UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

Aljoša Skočir

PROGRAMSKI VMESNIK ZA PRIKLOP NAPRAVE ZA ZAJEM PODATKOV NA VODILO USB

DIPLOMSKO DELO

Mentor: doc. dr. Boštjan Murovec

Ljubljana, september 2009

Zahvala

Zahvaljujem se mentorju doc. dr. Boštjanu Murovcu za pomoč, strokovne nasvete ter njegovo potrpežljivost pri pregledovanju diplomskega dela. Zahvala gre tudi sodelavcem v podjetju L-tek, d.o.o in Robiju Rostoharju za vso pomoč pri projektu.

Posebna zahvala pa gre Jasmini in obema družinama za vso moralno podporo.

Povzetek

Današnja tehnologija in zmogljivosti računalnikov omogočajo, da lahko posameznik na enostaven in predvsem poceni način spremeni svoj osebni računalnik v preprost merilni sistem, ki zagotavlja dovolj zmogljiv zajem podatkov.

V diplomskem delu je opisana izdelava programskega vmesnika, imenovanega Octopus, k obstoječi programski opremi za uporabo merilnih sistemov, ki jih lahko priključimo na osebni računalnik. Programska oprema PoScope 4 je razvita za operacijski sistem Microsoft Windows in se naslanja na nekatere njegove storitve, ki so ključne tudi za implementacijo našega programskega vmesnika.

Za PoScope 4 smo razvili programski vmesnik (vtičnik), ki predstavlja vez med priključeno merilno napravo in glavno aplikacijo.

Z razvojem vodila USB so se odprle nove možnosti priključitve merilnih naprav na osebni računalnik. Na trgu je mnogo mikrokrmilnikov, ki omogočajo prenos podatkov preko USB vodila, hkrati pa že vsebujejo analogno-digitalne pretvornike, ki omogočajo zadovoljivo kakovost pretvorbe analogne napetosti v digitalno vrednost, kar omogoča cenovno ugodno realizacijo sistemov zajemanja podatkov, ki vsebujejo zmogljivo povezavo z osebnim računalnikom.

Delo se osredotoča na razvoj programskega vmesnika, ki bo uporabniku programske opreme omogočil uporabo nove merilne naprave, ki za prenos podatkov uporablja vodilo USB.

Ključne besede: USB vodilo, Osciloskop, Octopus

Abstract

This thesis describes the development of the software plugin named Octopus for existing measurement software for usage on personal computers. Measurement software PoScope 4 is developed for Microsoft Windows 2000/XP/Vista operating systems.

We developed plugin for PoScope 4 which represents a bridge between connected measurement device and main measurement software.

USB bus adds ability to connect measurement devices to personal computers. Market offers numerous microprocessors with integrated USB bus support and analog to digital converters with good quality of data conversion. Affordable data acquisition system with good performances can readily be made.

Our work is focused on developement of software plugin with support for USB bus and therefore usage of measurement device with USB bus.

Key word: USB bus, Oscilloscope, Octopus

Kazalo

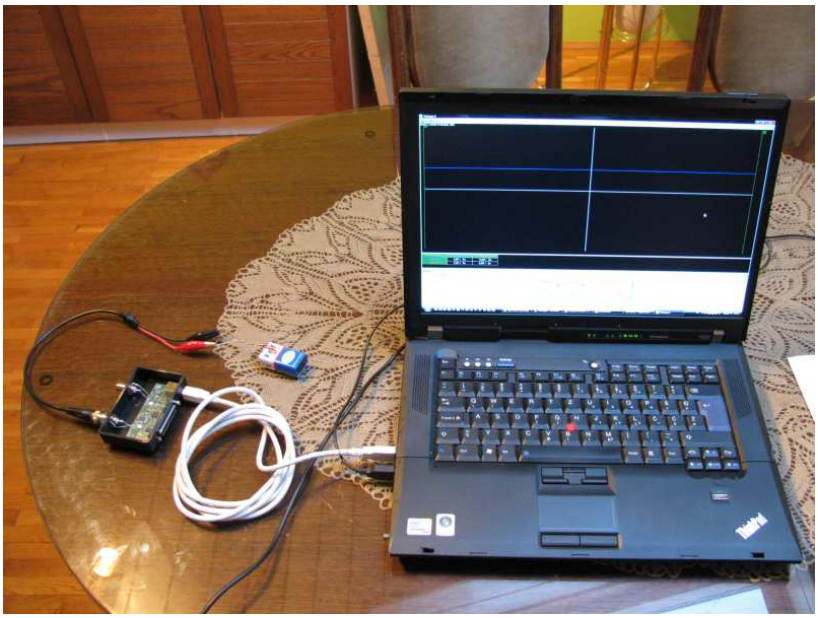
Kazalo slik

Kazalo tabel

1 Uvod

V zadnjih letih smo priča hitremu naraščanju zmogljivosti računalnikov in njihovi razširjenosti. Danes računalnik spremlja praktično vsako gospodinjstvo. Visoka zmogljivost je pripeljala do tega, da se lahko računalniki in računalniški sistemi uporabljajo za izvajanje kompleksnih procesov, zbiranje velike količine podatkov ter njihovo analizo in shranjevanje.

Novejši merilni sistemi v svojo verigo uporabe vključujejo tudi računalniške sisteme, ki ponujajo izredno fleksibilnost pri zajemu merjenih podatkov, njihovi obdelavi in analizi. Velikokrat merilni sistem predstavlja kar računalnik sam s primerno strojno in programsko opremo, pri čemer slednja pogostokrat predstavlja most med računalnikom in vgrajenimi ali zunanjimi napravami zajemanja fizikalnih veličin. Slika 1 prikazuje merilni sistem, ki meri napetost baterije.



Slika 1: Merilni sistem

Vodila, preko katerih je zunanja merilna strojna oprema priklopljena na računalnik, se med seboj razlikujejo po specifikacijah, zmogljivosti in protokolu prenosa podatkov. Nekatera so primernejša za aplikacije, kjer je pomembna količina prenesenih podatkov, druga za situacije, kjer je pomembna zanesljivost

prenosa podatkov. Izbira primernega vodila je v zadnjem času precej omejena, ker vsak nov osebni računalnik oz. prenosni računalnik ponuja samo najbolj razširjena in popularna vodila (npr. USB).

USB vodilo je zaradi preprostega načina uporabe med uporabniki postalo izredno popularno. Splošno gledano bo vsak uporabnik znal napravo priključiti in izključiti iz vodila na čelni plošči računalnika, ki zahteva samo kabel. USB vodila v osebnih računalnikih mnogokrat predstavljajo edino vodilo, ki je na voljo uporabniku za priklop naprav, kot so tipkovnica, miška, zvočniki in podobno.

2 Vodilo USB

Poglavje je povzeto po virih [1], [3] in [10].

USB (angl. Universal serial bus), ki je bil leta 1996 predstavljen s strani združenja podjetij Intel, Compaq, Microsoft, Digital, IBM in Northern Telecom, predstavlja serijsko vodilo za povezovanje zunanjih naprav z osebnim računalnikom. Njegov glavni namen je bil omogočiti priklop različnih naprav (po funkciji) preko standardiziranega vmesnika, ki dovoljuje priklop in odklop naprav brez ponovnega zagona računalnika ter omogočiti napajanje zunanjih naprav z nizko porabo in možnost uporabe naprav brez posebnih gonilnikov. Hkrati pa so poskušali odpraviti pomanjkljivost stare arhitekture, ki je zahtevala, da vsaka naprava potrebuje svojo prekinitveno sponko (angl. interrupt pin).

Tabela 1 prikazuje različna vodila, njihove prednosti, načine uporabe in zmogljivosti.

Vmesnik	**Tip**	**Število naprav**	**Največja razdalja (m)**	**Najvišja hitrost (b/s)**	**Primer uporabe**
USB	asinhronski serijski	127	do 5m oz. 30m s petimi razdelilniki	1,5M, 12M, 480M	miška, tipkovnica, itd.
Eternet	serijski	1024	do 500m.	10G	omrežje
IEEE-1394b (FireWire 800)	serijski	64	do 100m.	3,2G	video
IEEE-488 (GPIB)	paralelni	15	do 20m.	8M	merilni inštrumenti
IrDA	asinhronski serijski infrardeči	2	do 2m.	16M	tiskalniki, dlančniki…
I^2C	sinhronski serijski	40	do 6m.	3,4M	komunikacije v mikrokontrolerjih
Microwire	sinhronski serijski	8	do 3m.	2M	komunikacije v mikrokontrolerjih
MIDI	serijski s tokovno zanko	2	do 15m.	31,5k	glasbeni inštrumenti
Paralelni tiskalniški port	paralelni	2	3-10m	8M	tiskalniki, skenerji
RS-232	asinhronski serijski	2	15-30m	do 115k	modem, miška
RS-485	asinhronski serijski	2	1,2km	10M	merilni instrumenti in nadzorni sistemi
SPI	sinhronski serijski	8	10	2,1M	komunikacije v mikrokontrolerjih

Tabela 1: Vodila in njihove značilnosti [1, Poglavje 1, str. 3]

2.1 Pridobitve za uporabnika

Za uporabnika so prednosti USB vodila enostavna uporaba, hiter in zanesljiv pretok podatkov, fleksibilnost, cenenost in možnost napajanja naprav z nizko porabo.

Uporaba USB vodila naj bi bila enostavna in bi imela prednosti:

- **En vmesnik za več različnih naprav**. USB vodilo je vsestransko in ga lahko uporabljamo z različnimi napravami.

- **Samodejna konfiguracija**. Ko uporabnik vklopi napravo preko USB vodila na PC, jo operacijski sistem zazna in naloži primerne gonilnike. Ko so gonilniki nameščeni, ni potreben ponovni zagon računalnika.

- **Enostaven priklop.** Značilnost USB vodila je, da ni potrebno za vsako napravo odpirati ohišja računalnika in vanj vstavljati posebnih razširitvenih kartic. Tipičen osebni računalnik ima 4 ali več USB priključkov. Priključki se lahko po želji razširjajo s t.i. USB razdelilniki (angl. USB hub).

- **Preprosti kabli.** Priključki na USB kablih so razvrščeni tako, da ni mogoč napačen priklop. En kabel je lahko dolg do 5 metrov, z razdelilniki pa lahko meri skupna dolžina do 30 metrov. V primerjavi s priključki RS-232 so priključki majhni in kompaktni. Da se zagotovi zanesljivost prenosa, specifikacija strogo določa električne lastnosti kablov in priključkov.

- **Neposreden priklop/odklop.** USB naprava je lahko priklopljena ali odklopljena kadarkoli, ne glede na to ali je sistem ali naprava vklopljena, brez rizika okvare. Operacijski sistem zazna kdaj je naprava priključena in jo pripravi za delovanje.

- **Ni uporabniških nastavitev.** USB naprave nimajo uporabniško definiranih nastavitev, kot so naslovi in prekinitvene linije, zato ni pripomočkov za konfiguracijo, s tem pa je manj zmede med uporabniki.

- **Prosta strojna sredstva za druge naprave.** Z uporabo večih naprav na USB vodilu sprostimo prekinitvene linije za naprave, ki jih za svoje delovanje dejansko potrebujejo. Osebni računalnik določi serijo naslovov in eno prekinitveno linijo USB krmilniku. Posamezna USB naprava ne zahteva dodatnih sredstev (npr. prekinitvenih linij) v nasprotju z drugimi napravami.

- **Napajanje naprav.** USB vmesnik vsebuje in omogoča uporabo napajanja +5 V iz USB razdelilnika, če poraba naprave ne prekorači toka 500 mA.

- **Hitrost.** USB podpira tri hitrosti vodila: visoko hitrost do 480 Mb/s, polno hitrost do 12 Mb/s, in nizko hitrost do 1,5 Mb/s. Krmilnik USB različice 2.0 podpira vse tri hitrosti.

Hitrost vodila odraža hitrost potovanja surovih bitov po vodilu. Glede na podatke mora vodilo prenašati tudi signale za status, kontrolo in preverjanje napak. To pomeni, da je prenos podatkov manjši od hitrosti vodila. Teoretični maksimum hitrosti za enkratni prenos podatkov je okrog 53 MB/s pri visoki hitrosti, 1,2 MB/s pri polnih hitrostih in 800 B/s pri nižji hitrostih. USB 1.0 specifikacija definira nižjo in polno hitrost.

Nižja hitrost je podprta zaradi dveh razlogov. Miška zahteva fleksibilne kable za lažje premikanje naprave po površini. Kabli za nižje hitrosti ne zahtevajo paric ter oklopa in so zaradi tega bolj fleksibilni, kot pri polni in višji hitrosti; posledično so nižji tudi stroški izdelave naprave. Naprave za polno hitrost so bile mišljene, da nadomestijo večino naprav, ki uporabljajo RS-232 in paralelna vodila. Hitrost prenosa podatkov pri polni hitrosti je primerljiva ali še boljša, kot pri predhodnikih. Naprave z visoko hitrostjo so postale možnost izbire z USB različico 2.0.

- **Zanesljivost.** USB protokol omogoča zaznavanje napak v prejetih podatkih in obveščanje pošiljatelja z namenom, da jih ponovno pošlje. Detekcija, obveščanje in ponovno pošiljanje so implementirane na strani strojne opreme in ne zahtevajo programiranja s strani uporabnika.

- **Nizka cena.** Ne glede na to da je USB vodilo precej bolj kompleksno glede na predhodnike, so komponente in kabli poceni. Primerjava v ceni bi USB napravo postavila ob bok starejši napravi, ki podpira standard IEEE-1394.

- **Nizka poraba energije.** Vezja, ki zmanjšajo porabo in koda, ki avtomatsko izključi USB napravo, lahko prihranijo precej energije. Zmanjšana poraba energije privarčuje denar, je okolju prijazna in za naprave, ki so napajanje z baterijami predstavlja daljše delovanje.

- **Brezžične komunikacije.** USB vodilo je bilo razvito kot žični vmesnik, vendar se že pojavljajo možnosti naprav za brezžično povezavo USB z osebnim računalnikom.

2.2 Slabosti vodila

Poleg naštetih prednosti omenimo naslednje slabosti USB vodila.

- **Omejitve standarda.** Vsak standard ima omejitve in USB ni izjema. USB ima omejitve vodila, ki se navezujejo na hitrosti prenosa po vodilu in razdalje, pomanjkanje podpore komunikacijam med enakovrednimi uporabniki oz. napravami (angl. peer to peer), pomanjkanje podpore starejši strojni opremi in operacijskim sistemom.

- **Hitrost.** Visoka hitrost USB vodila, ga postavlja ob bok vmesniku IEEE-1349a (FireWire) s hitrostjo 400 Mb/s, vendar je IEEE-1394b še vedno hitrejši s 3,2 Gb/s.

- **Razdalja.** USB vodilo je bilo ustvarjeno kot namizno razširitveno vodilo s pričakovanji, da so naprave pri razdalji rok, zato je dolžina USB kabla

omejena na 5 metrov. Drugi vmesniki npr. RS-232, RS-485, IEEE-1394b in Ethernet dovoljujejo daljše kable. Kabli so lahko podaljšani s petimi razdelilniki v skupni dolžini 30 m. Če želimo prenose na daljše razdalje lahko pretvorimo USB vodilo v RS-485.

- **Oddajanje.** USB ne omogoča hkratnega pošiljanja sporočil večim napravam na vodilu. Gostitelj mora poslati sporočilo vsaki napravi posebej z razliko od vodil IEEE-1394 ali Etherneta.

- **Starejša strojna oprema.** Starejši računalniki nimajo možnosti uporabe USB naprav. Če želimo uporabljati starejšo napravo na USB vodilu, je rešitev v pretvorniku, ki pretvori med USB in starejšim vmesnikom npr. RS-232 ali RS-485. Če želimo uporabljati USB vodilo z računalnikom, ki nima USB priključkov je potrebno dodati USB krmilnik in namestiti operacijski sistem, ki podpira USB. Strojna oprema je v obliki razširitvene kartice, ki jo vklopimo v PCI režo na matični plošči. Različica oken mora biti Windows 98 ali kasnejša.

2.2.1 USB različica 2.0

Uporaba USB različice 1.x je med uporabniki postajala vse bolj razširjena, zato se je standard razvijal naprej. Standard za USB različico 2.0 predpisuje višje zmogljivosti kot njegov predhodnik. Hitrosti, ki jih dosega USB različica 2.0 so zanimive tudi za naprave, ki za prenos podatkov potrebujejo hitra in zmogljiva vodila (npr. video kamera, disk, TV). USB različica 2.0 je kompatibilna s starejšo USB različico 1.1 in naprave z USB različico 2.0 delujejo tudi, če so priklopljene na krmilnik z USB različico 1.x.

USB On-The-Go je nadgradnja USB različice 2.0, kjer naprava lahko deluje kot gostitelj. Primer take uporabe je direktni priklop tiskalnika na fotoaparat ali pa celo fotoaparata na drug fotoaparat.

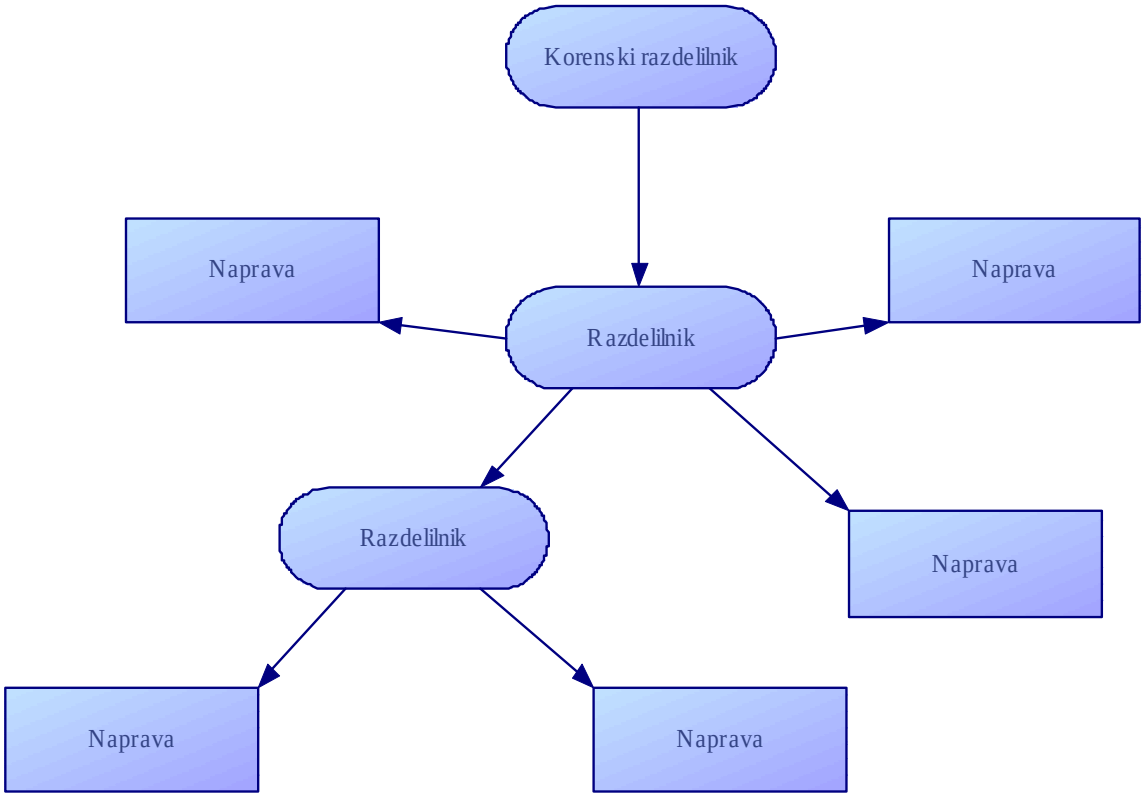
Dodatek k USB je tudi brezžični USB, ki omogoča brezžično komunikacijo z napravami s hitrostjo do 480 Mb/s. Specifikacija je na voljo od leta 2005.

2.3 Komponente USB vodila

Fizične komponente USB vodila so sestavljene iz vezij, priključkov in kablov med gostiteljem in eno ali več napravami.

Gostitelj je osebni računalnik, ki vsebuje USB krmilnik in korenski razdelilnik (angl. root hub) Te komponente delujejo skupaj tako, da omogočajo operacijskemu sistemu komunikacijo z napravami na vodilu. Gostiteljski krmilnik oblikuje podatke za prenos na vodilu in prevaja prejete podatke v obliko, ki jo operacijski sistem lahko uporabi. Ravno tako je način komunikacije naloga gostiteljevega krmilnika. Korenski razdelilnik (angl. Root hub) ima enega ali več priključkov za priklop naprav. Korenski razdelilnik v kombinaciji z gostiteljevim krmilnikom zazna priklop in odklop naprav, prenaša zahteve iz gostiteljevega krmilnika in prenaša podatke.

Razdelilnik (ang. Hub) ima enega ali več priključkov za priklop naprav. Vsaka naprava vsebuje vezje in programsko kodo, ki ve kako komunicirati z gostiteljem. Slika 2 prikazuje uporabo razdelilnika.



Slika 2: Prikaz uporabe razdelilnika (angl. Hub) [1, poglavje 1, str. 18]

2.4 Razdelitev dela

Gostitelj in naprava imata definirane odgovornosti in dela. Gostitelj skrbi za vzdrževanje komunikacije, medtem ko naprava vsebuje inteligenco za odzivanje in komunikacijo ter druge dogodke na vodilu, ki jih zahteva oz. sproža gostitelj.

2.4.1 Dolžnosti gostitelja

Da lahko računalnik deluje kot gostitelj in komunicira z napravami na USB vodilu potrebuje podporo strojne in programske opreme. Strojni del je sestavljen iz gostiteljevega krmilnika in korenskega razdelilnika z enim ali več priključki. Programska podpora predstavlja operacijski sistem, ki zagotavlja mehanizem za gonilnike, ki lahko komunicirajo s strojno opremo. Danes vsak osebni računalnik vsebuje USB krmilnik in enega ali več USB priključkov. Veliko osebnih računalnikov vsebuje več gostiteljskih krmilnikov. Če osebni računalnik nima podpore vodilu USB na matični plošči, se lahko brez težav doda razširitvena kartica v eno izmed rež na matični plošči. Prenosni računalniki imajo običajno leto vgrajeno na matični plošči.

Gostiteljev krmilnik nadzoruje naprave na vodilu in njihove zmožnosti, hkrati pa mora vsaki napravi na vodilu zagotavljati oddajo in sprejem podatkov. Vodilo lahko vsebuje več naprav, vsako z drugačnimi zahtevami.

2.4.1.1 Zaznava naprav

Pri priklopu naprave razdelilnik opozori gostiteljev krmilnik o vseh priključenih napravah. V postopku, ki se imenuje enumeracija, gostiteljev krmilnik povzame naslov in od vsake naprave zahteva dodatne informacije. Po zagonu, kadarkoli je naprava priklopljena ali izklopljena, gostitelj dobi informacijo in enumerira vse na novo priklopljene naprave ter odstrani vse odklopljene naprave iz seznama naprav.

2.4.1.2 Urejanje pretoka podatkov po vodilu

Gostiteljev krmilnik ureja pretok podatkov po vodilu. Več naprav lahko prenaša podatke hkrati in krmilnik razdeli čas, ki je na voljo v segmente, ki se imenujejo okvirji (angl. Frames) in mikrookvirji (angl. Microframes) ter ga dodeli vsakem prenosu. Časovno kritični prenosi imajo v vsakem okvirju rezerviran del časa, ki je potreben za njihovo izvršitev. Med enumeracijo gonilnik naprave zahteva pasovno širino (angl. bandwidth), pri kateri se bodo vršili prenosi, ki so časovno kritični. Če ta pasovna širina ni na voljo, krmilnik naprave ne dovoli konfigurirati na zahtevani način. Prenosi, ki nimajo garantiranega časa prenosa, uporabijo preostali čas okvirja in morajo počakati, da je vodilo prosto.

2.4.1.3 Preverjanje napak

Pri prenosu podatkov gostiteljev krmilnik doda kodo CRC-32 za odkrivanje napak. Pri sprejemu podatkov naprava izvrši izračune na podatkih in primerja rezultat z prejeto kodo CRC-32. Če se rezultat ne ujema, naprava ne sprejme podatkov in gostiteljev krmilnik ve, da mora podatke poslati ponovno. USB podpira tudi način prenosa, ki ne dovoljuje prepošiljanja podatkov v interesu, da se vzdržuje konstantni prenos podatkov. Gostiteljev krmilnik lahko prejme tudi druge indikatorje, ki sporočajo težave z napravo in jih zna pravilno obdelati, da aplikaciji sporoči npr. da naprava ni na volji itd.

2.4.1.4 Zagotavljanje napajanja

USB kabel ima dve napajalni žici za 5 V napajanje naprav. Nekatere naprave se lahko brez težav napajajo preko teh žic. Gostiteljev krmilnik zagotavlja napajanje vsem napravam pri zagonu ali pri preklopu. Naprave, ki jih napaja vodilo, lahko porabijo do 500 mA toka. Naprave imajo lahko tudi svoje napajanje.

2.4.2 Naloge naprave

V večini primerov so naloge naprave zrcalna oblika nalog gostiteljevega krmilnika. Ko gostitelj vzpostavi komunikacijo, se mora naprava odzvati. Naprave pa imajo tudi precej unikatna opravila in nekatere razlike glede na krmilnik kot je, da naprava ne more vzpostaviti povezave sama od sebe. Namesto tega mora naprava počakati in se odzvati na gostiteljevo pobudo za komunikacijo.

2.4.2.1 Naslavljanje naprave

Vsaka naprava pregleduje naslov naprave, ki je vsebovan v vsakem podatkovnem paketu. Če se naslov razlikuje od naslova, shranjenega na napravi, le-ta paket ignorira, v nasprotnem primeru pa podatke shrani in uporabi. USB podprta integrirana vezja vršijo naslavljanje avtomatično in ne zahtevajo posebne podpore v kodi.

2.4.2.2 Odziv na standardne zahteve

Ob priklopu in zagonu se mora naprava odzivati na standardne zahteve, ki jih pošlje gostiteljev krmilnik med enumeracijo. Gostitelj lahko pošlje zahteve tudi po enumeraciji.

Vse naprave se morajo odzivati na standardne zahteve. Ko naprava prejme zahtevo po podatkih ali statusnih informacijah, le-te pošlje gostitelju.

Specifikacija USB vodila definira 11 zahtev. Proizvajalec naprave lahko definira dodatne zahteve. Napravi ni potrebno odgovoriti na vsako zahtevo, važno je le da se odzove v razumljivem načinu. Npr. gostitelj zahteva konfiguracijo, ki jo naprava ne podpira in se naprava odzove s kodo, ki indicira da ta konfiguracija ni mogoča oz. podprta.

2.4.2.3 Preverjanje napak

Tako kot gostiteljev krmilnik naprava doda podatkom za pošiljanje kodo CRC-32 za preverjanje pravilnosti prenosa. Pri prejemu podatkov, ki vsebujejo bite za preverjanje, naprava izvrši izračun na podatkih. Odziv naprave ali pa neodziv naprave pove gostitelju, da naj podatke ponovno pošlje. Ta funkcionalnost je že vgrajena v krmilnik in ne zahteva posebnega programiranja. Če je vse v redu naprava gostitelju to sporoči, s čimer se prenos določenega paketa zaključi.

2.4.2.4 Upravljanje porabe

Naprava se lahko napaja iz vodila ali pa ima svoje napajanje. Za naprave, ki uporabljajo napajanje iz vodila in kadar vodilo ni aktivno, morajo naprave omejiti porabo toka na vodilu. Ko gostiteljev krmilnik preide v nižjo porabo napajanja, prenehajo vse komunikacije na vodilu. Ko se zazna odsotnost aktivnosti vodila za več kot 3 ms mora naprava v suspendirano stanje in omejiti porabo toka na vodilu. Ko je v suspendiranem stanju, naprava opazuje dogajanje na vodilu in preide iz suspendiranega v aktivno stanja, ko vodilo ponovno postane aktivno.

Naprave, ki ne podpirajo zbujanja iz stanja suspendirano, ne smejo imeti porabe večje od 500 mA. Če naprave podpirajo zbujanje iz suspendiranega stanja in gostiteljev krmilnik to omogoča, potem je omejitev toka 2,5 mA.

2.4.2.5 Izmenjava podatkov z gostiteljem

Za večino prenosov, kjer gostiteljev krmilnik pošlje podatke napravi, se mora naprava odzvati z odgovorom ali je prejela podatke ali ne. Tipično se strojni del odziva avtomatično glede na nastavitve na napravi. Nekateri prenosi ne uporabljajo obveščanja in pošiljatelj predvideva, da je naslovnik podatke prejel. Integrirano vezje krmilnika skrbi za celostno oblikovanje podatkov za na vodilo.

2.5 Hitrost vodila

Krmilnik naprave mora podpirati eno ali več hitrosti. Virtualno vsi razdelilniki podpirajo nižje in polne hitrosti naprav. Naprave z nižjo in polno hitrostjo se lahko povežejo na katerikoli USB razdelilnik. Uporabnikom ni potrebno vedeti ali gre za napravo z nižje ali polno hitrost, ker zanju ni nastavitev.

Naprave z višjo hitrostjo so pravzaprav dvojno hitrostne naprave, ki funkcionirajo s polno hitrostjo. USB različica 1.x ni podpirala višjih hitrosti. Da naprave, ki podpirajo višjo hitrost ne zmedejo krmilnika, ki podpirajo samo USB različico 1.x morajo na enumeracijo odzivati s polno hitrostjo. V tem primeru lahko vsak gostitelj identificira napravo. Pri polni hitrosti se vršijo ponastavitve vodila in standardne zahteve.

Dejanska prepustnost podatkov med napravo in gostiteljevim krmilnikom je manjša od hitrosti vodila in ni vedno predvidljiva. Nekateri preneseni biti so uporabljeni za identifikacijo, sinhronizacijo in preverjanje napak, zaradi česar je prepustnost odvisna tudi od tipa prenosa in zasedenosti vodila.

2.5.1 Končne točke

Ves promet po vodilu potuje od ene do druge končne točke naprave (angl. Device Endpoint). Ta končna točka (angl. Endpoint) predstavlja pomnilnik (angl. Buffer), ki hrani podatke. Tipično je končna točka blok podatkovnega pomnilnika ali register v krmilniku. Podatki, shranjeni na končni točki so lahko prejeti podatki ali pa podatki, ki čakajo na razpošiljanje. Gostitelj ima pomnilnik, ki drži prejete podatke nima pa končnih točk. Namesto tega gostitelj servira začetek in konec komunikacije s končno točko.

Specifikacija USB vodila definira končno točko kot unikaten naslovni del USB naprave, ki je izvor ali ponor podatkov v komunikaciji med gostiteljevim krmilnikom in napravo. Specifikacija predpisuje da končna točka nosi podatke samo v eni smeri. Končna točka je sicer lahko tudi dvosmerna.

Naslov končne točke je sestavljen iz številke končne točke in smeri. Številka je med 0 in 15. Smer je definirana z gostiteljeve perspektive: vhodna (angl. IN) končna točka zagotavlja tok podatkov v gostiteljev krmilnik, medtem ko izhodna (angl. OUT) končna točka prejema podatke iz gostiteljevega krmilnika. Končna točka, definirana za kontrolne prenose, mora prenašati v obe smeri, zato vsebuje dva naslova, ki si delita isto številko končne točke. Vsaka naprava mora imeti končno točko 0 definirano kot kontrolno končno točko.

Glede na končno točko 0, lahko imajo naprave ki podpirajo polno ali visoko hitrost do 30 dodatnih naslovov končnih točk (1 do 15 za vsako smer). Naprave z nižjo hitrostjo so omejene na dva naslova dodatnih končnih točk v katerikoli kombinaciji.

Vsaka transakcija na vodilu se začne s paketom, ki vsebuje številko končne točke in kodo, ki označuje smer toka podatkov.

2.5.2 Cevovodi

Preden se lahko prenos izvrši, morata gostitelj in naprava vzpostaviti cevovod (angl. pipe), ki je povezava med končno točko naprave in medpomnilnikom gostiteljevega krmilnika.

Gostiteljev krmilnik vzpostavi cevovode med enumeracijo. Če je naprava odstranjena iz vodila, gostiteljev krmilnik odstrani neuporabljene cevovode. Gostiteljev krmilnik lahko tudi zahteva nove cevovode ali odstrani neuporabljene pri spremembi konfiguracije naprave. Vsaka naprava ima osnovni kontrolni cevovod, ki uporablja končno točko 0.

Konfiguracijske informacije, ki jih prejme gostiteljev krmilnik, vključujejo opise končnih točk za vsako točko posebej. Vsaka končna točka je blok informacij, ki gostitelju povedo, kako naj komunicira z njo. Informacija vsebuje naslove končne točke, tip prenosa, maksimalno velikost podatkovnih paketov in želen interval prenosa.

2.6 Načini prenosa podatkov

USB vodilo pokriva potrebe različnih naprav z različnimi zahteve glede hitrosti prenosa podatkov, odzivnih časov in popravljanja napak, zaradi česar vodilo definira štiri načine prenosa podatkov.

Kontrolni prenos (angl. Control transfer) je edini prenos, ki ima funkcije definirane s strani specifikacije USB. Kontrolni prenos omogoča gostiteljevemu krmilniku prebrati informacije o napravi, nastaviti napravi naslov, nastaviti konfiguracije in druge nastavitve. Poleg tega se kontrolni prenos lahko uporablja za zahteve, ki so specificirane s strani proizvajalca naprave. To vrsto prenosa morajo podpirati vse USB naprave.

Obsežni prenos (angl. Bulk transfer) se uporablja, ko hitrost prenosa ni kritična (npr. pošiljanje datoteke na tiskalnik, prejemanje datoteke iz čitalnika), pomembna pa je pravilnost prenesenih podatkov. Če je vodilo zasedeno se ti

prenosi zakasnijo. Ta način prenosa lahko uporabljajo samo naprave, ki podpirajo polno in višjo hitrost vodila.

Prekinitveni prenos (angl. Interrupt transfer) je primeren kadar mora imeti naprava periodično pozornost gostiteljevega krmilnika. V nasprotju s kontrolnim prenosom je prekinitveni prenos edini način prenosa podatkov za naprave z nižjo hitrostjo. Tako tipkovnice in miške uporabljajo prekinitveni način prenosa podatkov.

Periodični prenos (angl. Ischronus transfer) ima zagotovljen dostavni čas za podatke, vendar ne omogoča popravljanja napak. Primeren je za prenos zvočnih ali slikovnih podatkov, ki se prenašajo v realnem času. Ta način prenosa je tudi edini, ki ne omogoča ponovnega pošiljanja podatkov, če le-ti vsebujejo napako. Zaradi tega je možnost napak podatkov relativno velika . Samo naprave s polno in visoko hitrostjo lahko prenašajo podatke na tak način.

2.6.1 Obsežni prenos

Ta način prenosa je primeren za prenos podatkov, kadar čas ni kritična komponenta prenosa. Obsežni prenos lahko prenese veliko količino podatkov brez oviranja vodila, ker se prenos podredi drugim načinom prenosov in počaka dokler vodilo ni na voljo. Če je vodilo prosto, je tak način prenosa najhitrejši.

2.6.1.1 Razpoložljivost

Samo naprave s polno in visoko hitrostjo lahko uporabljajo obsežen način prenosa. Za primer: Naprava v razredu naprav za masovno shranjevanje podatkov mora imeti vsaj eno končno točko tipa za obsežen prenos definirano v obe smeri.

2.6.1.2 Struktura

Obsežni prenos je sestavljen iz ene ali več vhodnih ali izhodnih transakcij in je enosmeren: transakcije so lahko samo ali vhodne ali izhodne. Prenos podatkov v obe smeri zahteva ločena cevovoda za vsako smer.

Obsežen prenos se konča v treh primerih: kadar se zaključi prenos pričakovane količine podatkov, kadar transakcija vsebuje prazne zloge (ničle) ali pa je število prenesenih zlogov manjše od velikosti podatkovnega paketa. Struktura podatkov je lahko poljubna.

2.6.1.3 Velikost paketov

Naprave s polno hitrostjo in obsežnim prenosom imajo lahko največje velikosti paketov v mejah 8, 16, 32 ali 64 zlogov. Za naprave z višjo hitrostjo je lahko največja velikost paketa 512 zlogov. Med enumeracijo gostiteljev krmilnik prebere maksimalno velikost paketa za vsako končno točko iz opisa naprave (angl. Device's descriptor).

2.6.1.4 Hitrost

Gostiteljev krmilnik zagotavlja, da se obsežen prenos konča, vendar ne zagotavlja pasovne širine za prenos. Ko je vodilo nedejavno, lahko obsežen prenos uporabi skoraj celo pasovno širino in ima majhen presežek (angl. overhead), zato je najhitrejši. Kadar je največja velikost paketa manjša od največje dovoljene, nekateri krmilniki ne prenesejo več kot enega paketa znotraj enega okvirja, kljub temu da je na voljo več pasovne širine. Zato je najbolje vedno izbrati največjo možno velikost podatkovnega paketa za obsežen prenos podatkov.

Pri polni hitrosti je mogoče prenesti do devetnajst 64 zlogov velikih podatkov, kar pomeni 1216 zlogov na okvir oziroma hitrost okrog 1,216 MB/s. To pušča 18% pasovne širine vodila na voljo za druge operacije. Presežek za obsežni prenos z enim podatkovnim paketom je 13 zlogov pri polni hitrosti in 55 zlogov pri višji hitrosti.

Pri višji hitrosti je možno prenesti do trinajst 512 zlogov velikih paketov, kar pomeni 6646 zlogov na okvir in pomeni hitrost okrog 53,248 MB/s in pušča 2% pasovne širine proste. Presežek za en podatkovni paket znaša 55 zlogov. V resnici se ti rezultati precej razlikujejo od sistema do sistema.

2.6.1.5 Zaznava in odprava napak

Ta način prenosa uporablja zaznavanje napak. Če naprava ne pošlje paketa s pravilnimi podatki, ga gostiteljev krmilnik poskuša poslati še dvakrat.

3 Razvoj programskega vmesnika

Ločen dodatek programske kode k obstoječemu programu imenujemo vtičnik ali preprosto programski vmesnik. V računalništvu v angleškem jeziku jih poznamo pod več imeni npr. plugin, extension, add-on. Naloga programskega vtičnika je, da obstoječemu programu (pod pogojem, da le-ta omogoča vmesnike) razširi uporabnost na specifično področje (npr. tiskanje posebnih obrazcev, obdelava podatkov, izvoz oz. uvoz podatkov).

V operacijskem sistem Windows so najbolj razširjeni vtičniki v obliki datoteke DLL in jih imenujemo dinamične knjižnice (angl. Dynamic Linked Library). [2]

Dinamične knjižnice lahko vsebujejo funkcije in podatke, ki jih uporablja poljuben program ali druga knjižnica. Funkcije so lahko notranje (lokalne) ali izvožene (eksportirane). Slednje se lahko kličejo iz drugih programov in iz same knjižnice, medtem ko so notranje funkcije omejene na klice znotraj knjižnice. Knjižnica lahko izvaža podatke, vendar samo v obliki parametrov izvoženih funkcij.

Uporaba dinamičnih knjižnic zmanjša porabo pomnilnika, kadar več programov uporablja enako funkcionalnost ob istem času. Vsi programi uporabljajo enako kodo iz knjižnice, medtem ko ima vsak svojo kopijo podatkov.

Za uporabo dinamičnih knjižnic moramo poznati imena in parametre funkcij, ki jih le-te izvažajo. Težava pri dinamičnih knjižnicah je, ker ne izvažajo novih tipov razredov in spremenljivk.

Načelno novih podatkovnih tipov glavna aplikacija ne pozna dokler nima nekega vmesnika (angl. Interface), ki implementira prototip ustreznega razreda. Če v izvorni kodi (npr. v posebni datoteki) naredimo prototip nekega novega tipa razreda in vključno s to datoteko prevedemo glavno aplikacijo in dinamično knjižnico, ustvarimo vez in hkrati omogočimo dostop do nadaljnjih izpeljank tega razreda v dinamični knjižnici.

Na tem mestu govorimo o neki vrsti programskega razvojnega dodatka (angl. SDK – Software developement kit). Ker je SDK je skupen glavni aplikaciji in dinamični knjižnici oba dela poznata prototipe osnovnih razredov. Vsi razredi, ki so nadalje izpeljani iz teh prototipov, se lahko uporabljajo v glavni aplikaciji, čeprav so implementirani v dinamični knjižnici. Slabost tega je da je pri spremembi SDK potrebno ponovno prevesti glavno aplikacijo in vse dinamične knjižnice, ki SDK uporabljajo.

Primer: V SDK definiramo prototip razreda TMojeOknoPrototip. Vse metode in funkcije definiramo kot virtualne in abstraktne. S to določitvijo ustvarimo čisti prototip razreda, torej zgolj opis nekega novega tipa. V kodo za dinamično knjižnico vključimo SDK datoteko s prototipi. Nadalje definiramo nov tip razreda TMojeOkno, ki podeduje lastnosti tipa TMojeOknoPrototip. V tem razredu realiziramo prototipne metode in funkcije. V glavni aplikaciji medtem poskrbimo, da se dinamična knjižnica naloži in izvožena funkcija vrne kazalec na naš novi razred. Kazalec tipiziramo (angl. Typecast) z TMojeOknoPrototip in lahko uporabljamo metode ter funkcije. Čeprav ima ta razred v sebi definirane virtualne funkcije in metode, kazalec kaže na objekt, ki ima te funkcije realizirane, zato se bodo klicale funkcije izpeljanega razreda.

Prednosti pri uporabi vtičnikov s stališča razvoja so:

- razširjanje funkcionalnosti glavni aplikaciji,
- posodobitve in nadaljnji razvoj brez posega v glavno aplikacijo ter
- širok spekter uporabe.

Prednosti pri uporabi s stališča uporabnikov:

- pri prostem dostopu do SDK-ja imajo uporabniki možnost razvoja lastnega vtičnika ter
- razvoj po naročilu oz. specifikaciji stranke.

Seveda pa obstajajo tudi slabe lastnosti [2]:

- kot smo omenili, v vtičnikih ne moremo implementirati novih tipov razredov in spremenljivk ter jih potem izvoziti (brez uporabe SDK),

- glavno aplikacijo in DLL vtičnik je potrebno prevesti z enako različico SDK datotek, ki jih uporabljamo (da ni težav s tipi razredov, ki se med različicami SDK-ji razlikujejo).

3.1 Projekt OCTOPUS

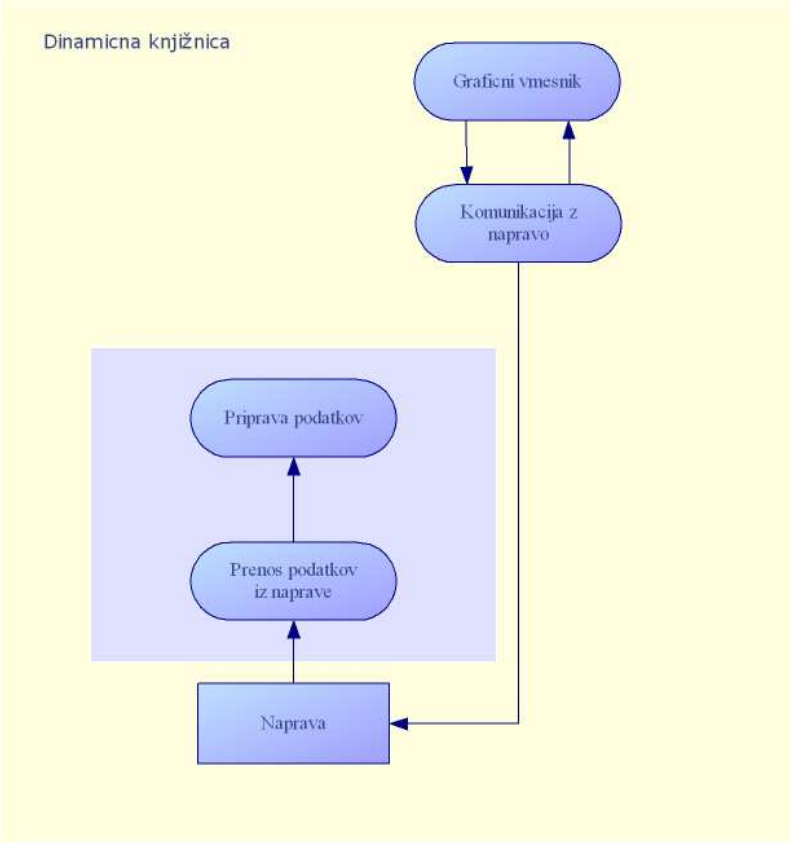
Projekt OCTOPUS zajema realizacijo naprave in vtičnika, ki obstoječi programski opremi doda funkcionalnost osciloskopa na osebnem računalniku.

Pri tem je potrebno razviti vse osnovne funkcije profesionalnih osciloskopov:

- prikaz analognih signalov v časovni odvisnosti,
- prikaz analognih signalov v medsebojni odvisnosti,
- kazalci vrednosti,
- različni načini proženja,
- frekvenčna analiza,
- logični analizator,
- itd…

Naš del naloge je realizirati vtičnik, ki vsebuje grafični vmesnik, komunikacijo z napravo in dekodiranje podatkov. Najnižji nivo predstavlja prenos podatkov iz naprave, sledi dekodiranje in obdelovanje podatkov za nadaljnji prikaz in nazadnje še grafični vmesnik, preko katerega nadzorujemo dogajanje na napravi.

Najprej smo razvili kodo, ki skrbi za prenos trenutnih podatkov iz naprave. S pomočjo uporabe prototipnih razredov iz SDK, smo implementirali svoje razrede za komunikacijo z USB napravo, za prenos trenutnih podatkov iz naprave ter za grafični vmesnik. Novi tipi razredov so nam omogočili, da je npr. komunikacija z napravo domena dinamične knjižnice. Glavna aplikacija, ki dinamično knjižnico uporablja ne ve, da podatki, ki prihajajo po pomnilniku (angl. Buffer) prihajajo po USB vodilu. To je pomembna prednost, ker komunikacijo omejimo na dinamično knjižnico in s tem celoten sistem naredimo modularen. Slika 3 prikazuje koncept delovanja naše dinamične knjižnice.



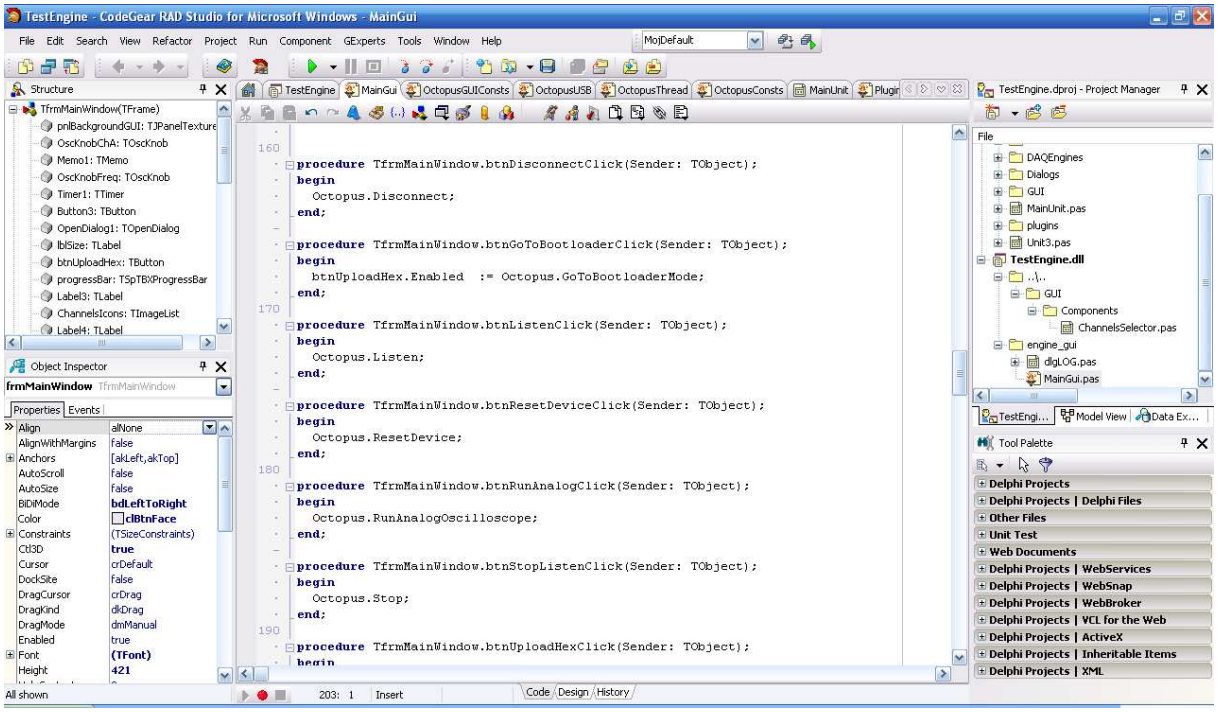
Slika 3: Koncept delovanja dinamične knjižnice Octopus

3.1.1 Izbira programskega jezika

Pri izbiri programskega jezika za razvoj vtičnika smo se oprli na:

- poznavanje jezika
- predhodna znanja in izkušnje,
- razširjenost,
- podporo razvijalcev in razvojnih skupin.

Pri izbiri je pretehtala odločitev o poznavanju programskega jezika in preteklih izkušnjah, zato smo izbrali Delphi 2007, ki je baziran na programskem jeziku Pascal in ima podporo večini Windows API funkcijam [6], bogat nabor vključenih grafičnih komponent in razvijalcu prijazen grafični vmesnik oz. programsko okolje (IDE, angl. Integrated Development Environment). Slika 4 prikazuje IDE Delphi 2007.



Slika 4: Grafični vmesnik razvojnega okolja Delphi 2007

Projekt v Delphi-ju je sestavljen iz projektne datoteke (Delphi Project, *.dpr) na katero so navezane datoteke enot (Units, *.pas) in datoteke obrazcev (*.dfm). Datoteki enote, ki vsebuje dialoge, okna in ostale vizualne kontrole, pripada datoteka z enakim imenom in končnico *.dfm. V DFM datoteki so zapisane lastnosti vizualnih kontrol (pozicija gumbov, vnosnih polj, uvoženih slik itd).

3.1.2 Gonilnik naprave

Za fizični dostop in prenos podatkov med napravo in osebnim računalnikom imamo na voljo vodilo USB. Pri naši napravi, ki podpira polno hitrost smo za pretok podatkov uporabili način obsežnega prenosa. Za dostop do podatkov preko vodila USB pa potrebujemo tudi gonilnik naprave, ki poskrbi za virtualizacijo dostopa programske opreme do naprave.

3.1.2.1 Gonilnik WinUSB

Na tržišču obstaja mnogo podjetij, ki se ukvarjajo z razvojem univerzalnih gonilnikov naprav. Univerzalni gonilnik je v gonilnik naprave, ki mu sami specificiramo nastavitve in opise za delovanje, tako da ga lahko uporabimo večkrat za različne načine delovanja in za različne naprave. Če govorimo o univerzalnem gonilniku USB naprave, imamo v mislih, da lahko ta gonilnik uporabimo za različne USB naprave tako po tehničnih nastavitvah kot po delovanju (npr. osciloskop).

Gonilnik naprave je največkrat sestavljen iz datoteke (*.sys), ki predstavlja vez med jedrom operacijskega sistema in napravo, ter datoteko uporabniških funkcij (*.dll) in klicev, s katerimi posredno dostopamo do naprave. (npr. winusb.sys - gonilnik, winusb.dll – uporabniške funkcije). Kakšna je struktura in način delovanja gonilnika, je predpisano s strani proizvajalca naprave, ki določa standarde pri izdelavi gonilnika itd.

Pri izbiri primernega gonilnika moramo biti pozorni na sledeča merila:

- prednosti in širina spektra uporabe (univerzalnost),
- podpora s strani proizvajalca,
- dokumentacija,
- nabor uporabniški funkcij in klicev,
- podprtost razvojnih okolij,
- nadaljnji razvoj in podpora,
- podpora različnim platformam,
- podpora različnim operacijskim sistemom in
- cena.

Za našo aplikacijo smo izbrali WinUSB, ki je proizvod firme Microsoft [7]. S tem si nekako zagotovimo podporo na vseh Microsoftovih produktih. Prednost je tudi, da je lahko dostopen razvijalcem preko WDK (angl. Windows Development Kit) in je brezplačen za uporabo. V Windows Vista je gonilnik že vključen, na Windows XP pa ga je potrebno namestiti. Namestitev je enostavna, dokumentacija je javno dostopna, uporabniške funkcije so enostavno dostopne, pa tudi precej razvojnih skupin uporablja ta gonilnik. Izkušnje kažejo na

razširjeno uporabo med razvijalci in posledično tudi enostavno implementacijo. S strani Microsofta lahko pričakujemo nadaljnje posodobitve in podporo.

V paket WDK so vključene datoteke, ki vsebujejo prototipe uporabniških funkcij in strukture za uporabo z uporabniško knjižnico za dostop do vodila. Slabost je, da so te datoteke oz. glave (angl. Headers) namenjene programskim jezikom, ki bazirajo na jeziku C/C++.

Kljub temu smo precej hitro in brez posebnih težav prevedli datoteke za naš izbrani programski jezik Delphi 2007 (Units, *.pas). Test delovanja uporabniških funkcij s prevedenimi datotekami pa smo opravljali sproti in brez večjih težav.

3.1.2.2 Prenos podatkov

Po dogovoru označujemo tipe razredov z dodano črko T pred imenom (npr. TThread), iz obstoječih izpeljane razrede pa s črko F pred imenom (npr. FThread).

Za implementacijo prenosa podatkov smo definirali nov razred z imenom TOctopusDAQ, ki izhaja iz splošnega tipa razreda TObject. Ta objekt se ne izvaža iz dinamične knjižnice, zato ni potrebe po uporabi SDK. Naloga objekta tega tipa je prenos in urejanje podatkov med gonilnikom USB naprave in glavno aplikacijo.

Sledi je implementacija razreda FThread, ki je izpeljan iz razreda TThread. FThread predstavlja podedovani razred za implementacijo niti. Razred FThread ima glavno metodo Execute, v kateri je implementiran ves prenos in urejanje podatkov. Ob klicanju inicializacije dinamične knjižnice se ustvari objekt tipa TOctopusDAQ in Fthread, s katerim se ustvari ločena nit za branje.

Struktura razreda **TOctopusDAQ**:

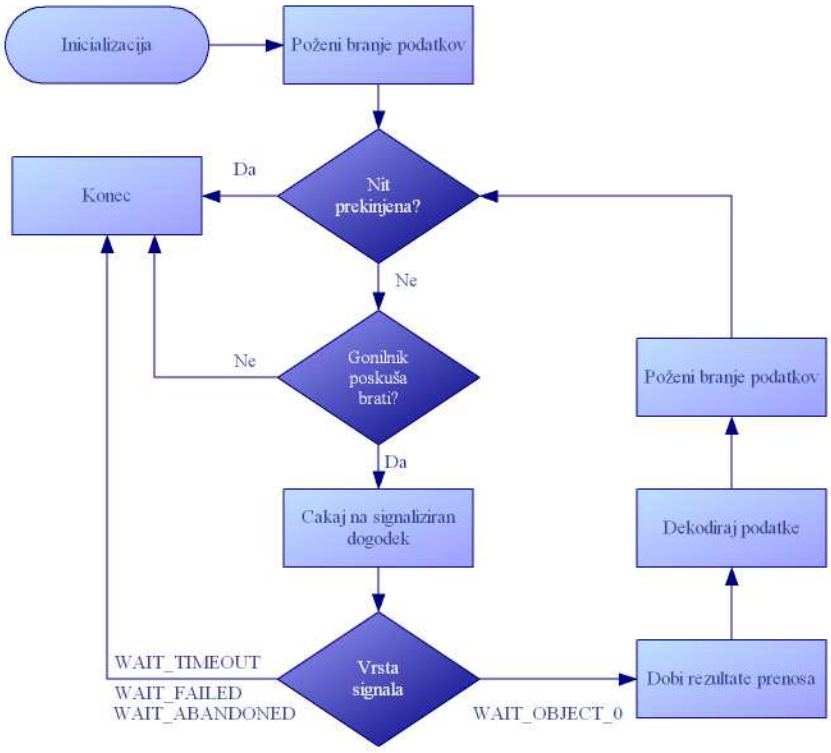
- **FThread (tip TThread):** Referenca na nit, ki opravljala operacije prenosa. Ko se pokliče funkcija Initiate razreda TOctopusDAQ, se ustvari nit za branje podatkov. Dinamična knjižnica ima v inicalizacijskem bloku del kode, ki nastavi trenutnemu procesu visoko prioritetni razred. S tem lahko tej niti, ki bere podatke, nastavimo prioriteto na časovno realno stopnjo, (angl. Real time priority), ki pa v resnici ni časovna realnost v dobesednem pomenu, ampak zgolj višja prioriteta obdelave oz. bolj pogosto dodeljevanje procesorskega časa tej niti.
- **FChannelsList (tip TEngineChannelsList)**: Referenca na drevesno strukturo kanalov s tekočimi podatki. Kanal je tipa TEngineChannel in v sebi nosi referenco na del pomnilnika, namenjenega podatkom in implementaciji metod za rokovanje s podatki. Njihova implementacija je realizirana v našem SDK-ju.
- **FOctopusBackbone (tip TOscilloscopeBackbone)**: Referenca na tip razreda, ki definira osnovo za vse naprave, ki imajo t.i. osciloskopove lastnosti. Gre za univerzalni razred, v katerem implementiramo komunikacijo preko poljubnega vodila in funkcije za komunikacijo z napravo. Prototip je definiran v SDK, implementacija pa v vtičniku.
- **FDataSync (tip TCriticalSection)**: To je referenca na uporabniški objekt, ki se uporablja za sinhronizacijo podatkov med nitmi, ki posegajo po podatkih na istem naslovu v pomnilniku. Ko se ustvari seznam kanalov, se skreira tudi objekt tega tipa in njegova referenca se prenaša po glavni aplikaciji ter po dinamični knjižnici. Uporablja se pri vpisovanju podatkov v pomnilnik in pri branju podatkov iz pomnilnika. Ker uporabljamo več niti (za branje, pisanje) nikoli ne vemo, kaj se zgodi prej: pisanje ali branje. Kritična situacija nastane, kadar poskušata dve niti hkrati pisati na isti naslov v pomnilniku. Ker ne morem določiti, katera nit je prva oz. druga, moramo poskrbeti za ekskluzivni dostop samo ene niti. To naredimo tako, da s *FDataSync.Enter* in *FDataSync.Leave* označimo vsak del bloka kode, kjer se dogajajo operacije pisanja in branja podatkov. Na ta način bomo, dovolili oz. omejili dostop do spremenljivke na nekem naslovu samo eni nit hkrati. Vsak vtičnik ima svojo referenco na sinhronizacijski objekt [8, 9].

- **FscheduleList (tip TList)**: Zaradi načina, kako podatki prihajajo po USB vodilu v računalnik, se izkaže, da nam prav pride prioritetno urejen seznam kazalcev na kanale. Podatki po vodilu prihajajo v točno določenem vrstnem redu. Vrstni red je določen s prioriteto posameznega kanala in tako je določeno s strani protokola, ki ga predpisujejo razvijalci naprave. Načinu razvrščanja podatkov po kanalih je namenjeno poglavje 3.1.2.3 .

Metode, implementirane v razredu **TOctopusDAQ**:

- **Initiate**: preveri, če je naprava priklopljena, ustvari prioritetno urejen seznam za vpis dekodiranih podatkov (ustvari nit FThread, kjer se ji nastavijo parametri, kot je sinhronizacijski objekt, struktura za sinhronizacijo, prioriteta). Na koncu se nit še požene.
- **Clear**: Prekine delovanje niti.

Diagram na sliki 5 prikazuje potek branja iz naprave.



Slika 5: Branje podatkov iz naprave (ločena nit)

1. Inicializacija: S funkcijo Initiate sprožimo branje podatkov iz naprave. Najprej se ustvarijo objekti, potrebni za podatke in nit ter se jim nastavijo parametri. Ko je nit zagnana, se izvrši funkcija Execute, v kateri je implementirana programska koda po zgornjem grafu poteka (slika 5). Najprej se nastavijo začetne vrednosti spremenljivk in kazalci na razne strukture.

   Branje poteka asinhrono, ki je v Windows API zelo uporaben način [6]. Pri asinhronem branju funkcija, ki jo kličemo sprejme parameter, ki je kazalec na strukturo prekrivanja. Struktura prekrivanja vsebuje parametre, ki povedo število prenesenih podatkov, odmik od ničle pri branju in ročico na objekt za sinhronizacijo.

   Urnik za niti nam zaradi visoke prioritete naše niti nameni precej več časa za obdelovanje kot ostalim nitim. Ker je v metodi Execute neskončna zanka za branje, bi zaradi tega naša nit odvzela precej časa ostalim nitim z manjšimi prioritetami. To bi se odražalo v tem, da bi sistem začel delovati počasneje. To zagato rešimo tako, da v neskončno zanko damo ukaz, ki našo nit zaustavi za določen čas (npr. dokler gonilnik ne prenese podatkov) zaradi česar dobijo čas za obdelavo tudi druge niti, vendar nam fiksen čas zaustavitve ne ponuja optimalnih rezultatov.

   Ker včasih lahko branje traja malo dlje včasih malo manj oz. je ta čas precej fleksibilen, lahko to povzroči neoptimalno izkoriščen čas. Ker pri branju funkciji posredujemo kazalec na strukturo prekrivanja, se gonilniku posreduje ročico na dogodek, ki nam postavi v signalizirano stanje, ko prenese pravo število podatkov. S funkcijo *WaitForSingleObject* [4,5] ustavimo nit za toliko časa, dokler gonilnik ne postavi dogodek v signalizirano stanje. S tem dosežemo, da je ustavljanje niti najbolj optimalno. Na ta način optimalno rešimo zagato s časom, hkrati pa tudi poskrbimo, da sistem ne postane neodziven.

   Objekt za signaliziranje pri asinhronem branju se imenuje dogodek. Dogodek je preprost objekt v jedru operacijskega sistema, ki ima dve

stanji, in sicer signalizirano in nesignalizirano stanje. Uporabniške funkcije za uporabo USB gonilnika delujejo tako, da kot parameter (pri branju/pisanju) sprejmejo kazalec na strukturo prekrivanja, ki nosi ročico na dogodek.

Razlika med operacijami s podajanjem kazalca na strukturo prekrivanja in brez kazalca je sledeča. Če kazalca na strukturo prekrivanja ne podamo, se bo klicana funkcija prenosa (branje ali pisanje) končala šele, ko bodo izpolnjeni vsi zahtevani pogoji pri prenosu (npr. preneslo se bo zahtevano število podatkov).

Pri podanem kazalcu na strukturo prekrivanja in z veljavnim parametrom ročice na dogodek pa se bo klicana funkcija prenosa končala takoj in program se nadaljuje. V naslednjem koraku lahko počakamo na singalizirano stanje dogodka, ki se spremeni ko gonilnik npr. prebere/zapiše zahtevano število podatkov.

Dogodek ustvarimo z WinAPI funkcijo *CreateEvent* in njegovo ročico shranimo v strukturo. Pri ustvarjanju podamo parameter, ki nam dogodek postavi v nesignalizirano stanje in mu nastavimo lastnost, da se po signaliziranem stanju sam ponastavi. Podamo še kazalce na del pomnilnika, kamor se shranijo surovi podatki. Branje prožimo s funkcijo *WinUSB_ReadEx*.

```
function USBDevice_ReadEx
                 (hWinUsbDevice:WINUSB_INTERFACE_HANDLE;
                  PipeID:  UCHAR;
                  Buffer:  PUCHAR;
                  BufferLength:  ULONG;
                  var LengthTransferred:  ULONG;
                  Overlapped:LPOVERLAPPED):TWDUTransferResult;
```

Funkciji podamo parametre za ročico na WinUSB napravo [7], številko cevovoda, kazalec na pomnilnik, zahtevano dolžino branja, spremenljivko

za število prenesenih podatkov in kazalec na strukturo prekrivanja. Funkcija se konča takoj (preden se prenos izvrši).

2. Vstopimo v neskončno zanko oz. zanko, ki se izvaja, dokler ni prekinjena s strani niti same ali gonilnika. Takoj zahtevamo, da nit miruje toliko časa, dokler ne dobimo signaliziranega stanja dogodka, ki nam ga proži gonilnik. Uporabimo WinAPI ukaz *WaitForSingleObject*, kjer kot parameter podamo ročico na naš dogodek in maksimalen čas čakanja. Nit čaka, dokler ni signaliziran dogodek oz. dokler ne poteče maksimalen čas čakanja. Na podlagi tega dobimo rezultat izvršene funkcije v obliki konstante, ki predstavlja ali WAIT_OBJECT_0 ali WAIT_TIMEOUT. WAIT_TIMEOUT se vrne, če poteče maksimalen čas čakanja, WAIT_OBJECT_0 pa, ko je dogodek v signaliziranem stanju. Na podlagi rezultata se odločimo, kako dalje. Če preteče maksimalen čas, je nekaj narobe z napravo, zato podatkov nima smisla več zahtevati, nit za branje pa lahko ustavimo. Če dobimo signaliziran dogodek, sledimo naslednjemu koraku.

3. Pri signaliziranem dogodku nadaljujemo s postopkom. Najprej z ukazom *WinUsb_GetOverlappedResult* [7] pridobimo informacijo o število prenesenih podatkov, hkrati pa postavimo signalizirano stanje dogodka v nesignaliziranega.

4. Sledi dekodiranje pomnilnika prejetih podatkov (poglavje 3.1.2.3). Po dekodiranju takoj sprožimo novo branje in zanka se vrne na začetek, od koder se vse ponovi.
5. Če nit ustavimo je potrebno uničiti še dogodek in opraviti formalnosti s kazalci in objekti.

3.1.2.3 Priprava podatkov in dekodiranje

Priprava podatkov za uporabo in prikazovanje je odvisna predvsem od strukture prenesenih podatkov. Protokol naše naprave določa, da imamo na voljo uporabo 12-bitnih podatkov, zato smo izbrali obliko podatkov tako, da 1 dvozložni podatek predstavlja 1,5 zloga. Ker v računalniških sistemih ne poznamo tipa, ki bi bil sestavljen iz 1,5 zloga, uporabimo dvozložni tip zapisa. Torej iz treh zlogov

dobimo dva (dvozložna) podatka. Pomnilnik za podatke iz naprave je v obliki zložnih podatkov, pomnilnik za pretvorjene podatke pa je v obliki dvozložnih podatkov. Pretvorba poteka tako, da iz zložnih podatkov pridobimo dvozložne podatke, ki jih potem pomnožimo še s kalibrirnimi koeficenti, da dobimo prave vrednosti, ki pa so tipa plavajoča vejica (*double*), ki zaseda v pomnilniku 8 zlogov.

3.1.2.3.1 Struktura pomnilnika

Vsebino pomnilnika, ki jo prenesemo iz naprave, ponazarja tabela 2.

Indeks	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Vsebina	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	…

Tabela 2: Vsebina pomnilnika za podatke iz naprave (zložni pomnilnik)

Z B označimo zložni podatek v prispelem pomnilniku, s S pa označimo vzorec, ki je sestavljen iz 1,5 zloga. Z malo črko b označimo bit v zlogu, kjer zgornji indeks označuje številko zloga, spodnji pa zaporedno številko bita v zlogu. Tako sta vzorec S_0 in S_1 sestavljena takole:

$$S_0 = \left\{ B_0 : \frac{B_1}{2}_{zg} \right\} = \left\{ b_7^0 b_6^0 b_5^0 b_4^0 b_3^0 b_2^0 b_1^0 b_0^0 : b_7^1 b_6^1 b_5^1 b_4^1 \right\} \qquad (1)$$

$$S_1 = \left\{ \frac{B_1}{2}_{sp} : B_2 \right\} = \left\{ b_3^1 b_2^1 b_1^1 b_0^1 : b_7^2 b_6^2 b_5^2 b_4^2 b_3^2 b_2^2 b_1^2 b_0^2 \right\} \qquad (2)$$

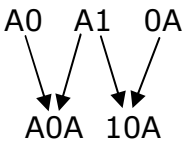
Po zapisu (1) in (2) sledi, da je S_0 zgrajen iz zloga B_0 in zgornje polovice zloga B_1, S_1 pa je zgrajen iz preostanka spodnje polovice zloga B_1 in zloga B_2.

Omenimo še dve pomembni pravil, ki smo jih določili in jih upoštevamo pri pripravi in dekodiranju podatkov:

1. priprava in dekodiranje podatkov morata steči čim hitreje,
2. v enem preletu pomnilnika je potrebno opraviti čim več operacij.

V dogovoru z razvijalci naprave nam zaradi lažjega procesiranja podatkov po USB vodilu v paketu z maksimalno velikostjo 64 zlogov (obsežni tip prenosa) pošljejo 60 koristnih zlogov, zadnje štiri zloge pa postavijo na nič (angl. Zero padding). Glavni razlog je v 1,5 zlognih podatkih. Pri 60 uporabnih zlognih podatkih dobimo pri pretvorbi 40 koristnih dvozlognih podatkov (sestavljenih iz 1,5 zloga). Če bi bilo 64 zlognih podatkov, bi bilo dekodiranje oteženo.

Primer pretvorbe enozložnih podatkov v dvozložne:



Za pretvorbo uporabimo sledeči funkciji:

```
function Convert3BytesTo2Left(B1, B2: Byte): DWORD;
begin
  Result :=  B1 shl 4 + B2 shr 4;
end;

function Convert3BytesTo2Right(B2, B3: Byte): DWORD;
begin
  Result := (B2 and $0f) shl 8 + B3;
end;
```

Pomembna pa je tudi struktura toka zložnih podatkov v pomnilniku. Predvsem je potrebno poudariti to, da naša naprava podpira zajemanje podatkov iz več kanalov. Točno določeno je tudi, h kateremu kanalu spada kateri par oz. 1,5 zloga.

Mikrokrmilnik na napravi ima vgrajena dva analogno digitalna pretvornika z zmogljivostjo pretvorbe 1MS/s in s tem dva glavna kanala. Poimenujmo kanale naprave z oznakami Kanal A, Kanal B, Kanal B0, Kanal B1 …. Kanala poimenovana A in B sta glavna kanala, kanali, ki se poimenovani z B(n) pa so podkanali. A/D pretvornik B kanala vsebuje tudi multiplekser s čimer imamo lahko dodatno 5 kanalov, ki jih poimenujemo podkanali. Njihova značilnost je da si delijo pasovno širino ter frekvenco vzorčenja z glavnim B kanalom. Kanal A

nima podkanalov. Kanala A in B sta enakovredna, njuna frekvenca vzorčenja je enaka in sicer 1 MS/s. Na podlagi tega dogovora zgradimo tabelo kanalov in sestavimo urnik, ki je razložen v nadaljevanju.

Za lihe vzorce velja, da so sestavljeni po pravilu (1) za sode pa po pravilu (2).

Tabela 3 nam prikazuje stanje, ko je vklopljen samo kanal A:

Indeks	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Vzorec/kanal	A_0^0	A_1^0	A_2^0	A_0^1	A_1^1	A_2^1	A_0^2	A_1^2	A_2^2	…	…	…	…	…	…

Tabela 3: Struktura podatkov v pomnilniku, pri čemer je vklopljen samo kanal A

Vzorci se sestavljajo po omenjenih pravilih. Torej je vzorec $A^0{}_0$ sestavljen po pravilu (1), vzorec $A^0{}_1$ pa po pravilu (2).

Tabela 4 prikazuje stanje, ko sta vklopljena kanala A in B:

Indeks	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Vzorec/kanal	A_0^0	B_0^0	A_1^0	B_1^0	A_2^0	B_2^0	A_0^1	B_0^1	A_1^1	B_1^1	A_2^1	B_2^1	…	…	…

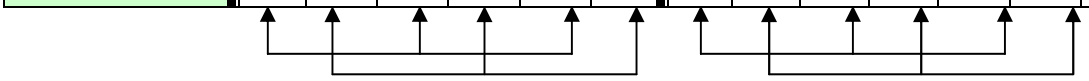


Tabela 4: Struktura podatkov v pomnilniku, vklopljen kanal A in kanal B

V tabeli 4 vidimo, da se pri vklopljenem kanalu B vzorci za kanal A ($A^0{}_0$) sestavljajo po prvem (1) pravilu, vzorci za kanal B ($B^0{}_0$) pa po drugem pravilu (2).

Tabela 5 prikazuje stanje, ko so vklopljeni kanal A, kanal B in pod kanal B0.

Indeks	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Vzorec/kanal	A_0^0	B_0^0	A_1^0	$B0_0^0$	A_2^0	B_1^0	A_0^1	$B0_1^0$	A_1^1	B_2^0	A_2^1	$B0_2^0$	…	…	…

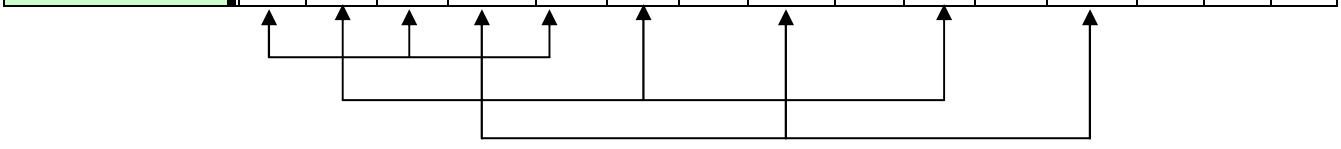


Tabela 5: Struktura podatkov v pomnilniku, vklopljen kanal A, B in B0

Če vklopimo še podkanale, je struktura bolj zapletena. Vzorci so sestavljeni ali po pravilu (1) ali po pravilu (2).

Glede na zgornje tabele se kanali na dvojico zložnih podatkov zamenjujejo v smiselnem vrstnem redu, ki temelji na njihovi prioriteti, oz. kako so porazdeljeni v drevesni strukturi kanalov (objekt FChannelsList razložen v poglavju 3.1.2.2). Najvišjo prioriteto imata kanala A in B, vsota prioritet B(n) podkanalov pa je enaka prioriteti kanala B. Torej se prioriteta kanala B razdeli na število vključenih podkanalov.

Vzorci se ponavljajo v urejenem vrstnem redu glede na prioriteto. Tisti z višjo prioriteto se bodo v pomnilniku pojavili pogosteje kot tisti z manjšo. Ker se držimo pravila, da pri enem preletu opravimo čim več operacij, namesto da se pomikamo po pomnilniku in iščemo vzorce, vpeljemo prioritetno urejen seznam kanalov.

Omenjena drevesna struktura seznama kanalov (FChannelsList) nam omogoča, da poznamo tudi prioriteto posameznega kanala. Ker poznamo prioriteto posameznega kanala lahko ustvarimo prioritetno urejen seznam kanalov. Ideja je, da namesto nepotrebnega premikanja in iskanja vzorcev raje uporabimo kanal (iz prioritetno urejenega seznama), ki pripadaj trenutnemu vzorcu.

3.1.2.3.2 Urnik - FScheduleList

Omenjene lastnosti so nam dale idejo, da zgradimo urnik kanalov. V vtičniku se ob inicializaciji ustvarijo kanali tipa TEngineChannel, ki ustrezajo specifikaciji naprave in se vstavijo v seznam TEngineChannels (definirano v SDK). Seznam je v obliki drevesne strukture (FChannelsList), kar omogoča, da ima vsak kanal podkanale itd. Zaradi drevesne strukture imamo avtomatsko podano tudi prioriteto kanala, glede na globino v drevesni strukturi. Tako vemo, kako pomemben je določen kanal oz. hkrati tudi na koliko časovnih rezin mu pripada

vzorec. Kanali na isti globini v drevesu si delijo maksimalno prioriteto višje uvrščenega kanala itd.

Urnik kanalov je prioritetno urejen seznam kazalcev na vklopljene kanale tipa TEngineChannel. Postopek prioritetnega urejanja začnemo tako, da najprej rekurzivno iz drevesne strukture pridobimo vse vklopljene/uporabljene kanale.

Prim. Vklopljene imamo kanale A, B, B0 in B1.

Skupna vsota prioritet mora biti vedno enaka 1 oz. mora odražati 100 % zasedenost urnika. Kanal A in kanal B sta na istem nivoju (sta enakovredna), zato je njuna prioriteta enaka ½. To bi v konkretnem primeru pomenilo, da se vzorci izmenjujejo enakomerno (npr. A, B, A, B…). Vzorci za kanal A so na sodem indeksu, vzorci za kanal B pa na lihem indeksu pomnilnika dvozložnih vzorcev (tabela 5). Ker pa ima kanal B vklopljena še podkanala B0 in B1, si morata podkanala med seboj enakovredno razdeliti ½ prioritete kanala B, torej se ½ deli na tri enake dele. Zato kanalu B popravimo prioriteto iz ½ na 1/6, kanaloma B0 in B1 pa prioriteto nastavimo na 1/6. Če izračunamo vsoto ½ + 3*1/6 = 1. Ko imajo kanali določene prioritete, sestavimo urnik.

Urnik nosi informacijo o tem, v kakšnem zaporedju si sledijo kanali za vsak dvozložni vzorec, ki ga dobimo iz dveh enozložnih.

Najprej izračunamo dolžino urnika SchLength, ki je enaka:

```
SchLength = 2 *(Število vklopljenih kanalov - 1)

Naredimo še korekcijo (max poskrbi da je mininalna SchLength = 1)
SchLength = 3 * Max(SchLength,1)
```

Množenje s tri je potrebno, ker prenašamo 3 zloge za prenos 2 podatkov.

Sledi še rezervacija pomnilnika za urnik in vpisovanje kanalov vanj. To storimo tako, da se z zanko sprehodimo skozi urejen seznam kanalov in si zapomnimo njihove kazalce. Za urnik rezerviramo prostor v pomnilniku.

Najprej vzamemo kanal, ki ima najvišjo prioriteto. V našem primeru je to kanal A, ki ima prioriteto 1/2. V urniku poiščemo prvo prazno mesto, ki je na voljo, vpišemo kazalec in se pomaknemo za obratno vrednost prioritete izbranega kanala naprej. V našem primeru je to 2. Torej števec povečamo za 2 in zopet vpišemo kanal. To ponavljamo, dokler ne pridemo do konca urnika. V nadaljevanju vzamemo naslednji kanal, ki sledi po prioriteti. To je kanal B s prioriteto 1/6. V urniku zopet poiščemo prvo prazno mesto za vpis. Potem se premaknemo za obratno vrednost prioritete naprej, kar tokrat znese šest mest. Ponavljamo do konca urnika. Tako postopek ponovimo za vsak nadaljnji kanal. S takšnim postopkom se ustvari urnik, urejen po prioriteti.

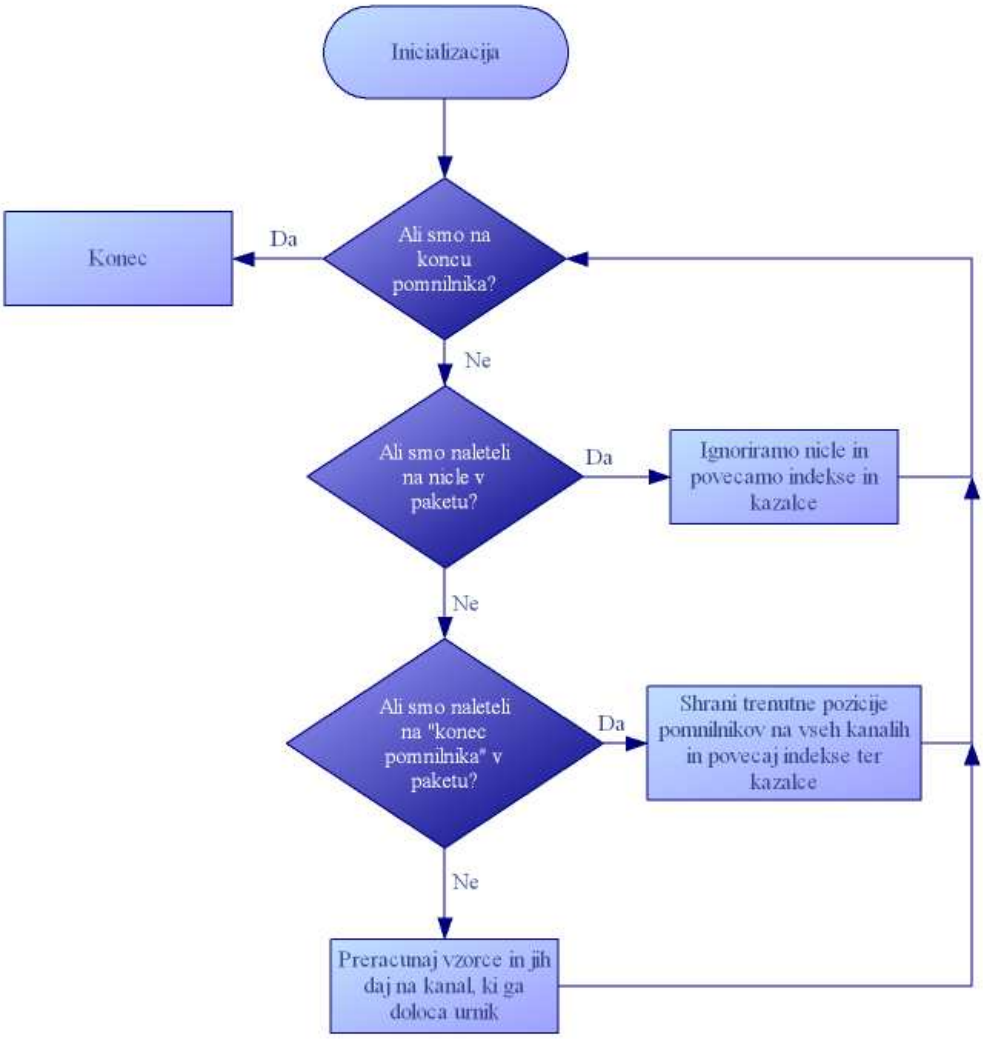
```
for InputIdx := 0 to InputList.Count - 1 do
begin
  CurrChannel := TEngineChannel(InputList.Items[InputIdx]);
  SchIdx := FindFirstEmpty;
  while SchIdx < SchLength do
  begin
    Schedule[SchIdx] := CurrChannel;
    Inc(SchIdx, Round(1 / CurrChannel.Priority));
  end;
end;
```

3.1.2.3.3 Vpis podatkov po kanalih

Z urnikom pravilno vpisujemo podatke v glavni pomnilnik vsakega izmed kanalov. Kanali tipa TEngineChannel vsebujejo kazalec na rezervirane dele pomnilnika, ki so namenjeni končni uporabi in prikazovanju. Vsak kanal ima svoj del pomnilnika in vanj lahko vpisujemo vrednosti različnih tipov, kolikor nam dovoljuje obseg rezerviranega pomnilnika. Le-ta se rezervira ob konstrukciji posameznega kanala. Poleg tega vsak kanal nosi informacijo oz. lokacijo zadnjega vpisanega podatka. Ko zadnji vpisani podatek prekorači celotno velikost rezerviranega pomnilnika, se kazalec vrne na začetek. Govorimo o krožnem pomnilniku. Slika 6 prikazuje vpis vzorcev v kanale.

Predstavimo še zadnji del obdelave podatkov, in sicer urejanje po kanalih. Ker se držimo pravila, da z enim preletom pomnilnika surovih podatkov iz USB vodila opravimo čimveč operacij, je postopek sledeč (koda je priložena):

1. pri preletu naenkrat preberemo 3 zloge,
2. preverimo, če smo prebrali koristne podatke in ne ničel,
3. preverimo, če naprava pred pošiljanjem lokalno shranjuje podatke v lastni pomnilnik (angl. Buffering mode) ali so podatki v direktnem toku,
4. prebrane zloge pretvorimo in jih s pomočjo urnika vpišemo v pomnilnik kanalov.



Slika 6: Diagram dekodiranja in obdelovanja pomnilnika vzorcev

Z zanko se sprehodimo čez prejeti pomnilnik in z enim preletom opravimo čim več operacij. Ob inicializaciji se nastavijo števci in kazalci, nato se zanka prične. Beremo po tri zloge naenkrat, s čimer dobimo dva vzorca. Najprej preverimo, če se nahajamo v delu pomnilnika, kjer so ničle (angl. Zero padding) in v tem primeru povečamo števce in premaknemo kazalce na podatke za ustrezno vrednost.

V primeru, da smo na mestu s podatki, nadaljujemo dalje in preverimo, če smo naleteli na podatke, ki signalizirajo konec pomnilnika v napravi (naprava ima omejeno velikost pomnilnika). To nam sporočijo trije zaporedni zlogi, ki imajo vrednost 255. Če najdemo tako trojico, potem zabeležimo zlom pomnilnika naprave (poglavje 3.1.2.3.4).

Vsak kanal v svoji definiciji vsebuje seznam indeksov zadnjih vpisanih podatkov, ko se na napravi zlomi pomnilnik. Ko naletimo na tako trojico s proceduro *AddBufferBreak* vsem uporabljenim kanalom v seznam prelomov vpišemo trenutni indeks zadnjega vpisanega podatka. Prikazovalne komponente pri prikazovanju podatkov od vsakega kanala uporabijo njegov seznam prelomov. Za tisti kanal za katerega prikazujemo podatke vzamemo iz seznama prelomov dva zadnja indeksa in prikažemo podatke med njima.

Če do zloma pomnilnika ne pride, se postopek nadaljuje naprej in iz treh zlogov izračunamo dva vzorca in s pomočjo indeksa urnikov pridobimo kanala, ki po urniku sledita ter vsakemu vpišemo vzorec. Na koncu še povečamo indekse za urnike in podatke ter kazalce.

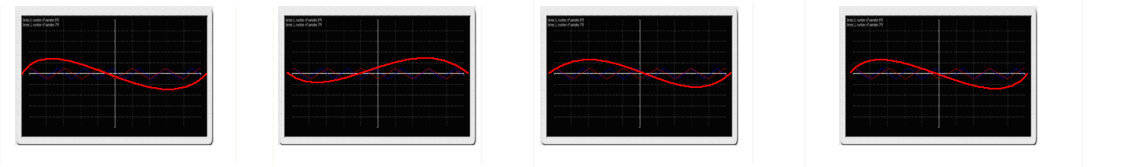
3.1.2.3.4 Prelom pomnilnika na napravi

Naprava zajema podatke z izbrano frekvenco vzorčenja. Za hranjenje svežih podatkov ima na voljo notranji pomnilnik. Količina le-tega je odvisna od števila vklopljenih funkcij (analogni osciloskop, logični analizator, itd). Pri višjih frekvencah vzorčenja se ta pomnilnik hitreje polni kot pri nižjih, kjer zajamemo manj vzorcev v časovni enoti. Prenos podatkov po vodilu USB ima omejeno prepustnost, kar vpliva na hitrost praznjenja pomnilnika v napravi. Če prenašamo podatke s hitrostjo, ki je manjša od hitrosti polnjenja pomnilnika v napravi, se zgodi prelom pomnilnika. Takrat naprava prekine z vzorčenjem in počaka da prenesemo dovolj podatkov, da lahko zopet prične z vzorčenjem oz. je pomnilnik zopet na voljo. To se imenuje način pomnjenja (angl. Buffer mode).

Pri nižjih frekvencah vzorčenja, kjer je hitrost prenosa hitrejša od pomnjenja podatkov v pomnilniku, ne pride do zloma in lahko prikazujemo fiksno število podatkov. Tak način imenujemo cevovodni način (angl. Pipe mode), ker podatki prehajajo iz naprave direktno na računalnik.

Slika 7 prikazuje koncept prikazovanja signala pri prelomih pomnilnika v napravi. Prelom pomeni prekinitev vzorčenja, zato nam podatki ob času preloma oz.

nedejavnosti naprave manjkajo. Pri prikazu si zato zapomnimo, kdaj naprava javi prelom pomnilnika in prikažemo signal med prelomi, saj je to edino, kar imamo na voljo. Na sliki 7 so prelomi označen s črko P. Prikazovalniki simbolično ponazarjajo prikaz signala med prelomi.



Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Podatek	1758	270	2546	2853	1051	2723	2178	1705	2389	2596	1376	377	1218	3397	2275	1123	2629	3268	1769	1981	3272	378	1670	728
Prelom						P					P						P							P

Slika 7: Prikazovanje podatkov med prelomi

3.1.3 Grafični vmesnik

Preko grafičnega vmesnika uporabnik posredno določa načine delovanja naprave, hkrati pa vidi tudi njeno trenutno stanje. Grafični vmesnik je stvar oblikovanja in ideje. Glede na to katere funkcije naj bodo dostopne uporabniku, izberemo kontrole, preko katerih uporabnik napravo nastavlja.

Pomembna je preglednost in enostavnost uporabe grafičnega vmesnika. Bolje je vgraditi en gumb manj in razviti kompleksnost v kodi, kot gumb več in odločitvi prepuščen uporabnik. Več kot je gumbov, več je možnih kombinacij nastavitev in več je možnosti za zmedo uporabnikov.

Našemu vtičniku smo nadeli grafično podobo kot pri pravem osciloskopu. Izgled nekaterih gumbov je skoraj tak kot pri pravem osciloskopu. S tem želimo uporabniku predstaviti grafični vmesnik tako, kot bi sedel pred pravo napravo. Če je uporabnik tako napravo že uporabljal mu bo zaradi tega tudi naš grafični vmesnik takoj jasen, novi uporabniki pa tudi ne bodo rabili več kot 5 min, da se z metodo preizkušanja spoznajo z njim. Pregovor slika pove več kot tisoč besed vsekakor nekaj velja, zato je prvi vizualni vtis pri uporabnikih zelo pomemben.

Slika 8 predstavlja grafični vmesnik našega programskega vmesnika s podporo v angleškem jeziku.

Slika 8: Grafični vmesnik v programskem vmesniku

Grafični vmesniki programskih dodatkov se v gostujoči aplikaciji naložijo na točno določen del. Slika 9 prikazuje celotno aplikacijo z vtičnikom Octopus.

Slika 9: Prikaz združitve vtičnika in gostujoče aplikacije

4 Zaključek

S programskim vmesnikom Octopus smo obstoječi napravi dodali možnost priklopa in uporabe z obstoječo programsko opremo. Zaradi kompleksnosti naprave je vtičnik zgrajen iz različnih programskih plasti. Najnižja plast skrbi za uspešno komunikacijo naprave in vtičnika, sledi implementacija ukazov, ki jih naprava sprejme, in odzivi na zahteve uporabnika. Sledi branje in urejanje večje količine podatkov in nazadnje še grafični vmesnik, preko katerega napravi sporočamo način delovanja in uporabniku prikazujemo trenutno stanje naprave.

Prednost vtičnika je, da se ga pri nadaljnjem razvoju in podpori novih naprav lahko brez težav uporabi kot predlogo, saj je struktura programske kode temu primerna in namenjena. S tem pri podpori novim napravam pridobimo precej dragocenega razvijalskega časa, ki bi ga drugače porabili za razvoj od začetka. Vsekakor je potrebno omeniti prednost pri razvoju, ki jo vtičnik omogoča, in sicer, da se vtičnik in gostujoča naprava brez težav lahko razvijata ločeno (pomembna je le enaka verzija SDK). Vtičniku se brez težav neodvisno od gostujoče aplikacije dodaja funkcije in posodobitve, tako uporabnikom npr. ni potrebno s spleta prenesti celotne aplikacije ampak zgolj vtičnik.

Viri

[1] Jan Axelson, USB Complete, Third Edition, Lakeview Research LLC, Madison, WI 53704

[2] Dynamic-Link libraries, http://msdn.microsoft.com/en-us/library/ms682589(VS.85).aspx, dostopnost preverjena 18.6.2009

[3] Don Anderson, Universal serial bus system architecture, Mindshare Inc, 1997

[4] Jeffrey Richter, Christophe Nassare, Windows via C/C++, Fifth edition, Microsoft press, 2006

[5] Jeffrey Richter, Programming applications for Microsoft Windows, Microsoft press, 1999

[6] Windows API reference, http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx, dostopnost preverjena 18.6.2009

[7] WinUSB, http://msdn.microsoft.com/en-us/library/aa476426.aspx, nazadnje dostopno 18.6.2009

[8] Critical Section Object, http://msdn.microsoft.com/en-us/library/ms682530(VS.85).aspx, dostopnost preverjena 18.6.2009

[9] Thread Synchronization: Critical sections, http://www.delphicorner.f9.co.uk/articles/op4.htm, dostopnost preverjena 18.6.2009

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Boštjana Murovca, univ. dipl. inž. el. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Aljoša Skočir