Szegedi Tudományegyetem

Informatikai Intézet

IoT eszközök energiafelhasználásának mérése

Diplomamunka

Készítette:

Vince Dániel

Programtervező informatikus MSc szakos hallgató

Témavezető:

Dr. Kiss Ákos

adjunktus

Lóki Gábor

PhD hallgató

Szeged

2019

Feladatkiírás

Az iparban évtizedek óta kiforrott technológia a beágyazott rendszerek használata. Beágyazott eszközök találhatók az autókban, gyárakban, nyersanyagfeldolgozó telepekben és háztartásokban. Ezen rendszerek feladata szenzoradatok olvasása, ezek alapján döntéshozás és beavatkozás, azaz a befogadó rendszer vezérlése. Az Internet of Things fogalom 2010 környékén terjedt el világszerte, mely lehetővé tette a beágyazott rendszerek szélesebb körű elterjedését, megjelentek a Raspberry Pi és az Arduino sorozat elemei.

Az iparban használt szoftverek nagy része C/C++ programozási nyelven íródott, annak hatékonyságát kiaknázva, viszont az Internet of Things által meghódított közönség nem feltétlenül rendelkezik célirányos tudással ezen a területen. Ezt áthidalva lehetőség van több, különböző magasszintű programozási nyelvet használni, mint pl. a Python és a JavaScript.

Az iparban használt szoftverek tervezése során hangsúlyt fektetnek a készülő eszköz energiahatékonyságára, hiszen a vezérlési folyamatot megszakítás nélkül kell ellátni a hét minden napján. Felmerül a kérdés, hogy az iparban elterjedt technológiák mellett Python és JavaScript nyelven írt programok milyen energiahatékonysági mutatókkal rendelkeznek.

A hallgató feladata egy energiamérő modul elkészítése, mely képes mérni az eszközök energiafogyasztását és menteni azt későbbi feldolgozásra. Az energiamérő egységnek tudnia kell kommunikálni a mérendő eszközzel, lehetővé téve szoftverfejlesztési konstrukciókat célzó mérések végzését. Használati eseteket állít fel, melyeket a fent említett nyelvek segítségével megvalósít és lemér, ily módon a különböző nyelveken írt programok összehasonlíthatók, az összehasonlítást a hallgató elvégzi. Ezen felül megvizsgálja a különböző futtatókörnyezetek hatását az energiafogyasztásra (Python 3, IoT.js (JerryScript)).

Tartalmi összefoglaló

- **Téma megnevezése:**

IoT eszközök energiafogyasztásának vizsgálatát támogató eszköz tervezése, megvalósítása és validálása. Az eszköz segítségével különböző programozási nyelveken írt alkalmazások fogyasztásának összehasonlítása.

- **A feladat megfogalmazása:**

Fogyasztásmérő eszköz megtervezése és kivitelezése. A szükséges eszközök beszerzése, megfelelő illesztése. A meghajtóprogramok implementálása, a mérési eredmények könnyű elérhetőségének biztosítása. Az elkészült eszköz segítségével egy népszerű IoT projekt modellezése és megvalósítása C++, JavaScript és Python nyelveken, a megvalósítások összehasonlítása energiafogyasztás szempontjából.

- **Megoldási mód:**

A mérések végrehajtását szolgáló Unix háttérszolgáltatás Python nyelven került megvalósításra, az eredmények egy lokális webszerveren elérhetők. A mérések programozottan indíthatók, ez ellenőrzésre került a felhasználói eset megvalósítása közben mindhárom programozási nyelven. A felhasználói eset mellett három Unix teljesítményteszt is vizsgálva lett: *Sysbench, Polybench* és *Iperf3*. A nyers mérési adatokat feldolgozó szkriptek is Python nyelven kerültek megvalósításra.

- **Alkalmazott eszközök, módszerek:**

A mérő egység alapja egy Raspberry Pi 3B+ számítógép és egy INA219 árammérő szenzor. A mért eszköz egy Raspberry Pi 3B volt SenseHat bővítőpanellel kiegészítve, a dolgozatban szereplő eredmények az eszközök méréséből származnak. Az alkalmazások a nyílt forráskódú Visual Studio Code integrált fejlesztői környezetben, a kapcsolási rajzok a Fritzingben készültek. Az eredmények elérhetőségét a *ProfFTP* alkalmazás biztosítja.

- **Elért eredmények:**

A fogyasztásmérő eszközt és a felhasználói eset modellezését megvalósítottam. Mind a saját alkalmazások, mind a teljesítménytesztek méréseit és az eredmények összehasonlítását elvégeztem. Megvizsgáltam a GCC fordító optimalizációs kapcsolóinak hatását a fogyasztásra.

- **Kulcsszavak:**

energiafogyasztás mérése, beágyazott rendszerek, IoT, JavaScript, Python, C++, GCC

Tartalomjegyzék

Táblák, ábrák jegyzéke

1. Bevezetés

Szoftverek tervezése és megvalósítása során sokszor megemlítik a nem funkcionális követelményeket, vagyis a készülő rendszernek hogyan kell megvalósítania a kívánt funkcionalitást. Ezen követelmények ritkán vonatkoznak a rendszer egyedi összetevőire, inkább az eredendő tulajdonságait írják le, mint pl. teljesítményt, tárhelyigényt és biztonságot [1], viszont csak ritkán esik szó az energiafelhasználás fontosságáról.

Alapvetően az energiahatékonyság az a képesség, amely lehetővé teszi, hogy többet érjünk el kevesebből [2]. Amikor a szoftverek, és ezáltal a számítógépek energiafelhasználásáról esik szó, legtöbbünknek az adattárházak és szerverparkok jutnak eszébe, és nem alaptalanul. Egy-egy ilyen park több millió megawattóra (MWh) energiát fogyaszt évente [3], amely gazdasági és környezeti problémákat hordoz magában. Ennek fényében a szerverparkok üzemeltetői tisztában vannak azzal, hogy milyen következményei vannak, ha rosszul optimalizált szoftvereket futtatnak. Ezért fejlesztések is történnek a fogyasztás mérésére és csökkentésére [4]. Egy kicsit tovább gondolva a problémát eljuthatunk a zsebünkben lapuló okostelefonhoz vagy az ölünkben tartott laptophoz. Ezeket az eszközöket mindennap használjuk mobilan, tehát nem a hálózati töltőcsatlakozón élik életüket. Mivel az akkumulátoridejük véges, már saját személyünkhöz közelibbnek érezhetjük a szoftverek energiafogyasztásának kérdését. Egy rosszul megírt program nem okoz többmilliós kiadást a felhasználónak, még akkor sem, ha gyorsabban meríti az akkumulátort, viszont bosszúságot okozhat, ha nem tudja befejezni a munkáját időben, vagy lemerül a telefonja hazafelé. A cégek legtöbbször nem a zavaró ok forrását szüntetik meg, hanem egyszerűen nagyobb akkumulátort helyeznek az eszközeikbe.

A legtöbb ember itt megáll. Mivel a környezetünkben elhelyezkedő beágyazott rendszerek nagy része láthatatlan a laikusok számára, így erről a rétegről nem is tudhatnak. Ez a tény viszont nem szünteti meg létezésüket. Egy–egy kis eszköz elhanyagolhatónak tűnik, viszont összeadva a milliónyi eszközt már jelentős energiafogyasztást kapunk. Gyárakat, nyersanyagfeldolgozó üzemeket, a közértet, vagy akár a saját házunkat szemlélve, találkozhatunk beágyazott rendszerekkel. Ezen elemek egy nagy egész kis részét képezik, feladatuk adatokat gyűjteni szenzorok segítségével, különböző parancsokat végrehajtani aktuátorok segítségével, tehát valamilyen szabályozási kör részeként segítik az életünket. Az Internet of Things (IoT) elterjedésével a sok terepi eszköz, szenzorok, aktuátorok egy hálózatra

kapcsolódnak, sokszor valamilyen vezetéknélküli protokollon (WiFi, Bluetooth Low Energy), így már az sem feltétlenül szükséges, hogy hálózati tápfeszültségre legyenek kötve, elemről működnek. Ez a felhasználási terület új szemléletmódot igényel az energiafelhasználás kérdését illetően. Az, hogy egy rendszer mennyit fogyaszt meghatározza a tervezési fázist és a működtetési folyamatot is.

Személyi számítógép méretű eszközök esetében a fogyasztásmérés megvalósulhat céleszközökkel, amikor is a mérni kívánt egység (pl. GPU, HDD) tápellátását megszakítva egy szenzor méri a fogyasztást [5]. Ettől eltérően az IoT eszközök nagy része úgynevezett egylapkás rendszer (System on a Chip – SoC), így nem hatékony az egyes részegységek tápkábeleit elvágva (forrasztva) energiafelhasználást mérni.

Az energiahatékonyságért tett lépések megértése értekében meg kell ismernünk a jelenleg használt programozási nyelveket. Napjaink trendjei egyre inkább lehetővé teszik beágyazott rendszerek programozását JavaScript és Python nyelveken is. Ezen nyelvek népszerűsége révén nagyobb tömegek nyitottak az irányzat felé, mely indukálja azt a kérdést, hogy ezen nyelvek és futtatókörnyezeteik energiafogyasztása milyen mértékben tér el a C/C++ natív programokétól.

Célom egy olyan eszköz megvalósítása, amely egylapkás rendszerek fogyasztását képes mérni, az adatokat az eszköz tárolja és megosztja a mérést végző kutatóval. Részletes fogyasztásinformációt szeretnék kapni, ezért a mérendő eszközt nem tekinthetem fekete dobozként, a mért kódot pedig kiegészítő információkkal kell ellátnom.

A dolgozat további része a következők szerint épül fel: a második fejezet elméleti áttekintést ad arról, hogy mit is tekintünk beágyazott rendszereknek és a dolgok internetének, ezek után az ARM architektúráról és számítógépek energiafogyasztásáról lesz szó. A fejezet az energia fogalmával és az energiafelhasználás mérésével folytatódik és a kapcsolódó tanulmányok ismertetésével zárul. A harmadik fejezet az elkészült energiamérő modult mutatja be, majd a negyedik fejezetben használati esetek modelljei és mérési eredményeik kapnak helyet. Végezetül az ötödik fejezet tartalmazza az összefoglalást és az elért eredményeket.

A dolgozatban szereplő kapcsolási rajzokat a Fritzing [6] nyílt forráskódú [7] alkalmazással készítettem.

2. Irodalmi áttekintés

2.1. Beágyazott rendszerek

Életünk során folyamatosan használunk beágyazott rendszereket, viszont felépítésükkel, működésükkel, sőt gyakran létezésükkel sem vagyunk tisztában. Vezetés közben elvárjuk, hogy autónk gyorsuljon vagy lassuljon aszerint, hogy a pedálokkal milyen mozdulatokat teszünk, sőt már azt is elvárjuk, hogy az ajtó kinyíljon, ha egy épület bejáratához értünk [8]. Ezeket – és sok más manapság természetesnek gondolt folyamatot – beágyazott rendszerek vezérlik.

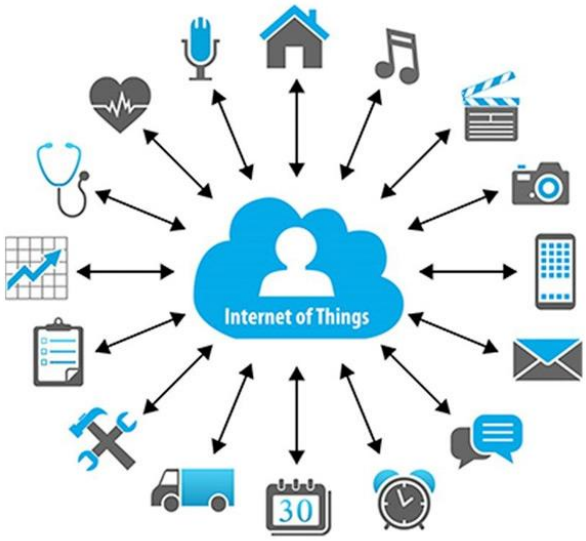
Ezek céleszközök, amelyek egy adott funkció ellátására lettek tervezve és kivitelezve. Tartalmaznak hardver és szoftverelemeket, melyek együttesen, egy feladatot látnak el. Ellentétben az általános célú számítógépekkel – melyeken több program futhat egyszerre – a beágyazott rendszerek feladata egy alkalmazás futtatása, mely legtöbbször az operációs rendszer részét képezi [8]. Ezzel egyidőben korlátozott számítási kapacitással és tárolási lehetőséggel rendelkeznek [9], mely merőben más szoftverfejlesztési technikákat igényel, mint az asztali–, vagy webalkalmazások. Egy rendszer tervezési fázisában behatóan ismerni kell a megoldandó feladatot, a rendszerrel szemben támasztott funkcionális és nemfunkcionális követelményeket, hiszen ezek kihatással vannak a készülő hardverek és szoftverek mivoltára. Ennek fényében az általánosabb feladatokra található a piacon úgynevezett dobozolt megoldás, viszont az egyéni igényekhez mindig új termék születik.

Különböző iparágak különböző követelményeket támasztanak a felhasznált komponensekkel szemben, hiszen számukra a komponensek építőkövek a különböző folyamat– és gyártásirányítások végrehajtásához. Ezek közül a két fő követelmény [9]:

- *Idő*: Egy esemény bekövetkeztekor azt a rendszernek meghatározott időn belül le kell kezelnie. Az ilyen rendszereket *valós idejű rendszereknek* nevezzük.
- *Biztonság*: Ha egy komponens feladata olyan folyamat vezérlése, mely hibás működés esetén anyagi kár vagy egészségkárosodás léphet fel, akkor biztosítani kell a helyes működést minden körülmények között.

2.1.1. Internet of Things (IoT)

Manapság egyre inkább igaz, hogy magában egy–egy beágyazott eszköz nem képes összetett feladatokat ellátni, ahhoz kommunikálniuk kell egymással. Területenként és szabványonként a kommunikáció módja változhat, de mindenképpen jelen van. Egy szenzor az érzékelt mennyiségeket továbbítja a központi feldolgozóegység felé, az döntést hoz és jelet küld az aktuátoroknak, ezzel a szabályozási kör létrejön. Egy összetett termelési folyamathoz több szabályozási körnek kell együttesen működnie, melyeket egy magasabb szintű felügyelőegység ellenőriz.



1. ábra Internet of Things (IoT) [10]

Ennek a koncepciónak az általánosításából jött létre az IoT. Definíció szerint az Internet of Things különböző fizikai eszközök hálózata, ahol az eszközök kommunikálnak egymással, adatokat cserélnek [11]. Egy IoT eszköz lehet az okostelefontól a kávégépen és a mosógépen keresztül szinte bármi, ami ismer kommunikációs protokollt [12]. Ez az elképzelés már a 2000–es évek elején létezett, viszont a 2010–es évekre tett szert széleskörű ismeretségre.

A technológiai fejlődés és a popularitás egyik hatása, hogy egyre több fejlesztő és cég nyit ezirányba. Az eszközök programozása szempontjából több nyelv áll rendelkezésre [13] és ezek közül három kerül vizsgálat alá jelen tanulmányban: C/C++, Python és JavaScript.

Jelen tanulmányban a C/C++, mint referencia kerül felhasználásra, hiszen az alkalmazások a célarchitektúrára vannak fordítva és optimalizálva [8]. A Python és a JavaScript úgynevezett interpretált nyelv, futtatókörnyezet szükséges az alkalmazások futtatásához. Python esetében összehasonlításra kerül a *3.6* nyelvi specifikáció és futtatókörnyezet, JavaScript esetében pedig a Szegeden is aktívan fejlesztett *JerryScript* [14] az IoT.js keretrendszerrel kiegészítve.

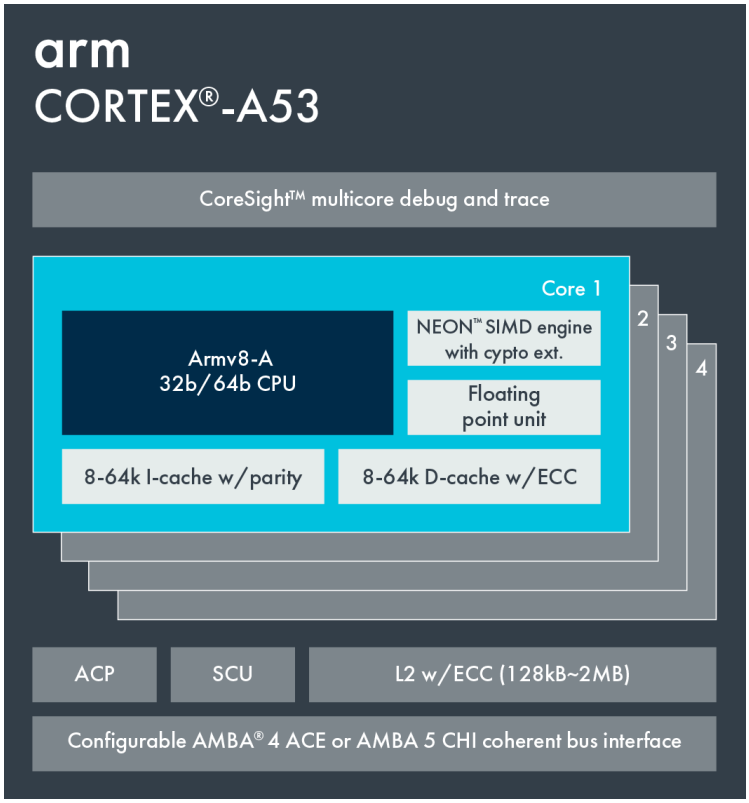
2.2. Architektúra és energiafogyasztás

Noha mára már a laptopok és szerverek piacára is betör az ARM Ltd. [15] [16], a mobil és a beágyazott eszközök piacát e cég uralja hosszú évek óta. Eredetileg az *Advanced RISC Machines* rövidítéseként állt elő az ARM, amiből a felhasznált architektúra expliciten kiderül. A CISC (*komplex utasításkészlet*) architektúrával szemben a RISC (*csökkentett utasításkészlet*) architektúrájú processzorok előállításához kevesebb tranzisztor szükséges, így az energiafelhasználásuk is kisebb. Amíg a versenytársak a teljesítmény növelését tűzték ki célul, addig az ARM alacsony energiafelhasználású integrált áramköröket fejlesztett ki.

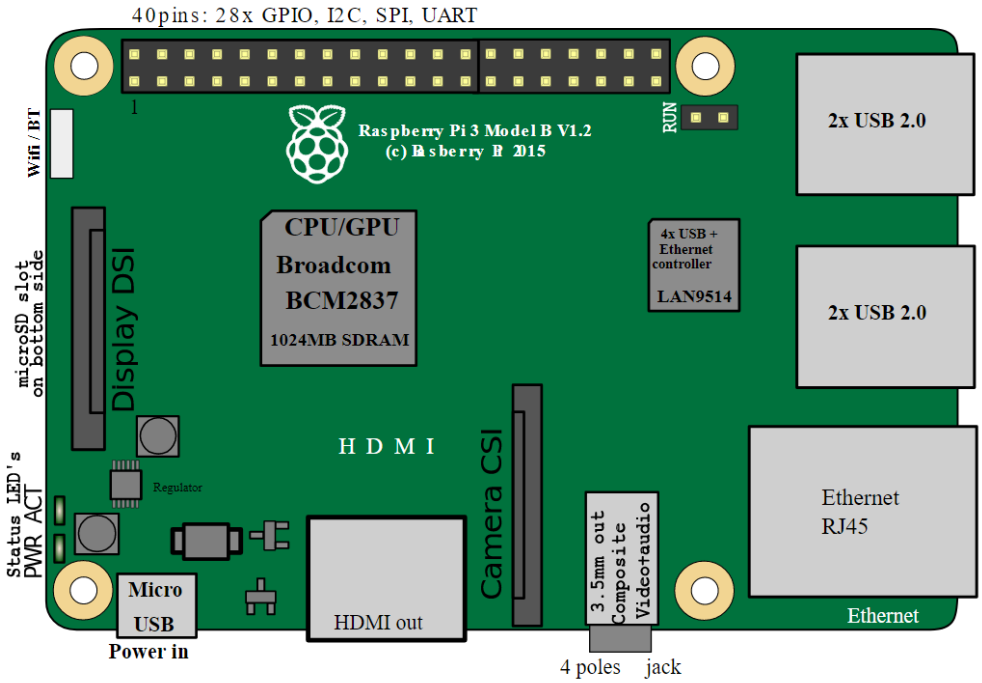
Jelen tanulmányban két Raspberry Pi 3 (B, B+) egylapkás számítógép került felhasználásra, így ezek modellje kerül bemutatásra. Mindkét modell egy 4 magos Broadcom BCM2837 chipkészlettel rendelkezik (órajelben eltérnek), amely az ARM Cortex®–A53 processzorra épül. Ennek modelljét ábrázolja a 2. ábra.

Az eszközökben központi szerep jut a chipkészletnek, viszont energiafelhasználás szempontjából nem ez az egyedüli aktív elem a lapon. Szemlélve a Raspberry Pi teljes modelljét (3. ábra), láthatók azok a CPU–n kívüli eszközök is, melyek energiát fogyasztanak: USB perifériák, USB és Ethernet vezérlő, WiFi / Bluetooth stb.

Mint minden kutatási témánál vagy fejlesztési tervnél, egy vagy több előfeltételezéssel élünk, melyek korábbi kutatások eredményei voltak [17]. Jelen esetben ez az, hogy a CPU fogyasztja a legtöbb energiát, majd a kommunikációs interfészek (Ethernet, WiFi, Bluetooth) majd a fájl I/O. Mivel az eszközök, mint beágyazott eszközök vannak szemlélve, így a Raspbian operációs rendszer azon változatát tartalmazzák, amely nem képes grafikus kimenetre, illetve a TCP/IP protokoll felett elhelyezkedő *ssh* szervizen keresztül vannak vezérelve.



2. ábra ARM Cortex A-53 processzor [18]



3. ábra Raspberry Pi 3 B modellje [19]

2.3. Mérések elmélete

Mérnöki szemmel tekintve a mérés egy kiválasztott mennyiség nagyságának jellemzése a kiválasztott mértékegységben kifejezett számértékkel [20]. Az eredmény egy szám, mértékegységgel kiegészítve (pl. 5 *V*), mely mérési hibával terhelt. A hiba a tényleges érték és a mérésből származó érték különbsége. Minden mérőműszer bizonyos pontossággal rendelkezik, így a mérési hibák nem tüntethetők el teljesen, csak a nagyságuk csökkenthető.

Szoftverek nemfunkcionális követelményeinek vizsgálatakor is elkerülhetetlenek a mérések. A futás– és válaszidők, a felhasznált memória és a bináris mérete, a programkód nagysága és komplexitása mind mért mennyiségek, melyek alapján tekinthető egy követelmény teljesítettnek. Az eredmények validálása érdekében fontos szempont a reprodukálhatóság. A szoftver életciklusa során ez lerögzített fordító– és futtatókörnyezettel van biztosítva, illetve a mérés során a szoftver verziószáma visszakövethető kell legyen (*verziókövetés*).

2.3.1. Eszközök energiafogyasztásának mérése

Egy villamos eszköz feszültségének és az azon átfolyó áram erősségének szorzatát teljesítménynek, vagy munkavégző képességnek nevezzük [21]. Ennek jele *P* és mértékegysége a watt: *W*.

$$P = U \cdot I \tag{1}$$

$$1\,W = 1\,V \cdot 1\,A \tag{2}$$

Ezután a villamos munka, *W* a teljesítmény és a munkavégzésre szánt idő szorzataként áll elő, mértékegysége a wattszekundum.

$$W = P \cdot t = U \cdot I \cdot t \tag{3}$$

$$1\,Ws = 1\,V \cdot 1\,A \cdot 1\,s \tag{4}$$

A wattszekundum viszonylag kis mértékegység, így a köztudatban a háztartások, gyárak fogyasztásának jellemzésére a kilowattórát (kWh) használják.

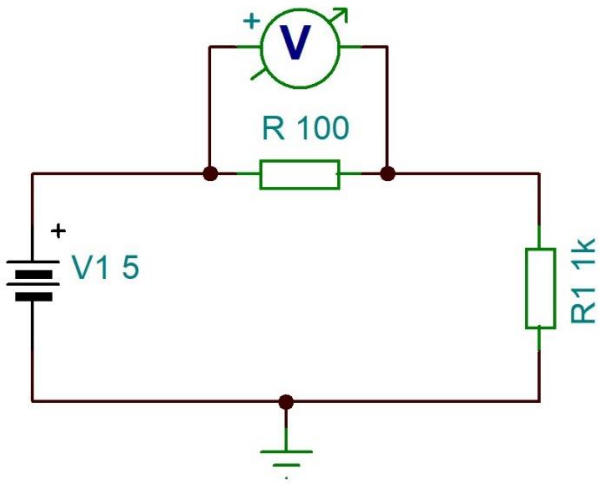
$$1\,kWh = 1000\,W \cdot 1\,h = 1000\,W \cdot 3600\,s = 3{,}6 \cdot 10^6\,Ws \tag{5}$$

A tanulmány során végzett mérések hossza másodpercekben mérhető, így a *Ws* mértékegység kerül felhasználásra.

A fentiek alapján egy eszköz fogyasztása megállapítható, ha a tápellátását megszakítva mérjük a felvett áramot és ezt összeszorozzuk a tápfeszültséggel (1. egyenlet). Az áram mérése a hálózat megszakításával valósul meg, amikor is a hálózatot egy árammérővel egészítjük ki, melynek ideális ellenállása 0 Ohm. Egy másik, elterjedtebb módszer, amikor az áramforrás és a terhelés (eszköz) közé egy kis, ismert értékű ellenállást helyezünk és a rajta eső feszültséget mérjük. A feszültség és az ellenállás értékét ismerve Ohm törvényével kiszámítható az áramerősség.

$$I = \frac{U}{R} \tag{6}$$

A fogyasztásmérés ily módon a 4. ábra szerint valósulhat meg. A tápfeszültség az eszköz tápegysége, melynek vezetékét egy ismert értékű ellenállással megszakítva mérhető az eszköz áramfelvétele.



4. ábra A fogyasztásmérés kapcsolási rajza

Az ábrán *V1* jelöli az eszköz tápegységét, míg *R1* magát az eszközt. A tápellátást megszakító kis értékű ellenállást *R* jelöli, amin a voltméter segítségével mérhető a feszültségesés. Amennyiben *R* és *R1* közé egy ampermétert helyezünk és szimulációt futtatunk, akkor a voltméter által mért feszültség *454,55 mV*, amire alkalmazva a 6. egyenletet megkapjuk az áramerősség értékét: *4,545 mA*. Az ampermétereen ugyanez az érték látható.

Az ábra illusztráció, a feltüntetett értékek demonstrációs jellegűek. A tényleges megvalósítást a 3.1 fejezet taglalja.

2.4. Kapcsolódó tanulmányok

A beágyazott és hordozható eszközök energiafogyasztásának becslése és mérése nem kiaknázatlan terület, több kutató és cikk is foglalkozik a témával valamilyen felhasználói esetet vizsgálva. I. U. Marqués doktori tézisében [2] a mobil eszközök WiFi kommunikációjának fogyasztását, míg A. Viswanathan a BSc szakdolgozatában [22] az IoT szemléletben elterjedt MQTT protokollt vizsgálta. C. M. Paradis MSc diplomamunkájában [17] a Linux *perf* eszközt bővítette ki annak érdekében, hogy pontosabb mérési eredményekhez jusson. S. Schubert és társai [23] is hasonló módon, a Linux kernel változtatásával érték el az eszközök fogyasztásadatait.

M. Roth és társai cikkükben [24] modellezték és mérték a mikrovezérlők energiafogyasztását fordítóprogramok optimalizálásának céljából. A mért adatokat a mérőberendezés folyamatosan küldi a fejlesztő számítógépe felé USB interfészen keresztül, így a mérés helyhez és fejlesztőhöz kötött.

A tanulmányok különböző protokollokat és hardver elemeket vizsgáltak energiafogyasztás szempontjából, viszont különböző programozási nyelvek viszonyát nem hasonlították össze. Célom az energiahatékony szoftverfejlesztési környezet megtalálása anélkül, hogy az architektúrát vagy az operációs rendszert megváltoztatnám. Különböző programozási nyelvek fogyasztását csak úgy tudom hatékonyan megvizsgálni, ha a befogadó hardvert és operációs rendszert érintetlenül hagyom, így magának az alkalmazásnak a fogyasztását vizsgálhatom.

3. Energiafogyasztás mérése

A Mérések elmélete fejezetben megállapításra került, hogy mit jelent az, hogy egy eszköz energiát fogyaszt. Már csak a berendezés megvalósítása hiányzik. A 4. ábra szerint az eszköz tápellátását meg kell szakítani egy ellenállással és ezen mérni a feszültségesést egy voltméterrel. Ez a megoldás a mintavételezésről és az adatok mentéséről nem ad tájékoztatást.

A cél egy fogyasztásmérő eszköz létrehozása, amely képes meghatározott mintavételezéssel – jelen esetben 200Hz – adatokat szolgáltatni és menteni azt későbbi feldolgozásra. Ennek elérése érdekében egy Raspberry Pi (3 B+) végzi a tápfeszültség és a söntellenálláson átfolyó áram erősségének mérését egy INA219 árammérő szenzor segítségével, mely a két mért értékből a pillanatnyi teljesítményértéket automatikusan kiszámítja. Az eszközt és használatát a 3.1. fejezet mutatja be részletesen. Az adatokat a szenzor szolgáltatja a Raspberry–nek, mely menti azokat későbbi feldolgozás céljából egy CSV (*comma separated values*) fájlba. A fájl szerkezete táblázatba foglalva a következő:

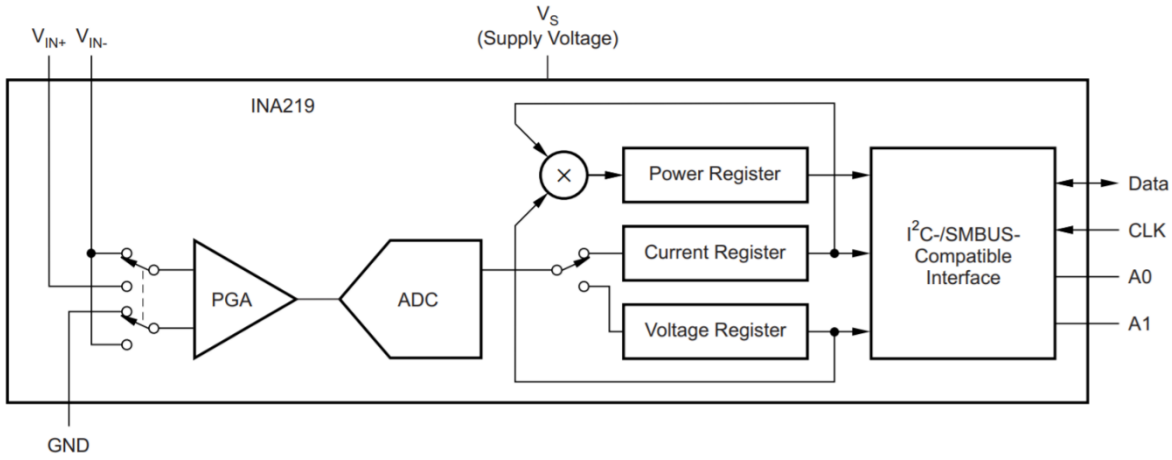
Időbélyeg	Tápfeszültség (V)	Áramerősség (mA)	Teljesítmény (mW)
0:00:00.012116	5,121	246,785	1263,858
0:00:00.030684	5,120	254,642	1303,769
0:00:00.049042	5,106	247,284	1262,749
0:00:00.067402	5,116	246,391	1260,532

1. táblázat A mentett log fájl szerkezete

A folyamatot lehet kézzel vezérelni, vagyis az eszközön elindítani a mérést, a mért eszközön pedig azt az alkalmazást, ami a vizsgálat tárgyát képezi. Ez a módszer megközelítő értékeket szolgáltat és nem teszi lehetővé programozási konstrukciók mérését. A megközelítő eredmények helyett szeretném elérni, hogy mérhető legyen egy algoritmus, esetleg egy vezérlési szerkezet, vagy akár egy utasítás futása is. Ehhez a mért és a mérő eszköz között szinkronizációnak kell történnie, melyet részleteiben a 3.2. fejezet ír le. Az elkészült modul egy helyi hálózaton keresztül elérhető, kutatók által használható eszköz, melynek biztosítania kell a mérés folytonosságát. Ez azt jelenti, hogy a kutatást végző egyénnek nem kell ismernie a mérő berendezés szerkezetét és a mérést kézzel elindítania. Ezen felül a mért adatok elérését webböngészőn keresztül biztosítja. Ezekről a megoldásokról szól a 3.3 fejezet.

3.1. INA219 szenzor és alkalmazása

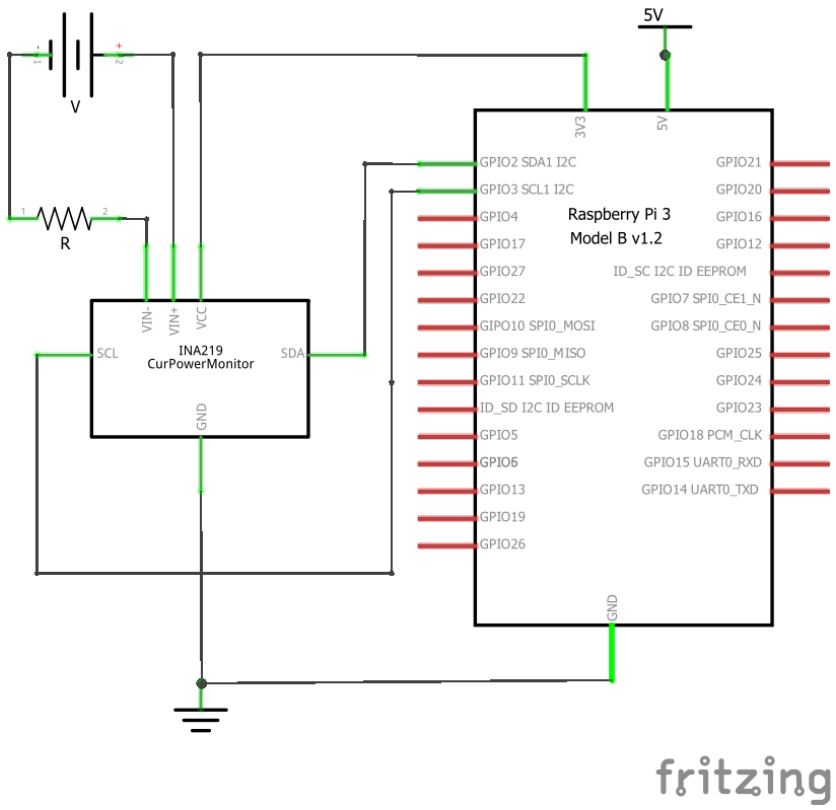
Az INA219 [25] egy energiamérő szenzor a Texas Instrumentstől, mely I^2C és SMBUS interfészekkel van ellátva. Söntellenállása 0,1Ω, melyen keresztül folyik a mért eszköz tápellátása. Az ellenálláson mért feszültség mellett szolgáltatja az áram és a teljesítmény értékét is 1%–os pontossággal. Működéséhez 3,3V tápfeszültséget igényel, melyet a Raspberry biztosít számára. Maximális mérhető áramerősség 3,2 A, a mérhető feszültségtartomány 0–26 V, rendeltetésszerű működése pedig –40°C – +125°C között garantált. Egyszerűsített kapcsolási rajza az alábbi ábrán látható.



5. ábra INA219 szenzor egyszerűsített kapcsolási rajza [25]

Az ábrán látható, hogy a programozható erősítőbe (*Programmable Gain Amplifier – PGA*) érkezik a bemenő feszültség és a földpotenciál, melynek kimenete egy analóg–digitális átalakítóhoz kapcsolódik. Az átalakító kimenete két regiszterbe betölti az áram és a feszültség értékeket, melyek a kommunikációs interfészen keresztül kiolvashatók. A teljesítményértéket a másik két regiszter értéke alapján számítja ki.

A Raspberry Pi–vel való összekapcsoláshoz négy vezeték szükséges: tápellátás, földpotenciál, SDA és SCL vezetékek. Ezután a V_{IN+} és a V_{IN-} lábakra kell kapcsolni a mérendő terhelés tápfeszültségét, és a terhelés által felvett áram kiolvasható az RPi–n keresztül.



6. ábra INA129 és Raspberry Pi összekapcsolása

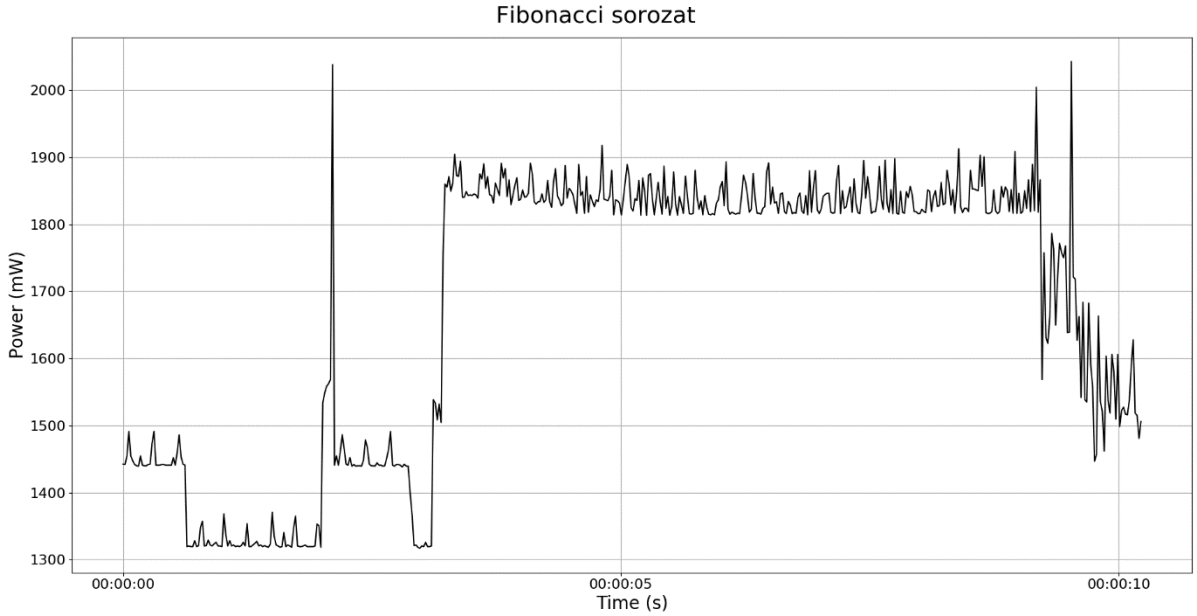
Az INA219 használatához az RPi–n engedélyezni kell az I^{2}C kommunikációt, majd az *i2cdetect* parancs használatával megtekinthető, hogy az RPi látja–e a szenzort. 16 eszközt lehet csatlakoztatni egy RPi–hez, az *i2cdetect* parancs által visszaadott szám a szenzor kommunikációs címe. Ez egy egyedi cím, melynek birtokában történhet kommunikáció a szenzor és az RPi között. A *pi-ina219* [26] nevű Python csomag segítségével egyszerűen használatba vehető, melyre egy példa a következő kódrészlet.

```
from ina219 import INA219
from ina219 import DeviceRangeError
SHUNT_OHMS = 0.1
SENSOR_ADDRESS = 0x40

if __name__ == "__main__":
    ina = INA219(SHUNT_OHMS, address=SENSOR_ADDRESS)
    ina.configure()
    try:
        print(ina.current())
        print(ina.power())
        print(ina.shunt_voltage())
    except DeviceRangeError as error:
        print(error)
```

Az első két sor importálja a nélkülözhetetlen elemeket, melyek szükségesek a szenzorral történő kommunikációhoz. A söntellenállást ki lehet cserélni más értékűre amennyiben a méréshatárt bővíteni kell, ezért lehetőség van beállítani a söntellenállás értékét a konstruktorban. A második paraméter az eszköz kommunikációs címe. Az objektum létrehozása után konfigurációs lépés szükséges, az itt átadott paraméterek határozzák meg a további működést, mint pl. az analóg–digitális átalakító pontossága (9, 10, 11 és 12 bit), és a mintavételezés sebessége. A konfiguráció után használható a szenzor, viszont a példában látható függvényhívások kivételt dobhatnak, amennyiben a mért értékek meghaladják a méréshatárt. Ez egy olyan esemény, melyet le kell kezelni *try-catch* blokkal.

3.2. Szinkronizáció az eszközök között

Az első kísérletek során a mérések manuálisak voltak, így pontatlan eredmények születtek. Az adatsorok elején és végén változó mennyiségű nyugalmi állapot található, melyet automatikus módszerrel nehéz szűrni. Olyan alkalmazások esetén, melyek futásuk végén már alacsony energiafogyasztást produkálnak nehéz eldönteni, hogy egy intervallum még a programfutáshoz tartozik, vagy már nyugalmi állapot.



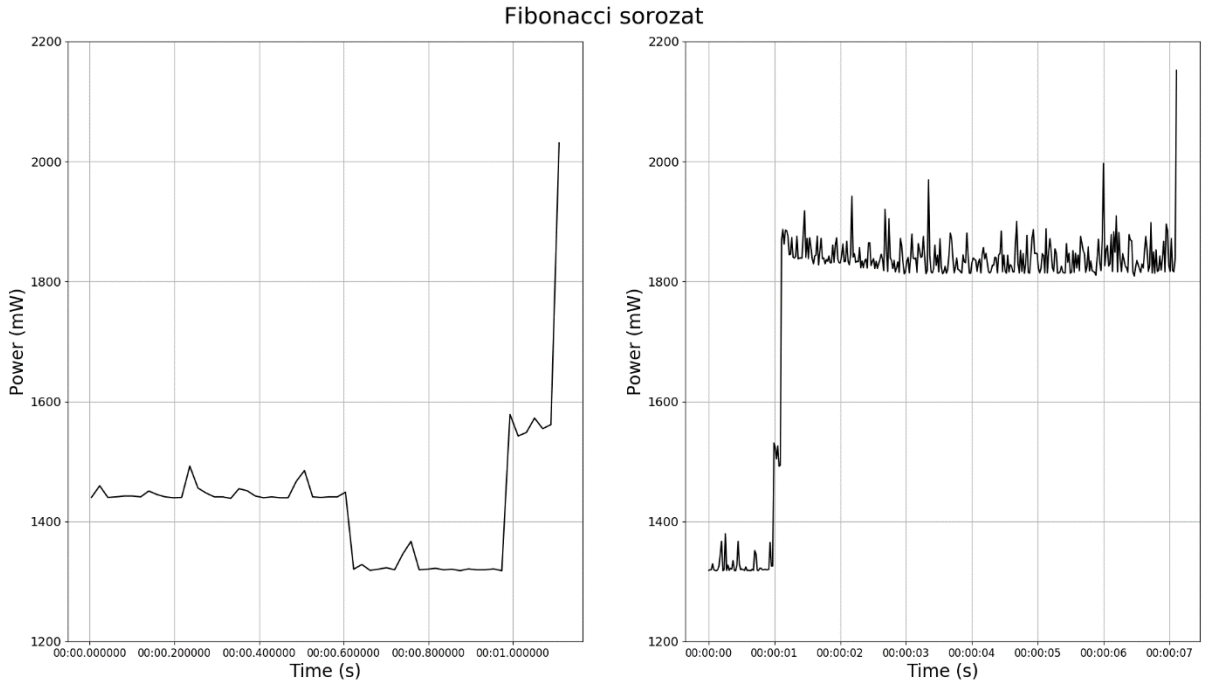
7. ábra Fibonacci sorozat kiszámításának mérési eredménye

A következő lépés tehát az indítás és leállítás automatizálása. A módszer előnye, hogy a program indulása előtt és a befejeződése után pontosan meghatározott intervallum van mérve, hátránya viszont, hogy csak teljes futást képes mérni. Amennyiben ennél részletesebb eredmények szükségesek, akkor a futó kódot ki kell egészíteni információkkal, melyek az adatfeldolgozás során elválasztják az intervallumokat.

A 7. ábra szemlélteti a Fibonacci sorozat első 500 elemének kiszámításának és fájlba írásának mérési eredményét. Az ábráról nem olvasható le, hogy meddig tart a sorozat kiszámítása és mikor kezdődik a fájlba írás. Ez a felbontás az alábbi pszeudó kóddal valósulhat meg.

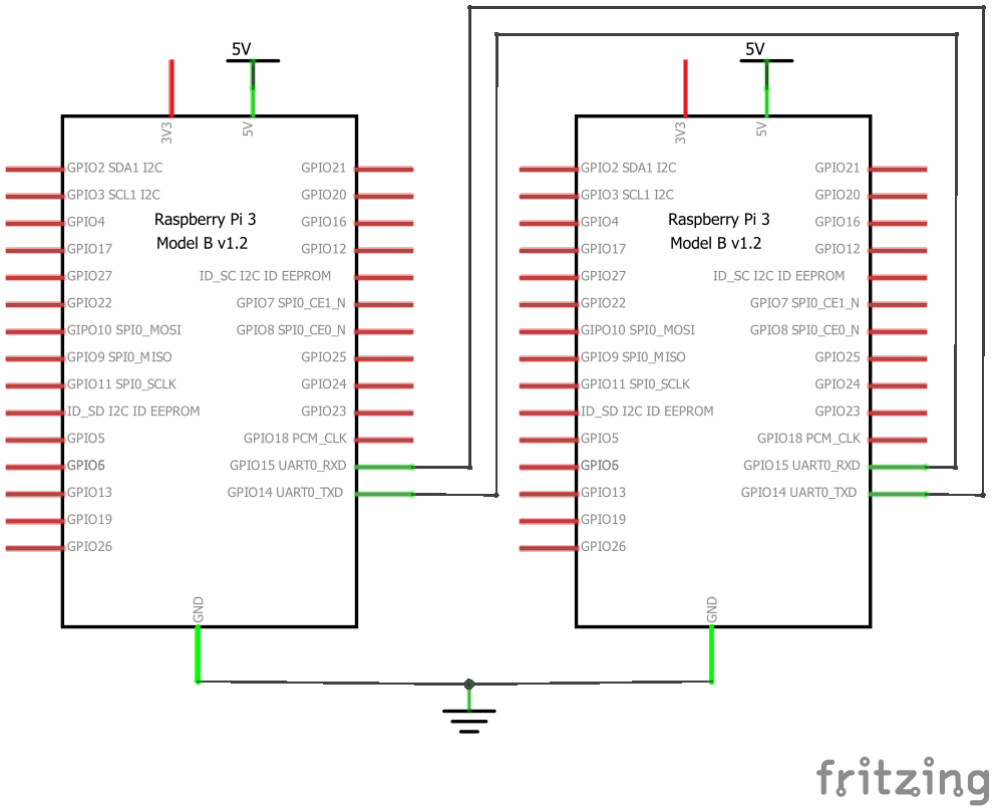
```
start_measurement('fibonacci_sequence')
calculate_fibonacci_sequence()
stop_measurement('fibonacci_sequence')

start_measurement('write_to_file')
write_results_to_file()
stop_measurement('write_to_file')
```

A felbontásnak megfelel a 8. ábra, ahol elkülönül a sorozat kiszámítása (balról) és az eredmény kiírása fájlba (jobbról).



8. ábra A Fibonacci sorozat kiszámítása részleteiben

A mérés indításához és leállításához a mért eszköznek kommunikálnia kell a mérést végzővel. Ez aszinkron soros kommunikáción [27] keresztül valósul meg, a két RPi *TXD* és *RXD* kivezetései összekötése révén.



9. ábra Eszközök összekapcsolása UART kommunikációhoz

Mindkét eszközre telepítve van a *pySerial* [28] nevű Python csomag, mely segítségével küldhető és fogadható üzenet a portokon keresztül. A C/C++, illetve JavaScript nyelveken írt felhasználói esetek során megfelelő nyelvi elemek segítségével történik a kommunikáció végrehajtása.

```
import serial
with serial.Serial('/dev/ttyS0', 115200, timeout=1) as ser:
    ser.write(b'Hello World\n')

    response = ser.readline()
    print(response)

    ser.close()
```

A mérés indítása és leállítása, illetve a hibák jelzése ily módon történik. A következő fejezet technikai betekintést nyújt a tényleges megvalósítás részleteibe.

3.3. Mérés és adatfeldolgozás

Az egyik eszköz képes mérni a csatlakoztatott INA219 szenzor segítségével, illetve a két eszköz tud kommunikálni egymással soros, aszinkron kommunikáció segítségével. Ezt a két tulajdonságot kihasználva létrehozható az az energiamérő modul, mely segítségével nem csak teljes alkalmazások, hanem programozási konstrukciók is mérhetők.

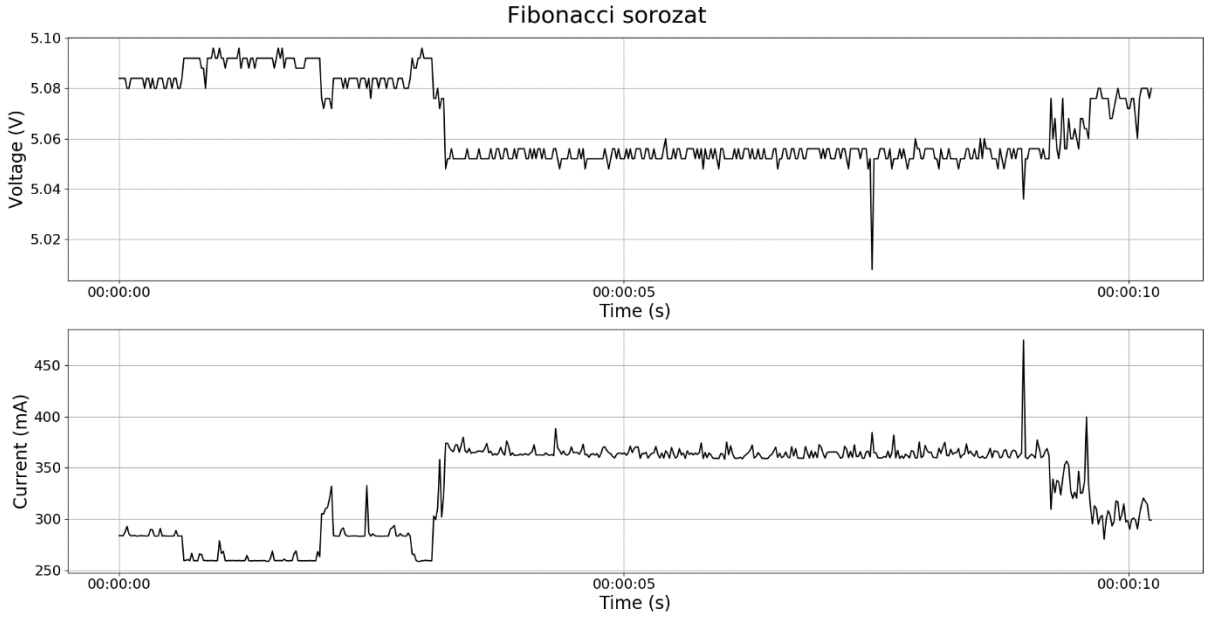
Legyen *E–RPi* a mérő és *M–RPi* a mért eszköz. Az E–RPi–n egy rendszerfolyamat figyeli a kommunikációs csatornát bejövő parancsokra várva. Két parancs került definiálásra: mérés indítása (*1*) és leállítása (*2*). A parancsok rendelkeznek szöveges paraméterrel, mely a kísérlet azonosítására szolgál.

```
{ "command" : 1, "msg": "fibonacci" }
{ "command" : 2, "msg": "fibonacci" }
```

Alapállapotban nem történik mérés az E–RPi által, csak figyeli a kommunikációs csatornát. Amikor egy indító parancs érkezik M–RPi–től, akkor a *msg* paraméter értéke azonosítja az indítandó adatgyűjtést. A leállító parancs során a rendszerfolyamat ellenőrzi, hogy a futó mérés leállítására érkezett–e kérés, amennyiben nem, akkor megszakítja az adatgyűjtést mentés nélkül és a kommunikációs csatornán visszaküld egy hibaüzenetet. Ellenkező esetben az összegyűlt információt lementi egy előre meghatározott mappába *időbélyeg_mérés_azonosító.csv* elnevezésű fájlba.

A fenti parancsokat többször kiadva nem történik fájl duplikáció, hiszen az időbélyeg mikroszekundum pontossággal határozza meg a fájl nevét. Az is hibaként van értelmezve, ha egy mérési folyamat közben újabb indító parancs érkezik. Ebben az esetben hibaüzenet visszaküldése mellett az eredeti folyamat fut tovább és kerül lementésre helyes leállítási parancs érkezése esetén. Az E–RPi–n egy FTP szerver is fut, melyen keresztül tetszőleges FTP kliens segítségével letölthetők a mérési eredmények. A böngészőben az *ftp://<e-rpi-ip-cím>* –re rákeresve az a mappa érhető el, ahová az eszköz a mentést végzi. Ily módon a kutatónak nem kell foglalkoznia a modul felépítésével és konzolt használva letöltenie a kívánt fájlokat.

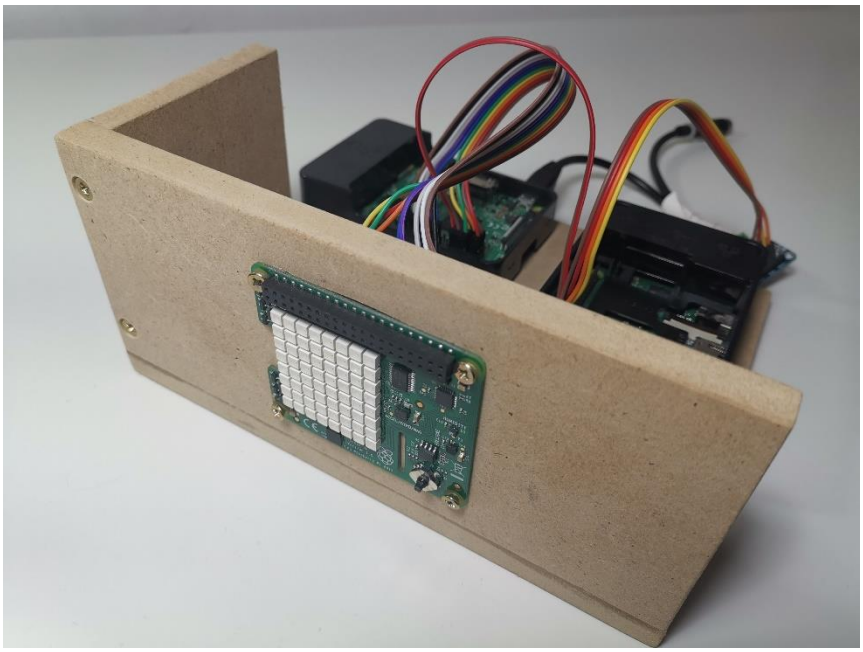
Mint azt már a 1. táblázat is szemléltette, az időbélyeg mellett a feszültség (*V*), az áramerősség (*mA*) és a pillanatnyi teljesítményérték (*mW*) kerül mentésre az adatgyűjtés során. A fájl megnyitható Excellel, sőt bármilyen szövegszerkesztővel, de a másodpercenkénti *200* adatsort tartalmazó fájl manuális áttanulmányozása nem hatékony folyamat. Egy–egy *csv* fájl még feldolgozható Excellel, viszont több esetén (8. ábra) már újra manuálissá válik a folyamat. Ezek az igények adták a motivációját annak az adatfeldolgozó szkriptnek, mely segítségével kiértékelhetők a mérési eredmények. Több bemeneti fájl esetén egymás mellé helyezi az ábrákat, ahogyan a 9. ábra mutatja. A három mentésre került adat bármilyen kombinációban ábrázolható, akár egyet–egyet kiemelve, de mindhárom feltüntethető ugyanazon az ábrán.



10. ábra A Fibonacci sorozat kiszámításának feszültség és áramerősségadatai

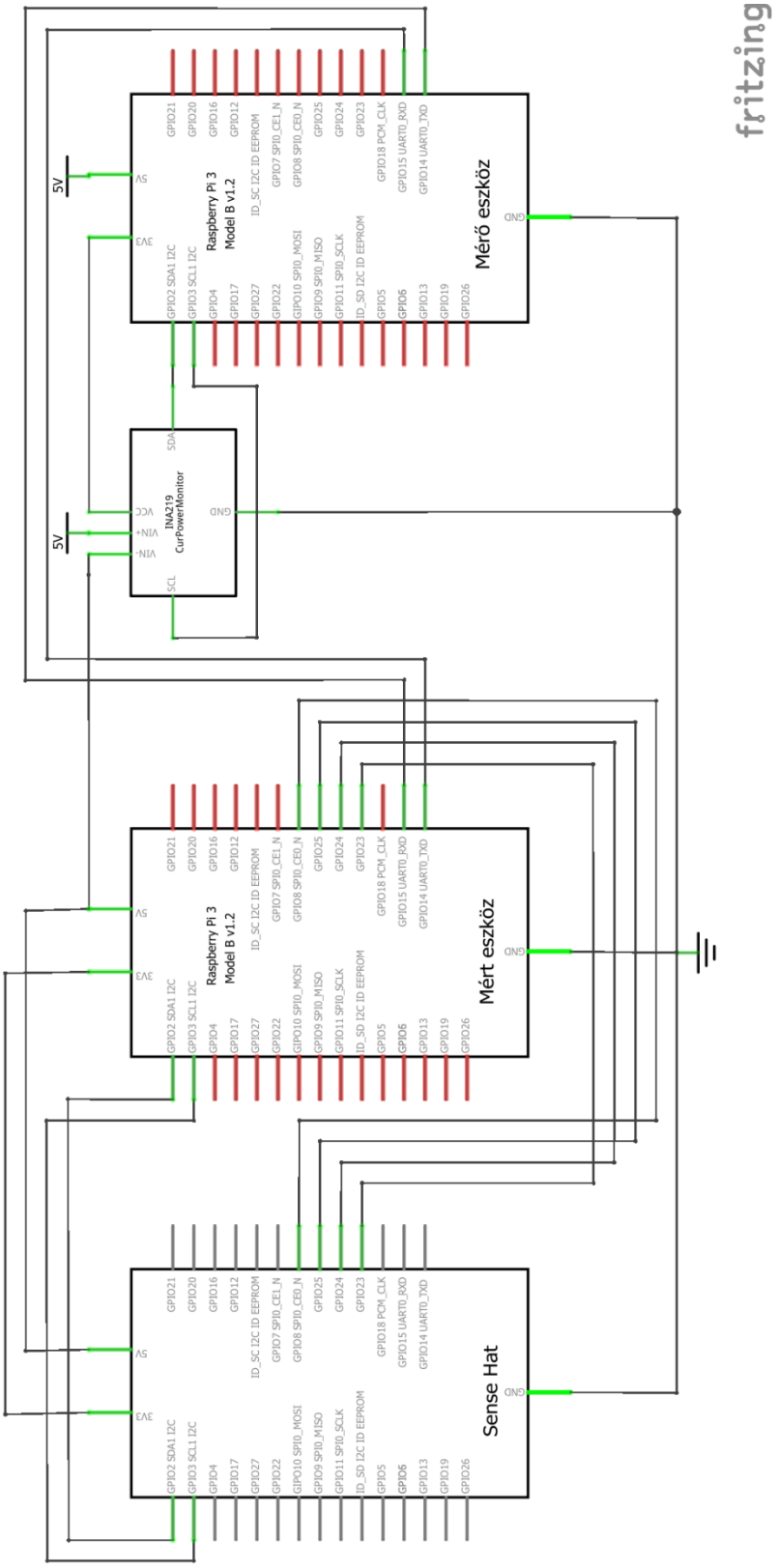
Mivel M–RPi úgy tekintendő, mint beágyazott eszköz, mely szenzorokkal és aktuátorokkal rendelkezik, ezért egy SenseHat [29] nevű bővítőpanellel van kiegészítve. A modul a 4.1. fejezetben kerül leírásra. Egy bővítőpanel csatlakoztatása az eszközhöz megnöveli annak energiafogyasztását, hiszen az M–RPi biztosítja számára a tápellátást. A 3. és 4. fejezet mérései mind úgy lettek kivitelezve, hogy modul csatlakoztatva volt M–RPi–hez. Bővítőpanel nélkül az eszköz átlagos energiafogyasztása *1.353,5 mW* (*266,7 mA*), míg a modul csatlakoztatása utána megnövekedett *1388,4 mW*–ra (*272,6 mA*) 15 másodperc tétlenségi állapot mérése során. Ez változhat a Raspbian rendszerfolyamataitól függően.

A teljes rendszer kapcsolási rajzát szemlélteti a 13. ábra, mely tartalmazza az E–RPi–t az INA219 szenzorral kiegészítve (6. ábra), a két RPi kommunikációját biztosító vezetékeket (9. ábra) és az M–RPi–t a SenseHat bővítőpanellel kiegészítve. A megvalósult rendszerről készült fényképeket pedig a 11. ábra és a 12. ábra mutatja be.



11. ábra Az elkészült modul elölnézetből

12. ábra Az elkészült modul hátulnézetből

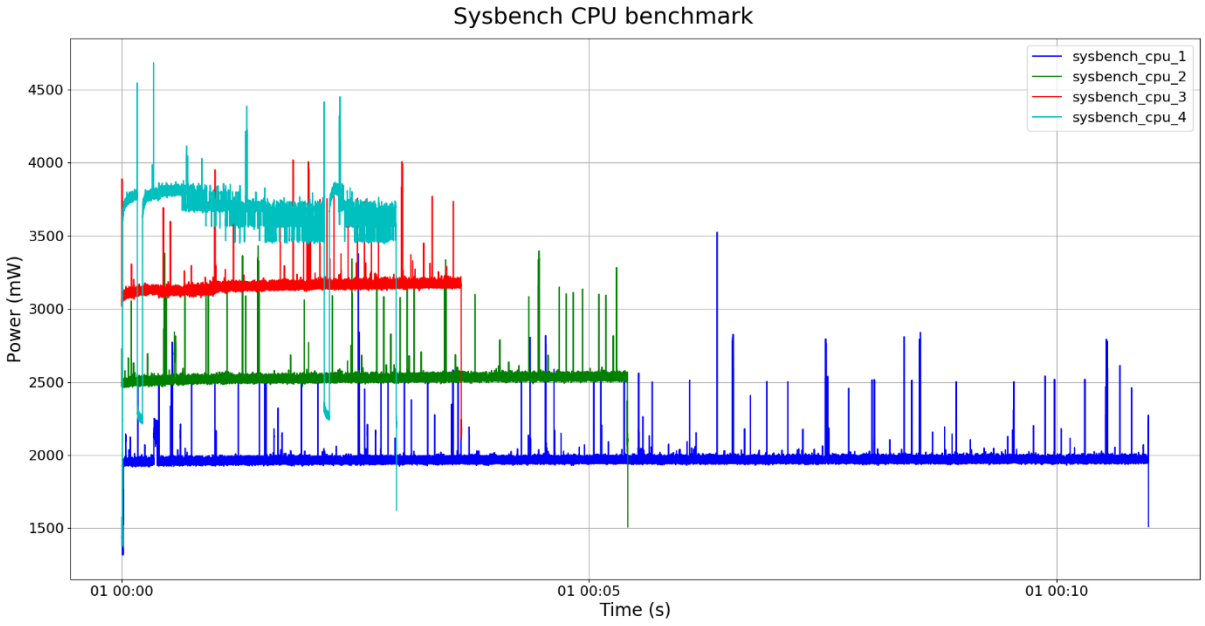


13. ábra A teljes rendszer kapcsolási rajza

3.4. Publikus teljesítménytesztek energiafogyasztása

A szoftverfejlesztési folyamat szerves részét képezi a tesztelés is, a teljesítmény pedig a nemfunkcionális követelmények közé tartozik [1]. Teljesítménytesztek – *benchmarkok* – használata széles körben elterjedt eszközök teljesítményének meghatározására. Egy teszthalmaz a rendszer különböző részeit hozza működésbe, így azokról külön–külön is információ gyűjthető. Ilyen benchmarkokat mutat be a fejezet. A másik véglet az eszközt egy egységként tekinti és komplexszebb tesztesettel közel az összes modulját működésbe hozza. Ilyen alkalmazásra példa a 4. fejezetben található. A különböző tesztek az energiafelhasználás szempontjából vannak vizsgálva.

A Sysbench [30] egy parancssori Linux benchmark, mely tartalmaz processzor, memória, fájl elérés teszteket. Az alkalmazást a disztribúció csomagkezelőjén keresztül lehet telepíteni, a tanulmányban a *0.4.12* verzió került felhasználásra. A fájlrendszert célzó teszt a rendelkezésre álló memóriánál kétszer nagyobb adatmennyiséget mozgat, hogy a gyorsítótár ne tudja megváltoztatni a fájlműveletek eredményét. Véletlenszerű írás és olvasás történik az SD kártyára. A processzor teljesítményét mérő alkalmazás prímszámokat oszt el 2-től indulva a prímszám négyzetgyökéig egyesével. Beállítható a használt processzormagok száma is, amely az RPi esetében maximum négy.



14. ábra Sysbench processzort terhelő tesztje

Az alkalmazás először egy processzormagot használ, majd kettőt és így tovább. Az utolsó felhasznált teszt típus a RAM memóriát vizsgálja. Puffereket allokál, használ, majd felszabadít, adatmozgatás a processzor és a memória között történik. A mérési eredményeket a 2. táblázat foglalja össze.

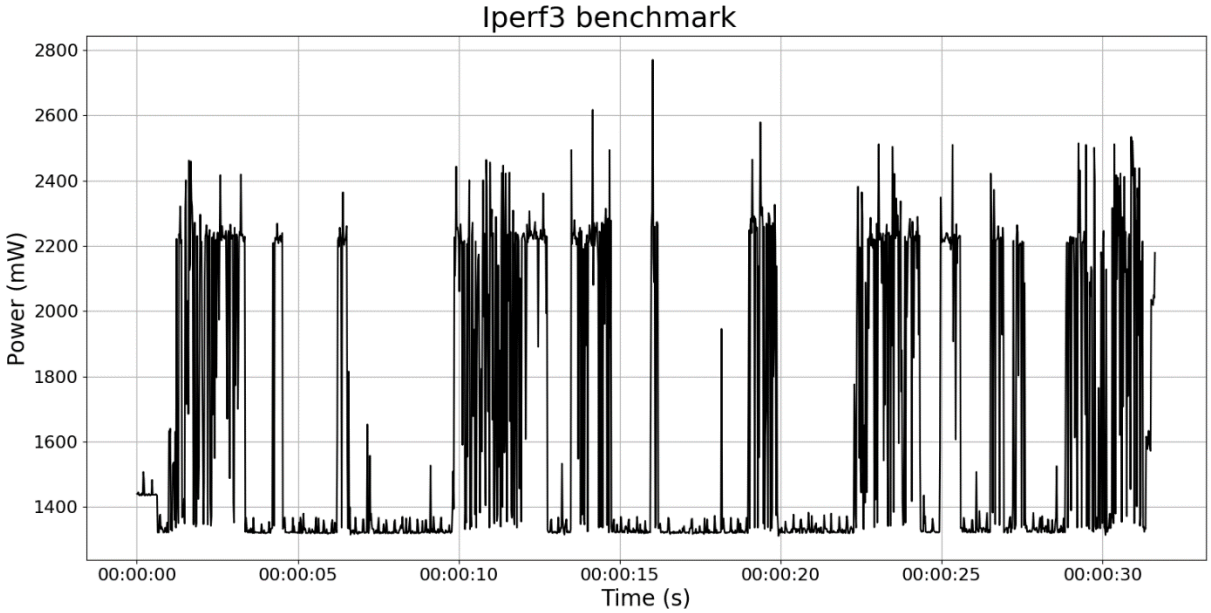
Név	Átlagos teljesítmény (mW)	Időtartam (s)	Energiafogyasztás (Ws)
IO előkészítés	1642,934	186,130	305,8
IO benchmark	1520,604	301,290	485,1
IO törlés	1811,277	1,748	3,2
CPU 1 mag	1957,131	658,963	1289,7
CPU 2 mag	2516,946	324,782	817,5
CPU 3 mag	3143,585	218,147	685,8
CPU 4 mag	3622,603	176,375	639,0
RAM 1 mag	1621,161	1,066	1,7
RAM 2 mag	1577,547	1,319	2,0
RAM 3 mag	1574,935	1,300	2,0
RAM 4 mag	1564,878	1,282	2,0

2. táblázat Sysbench mérési eredmények

Az számításigényes tesztek átlagos teljesítménye a processzormagok számának függvényében láthatóan növekszik (14. ábra, 2. táblázat), míg a futásidő és a fogyasztás csökken. A RAM memória teljesítményét vizsgáló tesztek futtatásakor is beállítható a használt processzormagok száma, viszont az eredmények alapján ez nem befolyásolja sem a futásidőt, sem az energiafogyasztást.

Az Ohio Állami Egyetem által gondozott Polybench/C 3.2 [31] harminc numerikus számítást végző tesztet összefoglaló alkalmazás. A számítások: lineáris algebrai feladatok megoldásai, képfeldolgozás, fizikai szimulációk futtatása stb. A hivatkozott weboldalon megtalálható a tesztek rövid leírása. A 30 soros táblázat terjedelme miatt a mérési eredmények a Függelékben kaptak helyet.

Az utolsó terület a hálózati kommunikáció, mely az *iperf3* [32] szoftverrel került tesztelésre. Az iperf3 nyílt forráskóddal rendelkezik és a tesztek futtatásához két számítógép szükséges: szerver és kliens. A kliens az M–RPi, a szerver pedig egy Intel i5–6300U processzorral rendelkező ThinkPad T470, Ubuntu 18.04 operációs rendszert futtatva. A két végpont között adatcsere történik (TCP vagy UDP protokollon) mindkét irányba és az átviteli sebességet vizsgálja. A vizsgálat során a tesztben résztvevő eszközök WiFi kapcsolattal csatlakoztak ugyanahhoz a hálózathoz. A teszt energiafelhasználását a 15. ábra szemlélteti.



15. ábra Iperf3 benchmark futási eredménye

Az ábrán látható egy periodicitás, mely az adatok küldésének és fogadásának eredménye. Az adatcsere végrehajtása után az eszköz visszatér nyugalmi állapotába a következőre várva. A teszt futásideje 31 másodperc, mely alatt átlagosan 5,086 V feszültséget, 320,730 mA áramot vett fel az eszköz, mely 51,73 Ws fogyasztásnak felel meg.

A benchmarkok eredményeinek vizsgálata közben látható, hogy az eszköz részeinek terhelése más–más eredményt ad. Legnagyobb energiafelhasználást a processzor terhelése eredményezi. Még egy processzormag terhelése is nagyobb fogyasztást indukál, mint a RAM memória vagy a fájl műveletek. Figyelembe kell venni, hogy beágyazott rendszerek esetében a RAM memória be van építve a CPU mellé, így külön nem mérhető az energiafogyasztása. Ezek legtöbbször perifériákkal kiegészített eszközök, melyekre ezek a mérések nem terjedtek ki. A 4. fejezet egy alkalmazás megvalósítását és mérését mutatja be perifériák felhasználásával.

4. Használati eset modellezése és mérése

Az előző fejezetben bemutatásra került az energiafelhasználás mérése és különböző publikus teljesítménytesztek is vizsgálva lettek. A dolgozat céljai között szerepel – a mérések definiálása után – megállapítani azt, hogy egy alkalmazás futtatókörnyezete mennyiben változtatja meg annak energiafelhasználását. A fejezet egy gyakori felhasználói eset megvalósításának modellezésére alapozza a kérdés megválaszolását.

Az internetet böngészve IoT projektek után a leggyakoribb találatok az okosház és a meteorológiai állomás [33] [34]. A Hackster portál gyűjteményében a „weather station" kifejezésre keresve 356 projekt található [35], míg a GitHub 5.230 repository–t listáz. A projekt popularitása mellett a Raspberry Pi 3B elérhető egy úgynevezett „IoT Learner Kit" csomagban [36], mely az eszköz mellett egy SenseHat bővítőpanelt [29] is tartalmaz, melyet a 4.1 fejezet ismertet részletesen. A bővítőpanel segítségével könnyedén megvalósítható egy meteorológiai állomás.

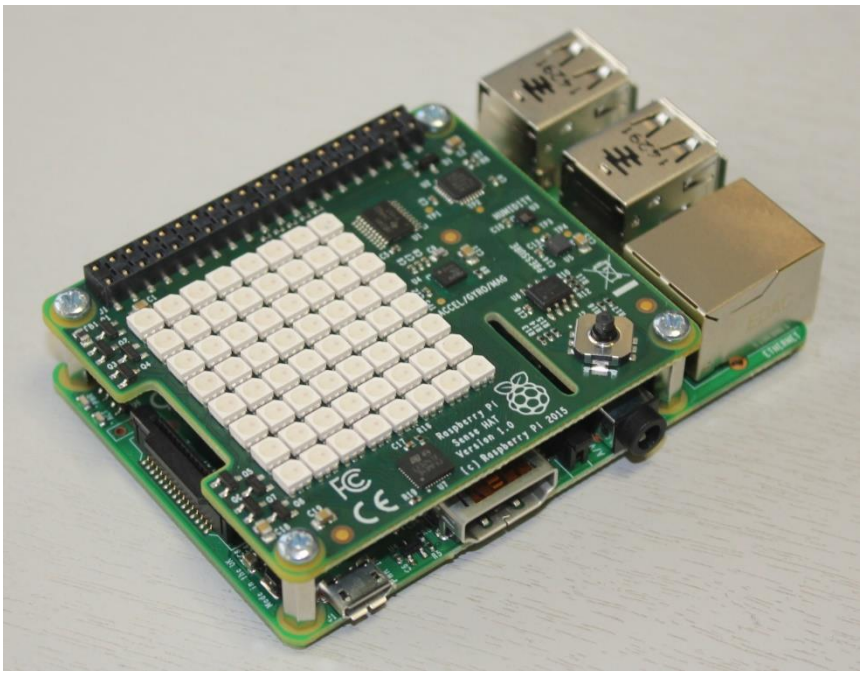
A cél nem az állomás újragondolása, hanem annak vizsgálata, hogy ugyanazon IoT alkalmazás miként viselkedik különböző futtatókörnyezettel megtámogatva. Referenciaként C++ alkalmazás szolgál, ahol is a fordítóprogram a célhardverre optimalizált binárist állít elő, kompetitorai pedig Python és JavaScript nyelven íródott alkalmazások. Python esetében a 3.6 [37] verziószámú futtatókörnyezet kerül vizsgálat alá, míg JavaScript esetében a JerryScript [14] futtatómotor verzió nélküli változata, melyet az IoT.js [38] nevű keretrendszer október 31.–i (@c35e675) verziója tartalmaz. A JavaScript esetében az 5.1 [39] szabvány kerül felhasználására, melyet a használt JerryScript verzió támogat.

Fordító / futtatómotor	Nyelvi specifikáció
g++ (Raspbian 6.3.0-18+rpi1+deb9u1) 6.3.0 20170516	C++14
JerryScript @4bdc3a1	ECMAScript 5.1
Python 3.6	Python 3.6

3. táblázat A vizsgált fordító és futtatómotorok

4.1. SenseHat

A SenseHat [29] egy bővítőpanel a Raspberry Pi–hez, amelyet az Astro Pi [40] versenyhez fejlesztettek ki. A megmérettetés európai iskolások részére van meghirdetve azzal a céllal, hogy a legjobbak programjai akár a nemzetközi űrállomáson lévő Raspberry–ken is futhatnak. Ennek fényében a fejlesztés egyszerűsítésére törekedtek és egy Python könyvtárral [26] egyszerűen használható. Fizikai összekapcsolása az RPi–vel sem bonyolult, a PIN–ek elhelyezése megegyezik, így csak rá kell helyezni a bővítőpanelt az eszközre. A csomag tartalmaz még négy db csavart, mellyel rögzíthető, továbbá egy erre a célra kialakított házat, melynek egy része levehető, átlátszó műanyag, így kihasználhatók a bővítőpanel előnyei.



16. ábra SenseHat bővítőpanel [41]

Amennyiben ily módon van összekötve a bővítőpanel az RPi–vel, akkor a mért eszköz nem tud kommunikálni a mérővel, hiszen a TXD és RXD PIN–ek is le vannak fedve. A könnyű használhatóság miatt jött létre ez a fizikai konstrukció, viszont a bővítőpanel működéséhez nem szükséges mind a 40 PIN [42]. A használt PIN–ek összekötésével működik a bővítőpanel és a két eszköz közötti kommunikáció is megvalósulhat, ahogyan a 12. ábra és 13. ábra szemlélteti. Ezen felül a hőmérséklet mérése hibával terhelt, hiszen az RPi hőtermelése melegíti alulról a szenzort.

A bővítőpanelen helyet kapott egy 8x8 méretű RGB LED mátrix, egy 5 gombos joystick és a következő érzékelők:

- giroszkóp
- gyorsulásmérő
- mágneses térerősségszenzor
- barométer
- hőmérséklet érzékelő
- páratartalomérzékelő

A felhasználói eset megvalósításához az utolsó három szenzor kerül felhasználásra, mely egy LPS25H [43] áramköri lapkán kapott helyet. A szenzor alapértelmezett beállítása szerint úgynevezett „one shot” konfiguráció szerint működik. A konfiguráció lényege, hogy nem szolgáltat adatot addig, amíg kérés nem érkezik. A kérés egy bit egyesre állításával történik, melyet a meghajtóprogram állít vissza nullára az adat előállta után, jelezve, hogy kiolvasható a mért érték a megfelelő regiszterből. Amennyiben periodikus mért értékekre van szükség, 1 – 25Hz között konfigurálható a szenzor 3 bit megfelelő beállításával.

4.2. Megvalósítás modellezése

Ahogyan a fejezet bevezetője is említi, nem a meteorológiai állomás újragondolása a cél, viszont egy kismértékű specifikáció mindenképp kell. A SenseHat bővítőpanel segítségével mérhető a légnyomás, a hőmérséklet és a páratartalom. Ezeket az értékeket az alkalmazás meghatározott időközönként leméri és lokális SQLite adatbázisban tárolja. WiFi kapcsolat segítségével egy szerverrel szinkronizálja a mért adatokat. A 8x8 LED mátrixon kijelzi az alkalmazás a legutolsó mérés eredményeit, alapértelmezetten a hőmérsékletet.

A specifikált alkalmazás szerver része nem fontos a vizsgálat szempontjából, hiszen a HTTP kérések energiafogyasztása nem függ attól, hogy a szerver milyen műveleteket végez a kapott adatokon. Ezért mind az adatok szinkronizációja, mint Szeged időjárásának lekérdezése teszt szerverhez intézett kérésekként lettek tesztelve.

A specifikáció alapján az alkalmazás fő egységei a következők:

1. a szenzoradatok lekérdezése
2. mért értékek tárolása SQLite adatbázisba
3. adatok lekérdezése az adatbázisból és küldése egy szervernek
4. adatok közlése a LED mátrixon keresztül
5. ütemezés

Amennyiben a specifikáció által definiált program kerülne mérésre, akkor a vizsgált időszak nagy részén az eszköz alvó állapota lenne látható, az időszak nagyságához képest kis időtartamú események (pl. szenzoradatok lekérdezése) a zajban elvesznének. Ennek okán a fenn felsorolt öt funkcionalitás kerül mérésre és összehasonlításra külön–külön.

4.2.1. Python

A Python nyelv kerül elsőnek leírásra, mert a SenseHat bővítőpanel API–ja Python nyelven érhető el [26] [44]. Ahogyan a 4.1 fejezetben kifejtésre került, a bővítőpanelt iskolások versenyeztetésére fejlesztették ki, így törekedtek a programozhatóság egyszerűsítésére. A felállított modell 1. és 5. pontjainak megvalósításához szükséges az API használata.

```
from sense_hat import SenseHat
sense = SenseHat()

temp = sense.get_temperature_from_pressure()
temp = sense.get_temperature_from_humidity()
pressure = sense.get_pressure()
humidity = sense.get_humidity()

sense.show_message('Hello World')
sense.show_letter('A')
sense.set_pixels([ pixels ])
```

A példa kódrészlet szemlélteti, hogy a bővítőpanel kezelése Python nyelvvel egyszerű és karbantartható kódot eredményez. A modell második pontja adatbázisműveleteket említ a fájlrendszeren létező SQLite adatbázishoz kapcsolódva. Ehhez a nyelvbe beépített *sqlite3* [45] eszköztár van a fejlesztők segítségére. A harmadik és a negyedik pont HTTP műveleteket igényel, melyek elvégzéséhez a *requests* [46], még az ütemezés megvalósításához a *sched* [47] modul került felhasználásra.

```
import time
import sched
scheduler = sched.scheduler(time.time, time.sleep)

def function_1():
  scheduler.enter(DELAY, PRIORITY, function_1, ())

scheduler.enter(DELAY, PRIORITY, function_1, ())
scheduler.run()
```

A kódrészlet szerint került megvalósításra az ütemezés. Először egy *scheduler* objektumot kell létrehozni, majd az *enter* metódusát megfelelően felparaméterezve meghívva megvalósítható az ütemezés.

A Python nyelvű megvalósítás során egyszerű programozási konstrukciók segítségével megvalósítható a kívánt funkcionalitás, nem különül el a fordítási és futtatási fázis és nem szükséges ismerni a célarchitektúrát. Ez azért lehetséges, mert a nyelv az interpretált nyelvek közé tartozik, azaz egy dedikált futtatókörnyezet először beolvassa a forrásfájlt, majd végrehajtja azt. Ennél a programozási modellnél csak azok a funkcionalitások vannak biztosítva a futtatókörnyezet által, melyek egy adott verzióban támogatva vannak.

4.2.2. C++

A C++ az iparban egyik legtöbbet használt programozási nyelv a hatékonysága miatt. Elkülönül a fordítási és a futtatási fázis. Fordítás során a fordítóprogram a célhardverre optimalizált binárist állít elő, mely futtatókörnyezet nélkül képes működni. Ez az alapvető lépés feltételezi azt, hogy a fejlesztő tudja milyen architektúrán fog futni az alkalmazása, illetve beszerezte a megfelelő keresztplatformos fordítóprogramot (*cross–platform toolchain*–t).

A SenseHat bővítőpanelhez hivatalos C/C++ API–nak az *RTIMULib* [48] tekinthető, melyet a fejlesztők használhatnak. Ki kell emelni, hogy az említett API nem fedi le a teljes bővítőpanelt, csak a szenzorokat. A joystick és a LED–ek használatához ismerni kell a panel hardver felépítését, a megfelelő pufferek értékeit olvasva és írva érhető el a kívánt funkcionalitás.

Az API–ra alapozva GitHubon található több projekt, melyekkel kiaknázhatók a bővítőpanelen található szenzorok. A használati eset megvalósítása során én is egy nyílt forráskódú projektet [49] használtam fel. A következő kódrészlet szemlélteti a nyomásadatok kiolvasását, melyből látható, hogy a következő adatstruktúrák ismerete szükséges a művelet végrehajtásához: RTIMUSettings, RTPressure, RTIMU_DATA.

```
RTIMUSettings *settings = new RTIMUSettings();
RTPressure *pressure; = RTPressure::createPressure(settings);
pressure->pressureInit();

RTIMU_DATA data;
if (!pressure->pressureRead(data))
  return;

if (!data.pressureValid)
  return;

double pressureValue = data.pressure;
```

Egy ilyen kódrészlet megértése több idő, mint a hivatalos Python API használata, viszont ebben az esetben a fejlesztők sokkal hatékonyabbak kiaknázhatják a szenzor adottságait. A *data* változóban megtalálható a mért adat pontos időbélyege, illetve az első futás során egy alapértelmezett konfigurációs fájl kerül lementésre, mely kézzel testreszabható.

Az adatbázisműveletek a Raspbian csomagkezelőjével elérhető *sqlite 3.6.12*, még a HTTP műveleteket a *curl 7.52.1* segítségével valósultak meg. Fordítás során a felhasznált eszközöket a binárishoz kell linkelni, illetve a futtató eszközön a csomagoknak telepítve kell lenniük.

4.2.3. JavaScript

Hasonlóan a Python nyelvhez, a JavaScript programoknak is szükségük van egy futtatókörnyezetre, amely értelmezi és futtatja a forrásfájlt. A programozási nyelv a webes alkalmazásfejlesztésben vált elterjedté, viszont manapság több beágyazott rendszereket célzó futtatómotor létezik, ilyenre példa a Szegeden is aktívan fejlesztett JerryScript. Ahhoz, hogy az alkalmazásfejlesztés hatékony legyen, nem elég a futtatómotor, egy modulokat támogató keretrendszert kell használni, ami jelen esetben az IoT.js. A keretrendszer lehetővé teszi JavaScript és natív modulok használatát ugyanazon alkalmazásban, így speciális és architektúrafüggő kódrészletek hatékonyan használhatók.

A HTTP küldés/fogadás és a feladatütemezés erősen támogatott az IoT.js–ben, így ezek megvalósítása egyszerű, viszont a SenseHat használatát nem specifikálja egyik ECMAScript szabvány sem. A bővítőpanel funkcionalitásának igénybevételéhez natív modult [49] kellett írni, mely a már meglévő C++ megvalósítást teszi elérhetővé JavaScript alkalmazásból. A keretrendszer rendelkezik modulgenerátorral, mely az egy projekten belüli *header* fájlok alapján legenerál egy úgynevezett *binding* réteget. Ez a réteg felelős a JS – C++ összekapcsolására, a

végrehajtás ide tud eljutni függvényhívások által. A modulgenerátor nagy terhet vesz le a fejlesztő válláról, viszont a különböző objektumok kezelését kézzel kell megoldani (erre kommentek formájában figyelmeztet is). A következő két kódrészlet szemléleti a modul használatát JavaScriptből, illetve egy *binding* függvény vázlatos felépítését.

```
var sense_module = require('sense_hat_module');
var senseHat = new sense_module.SenseHAT();

senseHat.get_temperature_from_humidity();
senseHat.get_temperature_from_pressure();
senseHat.get_pressure();
senseHat.get_humidity();
```

A binding rétegnek lehetővé kell tennie egy szenzor modul létrehozását, illetve a szenzoradatok lekérdezését. A kódrészlet egyszerűsége hasonló a Python nyelvnél illusztrálttal, viszont ehhez a köztes réteg megfelelő megvalósítása szükséges.

```
jerry_value_t SenseHAT_get_humidity_handler(/* … */)
  void *void_ptr;
  bool has_ptr =
    jerry_get_object_native_pointer(
      this_val, &void_ptr, &SenseHAT_type_info);

  if (!has_ptr) { /* hibakezelés */ }

  SenseHAT *native_ptr = (SenseHAT *)(void_ptr);
  double result = native_ptr->get_humidity();
  jerry_value_t ret_val = jerry_create_number(result);

  return ret_val;
```

A kódrészletből kimaradt a paraméterek kezelése, amit el kell végezni a natív függvény hívása előtt (jelen esetben nincsenek bemenő paraméterek). A *handler* függvényt be kell regisztrálni egy egyedi függvénynévvel, ami JavaScriptből elérhető. Ez a név tetszőleges, nem kell megegyezzen a hívott natív függvény nevével, de ajánlatos a végrehajtott funkcionalitással korreláló elnevezést adni.

```
func_name = jerry_create_string((const jerry_char_t *)"get_pressure");
func_ext = jerry_create_external_function(SenseHAT_get_pressure_handler);
jerry_set_property(js_obj, func_name, func_ext);
```

A kódrészlet ábrázolja a függvény hozzáadását a JavaScript *js_obj* objektumhoz *get_pressure* névvel. Az összes használni kívánt natív függvényt ily módon kell összekötni egy objektummal, amit a *require* segítségével lehet elkérni a keretrendszertől.

A bővítőpanel mellett az SQLite adatbáziskezelést is a már elkészített C++ forráskódból készült natív modul vezérli. Az várakozás – *sleep* – is egy olyan funkcionalitás, amely függ az architektúrától, így ezt is natív modulnak kell megvalósítani. A GitHubon található egy modulgyűjtemény [50], amely az általános célú – és publikusan elérhető – modulokat listázza, így megtalálható benne a kért várakozás is.

4.3. A programozási nyelvek összehasonlítása

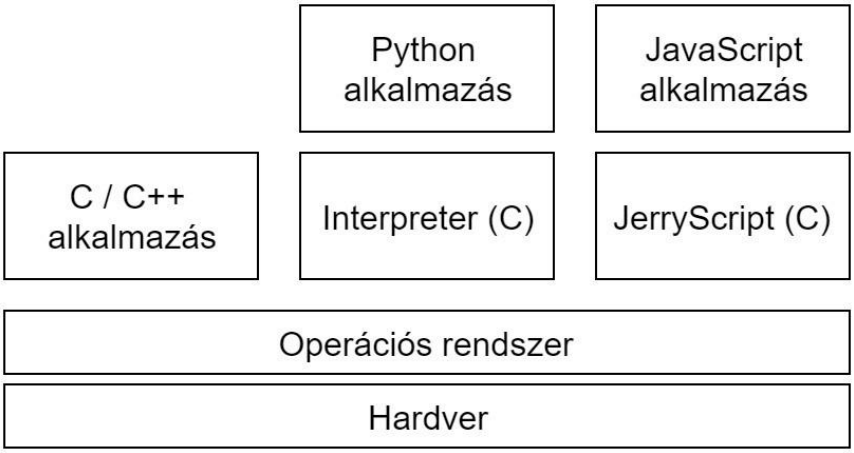
A három bemutatott megvalósítás programozási nyelve közül a Python a legegyszerűbb, utána következik a JavaScript és végül a C/C++. A megvalósítás modellezéséhez és méréséhez használt forráskódok statisztikáit mutatja be a 4. táblázat a kiegészítő fájlok feltüntetésével. Az értékek a fájlokban található nem üres programsorokat jelképezik (Source Lines of Code – SLOC).

Név	C++	Python	JavaScript
01_sensors	28	17	17
02_sqlite	95	33	10
03_http	76	31	42
04_led_matrix	172	43	154
05_scheduling	38	16	10
SenseHat	172	/	172
JavaScript bindings	/	/	348
∑	581	90	671

4. táblázat A megvalósítások összehasonlítása SLOC metrikával

Az egyes megvalósítások értékeit összeadva, látható, hogy a Python bizonyult a legkézenfekvőbb megoldásnak, majd a C++ és a JavaScript. Az átlag fejlesztők számára – akik a már kész natív JavaScript modulokkal dolgoznak – a táblázat nem pontos eredményt mutat, hiszen számukra a SenseHat és a JavaScript bindings sorok láthatatlanok (lásd. Python oszlopa).

Amennyiben a kód fenntarthatósága és érthetősége az elsődleges cél egy projekt során, akkor a forrásfájlok statikus vizsgálatából a Python és a JavaScript nyelvek vonzóbb választást nyújtanak, mint a natív C/C++. Szemlélve a 17. ábra tartalmát viszont az említett két nyelv hátrányból indul, amikor az alacsony sebesség és energiafogyasztás az elsődleges cél.



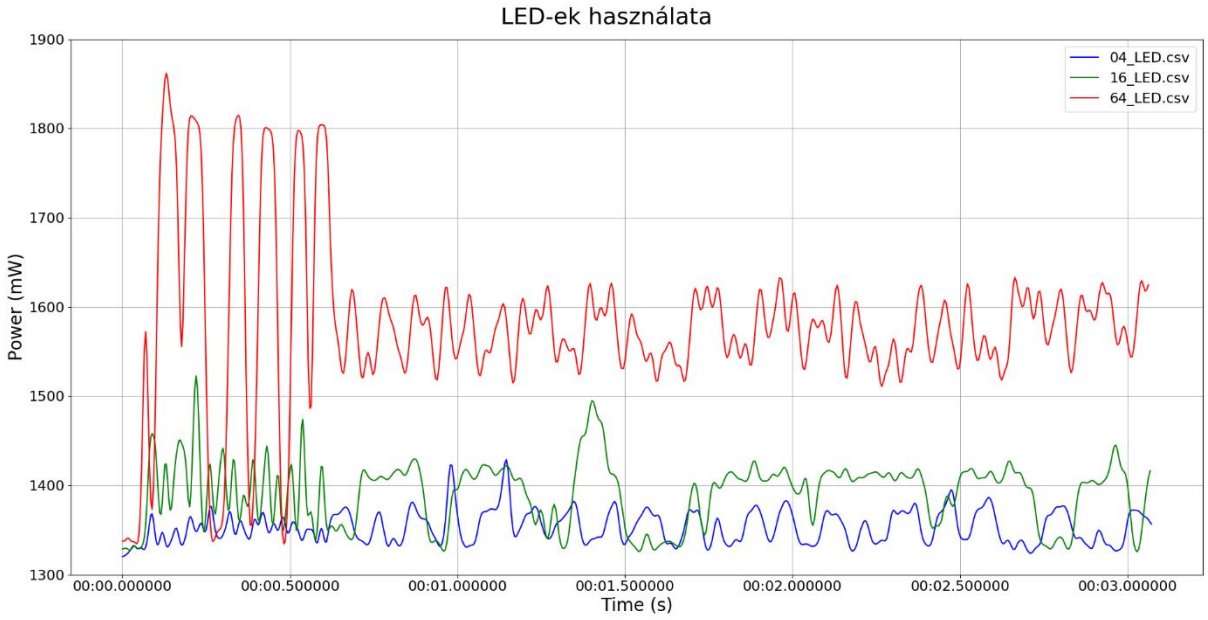
17. ábra A különböző nyelvek viszonya az operációs rendszerhez

A 5. táblázat a felállított modell feladatainak energiafogyasztását szemlélteti. Minden esetben gyors lefutású feladatokról van szó (<1 másodperc), így az értékek átlagértékek, tíz mérés átlagolásával készültek a zaj szűrése érdekében. A táblázatban szereplő értékek nagyon hasonlóak, a különbségek túl kicsik ahhoz, hogy érdemi következtetést lehessen belőlük levonni.

Név	C++ (Ws)	Python (Ws)	JavaScript (Ws)
hőm. a nyomásérzékelőből	1,337	1,337	1,336
hőm. a páratartalomérzékelőből	1,346	1,332	1,359
nyomás	1,320	1,329	1,341
páratartalom	1,321	1,332	1,328
adatmentés (db)	3,216	3,215	3,149
adat lekérdezés (db)	1,324	1,334	1,332
HTTP POST	3,099	3,141	3,081
HTTP GET	1,511	1,514	1,513
4 LED használata	5,405	5,392	5,414
16 LED használata	5,546	5,550	5,525
64 LED használata	6,154	6,232	6,226

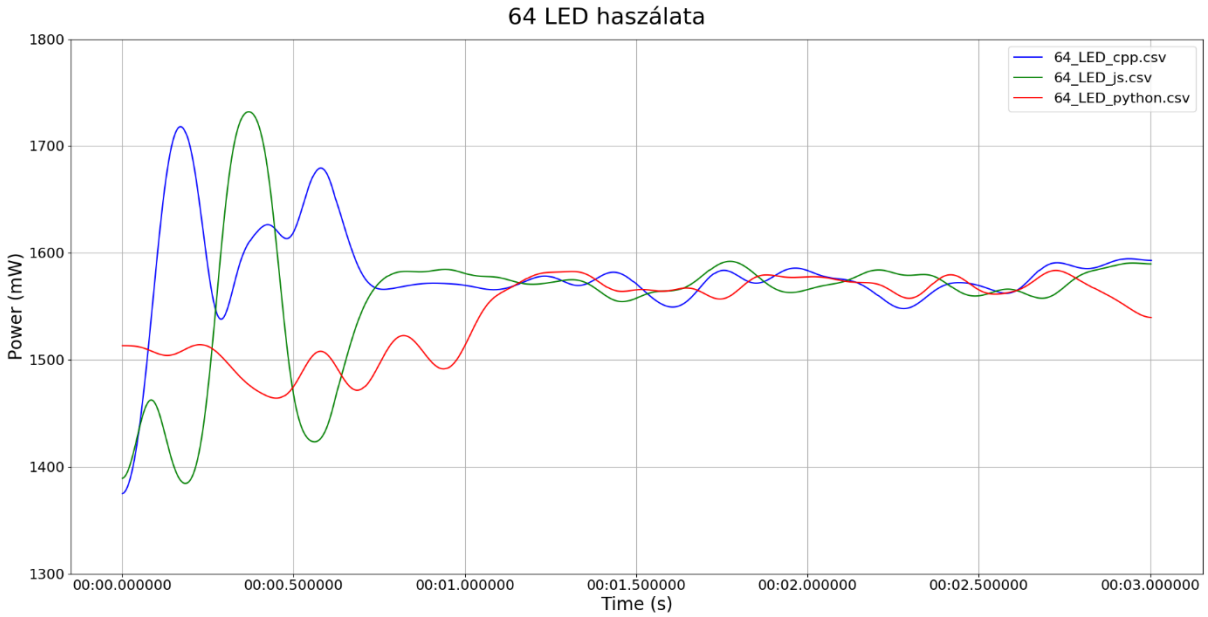
5. táblázat A megvalósítások energiafogyasztásának összehasonlítása

Ezen információk alapján azt ki lehet jelenteni, hogy a másodperc, vagy annak töredékéig tartó események mérése nem szolgáltat érdemi információval, mert a zajban elveszik a hasznos adat. A LED–ek használata hosszabb intervallumot – 3 másodpercet – ölel fel, így az ábrázolás is több információt nyújt. Az alábbi ábrán látható, ahogy először 4 LED kerül bekapcsolásra (kék), majd 16 (zöld) és végül mind a 64 LED fehéren világít (piros). A bekapcsolás C++ programmal történt. Az első fél másodperc az izzók bekapcsolását ábrázolja, majd a továbbiakban a bekapcsolt izzók energiafogyasztását, amikor a program maga alvó állapotban van.



18. ábra 4, 16 és 64 LED használatának energiafelhasználása

Az ábra szemlélteti egy programozási nyelvből származó szoftver energiafogyasztását, viszont a másik két nyelv viselkedéséről nem ad információt. Ezt a 19. ábra mutatja be. A C++ (kék) gyors felfutással indít és némi periodicitás után beáll egy stabil állapotba. A JavaScript (zöld) program is hasonlóan viselkedik, viszont ott a felfutási ideje elmarad a referenciájától. Végül a Python program (piros) nem produkált túllövést és periodikus beállást, hanem lassú és egyenletes energiafelhasználásbéli növekedés tapasztalható az esetében. Miután az alkalmazások egyenként bekapcsolták a LED izzókat, alvó állapotba kerülnek és csak az izzók működésének energiafogyasztása látható, ami konstansnak tekinthető.



19. ábra 64 LED bekapcsolásának és működtetésének összehasonlítása

Egy alkalmazás megvalósításához választott programozási nyelv nagyban függ attól, hogy milyen nyelvet támogat elsősorban egy adott eszköz. Jelen esetben a SenseHat bővítőpanel Python API–val rendelkezik és a mérésekből látható (5. táblázat, 19. ábra), hogy a hivatalos iránymutatásokat használva optimálisabb működést és alacsonyabb energiafogyasztást lehet elérni. Ezen felül a C++ nyelvhez képest a Python és a JavaScript könnyebben tanulható, ami hobbi fejlesztők számára vonzóbbá teszi őket. Ettől függetlenül a C++ alacsonyszintű hozzáférést biztosít a hardverhez, ami idő és biztonságkritikus rendszerek fejlesztésénél előnyt jelent. Az iparban nem feltétlenül a gyors tanulási ráta jelenti a piaci előnyt, hanem a hatékonyság, amit javítani lehet. Amennyiben a hivatalos API kerül felhasználásra, akkor a publikusan elérhető metódusokon nem lehet optimalizálni, viszont a nyílt forráskódú API–k és meghajtóprogramok testreszabhatók egy–egy alkalmazás igényeit figyelembe véve.

A gyors lefutású – 1 másodperctől gyorsabb – tesztek eredményei alapján azt lehet megállapítani, hogy ilyen rövid intervallumot nem érdemes mérni 200Hz mintavételezési frekvenciával, mert a zajban elveszik a hasznos adat. Hasonló következtetésre jutottak grafikus kártyák energiafelhasználásának mérésénél [5] is, ahol a mintavételezési frekvencia növelésével egy idő után nem tapasztaltak pontosságjavulást.

5. A GCC optimalizációk és az energiafogyasztás

Az előző fejezetből kiderült, hogy pillanatnyi futásidejű parancsok, algoritmusok mérése nem informatív, ezért ez a fejezet egy hosszabb lefutású algoritmust vizsgál. A 3. fejezetben a Fibonacci sorozat első 500 elemének kiszámítása és fájlba írása volt a példa (7. ábra), melyen bemutatásra került az energiamérés megvalósítása. A fejezet első fele ennek egy mutációját használja, amikor is az előállt számsorozat először megfordításra kerül, majd beszúró rendezéssel újra növekvő sorrend áll elő. Ezután a Polybench benchmark *3mm* tesztje kerül vizsgálat alá annak érdekében, hogy kiderüljön egy 10 percnél hosszabb futásidejű alkalmazás esetében milyen változásokat eredményez a fordítóprogram optimalizációja.

A GCC [51] fordítóprogram több optimalizációs kapcsolóval [52] rendelkezik, melyekkel ugyanabból a forrásfájlból különböző binárisokat képes előállítani. Referenciaként a Python és JavaScript megvalósítások szolgálnak és a vizsgálat tárgya az, hogy a C++ megvalósítás miként szerepel futásidőben és energiafelhasználásban a fordítóprogram különböző optimalizációs eljárásaira hagyatkozva. A felhasznált kapcsolók az alábbiak:

- -O0: rövid fordítási időre törekszik, optimalizáció nélkül – ez az alapértelmezett;
- -O1: a fordítóprogram próbál kisebb bináris méretet és rövidebb futásidőt elérni, de nem hajt végre olyan optimalizációkat, melyek nagymértékben növelik a fordítási időt;
- -O2: a fordítóprogram minden optimalizációt végrehajt, amit -O1 esetében is, viszont olyan műveleteket is végez, amik megnövelik fordítási időt és a bináris méretét;
- -O3: a fordítóprogram minden optimalizációt végrehajt, amit -O2 esetében is és olyan műveleteket is végez, melyek nagymértékben megnövelik a fordítási időt és a binárist;
- -Os: minél kisebb bináris méretre törekszik, megegyezik -O2-vel, de azok a műveletek ki vannak hagyva, melyek növelik a bináris méretét (pl. inline).

A C++ és a JavaScript megvalósítás algoritmusa megegyezik a referencia Python szkript algoritmusával, ami fontos az energiafelhasználás és a futásidő összehasonlításánál. Az, hogy a sorozat hány elemét kell kiszámítani parancssori argumentumként kerül átadásra, így kiküszöbölve a fordítóprogram konkrét értékre optimalizálását (a vizsgálat során *1400* volt az input). A futási eredményeket a 6. táblázat mutatja be. Az első két oszlop tíz futás átlagának eredménye.

Az utolsó oszlop esetében a leglassabbnak bizonyult futásidőhöz igazodott a többi. A leglassabb Python futási eredmény 3,215 másodperces volt, így az összes mérési eredmény végére, 3,215 másodperc időbélyeggel egy nyugalmi állapotot illusztráló sor került, ezzel szimulálva azt, hogy az algoritmus futtatása után nyugalmi állapotba tér vissza az eszköz, miközben energiát fogyaszt. Ezt az oszlopot szemlélve jobban elkülönül az egyes optimalizációs kapcsolók hatása az energiafogyasztásra. Az optimalizálatlan bináris energiafogyasztásához képest 0,1 Ws a nyereség, melyet az különböző optimalizációk eredményeztek. Jelen esetben az összes optimalizációs szint közel ugyanazt a fogyasztást indukálta. A 100 mWs fogyasztáskülönbségek jelentéktelennek tűnnek, viszont hosszútávon egy tápfeszültségként szolgáló akkumulátor élettartamát lerövidíthetik.

Név	Időtartam (s)	Energiafogyasztás (Ws)	Energiafogyasztás ugyanazon intervallumra (Ws)
Python	3,181	5,822	5,873
JavaScript	1,479	2,879	5,432
C++ -O0	0,188	**0,274**	4,476
C++ -O1	0,121	**0,179**	4,559
C++ -O2	0,126	**0,178**	4,291
C++ -O3	0,131	**0,183**	4,303
C++ -Os	0,129	**0,180**	4,283

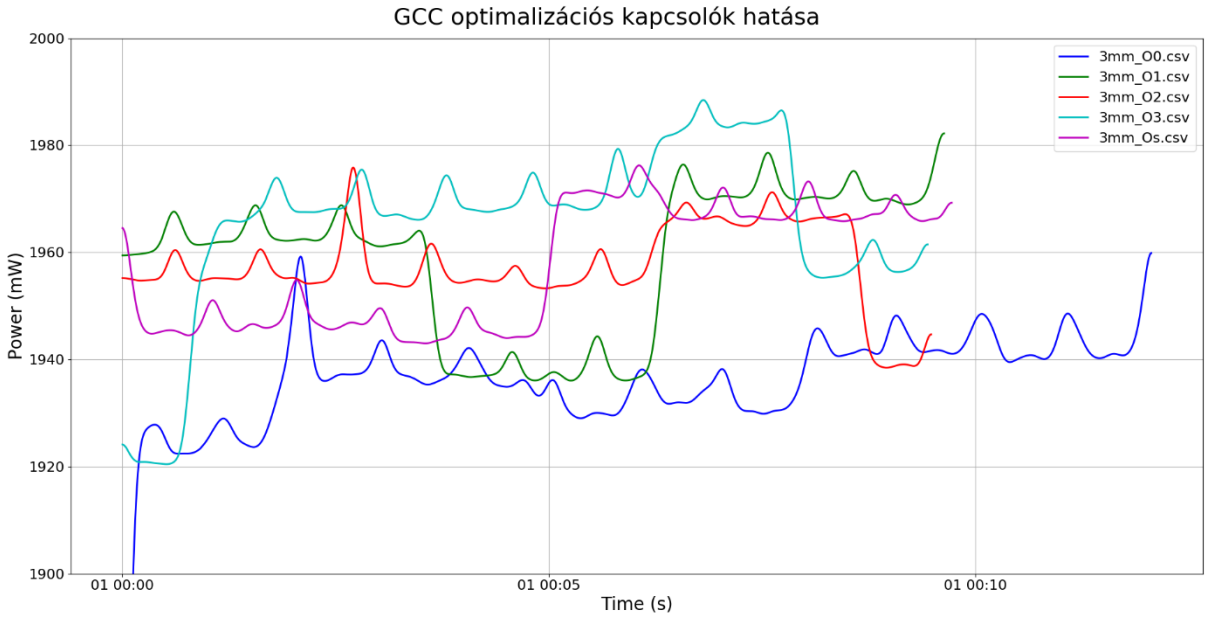
6. táblázat A GCC fordító optimalizációs kapcsolóinak hatása

A Polybench *3mm* futásának kiértékelése esetén (7. táblázat) is hasonló megállapításra juthatunk. Az optimalizálatlan bináris futásához képes az O1 kapcsoló 270 mWs fogyasztásjavulást eredményezett, amin az O2 és O3 kapcsolók 20 mWs–al javítottak. Az Os kapcsoló visszaesést jelent mind futásidőben (17 másodperc), mind fogyasztásban 10 mWs. A táblázatban szereplő értékek szintén tíz futás átlagolásával készültek.

Név	Időtartam (s)	Energiafogyasztás (Ws)	Energiafogyasztás ugyanazon intervallumra (Ws)
3mm -O0	723,919	**1.401,475**	1.401,475
3mm -O1	578,018	**1.132,378**	1.346.603
3mm -O2	569,000	**1.113,831**	1.341,524
3mm -O3	566,457	**1.113,461**	1.345,506
3mm -Os	583,449	**1.142,001**	1.349,280

7. táblázat Optimalizációs kapcsolók hatása a 3mm tesztre

A 20. ábra szemlélteti az energiafogyasztás változását a futások során. Látható, hogy a futásidőt minél inkább csökkenteni próbáló optimalizációs szintek egyre nagyobb teljesítményen működtették az eszközt, így egyre rövidebb futásidőt értek el. Az ábrát szemlélve a fogyasztás a görbe alatti terület. A kísérletek alapján ez a terület kisebb abban az esetben, ha nagyobb teljesítménnyel, de rövidebb ideig terheljük az eszközt.



20. ábra Optimalizációs kapcsolók hatása a 3mm tesztre

6. Összefoglalás

A dolgozat során célom egy IoT eszközök energiafogyasztásának vizsgálatát támogató eszköz tervezése és megvalósítása volt. Ennek érdekében először definiálásra került mit is tekintünk beágyazott eszköznek, miért fontos az eszközök közötti kommunikáció, ami az IoT koncepció magját alkotja. Az IoT eszközök nagyrésze – a mért eszköz is – ARM processzorral felszerelt, így az architektúra is bemutatásra került, majd az energia fogalma és az energiafogyasztás mérése is.

A mérő eszköz egy Raspberry Pi 3B+ egylapkás számítógép INA219 árammérő szenzorral kiegészítve. Az érzékelőn keresztülfolyik a mért eszköz tápellátása, így az mérhető és menthető későbbi feldolgozás céljából. A mérés rögzített, 200Hz–es mintavételezéssel zajlott. A mért alkalmazás nem fekete dobozként volt tekintve, kiegészítő kódrészlettel kellett ellátni a mérések indításához és leállításához. A módszer előnye, hogy így pontosan mérhető mely kódrészlet indukálta az energiafogyasztást. Ezen felül a mérési eredmények egy lokális webszerveren elérhetők így könnyítve azon kutatók és fejlesztők munkáját, akik használni szeretnék az eszközt.

Az elkészült eszköz működése helyességének bizonyítására több teljesítménytesztet is lemértem és az eredményeket a dolgozatban publikáltam. Ezen felül egy népszerű IoT projektet modelleztem és megvalósítottam C++, JavaScript és Python nyelveken. A szenzoradatok gyors lefutású lekérdezése során szignifikáns különbséget nem tapasztaltam a megvalósítások között, a LED izzók használata esetében pedig a hivatalosan elérhető Python API bizonyult a legköltséghatékonyabbnak.

Az algoritmusok vizsgálata esetében már szignifikáns különbség tapasztalható a programozási nyelvek között. A Fibonacci sorozat számítása esetén a C++ alkalmazás a versenytársai idejének töredéke alatt végzett, mindezt alacsonyabb átlagfogyasztással. A GCC fordító kapcsolói segítségével az eredmény tovább optimalizálható, az O1, O2 és O3 kapcsolók a fogyasztásra pozitív hatással vannak, még az Os kapcsoló hosszabb futásidővel és magasabb átlagfogyasztással jár.

A kísérleti eredmények alapján a JavaScript és a Python nyelveket ajánlom amennyiben az alacsony fejlesztési és karbantartási költség az elsődleges cél. Ha a készülő alkalmazás követelményei között az alacsony energiafogyasztás és a rövid válaszidő szerepel, akkor a C++ programozási nyelv előnyösebb megoldást kínál. A nyelv hardverközeli konstrukciói lehetővé teszik hatékony szoftverek írását, viszont ez előképzettséget és hosszabb betanulási időt igényel. Ezen felül a fordítóprogram a célhardverre optimalizált binárist állít elő.

A megvalósított projekt publikusan elérhető [53], az eredmények reprodukálásához szükséges részletes leírással és kapcsolási rajzokkal ellátva.

Irodalomjegyzék

[1] I. Sommerville, Szofterrendszerek fejlesztése, Budapest: Panem Könyvkiadó Kft., 2007.

[2] I. Ú. MARQUÉS, Energy Efficiency in Wireless Communications for Mobile User Devices, Madrid: Complutense University of Madrid, 2018.

[3] U. Awada, K. Li és Y. Shen, „Energy Consumption in Cloud Computing Data Centers," *International Journal of Cloud Computing and Services Science (IJ-CLOSER),* %1. kötetIII, pp. 31-47, 2014.

[4] T. Makris, Measuring and Analyzing Energy Consumption (MSc Thesis), Aakti University, 2017.

[5] Á. Kiss, „Repara-Project," 2 március 2015. [Online]. Available: http://repara-project.eu/wp-content/uploads/2015/03/ICT-609666-D6.4.pdf. [Hozzáférés dátuma: 10 szeptember 2019].

[6] „Fritzing," [Online]. Available: https://fritzing.org/home/. [Hozzáférés dátuma: 13 november 2019].

[7] „Fritzing - GitHub," [Online]. Available: https://github.com/fritzing. [Hozzáférés dátuma: 13 november 2019].

[8] R. Toulson és W. Tim, Fast and Effective Embedded Systems Design (Second Edition), Elsevier Ltd., 2017.

[9] A. Fodor és Z. Vörösházi, Beágyazott rendszerek és programozható logikai vezérlők, Typotex, 2011.

[10] J. Cass, „Internet of Things: What It Is, How It Works, Examples and More," 19 november 2018. [Online]. Available: https://justcreative.com/2018/11/19/internet-of-things-explained/. [Hozzáférés dátuma: 10 szeptember 2019].

[11] M. Eshaghi, „An Energy Harvesting Solution for IoT Sensors," 2018. [Online]. Available: https://scholar.uwindsor.ca/cgi/viewcontent.cgi?article=8488&context=etd. [Hozzáférés dátuma: 10 szeptember 2019].

[12] J. Morgan, „A Simple Explanation Of 'The Internet Of Things'," 13 05 2014. [Online]. Available: https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#694f68d21d09. [Hozzáférés dátuma: 10 szeptember 2019].

[13] A. D. Rayome, „Top programming languages IoT developers should learn," 17 április 2019. [Online]. Available: https://www.techrepublic.com/article/top-programming-languages-iot-developers-should-learn/. [Hozzáférés dátuma: 10 szeptember 2019].

[14] „JavaScript engine for Internet of Things," JS Foundation , [Online]. Available: https://jerryscript.net/. [Hozzáférés dátuma: 27 október 2019].

[15] „Windows 10 on ARM," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows/arm/. [Hozzáférés dátuma: 14 szeptember 2019].

[16] „ARM Server," GIGA-BYTE Technology Co., Ltd. , [Online]. Available: https://www.gigabyte.com/ARM-Server. [Hozzáférés dátuma: 13 november 2019].

[17] C. M. Paradis, Detailed Low-Cost Energy and Power Monitoring of Compuding Systems, University of Maine, 2013.

[18] „Cortex-A53," ARM Ltd., [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53. [Hozzáférés dátuma: 14 szeptember 2019].

[19] „Raspberry Pi," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi. [Hozzáférés dátuma: 14 szeptember 2019].

[20] G. Miklós, Méréselmélet, Typotex, 2011.

[21] H. László, Elektrotechnika, Győr: Széchényi István Egyetem, 2006.

[22] A. Viswanathan, „Analysis of Power Consumption of the MQTT Protocol," University of Pittsburgh, Pittsburgh, 2017.

[23] S. Schubert, D. Kostic, W. Zwaenepoel és K. G. Shin, „Profiling Software for Energy Consumption," *IEEE International Conference on Green Computing and Communications*, pp. 515-522, 2012.

[24] a. A. L. a. H. F. Mikko Roth, „Measuring and Modeling Energy Consumption of Embedded Systems for Optimizing Compilers," Hamburg University of Technology, Hamburg, 2018.

[25] T. Instruments, „INA219," Texas Instruments, [Online]. Available: http://www.ti.com/product/INA219. [Hozzáférés dátuma: 25 szeptember 2019].

[26] PyPi, „pi-ina219 - PyPi," Python Software Foundation, [Online]. Available: https://pypi.org/project/pi-ina219/. [Hozzáférés dátuma: 26 szeptember 2019].

[27] „The Raspberry Pi UARTs - Raspberry Pi Documentation," Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/documentation/configuration/uart.md. [Hozzáférés dátuma: 3 október 2019].

[28] C. Liechti, „pySerial 3.4 documentation," [Online]. Available: https://pyserial.readthedocs.io/en/latest/index.html. [Hozzáférés dátuma: 3 október 2019].

[29] „SenseHat - Raspberry Pi," Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/products/sense-hat/. [Hozzáférés dátuma: 5 október 2019].

[30] G. Linux, „Sysbench - Gentoo Wiki,” Gentoo Linux, [Online]. Available: https://wiki.gentoo.org/wiki/Sysbench. [Hozzáférés dátuma: 15 október 2019].

[31] O. University, „PolyBench / C,” Ohio University, [Online]. Available: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench. [Hozzáférés dátuma: 15 október 2019].

[32] „iPerf - The ultimate speed test tool for TCP, UDP and SCTP,” [Online]. Available: https://iperf.fr/. [Hozzáférés dátuma: 23 október 2019].

[33] „ElectroniClinic,” [Online]. Available: https://www.electroniclinic.com/top-10-arduino-iot-projects-2019-2020-with-tutorials-iot-projects-using-arduino/. [Hozzáférés dátuma: 27 október 2019].

[34] „Iot Tech Trends,” [Online]. Available: https://www.iottechtrends.com/iot-projects-do-it-yourself-raspberry-pi/. [Hozzáférés dátuma: 27 október 2019].

[35] „Hackster,” [Online]. Available: https://www.hackster.io/search?i=projects&q=weather%20station. [Hozzáférés dátuma: 27 október 2019].

[36] „Element14,” [Online]. Available: https://www.element14.com/community/docs/DOC-82034/l/ibm-watson-iot-platform-using-raspberry-pi3-board. [Hozzáférés dátuma: 27 október 2019].

[37] „Python 3.6.9 documentation,” [Online]. Available: https://docs.python.org/3.6/. [Hozzáférés dátuma: 27 október 2019].

[38] „Platform for Internet of Things with JavaScript,” JS Foundation, [Online]. Available: https://iotjs.net/. [Hozzáférés dátuma: 27 október 2019].

[39] „ECMA-252 Edition 5.1,” [Online]. Available: https://www.ecma-international.org/ecma-262/5.1/. [Hozzáférés dátuma: 27 október 2019].

[40] „Astro Pi,” [Online]. Available: https://astro-pi.org/. [Hozzáférés dátuma: 31 október 2019].

[41] D. Honess, „Astro Pi: Flight Hardware Tech Specs,” [Online]. Available: https://www.raspberrypi.org/blog/astro-pi-tech-specs/. [Hozzáférés dátuma: 31 október 2019].

[42] „Raspberry Pi Pinout - SenseHAT,” [Online]. Available: https://pinout.xyz/pinout/sense_hat. [Hozzáférés dátuma: 31 október 2019].

[43] „DS - LPS25H - MEMS pressure sensor: 260-1260 hPa absolute digital output barometer,” STMicroelectronics, október 2018. [Online]. Available: https://www.st.com/resource/en/datasheet/lps25h.pdf. [Hozzáférés dátuma: 13 november 2019].

[44] „API Reference - SenseHat,” [Online]. Available: https://pythonhosted.org/sense-hat/api/. [Hozzáférés dátuma: 13 november 2019].

[45] „DB-API 2.0 interface for SQLite databases,” [Online]. Available: https://docs.python.org/3.6/library/sqlite3.html. [Hozzáférés dátuma: 13 november 2019].

[46] „requests - PyPi,” [Online]. Available: https://pypi.org/project/requests/. [Hozzáférés dátuma: 13 november 2019].

[47] „sched — Event scheduler,” [Online]. Available: https://docs.python.org/3.6/library/sched.html. [Hozzáférés dátuma: 13 november 2019].

[48] „RTIMULib - GitHub,” RPI-Distro, [Online]. Available: https://github.com/RPi-Distro/RTIMULib. [Hozzáférés dátuma: 15 november 2019].

[49] „SenseHat - C++ interface to the Raspbery Pi Sense HAT,” [Online]. Available: https://github.com/vincedani/SenseHat. [Hozzáférés dátuma: 15 november 2019].

[50] „IoT.js modules,” [Online]. Available: https://github.com/jerryscript-project/iotjs-modules. [Hozzáférés dátuma: 15 november 2019].

[51] „GCC, the GNU Compiler Collection,” Free Software Foundation, Inc., [Online]. Available: https://gcc.gnu.org/. [Hozzáférés dátuma: 16 november 2019].

[52] „GCC - Options That Control Optimization,” Free Software Foundation, Inc., [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. [Hozzáférés dátuma: 16 november 2019].

[53] D. Vince, „Energy - measurement,” [Online]. Available: https://github.com/vincedani/energy-measurement. [Hozzáférés dátuma: 30 november 2019].

Nyilatkozat

Alulírott Vince Dániel programtervező informatikus MSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus MSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2019. november 30.

Aláírás

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőimnek, Dr. Kiss Ákosnak és Lóki Gábornak, akik észrevételeikkel és hasznos tanácsaikkal segítették a munkámat, továbbá közvetlen kollégáimnak, akikhez mindig fordulhattam technikai tanácsért.

Köszönöm a családomnak és élettársamnak, hogy szeretetükkel és türelmükkel támogattak egyetemi tanulmányaim alatt.

Függelék

Polybench benchmark mérési eredményei:

Név	Átlagos teljesítmény (mW)	Időtartam (s)	Energiafogyasztás (Ws)
2mm	1909,060	486,286	928,356
3mm	1925,840	719,174	1385,017
adi	1933,533	26,139	50,539
atax	1792,732	3,636	6,517
bicg	1768,886	3,577	3,327
cholesky	1888,215	8,052	15,211
correlation	1929,999	56,042	108,164
covariance	1940,102	55,742	108,155
doitgen	1885,190	20,635	38,307
durbin	1926,357	7,315	14,094
dynprog	1877,706	12,351	23,200
fdtd-2d	1976,427	12,502	24,718
fdtd-ampl	1903,118	18,204	34,652
floyd-warshall	1967,220	69,262	136,526
gemm	1932,733	244,168	471,927
gemver	1923,171	7,132	13,728
gesummv	1826,268	4,242	7,748
gramschmidt	1930,672	55,335	106,840
jacobi-1D	1564,287	1,348	2,104
jacobi-2D	1874,773	3,737	7,015
lu	1957,280	20,857	40,827
ludcmp	1917,345	25,635	49,154
mvt	1838,336	6,790	12,484
reg-detect	1672,517	2,489	4,167
seidel	1802,067	4,417	7,967
symm	1963,407	332,351	652,544
syr2k	1929,242	121,662	234,714
syrk	1917,983	61,807	118,552
trisolv	1731,360	2,534	4,388
trmm	1909,511	31,797	60,719