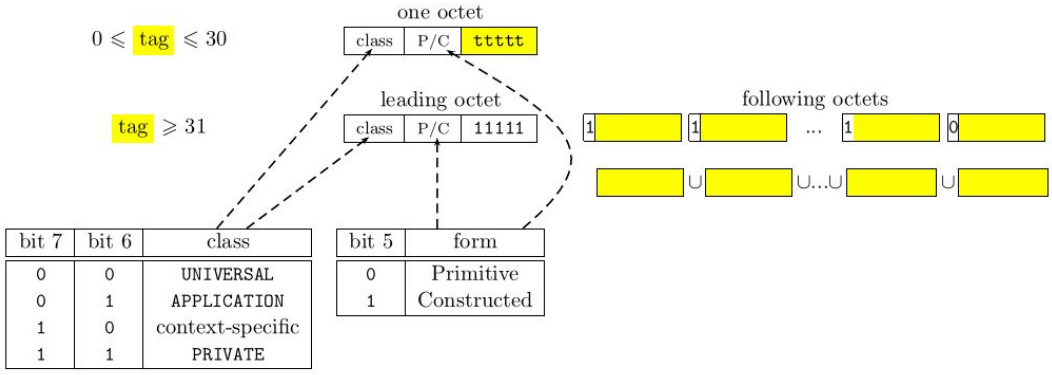
`https://www.openmuc.org/asn1/`

6.3. Basic Encoding Rules, BER

A használt átviteli szintaxis a basic endcoding rules, **BER**. A **BER** az **ASN.1** "eredeti" kódolási formája, első kiadásában is szerepelt.

6.2. ábra. *TLV-k egymásbaágyazása*

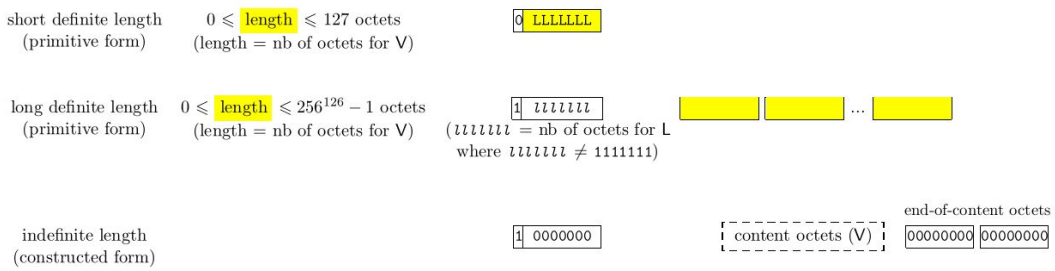
A **BER** átviteli szintakszis mindig **TLV** (type, length, value) hármasokból áll. Mind **T**, **L**, és **V** oktetek és big-endianok. Egymásbaágyazásról az gondoskodik hogy maga a **V**, value lehet egy másik **TLV** érték. Ezt az esetet jelzi a **TAG** primitive/constructed flagje. A **T** tageknek az alábbi formátum valamelyikének kell megfelelniük:



6.3. ábra. *BER TAG-ek*

Jelen alkalmazásában minden küldendő/fogadandó üzenet `APPLICATION` tageket használ (abban az esetben is ha alaptípusnak is megfelelthető lenne) a kapott üzenet könnyebb diszkriminációja érdekében. Így az első **TAG** alapján könnyen eldönthető az üzenet és neki megfelelő dekódolófüggvény. A belső **TAG**-ek már lehetnek univerzálisak. Automata **TAG**-elés használva van.

A hosszleírásnak az alábbi formátumok valamelyikének kell megfelelni:



6.4. ábra. *BER Hosszleírók*

Érdemes megjegyezni, hogy a **BER** (a **DER**-el ellentétben) lehetőséget biztosít indefinit hosszúságú üzenet létrehozására. Ebben az esetben a hosszmezőbe 0x80 kerül és a **TLV** végét egy dupla nulla oktet jelzi (0x0000). Ennek értelme hogy erőforrásszűk környezetben nem szükséges a teljes üzenetet ismerni, hosszát leszámolni, lehet on-the-fly enkódolni és küldeni. Emiatt használtam ezt a módot kontroller irányából történő üzenetküldésre (hiszen itt a kódolófüggvények a komplex típusokra manuálisan lettek írva). Mindenesetre a definit hossz is használható, hiszen az **asn1scc** működése (dinamikus memória mellőzése) miatt a maximális kódolandó oktetszám ismert. [10][5][16]

6.4. Hozzáférés szintjei

A szoftveráttekintés során ismertetett hozzáférési szintekhez konfigurációs és vezérlőüzenetek tartoznak. Alapvetően igyekeztem a dekódolás gyorsítása érdekében a belső adatstruktúrákat megőrizni és egybentartani, így egyből a megfelelő mailboxba dobható módosítás nélkül. Bizonyos extra információk bevételére viszont szükség volt (csatornaválasztás stb...). Ez gondot nem okoz hiszen specifikus `APPLICATION` tag-el látok el minden normál működés esetén várt üzenetet, így ezek mintegy wrappelik az adatstruktúra és kiegészítő **TLV**-ket. Minden normál működés közben várt üzenet `Msg` suffixel van ellátva.

6.4.1. Vezérlő és konfigurációs üzenetek

Csatornakonfiguráció

```
Sensor-t ::= SEQUENCE{
        name FnName,
        tick-usec REAL,
        mode Mode,
        lowticks INTEGER (3..12),
        master-pulse-min REAL,
        master-pulse-max REAL,
        msg-delay INTEGER (0..255),
        output-type Output-type
}

ChannelCfg-t ::= SEQUENCE {
        sensor Sensor-t,
        output Output,
        level Level,
```

```
        format Format-t,
        hformat HFormat-t
}

Channel ::= INTEGER (0..15)     --bitwise positions

ConfigMsg ::= [APPLICATION 5] SEQUENCE{
        ch Channel,
        cfg ChannelCfg-t
}
```

A szenzorkonfigurációs üzenet szükséges, minden nem alacsonyszintű működés esetén. A csatornakonfigurációs üzenet leírja a szenzort, a működés szintjét, kimenetét és az esetlegesen alkalmazott közép és magasszintű működés struktúráit.

A végül küldött üzenet ezenfelől tartalmazza kiválasztott csatornák leíróját amikre a konfigurációt alkalmazni kívánjuk. Ez a kiválasztás bitwise, így több csatornán azonos konfiguráció használata esetén elég egyszer küldeni (ami nem hátrányos lévén egy minden középszintű blokkot és magasszintű hozzáférést használó konfiguráció 1k-s is lehet).

Engedélyezés

```
EnableMsg ::= [APPLICATION 4] SEQUENCE {
        ch Channel,
        enable BOOLEAN
}
```

Az engedélyezés a már megismert csatornaleíróval és a boolean értékekel történik.

Státusz

```
StatusMsg ::= [APPLICATION 0] IA5String (SIZE(32))
```

Egyszerű string alapú státuszüzenet küldése. PC oldalon fogadva ezek lehetnek nyugtázó üzenetek. Mikrokontrolleren fogadva a tesztelést könnyítik meg, standard outputra íródnak mint stringek.

Konfiguráció mentés/betöltés

```
SaveLoad ::= ENUMERATED {save (1), load (2)}
SaveSlot ::= INTEGER (0..15)
SaveLoadMsg ::= [APPLICATION 6] SEQUENCE{
        ch Channel,
        slot SaveSlot,
        sl SaveLoad
}
```

Konfigurációk mentése és betöltése az **EEPROM**-ba/ból. A bitwise csatornaleíró megmondja melyik csatornákra öltjük be a slot-ban megadott konfigurációt. Mentésnél több csatorna megadása értelmetlen így ezt figyelmen kívül hagyja a feldolgozótaszk.

Lookuptábla mentés

```
StoreTableMsg ::= [APPLICATION 8] SEQUENCE{
        start REAL,
        end REAL,
        tablesize INTEGER (1..128),
        table SEQUENCE SIZE (1..128) OF REAL
```

```
}
```

A lookuptábla leírása a kezdeti és végértékének megadásával, nagyságának definiálásával és értékeinek felsorolásával történik. A független változó felbontása egyenletes.

AutoChange konfiguráció

```
AutoChangeMsg ::= [APPLICATION 7] SEQUENCE{
        ch Channel,
        interval REAL,
        assoc INTEGER,
        fnname FnName,
        params Params
}
```

`AutoChange` task konfiguráláshoz csatornakiválasztás, lépésköz választás szükséges. Meg kell továbbá adni melyik magasszintű paramétert kívánjuk periodikusan módosítani, melyik `AutoChange` függvénny alapján, miféle paraméterezéssel.

AutoChange engedélyezés

```
AutoChangeEnableMsg ::= [APPLICATION 9] SEQUENCE{
        enable BOOLEAN
}
```

`AutoChange` task indításának üzenet, egyszerű boolean ami felszólítja a kontrollert a task létrehozására. Leállítás a hamis értékére történik.

6.4.2. Alacsonyszintű/Bitminta alapú jelszintézis üzenetei

```
max-buffer INTEGER ::= 256
Buffer ::= OCTET STRING (SIZE(0..max-buffer))
RawMsg ::= [APPLICATION 1] SEQUENCE {
        ch Channel,
        buff Buffer
}
```

A bitminta alapú jelszintézis üzenete hordozza maga bitminta bufferét, a csatornák leíróját amelyeken a bitmintát ki akarjuk vezérelni. A csatonaleíró bitwise engedélyezésű, így azonos minta egyszerűen alkalmazható több csatornára. A bufferhossz a hardver által megengedett legnagyobb **DMA**-zható memóriatartomány. Ez $3\mu s$ tickidővel $6ms$-nyi jel leírását engedélyezi.

6.4.3. Középszintű/Blokkalapú jelszintézis üzenetei

```
ValuesMsg ::= [APPLICATION 2] SEQUENCE{
        ch Channel,
        idx INTEGER (0..max-dataunits),
        val INTEGER
}
```

Egyes mintagenerátorblokkok módosításához meg kell adnunk a csatornát, a módosítani kívánt blokk indexét és értékét. Magát a blokkok szekvenciális leírását a konfigurációs üzenet tárolja, ennek előzetes megadása szükséges.

6.4.4. Magasszintű/értékalapú jelszintézis üzenetei

```
HValuesMsg ::= [APPLICATION 3] SEQUENCE{
        ch Channel,
        idx INTEGER (0..max-dataunits),
        val REAL
}
```

Egyes fizikai valóságot reprezentáló paraméterek módosításához meg kell adnunk a csatornát, a módosítani kívánt paraméter indexét és értékét. A paraméterkonverziók szekvenciális leírását a konfigurációs üzenet tárolja, ennek előzetes megadása szükséges.

7. fejezet

Konklúzió

Az eszköz az előírt jelgenerálási feladatokat ellátja. Hardveresen képes előállítani a kívánt jeleket, a megfelelő jelszinteket és ütemést biztosítani. Különböző **SENT** és **SPC** protokollú **Infineon** szenzorokkal tesztkörnyezetbe ilesztve a leadott jelek megfelelőknek bizonyultak, de a tesztelés még folyamatban van. Kisebb időzítési, vélhetően szoftveres hibák kiütköztek. Szinkron **SPC** üzem esetén szporadikusan üzenetkimaradás történik (közelítőleg 100 ból 1). Ennek kiméréséhez, megfelelő hosszúságú mintatár képzése és hozzá tartozó loggolás szükséges. Ezzel diszkriminálhatjuk a jelenség okát az őt létrehozó paraméterek alapján (speciális értékekre történik-e, kései-e a mintabellítás, lassú-e mintagenerálás, **DMA** elmaradás történik-e stb...). Lassú generálás esetén a generátorfüggvények optimalizációja a megoldás. **DMA** problémák ellenőrizhetők javíthatók az átviteli beállítások változtatásával (bufferméret, prioritás, burst mód stb...)

A lokális kezeléshez és visszjelzéshez **LCD**-modul illesztését terveztem. Mivel a kontroller nem rendelkezik **LCD**-vezérlő perifériával és az SSI perifériák mind fel lettek használva jelgenerálásoz, így az **EPI** buszt kívántam használni egy külső kontroller párhuzamos eléréséhez. Ehhez az **EPI** busz-t használó drivert írtam **ILI9341**-es kontrollerhez. A **TivaWare grlib** grafikus könyvtárat használható kezelőfelüet összeállítása végül nem történt meg, de a hardverillesztőréteg hívásait portoltam.

A kezdetben elegáns megoldásnak tűnő **ASN1**-es kommunikáció, végül az implementációt bonyolította, mindkét választott fordítóval kisebb hibák, alkalmazási nehézségek akadtak (valós számok értelmezése, **BER** hiánya). Valószínűleg egyszerű parancssor jellegű vezérlés jobb, egyszerűbb megoldás lett volna.

A generált osztályokat használva a **Java API**-val a megfelelő üzenetek objektumai előállíthatók és kódolhatók/dekódolhatók. Rövid demonstráló kódok vannak a **Java API**-ban mellékelve a különböző hozzáférési szintek üzeneteinek előállításához és használatukhoz, valamint a szenzorkonfigurációs osztályok összeállításához. A kommunikáció külön szálon fut az **java.nio** felhasználásával. Ez üzenetsorokkal rendelkezik amibe a küldendő üzenetek objektumai bedobhatók. Ez a felépítés működöképes, feladatát ellátja, de nem végleges.

Terveztem továbbá egy egyszerű PC oldali, a **Java API**-t használó **GUI**-t, de ez nem

került befejezésre lévén backend kommunikációs szál változhat és prioritásosabb feladatok, debuggolás előnyt élveztek.

Az eszköz eredetileg egyunit magas rackszekrényben került volna elhelyezésre, két darab egymás mellett. Ez végül változott, egyszerű műszerdobozba kerül, így a kártyadimenziók nem illeszkednek tökéletesen. A kimeneti kártya **XLR** csatlakozójának magassága az rack unitmagasságát közelíti, emiatt került a főkártya elhelyezése headerökkel a csatlakozó középvonalához. A rögzítést így a csatlakozók átellenes frontoldali csavarjai és headerök biztosítják. A unitmagasság megkötése nélkül valószínüleg kultúráltabban is elrendezhető, rögzíthető.

Köszönetnyilvánítás

Ezúton is szeretném megköszönni Dr. Balogh Andrásnak (és rajta keresztül a ThyssenKrupp Prestának) és Dr. Sujbert Lászlónak (valamint rajta keresztül BME-MIT tanszék oktatóinak és dolgozóinak) a szakmai segítségüket, nem csak ezen téma kapcsán, hanem visszamenõleg is mind az önálló laborooratórium, mind az elmúlt évek egyetemi tárgyai és feladatai során a nyújtott tudásért és a biztosított eszközökért. Mivel tisztában vagyok, hogy néha nem egyszerû velem együttmûködni, ezért feltétlen köszönetet érdemel a nem szakmai szempontokat nézve a felém tanúsított türelmük és hozzáállásuk.

Ábrák jegyzéke

Irodalomjegyzék

[1] Analog Devices. *A Beginner's Guide to Ethernet 802.3*, 2005. Technical Notes.

[2] Analog Devices. *ADP2302/ADP2303 Nonsynchronous Step-Down Regulators*, 2012. Data Sheet.

[3] Joann Zimmerman Charles E. Spurgeoun. *Ethernet: The Definitive Guide*. O'Reilly, 2nd edition, 2014.

[4] IFSA Dr. Sergey Y. Yurish, International Frequency Sensor Association. Digital and intelligent sensors and sensor systems: Practical design, 2012.

[5] Olivier Dubuisson. *ASN.1: Communication between Heterogeneous Systems*. OSS Nokalva, 3rd edition, 2000.

[6] Jacob Fraden. *Handbook of Modern Sensors, Physics, Designs, and Applications*. Springer, 4th edition, 2010.

[7] Infineon Technologies AG. *TLE4998C3, TLE4998C4,Programmable Linear Hall Sensor*, 2008. Data Sheet.

[8] Infineon Technologies AG. *TLE4998S8D Grade1 High Performance Programmable Dual Linear Hall Sensor*, 2014. Data Sheet.

[9] Infineon Technologies AG. *TLE5014, Angle Sensor*, 2016. Data Sheet.

[10] ITU-T. *ITU-T RECOMMENDATION X.690: Information technology – ASN.1 encoding rules*, 2002.

[11] Andrew Mason Junwei Zhou. Communication buses and protocols for sensor networks, 2002.

[12] Josef Kramoliš. *MPC574xP SENT/SPC Driver*. Freescale Semiconductor, 2014. Application Note.

[13] Littlefuse Inc. *Protecting the Universal Serial Bus from Overvoltage and Overcurrent Threats*, 2001. Application Note.

[14] Microchip. *24AA02E48/24AA02E48 Serial EEPROMs with EUI-48 Node Identity*, 2010. Data Sheet.

[15] NXP. *KMA232 Dual channel programmable angle sensor with SAE J2716 SENT*, 2013. Data Sheet.

[16] Papp András Poos Krisztián. Asn.1 nyelv a protokolltervezésben, 2004/8. Híradástechnika.

[17] SAE International. *SENT - Single Edge Nibble Transmission for Automotive Applications*, 2007.

[18] SAE International. *SENT - Single Edge Nibble Transmission for Automotive Applications*, 2010.

[19] Anthony Seely. *Multi-Channel SAE-J2716 (SENT) Decoder Using NHET*. Texas Instruments, 2010.

[20] Ian R. Sinclair. *Sensors and Transducers*. Newnes, 3rd edition, 2001.

[21] Dr. Kovácsházy Tamás. *Hálózatba kapcsolt beágyazott rendszerek*, 2016. Jegyzet.

[22] Texas Instruments. *SYS/BIOS Inter Process Communication (IPC) 1.25*, 2012. User's Guide.

[23] Texas Instruments. *System Design Guidelines for TM4C129x Family of Tiva C Series Microcontrollers*, 2013. Application Report.

[24] Texas Instruments. *Tiva TM TM4C129XNCZAD Microcontroller*, 2014. Data Sheet.

[25] Texas Instruments. *ISO733x-Q1, Triple Channel Digital Isolator*, 2015. Data Sheet.

[26] Texas Instruments. *SYS/BIOS (TI-RTOS Kernel) v6.45*, 2015. User's Guide.

[27] Texas Instruments. *TI Network Developer's Kit (NDK) v2.25*, 2016. User's Guide.

[28] Texas Instruments. *TI-RTOS 2.16*, 2016. User's Guide.

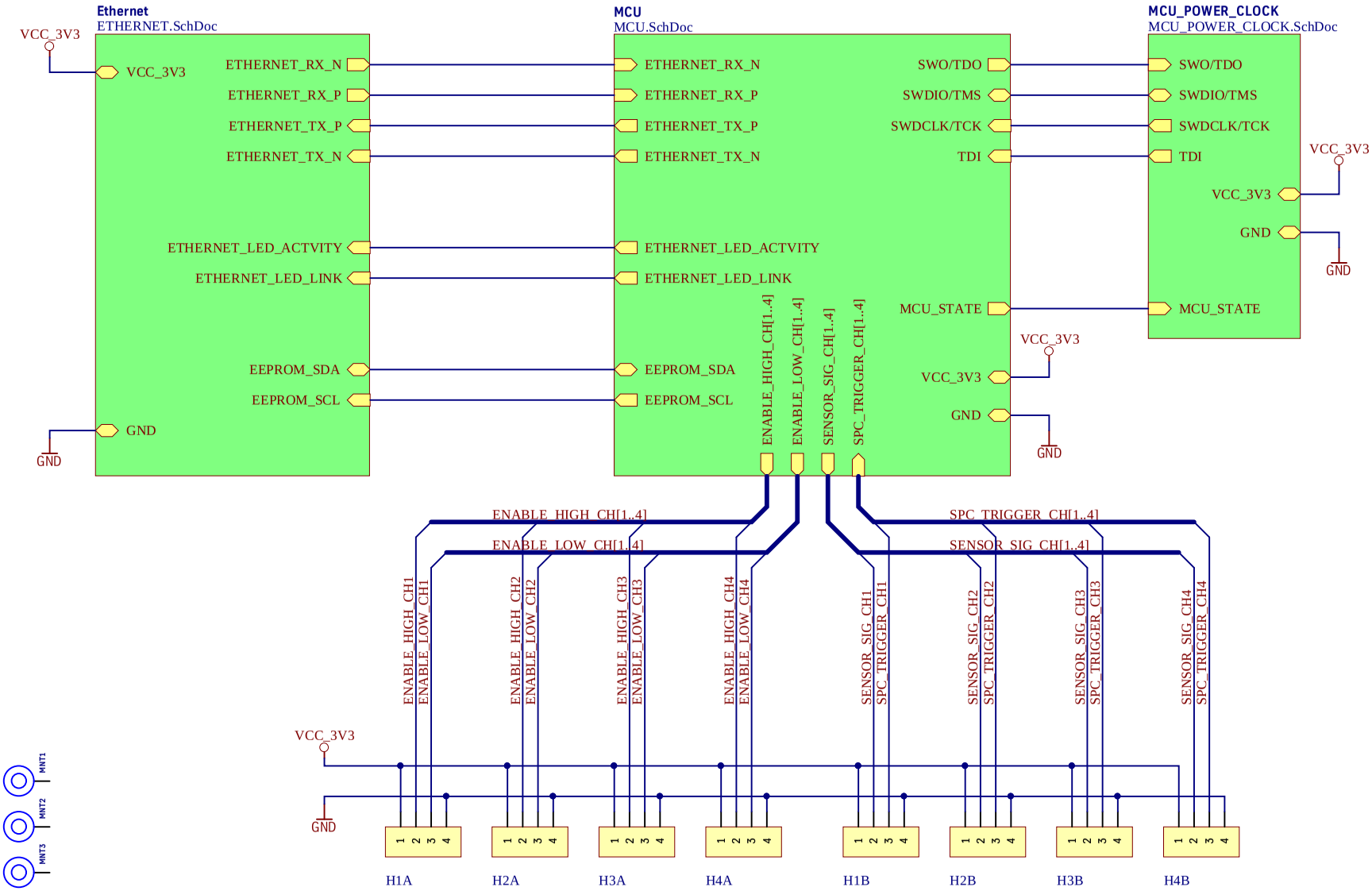
[29] Dr. Csubák Tibor. *Programozható irányítóberendezések és szenzorrendszerek*, 2014. Jegyzet.

[30] Lev Walkin. *Using the Open Source ASN.1 Compiler*, 2013.

[31] Tim White. *A Tutorial for the Digital SENT Interface*. Zentrum Mikroelektronik Dresden AG, 2014.

Függelék

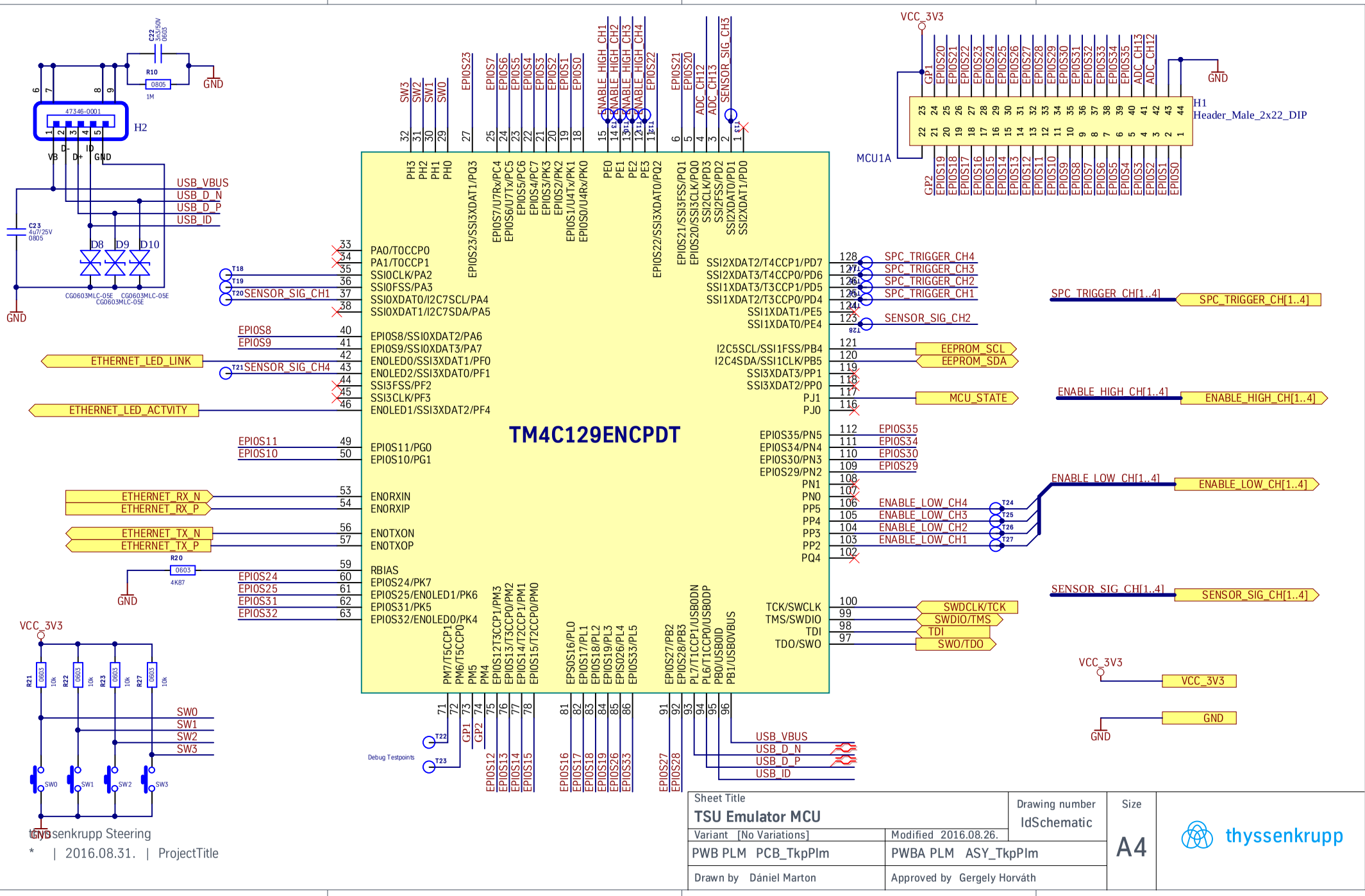
F.1. Kapcsolási rajz és BOM

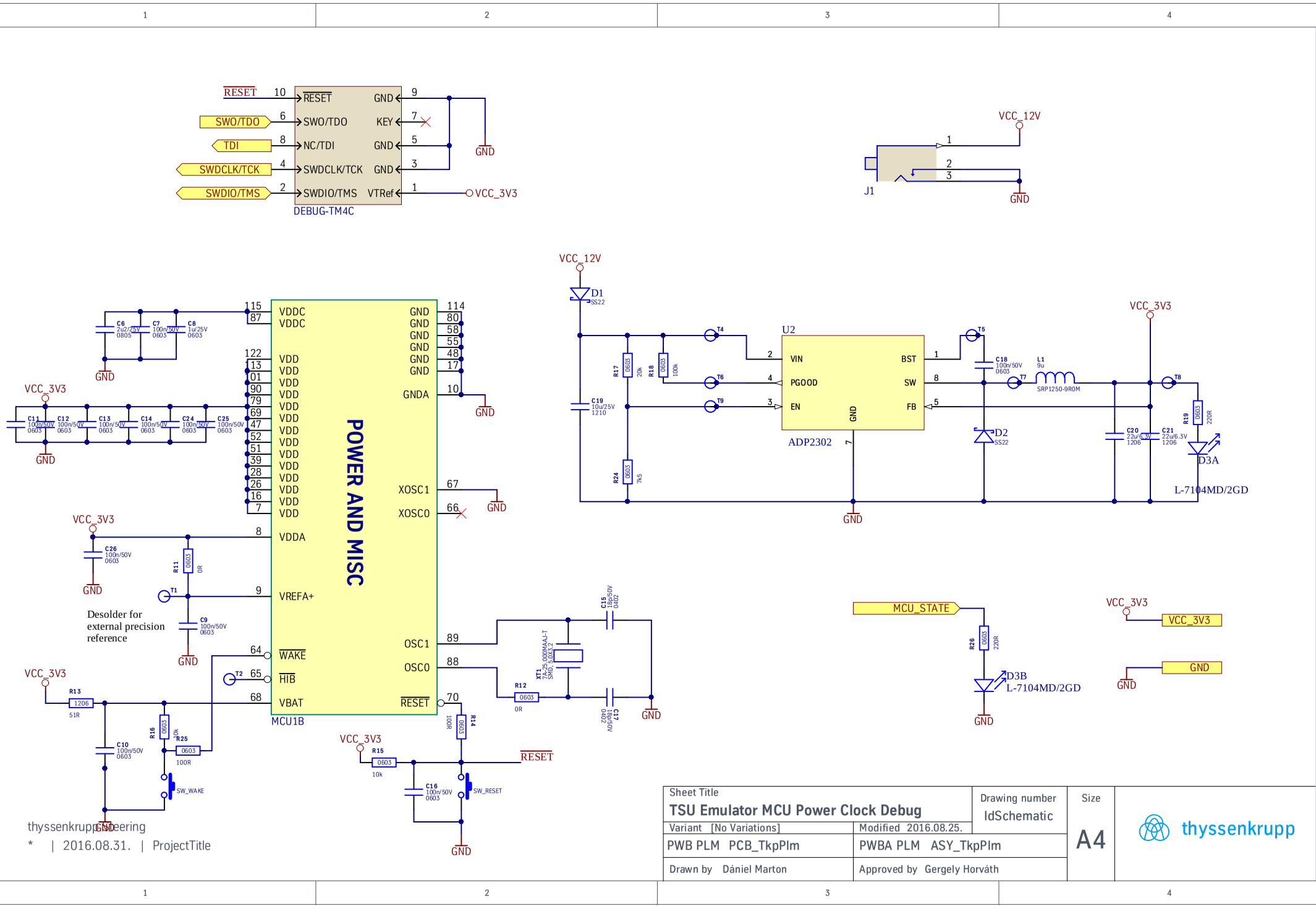


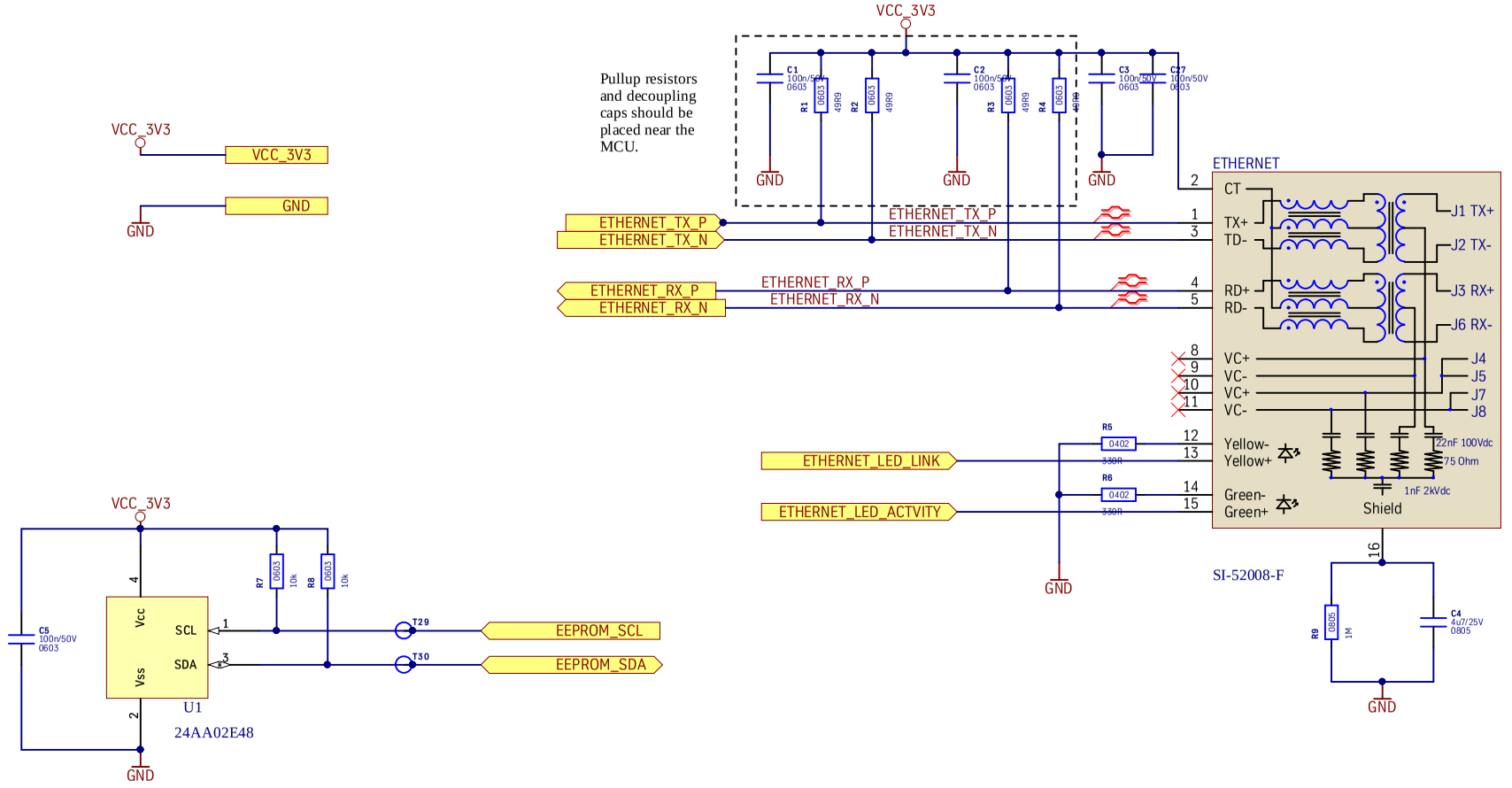
thyssenkrupp Steering
* | 2016.08.31. | ProjectTitle

Sheet Title: TSU Emulator Main Board		Drawing number: IdSchematic	Size: A4
Variant [No Variations]	Modified 2016.08.25.		
PWB PLM PCB_TkpPlm	PWBA PLM ASY_TkpPlm		
Drawn by Dániel Marton	Approved by Gergely Horváth		

thyssenkrupp


TM4C129ENCPDT
MCU1A
H1
Header_Male_2x22_DIP
H2
47346-0001
USB_VBUS
USB_D_N
USB_D_P
USB_ID
D8
D9
D10
CG0603MLC-05E
SENSOR_SIG_CH1
SENSOR_SIG_CH2
SENSOR_SIG_CH3
SENSOR_SIG_CH4
ETHERNET_LED_LINK
ETHERNET_LED_ACTVITY
ETHERNET_RX_N
ETHERNET_RX_P
ETHERNET_TX_N
ETHERNET_TX_P
SPC_TRIGGER_CH1
SPC_TRIGGER_CH2
SPC_TRIGGER_CH3
SPC_TRIGGER_CH4
SPC_TRIGGER_CH[1..4]
EEPROM_SCL
EEPROM_SDA
MCU_STATE
ENABLE_HIGH_CH1
ENABLE_HIGH_CH2
ENABLE_HIGH_CH3
ENABLE_HIGH_CH4
ENABLE_HIGH_CH[1..4]
ENABLE_LOW_CH1
ENABLE_LOW_CH2
ENABLE_LOW_CH3
ENABLE_LOW_CH4
ENABLE_LOW_CH[1..4]
SWDCLK/TCK
SWDIO/TMS
TDI
SWO/TDO
SENSOR_SIG_CH[1..4]
ADC_CH12
ADC_CH13
SW0
SW1
SW2
SW3
VCC_3V3
GND
Debug Testpoints
thyssenkrupp Steering
* | 2016.08.31. | ProjectTitle
Sheet Title
TSU Emulator MCU
Drawing number
IdSchematic
Size
A4
Variant [No Variations]
Modified 2016.08.26.
PWB PLM PCB_TkpPlm
PWBA PLM ASY_TkpPlm
Drawn by Dániel Marton
Approved by Gergely Horváth
thyssenkrupp


RESET
SWO/TDO
TDI
SWDCLK/TCK
SWDIO/TMS
RESET
SWO/TDO
NC/TDI
SWDCLK/TCK
SWDIO/TMS
GND
KEY
GND
GND
VTRef
DEBUG-TM4C
VCC_3V3
VCC_12V
J1
POWER AND MISC
VDDC
VDD
VDDA
VREFA+
WAKE
HIB
VBAT
GNDA
XOSC1
XOSC0
OSC1
OSC0
RESET
MCU1B
U2
ADP2302
VIN
PGOOD
EN
BST
SW
FB
D1
SS22
D2
SS22
D3A
L-7104MD/2GD
D3B
L-7104MD/2GD
MCU_STATE
Desolder for
external precision
reference
SW_WAKE
SW_RESET
thyssenkrupp Steering
* | 2016.08.31. | ProjectTitle
Sheet Title
TSU Emulator MCU Power Clock Debug
Variant [No Variations]
Modified 2016.08.25.
Drawing number
IdSchematic
Size
A4
PWB PLM PCB_TkpPlm
PWBA PLM ASY_TkpPlm
Drawn by Dániel Marton
Approved by Gergely Horváth
thyssenkrupp

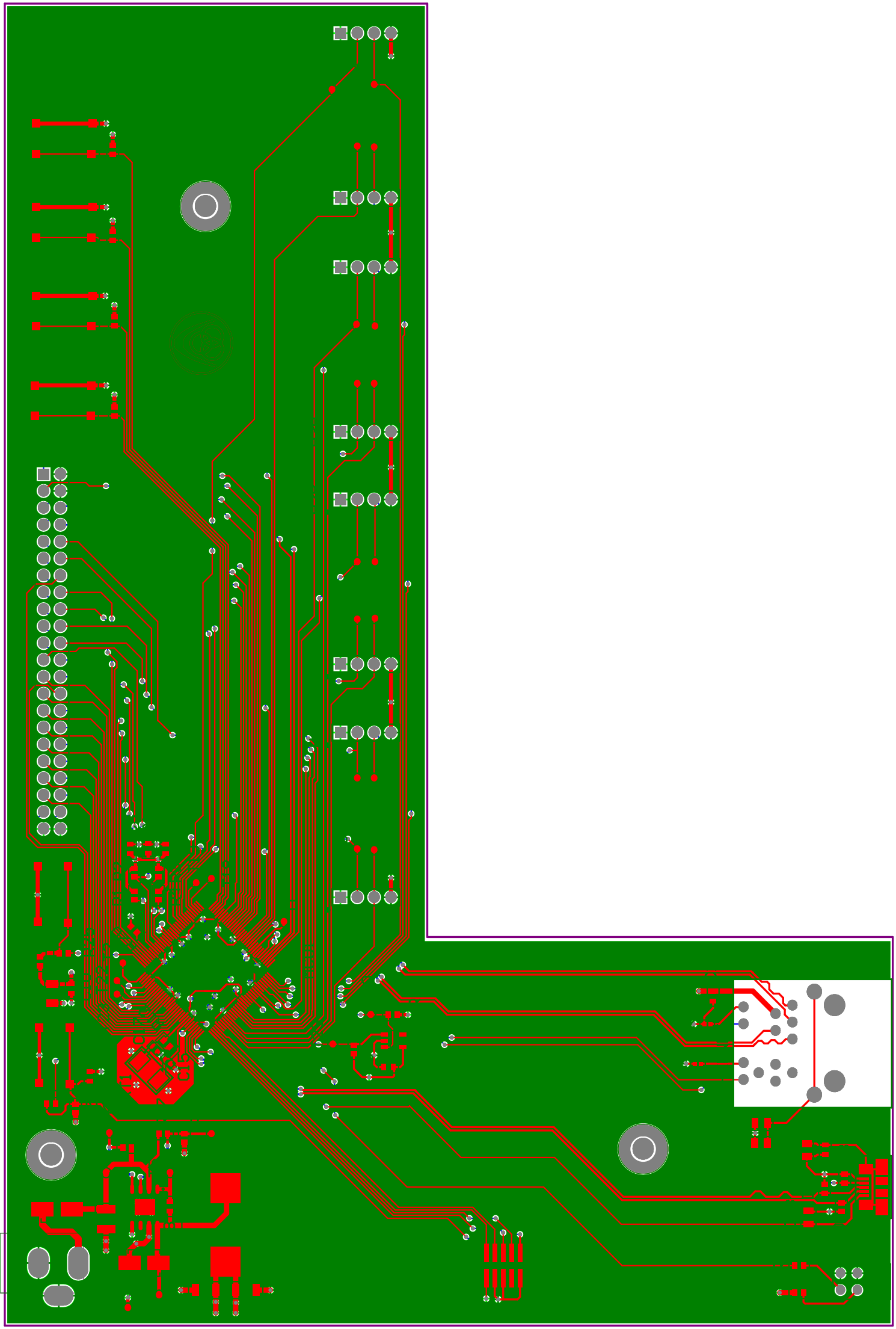


thyssenkrupp Steering

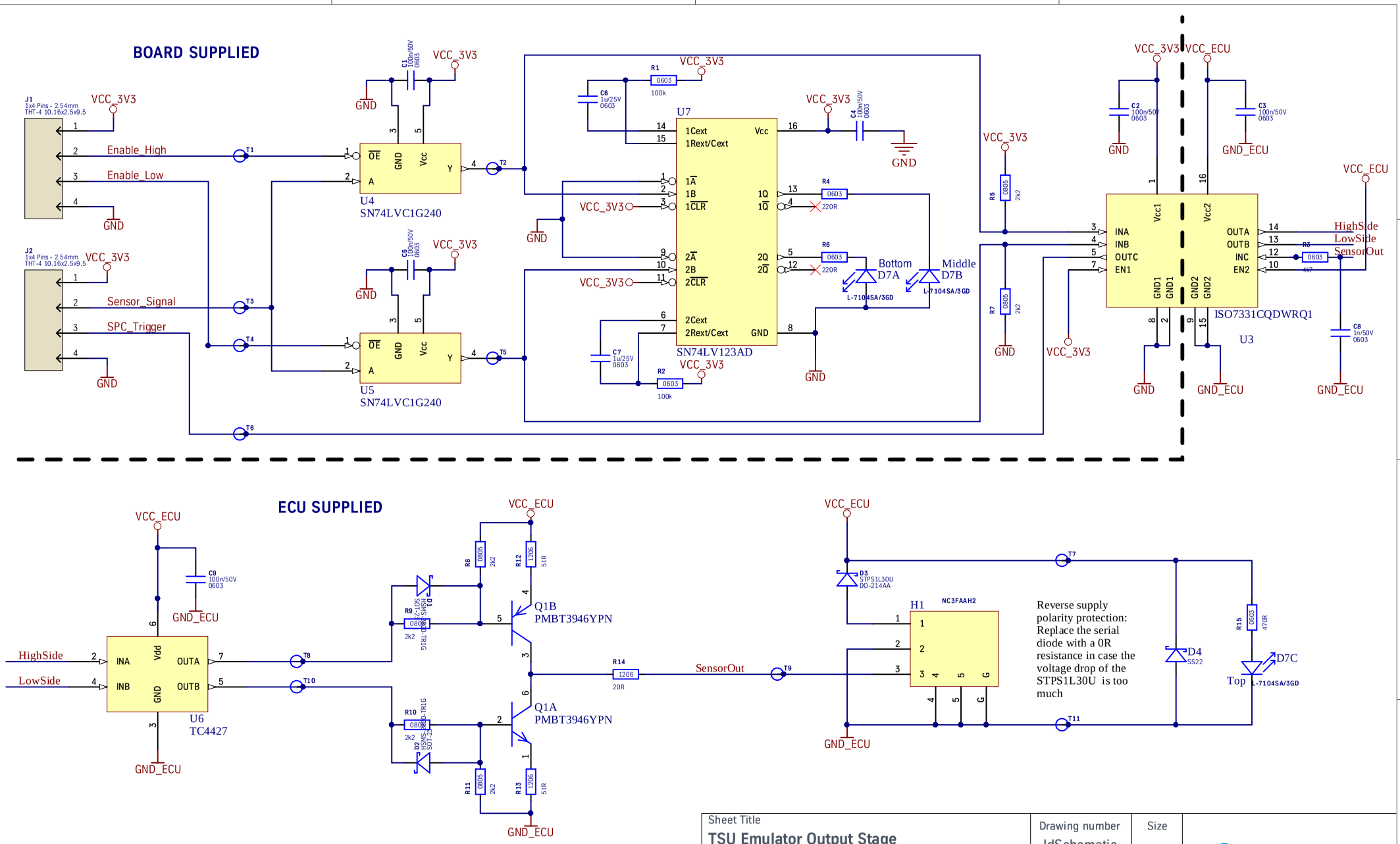
* | 2016.08.31. | eg. EPS controller, EPS power module, DV testbed adapter Card

Sheet Title: **TSU Emulator Ethernet**		Drawing number: IdSchematic	Size: A4
Variant [No Variations]	Modified 2016.08.25.		
PWB PLM PCB_TkpPlm	PWBA PLM ASY_TkpPlm		
Drawn by Dániel Marton	Approved by Gergely Horbáth		

thyssenkrupp



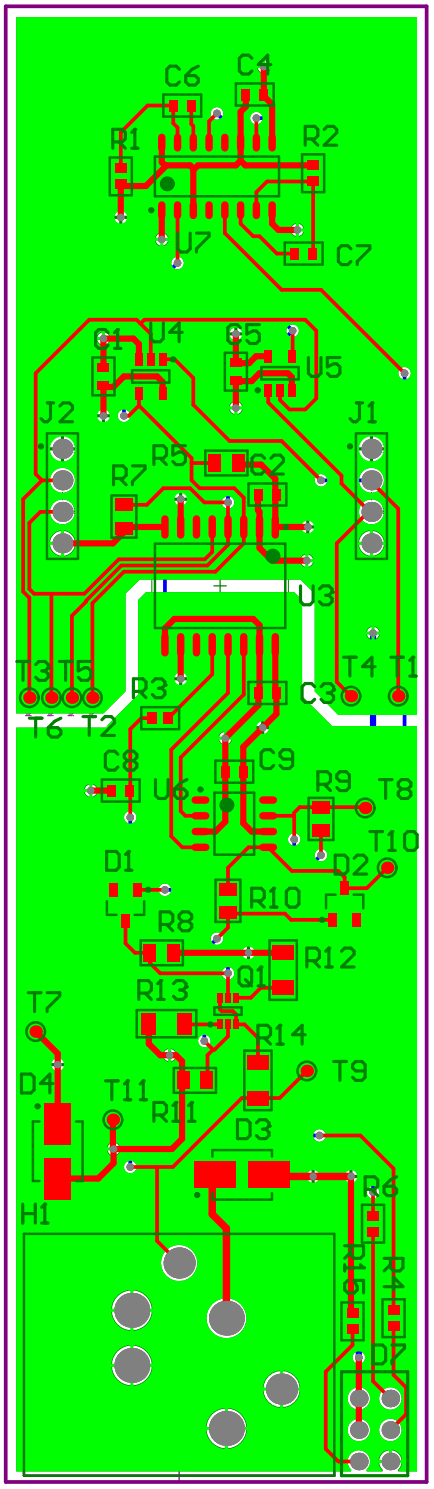
Comment	Description	Designator	Quantity	Supplier 1	Supplier Part Number 1
100n/50V	SMT capacitor	C1, C2, C3, C5, C7, C9, C10, C11, C12, C13, C14, C16, C18, C24, C25, C26, C27	17	Farnell	2346967
4u7/25V	Capacitor	C4, C23	2	Farnell	1735529
2u2/25V	SMT capacitor	C6	1	Farnell	2470436
1u/25V	SMT capacitor	C8	1	Farnell	2470425
18p/50V	SMT capacitor	C15, C17	2	Farnell	2210766
10u/25V	SMT capacitor	C19	1	Farnell	2470445
22u/6.3V	Capacitor	C20, C21	2	Farnell	1828808
3n3/50V	Capacitor	C22	1	Farnell	1828914
#NAME?	Diode Schottky	D1, D2	2	Farnell	1625255
#NAME?	T-1 (3mm) BI-LEVEL LED INDICATOR	D3	1	Farnell	2079990
#NAME?	CG0603MLC-05E	D8, D9, D10	3	Farnell	1838966
?		DEBUG-TM4C	1	Farnell	1099573
SI-52008-F	STEWART CONNECTOR SI-52008-F Modular Connector, RJ45, MagJack Series, Jack, 8 Contacts, 8 Ways, 1 Ports	ETHERNET	1	Farnell	1572195
Header_Male_2x22_DIP		H1	1	Farnell	2356149
Header_Female_1x4		H1A, H1B, H2A, H2B, H3A, H3B, H4A, H4B	8	Farnell	7991932
#NAME?	USB Connector, Micro USB Type B, USB 2.0, Receptacle, 5 Ways, Surface Mount, Right Angle	H2	1	Farnell	1568026
?	DC Power Connector, Jack, 4 A, 1.95 mm, Through Hole Mount	J1	1	Farnell	1737246
9u	Inductor	L1	1	Farnell	2329249
TM4C129ENCPDT	TEXAS INSTRUMENTS TM4C129ENCPDTI3 32 Bit Microcontroller, Tiva, ARM Cortex-M4F, 120 MHz, 1 MB, 256 KB, 128 Pins, TQFP	MCU1	1	Farnell	2444606
49R9	Resistor	R1, R2, R3, R4	4	Farnell	1469817
330R	SMT resistor	R5, R6	2	Farnell	1469709
10k	SMT resistor	R7, R8, R15, R16, R21, R22, R23, R27	8	Farnell	1469748
1M	Resistor	R9, R10	2	Farnell	1652946
0R	SMT resistor	R11, R12	2	Farnell	1469739
51R	SMT resistor	R13	1	Farnell	2307747
100R	SMT resistor	R14, R25	2	Farnell	1469752
20k	Resistor	R17	1	Farnell	1469774
100k	SMT resistor	R18	1	Farnell	1469649
220R	SMT resistor	R19, R26	2	Farnell	1652857
4K87	Resistor	R20	1	Farnell	1692529
7k5	Resistor	R24	1	Farnell	1469835
PushButton_MC32882	Tacetile Switch	SW0, SW1, SW2, SW3, SW_RESET, SW_WAKE	6	Farnell	2396053
24AA02E48	EEPROM, EUI-48 Node Identity, 2 Kbit, 2 BLK (128K x 8bit), 400 kHz, I2C, SOT-23, 5 Pins	U1	1	Farnell	1688855
ADP2302	Analog Devices, 2 A, 3-20 V, 700 kHz, Nonsynchronous Step-Down Regulator	U2	1	Farnell	2376914
7A-25.000MAAJ-T	Crystal - 2 Pin	XT1	1	Farnell	1841954



Sheet Title: TSU Emulator Output Stage		Drawing number: IdSchematic	Size: A4
Variant [No Variations]	Modified 2016.08.25.		
PWB PLM PCB_TkpPlm	PWBA PLM ASY_TkpPlm		
Drawn by Dániel Marton	Approved by Gergely Horváth		

thyssenkrupp Steering
* | 2016.08.31. |

thyssenkrupp


C6
C4
R1
R2
U7
C7
C1
U4
C5
U5
J2
J1
R5
C2
R7
U3
T3
T5
R3
C3
T4
T1
T6
T2
C8
C9
U6
R9
T8
T10
D1
D2
R10
R8
R12
Q1
R13
T7
R14
T9
D4
T11
R11
D3
R6
H1
R15
R4
D7

Comment	Description	Designator	Footprint	LibRef	Quantity
100n/50V	SMT capacitor	C1, C2, C3, C4, C5, C9	C_0603	C23	6
1u/25V	SMT capacitor	C6, C7	C_0603	C135	2
1n/50V	SMT capacitor	C8	C_0603	C169	1
HSMS-2820-TR1G	DiodeSchottky_HSMS-2820-TR1G	D1, D2	SOT23_ HSMS-2820-TR1G	DiodeSchottky_HSMS-2820-TR1G	2
STPS1L30U	Diode - Schottky	D3	SMB_DO-214AA	Diode - Schottky	1
SS22	Diode Schottky	D4	SMB_DO-214AA	SS22	1
L-7104SA/3GD	T-1 (3mm) TRI-LEVEL LED INDICATOR GREEN	D7	LED_INDICATOR _T1_TRIPLE	LED_INDICATOR_TRIPLE	1
NC3FAAH2	NC3FAAH2, Neutrik XLR connector	H1	NC3FAAH2	NC3FAAH2	1
1x4 Pins - 2.54mm	Connector	J1, J2	CON_THTS_1X4_2.54	M45	2
PMBT3946YPN		Q1	SOT363_PMBT3946YPN	PMBT3946YPN	1
100k	SMT resistor	R1, R2	R_0603	R26	2
4k7	SMT resistor	R3	R_0603	R36	1
220R	SMT resistor	R4, R6	R_0603	R32	2
2k2	SMT resistor	R5, R7, R8, R9, R10, R11	R_0805	R42	6
51R	SMT resistor	R12, R13	R_1206	R87	2
20R	Resistor	R14	R_1206	Resistor	1
470R	Resistor	R15	R_0603	Resistor	1
ISO7331-Q1	ISO7331-Q1 Robust EMC, Low-Power, Digital Isolator	U3	SOIC16_ISO7331-Q1	ISO7331-Q1	1
SN74LVC1G240	SN74LVC1G240, Single Buffer/Driver With 3-State Output	U4, U5	SOT23_SN74LVC1G240	SN74LVC1G240	2
TC4427	TC4427, 1.5A Dual High-Speed Power MOSFET Drivers	U6	SOIC8_TC4427	TC4427	1
SN74LV123AD	SNx4LV123A Dual Retrig	U7	SOIC16_SN74LV123AD	SN74LV123AD	1

F.2. ASN1 üzenetkatalógus

```
EMULATOR DEFINITIONS AUTOMATIC TAGS ::= BEGIN

max-dataunits INTEGER ::= 11
max-params INTEGER ::= 4
max-string INTEGER ::= 15
max-buffer INTEGER ::= 256

Channel ::= INTEGER (0..15)     --bitwise positions
Output ::= ENUMERATED {output-disable (0), output-enable (1)}
Mode ::= ENUMERATED {mode-sent (1), mode-spc (2)}
SaveLoad ::= ENUMERATED {save (1), load (2)}
SaveSlot ::= INTEGER (0..15)
Output-type ::= ENUMERATED {
        out-highz (1),
        out-pushpull (2),
        out-oc (3),
        out-oe (4)
}
Level ::= ENUMERATED {
        level-raw (1),
        level-pattern (2),
        level-high (3),
        enforce-size (1073741823) --enforce 32 bit size with 0x3FFFFFFF
}
FnName ::= IA5String (SIZE(max-string))

--Msg ::= SEQUENCE{
--msgtype INTEGER,
--msgdata CHOICE {
--      enableMsg EnableMsg,
--      valueMsg ValuesMsg,
--      hvalueMsg HValuesMsg,
--      configMsg ConfigMsg,
--      status StatusMsg
--}
--}

StatusMsg ::= [APPLICATION 0] IA5String (SIZE(32))

RawMsg ::= [APPLICATION 1] SEQUENCE {
        ch Channel,
        buff Buffer
}

Buffer ::= OCTET STRING (SIZE(0..max-buffer))

ValuesMsg ::= [APPLICATION 2] SEQUENCE{
        ch Channel,
        idx INTEGER (0..max-dataunits),
        val INTEGER
}

HValuesMsg ::= [APPLICATION 3] SEQUENCE{
        ch Channel,
        idx INTEGER (0..max-dataunits),
        val REAL
```

```
}

EnableMsg ::= [APPLICATION 4] SEQUENCE {
        ch Channel,
        enable BOOLEAN
}

Format ::= SEQUENCE {
        conv-fn-name FnName,
        default-values INTEGER (0..255)
}

Format-t ::= SEQUENCE (SIZE(0..max-dataunits)) OF Format

Params ::= SEQUENCE (SIZE(0..max-params)) OF REAL

HFormat ::= SEQUENCE {
        conv-fn-name FnName,
        params Params,
        association INTEGER (0..max-dataunits)
}

HFormat-t ::= SEQUENCE (SIZE(0..max-dataunits)) OF HFormat

Sensor-t ::= SEQUENCE{
        name FnName,
        tick-usec REAL,
        mode Mode,
        lowticks INTEGER (3..12),
        master-pulse-min REAL,
        master-pulse-max REAL,
        msg-delay INTEGER (0..255),
        output-type Output-type
}

ChannelCfg-t ::= SEQUENCE {
        sensor Sensor-t,
        output Output,
        level Level,
        format Format-t,
        hformat HFormat-t
}

ConfigMsg ::= [APPLICATION 5] SEQUENCE{
        ch Channel,
        cfg ChannelCfg-t
}

SaveLoadMsg ::= [APPLICATION 6] SEQUENCE{
        ch Channel,
        slot SaveSlot,
        sl SaveLoad
}

AutoChangeMsg ::= [APPLICATION 7] SEQUENCE{
        ch Channel,
        interval REAL,
        assoc INTEGER,
        fnname FnName,
        params Params
```

```
}

StoreTableMsg ::= [APPLICATION 8] SEQUENCE{
        start REAL,
        end REAL,
        tablesize INTEGER (1..128),
        table SEQUENCE SIZE (1..128) OF REAL
}

AutoChangeEnableMsg ::= [APPLICATION 9] SEQUENCE{
        enable BOOLEAN
}

END
```