TODO dokumentum formátum: margók betűtípus (doc: ~/Dokumentumok/szfk.docx)

TODO borító

Dankó Dávid

B35QC4

Cím: Uart buszrendszer és infrastruktúra.

A szakdolgozat célja:

Az EIB/KNX busz és eszköz modelljére hasonlító elosztott szenzor-aktor rendszer kialakítása a piacon könnyen és olcsón elérhető alkatrészekből.

1) Ehhez egy olyan busz rendszerre van szükségem, amivel:

- közepes távolságra (PCB méretein túl, de 1 km-en belül),

- nagyobb sebességgel (mikro vezérlő műveleti sebességéhez viszonyítva)

- több eszközt, elosztott módon (master-slave felépítését mellőzve)

- minél kevesebb vezeték felhasználásával

- minimális alkatrész igénnyel (de legalábbis egyszerű külső alkatrész függőséggel)

mikrovezérlőket lehet összekötni.

TODO-CONSULT: Ezek a kritériumok iteráció útján finomultak és válójában egy jelenleg ismert busz rendszer se elégíti ki az elvárásokat. Illetve csak a CAN volt egy időben, de úgy emlékeztem, hogy elég drága a periféria hozzá (de még így is a mikrovezérlő árához áll közelít. Ha ezt a külső eszközt mellőzzük, ugyanakkora összegből kétszer annyi busz eszközre lehet beruházni) ekkor gondolkodtam el, hogy egy minden mikrovezérlőben megtalálható perifériát kellene erre felhasználni és így bekerült az utolsó pont is a felsorolásba.

2) A mikrovezérlőket buszról és így távolról fel lehessen programozni, illetve a buszra csatlakozott eszközök valósítsanak meg egy olyan minimális funkcionalitást, amivel a csatlakoztatott eszközök felfedezhetőek (autodiscovery), kezelhetőek: mikrovezérlő újraindítás; alkalmazás program le-, feltöltése; eszköz típus, UUID lekérése; stb.

TODO-CONSULT: Egy fontos szempont az eszköz költsége. Hogy érdemes/szabad szakdolgozatban alkatrész árakat közölni és ez által fajlagos költségre hivatkozni? Problémák:

1. A kisker drágábban adja, mint a nagyker, a nagyker drágábban adja, mint a gyártó.

<https://hu.farnell.com/microchip/mcp2515-e-st/can-controller-spi-10ma-20tssop/dp/1605566>

<https://hu.farnell.com/microchip/mcp2551-i-sn/ic-can-transceiver-hi-speed-smd/dp/9758569>

1. Eltérőek az árak a rendelési mennyiséggel.
2. Előfordul, hogy összeszerelve olcsóbb egy modul:

<https://www.ebay.com/sch/i.html?_trksid=m570.l1313&_nkw=MCP2515+CAN+bus+module>

<https://www.ebay.com/sch/i.html?_from=R40&_trksid=m570.l1313&_nkw=atmega328+module>

(Ilyen atmega328p modulokon fejlesztek jelenleg is)

Irodalomkutatás: busz rendszerek.

Ahhoz hogy az ismert buszrendszereket össze lehessen hasonlítani, mérleget vonni és így a számunkra megfelelőt ki lehessen választani, egy közös séma mentén érdemes a busz tulajdonságait kutatni és tárgyalni. Erre megfelelő irányvonalat ad az OSI modell és annak egyes rétegein felmerülő tervezési kérdések.

A busz rendszerek kifejtésének szempontjai:

OSI-1) Fizikai réteg

* 1.1: Átviteli közeg, eszközönkénti vezeték vagy csatorna igény
* 1.2: Áthidalható távolság
* 1.3: Zavar kitettség
* 1.4: Átviteli sebesség

OSI-2) Adatkapcsolati réteg

* 2.1: Átvitel módja, moduláció vagy szinkronizáció
* 2.2: Tranzakció módja (ütközés kezelés és adatátvitel integritás biztosítása)
* 2.3: Átviteli egység keretezési többlet (overhead), átviteli egység nagysága

OSI-3) Hálózati réteg

* 3.1: Címzés
* 3.2: Egyéb busz képességek: broadcast, groups, TTL

OSI-+) Munkamenet, Megjelenítési, alkalmazás réteg

* 4: Egyéb magasabb szintű képességek (Transmission Control, RPC, stb.)

ALK) Alkalmazási terület.

TYP) tipikus alkalmazási példa

TODO-CONSULT: Kicsit el vagyok veszve, mely busz rendszereket kellene bemutatni és hogyan. Ott van pl.: az Ethernet amit régen busz rendszerbe illesztve használtak, de nem építettem még ilyet, így nem tudom hogy működhetne ilyen módon (a „ethernet line topology” és a pont-pont ethernet átvitel bemutatásán kívül más megoldást nem találtam). Illetve ott az UART ami nem busz rendszer, de erre épül a szakdolgozat.

EIB/KNX) (European Installation Bus)/(KNX a Konnex szójáték rövidítése)

EN 50090

ISO/IEC 14543

A szakdolgozat célja az EIB/KNX rendszerre hasonlító busz és eszközrendszer, infrastruktúra megvalósítása, így a szakdolgozatban lényeges ennek a rendszernek részletesebb ismertetése.

Az EIB egy olyan elosztott épületautomatizálási rendszer, amelyben a szenzorok és aktorok egy közös busz hálózatra vannak kötve. Az irányítás az felprogramozott eszközök egymás közötti adatcseréjével jön létre, ebből kifolyólag a rendszernek nincs központi egysége, így egyetlen eszköz funkcionális meghibásodás nem okozza a teljes automatizálási rendszer működésképtelenségét.

TODO 4 közeget támogat (TP1, PL110, RF, Ethernet)

KNX/ PL110)

1.1) Csupán egy vezetékpárt használ, ezen keresztül történik az eszközök tápellátása (SELV) és az adat forgalmazása.

1.2) Egy szegmens legfeljebb 1 km távolságot tud áthidalni, ezt ismétlőkkel 4 km-ig lehet kibővíteni.

2.1) 110Khz-es S-FSK

2.2) CSMA

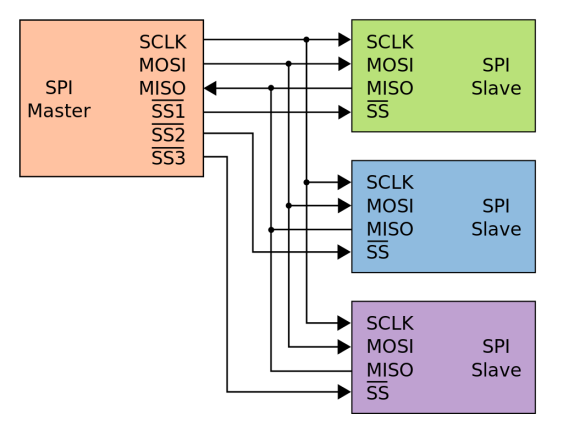
TODO Címzés, csoportok, alkalmazás.

* http://www.sti.uniurb.it/romanell/Domotica\_e\_Edifici\_Intelligenti/110504-Lez10a-

KNX-Architecture%20v3.0.pdf

* https://en.wikipedia.org/wiki/KNX\_(standard)

SPI) (Serial Periferial Interface)



https://en.wikipedia.org/wiki/Serial\_Peripheral\_Interface

Beágyazott rendszerek, egyszerű perifériákkal, PCB-n belüli összekötésére tervezett, full duplex szinkron soros, master-slave architektúrájú egy masteres busz rendszer. Különlegessége, hogy csupán D tárolók felhasználásával elkészíthető az adó és fogadó oldal további bonyolultabb hardware és software komponensek nélkül

* 1. 3 vezeték (CLK, MISO, MOSI). Továbbá eszközönként egy a kiválasztáshoz CS (chip select)
  2. A PCB méretein belül, legfeljebb 60 cm.
  3. Tervezési szempont, hogy a távolságok és a PCB vezető sávjai olyan rövidek legyenek, hogy az ne gyűjtsön össze annyi zajt, ami megzavarja, elrontja az átvitelt.
  4. A master órajelétől és az eszközök sebességétől függően akár több Mbites sebességgel is üzemeltethető.

2.1) Master-slave felépítés: az órajelet és az eszköz kiválasztás a master kezdeményezi.

2.2) Master biztosítja a szinkron órajelet egy külön vezetéken és indítja a tranzakciót, így ütközés nem fordulhat elő.

2.3) Nincs keretezési többlet. A ”csomag” végét a kiválasztás (CS) visszavonása jelzi.

Az átviteli egységre hosszára nincs megkötés.

ALK)

* SD kártya,
* CAN, Ethernet, stb. interface chip csatlakoztatása.
* GPIO Port bővítés küldő SIPO/PISO regiszterrel.
* Az eszközök általában támogatják a lánckapcsolást (daisy chain), így egy tranzakcióban több eszköz is írható.

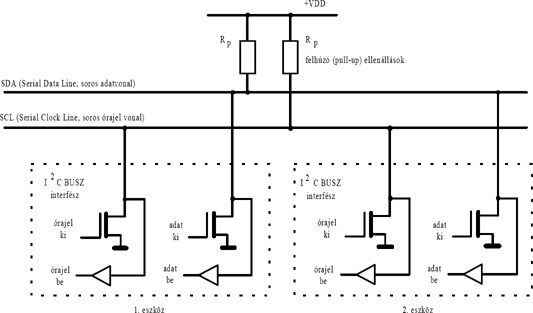
Érdekességek:

* Ha az eszköz csak ki vagy bemeneti periféria, az egyik (sorrendben MISO vagy MOSI) vezeték elhagyható.
* a master bit-banging megvalósítása egyszerű
* dupla D tároló sorral GPIO port bővítésre használható

TYP) TODO ábra a régi kiselőadásból

https://en.wikipedia.org/wiki/Serial\_Peripheral\_Interface

I2C) Inter Integrated Circuit



https://www.hobbielektronika.hu/cikkek/kommunikacio\_alapjai\_-\_soros\_adatatvitel.html?pg=5

Több mikrovezérlők és egyszerű eszközök, PCB közötti, half-duplex, szinkron soros, több master-es, csomag kapcsolt összeköttetés lehetővé tevő busz rendszer.

A Philips semiconductors (Ma NXP) alkotta meg 1982-ben.

* 1. Két vezetéket vagy vezetékpárt használ. Mindkét vonal egyik vége pull-up ellenállással tápfeszültségre fel van kötve. Az egy vezeték az órajel (SCL) a másik az adat (SCL).
  2. Busz feszültség, sebesség, illetve felhúzó ellenállástól függően 3-tól 250 méterig [1].

Jellemző hossza: 1,5 méter alatt.

* 1. A zaj kitettség rövid vezetékekkel kisebb értékű (ohm-ban kifejezve) felhúzó ellenállások vagy sodrott érpár használatával csökkenthető.
  2. 100 Kbits/s (eredeti ’82 design)

400 Kbit/s Fm, Fast mode ’92-től.

1 Mbit/s Fm+, Fast mode plus 2007-től

3,4 Mbit/s Hs, High speed ’98-tól

(Egyirányú buszként) 5 Mbit/s UFm, Ultra fast mode, push-pull logikáva (azaz felhúzó ellenállás nélkül)

2.1) Az átvitel szinkron, a master az SCL vonallal szinkronizálja az átvitt adatot.

Az ütközés figyelésére is ezt a vonalat használják.

2.2) Bármelyik mikrokontroller lehet master a buszon, azaz bármelyik egység kezdeményezhet adást.

2.3) Nincs keretezési többlet, viszont a bitek átviteléhez egy külön vezetéket (SCL) használ fel. Az átviteli egységnek nincs felső határa.

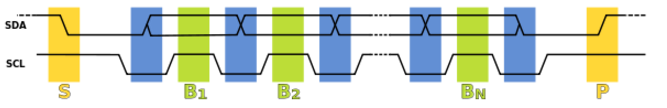
3.1) 7 bites mód (ebből 16 cím foglalt 0000|1111-XXX)

10 bites mód (ebből 128 cím foglalt 0000|1111-XXXXXX)

3.2) A csomagok olvasás és írás műveleteket tartalmaznak (Címkeret utáni első bit).

Kézfogásos visszaigazolás: A címzett slave (ha jelen van a buszon), a címkeret vétele után le kell húznia az SDA vonalat, ezzel jelezve a master-nek hogy figyeli a vételt.

TODO

 [1] [https://www.nxp.com/docs/en/data-sheet/P82B96.pdf (14](https://www.nxp.com/docs/en/data-sheet/P82B96.pdf%20(14) oldal)

1-WIRE

ETHERNET) a random visszalépési idő miatt (ütközés)

CAN

UART

TODO A konklúzió:

* SPI nem erre lett tervezve.
* Az I2C jó lenne, a legtöbb mikrovezérlőbe be van építve, de két vezeték(párt)t használ a kommunikációra.
* Az 1-Wire-nek 16,3 kbyte/s a maximális sebessége és master-slave architektúrára épül, ezért nem támogatja a több masteres hálózatot.
* Az Ethernet estén nem találtam olyan megoldást amivel egy vezeték párra lehetne redukálni az összeköttetést
* A CAN… jobban utána nézve nem is drága az inteface hozzá. Rendeltem 3 db csatolót, ha marad idő a szakdolgozat elkészítéséhez szükséges feladatok után, akkor lehet megpróbálkozom az UART helyett a CAN csatolással. Feljebb említettem hogy a mikorvezérlő árához közelítő eszöz, ennek elhagyásával kb. fele annyi a busz költsége.
* UART: nem lehet igazán busznak nevezni, pont-pont összekötetésre lett kitalálva, de módosítással buszrendszer készítésére fel lehet használni.

Az UartBus: UART-ra épül, ami sok eszközben megtalálható. Több feszültség szintre és elvileg több közegre illeszthető (tervbe van a rádiós átvitel tesztelése is). Az ütközés kezelését, címzést, softwares úton kerül megvalósításra.

Mivel a címzés is software-es megvalósítású, kitaláltam egy változó byte hosszúságú címzési módszert (TODO kifejtés)

Megjegyzés: Találtam hasonló szakdolgozatot, amelyek témája az 1 vezetékes multi master busz rendszert kialakítása:

1. <https://www.idosi.org/mejsr/mejsr23(ssps)15/64.pdf>

Itt az 1-Wire mintájára bit-banging technikával (mint ahogy a 1-Wire könyvtárak teszik) írják (AVR open-drain lábával) és olvassák a buszt. A bit-banging-el az a probléma hogy nagyobb sebességen nagyobb az írás/olvasás hiba aránya (SoftUart-tal is ezt tapasztaltam). Ezért használom a beépített UART egységet, így nagyobb sebességű is alacsony az átviteli hiba.

Fontos részlet: Két RPC réteg van a rendszerben és valóban mindkettő távoli eljáráshívás. Az egyik a CLI alkalmazások és az `ub rpcServer …` program közötti csatorna, amivel a PC-n futó alkalmazás dinamikusan be tud csatlakozni és csomagot feladni illetve a buszon folyó kommunikációba bele tud hallgatni. Az `ub rpcServer …` futó program a ttyUSB eszközön keresztül csatlakozik a busz csatolóhoz. Az atmega2560 busz csatoló egyik UART portja (Serial objektum az arduino könyvtárában) a PC egyik ttyUSB portjához csatlakozik. Ezen csomaghatár keretezési módszert használ mind az adáshoz, mint a vételhez. A másik UART portja (Serial3 arduino objektum) amire az uartbus rendszer van csatlakoztatva. A mega2560 TX3 és RX3 lába illesztőn keresztül a busz rendszerhez van csatolva. Így valósul meg az útvonal a CLI alkalmazás és a busz között.

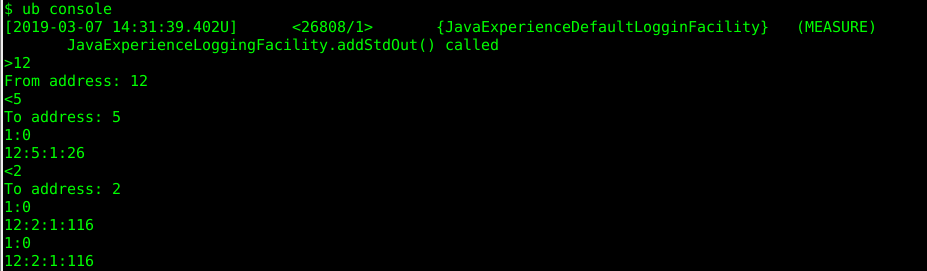
A másik RPC réteg a mikrovezérlőkben van megvalósítva. Végülis távoli eljáráshívással van megvalósítva az eszközök pingelés, az újraindítás meghívása, kódfeltöltés fázisának kezelése.

Egy hálózati csomag végülis csak számok sorozata, aminek a kód lefuttatása ad értelmet.

A konzollal demonstrálva így lehet rpc hívás végrahajtani:

A buszra két, az 2-es és az 5-ös eszköz van felcsatlakoztatva.

`ub console` //hacsak nem adunk meg a –h kapcsolóval hostot és a –p kapcsolóval porto

a 127.0.0.1:2112 címre csatlakozik. (Ezt SSH-val áttuneleztem a szerverre amin a csatoló van)

A konzol arra használható, hogy „csomagot szúrjunk be a hálózatra”, azaz mi adhatjuk meg a forrás és cél címet is, így debugolásra és tesztelésre is használható.

Minden páros sor mi írunk be, a páratlanok a válasz sorok.

„> 12” a feladó címének beállítása

„<5” a cél cím beállítása.

Az „1:0” beírása azt jelenti hogy „hívjuk meg az 1-es névtér (busz funkciók) 0. eljárását(ping)” ebből ez a csomag lesz (ami a konzolon nem látható): 5:12:1:0:$crc8

Erre a válasz: „12:5:1:26” csomag a 12-es eszköznek, az 5-től, 1-es névtérbeli hívásra: üres válasz (a 26 a csomag crc8 összege)

Ez a felépítése az SNMP MIB-ekre hasonlít.

Ezek funkció mutatót tömbökbe, „névterekbe (NS)” vannak összegyűjtve kompakt módon, hogy nem foglaljon sok kód területet és könnyen vagy akár dinamikusan is össze lehessen állítani.

Projekt teljesítés lista:

- Mikrovezérlő egy vezetékes illesztése minimális alkatrésszel (sikerült, ez volt a feszültség osztós vevő)

- Első adat forgalmazás két eszköz között (küldött csomag visszaküldése) 115,2 Kbps-en 1m-en belül (sikerült, eredetileg PC és mikrovezérlő között az időzítések figyelmen kívül hagyásával)

- idő alapú csomag keret szétválasztás (sikerült, PC-UC között. Ekkor 115,2 Kbps-en 1 ms volt az adás vége hogy a PC szét tudja választani a csomagokat, ekkor íródott a könyvtár első verziója, amibe függvény mutatókkal lehet adó-vevő funkciókat beregisztrálni és blokkolásos üzemmódban adni-venni)

- fairwait bevezetése (sikerült: mivel a küldő hamarabb „értesül” az adás végéről és ha újra adni kezd, akkor elárasztja a buszt. Ezért adás után extra várakozás van beiktatva, aminek a letelte előtt persze vételezhetünk a buszról)

- crc8 automatikus hozzáadása a csomag végéhez, továbbá adó és vevő bufferek bevezetése. (sikerült)

- arduino könyvtár illesztés, ami a Serial, Serial{1,2,3}-hoz csatlakozva egyszerű használatot biztosít (sikeült)

- megszakításos vétel (sikerült: a megszakítás kezelő helyezi el a vevő bufferbe az adatot)

- PC illesztése a busz rendszerhez (sikerült: atmega168 és SoftUart-tal elbukott a nagy csomag hiba arány miatt, de később atmega2560-nal aminek több UART-ja van azzal sikerült.)

- ub\_bootloader: Eszközök címzése és cím szerinti szelektálás: csak a nekünk érkező vagy broadcast csomagokat dolgozunk fel (sikerült, int16\_t to, from-mal kezdőik a csomag eleje)

- RPC hívási séma kidolgozása és megvalósítása (sikerült, sorszámozott névterekként. 1-31-ig a bootloader kezeli a csomagokat, 32 és felette az alkalmazásnak kerül a csomag kiszállításra. TODO ezt részletesebben leírni)

- Első sikeres RPC hívás PC-ről a busz csatoló segítségével a buszon lévő eszközökön (sikerült)

- Rpc szerver ami a ttyUSB, a konzol(ok) és programok közötti kapcsolatot teremti meg.

- CLI Konzol aminek segítségével megfigyelhetjük a buszon forgalmazott csomagokat és küldhetünk is. (sikerült)

- UART felhasználásával legalább 3 egység összekötése, oda-vissza kommunikáció létesítése (sikerült, 5 méteren, a feszültség osztós csatolással 115,2 Kbps-en nem, de 19,2 Kbps-en igen)

- Első buszon keresztüli eszköz felprogramozása (8FEF84B985B1809585B90895 //DDRB=0xff;PORTB ^=0xff;return;)

- változó hosszúságú címzés (sikerült)

(TODO kifejtés) a projekt software réteg lebontása:

* uartbus: eszköz független busz könyvtár (adó és vevő funkciókat függvény mutatókkal lehet beregisztrálni, így több buszhoz is lehet egyidejűleg csatlakozni, pl a mega2560-nak 4 UART interface is van)
* ub\_arduino: kényelmi illesztés arduino Serial (pontosabban Stream) objektumhoz
* librpc: pár funkció és struktúra, a kérésként érkezett byte tömb RPC hívásként való kezelésére, függvény láncon való dispatch-elésére és válasz összeépítésére.
* ub\_bootloader: összetett host alkalmazás, amibe kódot tölthetünk fel a buszon keresztül, ping csomagokra válaszol, UUID azonosítót szolgáltat kérésre, stb.

Az itt hivatkozott sebességek:

* kis sebesség: b <= 19\_200
* közepes sebesség: 19\_200 < b <= 250\_000
* nagy sebesség: 250\_000 < b

TODO közegek:

1. elvileg bármilyen olyan közegben történhet átvitel ahol az adásként értelmezett jelek „vagy” kapcsolatban hozhatóak: A vagy B húzza le a busz vezeték feszültségét. A vagy B sugároz ki fényt, rádió jelet. Így ki lehetne próbálni 433 Mhz adó-vevő pár illesztővel vagy infrán.
2. huzalon 5V-on fél illesztéssel
3. huzalon nagyobb feszültségekre (12V, 24V)
   1. kis sebességű illesztés (feszültség osztóval)
   2. közepese sebességű egyszerű illesztés (a BS170 +-20V Ugs feszültséget is kibír, vajon illik-e közvetlen bekötni)
   3. közepes sebességű illesztés komparátorral
   4. lehet nem lesz a szakdolgozat része, kísérlet: nagy sebességű (inverz schmitt trigger létezik-e?)

Ütemterv:

--------------------

doc: KNX kifejtése: TP1/PL110/RF/Ethernet/címzések/csomag formátum/ütközés kezelése/technológia ára

---------------------

konzultáció: Ha van lehetőség a TDK beadására, melyik részt és milyen részletességig dolgozzam ki?

---------------------

doc: UARTBUS részeinek kifejtése: a könyvtárak tagoltak, dinamikusan összeköthetőek extern és függvénymutatók segítségével. Menekülő útvonal: ha a megvalósítás valamely pontján megoldhatatlan vagy hanyagul megoldható problémába ütköznék és a fejlesztésével fel kellene hagyni, az infrastruktúrát fel lehet használni. Az UARTBUS részt le lehetne cseréli pl.: CAN-re. Az RPC busz és fedélzeti műveletek, távoli programozás változatlanul elérhetőek lennének, csak az alsó két OSI réteg kerülne lecserélésre.

Fejlesztés menete, eddigi eredmények:

A projektet GIT-en fejlesztem, ebbe találhatóak a C/CPP könyvtárak, utility alkalmazások, avr fordító és avrdude feltöltő scriptek, debian-ra a függőségeket telepítő script-ek:

`git clone http://git.javaexperience.eu/javaexperience/uartbus/.git/`

A javaba írt ennek kezelésére alkalmas „infrastruktúra rendszert” is ebből le lehet fordítani

`./scripts/assemble\_uartbus.sh`

Csináltam mega2506-hoz UARTBUS <=> PC csatlakozót, a PC nem alkalmas az idő szerinti byte csoportosításra így csináltam hozzá egy primitív csomag keretezőt (Arduino-hoz és PC-hez is), így amega át tudja adni a csomagokat a PC-nek blokkokba illetve fordítva. Ehhez van egy automata fordító és avrdude feltöltő kód:

. /source/uc/utils/apps/arduino/bus\_connector/target\_atmega2560\_make\_and\_upload.sh

Ez a Serial és Serial3 között végez összeköttetést. Az a ttyUSB vonal amit az avrdude használ felprogramozásra azt követően busz közvetítésre használható.

Ha lefordítottuk a java UARTBUS könyvtárat (előbbi script parancs) akkor a ./WD könyvtárban megjelenik az uartbus.jar, amiben minden busz kezelő program „utility” megtalálható. Ez egy CLI program. A „./scripts/add\_command\_ub.sh” futtatási kimente ad egy instrukciót hogyan tehetjük elérhetővé az `ub` parancsot a jelenlegi terminálunkban.

Ha ez után kiadjuk az `ub` parancsot a program kiírja hogy milyen alprogramok vannak benne.

PC oldali infrastruktúra:

Az elv az, hogy egy folyamatosan futó alkalmazás felcsatlakozik a busz illesztőn keresztül (jelenleg atmega2560) a buszra és ez az alkalmazás figyel egy TCP porton. Ez egy RPC felületet biztosít, amire felcsatlakozhatnak külső programok (egy időben több is) és csomagot küldhet a hálózatra, illetve olvashat onnan.

TODO később megvalósítani: Ennek a programnak érdemes naplóznia a csomagokat a hálózaton későbbi adatgyűjtés céljából.

Ehhez az RPC felülethez csatlakozik hozzá a többi program, ami különböző feladatokat valósít meg. pl.: alacsony szintű konzol (console), ismétlődő pingelés (ping), eszköz felprogramozása intel hex fájlból (TODO upload), autodiscover (TODO), stb. Az RPC protokoll primitív JSON soronkénti átvitellel működik, így könnyen fejleszthető hozzá alkalmazás már programnyelven is.

#így indíthatunk el egy rpc szervert:

$ ub rpcServer -b 19200 -s /dev/ttyUSB0

alapértelmezetten a 2112 porton nyitja meg az RPC szervert hacsak a –p kapcsolóval mást nem adunk meg.

Mivel innentől TCP-n keresztül tudunk becsatlakozni, ezt portot tunnelezni is lehet. Így akár egy helyi gépről egy távoli szerverre - ami egy ilyen busz rendszerhez van csatolva - fel tudunk csatalakozni és távolról parancsokat lehet adni, eszközöket felprogramozni.

TODO kifejtés: RPC protokoll (byte tömbön végigcsúszó értelmező) és dispatcher funkciók, funkció láncok.

TODO kifejtés: uartbus host és az alkalmazás összelinkelése, azaz hogyan használjuk a kliens alkalmazás a gazda program függvényeit.

TODO kifejtés: a kliens alkalmazás egy app\_wrapper mikorkönyvtár fordításával (ami az arduino szerű setup() és loop() sémát biztosítja) együtt, módosított kezdő című területre kerül feltöltésre.

TODO kifejtés: általánosított kódfeltöltési folyamat (remélhetőleg más mikrovezérlőre is alkalmazható a program feltöltési hívás séma)

TODO kifejtés: RPC névterek, ezekből van olyan kérés, ami az alkalmazás kód felé továbbításra kerül, így a buszon maga az eszköz és a feltöltött egyéni alkalmazás funkciói is elérhetőek.

----------

Fejlesztési mérföldkövek: (TODO-k)

----

Az ütközés kezelése még nem működik megfelelően, de tesztelési módszer már van rá:

ub ping –f 64 –t 0 –i 250

#64-es címről, 0 cím (broadcast) pingelése 250 ms küldési időközzel.

Mindkét mikrovezérlő egyszerre próbál válaszolni kb 100ns belüli eltéréssel, így következetesen előállítható az ütközés.

Itt jutott eszembe: az adás kezdetét csak az első byte megérkezésénél észleljük (UART0 ISR-ben) viszont ebben a helyzetben már ütközik a két adás. Ha rászánunk egy mikrovezérlőnek egy olyan bemenő lábát, amelyen lefutó élt tudunk detektálni, akkor elkaphatjuk az adás kezdetét és megakadályozhatjuk az adást a második mikorvezérlőn, ezzel elkerülve az ütközést. Továbbá azt sejtem, hogy az csomag vége várakozás után próbálnak mindketten egyből adni. Azaz az RPC logika lefutott, a csomag el lett helyezve a kimenő „soron”, csak csomag végét jelző időt kell kivárni. Ez esetben a lefutó él detektálás mellet a hatékonyságot növelheti még az adás kezdésének (vagy csomag vége plusz várakozásnak) a randomizációja.

----

Első nem AVR busz eszköz: A 8051 és/vagy STM32F103C8T6 (szállítás alatt) portolása és csatlakoztatása a busz rendszerbe. Először elég, ha meg lehet pingelni. Az újraindítás, eszköz azonosító lekérést, program feltöltést ekkor kellesz outsource-olni az uart\_bootloader kódbázisából.

----

Ez említett bus illesztők felhasználásával meg kellene próbálni a 115\_200 baud/s átviteli sebességet.

----

Ki kellene próbálni a busz rendszert legalább 200 m TP huzalon. (19,2/115,2 kbaud/s)

----

Arduino környezetbe illesztett kliens kód fordítás és feltöltés akár külső könyvtárak felhasználásával.

Pl.: IRReceiver-t, LCDDisaply-t használó programok feltöltése, ami az infra távirányító jelét kiküldi a buszra, illetve egy bizonyos csatornára írt szöveget kiír a kijelzőre.

----

Fejlemények:

Az arduino könyvtár nem bírja a 115 Kbaud feletti sebességet, túl nagy a byte-ok közötti szünet. Az időablak még növelhető, vagy lehet hogy hagyni kellene az arduino könyvtár csatolást… … vagy korlátozni a sebességét (felhasználói tájékoztatása)