UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tamara Vinko

PROGRAMIRANJE DELOVANJA PREPROSTEGA ROBOTA

DIPLOMSKO DELO

VISOKOŠOLSKEGA STROKOVNEGA ŠTUDIJA

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00117/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TAMARA VINKO**

Naslov: **PROGRAMIRANJE DELOVANJA PREPROSTEGA ROBOTA**
THE PROGRAMMING OF SIMPLE ROBOT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Proučite ogrodje .NET Micro Framework in v okviru tega raziščite, za katere vrste strojne opreme je s tem ogrodjem možno razvijati programsko opremo. V okviru raziskave se osredotočite na programsko opremo za robote. S pomočjo ogrodja razvijte tudi programsko opremo, ki upravlja delovanje preprostega robota. Programska oprema mora zagotoviti, da se robot med premikanjem izogiba oviram in pri premikanju na mizi ne pade iz mize.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Zahvala

Zahvaljujem se svojemu mentorju doc. dr. Roku Rupniku za napotke, prijaznost ter za pomoč, ki mi jo je nudil v času pisanja diplomske naloge. Zahvaljujem se tudi mojim staršem ter možu, ki so mi v času študija in pisanja diplomske naloge stali ob strani in me podpirali.

Kazalo

1. Uvod 1
1.1. Motivacija 1
1.2. Uporabljene tehnologije 1
1.3. .NET Micro Framework 1
1.4. Micro .NET naprave 3
1.4.1. Freescale I.MXS 3
1.4.2. Device Solutions Meridian and Tahoe 4
1.4.3. Digi Connect ME 5
1.4.4. Digi ConnectCore 9P 9215 6
2. Predstavitev problema 7
2.1. Pričakovani rezultati 7
2.2. Izbira razvojnega okolja 7
2.3. Izbira strojne opreme 8
3. Zmožnosti .NET Micro Frameworka 10
3.1. Uvod v .NET Micro Framework 10
3.2. Razvojno orodje za izdelavo Micro Framework aplikacij 10
3.3. Priprava okolja za izdelavo MF aplikacij 11
3.4. Dostop do strojne opreme 13
3.4.1. GPIO porti 13
3.4.2. RS232 Serijski port 16
3.4.3. I2C vodilo 16
3.5. Mreža 17
3.5.1. Vtiči 17
3.5.2. Web Services 18
3.6. Brezžična komunikacija 19
3.6.1. Brezžični LAN 19
3.6.2. Bluetooth 19
3.6.3. ZigBee 20
3.6.4. Z-WAVE 21
3.7. Kriptografija 22
3.7.1. XTEA algoritem 23
3.7.2. RSA Asimetrični algoritem 23

4. Izdelava aplikacije 25
4.1. Predstavitev razvojnega jezika 25
4.2. Emulator strojne opreme 26
4.3. Mehanski deli robota 29
4.3.1. Elektromotorji 29
4.3.2. Refleksivni senzorji 30
4.4. Implementacija 31
4.4.1. Razred ReflectiveSensor 31
4.4.2. Razred FEZMini_Robot 31
4.4.3. Razred Program – vstopna točka aplikacije 33
4.4.4. Slikovni prikaz delovanja aplikacije 35
5. Zaključek 38
5.1. Težave in omejitve 38
5.2. Ideje za nadaljni razvoj 38
6. Literatura 39
7. Izjava 40

Kazalo slik

Slika 1: Freescale I.MXS 3
Slika 2: Device Solutions 4
Slika 3: Digi Connect ME 5
Slika 4: Digi Connect Core 6
Slika 5: FEZ Mini 9
Slika 6: Primerjava posameznih različic Visual Studia 11
Slika 7: Izbira ustreznega vzorca 12
Slika 8: Razvijalsko orodje za izdelavo .NET Micro Framework aplikacij 13
Slika 9: DPWS arhitektura 19
Slika 10: Primer ZigBee omrežja 20
Slika 11: Primer Z-Wave omrežja 21
Slika 12: Primerjava WLAN, Bluetooth, ZigBee in Z-Wave tehnologij 22
Slika 13: Povezava C# jezika in Visual Studia 25
Slika 14: Razvoj C# jezika 25
Slika 15: Micro Framework vzorec 26
Slika 16: Zagona aplikacije v emulator načinu 28
Slika 17: Razhroščevanje aplikacije 29
Slika 18: Elektromotorji 29
Slika 19: Senzorji refleksije 30
Slika 20: Robot s senzorji refleksije, FEZ kartico in elektromotorji 35
Slika 21: Robot v premikanju 36
Slika 22: Robot zazna oviro 37

Povzetek

Namen diplomske naloge je predstaviti Micro .NET Framework razvojno okolje in njegovo praktično uporabo. Izdelava aplikativnega dela bo potekala v Microsoft Visual Studio Express okolju, strojnega pa na FEZ Mini kartici. MS Visual Studio Express je brezplačno orodje za izdelavo aplikacij. Potrebno je namestiti predlogo Micro Framework, ki je prav tako brezplačna. Za pisanje programske kode sem uporabila C#.

V uvodnem delu bom predstavila različne proizvajalce strojne opreme, ki podpira .NET Micro Framework arhitekturo.V nadaljevanju bom razložila, zakaj je bila moja odločitev za izdelavo sistema Visual Studio razvojno okolje. Predstavila bom zmožnosti .NET Micro Frameworka. Tu si bomo podrobneje ogledali način dostopa do strojne opreme preko GPIO in RS232 portev, ter I2C vodila. Ogledali si bomo, kako se napravice povezujejo med seboj, z osebnimi računalniki in s senzorji preko brezžičnega omrežja, Bluetootha, ZigBee-a in Z-WAVE-a. Naslednje poglavje bo govorilo o kriptografiji v .NET Micro Frameworku, saj je le ta nujno potrebna pri prenosih podatkov med posameznimi napravami. Poglavje, ki opisuje izdelavo aplikacije, se začne s kratko predstavitvijo programskega jezika, ki sem ga uporabila za izdelavo diplomske naloge. Sledi kratek opis strojne opreme, nato pa opis implementacije rešitve. V tem delu sem opisala gonilnike za senzorje refleksije in elektromotorje. Na koncu sem podala ideje za nadaljni razvoj.

KLJUČNE BESEDE: FEZ mini, C#, microsoft .NET framework, brezžično omrežje, analogni vhodi, I2C vodilo

Abstract

The purpose of the diploma thesis is to present the Micro .NET Framework developer environment and the practical use thereof. The creation of applicative portion will be implemented using the Microsoft Visual Studio Express environment while the hardware will be implemented on an FEZ Mini Micro SD card. MS Visual Studio Express is a free tool for developing applications. The Micro Framework which is also free, must first be installed. C# was used to write the programming code.

Various manufacturers of hardware equipment which support .NET Micro Framework architecture are presented in the introduction. The author then describes her decision to design a Visual Studio development environment system in continuation. The capabilities of .NET Micro Framework are presented in continuation as well as an in-depth look of the manner of accessing hardware via GPIO and RS232 ports and I2C buses. The thesis will also explain how the devices connect to each other, e.g. through personal PCs and sensors via a wireless network, Bluetooth, ZigBee and Z-WAVE. The next chapter will cover cryptography in the .NET Micro Framework as a requirement for transferring data between individual devices. This following chapter which describes the creation of an application begins with a short presentation of the programming language used to implement the diploma task. This is followed by a short description of the hardware used and subsequently, a description of the solutions implemented. The chapter also describes the drivers required for sensory reflection and electric motors. Ideas for continued development are presented at the end of the thesis.

KEYWORDS: FEZ mini, C#, Microsoft .NET framework, wireless network, analog inputs, I2C bus

1. Uvod

1.1. Motivacija

Eden izmed razlogov, zaradi katerih bom v nadaljevanju opisovala Micro .NET Framework, je njegova praktična uporabnost. Živimo v času, kjer so nam informacije dostopne na vsakem koraku. Že kar nekaj časa elektronska sporočila prejemamo na mobilne telefone, to pa pomeni, da nam ni potrebno sedeti za osebnim računalnikom, medtem ko prebiramo elektronsko pošto. Mobilni telefon je postal veliko več kot samo naprava za opravljanje klicev in pošiljanje kratkih sporočil.

Osrednja tema diplomske naloge je izdelava sistema, ki je po merah dovolj kompakten, da je, podobno kot mobilni telefon, lahko prenosljiv. Naprava, katero opisujem v diplomski nalogi, je velikosti škatlice za vžigalice, vendar zelo zmogljiva. Njena prednost je predvsem v razširitvah s posameznimi senzorji. Kombinacija napravice in senzorjev nam ponuja neskončno možnosti. Omislimo si lahko majhno napravico, ki nam služi kot e-organizator, pa vse do sistemov, kot so robotski sesalnik, nadzorni sistem, zalivalnik, ki glede na stopnjo vlažnosti zemlje avtomatsko zalije vrt. Pri tem pa lahko kartico z ustrezno razširitvijo opremimo z brezžično komunikacijo ter nam le-ta sporoča vse podatke in opozorila na domači računalnik.

1.2. Uporabljene tehnologije

S tehničnega vidika je diplomska naloga sestavljena iz dveh delov. Prvi del predstavlja aplikacija, drugi del pa strojna oprema. Za izdelavo aplikacije sem uporabila brezplačno programsko okolje Microsoft Express Visual Studio 2008. Strojni del pa predstavlja FEZ Mini kartica, katera podpira Micro Framework.

1.3. .NET Micro Framework

.NET Micro Framework je majhno in učinkovito izvajalno okolje, namenjeno izvajanju upravljane (»managed«) kode na napravah, ki so po svoji velikosti premajhne in omejene za gostitev večjih ogrodij, kot sta Windows CE in Compact .NET framework. Pri razvoju aplikacij lahko uporabljamo orodja, katerih smo že vajeni – Visual Studio 2005, 2008 in 2010. Še več, uporabimo lahko Express Edition Visual Studio, ki je brezplačen.

.NET Micro Framework za svoje izvajanje ne zahteva nameščenega operacijskega sistema. Pomanjšana različica CLR-ja (Common Language Runtime-a), kateremu pravimo tudi TinyCLR, komunicira neposredno s strojno opremo. Za svoj zagon in delovanje zahteva samo nekaj sto kilobajtev rama, zato ne zahteva zmogljivega procesorja, ampak zadostujejo že cenejši, 32 bitni.

TinyCLR skozi zgodovino:

- Začetek sega v leto 2001, ko se začne raziskovanje koncepta pomanjšane različice CLR-ja.
- V letu 2004 se pojavijo »pametne« ure.
- .NET Micro Framework 1.0 je bil leta 2006 nameščen na sumo robote, ki so bili predstavljeni na Mobile and Embedded Developers Conference (MEDC).
- Februarja 2007 je izšla različica 2.0.
- Leta 2007 je Microsoft predstavil Windows SideShow, ki je temeljil na .NET Micro Frameworku.
- Februarja 2008 je izšla različica 2.5.
- Trenutna verzija je 4.1.

Razvoj vgrajenih (embedded) sistemov je v preteklosti za programerja predstavljal velik izziv. Na voljo smo imeli orodja C, C++ ali pa celo zbirni jezik (assembler). Programer je pisal kodo, ki je komunicirala neposredno s strojno opremo. Tu se je pojavil problem – koda, ki je bila napisana za določen procesor, je bila v mnogih primerih neprenosljiva in ni delovala na drugem tipu krmilniku. Na tem področju je .NET Micro Framework našel rešitev. Ponuja nivo abstrakcije do strojne opreme, do katere dostopamo preko njegovih osnovnih razredov. Posamezne strojne komponente so nam dostopne preko objektov. Namesto da se ukvarjamo s podrobnostmi strojne opreme in nastavljamo razne bitne maske za nastavljanje periferije strojne opreme, to naredimo s pomočjo lastnosti objektov. Ta način lahko imenjujemo tudi upravljani (manage) gonilniki, ki nam zagotovijo, da so naše vgrajene (embedded) aplikacije neodvisne od strojne opreme in mikroprocesorjev.

Za izdelavo aplikacij lahko torej uporabimo Visual Studio in Micro Framework, ki tako kot njegov večji brat .NET Framework uporablja nadzorovano oz. managed kodo. Prednosti managed (upravljalne) kode so naslednje :

- Samodejno upravljanje s pomnilnikom, ki namesto nas upravlja smetlar (garbage collector)
- Sinhronizacija niti
- Varnost tipov (type safety)
- Razhroščevanje (debugging)

Poleg vseh teh prednosti pa ima seveda Micro framework tudi slabosti. Ni sistem v realnem času (real-time system). Kljub temu, da je izredno hiter in primeren za večino aplikacij, ne moremo od njega pričakovati delovanja v realnem času. Če obstaja zahteva, da mora naša aplikacija delovati do milisekunde natačno, se moramo odločiti za alternativna orodja. Zakasnitve v delovanju frameworka so lahko nekaj milisekund. Tudi nad upravljalcem smeti nimamo nadzora. Res je, da nam zelo olajša delo, ker nam ni potrebno skrbeti za upravljanje nad spominom, ima pa tudi slabost. Pri določeni zasedenosti pomnilnika se le-ta sam sproži in nam sprosti neuporabne objekte. Nadzora nad tem, kdaj se zažene, nimamo, se pa za ta čas ustavi celotno delovanje programa. To traja nekaj milisekund.

1.4. Micro .NET naprave

Poglavje je namenjeno kratkemu pregledu ponudnikov strojnih naprav, ki že podpirajo razvoj Micro Framework aplikacij. Je pa napredek v razvoju teh naprav zelo hiter, tako da je to le trenutna slika obstoječega stanja.

1.4.1. Freescale I.MXS

Tehnične lastnosti:

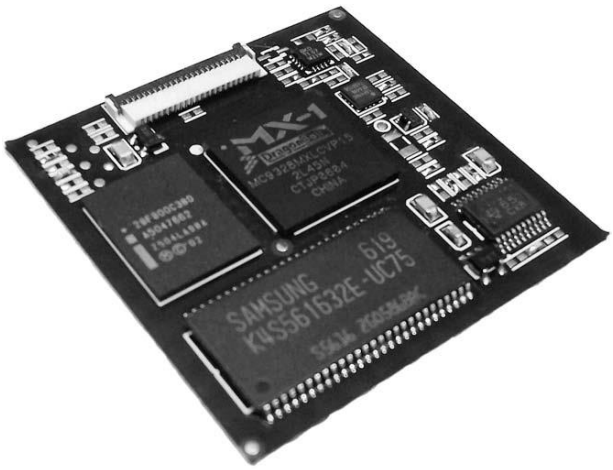
- Freescale i.MXS 100-MHz (ARM920T) procesor
- 32 MB SDRAM
- 8 MB flash spomina
- 2.5-inch QVGA (320 × 240 × 16 bpp) TFT LCD panel
- 10 input/output (GPIO) portev
- RS232 Universal Asynchronous Receiver Transmitter (UART) serijski port
- SPI vodilo
- I2C vodilo
- USB port za razhroščevanje in nameščanje aplikacij
- 5 V vhodnega napajanja
- Možnost napajanja preko USB

Slika 1: Freescale I.MXS

1.4.2. Device Solutions Meridian and Tahoe

Tehnične lastnosti:

- Freescale i.MXS 100-MHz (ARM920T) processor
- 8 MB SDRAM-a
- 4 MB flash pomnilnika
- Priključek za 2.7 inčni QVGA (320 × 240 × 16 bpp) TFT LCD panel
- 16 do 32 GPIO portev
- Dva RS232 UART serijska porta
- SPI vodilo
- I2C vodilo
- Eden PWM kanal
- Eden USB priključek za razhroščevanje in nameščanje aplikacij
- 5 V vhodnega napajanja
- Napajanje preko USB
- 3.3 V izhodne kapacitete za periferijo



Slika 2: Device Solutions

1.4.3. Digi Connect ME

Digi Connect ME je bila prva .NET Micro Framework naprava z Ethernet mrežnim adapterjem in podporo za TCP/IP protokol.

Tehnične lastnosti:

- 55-MHz ARM7 (Digi NS7520) procesor
- 8 MB SDRAM
- 2 MB flash spomina
- Pet GPIO portev
- Eden RS232 UART serijski port
- Integriran 10/100 Mb/s Ethernet adapter
- 3.3 V vhodnega napajanja

Slika 3: Digi Connect ME

1.4.4. Digi ConnectCore 9P 9215

Tehnične lastnosti:

- Digi NS9215 150-MHz procesor
- Do 16 MB SDRAM-a
- Do 16 MB flash spomina
- 10/100 Mb/s Ethernet adapter (ali 802.11b/g WLAN)
- Do 64 skupnih GPIO portev
- Štirje serijski porti (eden RS232/422/485, eden RS232, ter dva TTL porta)
- I2C vodilo
- SPI vodilo
- Dva DRPIC165 x FIM-a
- Osem kanalni 12 bitni analogno digitalni pretvornik

Slika 4: Digi Connect Core

2. Predstavitev problema

Celotni sistem je sestavljen iz elektronskih komponent in aplikacije, ki predstavlja povezavo med temi komponentami. Posamezne komponente so med seboj odvisne. Tako npr. informacijo iz senzorja merjenja refleksivnosti ustrezno prevedemo in se na podlagi pridobljene vrednosti odločamo, ali robota v kratkem pričakuje ovira. Elektronske komponente so dobavljive v prosti prodaji in ne predstavljajo lastnega razvoja. Potrebno jih je le poljubno kombinirati na ploščico. Problem, ki ga moramo rešiti, je izdelava aplikacije.

Arhitekturno je aplikacija sestavljena iz dveh delov. Prvi, nižji nivo, predstavlja neposredno komunikacijo s senzorji in pogonom robota. Drugi, aplikativni nivo, pa uporablja funkcije iz nižjega nivoja. Na tem nivoju v bolj »človeškem« jeziku vodimo robota in se ne obremenjujemo s klici na strojno opremo, saj smo si funkcije, kot so beep, stop, start pripravili v naši knjižnici. Na tem aplikativnem delu sem tudi skalibrirala vrednosti. To pomeni, da sem s testiranjem dobila optimalne mejne vrednosti, tako da se robot ne ustavi prehitro pred oviro, se ji pa izogne.

2.1. Pričakovani rezultati

Od sistema se pričakuje, da bo senzor za zaznavanje refleksivnosti pravočasno obvestil centralni sistem robota. Le-ta bo informacijo v realnem času obdelal in poslal ustrezne ukaze na elektro pogon, ki se bo oviri izognil.

Ker sem se odločila za sistem merjenja refleksije, torej odbojnosti svetlobe s površine predmeta, bodo rezultati najoptimalnejši na svetlejših podlagah. Sistem ne deluje v realnem času, saj nikoli ne vemo, kdaj se sproži smetlar (garbage collector). Za naš sistem, kot tudi za 99 % ostalih sistemov, je takšno delovanje sprejemljivo.

2.2. Izbira razvojnega okolja

.NET Micro Framework je razmeroma nova tehnologija. Je Open Source .NET platforma za naprave z omejenimi viri. Le-te morajo podpirati vsaj 256 Kbytev flash pomnilnika in 64 Kbytov RAM pomnilnika. Vključuje pomanjšano verzijo .NET CLR-ja in podpira razvoj v C# in VB .NET programskih jezikih. Razvoj in razhroščevanje (v emulatorju ali dejansko na strojni napravi) potekata v Microsoft Visual Studiu.

Micro Framework je torej podmnožica .NET knjižnic, ki vsebuje približno 70 razredov, s 420 metodami, GUI framework in dodatne knjižnice, specifične za razvoj embedded sistemov.

Micro framework je trenutno podprt na ARM arhitekturi procesorjev (ARM7 in ARM 9).

Microsoft dopušča razvijalcem brezplačno izdelavo aplikacij s pomočjo .NET Micro Frameworka. SDK je brezplačno prenosljiv z Microsoftove uradne strani, lahko pa ga uporabimo s katero koli različico Visual Studia, tudi z brezplačnimi Express različicami.

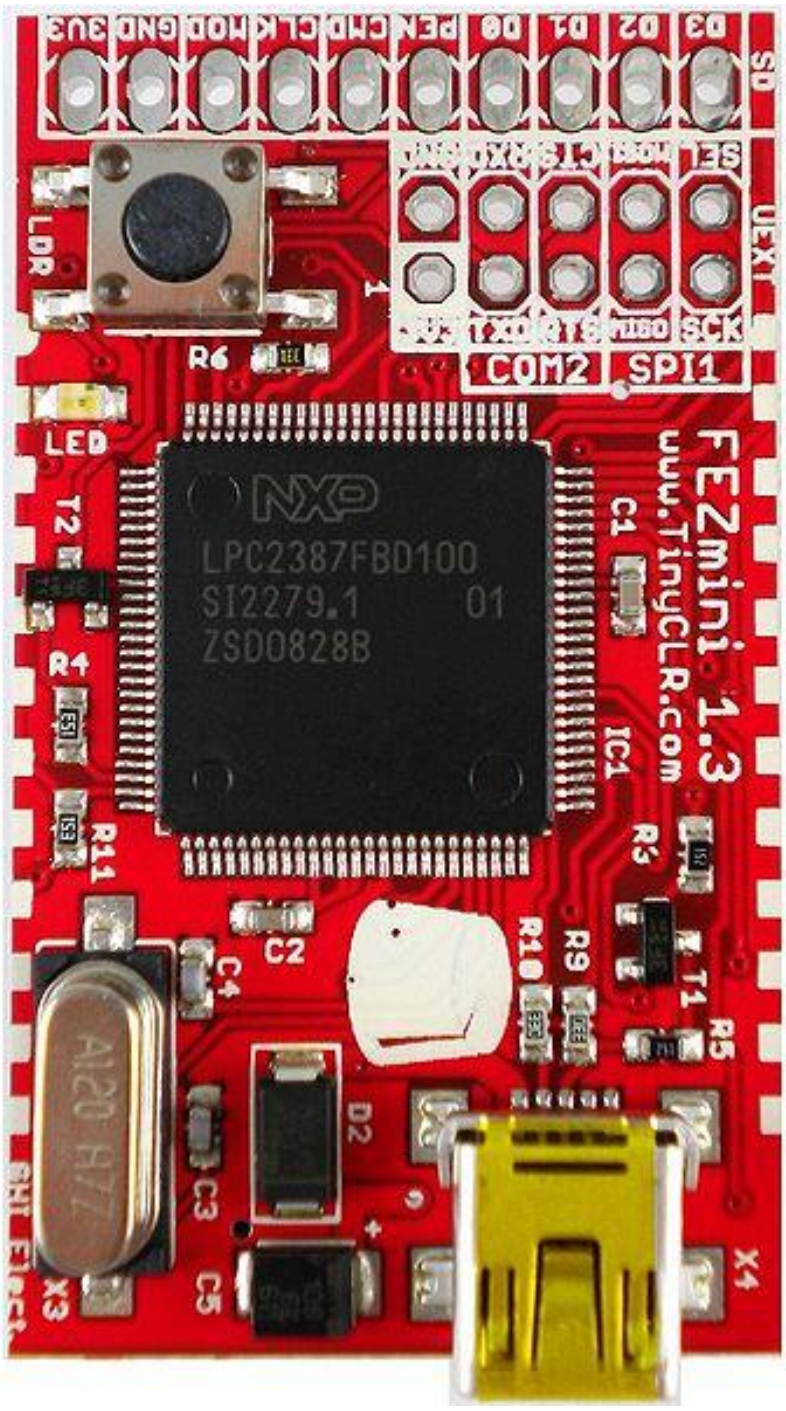
Če še enkrat povzamem, sem se odločila za razvoj v .NET Micro Frameworku zato, ker:

- Je brezplačen in omogoča razvoj v brezplačni različici visual studia (Visual Studio 2008 Express Edition).
- Potrebuje samo 300 KB spomina. Tukaj naj omenim, da naslednji najmanjši .NET Compact Framework, ki teče na Windows CE, potrebuje 12 MB spomina za svoje delovanje.
- Za svoje delovanje ne potrebuje operacijskega sistema, ampak deluje neposredno na strojni opremi.
- Podpira najpogostejše periferne naprave, vključno s Flash spominom, EEPROM, GPIO, I²C, SPI, USB.
- Optimiziran je za energijsko učinkovitost za baterijsko napajane naprave.
- Podpira večnitnost.
- Hardware abstraction layer nam omogoča prenos aplikacij na različne arhitekture.
- Podpira razvoj aplikacij v programskem jeziku C#.

2.3. Izbira strojne opreme

Izbira strojne opreme je bila pogojena s funkcionalnostjo, ki jo ponuja, in ceno. Vse to sem našla v kartici FEZ Mini. Nekaj funkcionalnosti kartice FEZ Mini:

- 72Mhz 32-bit ARM7 LPC2387 procesor
- USBizi – 100
- Približno 148 KB prostega pomnilnika
- Približno 62 KB prostega RAM pomnilnika
- USB klijent za razhroščevanje
- Podpora za MicroSD kartico, z MicroSD razširitvijo
- TCP/IP podpora
- RLP, ki omogoča naložiti C ali Assembler kode za izdelavo aplikacije v realnem času
- 36 izhodov in vhodov
- 3 x UART TTL, 1 x RS232
- 8 analognih vhodov, 1 analog Izhod
- 2 x SPI
- 1 x I2C
- 1 x CAN



Slika 5: FEZ Mini

3. Zmožnosti .NET Micro Frameworka

3.1. Uvod v .NET Micro Framework

Kot sem že omenila, .NET Micro Framework za svoje izvajanje ne potrebuje operacijskega sistema. Izvaja se neposredno na strojni opremi. Sestavljajo ga štirje nivoji:

a) Nivo uporabnikove kode. Je najvišji nivo, sestavlja ga nadzorovana »managed« koda, ki jo izdelujemo mi, kot uporabniki Micro Frameworka.
b) Nivo osnovnega razreda knjižnic. Je podmnožica razredov, ki se nahajajo v .NET Frameworku (delo z grafiko, mrežo ...).
c) TinyCLR – je nivo, ki je sestavljen iz naslednjih podnivojev:
   - Hardware Abstraction Layer (HAL), nivo, ki je tesno vezan na strojno opremo. Zagotavlja funkcije in metode, ki dostopajo do strojne opreme.
   - Platform Abstraction Layer (PAL) je dodatni nivo abstrakcije nad HAL nivojem.
   - .NET Micro Framework CLR. Sestavlja ga sistem za upravljanje kode ter zagotavlja varnost tipov. Med drugim vsebuje tudi smetlarja (garbage collector), ki je odgovoren za samodejno sproščanje objektov.
d) Nivo strojne opeme. Sestavlja ga fizična strojna oprema. To je lahko končna naprava (mikroprocesor) ali pa emulator, ki se izvaja na namiznem računalniku.

.NET Micro framework loči vso kodo, ki jo izdelamo mi, kot uporabniki Micro Frameworka, z dejanskimi klici na strojno opremo. To naredi s pomočjo Hardware in Platform abstrakcijska nivoja. Teoretično bi torej prenos kode iz ene strojne naprave na drugo moral biti enostavno opravilo, a je zelo kompleksno in zahteva poglobljeno znanje o strojni opremi. Je pa v praksi tako, da prenos kode omogočijo proizvajalci strojne opreme, ne pa uporabniki .NET Micro Framework knjižnic. Trenutno obstaja že kar nekaj platform, ki omogočajo povezavo strojne opreme z .NET Micro Frameworkom. Razlika med njimi je le v velikostih zaslonov ter mrežnih podporah.

3.2. Razvojno orodje za izdelavo Micro Framework aplikacij

Pri izdelavi Micro Framework aplikacij uporabljamo Visual Studio. Kot vsako drugo orodje za izdelavo aplikacij je tudi Visual Studio sestavljen iz urejevalnika kode, ki ima funkcionalnosti dokončanja kode (code completion), podčrtovanja sintakse (syntax highliting) itd. V veliko pomoč nam je v odkrivanju hroščev, saj nudi vse potrebno za hitro in učinkovito odkrivanje in odpravljanje napak. Razvoj Micro Framework aplikacij lahko poteka v vseh različicah Visual Studi-a. Jaz bom izbrala brezplačno različico Visual Studia (Express Edition), saj mi ponuja vse potrebno za razvoj tinyCLR aplikacij.

Brezplačna različica Visual Studia ima seveda omejitve glede na profesionalno različico, a je razvoj aplikacij za Micro okolje zaradi tega nemoten. Primerjalna tabela glede različic visual studia:

Product	Extensions	Setup Projects	MSDN Integration	Debugging	Profiling	Static Analysis	IntelliTrace	Unit Test	Code Coverage	Coded UI Test	Test Impact Analysis	Load Testing	Lab Management	Architecture and Modelling	Mobile Development
Express	No	Limited	Essentials	Yes	No	No	No	No	No	No	No	No	No	No	Windows Phone 7 only
Professional	Yes	Yes	Essentials or Full	Yes	No	No	No	Yes	No	No	No	No	No	No	Windows Phone 7 only
Premium	Yes	Yes	Full	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	Read-only	Windows Phone 7 only
Ultimate	Yes	Yes	Full	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Windows Phone 7 only
Test Professional	No	No	Full	No	No	No	No	No	No	No	Yes	No	Yes	No	No

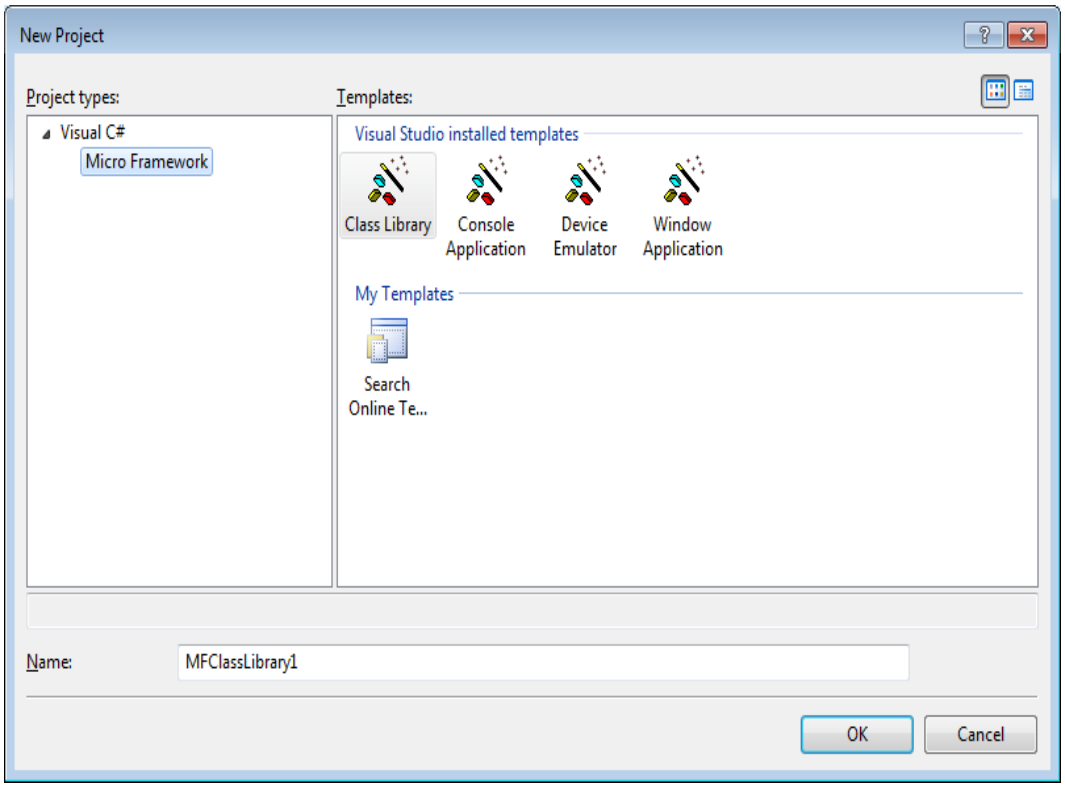
Slika 6: Primerjava posameznih različic Visual Studia

3.3. Priprava okolja za izdelavo MF aplikacij

Microsoft Visual Studio je naslednik Visual Studia 6. Njegova naloga je, da nam poenostavi razvoj .NET aplikacij (VC#.Net, VB.Net, VC++.Net, Jscript.Net, J#.Net, Asp.Net ...). Visual Studio IDE je enoten za vse omenjene in še več jezikov. Vsi .Net jeziki tako uporabljajo skupni IDE, razhroščevalnik, okna za orodja, pogled na razrede, projekte ...

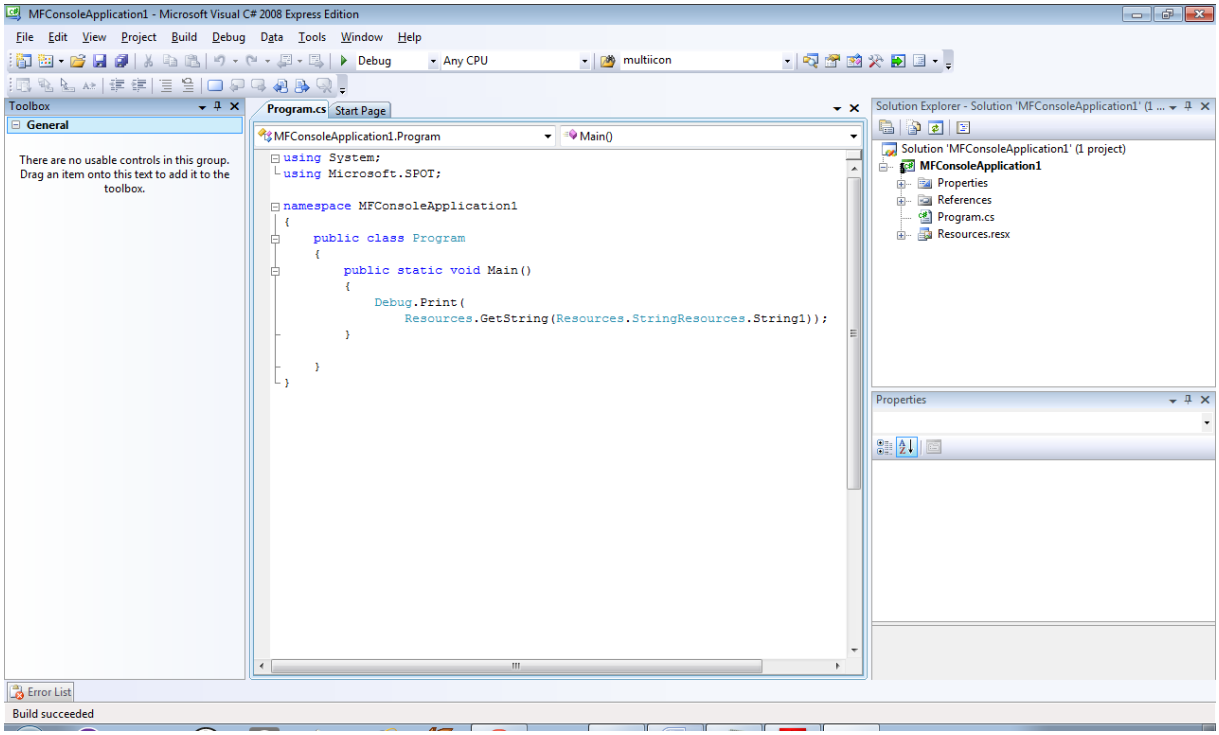
Projekt je skupek zagonskih in knjižnih datotek, ki skupaj tvorijo aplikacijo oz. modul. Informacija o projektu je shranjena v datoteki s končnico ».csproj«, kjer »cs« predstavlja C#. Podobno so informacije o Vb.Net projektu shranjene v datotekah s končnico ».vbproj«. Vzorci oz. »template-i«, ki so vgrajeni v razvojno okolje, nam omogočajo izdelavo različnih tipov projektov – konzolne aplikacije, namizne aplikacije, ASP.Net Web aplikacije, razredne knjižnice. Obstajajo template-i, ki se inštalirajo naknadno, s tem pa dobimo v Visual Studio možnost izdelave novega tipa projekta. Kot primer naj omenim, da smo za potrebe diplomske naloge namestili nov vzorec izdelave aplikacij, ki nam je omogočil razvoj Micro Framework aplikacij.

Razvoj aplikacije je potekal v orodju »Visual Studio 2008 Express Edition«. Orodje je na voljo brezplačno na Microsoftovi uradni strani. Potrebno je namestiti Micro Framework SDK, ki je prav tako na voljo brezplačno.



Slika 7: Izbira ustreznega vzorca

Po izbiri ustreznega vzorca se nam odpre glavno okno razvijalskega orodja. Kot je razvidno iz slike, nam Visual Studio v orodni vrstici ponuja podobne možnosti, kot jih imajo tudi druga vizualna programerska orodja. To so možnosti za dodajanje novega projekta, zagon projekta v razhroščevalnem načinu, prevajanje kode v strojni zapis ... Desna stran je namenjena pregledu posameznih projektov, ki so vključeni v celotno rešitev. Pri posameznih projektih imamo v obliki drevesne strukture pregled nad referencami, ki so potrebne za delovanje projekta in posameznimi razredi (*.cs datoteke).



Slika 8: Razvijalsko orodje za izdelavo .NET Micro Framework aplikacij

3.4. Dostop do strojne opreme

3.4.1. GPIO porti

»General Purposeinput/output« ali skrajšano GPIO porti so verjetno najpogostejši način interakcije mikrokontrolerja z okoljem. GPIO port je lahko inicializiran kot vhodni ali pa izhodni. Tipičen primer izhodnega porta je upravljanje LED diod.

GPIO port lahko ima dva statusa:

- Nizki (0 V)
- Visoki (pozitivna voltaža, najpogosteje 3.3 V)

V .NET Micro Frameworku je status GPIO porta, predstavljen z vrednostjo tipa Boolean. Vrednost »false« predstavlja nizki status, »true« pa visoki.

Izhodni porti

V Microsoft.SPOT.Hardware imenskem prostoru se nahaja razred »OutputPort«, kateri predstavlja GPIO izhodni port.

Spodnja koda predstavlja, kako inicializiramo prvo linijo našega CPU-ja kot GPIO izhodni port ter kako ga periodično prekinemo (na vsaki 2 s).

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;

namespace GpioOutputPortSample
{
    public class Program
    {
        public static void Main()
        {
            OutputPort outputPort = new OutputPort(Cpu.Pin.GPIO_Pin0, true);
            while (true)
            {
                Thread.Sleep(500);
                outputPort.Write(!outputPort.Read()); //toggle port
            }
        }
    }
}
```

Konstruktor razreda »OutputPort« izgleda takole:

```
public OutputPort(Cpu.Pin portId, bool initialState);
```

Pričakuje ID porta in inicialno stanje, kjer predstavlja »true« visoko vrednost, »false« pa nizko.

»portId« ni vrednost tipa »integer«, ampak enumerator tipa »Cpu.Pin«, kjer ima »GPIO_NONE« vrednost -1, obstaja pa 15 predefiniranih pinov z imeni GPIO_PIN0 do GPIO_PIN15.

Vhodni porti

Do vhodnih portov v Micro .Net Frameworku prav tako dostopamo preko Microsoft.SPOT.Hardware imenskega prostora.

Spodnja koda prikazuje, kako nastavimo CPU linijo mikrokontrolerja kot GPIO vhodni port in v intervalih prebiramo njeno stanje.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace GpioInputPortSample
{
    public class Program
    {
        public static void Main()
        {
            InputPort inputPort = new InputPort(Cpu.Pin.GPIO_Pin2,
                                                false,
                                                Port.ResistorMode.PullDown);
            while (true)
            {
                bool state = inputPort.Read(); //polling of port state
                Debug.Print("GPIO input port at pin " + inputPort.Id +
                            " is " + (state ? "high" : "low"));

                //enable device to sleep or emulator to react to Visual Studio
                Thread.Sleep(10);
            }
        }
    }
}
```

»InputPort« razred pričakuje naslednji konstruktor:

```
public InputPort(Cpu.Pin portId, bool glitchFilter, Port.ResistorMode resistor);
```

Pri ustvarjanju novega »InputPort« objekta podamo kot prvi parameter strojni pin porta (portId).

Drugi parameter, »glitchFilter«, označuje, ali je v vhodnem signalu prisoten kakšen šum, ki ga je potrebno prezreti. Kot primer naj podamo šum mehaničnih gumbov ali stikal. Enotni pritisk gumba lahko generira več impulzev. Filter napak (»glitch filter«) odstrani takojšnje zaporedne spremembe med visokim in nizkim stanjem. Intenzivnost filtra napak lahko definiramo s statično lastnostjo »GlitchFilterTime«. Naslednji primer prikazuje nastavitev filtra napak na 5 ms:

```
Cpu.GlitchFilterTime = new TimeSpan(0, 0, 0, 0, 5);
```

Prekinitveni porti

Predstavljajmo si aplikacijo, ki čaka na uporabnikov vnos znaka s tipkovnice. Branje GPIO portov v neskončni zanki je zelo neučinkovito, saj obremenjuje procesor. V micro .Net frameworku obstaja razred, ki nam reši to težavo. Imenjuje se InterrupPort, deduje pa iz razreda InputPort. Prekinitve so dogodki strojnih naprav. V primeru, da mikroprocesor nima trenutno nobene naloge, razen čakanja določenega GPIO dogodka, gre lahko v stanje varčevanja energije. Takoj, ko se zgodi sprememba signala na vhodnem pinu, se mikrokontroler prebudi in določena metoda »Interrup service routine« ali ISR je zagnana.

Konstruktor izgleda tako:

```
public InterruptPort(Cpu.Pin portId, bool glitchFilter, Port.ResistorMode resistor,
        Port.InterruptMode interrupt);
```

Z enumeratorjem InterrupMode določimo, kdaj sprožimo prekinitev. Možne vrednosti so:

- InterrupNone (Port je deaktiviran)
- InterrupEdgeLow (Sprememba iz visoke v nizko vrednost)
- InterrupEdgeHigh (Sprememba iz nizke v visoko vrednost)
- InterrupEdgeBoth (Katera koli sprememba stanja)
- InterrupEdgeLevelHigh (Visoka)
- InterrupEdgeLevelLow (Nizka)

3.4.2. RS232 Serijski port

Serijski port je v uporabi že kar nekaj časa, njegovo zamenjavo predstavlja USB. Vseeno pa še veliko današnjih naprav uporablja seriski port (gps sprejemniki, merilni inštrumenti ...), zato je podprt tudi v Micro Frameworku.

Serijski port se nahaja v Microsoft.SPOT.Hardware imenskem prostoru. Razred ni enak System.IO.Ports.SerialPort razredu iz .NET Frameworka oz. .NET Compact Frameworka, ampak je narejen prav za Micro Framework.

Konstruktor SerialPort razreda kot parameter pričakuje instanco SerialPort.Configuration razreda. S Configuration razredom specificiramo:

- COM Port številko
- Hitrost prenosa podatkov
- Nadzor pretoka

3.4.3. I2C vodilo

I2C uporabljamo za komunikacijo med mikrokontrolerjem in ostalimi strojnimi napravami. Je serijsko vodilo, sistem pa je razvil Philips Semiconductors v začetku leta 1980.

Posebnost I2C je v tem, da potrebuje le dve žici. Ne glede na to, da je veliko počasnejši kot podobni sistemi, ga uporabljamo zaradi nizkih stroškov. Po prvi žici, ki se imenuje serijski podatki (SDA – serial data), se serijsko pretakajo podatki. Druga žica, ki jo imenujemo serijska ura (SCL – serial clock), skrbi za časovno usklajevanje.

Vsak I2C modul je lahko izbran s pomočjo 7 bitnega naslova. Osmi bit je R/W (read / write) bit in nam pove, ali hoče mikrokontroler pisati ali pa brati iz modula.

Torej, v primeru, da uporabljamo enega od I2C senzorjev, uporabimo I2Cdevice, ki se nahaja v Microsoft.SPOT.Hardware imenskem prostoru.

3.5. Mreža

Poglavje je namenjeno predstavitvi tega, kako .NET micro framework komunicira z drugimi napravami preko ethernet omrežja. Omogoča nam izdelavo internet, LAN in WLAN aplikacij z uporabo enega nabora razredov.

3.5.1. Vtiči

Vtič je končna točka povezave. Z njihovo pomočjo lahko preko mreže na obeh straneh, client in server, beremo in pošiljamo podatke. Da bi lahko pošiljali ali brali podatke preko vtičev, moramo poznati IP naslov komunikacijskega partnerja na drugi končni točki.

V imenskem prostoru System.NET.Sockets se nahaja razred Socket, ki nam omogoča medmrežno komunikacijo. Je okrnjena različica razreda .NET Frameworka, ima pa vseeno precejšnje število metod in funkcij za izdelavo mrežnih aplikacij. Ovržene so metode za asinhrono pošiljanje podatkov po mreži.

Konstruktor Socket razreda sprejme tri parametre:

- addressFamily tip, ki opisuje shemo za razrešitev naslova. Vrednost za Internet in LAN aplikacije je InterNetwork.
- socketType označuje tip vtiča. Možni vrednosti sta Stream, ki je zanesljiva dvosmerna komunikacija, ter Dgram za nepovezane prenose podatkov, ki so nezanesljivi, vendar hitri.
- protocolType predstavlja protokol, ki ga vtič uporablja za komunikacijo. Možni vrednosti sta TCP in UDP.

TCP/IP je pravilna odločitev, kadar potrebujemo zanesljivo dvosmerno povezavo, ki zagotavlja, da so paketi prejeti v pravilnem zaporedju, ter da med prenosom ni izgube ali podvojenih podatkov.

TCP/IP vtič naredimo na naslednji način:

```
Socket socket = new
Socket(Addressfamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
```

UDP je nepovezan protokol, ki je hitrejši kot TCP/IP, so pa možne napake, ker se lahko podatki med prenosom izgubijo, ali pa prispejo na cilj več kot enkrat. UDP se uporablja za pretočne storitve, kot so Voice over IP, kjer je pomembno delovanje v realnem času, vendar se napake poznajo samo v kakovosti zvoka.

Primer konstruktorja vtiča za UDP protokol:

```
Socket serverSocket = new
Socket(AddressFamily.InterNetwork,SocketType.Dgram,ProtocolType.Udp)
```

3.5.2. Web Services

Device Profile for Web Services je bil objavljen maja 2004. Implementira podmnožico Web Services specifikacije, dodan pa je mehanizem, s pomočjo katerega lahko klijenti zlahka odkrijejo naprave. Ko je naprava odkrita, klijent pridobi opis storitve, ki gostuje na tej napravi, ter te storitve tudi uporabi.

Naprave so enote, ki se nahajajo na omrežju in lahko vključujejo nič ali več storitev. Primeri naprav so:

- Tiskalniki
- Skenerji
- Projektorji
- Video sistemi

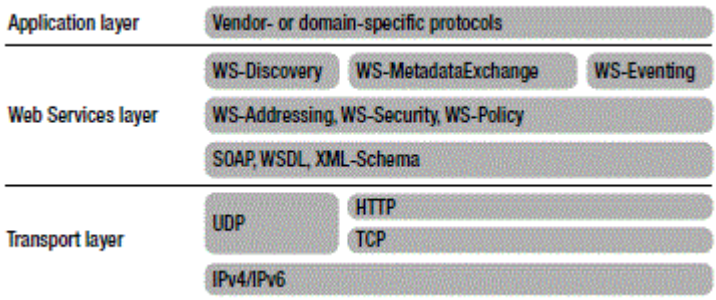
DPWS nam tako lahko omogoča predstavitev na projektorju z uporabo brezžičnega omrežja in brez uporabe kablov.

Tu naj omenim nekaj primerov uporabe DPWS-a v domačem gospodinjstvu:

- Regulacija temperature v zmrzovalniku preko daljinskega upravljavca, mobitela ali pc-ja.
- Pralni stroj, ki bi nas obveščal o svoji trenutni akciji na mobitel.
- Seveda obstaja veliko scenarijev, ki si jih lahko izmislimo, prav ta tehnologija pa ponuja veliko možnosti na tem področju.

DPWS definira standard, tako da se lahko med seboj povežejo naprave različnih ponudnikov.

Arhitekturo DPWS-ja prikazuje spodnja slika:



Slika 9: DPWS arhitektura

3.6. Brezžična komunikacija

Če mikrokontroler potrebuje brezžično komunikacijo z drugimi napravami, kot so osebni računalniki, dlančniki, drugi mikrokontrolorji ali senzorji, imamo na voljo štiri tehnologije. Te so:

- Brezžični LAN
- Bluetooth
- Z-WAVE
- ZigBee

3.6.1. Brezžični LAN

Brezžično lokalno omrežje je brezžična različica Ethernet LAN tehnologije. Tehnologija se imenuje tudi Wi-Fi. Doseg ima do 100 metrov na prostem, z uporabo zunanjih ojačevalnih anten pa doseže tudi do 300 metrov.

Za uporabo WLAN-a na Micro .NET napravi mora le-ta imeti integriran WLAN.

3.6.2. Bluetooth

Bluetooth tehnologija izhaja iz leta 1994. Bluetooth je integriran že v večino prenosnikov, mobilnih telefonov in dlančnikov.

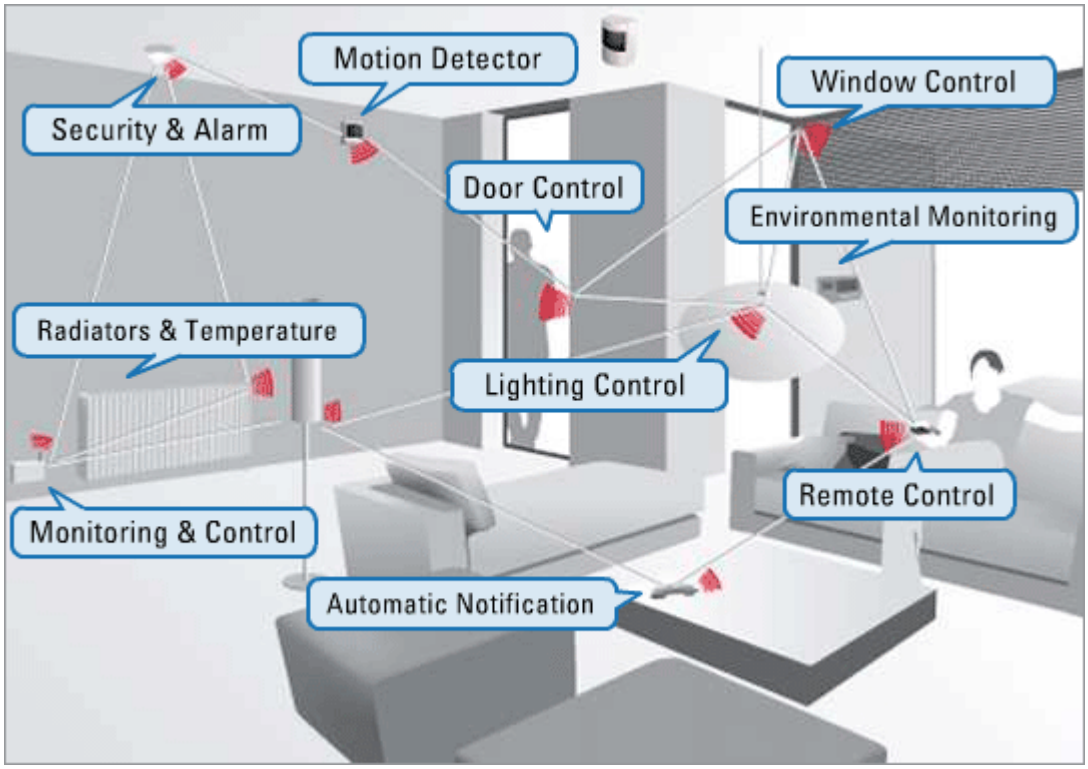
Standard določa, kako se naprave med seboj zaznajo in vzpostavijo povezavo in kako je komunikacija med njimi zavarovana.

.NET Micro Framework ne podpira neposredno Bluetootha. Na tržišču pa obstajajo Bluetooth-Serial adapterji, ki so na voljo kot OEM adapterji. Na takšen način lahko izmenjujemo podatke med dvema napravama zelo enostavno, kot če bi bili povezani s serijskim kablom.

3.6.3. ZigBee

ZigBee je nova tehnologija. Osredotoča se predvsem na povezovanje in nadzor domače opreme. Z ZigBee lahko dostopamo in nadzorujemo gospodinjske aparate, senzorje in podobne naprave na kratkih razdaljah.

.NET Micro Framework ne podpira neposredno tehnologije ZigBee. Vendar pa na tržišču že obstajajo OEM moduli, ki predstavljajo vmesnik za serijski port ali SPI vodilo.

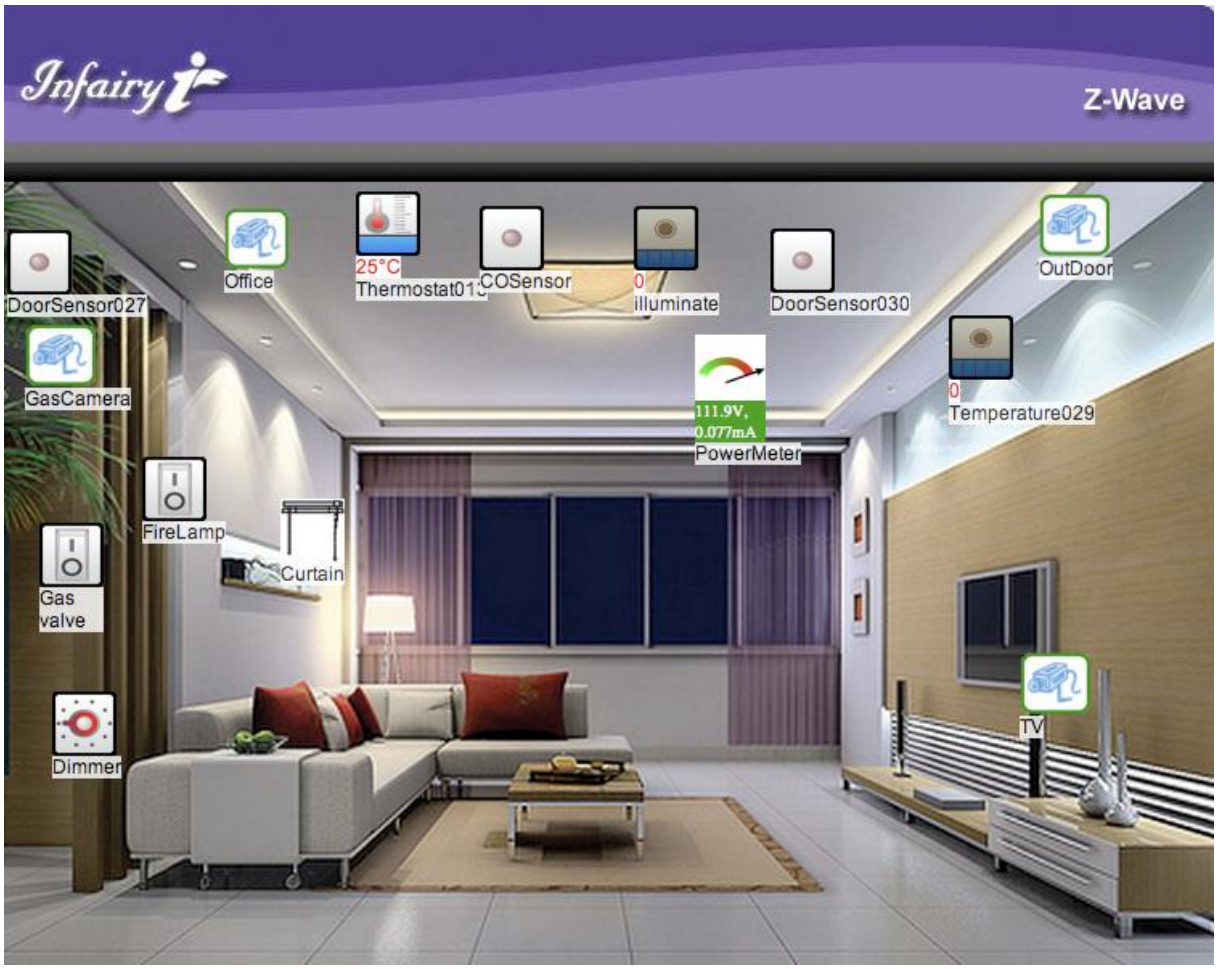


Slika 10: Primer ZigBee omrežja

3.6.4. Z-WAVE

V primerjavi s konkurenčno tehnologijo ZigBee je Z-WAVE namenjen za prenose z nizko hitrostjo podatkov, nizkimi stroški in nizko porabo energije.

Z-WAVE ni neposredno podprt v .NET Micro Frameworku, vendar obstajajo OEM moduli, ki predstavljajo vmesnik za serijski port. Podjetje ControlThink je razvilo knjižnico, imenovano ThinkRemote za dostop do Z-WAVE-a iz .NET Micro Frameworka.



Slika 11: Primer Z-Wave omrežja

Primerjavo opisanih tenologij prikazuje spodnja tabela:

Feature	WLAN	Bluetooth	ZigBee	Z-Wave
Maximum bandwidth	11 megabits/second (Mb/s)	1 / 2.1 Mb/s	0.2 Mb/s	0.1 Mb/s
Power consumption (battery life)	High (1–3 hours)	Medium (4–8 hours)	Very low (2–3 years)	Ultra low (several years)
Maximum distance	100/300 m	10/20/100 m (Class 3/2/1)	20–100 m	10–75 m
Hardware costs	High	Medium	Low	Very low
Footprint of protocol stack	>100 kilobytes (KB)	approximately 100 KB	approximately 32 KB	32 KB
Industry standard	IEEE 802.11 a, b, and g	IEEE 802.15.2	IEEE 802.15.4	Zensys proprietary technology
Frequency	2.4 GHz	2.4 GHz	2.4 GHz	900 MHz
Maximum number of network nodes	Nearly unlimited	7 (generally, 2 are used)	250	232
Spreading	High (PC, Notebook, PDA)	High (notebooks, PCs, PDAs, smartphones, and mobile phones)	Low (sensors and actors)	High (sensors and actors)
Example applications	PC networking, wireless Internet, and video streaming	Replacing cables, serving as a kind of wireless USB, and driving head sets, wireless printing, and file transfers	Home automation, industrial control, remote controls, sensors, switches, and smoke detectors	Home automation

Slika 12: Primerjava WLAN, Bluetooth, ZigBee in Z-Wave tehnologij

3.7. Kriptografija

.NET Micro Framework ponuja dva algoritma za šifriranje podatkov:

- Simetrični šifrirni algoritem XTEA, ki za kodiranje in dekodiranje uporablja isti ključ
- RSA asimetrična metoda, ki deluje z javnimi in zasebnimi ključi

Podpisane podatke lahko preverimo s pomočjo .NET Micro Frameworka. Na žalost pa metod za podpis podatkov v .NET Micro Frameworu še ni.

Šifriranje podatkov je potrebno, če se le-ti izmenjujejo z drugimi napravami, kot so osebni računalniki, pametni telefoni ali druge .NET Micro Framework naprave preko omrežja.

3.7.1. XTEA algoritem

Razvit je bil leta 1997. Kot že ime XTEA pove, je izvedenka od TEA, ki je popravila nekaj šibkih točk TEA algoritma. XTEA šifrirna metoda ima velikosti blokov 8 bytov in dolžino ključa 16 bytov. Algoritem uporablja simetrični ključ, kar pomeni, da se podatki šifrirajo in dešifrirajo z istim ključem.

XTEA sicer ne obstaja v .NET Frameworku, obstaja pa v .NET Micro Frameworku, v imenskem prostoru Microsoft.SPOT.Cryptography, kot razred Key_TinyEncryptionAlgorithm.

Primer .NET Micro Framework kode za šifriranje in dešifriranje s pomočjo XTEA algoritma:

```
byte[] key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 };

byte[] iv = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };

Key_TinyEncryptionAlgorithm xtea = new Key_TinyEncryptionAlgorithm(key);

string plainText = "Hello World!";

 byte[] plainBytes = System.Text.Encoding.UTF8.GetBytes(plainText);

//Encryption

byte[] cipherBytes = xtea.Encrypt(plainBytes, 0, plainBytes.Length, iv);

//Decryption

byte[] restoredBytes = xtea.Decrypt(cipherBytes, 0, cipherBytes.Length, iv);
```

3.7.2. RSA Asimetrični algoritem

Poleg XTEA algoritma je za .NET Micro Framework na voljo tudi RSA šifriranje. RSA je sposoben šifrirati in digitalno podpisovati podatke.

Uporablja princip javnih in zasebnih ključev. Zasebni ključ se uporablja za dešifriranje in podpisovanje podatkov. Z javnim ključem se šifrirajo podatki in preverjajo podpisi.

RSA je zelo počasen. Za šifriranje večje količine podatkov je priporočljivo, da se uporablja v povezavi z XTEA algoritmom.

Spodnja koda prikazuje primer RSA enkripcije. Zaradi večje preglednosti so nekateri deli opuščeni:

```
byte[] modulus = new byte[] { ... };

byte[] privateExponent = new byte[] { ... };

byte[] publicExponent = new byte[128];
```

```
publicExponent[0] = 0x01;
publicExponent[2] = 0x01;
string plainText = "Hello World!";
byte[] plainBytes = Encoding.UTF8.GetBytes(plainText);
//Encryption
Key_RSA encryptor = new Key_RSA(modulus, publicExponent);
byte[] cipherBytes = encryptor.Encrypt(plainBytes, 0, plainBytes.Length, null);
//Decryption
Key_RSA decryptor = new Key_RSA(modulus, privateExponent);
byte[] restoredBytes = decryptor.Decrypt(cipherBytes, 0, cipherBytes.Length, null);
```

4. Izdelava aplikacije

4.1. Predstavitev razvojnega jezika

Programski jezik, ki sem ga uporabila za izdelavo aplikativnega dela diplomske naloge, je C#. C# je zasnovan kot preprost, moderen, splošno namenski objektno usmerjen programski jezik. Trenutna različica je 4.0, ki je izšla 12. aprila 2010.

Ime C# je dobilo navdih po notnem zapisu, kjer znak # pomeni, da je napisana nota za pol tona višja. Podoben pomen ima tudi C++, kjer »++« pomeni povečanje spremenljivke za 1.

Spodnja tabela prikazuje razvoj C# programskega jezika ter njegovo povezavo z razvojnim orodjem Visual Studio:

Version	Language specification			Date	.NET Framework	Visual Studio
	ECMA	ISO/IEC	Microsoft			
C# 1.0	December 2002	April 2003	January 2002	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.2			October 2003	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	June 2006	September 2006	September 2005[note 1]	November 2005	.NET Framework 2.0	Visual Studio 2005
C# 3.0	None[note 2]		August 2007	November 2007	.NET Framework 2.0 (Except LINQ/Query Extensions) [1] .NET Framework 3.0 (Except LINQ/Query Extensions) [2] .NET Framework 3.5	Visual Studio 2008 Visual Studio 2010
C# 4.0			April 2010	April 2010	.NET Framework 4	Visual Studio 2010

Slika 13: Povezava C# jezika in Visual Studia

Kratek pregled dodanih funkcionalnosti skozi razvoj jezika prikazuje naslednja tabela:

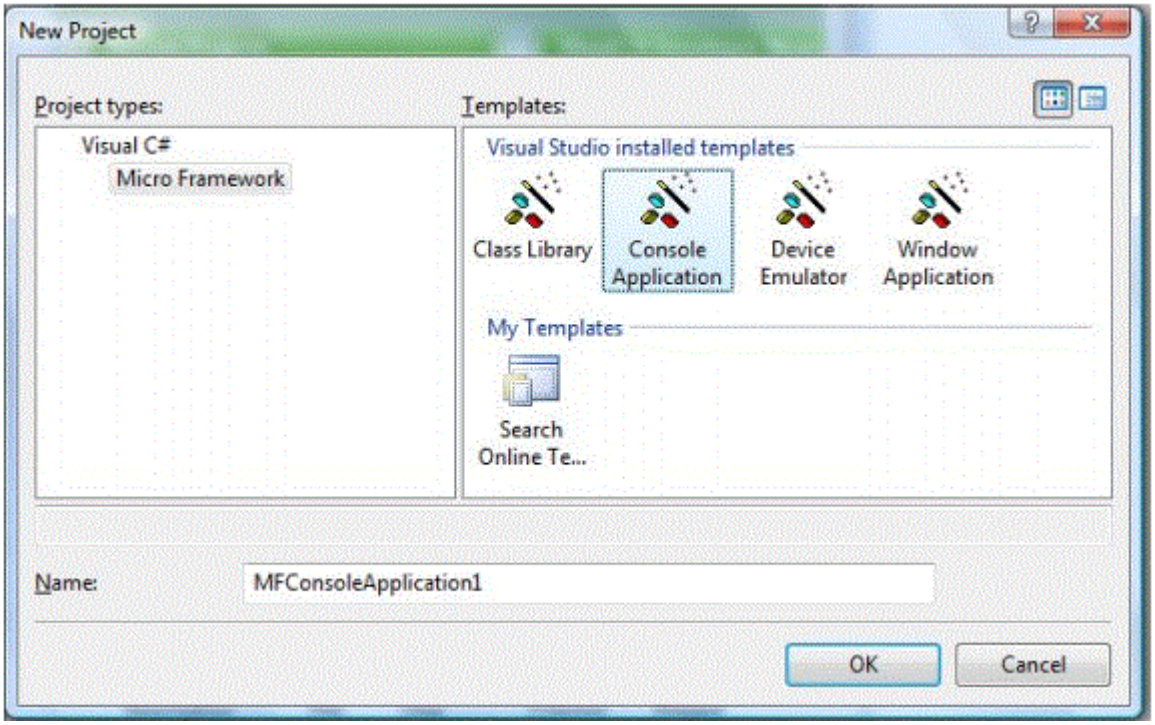
	C# 2.0	C# 3.0	C# 4.0	C# 5.0 (planned)[citation needed]
Features added	• Generics • Partial types • Anonymous methods • Iterators • Nullable types • Private setters (properties) • Method group conversions (delegates)	• Implicitly typed local variables • Object and collection initializers • Auto-Implemented properties • Anonymous types • Extension methods • Query expressions • Lambda expressions • Expression trees	• Dynamic binding • Named and optional arguments • Generic co- and contravariance	• Asynchronous methods • Compiler as a service

Slika 14: Razvoj C# jezika

4.2. Emulator strojne opreme

NETMF vsebuje emulator, ki nam omogoča, da se aplikacija izvaja na našem razvojnem računalniku.

Za izdelavo novega projekta v Micro .NET Frameworku odpremo visual studio in izberemo možnost File -> New Project. Po uspešni inštalaciji micro frameworka na razvojni računalnik se nam med predlogi prikaže opcija »Micro Framework«. Znotraj predloge se nahaja več tipov aplikacij. Za izdelavo diplomske naloge sem izbrala tip »Console Application«.



Slika 15: Micro Framework vzorec

Po izbiri ustrezne predloge nam Visual Studio zgradi projekt, ki ima samo eno datoteko tipa cs, to pa je Program.cs. Datoteka predstavlja vstopno točko v aplikacijo, vsebuje pa že nekaj vrstic kode, ki jo za nas zgenerira sam Visual Studio.

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
   public class Program
   {
      public static void Main()
      {
         Debug.Print(
            Resources.GetString(Resources.StringResources.String1));
      }

   }
}
```

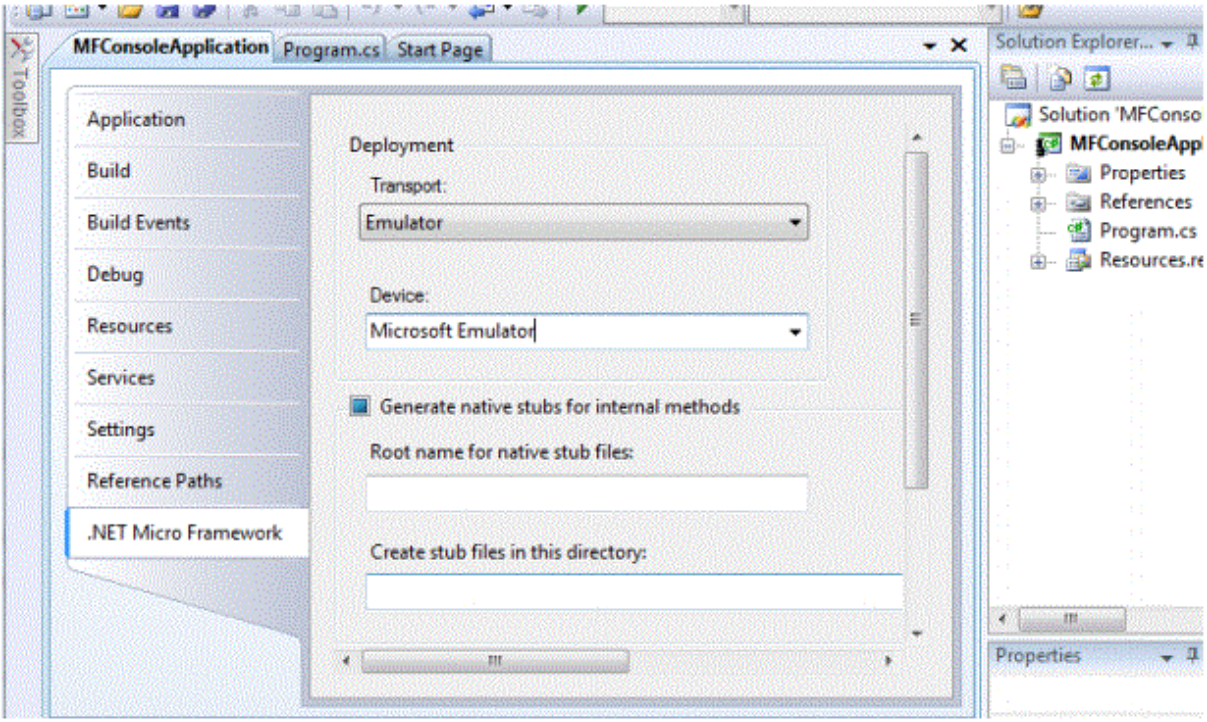
Za zagon aplikacije še ne potrebujemo .NET Micro Framework naprave, saj emulator poskrbi za simulacijo različne strojne opreme na našem računalniku. Takšen pristop nam ponuja hiter razvoj prototipa aplikacije in nam omogoča začetek razvoja, še preden imamo na voljo celotno strojno opremo.

Emulator ima vgrajeno podporo za komponente strojne opreme, ki so dostopne .NET Micro Frameworku. Tako lahko simuliramo komponente, kot so:

- GPIO porti
- SPI naprave
- I2C naprave
- Serijski porti
- LCD
- RAM, Flash spomin
- Baterije

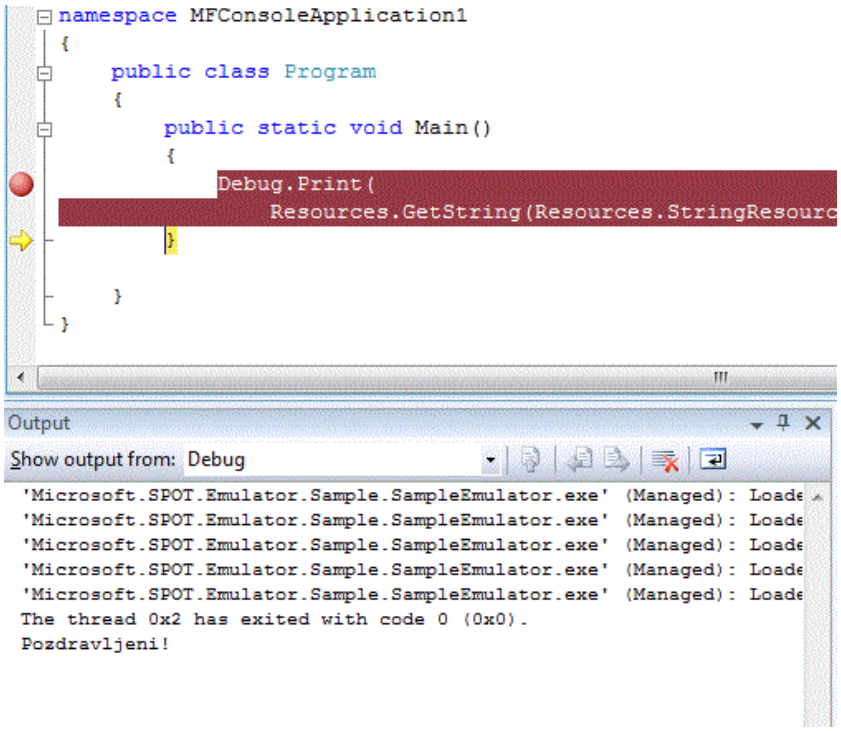
Emulator je računalniška aplikacija in ni nujno, da je njegov izgled identičen napravi, ki bo dejansko gostila aplikacijo.

Spodnja slika prikazuje pravilne nastavitve za zagon naše aplikacije v emulator načinu:



Slika 16: Zagona aplikacije v emulator načinu

Po zagonu aplikacije se začne proces razhroščevanja. Na voljo nam je vpogled v spremenljivke, kar nam zelo olajša iskanje napak in celoten razvoj aplikacije. V veliko pomoč nam je »Output Window«, v katerem se prikazujejo razna sporočila iz naše aplikacije.



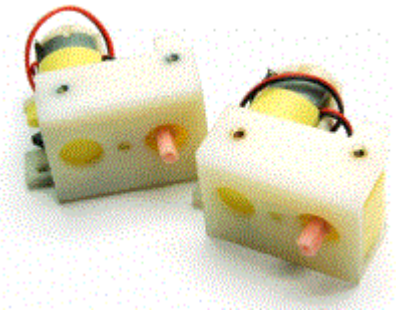
Slika 17: Razhroščevanje aplikacije

4.3. Mehanski deli robota

4.3.1. Elektromotorji

Za izdelavo diplomske naloge sem uporabila dva DC motorja, model BO-2 z IDC konektorjema. Lastnosti uporabljenih elektromotorjev so:

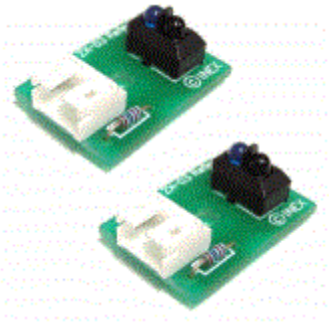
- Delovna napetost od +3 do +9 V dc
- Poraba toka 130 mA
- Povprečna hitrost od 170 do 250 vrtljajev na minuto
- Teža 30 g
- Dimenzije 42 X 45 X 22.7 mm



Slika 18: Elektromotorji

4.3.2. Refleksivni senzorji

Refleksivni senzor oddaja nevidno infrardečo svetlobo, potem pa jo skuša izmeriti nazaj, da ugotovi, če se svetloba odraža. Senzor sem v diplomski nalogi uporabila za odkrivanje robov mize. Senzor deluje tako, da spremeni svoj izhod iz visoke v nizko napetost, če ugotovi odsev.



Slika 19: Senzorji refleksije

4.4. Implementacija

Aplikativni del diplomske naloge je sestavljen iz dveh nivojev. Prvi, nižji nivo, sestavljajo gonilniki za elektromotorje, senzorje refleksije in zvočnika. Višji, drugi nivo, pa uporablja predvsem funkcije prvega nivoja, vsebuje pa logiko premikanja robota in zaznavanja ovir.

4.4.1. Razred ReflectiveSensor

Konstruktor razreda ReflectiveSensor prejme dva parametra. Prvi opiše, na katerem pinu je fizično pritrjen senzor refleksije, z drugim pa mu podamo procentualno vrednost, ki predstavlja občutljivost merjenja refleksije.

```
public ReflectiveSensor(FEZ_Pin.AnalogIn pin, int reflection_detection_trigger_percentage)
{
    adc = new AnalogIn((AnalogIn.Pin)pin);
    adc.SetLinearScale(0, 100);

    if (reflection_detection_trigger_percentage < 100 && reflection_detection_trigger_percentage >= 0)
        trigger_level = reflection_detection_trigger_percentage;
    else
        throw new Exception("You must enter a percentage value betweeon 0 and 100");
}
```

Spremenljivka adc nam predstavlja pin, na katerem je pritrjen senzor refleksije. S funkcijo SetLinearScale naredimo lestvico za analogno vrednost, ki jo preberemo iz senzorja. V konstruktorju hkrati nastavimo globalno spremenljivko »trigger_level«, ki jo uporabimo za primerjavo izmerjene vrednosti. Zelo pomembna je predhodna kalibracija robota glede na tip delovne površine, saj je natančnost zaznavanja refleksije odvisna od omenjene spremenljivke. Pri svetlih podlagah je ta vrednost lahko nekoliko višja, pri temnih pa nižja, saj je tu reflektivnost težje izmeriti.

Instanca razreda AnalogIn vsebuje funkcijo Read, katera vrne trenutno analogno vrednost opazovanega senzorja. Podatek, ali obstaja refleksija, dobimo na naslednji način:

```
public DetectingState GetState()
{
    return (adc.Read() >= trigger_level) ? DetectingState.ReflectionDeteced
        : DetectingState.ReflectionNotDetected;
}
```

4.4.2. Razred FEZMini_Robot

Vsak kanal (kontrolnik motorja) potrebuje dva signala:

- Signal za kontrolo hitrosti, s katero se vrti DC motorček za poganjanje robota.

- Signal za kontrolo smeri, v katero se premikata DC motorja.

Spodnji izsek kode prikazuje deklaracijo globalnih spremenljivk obeh motorjev in njihovih signalov. Signal za kontrolo hitrosti je povezan na digitalni izhod Di5, signal za kontrolo smeri premikanja motorja pa na Di3. Za drugi motor sem uporabila naslednja prosta izhoda, to sta Di6 in Di9.

Signal za kontrolo hitrosti je pridobljen iz PWM (Pulse Width Modulation) signala. PWM ali impulzno širinska modulacija je način moduliranja signala tako, da frekvenca impulzov ostaja nespremenjena, spreminja pa se odnos impulz/pavza. Frekvenca impulzov ni pomembna, pomemben je odnos impulz/pavza. Le-ta lahko variira od 0 do 100 %.

```
static PWM _pwm1         = new PWM((PWM.Pin)FEZ_Pin.PWM.Di5);
static OutputPort _dir1 = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di3, false);

static PWM _pwm2         = new PWM((PWM.Pin)FEZ_Pin.PWM.Di6);
static OutputPort _dir2 = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di9, false);
```

Primer spreminjanja hitrosti in smeri glede na podano hitrost za levi motor (speed_left) je prikazan spodaj:

```
if (speed_left < 0)
{
    _dir1.Write(true);
    _pwm1.Set(1000, (byte)(100 - Math.Abs(speed_left)));

}
else
{
    _dir1.Write(false);
    _pwm1.Set(1000, (byte)speed_left);
}
```

Logika spreminjanja hitrosti in smeri za desni motor je podobna zgornjemu izseku kode. Razlika je le v izhodnih portih.

```
if (speed_right < 0)
{
    _dir2.Write(true);
    _pwm2.Set(1000, (byte)(100 - Math.Abs(speed_right)));
}
else
{
    _dir2.Write(false);
    _pwm2.Set(1000, (byte)speed_right);
}
```

Funkcija razreda PWM ima metodo set, ki nastavi PWM frekvenco in obratovalni ciklus. Sprejme dva parametra. Prvi je tipa int in predstavlja frekvenco v Hz, drugi pa predstavlja obratovalni ciklus, ter je izražen v odstotkih. Obratovalni ciklus je čas, ki ga naprava preživi v aktivnem stanju glede na delček celotnega obravnavanega časa.

Nenadna hitra gibanja ustvarjajo val energije, ki lahko povzročijo ponastavitev robota. Zaradi tega je potrebno za spreminjanje hitrosti narediti funkcijo, ki ublaži morebitne nenadne sunke.

```
static public void MoveRamp(sbyte speed_left, sbyte speed_right, byte ramping_delay)
{
    sbyte temp_speed_left = _last_speed_left;
    sbyte temp_speed_right = _last_speed_right;

    while ((speed_left != temp_speed_left) || (speed_right != temp_speed_right))
    {
        if (temp_speed_left > speed_left)
            temp_speed_left--;

        if (temp_speed_left < speed_left)
            temp_speed_left++;

        if (temp_speed_right > speed_right)
            temp_speed_right--;

        if (temp_speed_right < speed_right)
            temp_speed_right++;

        Move(temp_speed_left, temp_speed_right);
        Thread.Sleep(ramping_delay);
    }
}
```

4.4.3. Razred Program – vstopna točka aplikacije

V razredu Program, ki vsebuje Main funkcijo in je tudi vstopna točka v aplikacijo, se nahaja statična funkcija MoveRobot. Funkcija predstavlja »možgane« robota. Izkorišča pomožne funkcije, ki sem jih opisala v prejšnjih poglavjih in predstavlja vezo med zunanjim svetom in robotom.

Prva vrstica funkcije sproži gibanje robota. Takoj zatem pridemo v neskončno zanko, v kateri se odvija logika zaznavanja ovir. V primeru, da robot zazna oviro, izvede naslednje korake:

- Ustavi. Funkcija Move (0, 0)
- Premakne malce nazaj. Funkcija MoveRamp (-40, -40, 10)
- Obrne. Funcija Move (-40, 40)
- Ustavi. Funkcija Move (0, 0)
- Nadaljuje v smeri, kamor se je predhodno obrnil. Funkcija MoveRamp (-40, -40, 10)

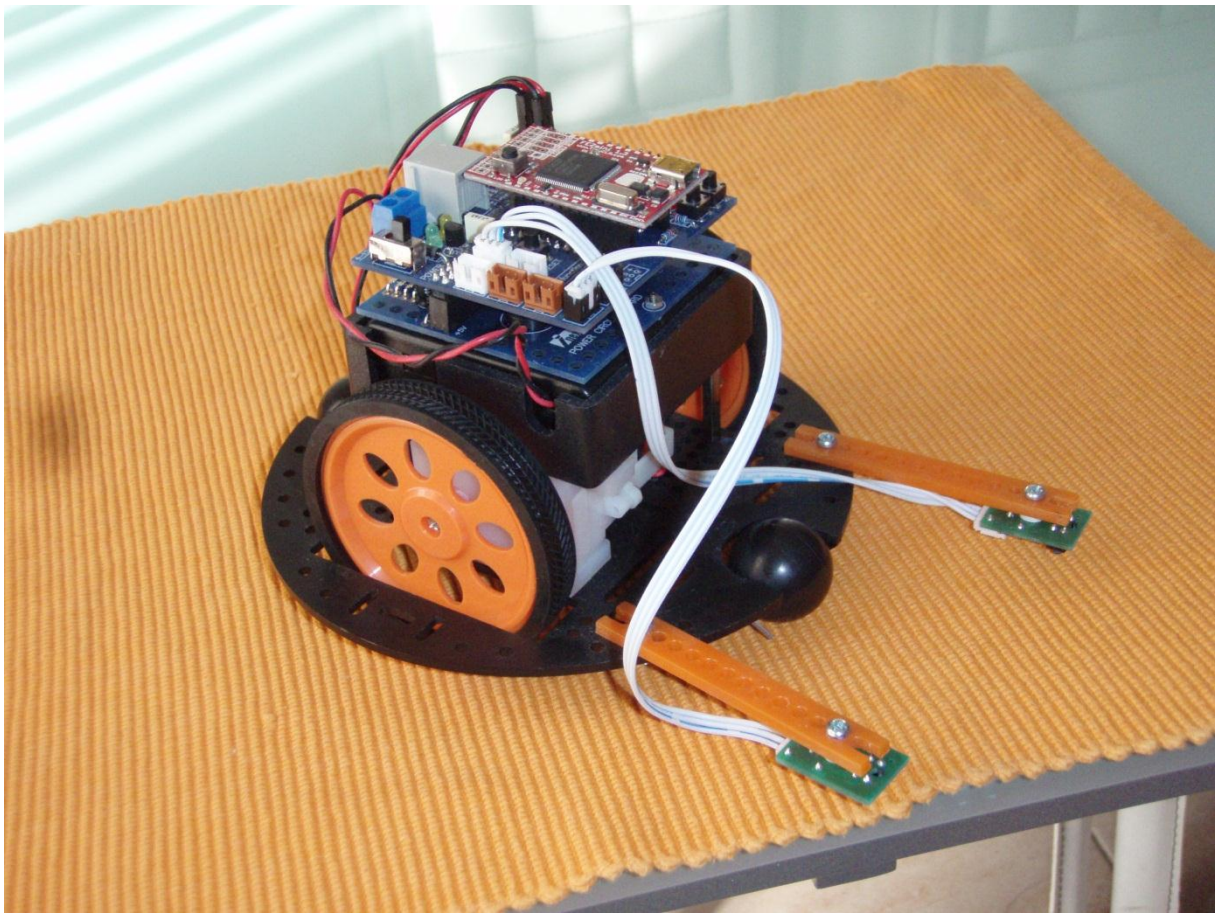
```
static void MoveRobot()
{
    FEZMini_Robot.MoveRamp(40, 40, 10);
    while (true)
    {
        if (LeftSensor.GetState() == FEZ_Components.ReflectiveSensor.DetectingState.ReflectionNotDetected
        || RightSensor.GetState() == FEZ_Components.ReflectiveSensor.DetectingState.ReflectionNotDetected)
        {
            FEZMini_Robot.Move(0, 0);
            FEZMini_Robot.MoveRamp(-40, -40, 10);
            Thread.Sleep(50);
            FEZMini_Robot.Move(-40, 40);
            Thread.Sleep(200);
            FEZMini_Robot.Move(0, 0);
            FEZMini_Robot.Beep(7000, 1000);
            FEZMini_Robot.MoveRamp(40, 40, 10);
        }
        Thread.Sleep(10);
    }
}
```

Levi in desni senzor sta priključena na analogna vhoda A1 in A6. V deklaraciji spremenljivk, ki predstavljata levi in desni senzor, navedemo tudi procentualni nivo praga zaznavanja refleksije. Na temnih podlagah je ta vrednost manjša, na svetlejših pa večja. Deklaracija izgleda takole:

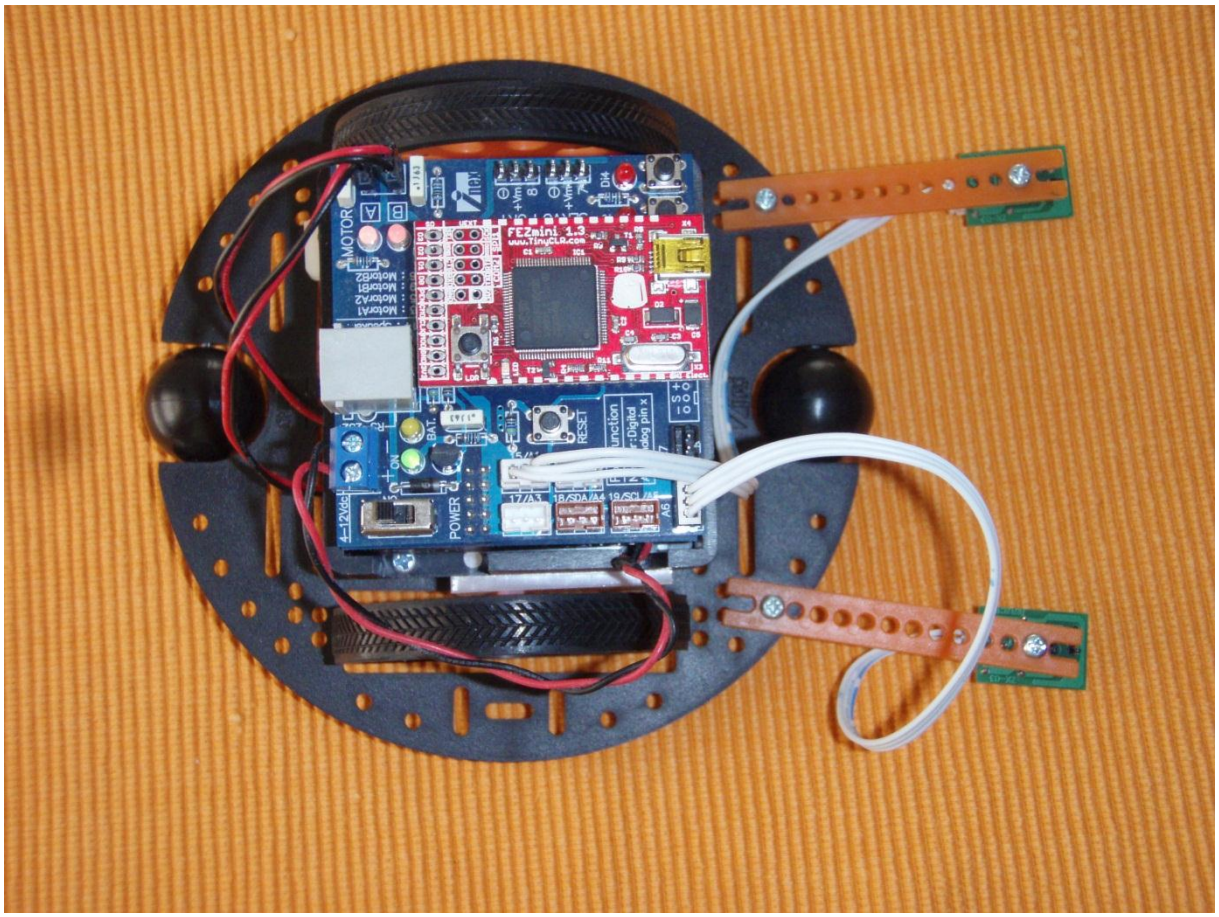
```
static FEZ_Components.ReflectiveSensor LeftSensor =
    new FEZ_Components.ReflectiveSensor(FEZ_Pin.AnalogIn.An1, 70);

static FEZ_Components.ReflectiveSensor RightSensor =
    new FEZ_Components.ReflectiveSensor(FEZ_Pin.AnalogIn.An6, 70);
```

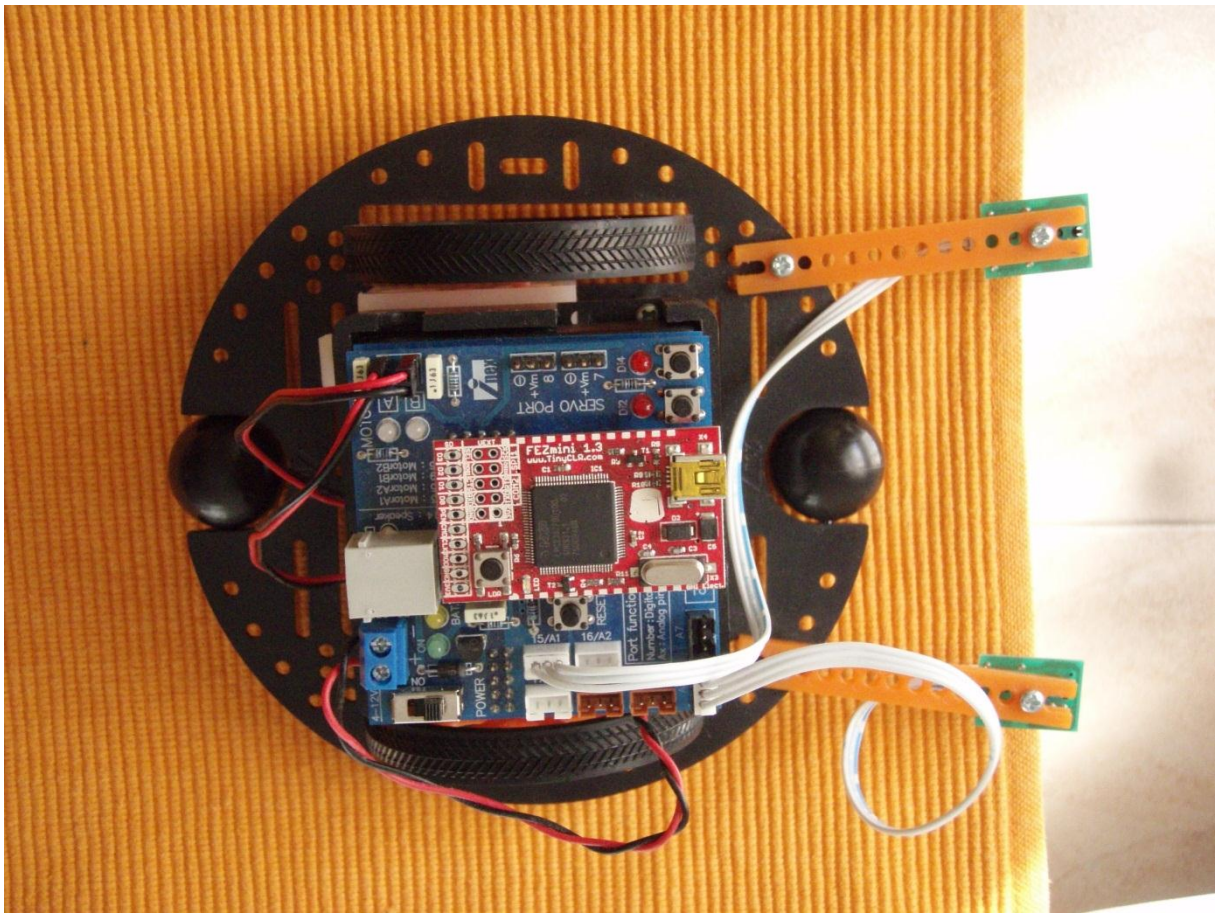
4.4.4. Slikovni prikaz delovanja aplikacije



Slika 20: Robot s senzorji refleksije, FEZ kartico in elektromotorji



Slika 21: Robot v premikanju



Slika 22: Robot zazna oviro

5. Zaključek

5.1. Težave in omejitve

Težave pri delovanju celotnega sistema se pokažejo predvsem v ekstremnih pogojih, kadar je podlaga zelo temna. Senzor merjenja refleksije v takih primerih ne zazna refleksije in posledično sporoča robotu, da se pred njim nahaja ovira, pa čeprav se ne. Robot se v takih primerih vrti okoli svoje osi. Glede na to, da sem uporabila nizkocenovni senzor merjenja refleksije, menim, da bi z zamenjavo senzorja, ki ima večjo meritveno občutljivost, pomanjklivost odpravila.

Kar se tiče splošnih omejitev .NET Micro Frameworka jih vidim predvsem v tem, da ni sistem v realnem času. Nimamo nadzora nad tem, kdaj se zažene smetlar, ki nam počisti neuporabljane objekte. Kadar se to zgodi, se ustavi delovanje celotnega programa, smetlar na kopici sprosti reference na neuporabljane objekte ter trenutne objekte prestavi v začetek nezasedenega prostora v pomnilniku.

Čiščenje pomnilnika poteka zelo hitro, tako da je za večino aplikacij sprejemljiv takšen način delovanja. Problem vidim predvsem v kakšnih specializiranih proizvodnih avtomatiziranih sistemih, kjer obstaja več samostojnih modulov, ki morajo biti časovno usklajeni. Tudi aplikacije s področja izstrelitev raket niso primerne za kodiranje v .NET Micro Frameworku, saj ima lahko milisekunden zamik pri izstrelitvi hude posledice. Za vseh 99 % ostalih aplikacij pa zaradi enostavnosti priporočam .NET Micro Framework.

5.2. Ideje za nadaljni razvoj

Diplomska naloga predstavlja temelje, na katerih se lahko zgradijo večji sistemi. V primeru, da razširimo funkcionalnosti, lahko naredimo:

- Robotski sesalnik.
- Dinamičen sistem varovanja. Če sistem nadgradimo s kamero in senzorji za zaznavanje človeške toplote, lahko izdelamo robota, ki varuje določen prostor in v primeru človeške prisotnosti to sporoči nadzornemu sistemu.
- Avtomatski zalivalnik vrta. Sistem nadgradimo s senzorji za merjenje vlažnosti zemlje. V primeru, da je vlažnost zemlje pod dovoljeno mejo, se avtomatsko zažene vodna črpalka in zalije vrt.

Ponudba senzorskih elementov na trgu je zelo bogata, ustrezna kombinacija senzorskih elementov pa nam ponuja veliko idej za nadaljni razvoj.

6. Literatura

[1] Tiny CLR. Dostopno na:

http://www.tinyclr.com/

[2] .NET Micro Framework. Dostopno na:

http://en.wikipedia.org/wiki/.NET_Micro_Framework

[3] Microsoft Visual Studio. Dostopno na:

http://en.wikipedia.org/wiki/Visual_studio

[4] C# programski jezik. Dostopno na:

http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[5] .NET Micro Framework. Dostopno na:

http://www.microsoft.com/en-us/netmf/default.aspx

[6] Jens Kühner, *Expert .NET Micro Framework*, New York, 2008.

7. Izjava

Izjavljam, da sem diplomsko nalogo izdelala samostojno pod vodstvom mentorja doc.dr.Rok Rupnika.

Tamara Vinko