ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Kucsera Gergő**

2021 Hallgató törzskönyvi száma: **T/006274/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kucsera Gergő**
Törzskönyvi száma: T/006274/FI12904/N
Neptun kódja: HVKTS7

A dolgozat címe:

Szabotázs detektáló modul hardveres tesztelési folyamatának automatizálása
Automation of the hardware testing process of a tamper detection module

Intézményi konzulens: Lovas István
Külső konzulens: Stancz Krisztián, i4p informatikai kft.

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

A feladat

Tervezzen és valósítson meg egy hardveres tesztelést is lehetővé tevő, automatizációs rendszert az i4p informatikai kft. szabotázs detektáló moduljához. A cél a valós idejű feldolgozás, az eredeti architektúrán futó üzleti logika, és a szenzorok helyes működésének ellenőrzése. Szemléltetésnek olyan tesztesetet válasszon, melyben az előbb említett tényezők mindegyike szerepel.
Vizsgálja meg a tesztelendő eszközzel való interakciós lehetőségeket; azok hardveres és szoftveres igényeit. Válassza ki a megfelelő számítógépes környezetet, majd állítson be rajta egy folyamatos integrációs eszközt. Lehetőség szerint biztosítani kell, hogy a forráskódok lefordítása is automatizált módon történjen. Ügyeljen arra, hogy a környezet számára az internet-hozzáférés nem megengedett; a magas szintű biztonsági követelmények miatt.

A dolgozatnak tartalmaznia kell:

- a témát nyújtó vállalat ismertetését és a megoldandó feladat leírását,
- a tesztelési technológiák áttekintését,
- a már meglévő tesztrendszer leírását és vizsgálatát,
- a hardveres interakciós lehetőségek elemzését,
- a lehetséges környezetek infrastrukturális elemzését,
- a rendszertervet, döntések indoklásait,
- a megvalósítás leírását,
- egy szempontoknak megfelelő teszt bemutatását és eredményeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar 1. Budapest

Ph.

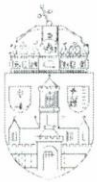
.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem.

Budapest, 2021. december 10.

Nemes Gergő

.....

hallgató aláírása

Titoktartási megállapodás

amely létrejött egyrészről a

I4P-Informatikai Kft.

Székhelye: 1125 Budapest, Fogaskerekű u. 4-6
Cégjegyzékszám: 01-09-199578
Adószám: 25098192-2-43
Képviseli: Rózsahegyi Zsolt
(a továbbiakban: **Társaság**),

másrészről a

Óbudai Egyetem

Székhelye: 1034 Budapest, Bécsi út 96/b
Intézményi azonosító: OM-FI12904
Adószáma: 15773063-2-41
Képviseli: Dr. Póser Valéria
(a továbbiakban: **Egyetem**)

együttesen Felek (a továbbiakban: **Felek**)

között az alább jelzett napon és helyen, a következő határidőkhöz és feltételekhez igazodva.

1. Előzmények

1.1. **Kucsera Gergő** (tanulmányi azonosító: **HVKTS7**) (a továbbiakban: Hallgató), az Óbudai Egyetem **Neumann János Informatikai** Karának **BSc** szakos hallgatója a „**Szabotázs detektáló modul hardveres tesztelési folyamatának automatizálása**" című **szakdolgozatában** (továbbiakban: Szakdolgozat) olyan bizalmasnak minősülő információkat, adatokat rögzít, amelyek zártkörű kezelése (a továbbiakban: titkosítása) indokolt a Társaság üzleti érdekeinek védelme érdekében.

1.2. Felek rögzítik, hogy az Óbudai Egyetem Tanulmányi és Vizsgaszabályzatának és Tanulmányi Ügyrend szabályzatának (a továbbiakban együtt: Szabályzat), megfelelően a Hallgató és a Társaság kérelmezték a Szakdolgozat titkosítását.

2. A megállapodás tárgya

2.1. Az Egyetem jelen megállapodás aláírásával tudomásul veszi, hogy a Szakdolgozatban szereplő, a Szakdolgozattal összefüggésben a Társaságtól származó személyes, informatikai, üzleti, műszaki és egyéb információ (a továbbiakban együttesen: Információ) bizalmas természetű és a Társaság tulajdonát képezi. Felek jelen megállapodás aláírásával tudomásul veszik és kijelentik, hogy az elkészült Szakdolgozat és mellékletei jelen titoktartási megállapodás hatálya alá tartoznak.

3. Felek jogai és kötelezettségei

3.1. A 2.1. pontban foglaltakra tekintettel az Egyetem a Szakdolgozatot a titoktartás szabályai és a Szabályzatban foglaltak szerint köteles kezelni. Az Egyetem vállalja, hogy a Szakdolgozatot üzleti titokként kezeli, azt a szerződés időtartama alatt harmadik személyek tudomására nem hozza, illetve semmilyen módon nem teszi hozzáférhetővé a Társaság előzetes írásbeli engedélye nélkül.

3.2. Társaság kijelenti, hogy az Óbudai Egyetem Szabályzatának a titkosított szakdolgozatok kezelésének eljárási rendjében foglaltakat megismerte és az abban foglalt rendelkezéseket a jelen megállapodás alkalmazásának vonatkozásában elfogadja.

3.3. Az Egyetem vállalja, hogy a Szabályzat szerinti eljárásrend alkalmazásával kerül sor a titkosított Szakdolgozat megvédésére. A 3.1. pontban előírt titoktartási kötelezettség alól kivételt képez a Szakdolgozat megvédésének folyamata, ahol az értékelő bizottság tagjai – előzetes titoktartási nyilatkozat aláírását követően – megismerik a Szakdolgozat tartalmát.

3.4. Az Egyetem kötelezettséget vállal arra, hogy kizárólag azok a közalkalmazottai ismerhetik meg a Szakdolgozat tartalmát, akik tekintetében ez feltétlenül szükséges a Szakdolgozat intézményen belüli, szabályos kezelése céljából. Az Egyetem köteles gondoskodni arról, hogy a titoktartási kötelezettség kiterjed az Egyetem közalkalmazottaira.

3.5. Társaság tudomásul veszi, hogy a Szakdolgozat elektronikus formátumú változata a Szabályzat értelmében feltöltésre kerül az intézményi könyvtárakba, ahol lehetőség van a titkosításhoz szükséges elérési jogosultsági szint beállítására.

3.6. Felek megállapodnak, hogy a titkosított Szakdolgozat kapcsán az alábbi adatok nyilvánosak, azaz nem képezik jelen megállapodás tárgyát:

a) a szakdolgozat címe, a szerző és témavezető neve és a védés időpontja

b) a titkosítás ténye és a titkosítás határidejének várható lejárta.

3.7. Felek megállapodnak, hogy jelen megállapodás időtartama az aláírásának napjától számított **5** év időtartamra szól. Ennek megfelelően a Szakdolgozatra vonatkozóan az Egyetem titoktartási kötelezettsége **5** év elteltével megszűnik és Társaság hozzájárul, hogy ezen idő leteltét követően a Szakdolgozat nyilvánosságra kerüljön.

3.8. Felek rögzítik, hogy az üzleti titok védelméről szóló 2018. évi LIV. törvény alapján az üzleti titok akkor is törvényes védelem alatt áll, ha külön szerzői jogi védelemben, szabadalmi oltalomban, használati mintaoltalomban vagy egyéb, jogszabályokban meghatározott, szellemi alkotásokra vonatkozó jogi védelemben nem részesül.

3.9. Felek rögzítik, hogy a jelen megállapodás szerinti titoktartási kötelezettség nem érvényesíthető államigazgatási (így különösen adóügyi) és bírósági eljárásban, továbbá azokban az esetekben, amikor jogszabály írja elő, hogy az információt a jogszabályban megjelölt személlyel közölni kell (pl. közérdekű, vagy közérdekből nyilvános adatok közlése), ezért ezekre nézve kölcsönösen és előzetesen mentesítik egymást a titoktartási kötelezettség alól, azzal a feltétellel, hogy Felek kötelesek előzetesen értesíteni egymást a jogszabályi kötelezettségről, illetve az eljárások tényéről és jogszabály alapján, illetve az eljárás során átadandó információk mértékéről.

3.10. Nem állapítható meg titoktartási kötelezettség továbbá az információk következő csoportjára:

(i) azon információ, amely nem az Egyetem hibájából vagy szerződésszegésének következtében került nyilvánosságra;

(ii) azon információ, amely a Társaság tudtán kívül már az átadás időpontjában közismert vagy bárki számára megismerhető volt;
(iii) azon információ, amely olyan személy által jutott nyilvánosságra, akiért Felek nem felelnek.
Felek a következő kapcsolattartó személyeken keresztül értesítik egymást jelen szerződéssel kapcsolatosan.

Az Egyetem részéről: Oktatási dékánhelyettes
Név, beosztás: Dr. Póser Valéria
Telefon: 0616665585
E-mail: poser.valeria@nik.uni-obuda.hu

A Társaság részéről:
Név, beosztás: Pető Ferenc
Telefon: +3617001230
E-mail: ferenc.peto@i4p.com

Felek a kapcsolattartók személyében bekövetkezett változásokról kötelesek egymást haladéktalanul, de legfeljebb három munkanapon belül írásban tájékoztatni.

4. Egyéb rendelkezések

4.1. Felek a jelen megállapodásból eredő vitás kérdéseket elsődlegesen tárgyalás útján egymás között rendezik, amennyiben ez 60 napon belül nem vezet eredményre, úgy Felek a Polgári Perrendtartásról szóló törvény mindenkori szabályai szerint járnak el.

4.2. A jelen megállapodásban nem szabályozott kérdések tekintetében a Ptk. rendelkezései, a hatályos magyar jogszabályok és a Szabályzat rendelkezései az irányadók.

4.3. Jelen megállapodás 4 (azaz négy) egymással szó szerint egyező eredeti példányban készült.

Felek jelen szerződést, mint akaratukkal mindenben megegyezőt, jóváhagyólag és saját kezűleg írták alá.

Az **I4P-Informatikai Kft.**
képviseletében:

Rózsahegyi Zsolt
ügyvezető

i4p-informati
1125 Budapest, Fogas
Adószám: 25098192

Az Óbudai Egyetem
képviseletében:

.....

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

A fenti megállapodásban foglaltakat megismertem és tudomásul vettem:

Budapest, 2021. 12.09.

Kucsera Gergő

Kucsera Gergő hallgató

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Kucsera Gergő	HVKTS7.....	Nappali .....

Telefon: [redacted] .....

Levelezési cím (pl: lakcím): [redacted] .....

Szakdolgozat / Diplomamunka címe magyarul:

Szabotázs detektáló modul hardveres tesztelési folyamatának automatizálása.....

Szakdolgozat / Diplomamunka címe angolul:

Automation of the hardware testing process of a tamper detection module .....

Intézményi konzulens:	Külső konzulens:
Lovas István.....	Stancz Krisztián .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.02.10	Témaválasztási konzultáció	
2.	2021.03.13	Szakirodalom áttekintése	
3.	2021.04.13	Lehetséges megvalósítás áttekintése	
4.	2021.05.04	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021. 05. 12.

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Kucsera Gergő

Neptun kód: HVKTS7

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka címe magyarul:

Szabotázs detektáló modul hardveres tesztelési folyamatának automatizálása

Szakdolgozat / Diplomamunka címe angolul:

Automation of the hardware testing process of a tamper detection module

Intézményi konzulens: Lovas István

Külső konzulens: Stancz Krisztián

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.21	Specifikáció áttekintése	
2.	2021.10.14	Rendszerterv áttekintése	
3.	2021.11.18	Fejlesztés, tesztelés ellenőrzés	
4.	2021.12.08	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021.12.09

.....
Intézményi konzulens

Tartalomjegyzék

1 BEVEZETÉS

1.1 A projekt háttere és célja

Szakdolgozatomat az I4P Informatikai Kft.-vel való együttműködéssel, egy céges projekt keretén belül valósítom meg. Az I4P kriptográfiával foglalkozik, azon belül főként hardveres biztonsági modul fejlesztéssel. A HSM[1]-ek olyan célorientált kriptoprocesszorok, melyek digitális kulcsokat tárolnak; azokkal titkosítást, hitelesítő és aláíró műveleteket végeznek. Többféle kialakításban léteznek; az I4P Trident HSM terméke készülék (appliance) típusú. Ez azt jelenti, hogy egy magas szintű biztonsággal ellátott házba egy komplett számítógép kerül, kiegészítő rendszerelemekkel. Különlegessége, hogy támogatja a többszereplős multiparty üzemmódot, azaz a kulcsadatok több eszközön elosztottan is jelen lehetnek, segítve a védelmet és a redundanciát. A Trident biztonsági tanúsítás szempontjából a Common Criteria EAL4+ szintjét célozza meg, melynek követelménye az aktív fizikai védelmet biztosító szabotázs detektáló modul megléte. Ilyen az I4P TDM[2], mely egy ARM processzoron alapuló mikrokontroller lapka, egyedi tervezésű nyomtatott áramkörrel, integrált szenzorokkal, külső I/O portokkal. Feladata a HSM adatainak védelme a belső és külső hatásokkal, támadásokkal szemben. Folyamatosan kommunikál a fő kriptográfiai szoftverrel, valós időben figyeli a szenzorok értékeit. Rendellenesség észlelése esetén azonnal leállítja a szoftveres folyamatokat, majd a rendszert újraindításra kényszeríti. Akkumulátort is tartalmaz, mely által szállítás vagy áramszünet közben is működőképes.

Mint általában egy hardverhez vagy szoftverhez, úgy a TDM-hez is szükséges teszteseteket kialakítani azon célból, hogy feltárjuk az emberi és gépi okokból adódó esetleges hibákat. A modulhoz korábban már készült egy tesztrendszer. A feladat célja egy olyan architekturális kiegészítés megtervezése és kivitelezése, mely a TDM hardveres tesztelését (pl. valós idejű feldolgozás, logika, IO/szenzorok) el tudja végezni automatikusan, folyamatos integrációs rendszerben.

Ebbe beletartozik :

- a tesztelő eszköz kiválasztása, programozása,
- a tesztelő szoftver kiegészítése,
- a *Jenkins*[3] folyamatos integrációs rendszer telepítése és konfigurálása, ezen belül:
    - a platform és operációs rendszer kiválasztása
    - a tesztelő szoftver automatikus fordítása, ütemezett futtatása

A folyamatos integráció egy DevOps területen használt fogalom. Olyan gyakorlati módszert és körforgást jelent, ami „lehetővé teszi a programkódok ellenőrzésének és tesztelésének felgyorsítását”, verziókezelő rendszerek és automatizálás segítségével. Általa a fejlesztésben eltérő szerepkörökkel bíró szakemberek közti kommunikáció hatékonysága lényegesen javulhat. [1]

[1] Hardware Security Module
[2] Tamper Detection Module
[3] Preferált szoftver: a cégnek ezzel van széles körű tapasztalata, más projekteknél is alkalmazzák

2 TESZTELÉSI FOLYAMAT ÁTTEKINTÉSE

2.1 A tesztelés fogalma

Tesztelési folyamat alatt információszerzési tevékenységet értünk. Tesztelés során egy termék minőségi és tartalmi hiányosságait, hibás működését próbáljuk felismerni, hogy azokon később javíthassunk. Ez magában foglalja az elvárt funkcionális működés időnkénti kipróbálását, kiegészítve szükség szerint a teljesítmény, a megbízhatóság, és a strapabíróság ellenőrzésével. A tesztek csak a hibák jelenlétét tudják megmutatni, a hibamentességet sohasem. A tesztelés feltárja az adott termékkel szemben támasztott követelmények teljesülésének állapotát, annak eltérését az elérendő célokhoz és a tervben leírtakhoz képest.

Kivitelezés szempontjából mindenképp szét kell választanunk a statikus analízist az automata teszteléstől. Míg előbbi a rendszer manuális, emberek általi elemzését jelenti, az utóbbi jól megválasztott lépések ismétlődő végrehajtásával igyekszik hibákat felfedezni.

Az automatizált tesztelési folyamatok tesztesetekből (test case) épülnek fel. Egy teszteset a tesztelendő rendszernek egyetlen, logikailag összetartozó elemi részét vizsgálja. Megadja, hogy adott feltételek mellett egy műveletsor eredménye megfelel-e az elvártaknak. A teszteseteket általában egységekbe, úgynevezett tesztkészletekbe szervezik. [2] [3]

Az informatikai szoftverek esetében az automatikus, számítógép által végzett tesztek elterjedtek. A programozott kódhoz tartozó teszteseteket is programozással állítják elő. Sok esetben a fejlesztéssel párhuzamosan, folyamatosan bővítik a tesztrendszer forráskódját is, melyet integrációs rendszerben lefordítanak, majd futtatnak.

2.2 A szoftvertesztelés típusai

A szoftverek tesztelésénél alapvetően kétféle megközelítési módot különböztetünk meg:

- Fehérdobozos (white-box) tesztelés
- Feketedobozos (black-box) tesztelés

2.2.1 Fehérdobozos tesztelés

A fehérdobozos, vagy strukturális tesztelés során magát a programkódot teszteljük közvetlenül, mely a teszt számára is látható. Az ilyen típusú tesztek alacsonyabb szintűek, kezelésüket legtöbb esetben a forráskódot fejlesztő szakember végzi, a hibakeresési folyamattal egybekötve.

Széles körben ismert fajtája az úgynevezett unit-teszt, vagy egységteszt. Ez a forráskód egy elemi részét (ami általában egy metódus vagy osztály) vizsgálja. A beadott paraméterek alapján az egység visszatérési értékét, bekövetkezett állapotváltozását, hatását hasonlítja össze az elvárttal.

A fehérdobozos tesztek közé tartoznak a modultesztek is, melyek a szoftverek nem-funkcionális működéssel kapcsolatos problémáit próbálják meg felismerni. Ilyen lehet például a maximális működési sebesség mérése, a memóriaszivárgások[4] megtalálása, és a szűk keresztmetszetek, optimalizációs problémák keresése.

[4] Felesleges memóriaterület foglalás, memory leak

Beszélhetünk még integrációs tesztekről, melyek a rendszerben lévő komponensek egymás közti kommunikációját, és az operációs rendszerrel való kapcsolódást tesztelik. Ezeket főleg nagyobb projekteknél alkalmazzák, ahol több fejlesztőcsapat dolgozik eltérő programrészeken. Az összeillesztés után megjelenő hibák feltárásában segít.

2.2.2 Feketedobozos tesztelés

A feketedobozos, vagy specifikáció alapú tesztelésnél a program forráskódját nem látjuk, csak a lefordított, kész szoftvertermék által biztosított interfészt. Specifikációnak nevezzük azt a dokumentumot, mely ezen interfész helyes működésének jellemzőit definiálja. Az ilyen típusú tesztek absztraktabbak, magasabb szintűek. Kezelésüket gyakran az erre szakosodott szoftvertesztelők, vagy külső partnerek végzik. A tesztelés helye és rendszere az esetek többségében eltér a fejlesztésétől.

A rendszerteszt feketedobozos teszt, melynek feladata a specifikációban leírtak teljesülésének, az egyes funkciók éles környezetben történő működésének analízise. Az itt megtalált hibákat hibabejelentő rendszerbe veszik fel, így tájékoztatva a fejlesztőket a javítási igényről. A rendszerteszteket a termék fejlesztői, vagy külső cég végzi.

Az átvételi (acceptance) teszt hasonló a rendszerteszthez. A különbség, hogy a tesztelési folyamatot ezúttal a felhasználók végzik el.

Több altípusa létezik: *alfa teszt, zárt béta teszt, nyílt béta teszt*

Az alfa teszteket a fejlesztő cég végzi el, de a fejlesztőktől független csapattal. Kézi tesztelést és segédprogramokat használnak olyan módon, hogy a valós használati esetekből adódó viselkedést is vizsgálni lehessen. Ez tulajdonképpen use-case tesztelésnek felel meg.

A zárt béta tesztet adott kritériumok alapján kiválasztott, külső felhasználók folytatják. A felfedezetlen hibák mennyisége még itt is magas lehet, főleg az eltérő számítógépes konfigurációk miatt.

A nyílt béta teszt, vagy felhasználói átvételi teszt a legutolsó szoftveres tesztelési fázis. Szinte az összes végfelhasználó megkapja a szoftver használati jogát. Ekkor a fő funkciók már mind működőképesek.

[4]

2.3 A TDM-tesztelés állapota

A TDM esetében az áramköri lap tesztelése és a szoftver fehérdobozos tesztelése az I4P offline zónában zajlik, a vezető embedded fejlesztő kolléga által. Offline zóna alatt egy olyan fejlesztési és adattárolási közeget értünk, ahol csak a belső intranet érhető el (tulajdonképpen egy külvilágtól elzárt hálózat). Mivel beágyazott rendszerről van szó, a unit tesztelést nem, vagy csak nehézkesen lehetne megoldani.

Tárgyalt témánk a TDM feketedobozos tesztelésén alapul, mégpedig olyan átvételi tesztelés és rendszertesztelés kombinációján, melynél a tesztelő felhasználó a fejlesztő vállalat egésze. Ennek oka, hogy a biztonsági követelmények miatt a szabotázs detektáló modult teljeskörűen, közvetlenül csak az I4P tagjai konfigurálhatják, a Trident HSM fejlesztése és előállítása során.

A HSM végfelhasználói csupán néhány korlátozottan elérhető parancsot használhatnak, illetve magát a modul nyújtotta védelmet kapják meg.

A black-box tesztelés középpontjában a mikrokontrollerbe égetett firmware program üzleti logikája áll, annak bemenetekre változó viselkedése, és a kimenetekre gyakorolt hatása. Ebből kifolyólag a hardverként felfogható portokat, szenzorokat, és a kimeneten megjelenő fogyasztókat is számításba kell venni, mint tesztelés során megjelenő tényező. Esetünkben tehát *nem egy hardvereszköz elektronikai jellemzőit kell tesztelni, hanem egy szoftvert*, hardveres tesztelési folyamattal az eredeti architektúrán.

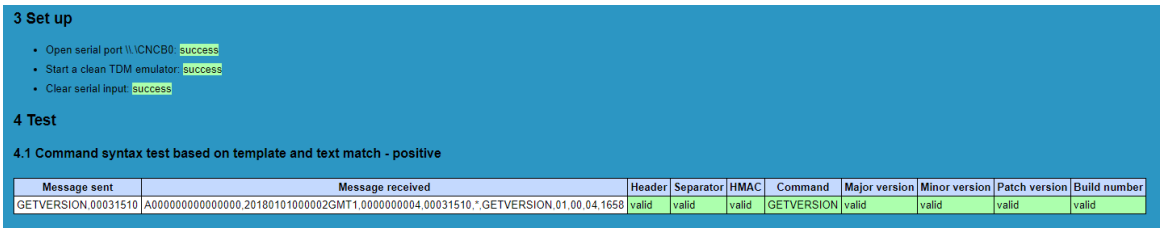
Az automata teszteléshez már készült egy tesztrendszer, melynek futtatása Windows 10, Ubuntu, és Red Hat Enterprise Linux operációs rendszereken zajlik. Ezekből az utolsó a Trident HSM operációs rendszere. Jelenleg a fejlesztői gépeken, illetve virtuális gépeken való kézi indítás fordul elő. Ez a tesztrendszer egy TDM emulátor segítségével csak a logikát ellenőrzi, a hardveres tesztelési folyamat ettől függetlenül manuálisan folyik. A cél a teljes automatizálás, ezen rendszer kiegészítésével.

A TDM-AT[5] tesztrendszer Java nyelven íródott, és a Maven fordítási menedzsert használja, mely a teszteket a Concordion keretrendszer segítségével futtatja. A Concordion egy olyan specifikációs eszköz, amellyel a megvalósítandó tesztesetek lépéseit HTML sablonokkal lehetséges megadni. A folyamat végén az eredményeket vizualizáló dokumentum is ebből generálódik. A sablonokon belül meghívhatóak a Java oldali, úgynevezett test fixture-ök függvényei, melyek a szükséges belső logikát biztosítják. Ez a módszer hasonlít a JavaService Pages (JSP) webtechnológia működési elvéhez. A következő két képen (2.1-es és 2.2-es ábra) egy példa látható a koncepcióra, a TDM-AT rendszerből.

```
<h4>4.1 Command syntax test based on template and text match - positive</h4>
<table concordion:execute="commandSyntaxTest(#msgToTdm, #getversion)">
    <tr>
        <th concordion:set="#msgToTdm">Message sent</th>
        <th concordion:echo="#getversion.msgIn">Message received</th>
        <th concordion:assert-true="#getversion.header.valid">Header</th>
        <th concordion:assert-true="#getversion.separator.valid">Separator</th>
        <th concordion:assert-equals="validateMsgHmac(#getversion.msgIn,#hex)">HMAC</th>
        <th concordion:assert-equals="#getversion.body.params[0].content">Command</th>
        <th concordion:assert-true="#getversion.body.params[1].valid">Major version</th>
        <th concordion:assert-true="#getversion.body.params[2].valid">Minor version</th>
        <th concordion:assert-true="#getversion.body.params[3].valid">Patch version</th>
        <th concordion:assert-true="#getversion.body.params[4].valid">Build number</th>
    </tr>
    <tr>
        <td>GETVERSION,00031510&amp;#10;</td>
        <td/>
        <td>valid</td>
        <td>valid</td>
        <td>valid</td>
        <td>GETVERSION</td>
        <td>valid</td>
        <td>valid</td>
        <td>valid</td>
        <td>valid</td>
    </tr>
</table>
```

2.1. ábra: Concordion sablon HTML részlete

[5] Tamper Detection Module - Acceptance Tests



3 Set up

- Open serial port \\.\CNCB0: success
- Start a clean TDM emulator: success
- Clear serial input: success

4 Test

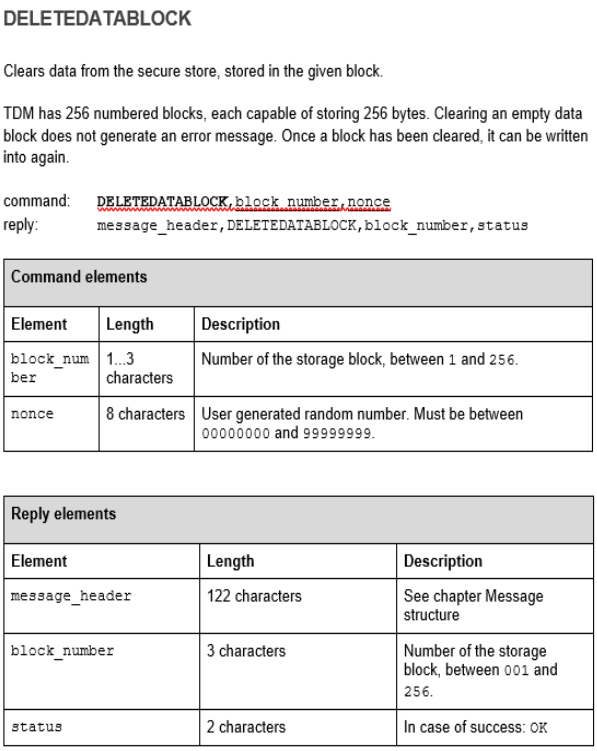
4.1 Command syntax test based on template and text match - positive

Message sent	Message received	Header	Separator	HMAC	Command	Major version	Minor version	Patch version	Build number
GETVERSION,00031510	A000000000000000,20180101000002GMT1,0000000004,00031510,*,GETVERSION,01,00,04,1658	valid	valid	valid	GETVERSION	valid	valid	valid	valid

2.2. ábra: Concordion eredmény HTML részlete webböngészőben

A teszteket típus szerint három részre bontottuk fel:

Amik a tesztelés legnagyobb részét kiteszik, azok a parancstesztek (2.3-as ábra). Az adott parancsok feldolgozásának, a hibák felismerésének helyességét figyelik. Esetenként az adott parancs által okozott állapotváltozást, viselkedést is ellenőrzik. Implementálásuk a „TDM felhasználói útmutató” specifikációs dokumentum alapján történik, mely belső használatra készült.



DELETEDATABLOCK

Clears data from the secure store, stored in the given block.

TDM has 256 numbered blocks, each capable of storing 256 bytes. Clearing an empty data block does not generate an error message. Once a block has been cleared, it can be written into again.

command: `DELETEDATABLOCK,block_number,nonce`
reply: `message_header,DELETEDATABLOCK,block_number,status`

Command elements

Element	Length	Description
`block_number`	1...3 characters	Number of the storage block, between `1` and `256`.
`nonce`	8 characters	User generated random number. Must be between `00000000` and `99999999`.

Reply elements

Element	Length	Description
`message_header`	122 characters	See chapter Message structure
`block_number`	3 characters	Number of the storage block, between `001` and `256`.
`status`	2 characters	In case of success: `OK`

2.3. ábra: Példa parancs specifikációra

A használati eset vagy use-case tesztek (2.4-es ábra) a TDM tanúsítási vizsgáztatásához szükséges, sűrűn előforduló és kritikus műveletsorokat tartalmazzák. Ez bizonyos szenzorok értékének változtatását és különböző parancsok sorozatát jelenti. Ebből a szempontból hasonlít a behavior részekhez azzal a különbséggel, hogy itt a teljes TDM reakcióját vizsgáljuk, nem a parancsokét. A use-case tesztek a Common Criteria-hoz szükséges „*Secure Functional Requirements (SFRs)*” dokumentációban szereplő specifikáció alapján készülnek.

<table>
<tr><td>Test ID</td><td>Tamper_test_TDM_3</td></tr>
<tr><td>Purpose of the test</td><td>Testing the tamper response based on maximum temperature monitoring of the Tamper Detection Module</td></tr>
<tr><td></td><td><table>
<tr><td>subsystem</td><td>Primary module</td><td>TSFI</td><td>SFR</td></tr>
<tr><td>HW</td><td>TDM</td><td>TDM_php.3</td><td>FPT_PHP.3</td></tr>
</table></td></tr>
<tr><td>Test scenario</td><td>Pre-conditions:
TDM has a battery installed.
TDM is connected to the PC via USB cable.
A LED is connected to the LED1 output. TDM is configured to light it up in case of tamper.
TDM internal temperature on the PCB is measured as 23.78 C
We set the allowed temperature range with the following command: SETONECONF,TEM2,-1000,2400,EVT1,EVT1,<nonce> (Generate tamper if the PCM temperature is less than -10C or more than 24 C).
Test case:
We heat the metal case of the TDM until it is surpassing the internal 24 C limit.</td></tr>
<tr><td>Expected test results</td><td>TDM:
automatically deletes its sensitive information (AES communication keys),
set its tamperstate to ‘1’,
LED connected to LED1 output lights up</td></tr>
</table>

2.4. ábra: Példa use-case specifikációra

Egyéb kategóriába sorolható tesztek is jelen vannak a rendszerben. Ezen teszteknek változó szerepe van. Van már közöttük stresszteszt, képességteszt[6], illetve hibajegyekben rögzített viselkedésbeli hibák ellenőrzésére szolgáló reprodukciós teszt is. Ezeknek a teszteseteknek forrásuk nincsen, tartalmukat mindig a körülmények határozzák meg.

[6] Speciális beállításokat, keretrendszerbeli funkciókat meg lehet-e valósítani hiba nélkül

3 A TDM BEMUTATÁSA

3.1 A TDM-interfész működése

A TDM feladata, hogy a környezetét monitorozza bemenetei és szenzorai segítségével, és ha bármilyen rendellenesség fordul elő, azt jelezze a védendő rendszernek.

A TDM USB-n keresztül kommunikál a számítógéppel, az UART protokoll segítségével. Virtuális soros port formájában jelenik meg az operációs rendszerek számára. Ezen az interfészen lehetséges szöveges üzenetek formájában beállításokat elvégezni, információkat lekérni. A TDM-AT tesztelés nagy része is az ezen interfésszel való „beszélgetésből" áll. Az üzenetek angol betűket és kifejezéseket használnak; végig nagybetűsök; lezáró karakterük az új sor (LF), vagy a kocsi vissza - új sor (CR LF) lehet. Az üzenet részeket, paramétereket a ' **,** ' [vessző] karakter választja el egymástól. Az eszköz alapesetben minden üzenetet és történést naplóz. A naplózott adatokat az interfészen keresztül le lehet kérni.

A TDM számára küldendő üzenetek sorrendben a parancs nevéből, annak paramétereiből, valamint a nonce-ból állnak. A nonce egy nyolcjegyű egész számból álló biztonsági elem, melyet minden válaszüzenet tartalmaz. Mindig más, véletlen számnak ajánlott lennie; az üzenetek hitelességben van szerepe. Általa le tudjuk ellenőrizni, hogy egy válasz valóban a beadott parancsra érkezett-e meg.

Minden TDM által visszaküldött válasz- és állapotüzenet két részből áll. Az első rész a fejléc vagy header, mely metaadatokat tartalmaz. Ennek tartalma fix és mindig megelőzi a tényleges üzenet tartalmát.

Részei:

893436140D192100,	20200101000929GMT1,	0000009022,	77711152,	BB101A6229540…,
A TDM sorozatszáma	Időbélyeg	Az üzenet sorszáma a logban	Nonce[7]	Üzenet hash

3.1. táblázat: A TDM üzenetfejléc részei

Ami a fentiek közül magyarázatra szorul, az a hash. Hash-nek nevezzük a rögzített méretű digitális lenyomatot az üzenet tartalmáról. A módszer HMAC algoritmust használ, mellyel ellenőrizhető a feladó eszköz hitelessége, illetve az üzenet integritása. Ez egy titkos kulcs (HMAC kulcs) alapján generálódik, melyet a TDM üzembehelyezésénél szükséges megadni. Hasonló módon a naplóbejegyzések is hitelesítve vannak.

A második rész az üzenet tartalma, mely a parancs- vagy jelzésnevet, közölt adatokat, illetve nyugtázást tartalmazhat. A paraméterszám és üzenethossz emiatt változó nagyságú. Az üzeneteknek többféle fajtája van, melyeket az alábbi táblázat ismertet:

[7] REPLY üzenet esetén, egyébként *-gal maszkolva

Üzenet típus	Funkció
REPLY	Az általunk beküldött parancs nyugtázása, eredményének visszaadása
HEARTBEAT	Időközönként: a TDM jelenlétének jelzése, szenzor értékek kiküldése Azonnal: behatolási esemény jelzése
INTERRUPT	Digitális szenzorok/bemenetek állapotváltozásakor keletkező megszakítás jelzése
UPDATE	TDM működésről, üzemmódváltásról való tájékoztatás
ERROR	Az előforduló hibák jelzése a felhasználó felé, szövegesen

3.2. táblázat: TDM üzenet típusok

Ezek közül jellemzően a REPLY és a HEARTBEAT üzenetek fordulnak elő a legtöbbször. Ebbe a két típusba adnak betekintést az alábbi példák.

Jelenlétet jelző HEARTBEAT üzenet

```
  2227485114555555,20180101020512GMT1,0000009934,********,**********
**********************************************,HEARTBEAT,EVT1,0,EVT2,0,EVT
3,0,EVT4,0,VTDM,04895,0,VUSB,04895,*,VBAT,03414,+,AIN1,00000,*,AIN2,00000,*
,AIN3,00000,*,AIN4,00000,*,TEM1,02900,0,TEM2,02631,0,TEM3,00920,0,DIN1,0,*,
DIN2,0,0,DIN3,0,0,DIN4,0,0,DIN5,0,0,DIN6,0,0,DIN7,0,0,DIN8,0,0,MIC1,0,*,OPT
1,0,0,OPT2,0,0,VIB1,0,*,PUL1,0,*,MOT1,0,*,CHRG,0
```

Számlálókat lekérő parancsüzenet

Parancs: `GETCOUNTERS,`**`12345678`**

Válasz:

```
2227485114555555,20180101002927GMT2,0000000177,12345678,**********
**********************************************,GETCOUNTERS,0000000013,000000
0001,0000000034
```

HEARTBEAT időközét beállító parancsüzenet

Parancs: `SETONECONF,CHEARTBEAT,8,`**`11223344`**

Válasz:

```
2227485114555555,20180413152646GMT1,0000011504,11223344,**********
**********************************************,SETONECONF,CHEARTBEAT,00008
```

A félkövér betűkkel kihúzott értékek a nonce értékeket mutatják, melyek célja itt jól látszik. HMAC kulcs nincs beállítva a jobb olvashatóság érdekében.

3.2 A TDM fizikai felépítése

Fizikai felépítés szempontjából számunkra a kommunikációs, interakciós egységek fontosak. A TDM mindkét oldalán, sorban foglalnak helyet a csatlakozók, melyekhez a PATA-éhoz hasonló szalagkábelek tartoznak. A TDM logikai feszültsége 2,8 V, amely kissé alacsonyabb, mint az elterjedt 3,3 V-os szabvány.

A TDM 12 darab bemeneti porttal rendelkezik. Ezekből nyolc (DIN1 - DIN8 jelzéssel) digitális input, amelyek átbillenéskor megszakítást hoznak létre. Általában a készülékház megnyitásának érzékelését végzik. A többi négy bemenet analóg (AIN1 – AIN4 jelzéssel). Egyszázad volt pontossággal, folyamatosan mérnek közvetetten feszültséget feszültségosztóval, maximum 15 V-ig. Feladatuk legtöbb esetben az alaplapi feszültségek mérése.

A bemeneti aljzatok mellett a TDM fel van szerelve különféle, komplikáltabb szenzorokkal is. Rendelkezik két, belső hőmérsékletmérővel, melyek -50 és 85 °C közötti értékeket képesek megkülönböztetni. Ezenfelül integrálva van rá gyorsulás- és rezgésérzékelő, fényérzékelő, és egy mikrofon is. Továbbiakat is rá lehet kötni egyéb portokon, illetve I^2C buszon keresztül.

Kimenetként az USB interfész, valamint relék (REL1, REL2) és négy digitális kimenet (LED1-LED4) érhető el visszajelzés, vagy külső áramkörök irányítása céljából.

A megvalósítási folyamat során kizárólag a digitális és analóg bemenetek, valamint a kimenetek lesznek kipróbálva, ugyanis a *fizikai jelenségeket figyelő* szenzoroknak a speciális környezeti feltételek mellett egyedi elektronikára is szükségük van arra, hogy tesztelni lehessen őket. Ezek elkészítése túlmutat a dolgozat témakörén, ezért jelenleg nem kerülnek kidolgozásra.

3.3 TDM-események

A TDM-ben eseményeket lehetséges hozzárendelni a szenzorok és bemenetek által mért értékekhez. Alsó és felső határértékeket, aktív/inaktív állásokat lehet rögzíteni, melyeket átlépve a folyamat aktiválódik. Az események kiválthatják belső műveletek elindítását, illetve kimeneteket vezérelhetnek. Utóbbira példa a Tridentben is alkalmazott, előlapi status LED-del való állapotjelzés. Egy eseményt egyszerre több bemenethez, egy kimenetet pedig több eseményhez is hozzá lehet rendelni. Konfigurációnál, ha nem szeretnénk társítani, akkor a null eseményt (EVT0) kell megadni, ami nem vált ki semmiféle eljárást. Bekövetkezés után az eseményeket paranccsal vissza kell állítani, ugyanis a szenzorokhoz hasonlóan állapotjelző flagekkel rendelkeznek.

A TDM négy különböző, hiearchikusan rendezett eseményt tud kezelni. Ha több aktiválódik párhuzamosan, mindig a legfontosabb fog érvényre jutni. A legalacsonyabb fontosságú az EVT4, majd következik az EVT3, EVT2, EVT1. A felsoroltakból kiemelkedő az EVT1-es, mely a behatolási, más néven tamperesemény. Ez halaszthatatlan, azonnali beavatkozást igényel. Alapértelmezett hatása, hogy leállítja a fő kriptográfiai szoftvert a HEARTBEAT segítségével (amely nem is tud ezután elindulni), majd újraindítja a HSM-et. A hitelesség fenntartásához szükséges HMAC kulcs is törlődik. Akár a TDM átkonfigurálása is letiltható ebben az állapotban.

4 INTERAKCIÓS LEHETŐSÉGEK

4.1 Az elemzés szempontjai

A TDM-mel és az elektronikával való interakcióhoz egy olyan hardvereszközt kell keresni, amely programozható, általános célú csatlakozókat (General Purpose Input-Output, GPIO) tartalmaz, és azok elérhetőségét biztosítja a tesztrendszer futtató eszköz felé. Több szempontot és követelményt kell figyelembe venni az elemzés és kiválasztás során, melyek alapján a döntés megszülethet. Ezeket a cég igényei és saját tapasztalataim alapján állítottam fel. Egyelőre csak egyetlen tesztkörnyezet vár beüzemelésre.

1. Infrastruktúra:
   - Linux operációs rendszer / kompatibilitás kötelező
   - Jenkins kötelező
   - Elegendő számú, 2,8 V-ra képes digitális kimenet, bemenet
   - A kimenetek időzítésének teszteléséhez szükség van külső megszakítások kezelésére
   - Hálózati csatlakozás megléte: vezetékes vagy vezeték nélküli is megfelelő
   - Szabványos, cserélhető tárhely; az integrált kizárt
   - Előny a platformfüggetlenség
   - Előny a meghajtók kezelése, a RAID képesség
   - Előny a dedikált analóg kimenet
2. Hibakezelés:
   - Napi 24 órás üzemidő
   - Javíthatósági szint
   - Ügyelni kell a megfelelő hűtésre; kiegészítők vásárlása is indokolt lehet
3. Teljesítmény elvárások:
   - A lefutási idő közelítse meg a jelenlegi szintet (30-40 perces a teljes tesztfutás)
   - A készre felépített rendszer ne legyen túlterhelve, a rá bízott feladatra legyen méretezve - a nem megfelelő teljesítményből eredő, futás közbeni időzítési hibák megengedhetetlenek
   - Jelentős pozitívum a későbbiekre nézve, ha a futtató számítógép bővíthető kialakítású
   - A tesztek nem igényelnek nagyfrekvenciás[8] kapcsolási sebességet
4. Fejlesztési idő, nehézség:
   - Könnyen használható, továbbfejleszthető legyen, ne legyen feleslegesen bonyolult
   - Előny, ha plusz alkalmazás nélkül, Java könyvtárral beilleszthető a TDM-AT tesztrendszerbe
5. Fogyasztás, méret:
   - Az energiafogyasztás és fizikai méret alacsony prioritású szempont
   - Akár szerver szintű számítógépbe is kerülhet a rendszer; van szerverterem
6. Költségek:
   - Nincs kimondott összeghatár, de ne legyen indokolatlanul drága
   - Nem feltétel egyetlen extra funkció sem a be- és kimeneteken kívül

[8] Nagyságrendi példa: 10.000 állapotváltozás/másodperc

4.2 Különálló eszközök

Az első csoportban azok az eszközök foglalnak helyet, melyek a futtató környezettől diszkrét módon elválaszthatók, saját magukban is működőképesek. Általában szabványos interfésszel kapcsolhatók a rendszerbe, ezáltal cserélhetők, hordozhatók. A futtató környezet ezen esetekben bármilyen x86 alapú számítógép lehet. A szükséges teljesítmény biztosítása, illetve az integrációs rendszer és függőségeinek beállítása ennek következtében biztos, hogy megoldható.

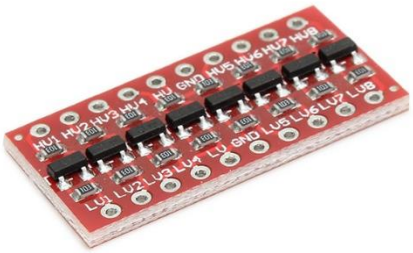
Ebbe az irányba már történt próbálkozás a cég részéről. Egy, a TDM-mel azonos mikrokontrollert használó tesztelő lapkát alkottak meg. Ez a módszer végül nem vált be a beüzemelési, konfigurációs nehézségek, és a fejlesztés miatt. Ezen firmware fejlesztése sok időt igényel, ugyanis sok mindent az alapoktól kell megírni, közösségi támogatás híján (nem széleskörűen elterjedt platform). Más, ipari szintű MCU-k is hasonlóan bonyolultak általában. Legnagyobb előnye az lett volna, hogy a logika feszültségszintje megegyezik a TDM-évl.

4.2.1 Arduino mikrokontrollerek

A hobbisták körében népszerű Arduino platform megoldás lehet a problémára. A keretrendszerrel kompatibilis MCU lapkák könnyen programozhatók és használhatók.

A referencia modellek közül az egyik legtöbb szolgáltatást nyújtó Due modult (4.1-es ábra) elemeztem részletesebben. 54 darab, bemenetnek és kimenetnek is beállítható digitális porttal rendelkezik, melyek a 16 digitális TDM csatlakozó és a jövőbeli szenzor áramkörök kiszolgálására bőven elegendő. Csak 2 darab, 0,55 és 2,75 volt közötti feszültségre képes digitális-analóg átalakítót (DAC) tartalmaz. A TDM-nek ez túl kevés mennyiségű, illetve túl szűk feszültségtartományban működnek. Az IO portok kimenet módban feszültségforrásként viselkednek, azaz működésnél a lapka által biztosított feszültség jelenik meg rajtuk. A legtöbb Arduino lapka logikai feszültsége 5 V, a Due-é viszont csak 3,3 V. Még ez is magasabb a kívántnál, ezért, hogy össze lehessen illeszteni a TDM-mel, feszültségosztót vagy szintillesztőt (4.2-es ábra) szükséges beiktatni. A szintillesztők olyan áramkörök, melyek két logikai feszültségszint közti konverziót tesznek lehetővé, akár mindkét irányban. A Due esetében csak a TDM bemenetek felé szükséges átalakítás, hogy azok ne menjenek tönkre. A 3,3 V-os Due bemenet a 2,8 V-os TDM kimenetet is logikai 1-nek érzékeli. [5] A Due minden digitális portja képes megszakítások kezelésére. [6]

4.1. ábra: Arduino Due



4.2. ábra: Szintillesztő áramkör

Az eszköz két microUSB csatlakozóval rendelkezik, melyek közül az egyikre a TDM-hez hasonlóan egy USB-UART csip csatlakozik. Virtuális soros portként látszódik az operációs

rendszerek számára, itt lehet az eszközzel kommunikálni, a programozást is ezen ajánlott elvégezni.

A Due relatív olcsón beszerezhető, 12 ezer forint körüli áron. Klónjaihoz akár féláron is hozzá lehet jutni. Meghibásodás esetén bármikor, bármilyen gyártmányúra kicserélhető, a megírt firmware nagy része (az esetleges alacsony szintű kódokon kívül) minden Arduino variánson és minden PC-vel ugyanúgy fog működni. Fogyasztása elhanyagolható, USB-ről táplálható.

A tesztrendszerbe való beillesztéshez egy firmware-t kell megírni és feltölteni rá C nyelven, illetve a TDM-hez hasonlóan egy Java soros kommunikációs osztályt kell neki létrehozni.

[7]

4.2.2 Advantech adatgyűjtő modul

Az Advantech egy tajvani székhelyű, beágyazott rendszerekkel és ipari automatizációval foglalkozó vállalat. Széles termékskálával bír, sok specializált eszközt és célhardvert fejleszt. Termékei között találtam rá a nagy teljesítményű és precíz adatgyűjtésre, adatelemzésre és kommunikációra lehetőséget adó DAQ[9] modulokra. Ezeket sokféle kivitelben és felszereltséggel gyártják; én a hordozható, *USB-4751* típusú eszközt néztem meg részletesebben (4.3-as ábra).

4.3. ábra: Advantech DAQ

Az Arduinóval ellentétben előre fel van programozva. USB-vel kapcsolódik a számítógéphez, de létezik teljesen önálló, illetve bővítőkártyás változat is. 48 darab, bemenetként és kimenetként egyaránt beállítható digitális csatornája van. 5 V-os feszültséggel működik, ezért szintén konverzióra van szükség az illesztéshez. Analóg kimenete nincs, megszakításokat támogat. Egyedi illesztőprogramot használ, melyhez egy grafikus segédprogram tartozik, így a tesztrendszerbe való beillesztése bonyolult lehet. Eszközmeghajtó elérhető Linuxra is, de támogatása alapvetően Windows-orientált.

Extra hardvervédelmet, kiemelt pontosságot és hatékony integrációt biztosít ipari szabványoknak megfelelő környezetben. Mivel nem logikai vezérlőként, hanem adatgyűjtőként való használatra van felkészítve, ezért a hozzá tartozó ökoszisztéma is erre koncentrál.

[9] Data Acquisition

Leginkább a gépipar használja okos gyárakban, automatizációs láncoknál, vagy laboratóriumok.

[8]

4.2.3 LucidControl IO modulok

A német Deciphe-it GmbH által kifejlesztett, előre programozott LucidControl USB IO modulok (4.4-es ábra) kifejezetten vezérlésre lettek kitalálva, de adatgyűjtésre is használhatók. Beüzemelésük és használatuk jóval egyszerűbb, mint az Advantech moduloké. A gyártó ipari, de otthoni és hobbi környezetbe is ajánlja. Az eszköz platformfüggetlen, Linux-szal is működik egy általános, sok rendszerben megtalálható illesztőprogram által. A vállalat egy parancssoros kezelőprogramot, illetve egy Java könyvtárat is készített a termékhez. A TDM-AT-ban mindkettőt lehetséges implementálni. Különböző megengedett feszültségszinttel (5, 10, 24 V stb.) és felszereltséggel kaphatók, 4 és 8 csatornás kivitelben. Moduláris felépítésűek; több egységet burkolatukkal fizikailag egymáshoz lehet illeszteni és minden modult egyetlen szoftverből kezelni. Csatlakozóihoz a kábelek rögzítése sorkapocs segítségével valósul meg; ez megkönnyíti a szerelést. Meghibásodás esetén a cég az alkotó komponensek javítását vállalja. Mindegyik modul USB-ről kapja a működéshez szükséges tápfeszültséget. [9]

4.4. ábra: LucidControl modul

A TDM hardveres tesztrendszerének kialakításához szenzorvizsgálat nélkül 4 darab, különböző kialakítású modell egyszerre való alkalmazása nyújthat megoldást:

1. 2x Digitális bemenetes (4 csatorna, 5 V)
2. Digitális kimenetes (8 csatorna)
3. Analóg kimenetes (4 csatorna, 0 – 24 V)

A digitális bemenetes modul a 0 és 1 értékeket különbözteti meg. Negatív polaritást is rá kell kötni, amit kiválthatunk a TDM GND pinjével minden portnál. Extra funkcióként egy megszakításhoz hasonló, de annál kevésbé hatékony pufferelt jelszélérzékelés érhető el. Sajnos szintillesztést követel meg (fordított eset), mert 5 V-tal operál a legkisebb elérhető típus, és annak igaz logikai jel minimuma 3,5 V, ami magasabb a TDM 2,8 voltjánál. [10] [11]

A digitális kimenetes modul szilárdtest reléket (Solid State Relay) tartalmaz. Ez a relé abban különbözik a hagyományostól, hogy félvezetők alkotják, így nincs mozgó alkatrésze. Hangtalan

és az élettartama is nagyobb, gyorsabban kapcsol. [12] Ennek működése eltér az Arduinónál tapasztaltaktól. Itt a modul nem egy feszültségforrást kapcsol ki-be az érintkezőin, hanem egy kaput. Ezáltal 24 V-ig bármekkora feszültséggel működtethető, de ehhez külső feszültségforrást igényel. Időzített és PWM[10] módot is kínál. [13] [14]

Az analóg kimenetes modul 4 darab DAC-ból áll, melyeknek földpontjai szintén az USB-n vannak, ezért egyesíteni kell őket a TDM-évl. 12 bites felbontás mellett 0,25% pontossággal képesek előállítani a kívánt értékeket. Számunkra a legnagyobb, 24 V-os változat szükséges a TDM maximumának - 15 V - eléréséhez. A cég szerint valódi, pontosan nulla potenciál is elérhető velük. [15]

A digitális IO-t szolgáltató típusoknál a biztonság növelése érdekében a portok optocsatolóval vannak elkülönítve az azokat kezelő elektronikától. Az optocsatolók olyan áramköri elemek, melyek az elektromos jeleket fénnyé alakítják át, majd vissza. Ezzel teljes, úgynevezett galvanikus leválasztást[11] hoznak létre. [16] A portok pillanatnyi logikai állapotát LED-ek jelzik.

A modulok összesített költsége viszonylag magas, egy belépőszintű PC árának környékén van.

4.3 Számítógépek GPIO-val

Ebbe a csoportba olyan eszközök tartoznak, melyek egy komplett számítógépet foglalnak magukban, de nem az asztali és szerver rendszerekben megszokott x86 – ATX architektúra alapján épülnek fel. Az okostelefonokban is alkalmazott ARM Cortex-A magokat kínálják, de azokkal ellentétben megengedik tetszőleges operációs rendszerek használatát a felhasználónak.

4.3.1 Raspberry Pi 4

A világszerte sikert arató Raspberry Pi fejlesztői lapka eredetileg 2012-ben jelent meg. A jelenlegi negyedik generációt 2019-ben adták ki. Jelentős felhasználása van a hobbielektronikában, mikroszerverként, az IoT és automatizáció területén. [17] Az Arduino Due-hoz hasonló méretű lapkáról van szó, melynek közösségi támogatása szintén kimagasló.

Ebben a fejezetben a Raspberry Pi 4 Model B kerül elemzésre (4.5-ös ábra). Négy Cortex-A72 magot tartalmazó, 64 bites ARMv8 CPU-ja van. 2, 4, vagy 8 GB integrált memóriával rendelhető. Ez nem cserélhető, így bővíteni csak új modul vásárlásával lehet. Négy darab USB csatlakozó, valamint gigabites Ethernet vezérlő található benne, WiFi-vel egyetemben. HDMI-t használ a képmegjelenítéshez. Könnyen megoldható a távoli elérés is, például SSH használatával. A Pi-vel használni kívánt operációs rendszert PC-ről kell telepíteni egy microSD memóriakártya háttértárra, mely behelyezéskor azonnal elindul. Nincsen sem M.2, sem SATA csatlakozó, ebből kifolyólag RAID sem. [18]

[10] Pulse Width Modulation
[11] Fizikai, fémes kontakt nincsen a két rész között

4.5. ábra: Raspberry Pi 4 Model B

Összesen 26 darab 3,3 V logikai szintű, bemenetnek és kimenetnek egyaránt beállítható digitális IO port érhető el vezérlési célokra. Analóg kimenetet nem találunk rajta, csak PWM funkciót. Minden IO port képes megszakításkezelésre. Működésük hasonlít az Arduinónál tapasztaltakhoz. [19] A TDM bemenetek felé itt is szükség van szintillesztésre. A lapka bemenetei nagy valószínűséggel jól észlelnék a TDM kimeneteinek jeleit, ugyanis az 1,6 V feletti feszültséget igaznak veszik. [20] A GPIO-t számos szoftverrel és programozási nyelvvel lehetséges kezelni. A Pi4J könyvtár Java nyelven íródott, az eddig megjelent összes Pi-vel kompatibilis. Köztes réteg beiktatása nélkül, egy Maven függőség letöltésével munkára fogható. Könnyen beépíthető a TDM-AT projektbe, azonban platformfüggővé teszi azt. [21]

Az eszközre elérhető az Ubuntu Linux, szerver és desktop változatban is. A Java futtatókörnyezet telepíthető rá; ebből adódóan a Jenkins[12] is. A virtuális gépnek[13] köszönhetően pedig meglévő tesztrendszerünk futtatható Raspberry-n is. Jelen feladat témájához hasonlóan, a Pi 4 Jenkins CI szerverként való felhasználása már felvetődött a StackExchange fórumán. A mérésekből következtetve, illetve a felhasználók tapasztalatai alapján az eszköz alkalmas a feladatra, bár csak SSD rendszermeghajtóval ad kielégítő teljesítményt. [22] SSD-t USB 3.0 adapteren keresztül, külsőleg lehetséges csatlakoztatni, melyre tükrözni kell a memóriakártyán lévő OS partícióit. [23] Megoldható esetleg az is, hogy a Pi a tesztek futtatásáért nem felel, csak a vezérlésért. Ez azonban egy PC-vel való hálózati kommunikációt igényelne, ami növeli a tesztrendszer komplexitását és extra késleltetést hozhat magával. Továbbá a lapka képességeit és számítási kapacitását így nem használnánk ki.

Hőmérsékleti szempontból a Pi 4 processzora tartós terhelés alatt akár a 70-80 Celsius fokot is elérheti. Ez alapvetően nem jelent kockázatot, mert bekapcsol a throttling[14] mechanizmus. [24] Véleményem szerint mégis érdemes szakboltokban kapható hűtőbordát és ventilátort alkalmazni azon célból, hogy ne legyenek teljesítményproblémák, valamint, hogy a számítógép élettartama növekedjen. A Raspberry Pi 4 tipikus fogyasztása meglepően alacsony egy x86 szerverhez képest, mindössze 3 W körüli. A tápfeszültséget egy USB-C csatlakozón át, fali adapterből kapja. [25] Árban az Arduino és az IO modulok között foglal helyet.

[12] A Jenkins Java nyelven íródott
[13] Java Virtual Machine, JVM
[14] A CPU órajel és üzemi feszültség lejjebbvétele a károsodás megelőzése érdekében

4.4 Egyéb eszközök

Az előzőekben leírtak mellett felmerültek még egyéb ötletek is. Ezeket csak említés szintjén mutatnám be. Már a felvetéskor, vagy nem sokkal azután meghiúsult felhasználásuk lehetősége.

A Controllino egy olyan Arduino és programozható logikai kontroller keverék, melyet felhasználója nyíltan testre tud szabni. Kinézetileg és funkciókban hasonló az IO modulokhoz, azonban ez számítógép nélkül is működőképes. Csak az ipari 12/24 voltos változatban kapható, így a TDM-hez nem illeszthető. [26]

A Banana Pi, Rock Pi lapkák a Raspberry Pi alternatíváiként foghatók fel. Az architekturális felépítés mellett néhány tulajdonságukban térnek el tőle (pl. CPU típusa, extra portok, csatlakozási lehetőségek), de alapvetően a funkcionalitásuk, teljesítményük, céljuk hasonló. Mivel ezen platformok kevésbé ismertek, támogatottságuk jóval alacsonyabb. Emiatt az elérhető szoftveres eszközök száma is kevesebb.

Az UDOO Bolt egy olyan x86-alapú egylapkás számítógép, amely integrált Arduinót tartalmaz. Specifikációi alapján teljesítménye elegendő lehet a kívánt célhoz. Az ARM egylapkás számítógépekhez képest jobban bővíthető és javítható, de csak korlátozottan. Otthoni és IoT használatra van tervezve, kis mérettel és fogyasztással. A normál számítógép és különálló Arduino kombinációval azonos funkcionalitást nyújt tesztelésnél, de drága és helyhez kötött. Nem megfelelő számunkra. [27]

Az I4P-nél való munkám során találkoztam a Supermicro cég x86 architektúrájú, Intel Xeon-D processzort felvonultató alaplapjaival. A Xeon-D egy olyan SoC (System on a chip), mely a Raspberry Pi-hez hasonlóan saját GPIO-val rendelkezik. Ezek elérése csak a processzor egyedi regiszterein keresztül lehetséges, assembly nyelven. Ez rendkívül platformfüggő megoldás; ha megszűnik a széria gyártása, nem lehet pótolni a hardvert. Emellett bonyolult és túl kevés portot biztosít, így elvetésre került.

4.5 Eszköz kiválasztása

4.1. táblázat: Interakciós eszközök összehasonlítása

	Arduino MCU-k	Advantech DAQ	LucidControl	Raspberry Pi
Linux támogatás	igen	kérdéses	igen	igen
Platformfüggetlen	igen	kérdéses	igen	nem
Fejlesztés nehézsége	egyszerű	bonyolult	egyszerű	közepes
Analóg kimenet	nem megfelelőek	nincs	van	nincs
Megszakításkezelés	igen	igen	nem	igen
Szoftveres testreszabhatóság	igen	nem	nem	igen
Hibák kezelhetősége	jó, könnyen helyettesíthető	közepes, gyártófüggő	közepes, gyártófüggő	rossz, nagy integráltság
Tesztkörnyezet teljesítménye	tetszőleges, bővíthető	tetszőleges, bővíthető	tetszőleges, bővíthető	fix, csere szükséges
Ár	alacsony	közepes	magas	közepes

A 4.1-es táblázat a fejezetben mélyebben elemzésre került eszközöket hasonlítja össze a főbb szempontoknak megfelelően.

A tervezés előtti utolsó lépés a számunkra ideális interakciós eszköz kiválasztása a lehetőségek közül. Az előző alfejezetekben leírtak, az összehasonlító táblázat és a céggel való konzultációs megbeszélések alapján az Arduino-kompatibilis, Raspberry Pi Pico mikrokontrollerre (4.6. ábra) esett a választásom. Ezen eszköz a gyártó első, nem egylapkás számítógép besorolású terméke. A kiválasztás többszöri mérlegelés után, a következő gondolatmenet alapján történt:

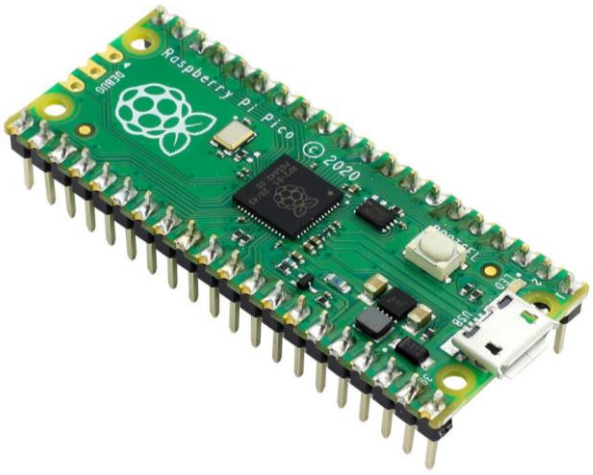
Az Advantech DAQ modulnak túl sok hátránya van szemszögünkből nézve: Windows-ra koncentráló szoftver, beüzemelési bonyolultság, a felépítés ipari jellege.

A LucidControl IO modul jó funkcionális és biztonsági felszereltséggel rendelkezik, kényelmes a használata. Ezzel szemben nem kezel megszakításokat, illetve képességeinek egy nagyobb része általunk kihasználatlan lenne. Felépítéséből adódóan nagy teljesítményű fogyasztók vezérlésére találták ki. Ez a négy modul összegzett árán is meglátszik; mely körülbelül 10-szerese(!) az Arduino Due-énak. Erre nincsen szükség, egyszerűbb megoldás is elegendő.

A Raspberry Pi 4 legnagyobb előnye, hogy egy eszközben csoportosul minden - a tesztelő komputer és vezérlési funkció is - így ezek összeillesztésére kevesebb figyelmet kell fordítani, kisebb a szoftveres hibalehetőség. Ezen túl, mivel többfeladatos operációs rendszereket tud futtatni, más szolgáltatásokat is üzemeltethetünk rajta az integrációs eszközzel párhuzamosan. Az ebből épített környezet hordozhatóvá válna. Olyan hátrányai vannak viszont, melyek a jövőben okozhatnak kellemetlen problémákat. Ilyen a teljesítmény kérdése, mely nem

növelhető lecserélés nélkül, ha szükség lenne rá. A külső rendszer SSD használata is anomáliákat idézhet elő. A RAID hiánya egy jövőbeli meghibásodásnál akár az egész futtatási környezet újratelepítését vonhatná maga után. Mindemellett a TDM-AT tesztrendszer hardveres módjának implementációja is csak Raspberry platformokon működne.

Az Arduino környezet univerzális megközelítési módja, egyszerű és testreszabható fejlesztési koncepciója, illetve jó hibakezelhetősége miatt lett a győztes. A Raspberry Pi Pico a szakdolgozat kutatási fázisában még nem volt elérhető (éppen akkor jelent meg a piacon), ezért utólag került kiválasztásra. Az IO portok száma kevesebb mint a Due-nél: 26 az 54 helyett, de ez is elegendőnek bizonyul. Nála jóval olcsóbban, 2 ezer forint körüli áron szerezhető be. Feszültségben is megegyeznek, viszont a Picót kialakítása révén próbapanelbe lehetséges helyezni, ami jelentősen segíti a bekötést, menedzselést. Csak azon szolgáltatásait kell használnunk, amire szükségünk van, nincsen feleslegesen túlbonyolítva. Ezzel szemben egy Due, vagy például egy ESP32-es modul választásánál sok kihasználatlan funkció és számítási teljesítmény maradt volna meg. Ha később szükség lenne rá, bármikor kiegészíthetjük a firmware-t, ugyanis a programot tartalmazó flash tároló sokszorosan újraírható. A firmware elkészítése ugyan többletmunkának számít, de a termék előnyei ezt kárpótolják. Mivel nem az a tesztelőeszköz fogja a tesztrendszert futtatni, így annak teljesítményére nem lesz kihatással.

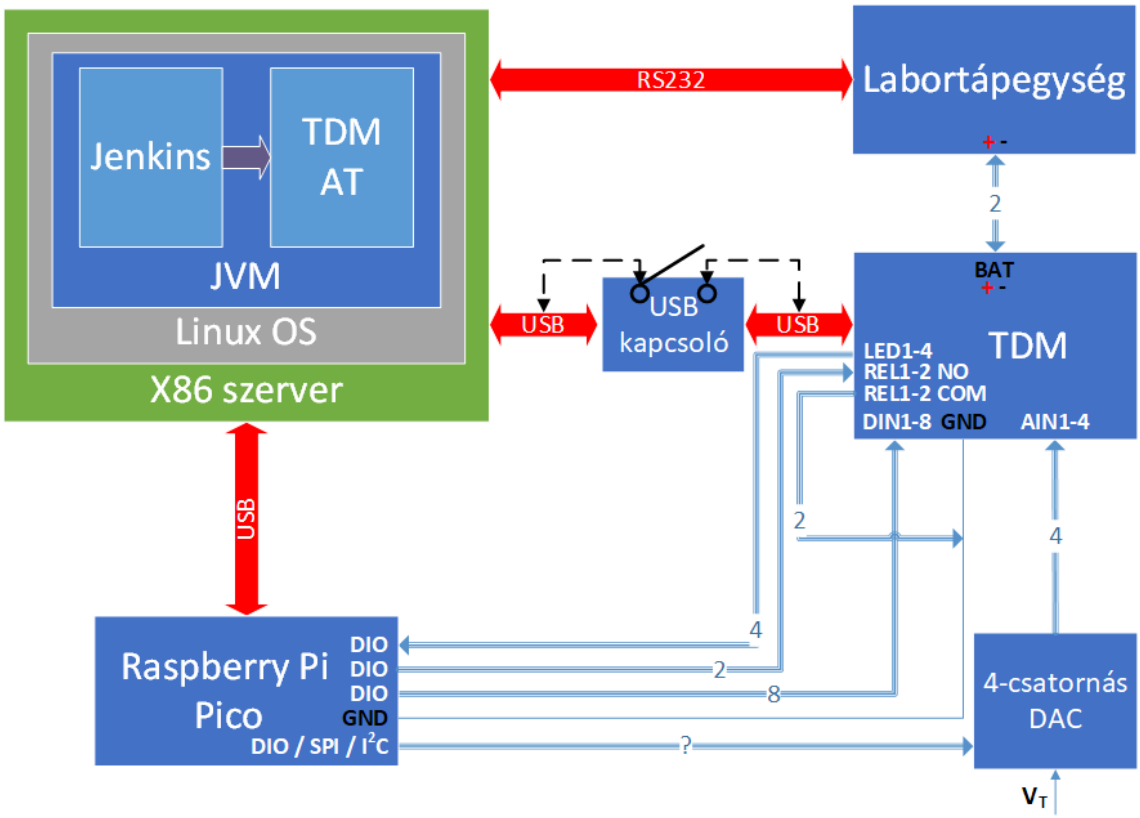


4.6. ábra: Raspberry Pi Pico MCU

5 RENDSZERTERV

5.1 Hardveres interakció terve

5.1.1 Hardveres környezet



5.1. ábra: A tesztrendszer blokkvázlata

Az 5.1-es ábrán az elkészítendő, hardveres tesztrendszer blokkvázlata tekinthető meg. A futtatókörnyezet egy x86 szerver számítógép lesz. A számítógéphez a TDM-et és az Picót is USB-n keresztül csatlakoztatni kell. A szerver szoftveres rétegződését is szemléltettem. Az egyes vastagságú nyilak tápvezetékeket, a vörös nyilak interfészeket, a számozott nyilak pedig vezetékcsoportokat jelölnek. Az MCU-kon

A kutatás óta megjelent a TDM-nek több, újabb hardver revíziója, melyekben a portok védelme fejlesztést kapott, így a kissé magasabb feszültségeket már probléma nélkül befogadják. Tesztelni már csak ezeket szükséges üzleti szempontból. Szintillesztő áramkörre így végül nem lesz szükség; a Pico portjait kimenet módba állítva azok közvetlenül összeköthetők a TDM digitális bemeneteivel.

A LED-ek állapotait a tesztelőeszköz csatlakozóit bemenet módba állítva, közvetlenül összekötve azokkal le tudjuk olvasni. A relék esetében kicsit bonyolultabb a helyzet. Itt a Pico megfelelő portjait felhúzott bemeneti állapotba fogom tenni, ami által azokon feszültségforrás jelenik meg. Ezt a relék alapból nyitott (Normally Open) kapuján átvezetve, a közös (Common) lábon keresztül a földpontba kerül az összeköttetés vége. Így ha a relé kinyit, azt meg tudjuk állapítani, ugyanis ekkor a Pico felől áram indul meg.

A digitális-analóg átalakító szerepére nem sikerült eszközt találnom a szokatlan paraméterek miatt. Egy ilyen DAC egyedi áramkört igényel, így azt meg kellene terveztetni külső partnerrel. Mivel a *nagyobb feszültségekkel való diszkrét bemenetellenőrzést a körülbelül 90 tesztesetből csupán egyetlen igényli*, az AIN bemeneteket is egyelőre digitális módon, 0-3,3 V-os jel szintjén fogom kezelni.

Az eredeti feladatkiíráshoz képest további változás, hogy az akkumulátorkezeléshez kapcsolódó hardveres tesztelési részek is megvalósításra kerülnek. Ebbe beletartozik például a szállítási módban való viselkedés, illetve a hirtelen energiaellátás megszűnéséhez kapcsolódó logika vizsgálata. Ehhez egyrészt segítségemre lesz az I4P-nél már korábban alkalmazott Ykush XS kapcsoló modul (5.2. ábra), mely egy hubként viselkedik. A TDM-et rácsatlakoztatva elvehető/visszaadható annak USB tápfeszültsége. Ezt egy vezérlőszoftverrel, a tesztrendszeren keresztül is lehetséges megtenni.

5.2. ábra: Ykush XS USB kapcsoló

Az akkumulátor szimulációját egy, szintén a cég által használt HP Agilent 66312A (5.3. ábra) típusú programozható labortápegységgel fogom megoldani. 0 és 20 volt közötti feszültséget képes szolgáltatni. A TDM számára szükséges, egycellás lítiumion akkumulátor feszültsége 3,2 – 4,2 V között ingadozik, így ez a tartomány megfelelő a feladathoz. A készüléket lehetséges az előlapon található kezelőgombokkal, illetve RS232 protokollon keresztül távolról is, automatikus módon vezérelni. Képes mérési adatokat is továbbítani, valamint kimenete is felkapcsolható, leválasztható remote módon. Legfőbb érdekessége azonban az, hogy ha kimenetére a beállított feszültségnél magasabbat kapcsolunk, képes annak negatív irányú áramát kezelni és azt eldisszipálni. Ezáltal lehetséges vele akkutöltési folyamatot imitálni, mely az ezt vizsgáló teszteknél hasznosítható[15].

[15] A TDM-re integrálva van egy töltőáramkör is

5.3. ábra: A labortápegység

5.1.2 Tesztelőeszköz

A tesztelő eszköz firmware-ét könnyedén el lehet készíteni C/C++-ban az Arduino könyvtár és a hivatalos Arduino IDE[16] segítségével. Ezek elfedik az MCU programozásához szükséges alacsony szintű részeket. Utóbbi fejlesztői környezet azonban prototípus programozásra lett kitalálva, ezért nem nyújt elegendő fejlesztést segítő funkciót, illetve limitációi vannak. [28] A későbbi kódbővítés megkönnyítése és az átláthatóság javítása céljából ezért érdemes lehet kipróbálni egy másik, Arduino-kompatibilis környezetet. Ehhez a Slober IDE-t választottam, mely a C++-os Eclipse fejlesztői környezet egy módosított változata.

A firmware-ben első körben az alapfunkciókat (kimenet átállítás, bemenet érték lekérés soros porton) fogom megvalósítani. Ezt a TDM-éhez hasonló nagybetűs szöveges üzenetek formájában célszerű, a megfelelő TDM portokat absztrakt módon megjelenítve:

- **READ,LEDx**
- **WRITE,DINx,[0/1]**
- **WRITE,AINx,[0-150]**

Korábbi tapasztalataim alapján a digitalRead() és digitalWrite() függvényeket, valamint az Arduinóhoz készült Serial és String osztályokat itt jól lehet használni. Egy jövőbeli DAC használatához lehetséges, hogy saját illesztőprogramot kell majd írni, ha egyedi, buszos megoldást választunk. Analóg lépésköznek 0,1 V elegendő lesz. A jövőbeli fejlesztésekhez célszerű implementálni azt is, hogy a TDM kimeneteinek megváltozását megszakítások is tudják jelezni.

Mivel a tesztrendszerben egymás után több teszteset is le fog futni, az előző állapotokat törölni szükséges, hogy azok ne okozzanak a következő teszt eredményében hamis kimenetelt. Ez például egy „RESET” paranccsal oldható meg, mely egy előre beállított értékre állítaná vissza az összes, TDM-mel használt csatlakozót minden teszt indulása előtt.

A hardveres teszter portjain megjelenő TDM portnevek sorban lesznek kiosztva. Míg bal oldalon a TDM bemeneteket (DIN, AIN), addig jobb oldalon a kimeneteket (LED, REL) kezeljük, hogy a bekötés leegyszerűsödjön.

[16] Integrated Development Environment

5.2 Tesztrendszer kiegészítés

A Java tesztrendszerben már implementálva van egy hardveres tesztelőeszköz interfész, mely az elemzés során említett kísérleti eszközhöz készült. A soros portos, szöveges megvalósítás miatt ez az interfész az új eszközzel is működőképes, kisebb módosításokkal. Ezenkívül még számos bővítést, átalakítást is kivitelezni kell. Ennek oka, hogy a tesztelés ezidáig teljes mértékben emulátor-orientált volt, illetve a kutatási fázis óta egy kódrefaktorálás is megtörtént. Így a teszteket fel kell készíteni hardveren való futtatásra.

Első körben a konfigurációs fájlt használó profilozási mechanizmust kell kiegészíteni. Ez azt jelenti, hogy a meglévő, „*tdmIsEmulated*” boolean konfigurációs paraméter mellé be kell illeszteni még három másikat, hogy a hardveres tesztelést modulárissá tehessük:

1. *ioTesterPresent*
2. *batTesterPresent*
3. *sensorTesterPresent*

Ezek arra szolgálnak, hogy a Concordion keretrendszer ki tudja értékelni, hogy a tesztesetek adott részeiből mit futtathat le hiba nélkül annak függvényében, hogy milyen eszközök állnak rendelkezésre. Ezen változók értékének ellenőrzését mind a belső logikában, mind a tesztspecifikációkban meg kell valósítani. Dolgozatomban a teszrendszer számára megadható három modulból az első kettőt fogom kivitelezni, ahogy korábban már részleteztem.

Az 5.1-es fejezetben leírtakhoz kapcsolódóan - mivel a TDM-nek már több hardverváltozata is van – meg kell oldani azok felismerését is, és a tesztekben kezelni kell a új és hiányzó hardvert érintő parancsokat, paramétereket. A bemutatott USB kapcsolónak, illetve labortápnak is létre kell hozni egy osztályt, és implementálni bennük az adott eszköz interfészét. Előbbinél ez - a TDM emulátorhoz hasonlóan - egy bináris fájl futtatását jelenti parancssoron keresztül, míg utóbbihoz egy harmadik soros kommunikációs osztály fog tartozni.

Meg kell oldani továbbá azt is, hogy a TDM-re újrainduláskor - mely legalább minden teszteset lefutása előtt megtörténik - ismét rá lehessen csatlakozni. A folyamat közben ugyanis az MCU lecsatlakozik a számítógépről, így az USB és a soros port is elérhetetlenné válik egy időre. Az algoritmusnak Linux operációs rendszer alatt is működnie kell, mely máshogyan viselkedik a Windows-os megoldásokhoz képest.

5.3 Automatizációs környezet terve

Ahogy már említettem, az automatizációs környezet futtató eszköze egy x86 alapú szerver számítógép lesz. A rá telepítendő operációs rendszer az Ubuntu Server 20.04 LTS. Az LTS a Long-Term Support, azaz hosszútávú támogatás rövidítése. Céges nézőpontból a standard kiadáshoz képest ez a változat biztonságosabb és jobban kiszámítható. 5 évig kap frissítéseket, mielőtt elavulttá válna. [29] Az internetelérés a telepítés időpontjában biztosított, csak az önálló futásnak szükséges működnie offline állapotban. Lehetőség szerint tükrözött, azaz RAID 1-es tömböt szeretnék alkalmazni.

Több helyen is hivatkoztam rá, hogy szervezeti kritérium folytán a Jenkins fogja az automatizációt biztosítani mint folyamatos integrációs rendszer. A Jenkins egy Javában írt,

ingyenes szoftver grafikus kezelőfelülettel. Támogatja a forráskódok ütemezett fordítását, tesztek és szkriptek futtatását. Különféle beépülőkkel (plugin) testre lehet szabni, bővíteni.

A CI rendszer szolgáltatás formájában fog futni. Meg kell oldani, hogy ezen szolgáltatás az operációs rendszerrel együtt, automatikusan elindulhasson. Legalább két adminisztrátori joggal rendelkező Jenkins felhasználót is létre kell hozni a menedzseléshez.

A Jenkins pipeline-okat („csővezeték”) alkalmaz ütemezett munkák, más néven jobok futtatásához. Ezek olyan állományok, vagy közvetlen szkriptek, melyek az adott munka részleteinek programozott leírását, sorrendjét, parancsait adják meg. Nekem is egy pipeline-t kell majd beállítanom a TDM-AT automatikus fordításához, futtatásához. Ehhez előtte hozzá kell adni a céges, Git alapú repository[17] elérhetőségét a rendszerhez, melyen a forráskód tárolódik. Mindenképp meg szeretném valósítani azt, hogy a tesztek lefutásának eredményét közvetlenül a Jenkins által generált fájlokból lehessen ellenőrizni, webböngészőből.

[30]

[17] Forrásadattár

6 MEGVALÓSÍTÁS

6.1 Interakciós környezet

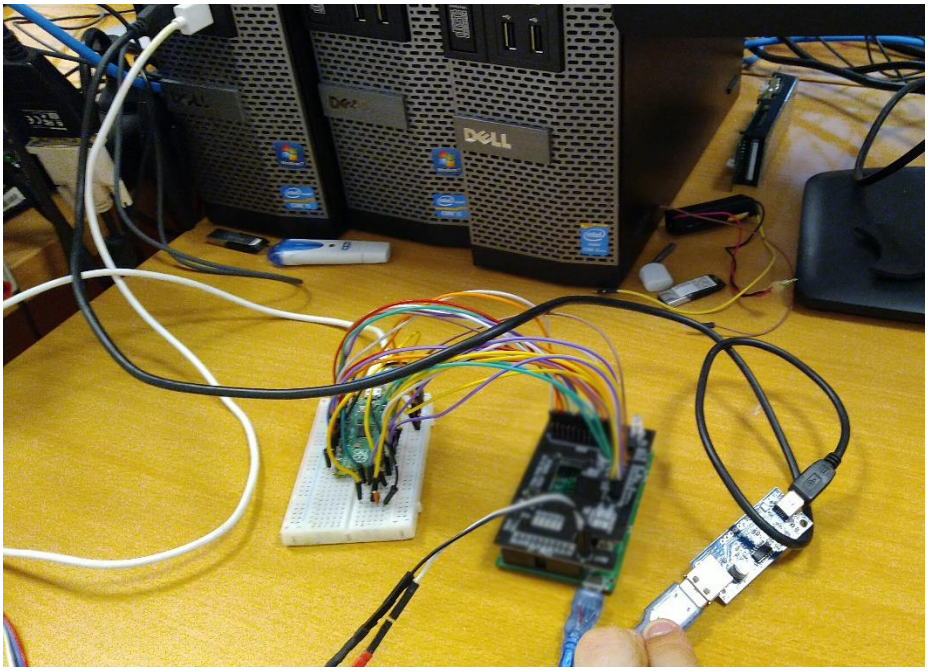
A hardveres tesztrendszer kialakítása során a rendszertervben leírtak alapján sorban haladtam. Elsőként az interakciós környezetet alakítottam ki, mely egy kisebb áramkörből, különböző eszközök összekötéséből, illetve egy firmware program megírásából állt.

6.1.1 Áramkör

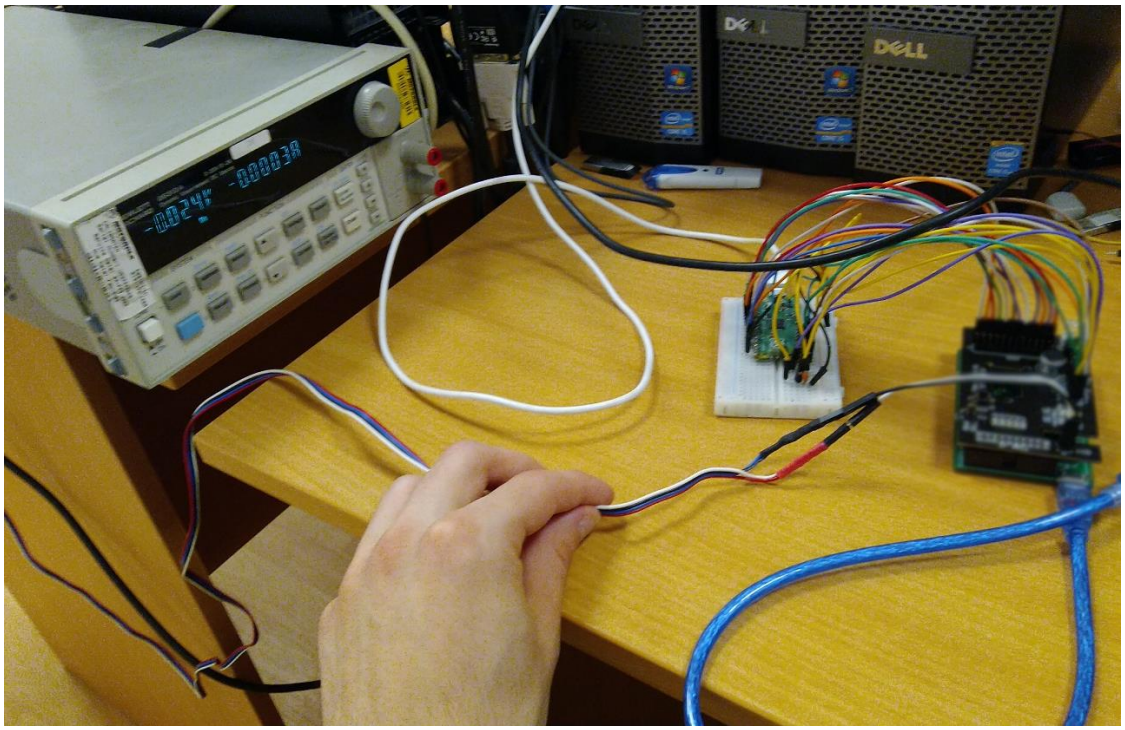
A tesztrendszerhez elektronikát - mivel a kiválaszott céleszközök már készen kaphatók – nem kellett készítenem, csupán egy kisebb, a 6.1-es ábrán látható áramkört állítottam össze. Az elemek itt még a céges számítógépemhez csatlakoznak; a fejlesztés és tesztelés ezen folyt.

A kép bal oldalán a tesztelőeszköz (Raspberry Pi Pico) található, egy fél egység méretű próbapanelben elhelyezve. A lábaiként szolgáló tüskesorokat magamnak kellett beforrasztani pákával, ugyanis a mikrokontroller szerelt változatban nem kapható. A Pico egy microUSB kábellel csatlakozik a PC-hez. A lábakhoz átkötő (jumper) kábelek csatlakoznak, melyek a rendszertervben megadott módon összekötik azokat a TDM-mel. Az analóg bemeneteket a digitálisokhoz hasonlóan, közvetlenül rákötöttem a tesztelőeszközre. Az azonos csoportba tartozó portok a Picón egy oldalra kerültek. A relékhez tartozó kábelcsoportba került két extra alkatrész is. Ennek okáról a következő, 6.1.2-es fejezetben lesz szó.

A TDM a kép jobb oldalán található, zöld NYÁK-on helyet foglaló eszköz, egy fekete színű kivezetőpanellel. Ez egy mikrokontroller világban elterjedt, úgynevezett shield, mely még korábban, a könnyebb bekötés és elrendezés végett lett megalkotva a teszteléshez. A csatlakozás módja a TDM esetében is microUSB port, mely a kék színű Ykush XS kapcsolóban végződik. Ezen eszköz pedig miniUSB segítségével ugyancsak a számítógéppel kommunikál, átengedve a TDM adatforgalmát is.



6.1. ábra: Az interakciós rendszer



6.2. ábra: Labortáp működés közben

A TDM akkumulátor bemenetének pozitív és negatív részére csatlakozik a labortápegység két vezetékével, mely a 6.2-es ábrán látható bekapcsolt állapotban. Az eszköz az elektromos energiát az épület hálózatából veszi fel. A számítógéphez egy régi típusú soros port kábellel (DB9) csatlakozik, mely a hátoldalon található.

6.1.2 Pico firmware

A firmware fejlesztése C++ nyelven történt meg, a cég saját Bitbucket repository-ában vezetve. A flashelés menete a Pico esetében hasonló az Arduinóknál megszokotthoz: USB-n keresztül történik. A program procedurális felépítésű; az Arduino keretrendszer szabályait követő módon, „sketch” formátumban készült el, azaz a fő *hwt_main.cpp* fájlban lévő ’*setup()*’ és ’*loop()*’ metódusokkal van ütemezve a működés. A közös logikához a nyelv beépített elemeit és az *Arduino.h* függvényeit használtam fel. Törekedtem a minél univerzálisabb, platformfüggetlen megoldásra. Az RP2040-es chiphez az Arduino implementációt külső fél végezte el, melyet munkám során felhasználtam. [31] A hardverspecifikus részek megírásához preprocesszor parancsokat, illetve a Pico SDK[18] komponenseit alkalmaztam. A fejlesztés során nem volt szükségem a fentieken kívül egyetlen külső könyvtárra és osztályra sem, a szoftvert kizárólag metódusokkal és struktúrákkal sikerült megalkotnom. A különböző feladatokért felelős programrészek külön forrásfájlokban foglalnak helyet, melyeket a hozzájuk tartozó fejlécfájlok kötnek össze.

[18] Software Development Kit

Fájlnév	Funkció	Platform-független?
hwt_main.cpp	Arduino belépési pont. Inicializálás megkezdése, ütemezés legfelső szintű ciklussal	igen
hwt_commands.cpp	Parancskiértékelő logika; feladata a paraméterellenőrzés, (hiba)üzenet létrehozás, parancsok futtatása	igen
hwt_interface.cpp	A bejövő üzenetek érzékelését, alapszintű ellenőrzését, pufferbe való beillesztését valósítja meg	igen
hwt_interrupts.cpp	A kimenetekről érkező megszakítások beállítását, lekezelését oldja meg	részben
hwt_io.cpp	Az áramkörökkel való interakciós algoritmusokat tartalmazza.	igen
hwt_pins.cpp	Pico-TDM pinek társítása, kezdeti állapotba állítása	részben
hwt_helper.cpp	Több rész által használt, kategorizálatlan segédfüggvényeket tartalmaz	igen
hwt_globals.h	Az egész programban elérhető globális változók deklarációi	igen
hwt_types.h	Egyedi típusdefiníciók leírása	igen
hwt_params.h	Kódbeli konfigurációs paramétereket, globális makrókat tartalmaz	részben

6.1. táblázat: A Pico firmware részei

A 6.1-es táblázatban a részegységek főbb funkciói láthatók összefoglalva. A *hwt* rövidítés a „*hardware tester”*-t takarja. A parancsokat feldolgozó részben egy pointert használó puffer található, mely az éppen aktuális parancsot tárolja. Ez azért van jelen, hogy parancs fogadását és feldolgozását a központi ütemezés szempontjából ketté lehessen választani. Ez a módszer a veremtúlcsordulás esélyét is csökkenti. Működése: az említett pufferbe a *hwt_interface* az ütemező által meghívott ’*check_incoming_msgs()*’ eljáráson keresztül beilleszti a fogadott parancsot. Ezután a vezérlés visszatér a *hwt_main*-ba, ahol következő lépésként meghívódik a *hwt_commands ’do_next_operation()*’ eljárása, mely kiolvassa a memóriából az eltárolt parancsot, majd végrehajtja.

A gyakorlati megvalósítás során a TDM formátumához hasonló utasítások lettek implementálva. A parancsok nagybetűsek, vesszővel vannak elválasztva, és új sor karakterrel zárulnak. Kiadáskor visszhangként visszaadják a beküldött paramétereket, majd a kért értékeket. Az utasításokat egy fejléc előzi meg, mely egyelőre csak a bekapcsolás óta eltelt időt adja vissza milliszekundumban, fix hosszúságban. Ennek a megszakításos üzenetek időzítésének mérésében lesz szerepe a jövőbeli teszteknél. Az extra funkció a cég kérése volt.

Az alábbi listában a fejlesztés során megvalósított parancsokat sorolom fel:

- **GETVERSION**: A három számból (minor, major, patch) álló firmware verziót adja vissza.
- **WRITE,[port],[érték]**: Adott TDM bemenet átállítása a kívánt értékre. DIN1-8, AIN1-4 állítható vele.
- **READ,[port]**: Adott TDM kimenet aktuális állapotának olvasása. LED1-4, REL1-2 olvasható vele.
- **RESET**: Visszaállítja az összes TDM-mel használt portot az induláskori állapotba.
- **INTERRUPTS,[érték]**: Ki- és bekapcsolható vele a megszakítások jelzése. Értéke 0 vagy 1 lehet.

- **GETGPIO,[port]**: Visszaadja az adott TDM porthoz beállított, fizikai Pico port számát. Megkönnyíti a tájékozódást.

A parancsok helytelen beadásakor hibaüzenetet tartalmazó válaszokat kapunk vissza a soros porton. Ezek nem tartalmaznak fejlécet, és mindig „ERROR,”-ral kezdődnek. Összesen ötféle üzenetre volt szükségem, hogy a hibaokot meg lehessen határozni. Az alábbi kifejezések lettek használva a kódban:

1. **COMMAND_MISSING**
2. **COMMAND_INVALID**
3. **PARAM_IS_NOT_NUMBER**
4. **NUMPARAM_OUT_OF_RANGE**
5. **INVALID_TDM_PORT**

Ezek közül a második többféleképpen értelmezhető: egyben jelentheti a nem létező vagy elírt parancsot, illetve a nem megfelelő paraméterszámot is.

A kimenő üzenetek a bejövő parancsoknál alkalmazott Arduino String-es megoldás helyett - mely a szövegek feldarabolásánál hatékony - karaktertömb és *sprintf()* függvény alapú, kézi allokációs megoldást használnak. Erre két okból volt szükség: egyrészt, a String nagymértékben való használata memóriatöredezettséget okozhat - mely elő is jött a fejlesztés során -, másrészt így könnyebb több szövegrészlet eltérő formátumban való összeillesztése, a „%[]” formátum specifikátorokat használva.

Az IO műveletek irányításáért felelős részek platformfüggetlenek, mert egyelőre csak az Arduino *digitalWrite()* és *digitalRead()* függvényeit használják eltérő logikákkal. Az analóg bemenetek állítása a tervben leírt diszkrét értékekkel lehetséges az interfészen keresztül, de ezen értékek digitális állapotra lesznek konvertálva a kódban: 33 alatt 0, felette 1.

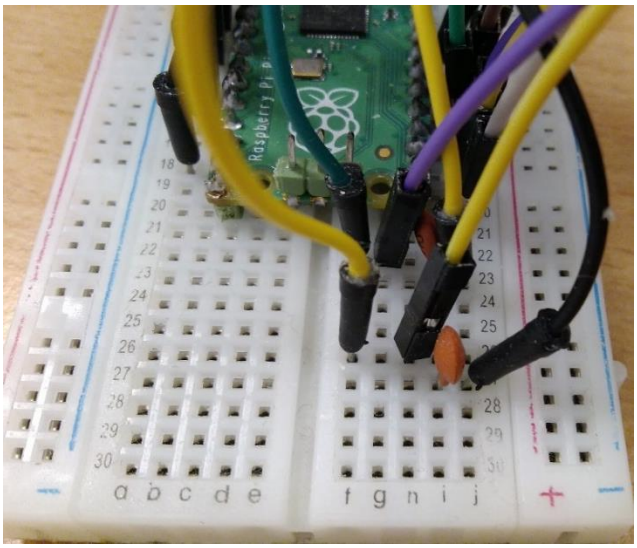
A program elinduláskor egy *pintable_t* típusú globális, szám tömböket tartalmazó szótár struktúrát hoz létre a TDM portok leképzése céljából, melyet feltölt a kódban előre definiált fizikai portszámokkal. A módszer alkalmazásának oka, hogy az asztali környezetben megszokott gyakorlattal ellentétben mikrokontrollereken nem javasolt hasítótáblákat használni azok nagy memóriaigénye miatt. [32] A DIN-eket és AIN-okat vezérlő portok a folyamat alatt OUTPUT módba kerülnek, a LED portok INPUT, a REL portok pedig INPUT_PULLUP állapotba.

Mint már említettem, a LED-ek és relék esetében megszakításkezelés is van. A funkciót a Pico SDK *hardware/gpio.h* fejlécében található ’*gpio_set_irq_enabled_with_callback()*’ eljárással valósítottam meg. Az Arduino keretrendszer szabványos megoldásával szemben ez képes paraméterben visszaadni a megszakítás forrását is. A jelek felmenő/lemenő éle során beérkezett megszakításainak pinjeit egy fix méretű interrupt pufferben tárolom el a callback függvény meghívásakor. Ezek feldolgozása az ütemező harmadik feladata, a ’*handle_interrupts()*’ metódus meghívásával. Hogy az eltárolt forrásokat a jelző üzenet számára TDM porttá alakíthassam, fel kellett vennem még egy szótárat, mely az ellenkező műveletet végzi el: egy szövegtömbből visszaadja az adott Pico pin TDM absztraktját. A megszakítások jelzése szintén az USB soros interfészen történik: INTERRUPT,LEDx/RELx formában. Az előző fejezetben felvetett két alkatrész itt került alkalmazásra. A relé ugyanis mechanikus

kapcsolónak számító alkatrész, így bezárása közben elektromos szikrák formájában pattogó-pergő hatás keletkezik. Ez néhány milliszekundumig tart mindaddig, míg az áramkör fel nem épül és stabillá nem válik. Hatása az érzékeny Pico bemeneteken ismétlődő, azonos megszakításokban jelentkezett. Ennek megoldására az alábbi két, prellmentesítő technikát vetettem be, melyek együttes alkalmazása volt szükséges a helyes működéshez:

- Elnyelő kondenzátorok beépítése
- Szoftveres szűrés használata

A hardveres megoldás esetében a teendő kerámiakondenzátorok beillesztése volt a próbapanelbe, a relék két vége közé párhuzamosan kapcsolva (6.3. ábra). Kapacitás értéknek teszteléseim alapján 1 nF lett választva.



6.3. ábra: Elnyelő kondenzátorok

Szoftveres esetben a feladat azon megszakítás üzenetek eldobása feldolgozás során a pufferből, melyek egy alkalomnál többször, egymás után következnek és azonos helyről származnak. Ezt azért szabad megtenni, mert az interrupt-kezelő rész nagyon gyorsan hívódik meg egymás után, és a puffer csak az egy időben párhuzamos, több helyről érkező események tárolására szolgál.

[33]

6.2 Tesztrendszer

A tesztelőeszköz és interakciós rendszer megvalósítása, hibajavítása után megkezdtem a TDM-AT tesztrendszer kiegészítését, hogy az képes legyen TDM hardver tesztelésére. Ez jelentett felépítésbeli módosításokat, valamint kiegészítéseket extra osztályrészek és modulok beillesztésével. A munkálatok kettéoszthatók: az egyik rész általános, hardverkezelést lehetővé tevő változtatásokból, míg a másik konkrét IO és akkumulátorkezelő részek megvalósításából állt.

6.2.1 Hardverkezelés kiegészítések

A teljes hardveres környezetet érintő egyik legfontosabb módosítás volt, hogy a lefutás folyamatát konfigurációfüggővé kellett tennem. Ez két lépésből állt.

Az első tennivaló a hardveres tesztelés modularitásának kialakítása volt. A 6.4-es ábrán a kiegészítés utáni tesztrendszer konfigurációs fájlja látható, melynek formátuma YAML. A már meglévő *tdmPort* és az új *hwtPort* változó esetében egy tetszőleges elnevezésű symlinket[19] kellett megadnom, melyekhez a fizikai eszköz társítását meg kell oldani az operációs rendszerben. A labortáp interfészét takaró *batPort* számára - mivel az valódi soros porttal rendelkezik és nem kérhető le eszközinformáció róla – a futtató számítógép első számú soros portját kell beírni. Az *usbSwitchType* lehetőséget biztosít arra, hogy más típusú USB kapcsolót állítsunk be feltéve, ha azt már implementáltuk. A tervezett moduláris paraméterek is láthatók az alsó részen, melyek automatikusan igazra állítódnak, ha a környezetet nem fizikai TDM-mel használjuk. Az új paramétereket fel kellett vennem a TDM-AT *Config* osztályába mezőként, melynek eredményeképp látszódnak környezet számára is.

```
1  # TDM test configuration values
2  tdmIsEmulated: true
3
4  #tdmPort: \\.\CNCB0
5  #tdmPort: COM6
6  tdmPort: /dev/tdm1
7
8  #### For hardware based testing only ####
9  #hwtPort: COM8
10 hwtPort: /dev/hwt1
11 batPort: /dev/ttyS0
12 #batPort: COM1
13 usbSwitchType: ykushxs
14 # These parameters are changed to true aut
15 ioTesterPresent: true
16 sensorTesterPresent: false
17 batTesterPresent: false
```

6.4. ábra: TDM-AT konfiguráció

A következő feladat a hardver revíziós szám megszerzése. Ez úgy lett megoldva, hogy a rendszer kiad egy ehhez kapcsolódó parancsot a TDM-nek a firmware verzió lekéréssel egyetemben, az első teszt lefutásánál. Az értékek statikus változókba tárolódnak el egy Maven parancs erejéig, így - a többi teszt során - már nem futnak le többször. Hozzáadásra került négy boolean függvény is, melyek segítségével a folyamatot követően megállapításokat lehet tenni:

- isEmulated(): igaz, ha a tesztelendő TDM emulálva van
- isHardware(): igaz, ha a tesztelendő TDM fizikai hardver
- isHardwareRev(szám): igaz, ha a tesztelendő TDM hardver és adott revíziójú
- isNotHardwareRev(szám): igaz, ha az előző nem

A részletezett információkat a Java logika felhasználja. Erre példát az alábbi kódkivonattal (6.5 ábra) szemléltetek, mely szintén fontos eleme a hardveres működésnek[20]. Jól látszik, hogy valamely metódusokat csak emulátor, másokat csak hardver használatakor, és van, melyeket minden alkalommal meg kell hívni. A hardveres függvények közül kiemelném a *'logic.setMode()'*, valamint a *'logic.testReset()'* megnevezésűt. Mindkettő az alaphelyzetbe való visszaálláshoz szükséges. Mivel az emulátorban a hardveres és állapottároló részek

[19] Parancsikon linuxos alternatívája. A Linux a hardvereszközöket is fájlrendszer szinten kezeli.
[20] Az initTdm() egy minden teszteset elején lefutó inicializáció

fájlokkal vannak megvalósítva, így azt clean, azaz tiszta indítással is működésre lehet bírni, míg a hardverről ez nem mondható el. A *'setMode("I")'* inicializációs módba állítja a TDM-et, mely a legkisebb interfész korlátozást rendeli el, így egy korábbi, teszt közbeni módváltás nem fog beleszólni az aktuális folyamatokba. A *'testReset()'* pedig elvégzi a fizikai TDM első indítási állapotba való visszaállítását: gyári beállításokat alkalmaz, valamint törli a logokat és státuszinformációkat. Ez csak a tesztelési folyamatnál tehető meg; az éles helyzetben használt gyártási módban a funkció véglegesen letiltásra kerül. Figyelnem kellett arra is, hogy gyártási módba a rendszer ne állítson át TDM hardvereket, mert az problémákat okoz.

```
private void initTdm() {
  if (this.tdm.isHardware() && this.env.getConfig().batTesterPresent) {
    this.tdm.setTdmSensorTo("VUSB", 5000);
  }
  this.tdm.openTdmPort(this.env.getConfig().tdmPort);
  this.tdm.clearSerialInput();
  if (this.tdm.isEmulated()) {
    this.tdm.startEmulator();
  } else {
    this.tdm.initHwtDevices();
    if (this.env.getConfig().batTesterPresent) {
      this.tdm.batterySetVoltage(3600);
      this.tdm.batteryOutputEnable(true);
    }
    this.logic.setMode("I");
    this.tdm.wait(500);
    this.logic.testReset();
  }

  this.logic.turnOffHeartbeat();
  this.logic.setKey(this.helper.getRandomHexString(8));
  this.tdm.clearSerialInput();
}
```

6.5. ábra: Az initTdm() függvény, profilozással

Az előzőleg részletezett profilozási technikát a specifikációkban is alkalmaztam, több szinten. Volt, hogy a teljes viselkedési tesztrészre kellett szűrés, de adott egységeket, különálló parancsokat is filterezni kellett. A TDM-AT átalakítással kapcsolatos munka idejének nagy része ezzel telt el. Az ellenőrzéseket egy Concordion kiterjesztéssel, az ExecuteOnlyIf-fel kiviteleztem. Ennek segítségével a HTML-ekben ' ' formájú címkék közé beillesztett Concordion parancsokat lehetséges a visszatérési értéktől függően, feltételesen feldolgozni. A 6.6-os ábrán a minden tesztspecifikáció elején megjelenő, környezet összefoglaló táblázat látható az újonnan beépített értékekkel, mely a teszt körülményeiről tudósítja a felhasználót.

Environment for this test

Is emulator	false
IO tester was present	true
Sensor tester was present	false
Battery tester was present	true
Serial port	/dev/tdm1
OS	linux (ubuntu)
TDM Firmware version	[redacted]
TDM Hardware revision	[redacted]

6.6. ábra: Környezeti összefoglaló

Utolsóként pedig a soros portra való újracsatlakozás módszerét alkottam meg. Ehhez a *SerialDevice* osztályt kellett módosítanom, melyet a TDM is használ kommunikációra.

Lényege, hogy amikor a TDM elérhetetlenné válik az újraindulás alatt, a port megnyitása NullPointerException-t vált ki, és ezt elkapva teszünk további lépéseket. Ilyenkor egy beállított, maximum számban próbálkozunk a port bezárásával, majd újranyitásával mindaddig, míg vissza nem jön a kapcsolat, szüneteket közbeiktatva. A tervben említettem azt, hogy Linuxon egy kicsit eltérően kell kezelni ezt az esetet. Ez olyan formában nyilvánult meg, hogy a megnyitásba egy plusz lépés került, mely a soros port paramétereit újra beleírja az SerialPort objektumba, ha a rendszer Linux operációs rendszert érzékel. Ezáltal ha az ugyanazon symlinkhez csatolt valódi port neve megváltozik, a tesztrendszer azután is a megfelelő eszközre fog csatlakozni.

6.2.2 IO és tápellátás részek

A konkrét funkciót ellátó részek közül elsőként az IO kezelést végző hardveres tesztelőeszköz osztályát készítettem el. Ebben az osztályban a 6.1.2-es alfejezetben ismertetett parancsok vannak Java függvénnyé alakítva. A hardveres tesztelőeszköz *SerialDevice* objektumát tartalmazza, aggregációval. A labortápegységet kezelő osztály is hasonlóan lett kialakítva, a soros port és a logika közti adapterként fogható fel. Az új elemek a *TdmResource* facade osztályba kerültek beágyazásra, mely összefogja a TDM-mel kapcsolatos közvetlen műveleteket, és azokhoz egy közös absztrakciós réteget biztosít a specifikációk számára, konfigurációtól függetlenül. Erre jó példa a *'setTdmSensorTo()'* IO portok és szenzorok állapotait beállító függvény, amely emulátor esetében egy alkalmazás parancsot hív meg, míg hardver használatakor a bevezetett adapteren keresztül soros üzenet küldését indítja el.

Az IO vezérlés helyes üzemeléséhez a fentieken kívül még egyetlen változtatást kellett eszközölni, mégpedig a fél másodperc várakozási idő bevezetését a digitális bemenetek állapotváltozásainál. Ennek hiánya emulációnál nem okozott gondot, de a hardvernél igen. Az operációs rendszer szintjén működő, milli-nanoszekundum nagyságrendű impulzusokkal szemben ugyanis az interakciós rendszer képtelen volt lépést tartani.

Az akkumulátoros üzemmód szimulálásához a tápegység adapter class mellett az USB feszültség leválasztást végrehajtó alrendszert is létre kellett hoznom. Ehhez letöltöttem az internetről a kapcsolópanelhez tartozó *ykushcmd* parancssoros applikációt. Windows platformra a futtatható fájl már előre el volt készítve, csupán be kellett másolnom a TDM-AT forrásadattárába. Linuxon való felhasználásnál azonban a forráskódból való lefordítás feltétel. A gyártó ehhez is mindent mellékelt, így egy *libusb-1.0-0-dev* függőség, valamint a *g++* és a *make* telepítése után sikeresen meg is történt. A program kezelése egyszerű, az átbillentést az alábbi módon lehet megtenni:

- `sudo ykushcmd ykushxs [-d/-u]`

A ”ykushxs” paraméter a Ykush eszköz típusa, a második paraméter pedig a kívánt állapot. A -d (down) kapcsolóval tudjuk leválasztani, míg a -u (up) kapcsolóval visszaadni a felfűzött eszköz tápellátását. A tesztrendszerben ez az *USBSwitch* osztályban kapott helyet. Feladata, hogy elindítson egy ilyen processzt az operációs rendszer típusának figyelembevételével, a megfelelő mappából végrehajtható fájlt választva.

6.3 Integrációs rendszer

A megvalósítás utolsó lépése az integrációs rendszert magába foglaló szerver számítógép létrehozása, beüzemelése volt. Ez is két fázisból tevődött össze: az operációs rendszer és a használandó programok telepítéséből, illetve a Jenkins konfigurálásából.

6.3.1 Telepítés

A kiválasztott platform egy régebbi generációs Dell mikroszerver számítógép lett. Processzora egy négymagos, Intel Xeon CPU Ivy Bridge alapokon, mely 8 GB DDR3 rendszermemóriával gazdálkodhat. Háttértárként merevlemez meghajtók állnak rendelkezésre BIOS által kezelt szoftveres, RAID1-es konfigurációban, 2 x 1 TB méretben. Egy pendrive-ról került telepítésre az operációs rendszer, az Ubuntu Server 20.04 LTS. Egy helyi adminisztrátori fiók létrehozásra került, valamint az SSH szerver is aktiválva lett a távoli, offline klienssel való elérés lehetősége miatt.

A programok közül legelsőnek a Java futtatási és fejlesztői környezetet installáltam fel, az *apt* csomagkezelőn keresztül:

- `sudo apt install openjdk-8-jdk`

Egy régebbi, 8-as verzió került letöltésre, amely 2014-ben jelent meg, és egy hosszabban támogatott kiadás. A cég biztonsági és kompatibilitási okok miatt használja ezt a változatot. A tesztrendszer is erre lett megírva; ennél újabb verzión futtatva jelenleg nem működik. A *'java -Xmx6G -Xms512M*' paranccsal beállítottam a virtuális gépnek a felhasználható maximális, illetve minimális memória méretét.

Ezt követte a Maven fordítási segédprogram telepítése, melyből a legfrissebb változat került fel a rendszerre:

- `sudo apt install maven`

Ahhoz, hogy az összes szükséges csomagot és függőséget el tudjuk érni a TDM-AT lefordítása során, a Maven '*/home/[felhasználó]/.m2*' könyvtárában található *settings.xml* fájljába bele kellett írnom az offline Nexus artifact szerver elérhetőségét, valamint az abba való bejelentkezéshez szükséges autentikációs adatokat. A Nexus egy olyan CI eszköz, mely egységes csomagletöltést/feltöltést és tárolást biztosít különböző felhasználói felületeken, API-kon keresztül. A harmadik féltől származó, valamint házon belüli fejlesztésű könyvtárak is ide kerülnek fel.

Továbbhaladva, hogy az interakciós rendszer eszközeit megfelelően el tudjuk érni, operációs rendszer szintű szabályokat kellett megfogalmaznom az *udev* Linux-kernel modul számára. Ezt úgy lehet elérni, hogy a '*/etc/udev/rules.d'* mappába létre kell hozni egy egyedi fájlt, mely esetünkben a *90-tdm-usb.rules* nevet kapta. Ebbe a fájlba különböző egysoros parancsokat, interakciós szkripteket lehetséges írni. A hardveres tesztelésben részt vevő eszközök közül magának a fizikai TDM-nek és a Pico-nak keletkezett szabálya; a következő cellákban ezek tartalma látható.

ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idVendor}=="ffff", ATTRS{idProduct}=="ffff", GROUP="jenkins", MODE:="0666", SYMLINK+="tdm1"
ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idVendor}=="2e8a", ATTRS{idProduct}=="000a", GROUP="jenkins", MODE:="0666", SYMLINK+="hwt1"

6.2. táblázat: Udev szabályok

A két szabály formailag megegyezik, és hasonlóra utasítják az eszközkezelőt. Az eszközök csatlakoztatása után, ha azok az USB alrendszerhez tartoznak, és a megadott gyártó-, valamint termékazonosító kóddal rendelkeznek, hozzon létre nekik egy symlinket a tesztrendszer konfigurációjában megadott neveken. Ez ki van egészítve azzal, hogy a fájlrendszerben a „jenkins” csoport tagjai legyenek, és hozzáférési számuk a „0666” legyen, azaz a tulajdonos, annak csoportja és minden más csoport is olvashassa, írhassa az eszközöket. Erre azért volt szükség, mert alapértelmezetten minden hardver a root felhasználó fennhatósága alá kerül. A kapcsolónak és a tápegységnek az előző fejezetekben már említett okok miatt nem kellett udev bejegyezés. Azonban, hogy a CI rendszernek elegendő jogosultága legyen a számítógép fizikai soros portjának eléréséhez, fel kellett venni a felhasználóját két csoportba:

- `sudo usermod -a -G tty jenkins`
- `sudo usermod -a -G dialout jenkins`

Végül pedig a Jenkins feltelepítése következett. A kiindulási Ubuntu repository-ban a szoftver nincsen benne, így hozzá kellett adnom a csomagkezelőhöz egy extra forrást is, melynek módjáról a Jenkins dokumentáció adott felvilágosítást:

- `wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add –`
- `sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'`

Az első parancs letölti a konzol pipeline-ba a szerver verifikációhoz szükséges publikus kulcsot, majd hozzáadja az *apt* csomagkezelőhöz. A második parancs pedig létrehoz egy apt forrásfájlt, ahová beírja a szerver elérhetőségét. Miután lefutottak, a telepítést el lehet végezni:

- `sudo apt update`
- `sudo apt install jenkins`

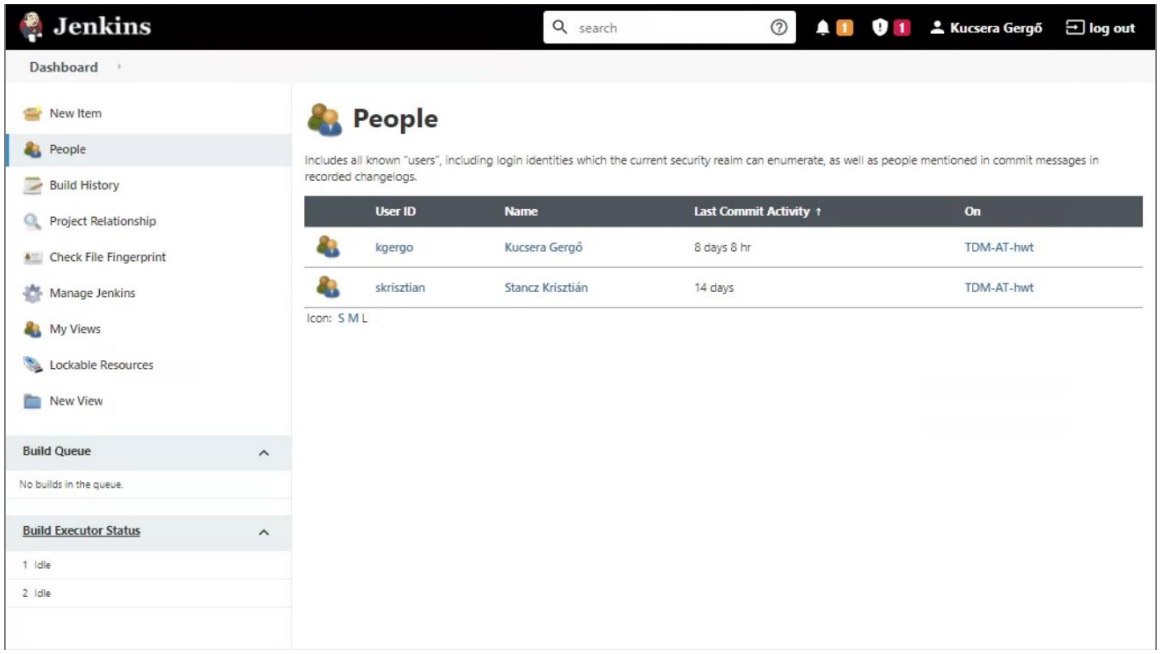
Miután a telepítés bejeződött, a rendszerben megjelenik szolgáltatásként a szoftver, melyet azonnal el is tudtam indítani:

- `sudo systemctl start jenkins`

Ekkor a ’*http://[IP]:8080*’ címen elérhetővé vált a webes kezelőfelület, mely megkezdi az első indítási folyamatot.

6.3.2 Jenkins konfigurálása

A telepítés végeztével a távoli menedzsment opció elérhetővé vált, és a továbbiakban ennek segítségével zajlott a beállítás. A Jenkins első indítási folyamatának megkezdéséhez szükséges, hogy beírjuk a kezdeti rendszergazda jelszót a fájlrendszerből. Ezt a '*/var/lib/jenkins/secrets/initialAdminPassword*' fájl kiíratásával tehetjük meg; a kapott kódot bemásolva a kért szövegdobozba. Ezt követően az extra beépülők hozzáadása zajlott le, melyek közül a Locale, a Timestamper, valamint az SSH Agent került kiválasztásra. Az első adminisztrátor felhasználó is ekkor jön létre, melyként magamat regisztráltam. A folyamat befejezését követően megjelenik a webes vezérlésért felelős Dashboard. Első dolgom volt, hogy megváltoztassam az alapértelmezett jelszót egy sajátra, majd egy második fiókot is beállítottam a külső konzulensem számára (6.7 ábra).



6.7. ábra: Jenkins Dashboard, People az angol nyelvre állítás után

Észrevettem, hogy a Jenkinsnek nem teljes a magyar fordítása, ezért félig magyar-félig angol szövegek jelennek meg default konfigurációban. Ez nem túl kényelmes, így a Locale plugint használva, a *Manage Jenkins → Configure System → Locale* menüpontban megadtam a használni kívánt nyelv kódját, ami esetemben az angol: *en_US*. Az „*Ignore browser preference and force this language to all users*" jelölőnégyzetet is bepipáltam, ugyanis anélkül a webböngészők nyelvét venné alapul a szerver, tehát esetünkben ugyanúgy viselkedne.

Folytatva a beállítást, fel kellett vennem az I4P Bitbucket szerverhez tartozó SSH privát kulcsot mint hitelesítő adatot[21], globálisan használható erőforrásként . Ezt a *Manage Jenkins → Manage Credentials → Jenkins → Global credentials (unrestricted) → Add Credentials* menüpontban tettem meg. Felhasználónévnek egy tetszőleges szöveget adtam meg, a Bitbucket szempontjából ennek nem számít a tartalma.

[21] Enélkül nem lehet hozzáférni a forráskódokhoz

Ezek után a *New item → Pipeline* gombra kattintva létrehoztam a folyamatos integrációs csővezetéket, melynek konzekvensen a „TDM-AT-hwt” nevet adtam. Három, grafikus felületen elvégezhető beállítást kapott:

- 1000 fordítás után a régi állapotokat dobja el a rendszer fokozatosan
- Ne engedjen egyidejűleg több munkamenetet futni párhuzamosan
- Periodikusan kezdje meg a munkát, minden reggel 7:00-kor, terheléselosztással (H)

Az első szabály célja a felesleges helyfoglalás megelőzése. A második egyértelmű hibamegelőzés; a sorosan kommunikáló hardvereszközöket egyszerre több folyamat nem használhatja. A harmadik pedig a folyamat automatizálását valósítja meg, ütemezéssel.

A befejező és egyben legfontosabb beállítás a pipeline szkript elkészítése volt, mely meghatározza a folyamaton belüli tevékenységsorozatot. A két elkészítési lehetőség közül én a Jenkinsen belüli, közvetlen szkript kreációját választottam. Ennek mikéntjéről szintén az online Jenkins dokumentáció adott felvilágosítást. Több szintre, úgynevezett stage-ekre bontottam fel az összefüggő részeket:

```
pipeline {
  agent {
    label 'ubuntu-controller'
  }
```

```
stages {
    stage('Repo checkout') {
        steps {
            checkout([$class: 'GitSCM', branches: [[name: 'master']],
extensions: [[$class: 'CloneOption', shallow: true, depth: 1, timeout:
20]], userRemoteConfigs: [[url: 'git@bitbucket.i2p.in:tdm//tdm-at.git',
credentialsId:'bitbucket-tdm']]])
      }
    }
```

```
   stage('Shell commands') {
        steps {
            sh "chmod 755 ${env.WORKSPACE}/ykushcmd/ubuntu/ykushcmd"
            sh "sed -i -e 's/tdmIsEmulated: .*/tdmIsEmulated: false/'
${env.WORKSPACE}/src/test/resources/config/config.yml"
        }
    }
```

```
   stage('Build & Test') {
        steps {
        catchError {
            sh 'mvn clean install -B -U -
Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true'
        }
      }
    }
```

```
   stage('Report') {
        steps {
            publishHTML(target: [
            allowMissing: false,
            alwaysLinkToLastBuild: false,
            keepAll: true,
            reportDir: 'target/concordion',
            reportFiles: 'MainIndexTest.html',
```

```
                reportName: 'Concordion Report'
                ])
            }
        }
        stage('Results') {
            steps {
                archiveArtifacts artifacts: 'target/concordion/**/*.*'
            }
        }
    }
}
```

6.3. táblázat: Pipeline szkript

Munkamenetet futtató ügynöknek jelen szervert választottam, mely „ubuntu-controller” cimkével szerepel a nyilvántartásban. Különálló csomópont számítógépek (node-ok) nem érhetők el, így az irányítás mellett ez a szerep is áthárult rá. A *Repo checkout* során letöltésre kerül az offline Bitbucket repository *master* ága, melyen a tesztrendszer forrásfájljai találhatók. Mindez csak a legutóbbi verzió klónozásával, hogy helyet és időt spóroljunk vele. Az előzőleg rögzített SSH kulcsra itt lesz szükség, melyet a *credentialsId* paraméterben adtam meg. Miután megtörtént a klónozás, folytatjuk a *Shell commands* résszel. A munkaterület mappáján belül a rendszer megkeresi a linuxos ykushcmd bináris fájlt, és hozzáférési jogosultágot ad neki. Ezt követően a helyi TDM-AT konfigurációs fájl módosítása történik meg egy *sed* paranccsal, melynek során átállítjuk emulátoros módból hardveres módba a profilt. Erre azért volt szükség, mert a már meglévő emulátoros tesztek egy másik szerveren is ugyanezt a branchet használják, megegyező tartalommal. A *Build & Test* részen a Maven-nel történő Java kódfordítás és tesztfuttatás parancs kiadása folyik. A legtöbb időt ezzel tölt el a rendszer (kb. 30 perc). A *clean* kulccsszóval biztosítva van az előző folyamatból származó maradvány állományok törlése; a *-B* paraméter bekapcsolja az automata (no input) módot, a *-U* pedig kikényszeríti minden alkalommal a függőségfrissítések keresését. A két hosszú, „Dmaven” kezdetű paraméter pedig kikapcsolja a TLS tanúsítvány hitelességvizsgálatát, intranetes https kapcsolat okán. A tesztek lefutása után a *Report* és *Results* rész archiválja az elkészült HTML riportokat, és megjeleníti azokat az adott build weboldalán.

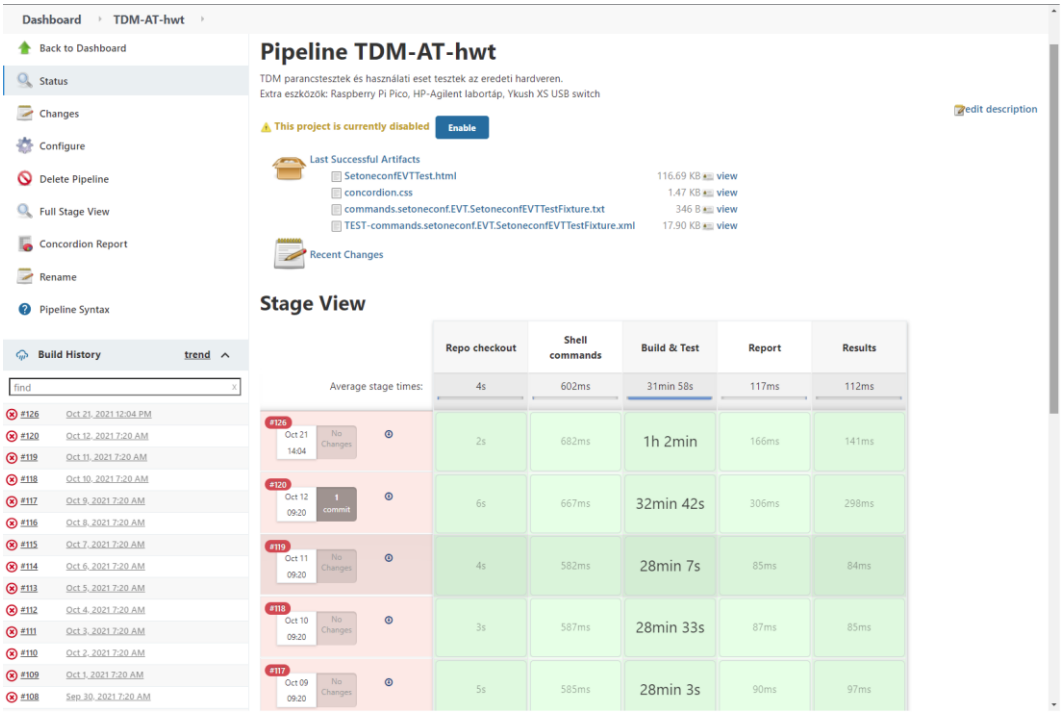
A fentebb leírtakkal befejeződött folyamatos integrációs rendszer konfigurálása a tervek alapján haladva, az igényeknek megfelelően. Az első próbák alatt azonban nem az elvárt módon teljesített a rendszer. Nem működött az USB kapcsoló, illetve a tesztjelentések formázása sem történt meg. A megjelent problémák orvosolására két, interneten talált procedúrát végeztem el.

Az USB kapcsolós probléma oka az volt, hogy a ykushcmd-nek root joggal kellett futnia a hardver eléréséhez. A *sudo* előtag ehhez nem volt elég, mert a konzolon az kérte az aktuális felhasználó jelszavát. Úgy oldottam meg, hogy a *’/etc/sudoers’* rendszerfájlhoz hozzáadtam a jenkins felhasználót, mint admin: „jenkins ALL=(ALL) NOPASSWD: ALL”. [34]

A Concordion HTML-ek formázása pedig azért nem történt meg, mert a Jenkins biztonsági politikája alapból nem engedi meg a CSS fájlok parszolását a Dashboard felületén. Ennek megoldása egy startup szkript segítségével történt. A *’${JENKINS_HOME}/init.groovy.d’* könyvtárban létrehoztam egy *startup-properties.groovy* fájlt, melybe a „System.setProperty(”hudson.model.DirectoryBrowserSupport.CSP”,””)” parancsot írtam bele. [35] Ez engedélyezi a CSS-t a Jenkins szolgáltatás felállásakor, mely által már minden

szín és elrendezés jól fog megjelenni, amikor teszteredményeket vizsgálunk. Természetesen ezt is hozzá kellett rendelni a Jenkins felhasználójához, hogy hozzáférhessen; *chown* és *chgrp* shell parancsokkal.

Offline IP-címet is kapott a belső hálózatból a rendszer, de DNS nevet egyelőre nem, mivel annak beállítása hatókörömön kívül esik. A TDM-AT pipeline kezdőlapjának jelenlegi állapota a 6.8-as ábrán figyelhető meg, mely statisztikai adatokat közöl.



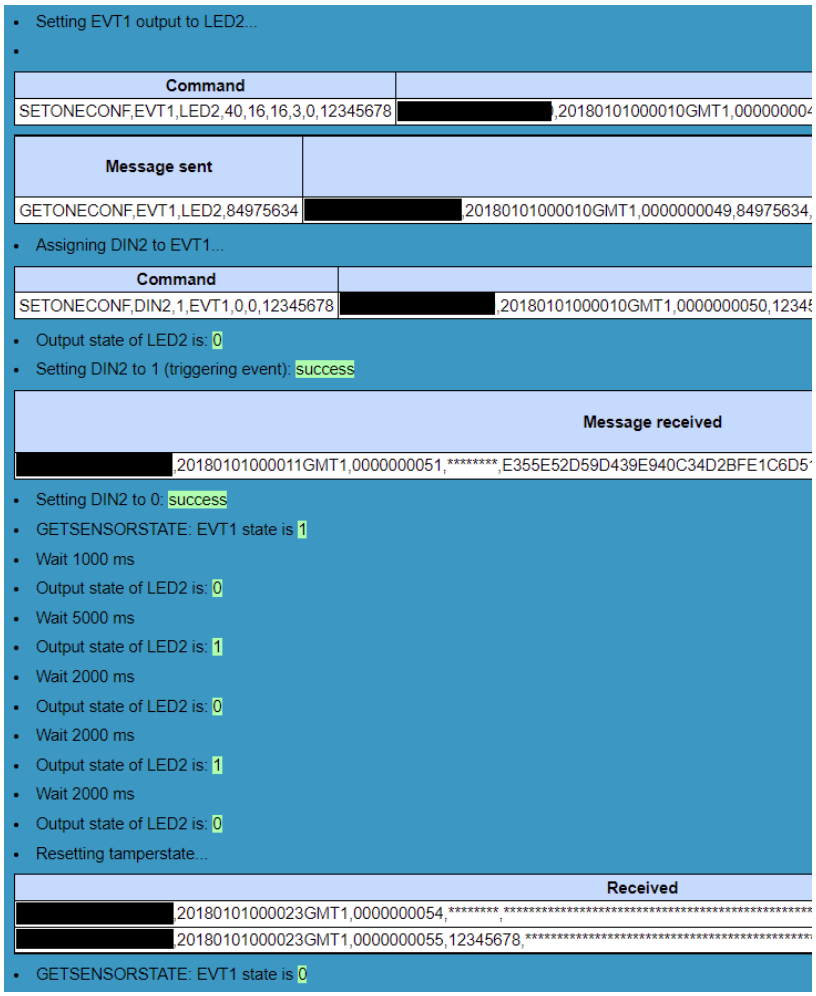
6.8. ábra: TDM-AT-hwt csővezeték kezdőlapja

7 TESZTEK

A dolgozat során elkészített rendszer működését valós, napi szinten lefutó teszteset példákkal fogom prezentálni. A rendelkezésre álló, rengeteg parancsteszt közül választottam ki néhányat, amelyek bemutatnak: egy hardveres tesztelőeszközzel való IO tesztelési folyamatot, egy akkumulátor töltés szimulációt, valamint egy akkumulátorról való üzemelést. Minden esetben a viselkedési tesztrészek vannak középpontban, az USB interfészen kívül eső változásokat ugyanis ezek elemzik. A folyamatok megértésében a Jenkins által megjelenített, parancssoros debug üzenetek is segítségünkre lesznek, mellyel nyomonkövethetjük az összes elküldött/fogadott üzenetet. Ezekben a Concordion teszt szempontjából fontos pontokat félkövér betűkkel jelöltem. Az üzenetekben a tesztelt TDM hardver sorozatszáma kitakarásra került; céges előírások miatt.

7.1 Be- és kimenetvezérlés

A hardveren lévő bemeneteket és kimeneteket is vizsgáló tesztek közül a SETONECONF parancskomplexum EVT alparancsát választottam példaként. Az adott bemenetekre aktiválódó események (3.3 fejezet) kimenetekre gyakorolt hatását állíthatjuk be vele. A tesztben a Pico hardveres tesztelőeszköz végzi ezen események kiváltását, majd időközönként a kimeneti állapotok leolvasását. A TDM által nyilvántartott állapotokat a GETSENSORSTATE parancs által visszaadott sztringgel szerezzük be, melyben különböző számértékek és flagek is vannak.



7.1. ábra: SETONECONF [EVT] teszt riport részlete

12:29:00:155 DEBUG SerialDevice - >>**tdm SETONECONF,EVT1,LED2,40,16,16,3**,0,12345678

12:29:00:182 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000010GMT1,0000000048,12345678,4561931A82A1C70CAC9E79A444428BC57C751CD541B3C0377B9689CD21BA43CA,SETONECONF,EVT1,LED2,00040,00016,00016,00003,0

12:29:00:182 DEBUG SerialDevice - >>tdm GETONECONF,EVT1,LED2,84975634

12:29:00:195 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000010GMT1,0000000049,84975634,1DB9090750FC02A2DF016AA9E20C483C9FCEFDB3361A37CCFE2CB5B3E5F4AD8D,GETONECONF,EVT1,LED2,00040,00016,00016,00003,0

12:29:00:198 DEBUG SerialDevice - >>**tdm SETONECONF,DIN2,1,EVT1**,0,0,12345678

12:29:00:224 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000010GMT1,0000000050,12345678,C45C29AC96C0C6CE674950C1209987481DF743DA8984ED56582C2A9B3D803E6F,SETONECONF,DIN2,1,EVT1,0,0

12:29:00:224 DEBUG TdmResource - [Waiting for 500 ms]

12:29:00:725 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:00:728 DEBUG SerialDevice - <<**hwt 0001869309,READ,LED2,0**

12:29:00:728 DEBUG SerialDevice - >>**hwt WRITE,DIN2,1**

12:29:00:731 DEBUG SerialDevice - <<hwt 0001869312,WRITE,DIN2,1

12:29:00:741 DEBUG SerialDevice - <<**tdm** XXXXXXXXXXXXXXXX,20180101000011GMT1,0000000051,********,E355E52D59D439E940C34D2BFE1C6D519D1372CA1076C843E34E0095061B9631,**INTERRUPT,DIN2,1,**+

12:29:00:797 DEBUG SerialDevice - <<**tdm** XXXXXXXXXXXXXXXX,20180101000011GMT1,0000000052,********,**********,**HEARTBEAT,EVT1,1**,EVT2,0,EVT3,0,EVT4,1,VTDM,04997,0,VUSB,04997,*,VBAT,03609,+,AIN1 […]

12:29:00:797 DEBUG SerialDevice - >>hwt WRITE,DIN2,0

12:29:00:800 DEBUG SerialDevice - <<hwt 0001869382,WRITE,DIN2,0

12:29:00:800 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,**12345678

12:29:00:845 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000011GMT1,0000000053,12345678,**********,**GETSENSORSTATE,EVT1,1**,EVT2,0,EVT3,0,EVT4,1,VTDM,04943,0,VUSB,04943,*,VBAT,03604,+, […]

12:29:00:845 DEBUG TdmResource - **[Waiting for 1000 ms]**

12:29:01:846 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:01:847 DEBUG SerialDevice - <<**hwt 0001870429,READ,LED2,0**

12:29:01:847 DEBUG TdmResource - **[Waiting for 5000 ms]**

12:29:06:848 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:06:851 DEBUG SerialDevice - <<**hwt 0001875432,READ,LED2,1**

12:29:06:851 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:29:08:852 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:08:856 DEBUG SerialDevice - <<**hwt 0001877436,READ,LED2,0**

12:29:08:856 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:29:10:857 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:10:861 DEBUG SerialDevice - <<**hwt 0001879441,READ,LED2,1**

12:29:10:861 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:29:12:862 DEBUG SerialDevice - >>**hwt READ,LED2**

12:29:12:863 DEBUG SerialDevice - <<**hwt 0001881445,READ,LED2,0**

12:29:12:863 DEBUG SerialDevice - >>**tdm RESETTAMPERSTATE,12345678**

12:29:13:171 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000023GMT1,0000000054,********,**********,UPDATE,SECURE_DATA_STORE_INITIALISED

12:29:13:193 DEBUG SerialDevice - <<tdm XXXXXXXXXXXXXXXX,20180101000023GMT1,0000000055,12345678,**********,**RESETTAMPERSTATE,OK**

12:29:13:193 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,**12345678

12:29:13:238 DEBUG SerialDevice - <<**tdm** XXXXXXXXXXXXXXXX,20180101000023GMT1,0000000056,12345678,**********,**GETSENSORSTATE,EVT1,0**,EVT2,0,EVT3,0,EVT4,1,VTDM,05013,0,VUSB,05013,*,VBAT,03604,+ [...]

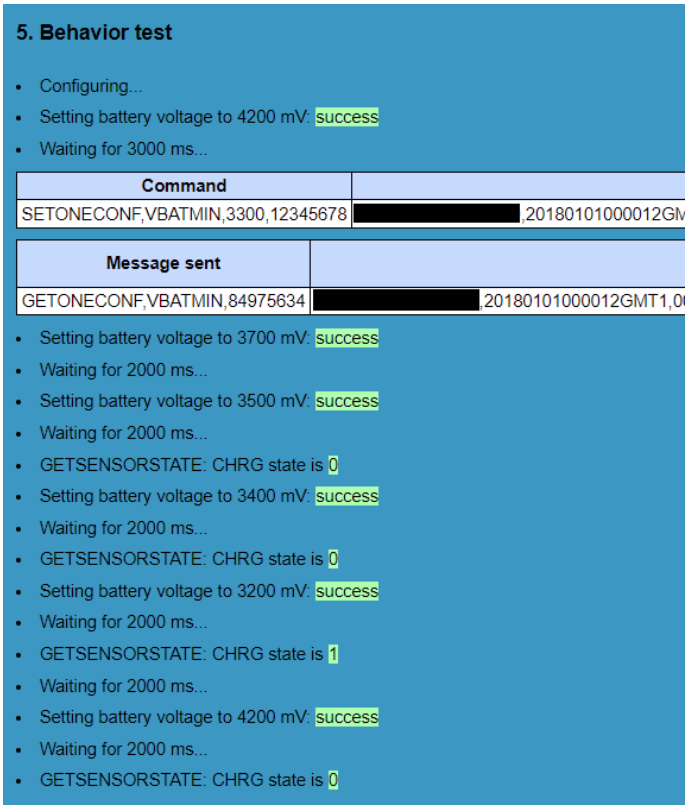
7.2. ábra SETONECONF [EVT] teszt parancssori kimenetének részlete

A viselkedés tesztrész során (7.1, 7.2 ábra) egy behatolási esemény indikálta LED villogást ellenőrzünk. Elsőként beállítjuk a LED2 kimenet eseményre való reakcióját a „SETONECONF,EVT1,LED2[…]” paranccsal: 5 mp. várakozási idő, 2 mp. aktivitás, 2 mp. szünet, 3-szori ismétléssel. Ezután hozzárendeljük a DIN2-es bemenetet a tampereseményhez a „SETONECONF,DIN2,EVT1[…]” paranccsal. A LED2 kezdeti állapota ekkor 0 kell, hogy legyen. Előidézzük a behatolási eseményt a teszterrel (logikai 1-es feszültséget kapcsolunk DIN2-re), amire a TDM egy INTERRUPT és HEARTBEAT

üzenettel reagál. Az EVT1-es esemény ekkor 1-re vált. Ezt követően a beállított időtartamokig várunk, majd leolvassuk a LED2 TDM port állapotait. Ahogy látszik, a digitális eredmények helyesek; a TDM megfelelően működik ebben a helyzetben. A mérés befejeztével megszűntetjük a behatolási állapotot a RESETTAMPERSTATE utasítás kiadásával, mely által az EVT1 tamperesemény 0-ra visszaáll.

7.2 Akkumulátor töltés

A hardverrel történő akkumulátor feszültség manipuláció egy lemerülés-feltöltés szimulációval, a SETONECONF [VBATMIN] tesztben szemléltethető látványosan. Az utasítással megadhatjuk azt a minimális akkufeszültséget, amely alatt a töltésnek meg kell indulnia. Az akkumulátor szerepét a rendszertervben említett labortápegység játssza. A kapcsoló hub mindig felkapcsolt állapotban van, a töltés az USB tápfeszültség által biztosított.



7.3. ábra: SETONECONF [VBATMIN] teszt riport részlete

A 7.3-as és 7.4-es ábrán látható viselkedési tesztrészben végigkövethető a töltési folyamat. Az „akkumulátor” kezdetben teljesen feltöltött állapotban van 4200 mV feszültséggel. Beállítjuk a minimális feszültséget 3300 mV-ra a „SETONECONF,VBATMIN,3300[…]” paranccsal. Kis léptékben haladva csökkentjük a töltöttségi szintet, két másodperces mérési és feldolgozási időt hagyva az MCU logikának. Közben ellenőrízzük a TDM állapotok közül a CHRG töltést jelző flaget, melynek ezidáig 0-nak kell lennie. Amint átlépjük a határt 3200 mV-nál, a töltés megindul, és a CHRG 1-re vált át. Amikor visszaállítjuk a kezdeti állapotot (100% töltöttség), a töltés megáll, és a CHRG is 0-ra billen vissza.

12:50:20:685 DEBUG SerialDevice - >>**bat VOLT 4.20**

12:50:20:685 DEBUG TdmResource - **[Waiting for 3000 ms]**

12:50:23:685 DEBUG SerialDevice - >>**tdm SETONECONF,VBATMIN,3300,12345678**

12:50:23:711 DEBUG SerialDevice - <<tdm
XXXXXXXXXXXXXXXXX,20180101000012GMT1,0000000011,12345678,2A2D17926C718775B3745B23D56F52589B497BD17F0D12C7E566498343E5D3B2,**SETONECONF,VBATMIN,03300**

12:50:23:711 DEBUG SerialDevice - >>tdm GETONECONF,VBATMIN,84975634

12:50:23:724 DEBUG SerialDevice - <<tdm
XXXXXXXXXXXXXXXXX,20180101000012GMT1,0000000012,84975634,A7EB2EBBB02E8D56AFFD07AC68CBB93781DAEBD51D730D2BEAB2191BB7DEFDBF,GETONECONF,VBATMIN,03300

12:50:23:724 DEBUG SerialDevice - >>**bat VOLT 3.70**

12:50:23:725 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:50:25:725 DEBUG SerialDevice - >>**bat VOLT 3.50**

12:50:25:725 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:50:27:726 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,12345678**

12:50:27:775 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXXX,20180101000016GMT1,0000000013,12345678,807269A1A5F42C83190633758709A92DA2755ABBEF1D5AAA17E72E8BABD6986B,**GETSENSORSTATE**,EVT1,0,EVT2,0,EVT3,0,EVT4,1,VTDM,04992,0,VUSB,04992,*,VBAT,03493,+,AIN1,00000,*,AIN2,00000,*,AIN3,00000,*,AIN4,00000,*,TEM1,02800,0,TEM2,02646,0,TEM3,00850,*,DIN1,0,*,DIN2,0,0,DIN3,0,0,DIN4,0,0,DIN5,0,0,DIN6,0,0,DIN7,0,0,DIN8,0,0,MIC1,0,*,OPT1,0,*,OPT2,0,*,VIB1,0,*,PUL1,0,*,MOT1,0,*,**CHRG,0**,*,BATX,0,0

12:50:27:775 DEBUG SerialDevice - >>**bat VOLT 3.40**

12:50:27:775 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:50:29:775 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,12345678**

12:50:29:824 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXXX,20180101000018GMT1,0000000014,12345678,2B39E77017FA81F5CE3F3ACB25300D4834A3FA893482E97C11BC6B0A5D2D5BDD,**GETSENSORSTATE**,EVT1,0,EVT2,0,EVT3,0,EVT4,1,VTDM,04992,0,VUSB,04992,*,VBAT,03398,+,AIN1,00000,*,AIN2,00000,*,AIN3,00000,*,AIN4,00000,*,TEM1,02800,0,TEM2,02644,0,TEM3,00850,*,DIN1,0,*,DIN2,0,0,DIN3,0,0,DIN4,0,0,DIN5,0,0,DIN6,0,0,DIN7,0,0,DIN8,0,0,MIC1,0,*,OPT1,0,*,OPT2,0,*,VIB1,0,*,PUL1,0,*,MOT1,0,*,**CHRG,0**,*,BATX,0,0

12:50:29:825 DEBUG SerialDevice - >>**bat VOLT 3.20**

12:50:29:825 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:50:31:825 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,12345678**

12:50:31:874 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXXX,20180101000020GMT1,0000000015,12345678,50CAE00C845D9F0789FE2575BB1841584CC3FEA7B182089CE8E04A070287BCD9,**GETSENSORSTATE**,EVT1,0,EVT2,0,EVT3,0,EVT4,1,VTDM,04734,0,VUSB,04734,*,VBAT,03794,+,AIN1,00000,*,AIN2,00000,*,AIN3,00000,*,AIN4,00000,*,TEM1,02800,0,TEM2,02646,0,TEM3,00850,*,DIN1,0,*,DIN2,0,0,DIN3,0,0,DIN4,0,0,DIN5,0,0,DIN6,0,0,DIN7,0,0,DIN8,0,0,MIC1,0,*,OPT1,0,*,OPT2,0,*,VIB1,0,*,PUL1,0,*,MOT1,0,*,**CHRG,1**,*,BATX,0,0

12:50:31:874 DEBUG TdmResource - **[Waiting for 2000 ms]**

12:50:33:875 DEBUG SerialDevice - >>**bat VOLT 4.20**

12:50:33:875 DEBUG TdmResource - **[Waiting for 2000 ms]**

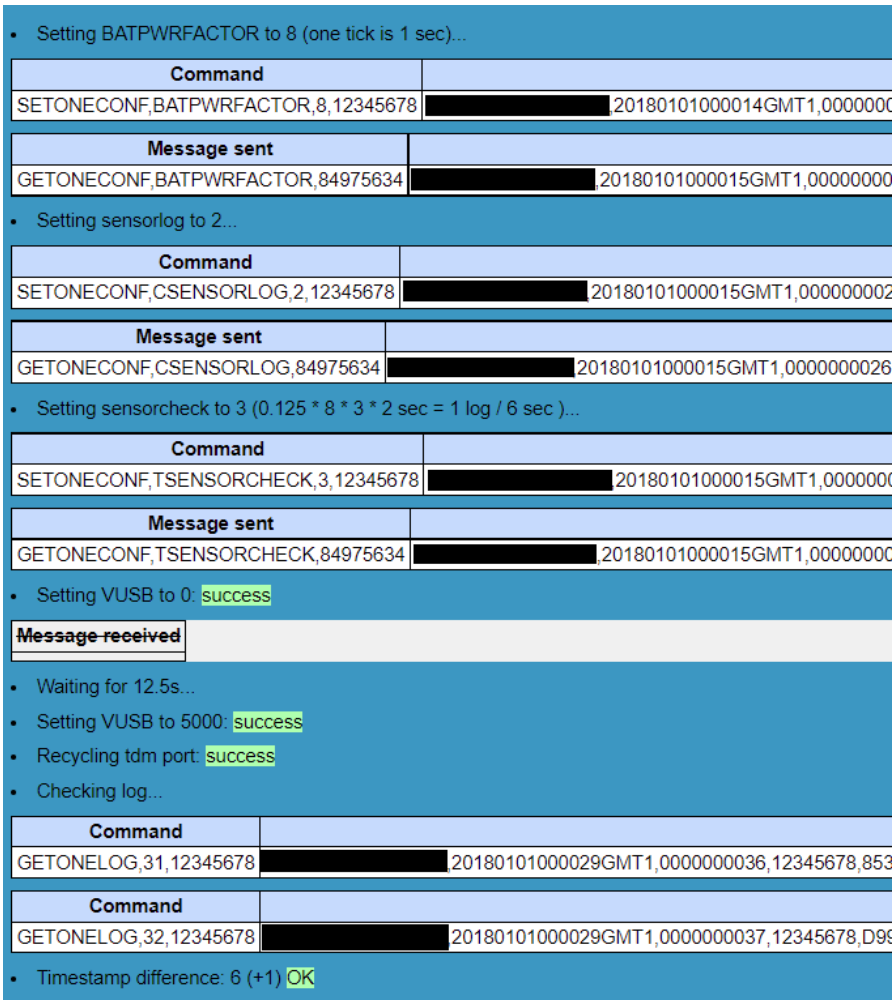
12:50:35:875 DEBUG SerialDevice - >>**tdm GETSENSORSTATE,12345678**

12:50:36:030 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXXX,20180101000024GMT1,0000000016,12345678,30433B9E605D47C982D8A84DC954948A27552B2E84F64D504677CF3472A0D3EB,**GETSENSORSTATE**,EVT1,0,EVT2,0,EVT3,0,EVT4,1,VTDM,05029,0,VUSB,05029,*,VBAT,04148,+,AIN1,00000,*,AIN2,00000,*,AIN3,00000,*,AIN4,00000,*,TEM1,02800,0,TEM2,02640,0,TEM3,00850,*,DIN1,0,*,DIN2,0,0,DIN3,0,0,DIN4,0,0,DIN5,0,0,DIN6,0,0,DIN7,0,0,DIN8,0,0,MIC1,0,*,OPT1,0,*,OPT2,0,*,VIB1,0,*,PUL1,0,*,MOT1,0,*,**CHRG,0**,*,BATX,0,0

7.4. ábra: SETONECONF [VBATMIN] teszt parancssori kimenetének részlete

7.3 Akkumulátorról üzemelés

Az akkumulátorról való futáshoz tesztként ismét a SETONECONF egyik alparancsát, a [CSENSORLOG]-ot választottam. Ezzel az utasítással azt lehet megadni, hogy a szenzorállapot sztring hanyadik leolvasás után legyen naplózva. A akkumulátor névleges, 3,6 V-os feszültsége lett beállítva a labortápegységen, mely végig megmarad, változatlan. A működési módok közti váltás lebonyolítását az USB kapcsoló fogja végezni.



7.5. ábra: SETONECONF [CSENSORLOG] teszt riport tészlete

A műveletek a 7.5-ös, illetve a 7.6-os ábrán szemrevételezhetők. A TDM ütemezőjének alapegysége 1/8 szekundum, melyet USB táplálás megléténél használ. Akkumulátoros módban ez az érték felszórzódik a BATPWRFACTOR-ban megadott számmal energiatakarékossági okokból. A naplózást minden második olvasásra állítjuk be, majd TSENSORCHECK-nek (hány ütemenként legyen szenzorleolvasás) 3 lesz beadva. Ezeket összeszorozva kijön, hogy szállítási módban 6 másodpercenként kellene egyetlen szenzorleolvasásnak elmentődnie a logtárba. A konfiguráció után a YkushXS kapcsolóval leválasztjuk az USB portot. Eközben keletkezik egy üzenet, amit TDM hardver esetében nem tudunk fogadni a csatorna bezáródása miatt, de naplózásra kerül: UPDATE,PWR_BATTERY. Várunk 12,5 másodpercet, hogy két naplózás biztos megtörténjen, majd visszakapcsoljuk a TDM-et a rendszerbe. Ilyenkor egyaránt bejegyződik az energiaforrás megváltozása: UPDATE,PWR_USB. Mivel a tesztrendszerrel való kommunikáció megszakadt, így szükség van a soros kapcsolat újranyitására. A port megnyitásakor nem kaptunk UPDATE,BOOTING_UP értesítést, ami azt jelenti, hogy a TDM az előzőektől nem állt le, hanem ténylegesen szállítási módba kapcsolt. Végül a naplóból lekérésre kerül GETONELOG utasítással a két, leválasztás után létrejött SENSORLOG[22] állapotüzenet, melyek időkülönbségének helyessége a vizsgálandó tényező.

[22] A GETSENSORSTATE-tel azonos a tartalma, csak az elnevezése más

12:27:21:991 DEBUG SerialDevice - >>**tdm SETONECONF,BATPWRFACTOR,8,12345678**

12:27:22:016 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXX,20180101000014GMT1,0000000023,12345678,2528C72CC16ECC4C98319141BB033BEB96BEAB76A95D0068325CDBA3C3AB7E70,**SETONECONF,BATPWRFACTOR,8**

12:27:22:016 DEBUG SerialDevice - >>tdm GETONECONF,BATPWRFACTOR,84975634

12:27:22:029 DEBUG SerialDevice - <<tdm
XXXXXXXXXXXXXXXX,20180101000015GMT1,0000000024,84975634,4288F54EED3D0C3ECD4A955980C0026DF12D67B19D9FCA245A41DE2ACCED1D97,GETONECONF,BATPWRFACTOR,8

12:27:22:030 DEBUG SerialDevice - >>**tdm SETONECONF,CSENSORLOG,2,12345678**

12:27:22:056 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXX,20180101000015GMT1,0000000025,12345678,AA69AB5A67F3B5250AED0C02FD8E3151AFE6F06878FD1827DD65C99E1CAEC95D,**SETONECONF,CSENSORLOG,00002**

12:27:22:056 DEBUG SerialDevice - >>tdm GETONECONF,CSENSORLOG,84975634

12:27:22:069 DEBUG SerialDevice - <<tdm
XXXXXXXXXXXXXXXX,20180101000015GMT1,0000000026,84975634,8F7C3563044974F3996842B0600F28AA645F3A837499E8D03D270D06645C9268,GETONECONF,CSENSORLOG,00002

12:27:22:069 DEBUG SerialDevice - >>**tdm SETONECONF,TSENSORCHECK,3,12345678**

12:27:22:095 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXX,20180101000015GMT1,0000000027,12345678,D2D7BBB78A633C0F08C810836E6ACD3EFBA42B9F23C5ECD7841C5BDF8A215452,**SETONECONF,TSENSORCHECK,00003**

12:27:22:095 DEBUG SerialDevice - >>tdm GETONECONF,TSENSORCHECK,84975634

12:27:22:108 DEBUG SerialDevice - <<tdm
XXXXXXXXXXXXXXXX,20180101000015GMT1,0000000028,84975634,265A3C83B2AC84CAF5F10C8A340B9D4A8B44717643CE8C3EA0ABF729648CCEE7,GETONECONF,TSENSORCHECK,00003

12:27:22:117 DEBUG OsCommand - [ubuntu]

12:27:22:117 DEBUG **USBSwitch - Switching USB via ykushxs to OFF**

12:27:22:118 DEBUG TdmResource - **[Waiting for 12500 ms]**

12:27:34:631 DEBUG OsCommand - [ubuntu]

12:27:34:631 DEBUG **USBSwitch - Switching USB via ykushxs to ON**

12:27:34:632 DEBUG SerialDevice - **Closing ttyACM1**

12:27:36:633 DEBUG SerialDevice - **Opening ttyACM1 - User-Specified Port**

12:27:36:633 DEBUG SerialDevice - >>**tdm GETONELOG,31,12345678**

12:27:36:662 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXX,20180101000029GMT1,0000000036,12345678,85390395D160F13DCB6DDEF7322AC38DC39F91B0CEF825ABA09E8F41E30E3180,**GETONELOG**,0000000031,**20180101000021**GMT1,********,94FEB33E12D9EA914CC555D13BD3ED122328AA8B9210CF1742973C8E62056C0F,**SENSORLOG** [...]

12:27:36:663 DEBUG SerialDevice - >>**tdm GETONELOG,32,12345678**

12:27:36:686 DEBUG SerialDevice - <<**tdm**
XXXXXXXXXXXXXXXX,20180101000029GMT1,0000000037,12345678,D99DAD6BA27C735D6B0DBD06E35A0E23BFF018523AB17726F4142BCEF80C2798,**GETONELOG**,0000000032,**20180101000027**GMT1,********,A7371594F89F37CCA59C5420AE076BD54194A225875B73E6F1A46CAE42B0C58B,**SENSORLOG** [...]

7.6. ábra: SETONECONF [CSENSORLOG] teszt parancssori kimenetének részlete

8 ÖSSZEGZÉS

8.1 Összefoglalás

A dolgozat során áttekintettem a tesztelést mint fogalmat, annak csoportosítását. Ebben elhelyeztem az I4P TDM szabotázs detektáló modul már meglévő folyamatát, hogy képet kaphassak az elvégzendő módosítási feladatokról. Bemutattam a tesztelendő eszközt célja, valamint kommunikációs csatornái tekintetében. Ezt követően felállítottam egy követelményrendszert, mely alapján kiválasztottam az optimális interakciós eszközt. A vizsgált eszközfajták közül végül egy Arduino-kompatibilis mikrokontroller, a Raspberry Pi Pico került alkalmazásra. A tervezés során a feladat kiegészült az akkumulátorkezelés kialakításával, mely egy cég által biztosított USB kapcsoló és labortápegység segítségével történt meg.

A TDM hardveres tesztelését biztosító, automatizációs projekt megvalósítását sikeresen befejeztem. Egy olyan rendszert hoztam létre, mely képes az USB interfésszel, ki- és bemeneti portokkal, valamint a tápellátással kapcsolatos teszteseteket emberi beavatkozás nélkül elvégezni, kellő alapot nyújtva további modulok beillesztésére. A rendszert rögtön használatba is vette az I4P, melyet az új TDM verziók auditálást megelőző hibakeresésére, ezen folyamat gyorsítására használ. A teszteredmények nagy részének elkészülési ideje a fejlesztés révén napokról órákra csökkent. Mindezek ellenére, a rendszernek több ponton fejlődnie szükséges, például a kimaradt digitális-analóg átalakító kérdésében. Egy másik hiányosság, hogy a Jenkinst a központi beléptetést végző, LDAP-alapú szolgáltatáshoz lehetett volna csatlakoztatni helyi felhasználók beállítása helyett.

Továbbfejlesztési lehetőségként felmerült nálam egy modernebb tápegység vagy akkuszimulátor céleszköz beszerzése, mely felváltaná a korosodó mostanit. Ennek költségbeli korlátai vannak, ugyanis egy ilyen eszköz ára az egymillió forinthoz közelít. A TDM érzékelők tesztelésére alkalmas fizikai hatáskeltő kamra megépítése és szoftveres implementálása is ide tartozik, mely - bonyolultsága miatt - alapos tanulmányozást igényel. A TDM és a hardveres teszter firmware új verzióinak automatikus fordítása és feltöltése is megoldható, de időigényes feladatok. A leírtakon felül számos olyan, bővítést célzó ötlet születhet a mindennapi munka és a TDM termék fejlődése során, melyre jelen pillanatban még nem is gondolnánk.

8.2 Summary

During the thesis I reviewed testing as a concept, and its grouping. In this, I placed the pre-existing process of the I4P TDM tamper detection module to get an idea of the modification tasks to be performed. I presented the device in terms of its purpose as well as its communication channels. After that, I set up a system of requirements, based on which I selected the optimal interaction tool. Among the device types examined, an Arduino-compatible microcontroller, the Raspberry Pi Pico, was finally used. During the design, the task was supplemented by the implementation of the battery management, which was done with the help of a USB switch and a laboratory power supply provided by the company.

I have successfully completed the implementation of an automation project that provides hardware testing for TDM. I have created a system that can perform test cases related to the USB interface, output and input ports, and power supply without human intervention, providing a good basis for inserting additional modules. The system was immediately put into use by I4P,

which it uses to debug new TDM versions before auditing, and speed up this process. The completion time for most test results has been reduced from day to hour level through development. Nonetheless, the system needs to evolve at several points, such as the issue of the missing digital-to-analog converter. Another shortcoming is that Jenkins could have been connected to the central access LDAP-based service instead of setting up local users.

As for further development, it would be great to obtain a more modern power supply or battery simulator target device, which would replace the aging one used by now. This has cost limitations, as the price of such an asset is close to one million forints. The construction and software implementation of a physical exposure chamber suitable for testing TDM sensors is also planned, which - due to its complexity - requires thorough study. Automatic building and uploading the new versions of TDM and hardware tester firmware can also be accomplished, but are time consuming tasks. In addition to what has been described, there can be a number of ideas for expansion in the day-to-day work and development of the TDM product that we would not even thinking about at the moment.

9 HIVATKOZÁSOK

[1] C. Máté, „Continuous integration & delivery,” [Online]. Available: https://mcserep.web.elte.hu/data/education/2018-2019-2_SZT/elte_szt_ea10_ci.pdf. [Hozzáférés dátuma: 25. 04. 2021.].

[2] M. I. Micskei Zoltán, „A szoftver tesztelés alapjai,” [Online]. Available: https://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/szoftverellen%C5%91rz%C3%A9si-technik%C3%A1k/12/SZET-2012_EA06_teszteles_alapjai.pdf. [Hozzáférés dátuma: 30. 03. 2021.].

[3] M. István, „Beágyazott rendszerek ellenőrzéstechnikája, előadásvázlat,” [Online]. Available: https://www.mit.bme.hu/system/files/oktatas/targyak/7049/majzik_vazlat2006.pdf. [Hozzáférés dátuma: 30. 03. 2021.].

[4] D. K. Gábor, „Szoftvertesztelés (1. fejezet),” [Online]. Available: http://aries.ektf.hu/~gkusper/SzoftverTeszteles.pdf. [Hozzáférés dátuma: 05. 04. 2021.].

[5] Arduino.cc, „Arduino Logical Constants,” [Online]. Available: https://www.arduino.cc/reference/en/language/variables/constants/constants/. [Hozzáférés dátuma: 19. 04. 2021.].

[6] Arduino.cc, „Arduino Interrupts,” [Online]. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/. [Hozzáférés dátuma: 26. 04. 2021.].

[7] Arduino.cc, „Arduino Due information page,” [Online]. Available: https://store.arduino.cc/arduino-due. [Hozzáférés dátuma: 19. 04. 2021.].

[8] Advantech Co., Ltd., „Advantech USB-4751,” [Online]. Available: https://www.advantech.com/products/1-2mlkno/usb-4751/mod_5896d86d-dfeb-4b9e-b1c5-a9529a401a5f. [Hozzáférés dátuma: 19. 04. 2021.].

[9] Deciphe it GmbH, „LucidControl homepage,” [Online]. Available: https://www.lucid-control.com/. [Hozzáférés dátuma: 20. 04. 2021.].

[10] Deciphe it GmbH, „LucidControl Digital Input Module page,” [Online]. Available: https://www.lucid-control.com/product/lucidcontrol-di4-4-channel-digital-input-usb-module/. [Hozzáférés dátuma: 20. 04. 2021.].

[11] Deciphe it GmbH, „LucidControl DI Module documentation,” [Online]. Available: https://www.lucid-control.com/wp-content/uploads/Documentation/Current/User-Manual-LucidControl-DI4.pdf. [Hozzáférés dátuma: 20. 04. 2021.].

[12] Electrical Classroom, „Az elektromechanikai és szilárdtest relé összehasonlítása,” [Online]. Available: https://www.electricalclassroom.com/electromechanical-vs-solid-state-relay/. [Hozzáférés dátuma: 20. 04. 2021.].

[13] Deciphe it GmbH, „LucidControl Digital Output Module page,” [Online]. Available: https://www.lucid-control.com/product/lucidcontrol-do4-4-channel-digital-output-usb-module/. [Hozzáférés dátuma: 20. 04. 2021.].

[14] Deciphe it GmbH, „LucidControl DO Module documentation,” [Online]. Available: https://www.lucid-control.com/wp-content/uploads/Documentation/Current/User-Manual-LucidControl-DO4.pdf. [Hozzáférés dátuma: 20. 04. 2021.].

[15] Deciphe it GmbH, „LucidControl Analog Output Module page,” [Online]. Available: https://www.lucid-control.com/product/lucidcontrol-ao4-4-channel-analog-output-usb-module/. [Hozzáférés dátuma: 20. 04. 2021.].

[16] Elektrotanya.com, „Optocsatolók,” [Online]. Available: https://elektrotanya.com/files/forum/2011/05/opto.PDF. [Hozzáférés dátuma: 20. 04. 2021.].

[17] Hlács Ferenc, „Több mint 30 millió Raspberry Pi talált gazdára 2012 óta,” [Online]. Available: https://www.hwsw.hu/hirek/61238/raspbery-pi-fejlesztoi-lapka-eladas.html. [Hozzáférés dátuma: 22. 04. 2021.].

[18] Raspberry Pi Foundation Ltd., „Raspberry Pi 4 Model B page,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b. [Hozzáférés dátuma: 22. 04. 2021.].

[19] Raspberry Pi Foundation Ltd., „Raspberry Pi GPIO documentation,” [Online]. Available: https://www.raspberrypi.org/documentation/usage/gpio/. [Hozzáférés dátuma: 22. 04. 2021.].

[20] Raspberry Pi Foundation Ltd., „Raspberry Pi feszültségszintek,” [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md. [Hozzáférés dátuma: 22. 04. 2021.].

[21] pi4j.com, „Pi4J Minimal Example Application,” [Online]. Available: https://pi4j.com/getting-started/minimal-example-application/. [Hozzáférés dátuma: 22. 04. 2021.].

[22] StackExchange Inc., „Raspberry Pi 4 CI server performance,” [Online]. Available: https://raspberrypi.stackexchange.com/questions/114144/pi-4-performance-against-x86-ci-cd-server-with-java-maven. [Hozzáférés dátuma: 22. 04. 2021.].

[23] Avram Piltch, Les Pounder, „How to Boot Raspberry Pi 4 From a USB SSD or Flash Drive,” [Online]. Available: https://www.tomshardware.com/how-to/boot-raspberry-pi-4-usb. [Hozzáférés dátuma: 22. 04. 2021.].

[24] Raspberry Pi Foundation Ltd., „Raspberry Pi Thermal Control,” [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/frequency-management.md. [Hozzáférés dátuma: 22. 04. 2021.].

[25] Raspberry Pi Foundation Ltd., „Raspberry Pi Power Supply,” [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md. [Hozzáférés dátuma: 22. 04. 2021.].

[26] CONELCOM GmbH, „Controllino homepage,” [Online]. Available: https://www.controllino.com/. [Hozzáférés dátuma: 20. 04. 2021.].

[27] SECO S.p.A., „UDOO Bolt page,” [Online]. Available: https://www.udoo.org/discover-the-udoo-bolt/. [Hozzáférés dátuma: 20. 04. 2021.].

[28] StackExchange Inc., „C++ vs. The Arduino Language,” [Online]. Available: https://arduino.stackexchange.com/questions/816/c-vs-the-arduino-language. [Hozzáférés dátuma: 25. 04. 2021.].

[29] Canonical Ltd., „What is Ubuntu LTS release?,” [Online]. Available: https://ubuntu.com/blog/what-is-an-ubuntu-lts-release. [Hozzáférés dátuma: 25. 04. 2021.].

[30] jenkins.io, „Jenkins User Documentation,” [Online]. Available: https://www.jenkins.io/doc/. [Hozzáférés dátuma: 25. 04. 2021.].

[31] E. F. P. III., „Arduino port for Raspberry Pi Pico,” [Online]. Available: https://github.com/earlephilhower/arduino-pico. [Hozzáférés dátuma: 19. 10. 2021.].

[32] StackExchange Inc., „Arduino Hash table usage,” [Online]. Available: https://stackoverflow.com/questions/15727814/arduino-hash-table-dictionary. [Hozzáférés dátuma: 09. 12. 2021.].

[33] Wikipedia HUN, „Prell-hatás,” [Online]. Available: https://hu.wikipedia.org/wiki/Prell. [Hozzáférés dátuma: 28. 11. 2021.].

[34] StackExchange Inc., „How to run a script as root in Jenkins,” [Online]. Available: https://stackoverflow.com/questions/11880070/how-to-run-a-script-as-root-in-jenkins. [Hozzáférés dátuma: 23. 10. 2021.].

[35] StackExchange Inc., „Jenkins no CSS displayed when report is viewed,” [Online]. Available: https://stackoverflow.com/questions/35783964/jenkins-html-

publisher-plugin-no-css-is-displayed-when-report-is-viewed-in-j. [Hozzáférés dátuma: 23. 10. 2021.].

10 ÁBRAJEGYZÉK

11 TÁBLÁZATJEGYZÉK