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

Kritikus kiberfizikai rendszerek koszimuláció-alapú megbízhatósági vizsgálata heterogén szimulációs környezetben

TDK dolgozat

Készítette:

Dóra Márton

Konzulens:

dr. Vörös András
Nagy Simon József

2023

Tartalomjegyzék

Kivonat

A kritikus kiberfizikai rendszerek az élet egyre több területén megtalálhatóak és fontos funkciókat nyújtanak számunkra. Ilyenek többek között az autóipari, a vasúti, az űripari és villamos energetikai alkalmazások is, amelyek esetén kiemelten fontos a megbízható és biztonságos működés. Emiatt a kritikus kiberfizikai rendszerek fejlesztése során elsődleges szempont a robosztus, hibatűrő architektúrák tervezése és verifikálása. Mivel az architektúra hibáinak javítása egyre nehezebb a fejlesztés előrehaladtával, ezért kiemelten fontos a rendszerben hibák hatásainak a minél korábbi kiértékelése a fejlesztési folyamat során. Emiatt a kritikus kiberfizikai rendszerek fejlesztése során előszeretettel használják a modellvezérelt tervezési paradigmát és szimuláció-alapú verifikációs technikákat.

Komplex kiberfizikai rendszerek vizsgálata esetén a fizikai rendszerek megépítése és tesztelése helyett hatékonyabb, olcsóbb és gyorsabb megoldás lehet szimulációkat készíteni, amely használatával a hibák a tervezés korai fázisaiban is hatékonyan megtalálhatóak. A fizikai rendszerek modellezése során a megfelelő absztrakciót kiválasztva átlátható és kezelhető modelleket kaphatunk, amelyek támogatják a mérnöki megértést. Azonban a mérnöki gyakorlatban a rendszer különböző aspektusait a rendszernek más és más modellező nyelvek és eszközök segítségével reprezentálhatják. A rendszer architektúra heterogén természete problémát jelenthet a verifikáció során.

Egy rendszer megbízhatóságának vizsgálata során a legkülönbözőbb külső és belső eseményekre, azaz hibákra kellhet felkészülni, többek között tranziens kommunikációs és mérési hibák, permanens hardver hibák és gyártási hibák is megjelenhetnek. Létezhetnek olyan kritikus hiba kombinációk, amelyek hatására a rendszer veszélyes állapotba kerülhet. A rendszer működésének vizsgálatakor célom volt az architektúrák biztonságosságának kiértékelése és a variánsok összehasonlítása.

Dolgozatomban bemutatok egy új és innovatív rendszertervezési megközelítést, amely lehetővé teszi a szoftver és hardver komponensek hibatűrésének verifikációját a szoftveres, hardveres, környezeti és hibamodellek koszimulációjának segítségével. A lehetséges hibák vizsgálatának céljából támogatjuk a folytonos állapotú és idejű fizikai rendszermodellek hiba modellekkel való kibővítését. A hardveres és szoftveres megoldások szorosan összekapcsolódnak, egy rendszer hardveres és szoftveres részét együtt szükséges kezelni, így koszimulációs módszert alkalmaztam szabványos interfészeken keresztül. A modellezésben lehetőség van a környezet hatásának figyelembe vételére is. A megközelítésem alkalmazhatóságát esettanulmányok segítségével vizsgáltam.

Abstract

Critical cyber-physical systems play a critical role in automotive, rail, aerospace, and electrical energy applications, where reliability and safety are primary concerns. For this reason, the design and verification of robust, fault-tolerant architectures is essential in the development process of critical cyber-physical systems. As architecture failures become increasingly difficult to correct as development progresses, it is of paramount importance to evaluate the impact of failures in the system as early as possible in the development process. For this reason, the model-driven design paradigm and simulation-based verification techniques are preferably used in the development of critical cyber-physical systems.

In the early phases of development, using simulation for verification can be more efficient, cheaper, and faster than building a prototype. We can obtain transparent and tractable models that support engineering understanding by choosing the right abstraction during modeling. However, different modeling languages and tools are used in the engineering practice to represent different aspects of the system. The heterogeneous nature of the system architecture might pose a challenge during verification.

The verification of complex cyber-physical systems may evaluate a wide variety of external and internal phenomena, i.e., transient communication and measurement faults, permanent hardware failures, and manufacturing defects. The introduced simulation approach can identify critical combinations of failures that can cause the system to enter a dangerous state. In examining the operation of the system, I aimed to evaluate the reliability and safety of the architectures and compare the variants.

This thesis presents a new and innovative system design approach that facilitates the verification of fault tolerance of software and hardware components through the co-simulation of software, hardware, environment, and fault models. We support the extension of continuous-state-space and continuous-time physical system models with fault models to investigate the effect of potential faults. Hardware and software solutions are closely coupled. Consequently, the hardware and software models need to be verified together. As a result, I developed a co-simulation approach that uses standardized interfaces to evaluate hardware software and environmental behavior simultaneously. I tested the applicability of the presented approach through industrial case studies.

1. Bevezetés

A modell-alapú rendszertervezés napjainkra egyre elterjedtebbé vált a különböző iparágakban. A modellezés számos előnyt biztosít a mérnöki csapatok számára, hiszen a hatékonyságot növeli, a költségeket csökkenti, elősegíti a különböző területek közötti jobb együttműködést. Sokszor nagyon bonyolult modellek megalkotására van szükség, ezért elengedhetetlen a számítógépes modellezési technikák használata. A mérnökök munkáját a számítógépes szoftverek és hardverek fejlődése megkönnyíti.

A kiberfizikai kritikus rendszerek tervezésénél fontos a hiba modellezése, hiszen a kiberfizikai rendszerek biztonságát garantálni kell. A biztonságos rendszerek létrehozásánál sokféle kihívással kell szembenézni, amelyek megnehezítik a működésüket. Például ilyen kihívás lehet az, hogy sokféle hiba előfordulhat, a rendszerek viselkedése bonyolult, vagy az, hogy sok különböző hardver és szoftver komponens együttes viselkedését kell kiértékelni a rendszer biztonságosságának ellenőrzése során.

A jelenleg elérhető legjobb modellező és szimuláló eszközök között tartják számon az OpenModelicát. Ebben a dolgozatban én is az OpenModelicát fogom használni számos előnye és könnyű hozzáférhetősége miatt. Az OpenModelicában előre elkészített komponenseket össze lehet kapcsolni, ezáltal létrehozni egy rendszer fizikai modelljét.

Dolgozatomban négy különböző modell megalkotását, ezek jellemzőit és működését fogom bemutatni. A hőmérsékleti rendszereken végzett szimulációkkal azoknak a feltételeknek a megismerése a célom, melyek mellett a modellek biztonságosan működnek. A koszimulációs modellhez egy hibamodellt is fogok kapcsolni, mivel a lehetséges veszélyes hibakonfigurációk kiszűrése a tervezés korai szakaszában célszerű.

2. Háttérismeretek

Ebben a fejezetben mutatom be a témához kapcsolódó háttérismereteket.

2.1. Háttér áttekintése

A kiberfizikai rendszerek a legtöbb iparágban fontos szerepet töltenek be, mint például az űriparban, a vasúti vagy az autóiparban [13]. A modellezés egy jól bevált technika a bonyolult rendszerek kezelésére. A modell a fizikai rendszer absztrakt reprezentációja. A modellezés során csak a legszükségesebb részleteket ábrázoljuk, ezáltal a rendszer viselkedése és a szerkezete könnyebben átlátható. A szimuláció abban segít, hogy az adott feltételek mellett megtaláljuk a legjobb megoldásokat és elkerüljük a gazdaságtalan vagy hibás működést. [6] Alapvető elvárás a kiberfizikai rendszerek megfelelő működése, ezért a modellezés során nem csak a fejlesztésre, de az ellenőrzésre is komoly hangsúlyt kell fektetni [3].

A rendszer megbízhatósági vizsgálata során fel kell készülni a legkülönbözőbb eseményekre. A fejlesztés előrehaladtával egyre nehezebb a hibák javítása, ezért fontos a hibák minél korábbi kiértékelése. A hibadiagnosztikával több szakirodalom is kiemelten foglalkozik [5], [9].

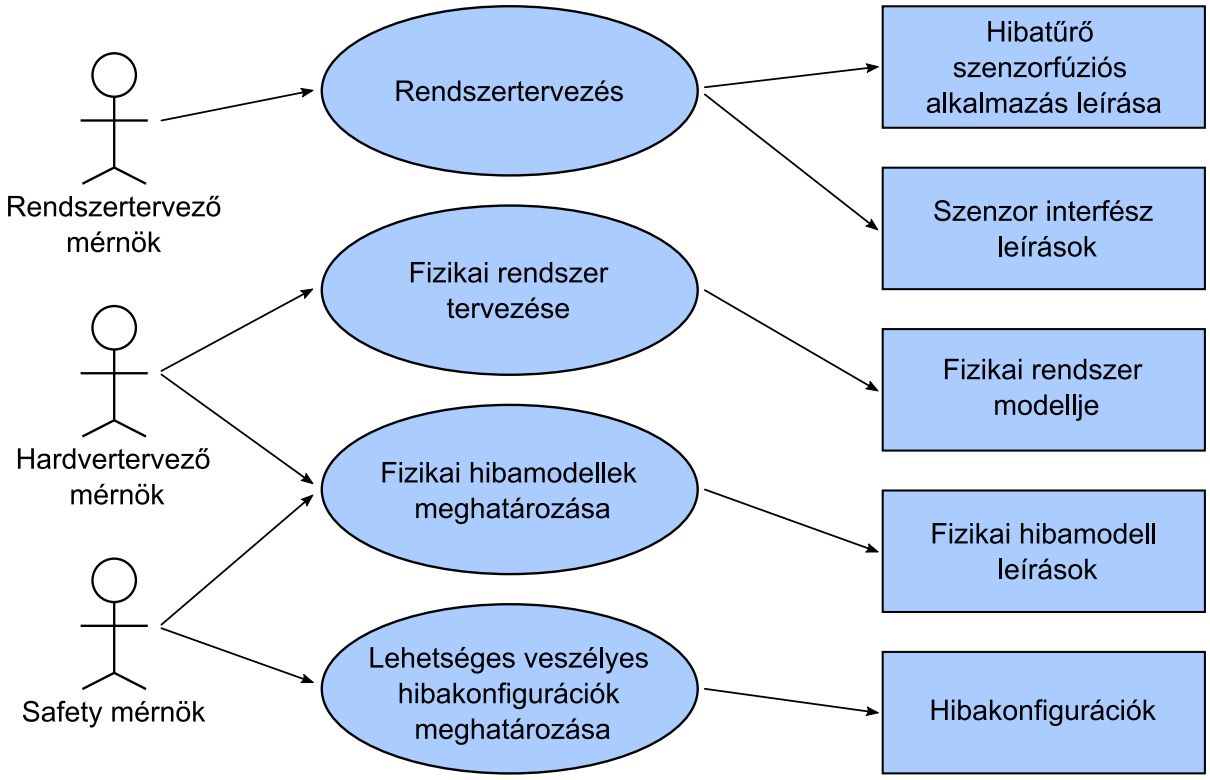
A megbízhatósági vizsgálatok legtöbbször olyan koszimulációs környezetben történnek, melyekben különböző területekről rendelkezésre álló eszközök vesznek részt. Ilyenkor több szoftverkomponens alkalmazása történik párhuzamosan [10]. A koszimuláció elvégzésének klasszikus algoritmusa az, hogy minden egyes különálló modell végrehajt egy szimulációs lépést, információt cserél a modellek között, majd újabb lépést tesz. Ezt a folyamatot ismétli. (A 3.2. fejezetben részletesebben is bemutatom a koszimuláció működését.)

2.2. A tervezés folyamata

A tervezői folyamatban több szakterület mérnökei együttesen vesznek részt. A rendszertervezés a rendszertervező mérnök feladata, többek között ő végzi el a hibatűrő szenzorfúziós alkalmazások leírását, valamint a szenzor interfész leírások elkészítését is.

A hardvertervező mérnök fő feladata az adott fizikai rendszer tervezése és ehhez kapcsolódóan a fizikai rendszer modelljének megalkotása. A hardvertervező mérnök a fizikai hibamodellek meghatározásában és a fizikai hibamodell leírásainak elkészítésében is részt vesz, a safety mérnökkel együttműködve.

A safety mérnök ezen kívül meghatározza a lehetséges veszélyes hibakonfigurációkat is. A folyamat szereplőit és főbb feladatait a 2.1. ábra szemlélteti.



2.1. ábra. A folyamatban résztvevő mérnökök

2.3. Felhasznált számítógépes eszközök

Munkám során többféle szoftvert és interfészt használtam.

2.3.1. OpenModelica

A rendszerek modellezéséhez és szimulálásához az OpenModelica nyílt forráskódú eszközt használtam. Az OpenModelica egy olyan eszköz, amely számos területen használható, a mechanikai, elektromos, termikus és hidraulikus rendszerek modellezésére és szimulációjára. Az OpenModelicát a hatékonysága és számos előnye miatt választottam. Az OpenModelica modellszerkesztője olyan grafikus felhasználói felületet biztosít, amely lehetővé teszi a modellek egyszerű kialakítását és szerkesztését. Rendelkezik olyan könyvtárral, mely a modell elkészítéséhez előre felépített komponenseket tartalmaz. Szimulációs környezet teszi lehetővé a modellek szimulálását és az eredmények áttekintését. Az OpenModelica olyan vizualizációs eszközt tartalmaz, mely a felhasználók számára lehetővé teszi, hogy modelljeiket és szimulációs eredményeiket grafikusan, diagramokon megjelenítsék. [12]

Az OpenModelica megfelelő modellező és szimulációs környezetet biztosít a mérnököknek az ipari fejlesztésekhez és a diákoknak is, egyetemi felhasználásra.

2.3.2. Python

A Python nyelven való programozás munkám egyik fontos részét képezi. A Python erőssége, hogy könnyen tanulható és jól kommunikál egyéb környezetekkel. A Python nyelvet elsősorban azért választottam, mert ebben egyszerűen és gyorsan lehet prototípusokat létrehozni. [14]

2.3.3. FMI és FMU

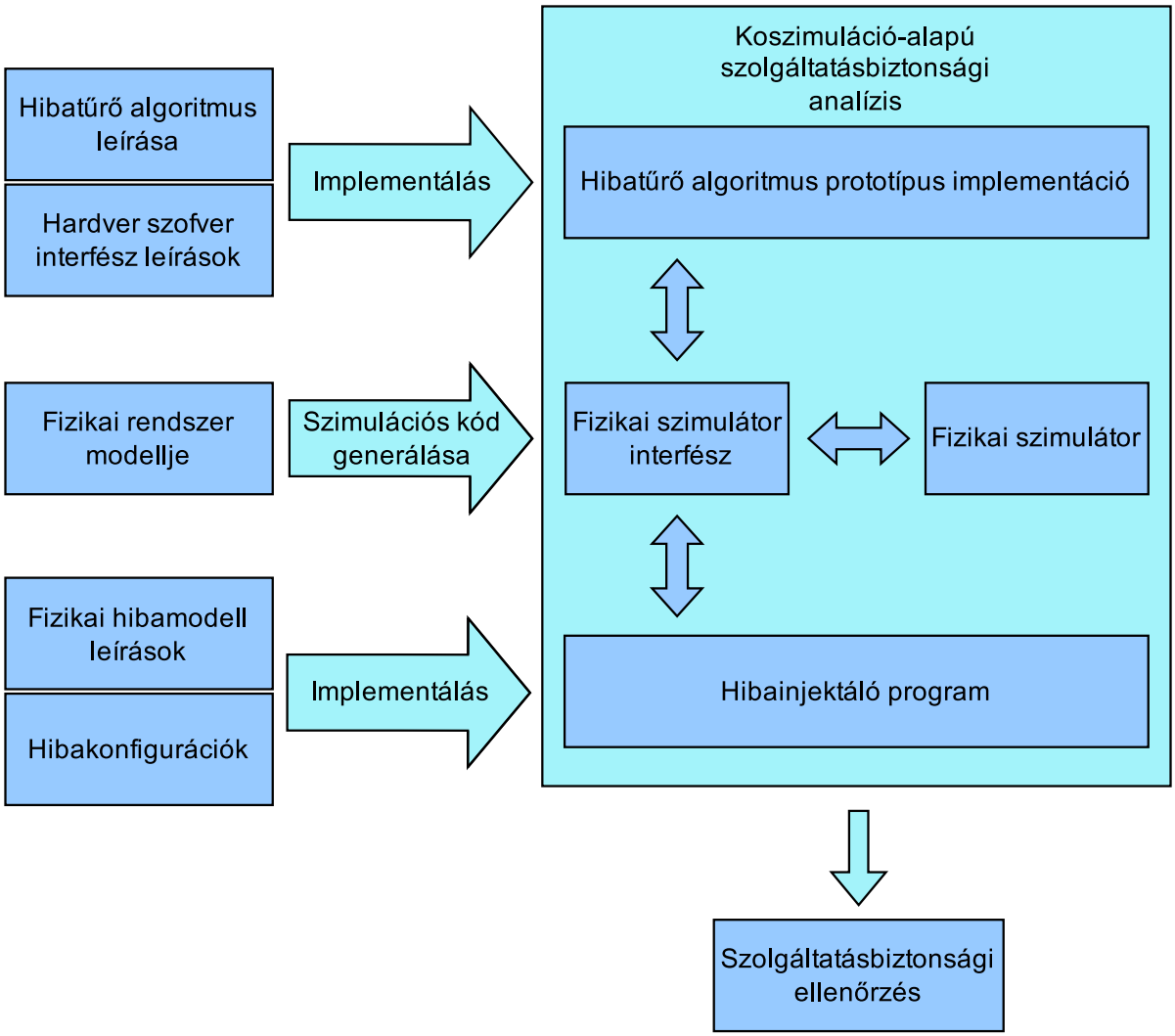
A Functional Mock-up Interface (FMI) technológia zökkenőmentesen összekapcsolja a szoftverkomponenseket a hardverkomponensek modelljével. A Python egyik célja az FMI szabványnak megfelelő modellekkel való munka. Úgy tervezték, hogy könnyen használható felületet biztosítson az Functional Mock-up Unittal (FMU) való munkához. Kihasználtam az OpenModelicának azt a sajátosságát, hogy támogatja a modellek FMI szabványnak megfelelő exportálását.

3. Koszimuláció alapú megbízhatósági vizsgálat

3.1. Koszimuláció alapú megbízhatósági vizsgálat folyamata

Dolgozatom központi eleme a koszimuláció alapú szolgáltatásbiztonsági analízis. Első lépés a *hibatűrő algoritmus leírása*, mely meghatározza hogy a rendszer hogyan tudja detektálni a hibákat, és elhárítani a hatásukat. A hibatűrő algoritmusokat a fizikai világgal a *hardver szoftver interfész leírások* kapcsolják össze. Ez leggyakrabban egy analóg digitális átalakító. Ezek alapján valósul meg a *hibatűrő algoritmus prototípus implementáció*. A *fizikai rendszer modellje* differenciálegyenletekből áll. A koszimuláció alapú megbízhatósági vizsgálat előkészítéseként a fizikai rendszer modellje alapján *szimulációs kód generálódik*. Ennek eredménye lesz a *fizikai szimulátor*. A fizikai szimulátorral a *fizikai szimulátor interfész*en keresztül tudunk kapcsolatot teremteni. A rendszer komponenseinek lehetséges meghibásodásait a *fizikai hibamodell leírások* és *hibakonfigurációk* adják meg. Ezekből megvalósítható a *hibainjektáló program*, mely a fizikai szimulátor interfészen keresztül megváltoztatja a fizikai szimulátor paramétereit. A koszimuláció-alapú szolgáltatásbiztonsági analízis eredménye a *szolgáltatásbiztonsági ellenőrzés*.

A megbízhatósági vizsgálat teljes folyamatát a 3.1. ábra szemlélteti.



3.1. ábra. A megbízhatósági vizsgálat folyamata

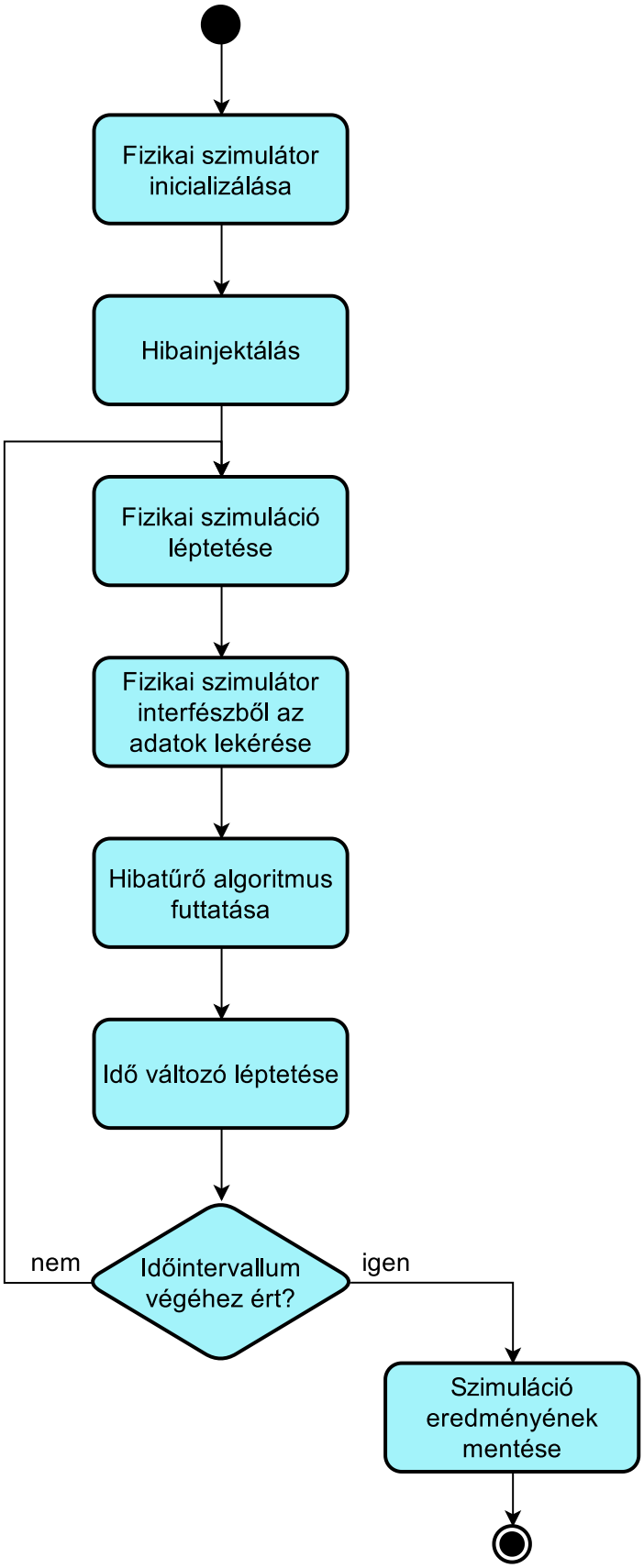
3.2. A koszimuláció működése

Munkám egyik központi eleme a koszimuláció alapú megközelítés, vizsgálódásom erre a módszertanra épül. Röviden összefoglalom a koszimuláció működését.

A koszimuláció működése a 3.2. ábrán látható. Az első lépés a fizikai szimulátor inicializálása. Ennek során beállításra kerül a fizikai rendszer paramétereinek értéke, és a szimuláció időtartama. Majd megtörténik a hibainjektálás a fizikai szimulátorba. Ezáltal a fizikai szimulátor bizonyos paraméterei megváltoznak.

Léptetjük a fizikai szimulációt, lekérjük az adatokat a fizikai szimulátor interfészből. Ezekkel futtatjuk a hibatűrő algoritmust. Ennek eredménye alapján változtatni kell a fizikai szimuláció paramétereit. Léptetjük az időváltozót. Ezt a folyamatot addig ismételjük, amíg szimuláció időtartamán belül van az időváltozó.

Ha végeztünk ezzel a folyamattal, el kell mentenünk a szimuláció eredményeit.



3.2. ábra. A koszimuláció működése

4. Kritikus hibakombinációk keresése

A dolgozatom témáját képező koszimuláció alapú megbízhatósági vizsgálat központi eleme a hibavizsgálat. Az áramköri elemek leggyakoribb meghibásodásait nemzetközi szabványok írják le. Eben a fejezetben bemutatom az ellenállások néhány lehetséges meghibásodási formáját. Ezek után bemutatok két módszert a kritikus hibakombinációk keresésére. Ezeket a módszereket az 5. fejezetben alkalmazom. Egy-egy hiba külön-külön történő előfordulását vagy néhány hiba együttes előfordulását a rendszer még képes megbízhatóan kezelni. Vannak azonban olyan hibakombinációk, amelyek előfordulásakor a rendszer nem működik megfelelően. Ezeket a hibakombinációkat kritikus hibakombinációknak nevezzük.

4.1. Ellenállások hibái

A későbbiekben az ellenállások lehetséges hibáinak és ezek szimulálásának fontos szerepe lesz. Háromféle lehetséges meghibásodást vizsgálok:

- az ellenállás értéke $\pm 5\%$-kal megváltozik
- az ellenállás rövidre záródik, értéke jelentősen lecsökken
- az ellenállás vezetőképessége megszűnik, ellenállása jelentősen megnő

4.2. Kritikus hibakombináció keresés kézi módszerrel

Ezt a módszert használva a hibákat a rendszerben egyesével, kézzel kell generálni. Ennek előnye, hogy a rendszer működését jól meg lehet érteni a hibák hatásának vizsgálatával, hátránya, hogy nehezen alkalmazható kritikus hibakombinációk keresésére, mivel nagyon sokféle hiba, rengeteg kombinációja létezik.

4.3. Automatizált kritikus hibakombináció keresés

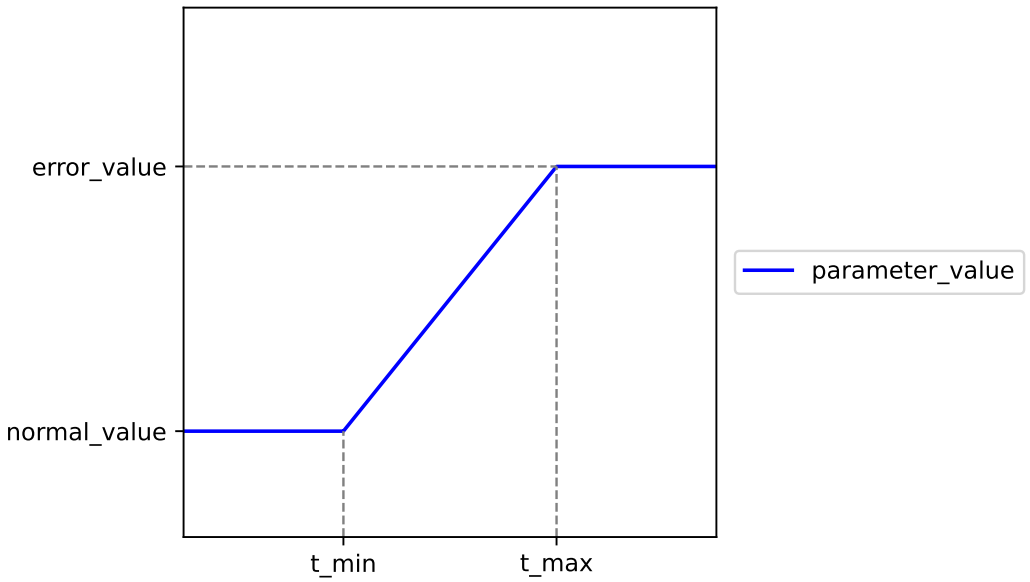
Egy másik módszer során fel kell sorolni, hogy milyen hibákat szeretnénk vizsgálni. Ebből egy program automatikusan kiválasztja az összes lehetséges kombinációt, lefuttatja azokkal a szimulációt és megmondja, hogy a rendszer helyesen működött-e. Ennek a módszernek hátránya, hogy számításigényes, a szimulációk futtatása jelentős időt vesz igénybe. Előnye viszont, hogy nagyobb eséllyel megtalálja a kritikus hibakombinációkat, mivel sok lehetséges esetet végigpróbál. (Az összes hibakombinációt elméletileg sem lehet kipróbálni, mivel szerepelnek folytonos értékek a hibák leírásában.)

4.4. Pythonban megvalósított automatizált kritikus hibakombináció keresés

4.4.1. Hibák leírása

Tekintsük a hibát egy paraméter időbeli változásának. Ha ez a változás lineáris, akkor a hibát négy paraméterrel lehet leírni. Ez a négy paraméter és egy hiba bekövetkezése a 4.1. ábrán látható. A könnyebb használhatóság miatt a hibába célszerű belefoglalni, hogy melyik alkatrészre vonatkozik, így öt paraméterrel lehet leírni:

- `name`: az alkatrész neve, melyre a hiba vonatkozik
- `normal_value`: a paraméter értéke a hiba bekövetkezése előtt
- `error_value`: a paraméter értéke a hiba bekövetkezése után
- `t_min`: a meghibásodás kezdetének időpontja
- `t_max`: a meghibásodás teljes kialakulásának időpontja



4.1. ábra. A meghibásodás folyamata

Egy hiba leírása Pythonban Dictionary segítségével:

```
error = {
    'name' :          'RA1.R',
    'normal_value' :  100,
    'error_value' :   110,
    't_min' :         60,
    't_max':          90
}
```

Egy ilyen módon megadott hibából az `error_value()` függvénnyel lehet előállítani a paraméter értékét minden időpillanatban. A függvény bemenete a hiba és az idő.

```
def error_value(error, t):
    if t <= error['t_min']:
        return error['normal_value']

    if t >= error['t_max']:
        return error['error_value']

    dx = error['t_max'] - error['t_min']
    dy = error['error_value'] - error['normal_value']
    dt = t - error['t_min']

    return error['normal_value'] + dy / dx * dt
```

Egy rendszerben a lehetséges hibák listájának készítésekor fel kell sorolni az összes hibát, amely felléphet. Írjuk az összes alkatrész lehetséges hibáinak listáját egy tömbbe (`all_errors`):

```
all_errors = [
 {'name': 'RA1.R', 'normal_value': 100, 'error_value': 95,
  't_min': 60, 't_max': 80},

 {'name': 'RA1.R', 'normal_value': 100, 'error_value': 105,
  't_min': 75, 't_max': 90}

 {'name': 'RA2.R', 'normal_value': 100, 'error_value': 95,
  't_min': 60, 't_max': 80},

 {'name': 'RA2.R', 'normal_value': 100, 'error_value': 105,
  't_min': 75, 't_max': 90}
]
```

4.4.2. Hibakombinációk generálása

Hibakombinációk generálásához először el kell döntenünk, hogy egy hibakombináció hány hibát tartalmazzon (`error_count`), azaz hány alkatrész romoljon el. Ezután alkalmazhatjuk a `generate_error_combinations` függvényt a hibakombinációk generálásához. Ez a függvény figyelembe veszi, hogy egy alkatrész egyszerre csak egyféleképpen tud meghibásodni. (Nem lehet, hogy ha egy ellenállás meghibásodik, szakadásként és rövidzárként is viselkedik egyszerre.)

```python
def generate_error_combinations(all_errors, error_count):
    all_errors_groupped = []
    keys = []

    for i in all_errors:
        if i['name'] not in keys:
            keys.append(i['name'])
            all_errors_groupped.append([])
        all_errors_groupped[keys.index(i['name'])].append(i)

    error_combinations = []

    for i in list(itertools.combinations(all_errors_groupped, error_count)):
        error_combinations += list(itertools.product(*i))
    return error_combinations
```

Egy lehetséges, két hibát tartalmazó hibakombináció a következőképpen néz ki:

```
[
 {'name': 'RA1.R', 'normal_value': 100, 'error_value': 95,
  't_min': 60, 't_max': 80},

 {'name': 'RA2.R', 'normal_value': 100, 'error_value': 95,
  't_min': 60, 't_max': 80}
]
```

5. Modellezés és analízis

5.1. Az esettanulmány leírása

Vizsgálódásaimat a fűtőszál egyszerű modelljével kezdtem, amely egy szabályozó kapcsolóval is rendelkezik. Következő modellem egy hőmérsékletmérő áramkör, mely négy ellenállást tartalmaz. A harmadik esetben egy egyszerű termosztát modelljét építettem fel, melyben az előző két rendszer egy-egy alegységként szerepel, azok összekapcsolásával jött létre. Végül az utolsó, negyedik esetben kibővítettem a rendszert még két hőmérsékletmérő áramkörrel és egy vészkapcsolóval.

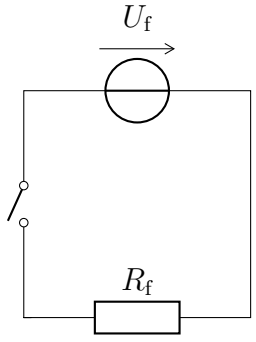
A létrehozott modelleket felhasználtam a lehetséges hibák azonosítására is. A rendszerek működésének vizsgálatakor célom volt az architektúrák biztonságosságának kiértékelése és a variánsok összehasonlítása. A rendszerek feladata, hogy egy ellenállás hőmérsékletét egy megadott intervallumban tartsák. Amennyiben ez nem lehetséges, kapcsoljanak ki, ezzel megakadályozva a túlmelegedést.

Azoknál a rendszereknél ahol a rendszer elméletileg képes helyesen működni még akkor is, ha egyes alkatrészek nem megfelelően működnek, van értelme a hibák hatását vizsgálni. Az esettanulmányban két módszert mutatok be.

5.2. Fűtőszál egyszerű modellje

Ebben az alfejezetben azt mutatom be, hogy az általam modellezett fűtőszál működik az idő múlásával. Először a fizikai modellt ábrázolom, majd bemutatom azt a szoftvert, amely a hőmérséklet értéke alapján szabályozza a rendszerben elhelyezett kapcsolót. A modell fizikai részét OpenModelicában hoztam létre, a szoftvert Pythonban írtam. Majd szimuláltam a modell működését. Végül a szimuláció eredményét is bemutatom grafikonnal szemléltetve.

5.2.1. Fizikai modell



5.1. ábra. Fűtőszál egyszerű modellje

A modellben (5.1. ábra) egy egyenáramú feszültségforrás (U_f), egy ellenállás (R_f) és egy kapcsoló szerepel. Az ellenállás értéke függ a hőmérséklettől az 5.1 egyenlet alapján.

$$R_\mathrm{f}(T) = R_\mathrm{f0}\left[1 + \alpha\left(T_0 - T\right)\right] \tag{5.1}$$

A modellben az ellenállás a környezetének hőt kétféleképpen tud leadni: hősugárzással és hővezetéssel.

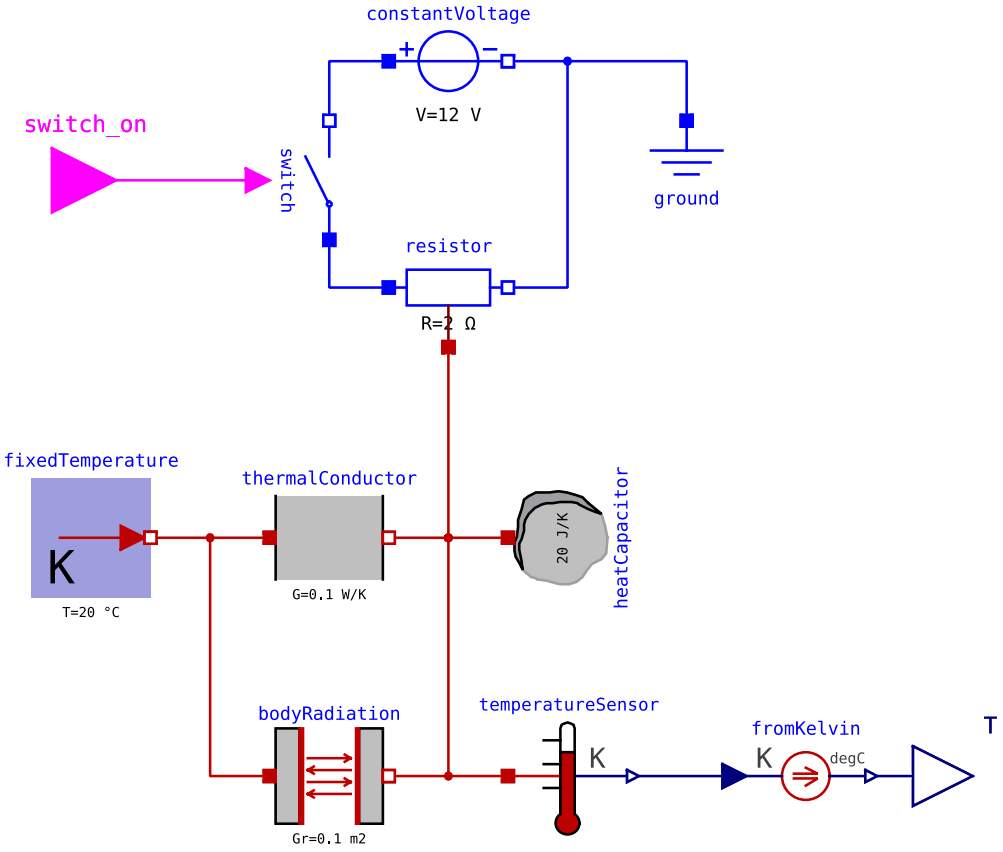
5.2.2. A modell szoftveres része

Célunk, hogy a hőmérsékletet adott intervallumban tartsuk: 60 °C és 70 °C között. Ennek érdekében a kapcsolót egy Python programmal kapcsoljuk ki és be. A program bemenete a hőmérséklet: `T`, és a kapcsoló jelenlegi állapota: `switch_state`, kimenete a kapcsoló állapota.

```
def switch_on(T, switch_state):
    T_min = 60
    T_max = 70

    if T > T_max:
        return False

    if T < T_min:
        return True

    return switch_state
```

5.2.3. A modell megvalósítása OpenModelicában

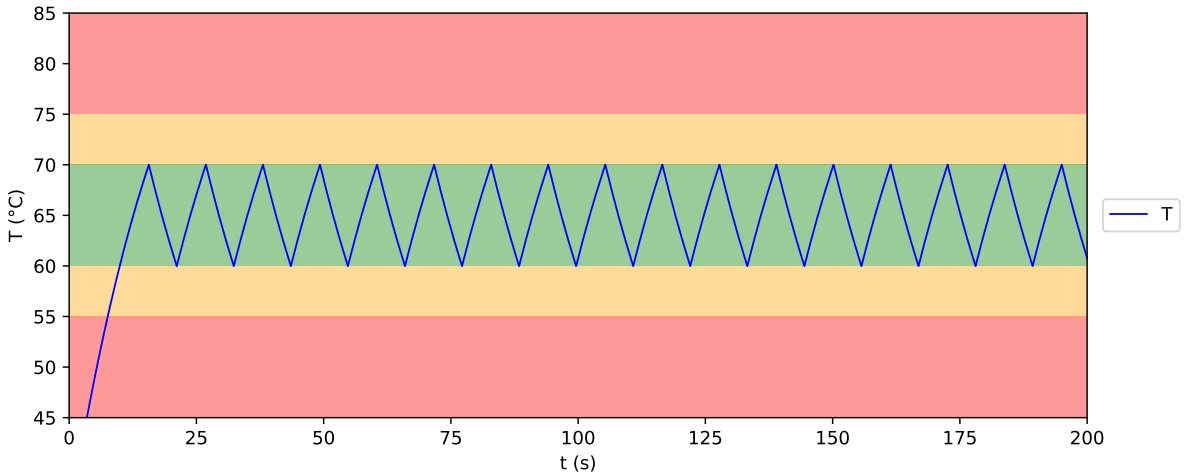
Az 5.2. ábrán a rendszer fizikai részének OpenModelicában megvalósított modellje látható. A modell kimenete: `T`, az ellenállás hőmérséklete, bemenete pedig a `switch_on`, a kapcsoló állapota. Az igaz érték a kapcsoló zárt, a hamis a nyitott állapotát jelenti.



5.2. ábra. A fűtőszál egyszerű modellje OpenModelicában

5.2.4. A szimuláció eredménye és kiértékelése

A szimulációval azt vizsgáltam, hogy a hőmérséklet hogyan változik az idő múlásával. Célom az volt, hogy a hőmérséklet egy adott intervallumban maradjon, ne lépje túl az elvárt értéket.



5.3. ábra. A hőmérséklet az idő függvényében

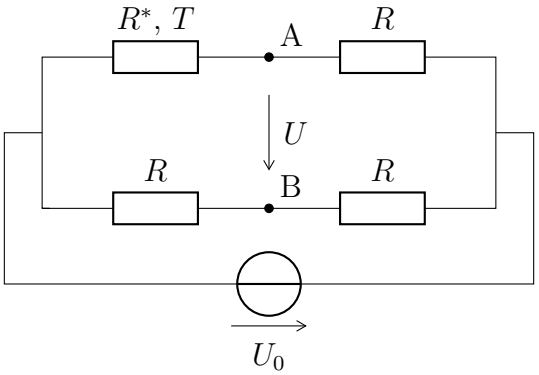
Az 5.3. ábrán az ellenállás hőmérséklete látható az idő függvényében. A szimulációt 200 másodpercig futtattam, mert ennyi idő elegendő ahhoz, hogy a rendszer működését megismerjük. Az ábrán látható zöld sáv a hőmérséklet optimális tartományát, a sárga sávok

a tűréshatárt jelölik. Ha a hőmérséklet a zöld sávba esik, az jó, ha a sárgába, az még elfogadható. Látható, hogy kezdetben a hőmérséklet az optimális értéknél jóval alacsonyabb. Az a rendszer kezdeti állapotából következik. Az is megfigyelhető, hogy miután a hőmérséklet elérte az optimális sáv felső határát, csökkenni kezd, majd amikor az optimális sáv alsó határához ér, ismét növekedni fog. Tehát a rendszer megfelelően működik, a szoftver jól kezeli a kapcsolót.

5.3. Hőmérsékletmérő áramkör

Ebben az alfejezetben egy hőmérsékletmérő áramkört mutatok be. Erre a modellre azért van szükség, mert a valóságban nehéz közvetlenül a hőmérsékletet mérni, egy feszültségmérés sokkal egyszerűbben megvalósítható. A mért feszültségből kiszámítható a hőmérséklet.

5.3.1. Fizikai modell



5.4. ábra. Hőmérsékletmérő áramkör

Az 5.4. ábrán a hőmérsékletmérő áramkör kapcsolási rajza látható. A modellben négy ellenállás van. T_0 hőmérsékleten az ellenállások érték azonos. A méréshez az R^* ellenállást és a mérendő objektumot termikus kapcsolatba hozzuk, így azok hőmérséklete meg fog egyezni. R^* ellenállás értéke a 5.2. egyenlet alapján függ a hőmérséklettől.

$$R^*(T) = R\left[1 + \alpha\left(T - T_0\right)\right] \tag{5.2}$$

A és B pont között mérjük a feszültséget, ebből az R^* ellenállás hőmérsékletet kiszámítható az 5.3. egyenlet alapján.

$$T(U) = \frac{2T_0U\alpha - 4U + T_0U_0\alpha}{2U\alpha + U_0\alpha} \tag{5.3}$$

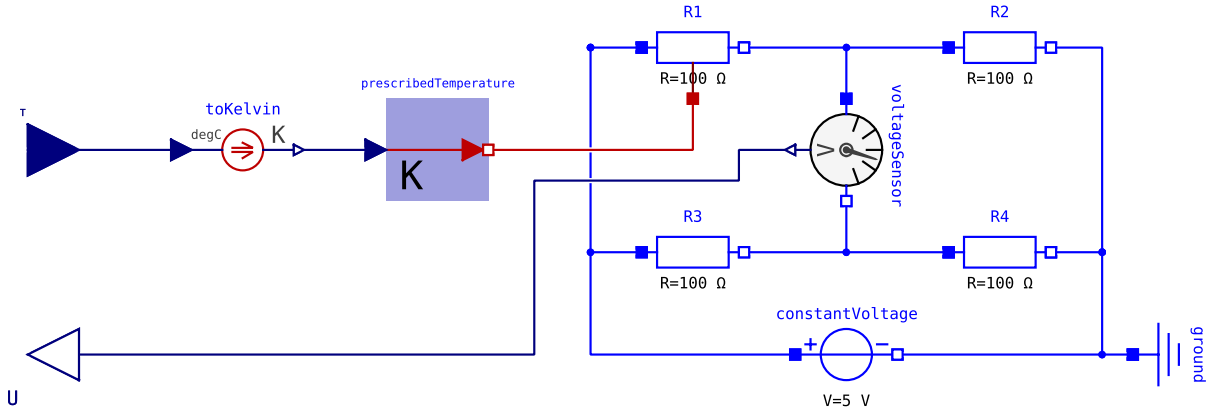
5.3.2. Modell szoftveres része

A modell szoftveres része egy függvényéből áll, amely elvégzi a hőmérsékletet kiszámítását az 5.3. egyenlet alapján. A függvény bemenete a mért feszültség voltban, visszatérési értéke a hőmérséklet °C-ban.

```
def T_calc_from_U(U):
    T0 = 27
    alpha = 0.004
    U0 = 5

    num = (2*T0*U*alpha - 4*U + T0*U0*alpha)
    den = (2*U*alpha + U0*alpha)
    return num / den
```

5.3.3. Modell megvalósítása OpenModelicában

A modell bemenete: `T`, a hőmérséklet, és kimenete az `U` feszültség.



5.5. ábra. Hőmérsékletmérő áramkör OpenModelicában

5.3.4. A szimuláció eredménye és kiértékelése

A szimulációt lefuttatva azt láttam, hogy a beállított és a feszültségből kiszámított hőmérséklet megegyezik.

5.4. Egyszerű termosztát

Ebben az alfejezetben az előzőekben bemutatott modellek kombinációjával hoztam létre egy új modellt. Ebben a modellben a kapcsoló már a hőmérséklet ténylegesen megmért értéke alapján működik. Célom most is az volt, hogy a hőmérséklet az elvárt tartományon belül maradjon. Miután a szimuláció igazolta a megfelelő működést, hibát injektáltam a rendszerbe és a hiba fellépésének hatását is vizsgáltam.

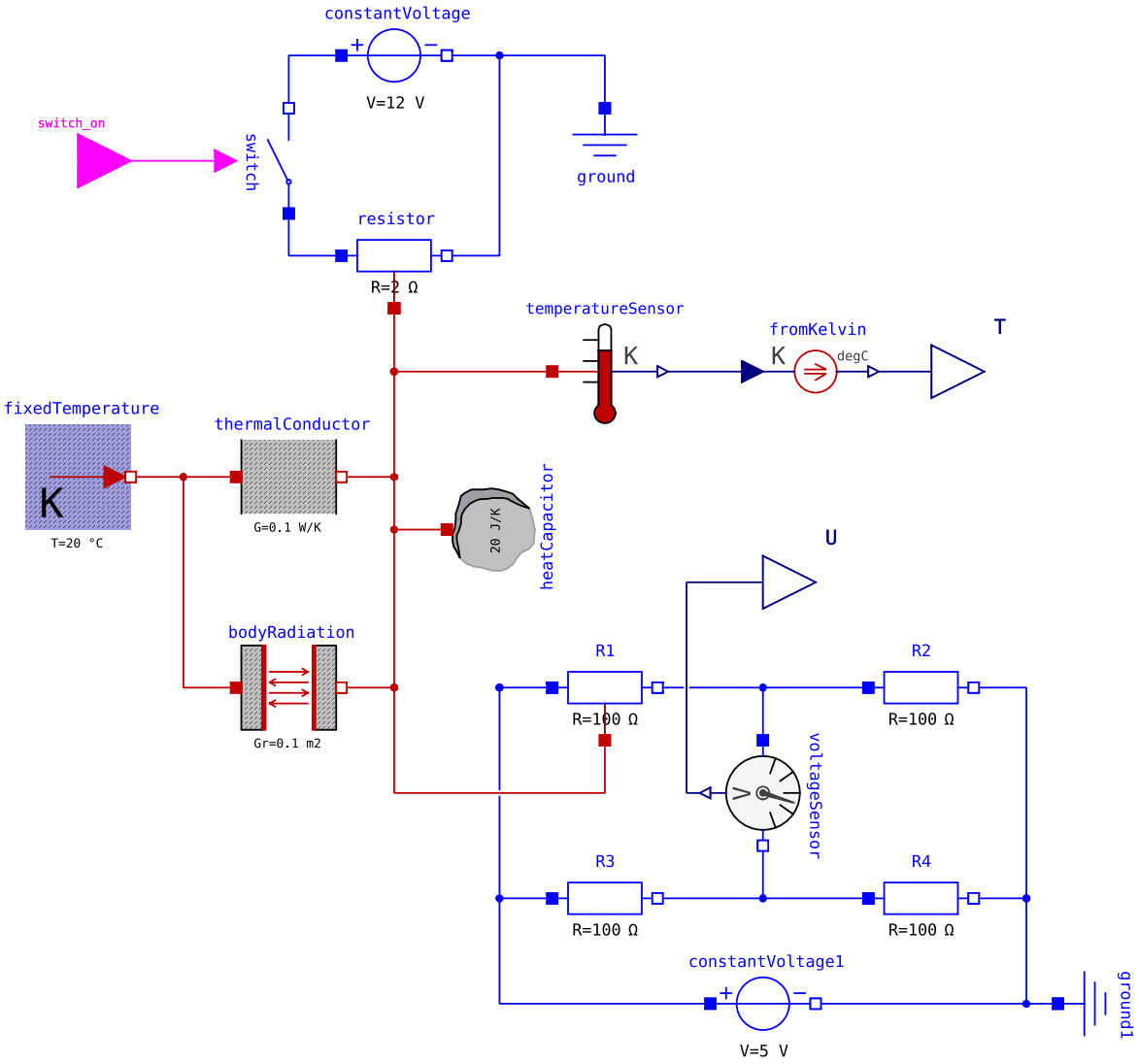
5.4.1. Fizikai modell

Az egyszerű termosztát fizikai modelljében a fűtőszál egyszerű modellje (5.2) és a hőmérsékletmérő áramkör (5.3) modellje kapcsolódik össze. A cél itt is a hőmérséklet megfelelő értéken tartása, azonban a hőmérséklet mért értéke alapján történik a fűtőszál kapcsolójának állítsa. Ezzel a modell sokkal jobban hasonlít egy valós fizikai rendszerhez.

5.4.2. Modell szoftveres része

A modell szoftveres része ugyanazon függvényeket tartalmazza, mint a fűtőszál egyszerű modellje (5.2) és a hőmérsékletmérő áramkör (5.3) modellje.

5.4.3. Modell megvalósítása OpenModelicában



5.6. ábra. Egyszerű termosztát OpenModelicában

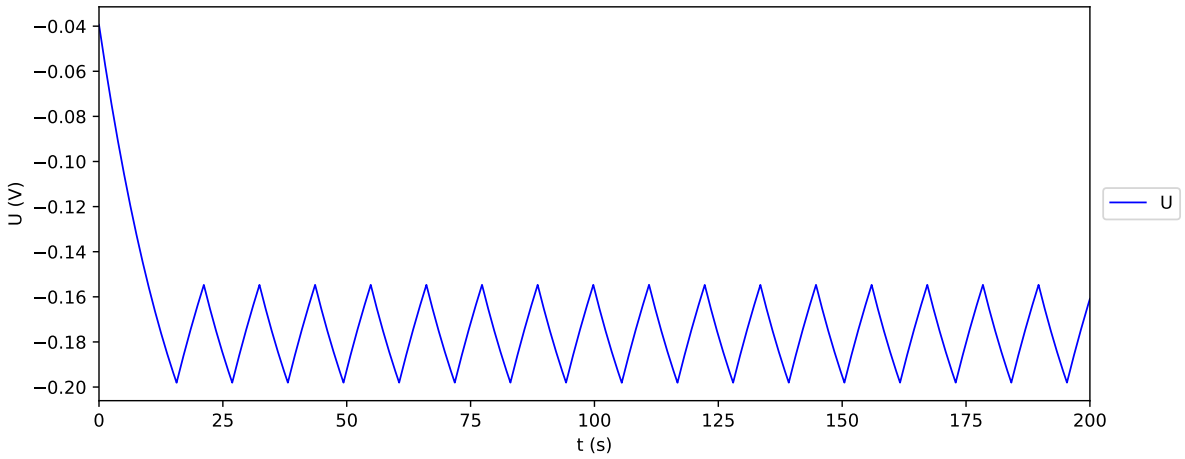
Az 5.6. ábrán az egyszerű termosztát megvalósítása látható OpenModelicában. A modell bemenete a kapcsoló állapota `switch_on`, kimenete a mért feszültség (`U`). Ellenőrzésképpen a tényleges hőmérséklet is kimenete a modellnek (`T`), a szoftveres rész azonban csak a mért feszültségre támaszkodik.

5.4.4. A szimuláció eredménye és kiértékelése

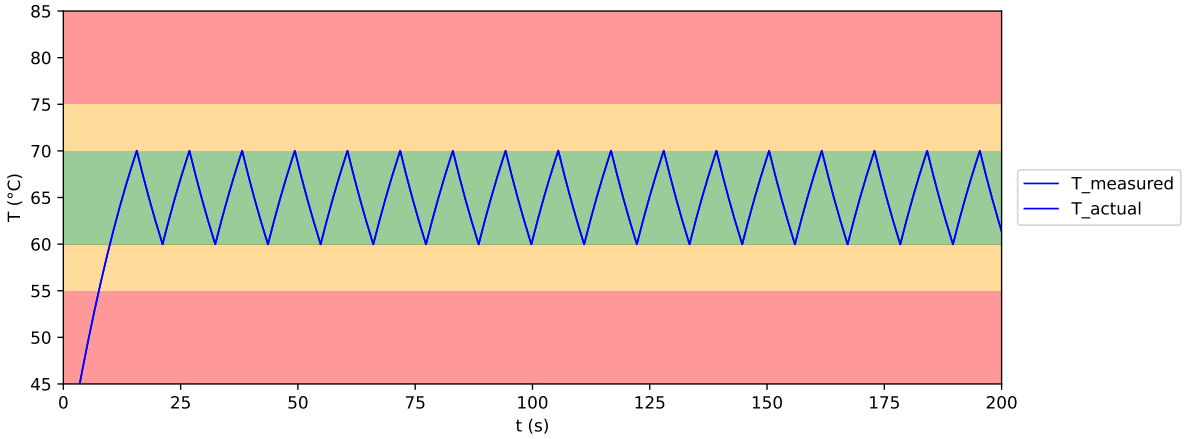
Az 5.7. ábrán a mért feszültség (`U`) látható az idő függvényében.

Az 5.8. ábrán a feszültségből számított hőmérséklet (`T_measured`) és a tényleges hőmérséklet (`T_actual`) látható. A két érték minden időpillanatban megegyezik egymással, tehát a hőmérsékletmérő áramkör megfelelően működik. A hőmérséklet megegyezik a 5.3. ábrán láthatóval, ez is várható volt.

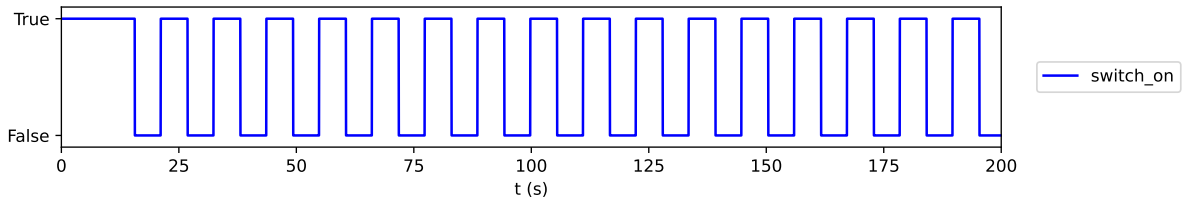
Az 5.9. ábrán a kapcsoló állapota látható az idő függvényében. Megfigyelhető, hogy amikor a kapcsoló be van kapcsolva (`True`), akkor a hőmérséklet emelkedik, amikor ki (`False`), akkor csökken.



5.7. ábra. A feszültség az idő függvényében



5.8. ábra. A hőmérséklet az idő függvényében



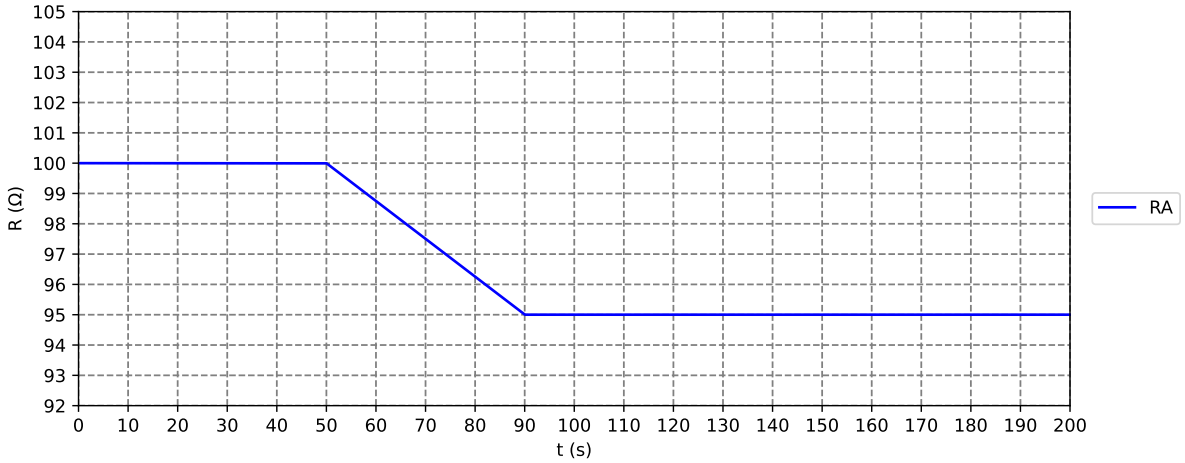
5.9. ábra. A kapcsoló állapota az idő függvényében

5.4.5. Hibajelenségek vizsgálata

Ebben a modellben már vizsgáltam az egyes alkatrészek meghibásodásának hatását. Ennél a szimulációnál a hőmérsékletmérő áramkörben lévő ellenállások meghibásodását vizsgáltam. Vizsgálatom a korábban bemutatott (4.1) három típusú meghibásodásra terjedt ki.

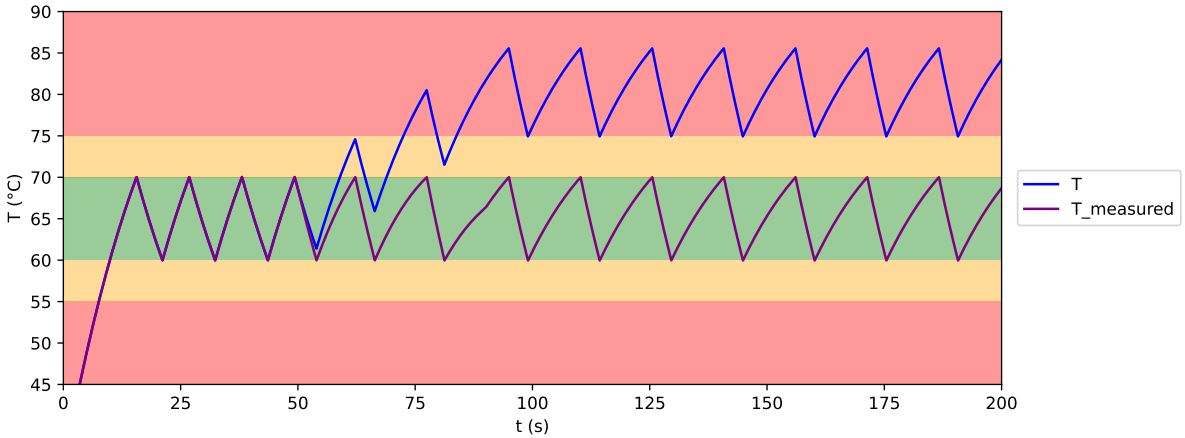
Itt a kevés hibalehetőség miatt ezt a hibák hatását a korábban bemutatott első módszerrel vizsgáltam, a hibákat egyesével, kézzel generáltam a rendszerbe (nem volt szükség az automatikus módszerre).

Az `R4` ellenállás értéket elkezdtem csökkenteni `t=50s`-nál, egészen addig, amíg `t=100s` időpillanatban elérte az eredeti érték 95%-át. Ez a folyamat a 5.10. ábrán látható.



5.10. ábra. Az `RA` ellenállás értéke az idő függvényében

Az 5.11. ábrán a rendszer tényleges hőmérsékletét (`T`), és a hőmérsékletmérő áramkör által mért értéket (`T_measured`) ábrázoltam. Látható, hogy a két érték eltér egymástól, miután az ellenállás meghibásodik. A rendszer `T_measured` érték alapján állítja a kapcsolót, ez azonban nem felel meg a valós hőmérsékletnek, az ennél magasabb.



5.11. ábra. A hőmérsékletek az idő függvényében

Tehát az egyszerű termosztát nem elég biztonságos, mert létezik olyan hiba, melynek hatására a rendszer túlmelegszik.

5.5. Biztonságos termosztát

Ebben az alfejezetben egy biztonságosabban működő modellt mutatok be. Olyan szimulációkat mutatok be, melyek igazolják hogy a rendszer sok esetben a hibák hatására is megfelelően működik. Azt is bemutatom hogy ez a rendszer se teljesen biztonságos, ugyanis az automatikus hibakereső rendszer talált olyan kritikus hibakombinációt, melynek hatására a rendszer túlmelegedett.

5.5.1. Fizikai modell

A modell az egyszerű termosztát továbbfejlesztett változata. Három hőmérsékletmérő áramkört és egy vészleállító kapcsolót is tartalmaz. A három mért hőmérsékletet a mo-

dell szoftveres része kiértékeli, és ez alapján állítja a kapcsolókat. A vészleállító kapcsoló működtetésekor a rendszer lekapcsol.

5.5.2. Modell szoftveres része

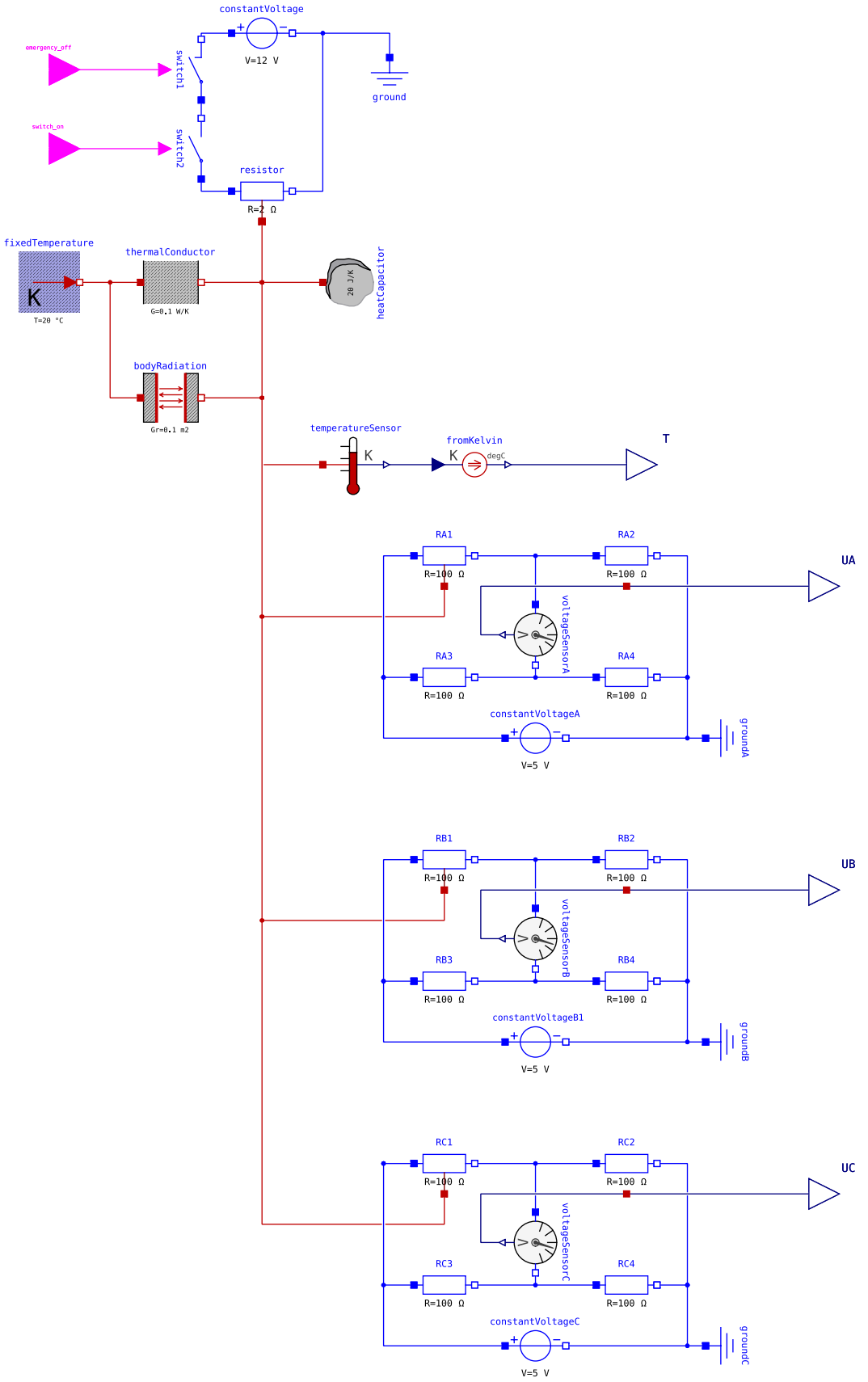
A modell szoftveres része részben ugyanazon függvényeket tartalmazza, mint a fűtőszál egyszerű modellje. Ezen felül tartalmazza még a `T_valiadte` függvényt is.

Ez a függvény a modellben szereplő három műszer által mért hőmérsékleti érték alapján megpróbálja kitalálni a tényleges hőmérsékletet és szabályozza a vészkapcsoló működését. Az egyik bemenetet a műszerek által mért adatok (`T_array`) képezik. A másik bemenet a `max_diff` érték, ami a mért hőmérsékleti adatokból számított szórás elfogadható maximumát jelöli. Kifejezi, hogy mennyire vagyunk szigorúak a programmal: ha magasabb ez a szám, akkor nagyobb eltérést engedünk, ha kisebb, akkor csak minimális eltérést fogadunk el.
A függvény visszatérési értékei:

- `T_guess`: a rendszer vélt hőmérséklete
- `T_state`: egy tömb, minden mérőműszerről tartalmazza
- `emergency_off`: megmondja hogy szükség van-e a vészkikapcsolásra

```
import numpy as np

def T_validate(T_array, max_diff):
    T_state = [True for i in range(len(T_array))]
    emergency_off = False

    if np.std(T_array) > max_diff:
        emergency_off = True
        for i in range(len(T_array)):
            if np.std(T_array[:i]+T_array[i+1:]) < max_diff:
                T_state = [True for i in range(len(T_array))]
                T_state[i] = False
                emergency_off = False
                break

    T_guess = np.average(T_array, weights=T_state)

    return T_guess, T_state, emergency_off
```

5.5.3. Modell megvalósítása OpenModelicában



5.12. ábra. A biztonságos termosztát OpenModelicában

5.5.4. A szimuláció eredménye és kiértékelése

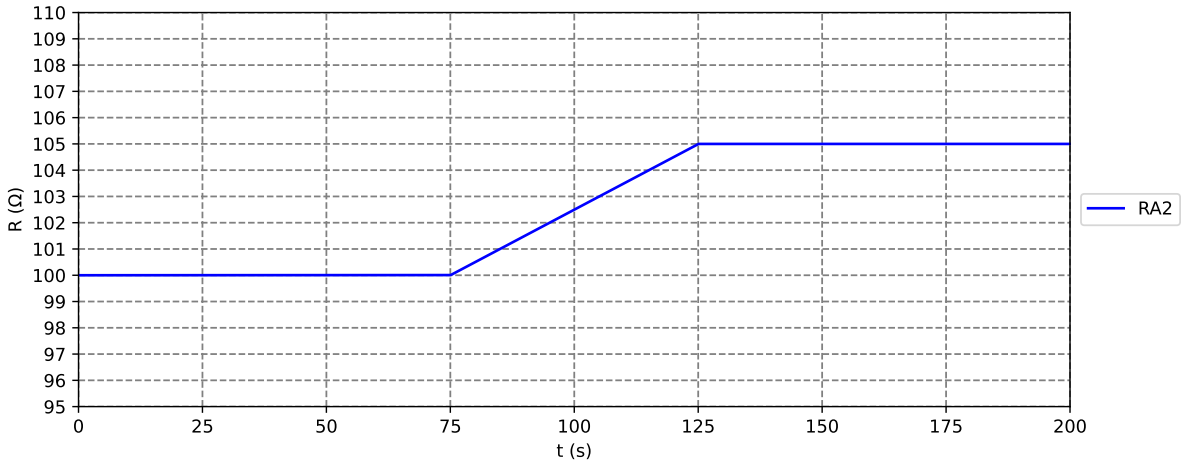
A szimulációkból kiderült, hogy a modell megfelelően működik. Az eredmények megegyeznek az 5.4.4. fejezetben bemutatottakkal.

5.5.5. Hibajelenségek vizsgálata hibák kézi generálásával

Ennél a modellnél szintén a hőmérsékletmérő áramkörökben lévő ellenállások meghibásodását vizsgáltam a korábban ismertetett három lehetséges hibát feltételezve.

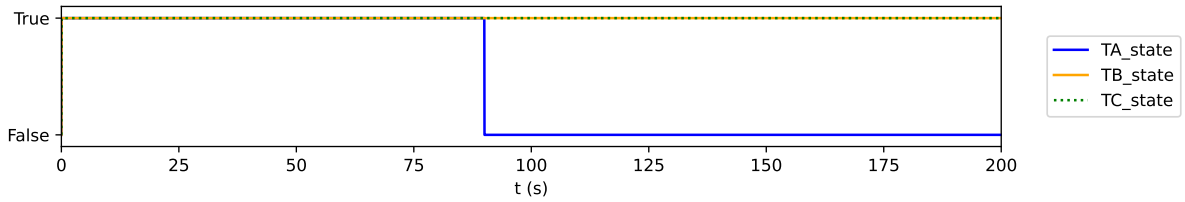
5.5.5.1. Első hibajelenség

Ebben a példában az OpenModelicában megvalósított modellben (5.12. ábra) az `RA2` jelű ellenállás értékét `t=75s`-nél elkezdtem növelni az idő függvényében lineárisan úgy, hogy `t=125s` időpontban érje el az 5%-os növekedést. Az ellenállás az idő függvényében az 5.13. ábrán látható.



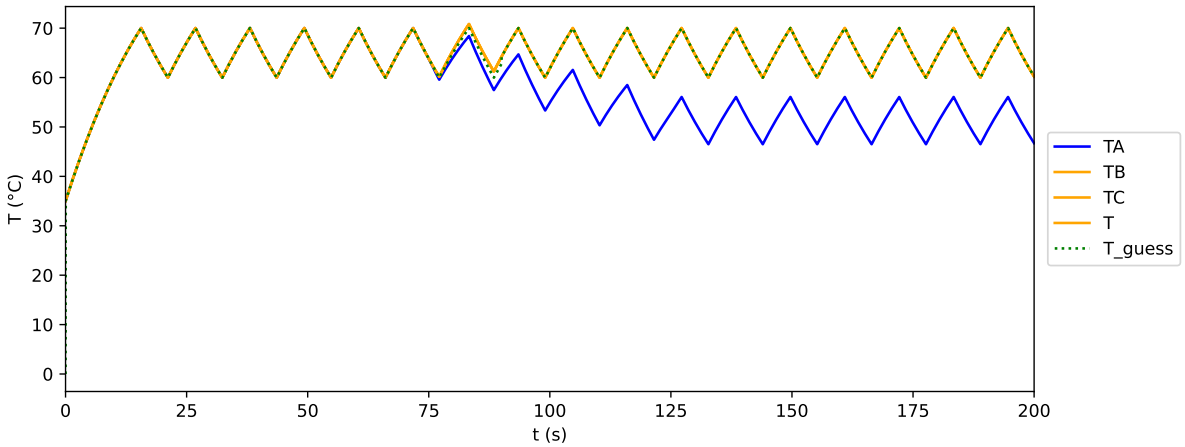
5.13. ábra. Az `RA2` ellenállás értéke az idő függvényében

A 5.14. ábrán látható, hogy a `T_validate` függvény az `A` hőmérsékletmérő áramkört egy idő után megbízhatatlannak nyilvánította. Az is látható, hogy nem egyből, amint az ellenállás elkezdett megváltozni, hanem csak `t=90s` környékén.



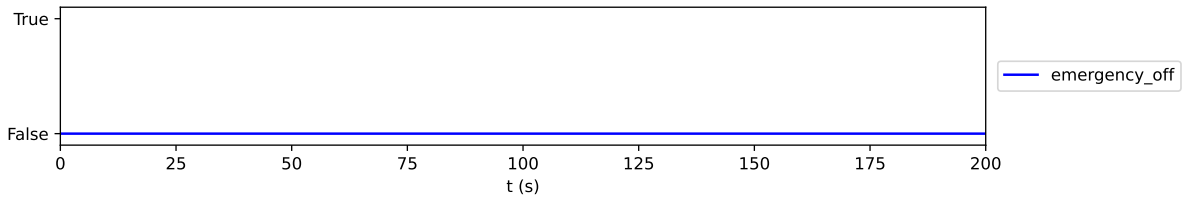
5.14. ábra. A hőmérsékletmérő áramkörök megbízhatósága

Az 5.15. ábrán a hőmérsékletek láthatók az idő függvényében. Megfigyelhető, hogy a szimuláció elején valamennyi érték egybeesik. Látható, hogy amikor fellép a hiba, `TA` érték elkezd eltérni a többi műszer által mutatott értéktől. Az is megfigyelhető, hogy a hiba jelentkezésekor `T_guess` is eltér `T` értéktől, majd kis idő múlva az eltérés megszűnik. Ez a jelenség azzal magyarázható, hogy a `T_validate` függvény először még nem nyilvánította megbízhatatlannak a meghibásodott műszert.



5.15. ábra. A hőmérsékletek az idő függvényében

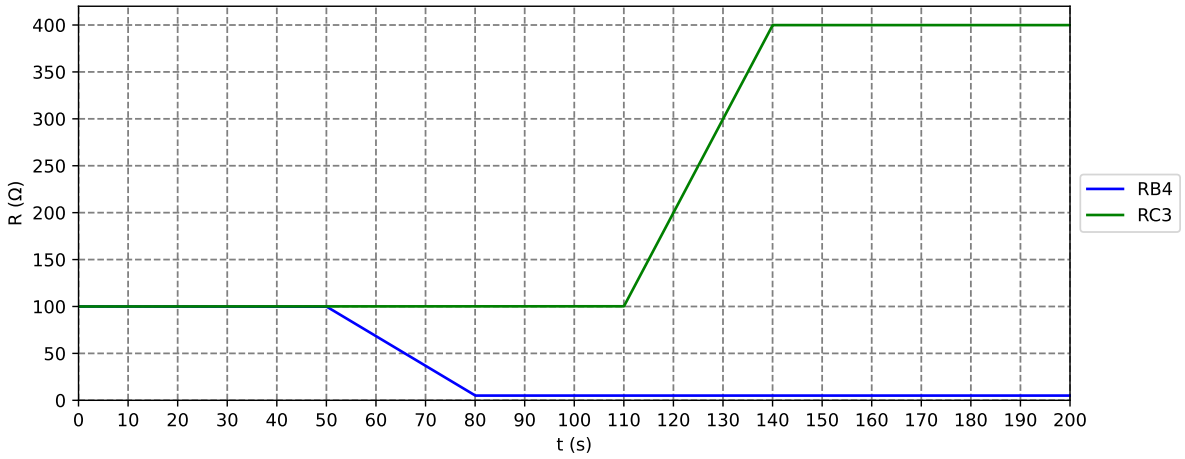
Az 5.16. ábrán látható, hogy a `T_validate` függvény nem tartotta indokoltnak a rendszer vészkikapcsolást.



5.16. ábra. A vészkikapcsoló állapota

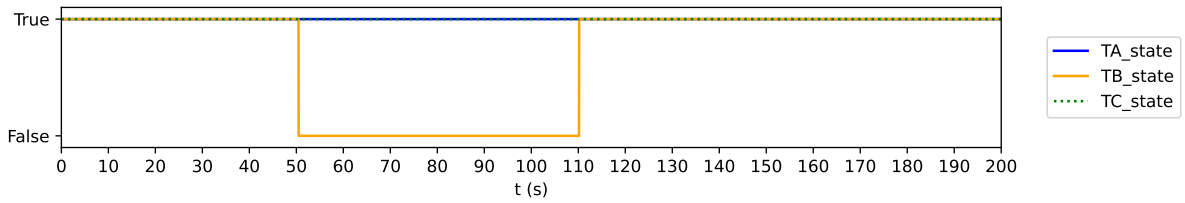
5.5.5.2. Második hibajelenség

A második a példában két hiba együttes előfordulását vizsgáljuk: az `RB4` jelű ellenállás értékét `t=50s`-nél elkezdtem csökkenteni, és `t=80s`-ra szinte nullává válik. Azután a `RC3` ellenállás érték növelni kezdem `t=110s`-nál, és 30s alatt egészen 400Ω-ig növelem. Az ellenállások értéke az 5.17. ábrán látható az idő függvényében.

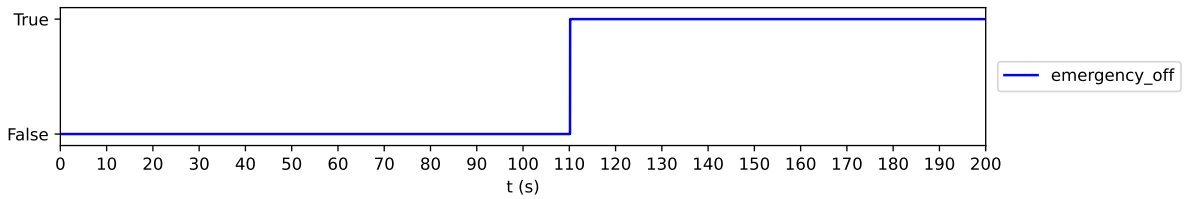


5.17. ábra. Az `RB4` és `RC3` ellenállás értéke az idő függvényében

Az 5.18. ábrán látható, hogy a `T_validate` függvény helyesen megállapította, hogy `TB` a `t=50s` időpont után meghibásodott, ezért nem megbízható. A második hiba bekövetkezik `t=110s`-nál, ezután `T_validate` mindhárom műszert megbízhatónak jelöli, ennek azonban itt már nincs jelentősége, mert ebben az időpontban bekapcsolta a vészleállítást, ahogy az a 5.19. ábrán látható.

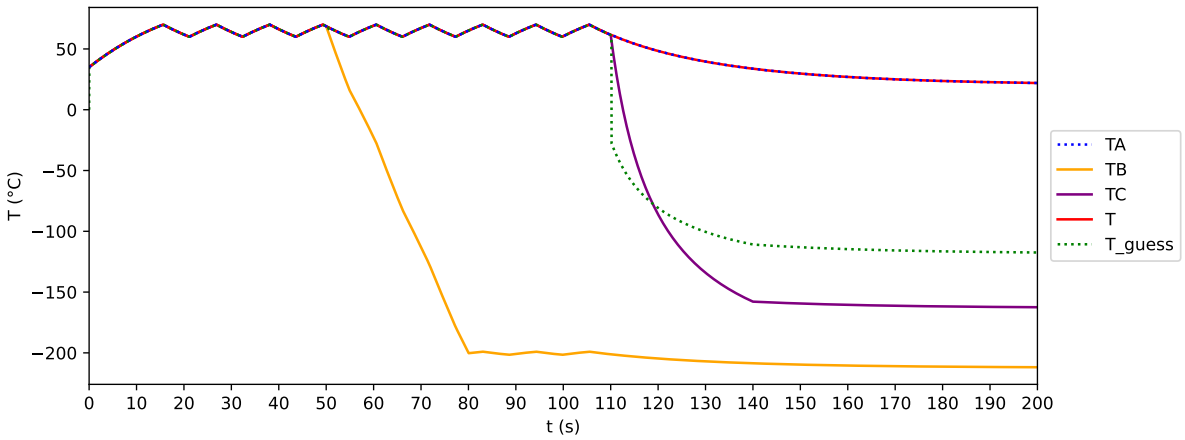


5.18. ábra. A hőmérsékletmérő áramkörök megbízhatósága



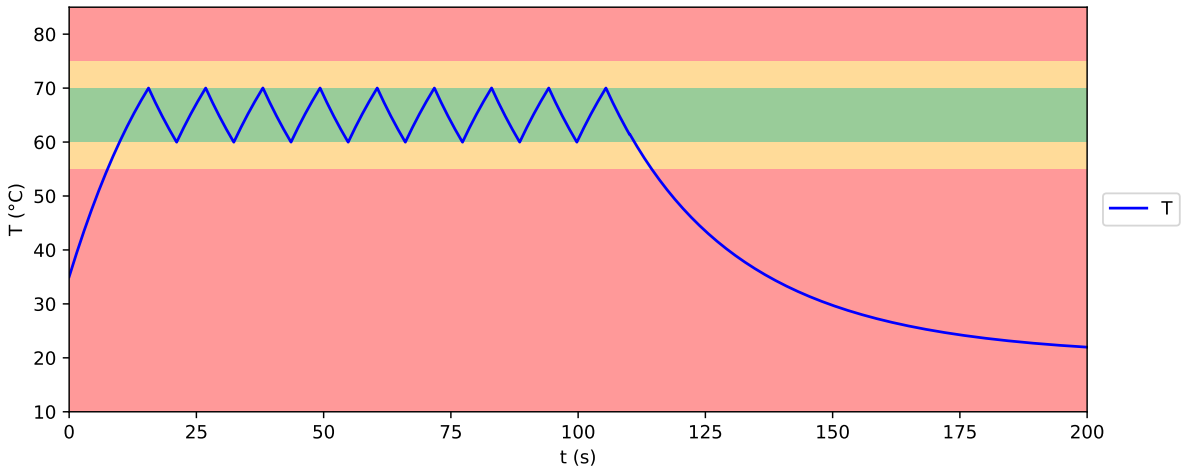
5.19. ábra. A vészkikapcsoló állapota

Az 5.20. ábra a hőmérsékleteket mutatja az idő függvényében. A szimuláció elején minden érték megegyezik. `t=50s`-tól a `TB` érték eltér, ezt a rendszer helyesen érzékeli és innentől nem is veszi figyelembe. A második hiba kezdetekor `TC` érték is eltér a valóstól, így a rendszer működteti a vészkikapcsolót. Ez helyes lépés, hiszen itt már a `T_guess` is eltér a valódi értéktől, így a hőmérséklet szabályozása nem lehetséges.



5.20. ábra. A hőmérsékletek az idő függvényében

Az 5.21. ábra a rendszer hőmérsékletét ábrázolja az idő függvényében. Látható, hogy a kezdeti felmelegedés után a második hiba megjelenéséig minden rendben van, utána a hőmérséklet lecsökken.



5.21. ábra. A rendszer hőmérséklete az idő függvényében

Látható, hogy ez a rendszer már biztonságosabb az előzőnél. Egy ellenállás meghibásodása esetén még normál üzemben működött. Amikor egy másik hőmérsékletmérő áramkörben lévő ellenállás is meghibásodott, észlelte hogy a mért hőmérsékletek helytelenek, és a vészkikapcsolót használva leállította magát, ezzel megakadályozva a túlmelegedést.

5.5.6. Automatizált kritikus hibakombinációk keresése

Ebben a részben a biztonságos termosztát három hőmérsékletmérő áramkörében elhelyezett 12 ellenállás lehetséges kritikus hibakombinációit kerestem a 4.4. alapján.

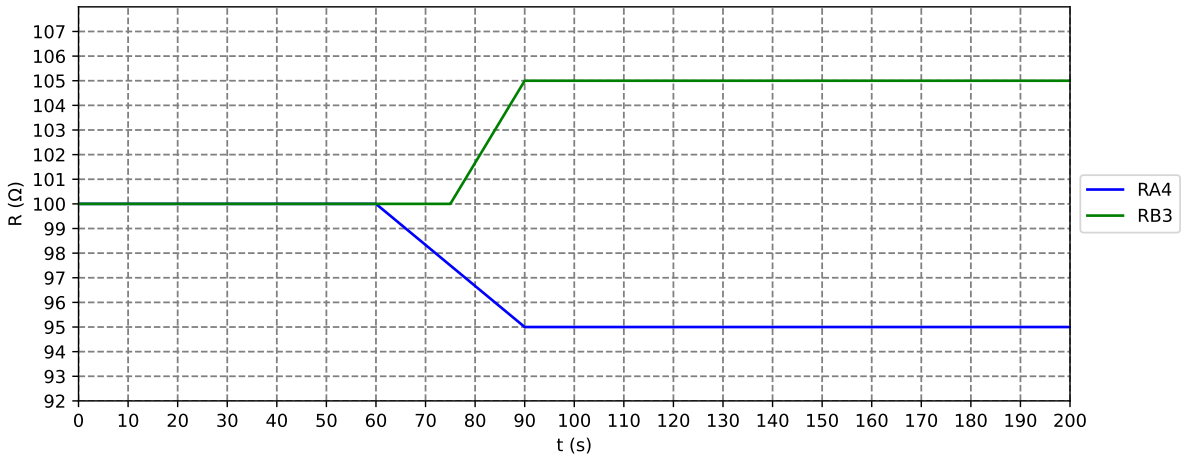
Első lépésként felvettem mind a 12 ellenállásra az összes lehetséges hibát, ellenállásonként négyet. Ezután legeneráltam a lehetséges hibakombinációkat `error_count=1` értékkel. Ezekkel a hibakombinációkkal futtatva a szimulációt megállapítható volt, hogy a rendszer minden esetben megfelelően működött.

A hibakombinációkat `error_count=2` értékkel is legeneráltam. Újabb szimuláció futtatások után kiderült, hogy itt már vannak olyan hibakombinációk, melyek hatására a rendszer hibásan működik, a hőmérséklet a megengedett értéknél nagyobb lesz. Ezek közül egy példát most részletesebben is bemutatok.

A kritikus hibakombináció leírása Pythonban:

```
error_combination = (
{'name': 'RA4.R', 'normal_value': 100, 'error_value': 95,
 't_min': 60, 't_max': 90},

{'name': 'RB3.R', 'normal_value': 100, 'error_value': 105,
 't_min': 75, 't_max': 90}
)
```

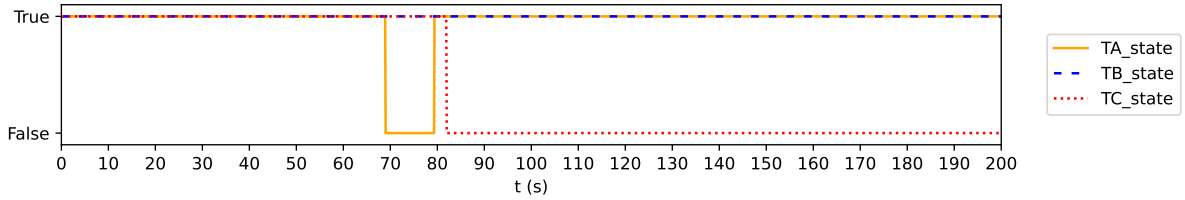
A meghibásodott ellenállások értéke az idő függvényében az 5.22 ábrán látható. Kezdetben `RA4` és `RB3` ellenállások értéke 100Ω, majd `RA4` ellenállás értéke 5%-kal csökken, és `RB3` ellenállás értéke 5%-kal nő.



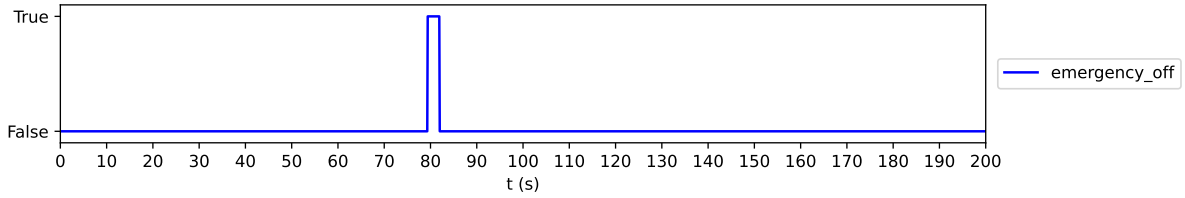
5.22. ábra. A meghibásodott ellenállások értéke az idő függvényében

Az 5.23. ábrán a hőmérsékletmérő áramkörök megbízhatósága látható. Megfigyelhető, hogy eleinte mindhárom műszert helyesnek minősíti a rendszer. Majd `t=69s` körül az `A` hőmérsékletmérő áramkört a rendszer helyesen hibásnak jelöli meg. Kis idő elteltével, `t=79s` körül a rendszer aktiválja a vészkikapcsolót. Ez az 5.24. ábrán látható. Eddig a rendszer megfelelően működött.

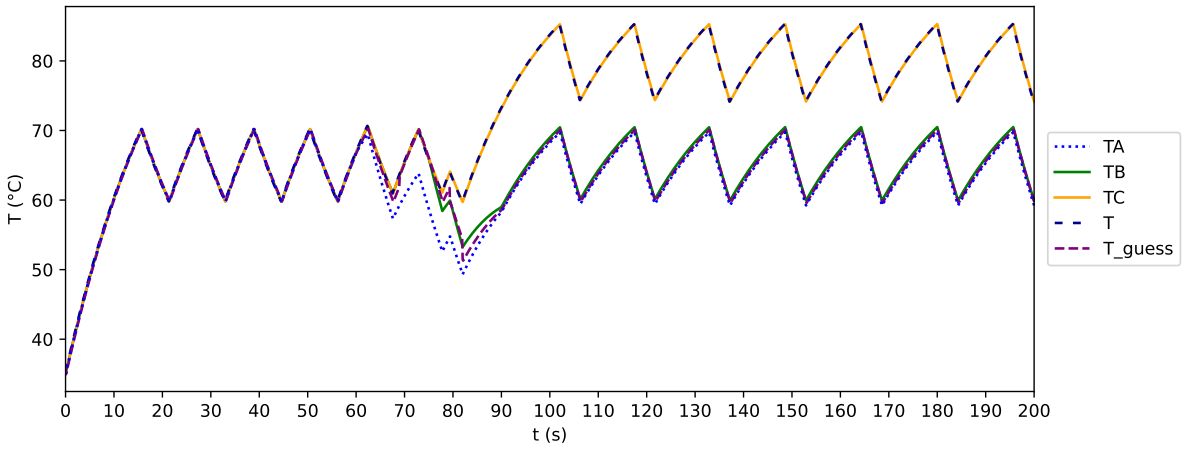
A hibás működést az idézi elő, hogy `t=82s` körül a hibás `A` és `B` hőmérsékletmérő áramkörök által mutatott értékek nagyon közel kerülnek egymáshoz, de eltérnek a rendszer tényleges hőmérsékletétől. Ez a 5.25. ábrán látható. Ennek hatására a rendszer visszavonja a vészkikapcsolást, hibásnak jelöli meg a helyesen működő `C` hőmérsékletmérő áramkört, és a hőmérsékletet a hibás `A` és `B` műszer alapján szabályozza.



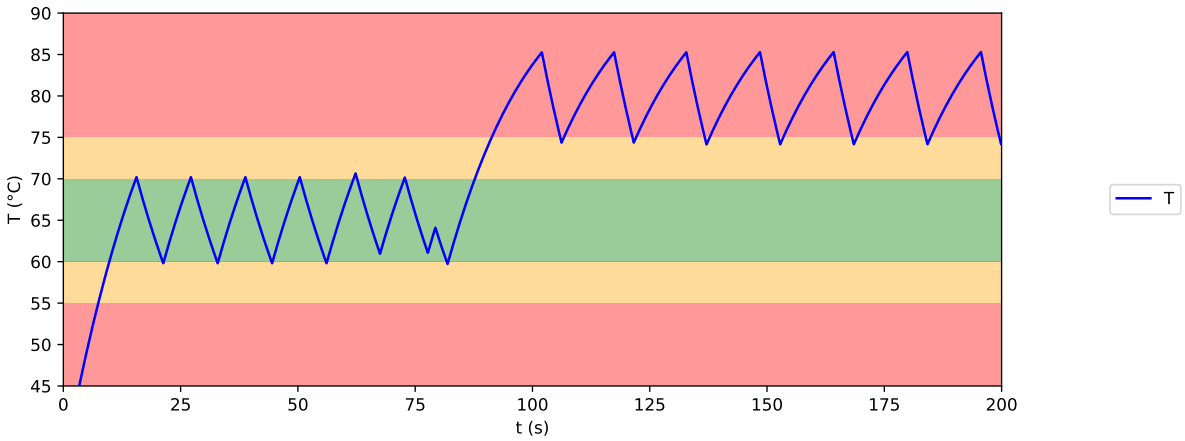
5.23. ábra. A hőmérsékletmérő áramkörök megbízhatósága



5.24. ábra. A vészkikapcsoló állapota



5.25. ábra. A hőmérsékletek az idő függvényében



5.26. ábra. A rendszer hőmérséklete az idő függvényében

Az 5.26. ábrán a rendszer hőmérséklete látható az idő függvényében. Megfigyelhető hogy a hibajelenségek bekövetkezése előtt rendben van, utána pedig magasabb a megengedettnél. Az automatizált kritikus hibakombináció kereső program jól működött, a rendszerben kritikus hibakombinációkat talált.

6. Összefoglalás és az eredmények értékelése

Megtapasztaltam, hogy a hagyományos rendszertervezéssel szemben a modellalapú rendszertervezésnek számos előnye van. Az összetett rendszereket egyszerűsítve tudjuk ábrázolni, könnyen tudjuk rögzíteni a modellben a rendszer legfontosabb jellemzőit. Változtatni tudjuk a különböző paramétereket, így elemezhetjük a rendszer viselkedését. A létrehozott modellt felhasználtam a rendszer viselkedésének szimulálására és a lehetséges problémák, hibák azonosítására. Célom az volt, hogy növeljem a megbízhatóságot. További előny, hogy a létrehozott modell alkalmas arra, hogy mások is megtekintsék, tanulmányozzák, fejlesszék, ami segíti a kooperációt, a hatékonyságot.

Az általam felépített négy modellen keresztül bemutattam, hogy a megbízhatósági vizsgálatok eredményei a modellek működésének finomításához, esetleg új modellek létrehozásához nyújtanak információt. Koszimulációs környezetben a bonyolultabb rendszerek is hatékonyan vizsgálhatók. A hibavizsgálat már a folyamat elején olyan támpontokat nyújt, melyekkel olcsóbban tudjuk a rendszert javítani, a jó működést garantálni.

Az elvégzett munka a fő hozadéka számomra az volt, hogy megismertem a rendszertervezés fő folyamatait és egy egyszerű, koszimuláció-alapú szolgáltatásbiztonsági analízist is elvégeztem. Olyan innovatív rendszertervezési megközelítést alkalmaztam, amely lehetővé tette a szoftver és hardver komponensek hibatűrésének verifikációját a szoftveres, hardveres és hibamodellek koszimulációjának segítségével.

További haszna volt számomra a dolgozat elkészítésének, hogy jártasságot szereztem a témához kapcsolódó magyar és angol nyelvű szakirodalomban.

Irodalomjegyzék

[1] Barbierato, Luca and Rando Mazzarino, Pietro and Montarolo, Marco and Macii, Alberto and Patti, Edoardo and Bottaccioli, Lorenzo: A comparison study of co-simulation frameworks for multi-energy systems: the scalability problem. *Energy Informatics*, 5. évf. (2022) 4. sz., 1–26. p.

[2] Clarke, Edmund and Grumberg, Orna and Jha, Somesh and Lu, Yuan and Veith, Helmut: Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50. évf. (2003) 5. sz., 752–794. p.

[3] Clarke, Edmund M: The birth of model checking. In *25 Years of model checking: history, achievements, perspectives*. 2008, Springer, 1–26. p.

[4] Di Maio, Dario and Stramaccioni, Elena and Misul, Daniela Anna and Napolitano, Pierpaolo and Beatrice, Carlo: A Multiphysics Co-Simulation Framework of a Gas Engine and Three-Way Catalyst toward a Complete Vehicle Design Model. *Machines*, 10. évf. (2022) 10. sz., 852. p.

[5] Paul M Frank – Birgit Köppen-Seliger: Fuzzy logic and neural network applications to fault diagnosis. *International journal of approximate reasoning*, 16. évf. (1997) 1. sz., 67–88. p.

[6] Heusser, Peter Andreas: Modelling and simulation of boiling channels with a general front tracking approach. *Doctoral Thesis*, 1998.

[7] Jhala, Ranjit and Majumdar, Rupak: Software model checking. *ACM Computing Surveys (CSUR)*, 41. évf. (2009) 4. sz., 1–54. p.

[8] McMillan, Kenneth L: Applications of Craig interpolants in model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (konferenciaanyag). 2005, Springer, 1–12. p.

[9] Nasir Mehranbod – Masoud Soroush – Chanin Panjapornpon: A method of sensor fault detection and identification. *Journal of Process Control*, 15. évf. (2005) 3. sz., 321–339. p.

[10] Modelon: Co-simulation using the open-source Python package PyFMI. *Modelon newsletter*, 2023.

[11] Modelon: FMI Standard Understand the two types of Functional Mosk-up Units. *Modelon newsletter*, 2023.

[12] URL `https://openmodelica.org/`.

[13] Alexander Pretschner – Manfred Broy – Ingolf H Kruger – Thomas Stauner: Software engineering for automotive systems: A roadmap. In *Future of Software Engineering (FOSE'07)* (konferenciaanyag). 2007, IEEE, 55–71. p.

[14] URL `https://www.python.org/`.