SULAUTETUT JÄRJESTELMÄT USB-POHJAISISSA SOVELLUKSISSA

Jori Loikkanen

Opinnäytetyö
Toukokuu 2015
Tietotekniikan KO
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

JORI LOIKKANEN:
Sulautetut järjestelmät USB-pohjaisissa sovelluksissa

Opinnäytetyö 78 sivua, joista liitteitä 36 sivua
Toukokuu 2015

Työn tavoitteeksi asetettiin USB-teknologian ymmärtäminen, toimivan USB-laitteen suunnitteleminen ja rakentaminen ja laiteohjelmistoon tutustuminen. Laite päätettiin rakentaa PIC18F2550-mikrokontrollerin ympärille, johtuen sen integroidusta USB-piiristöstä. Lisäksi pyrittiin toteuttamaan yksinkertainen isäntäohjelma, joka yhdistettynä suunniteltuun USB-laitteeseen mahdollistaisi käyttöjärjestelmän toimintojen ohjaamisen USB-laitteen painikkeita painamalla. Työn tavoitteiden saavuttamiseksi luettiin USB-dokumentteja ja –kirjallisuutta ja tutustuttiin valmiisiin USB-projekteihin. Laiteohjelmiston ja isäntäohjelman pohjana käytettiin valmiita projekteja, jotta työn määrä ei kasvaisi liian suureksi. Molempia muokattiin lisäominaisuuksien lisäämiseksi ja työn tavoitteiden saavuttamiseksi

USB-väylän toimintaa onnistuttiin selvittämään sekä elektroniikan että ohjelmoinnin näkökulmasta. USB-standardin asettamat vaatimukset määräävät muun muuassa käytettävien ylösvetovastusten ja kytkentäkondensaattoreiden suuruuden ja esimerkiksi kuinka pitkä USB-kaapeli voi olla ja näitä tietoja käytettiin hyväksi laitetta suunniteltaessa. Erilaisten pakettityyppien toimintaa tutkittiin, ja ylösvetovastusten merkitys USB-laitteen nopeuden valinnassa opittiin ymmärtämään. Laite saatiin suunniteltua ja rakennettua koekytkentäalustaan ja laiteohjelmisto saatiin ladattua laitteen mikrokontrollerille. Laiteohjelmiston raporttikuvailutietuetta muokattiin työhön sopivaksi. Isäntäohjelmaan lisättiin ominaisuus, joilla useamman painikkeen käyttäminen laitteessa ei tuottaisi ongelmia. Lisäksi ohjelmoitiin funktiot kolmelle painikkeelle, jolloin äänenvoimakkuuden ohjaaminen käyttöjärjestelmässä onnistui USB-laitteen painikkeita painamalla.

USB-väylän ja -ohjelmoinnin monimutkaisuudesta johtuen ei ollut mahdollista tuottaa niin paljon alkuperäistä koodia, kuin oli tarkoitus. Lisäksi ohjauskonsoli jäi todella yksinkertaiseksi HID-laitteeksi, jossa oli vain neljä painiketta ja ohjattava LED. Laiteohjelmistoon ja isäntäohjelmaan tehdyt lisäykset kuitenkin toimivat hyvin ja kyseisellä laitteella voidaan jo nyt tehdä mahdollisesti monia asioita Windows-käyttöjärjestelmässä. Tulevaisuudessa on mahdollista lisätä LCD-näyttö laitteeseen ja yrittää käyttää mikrokontrollerin A/D-muunninta, jotta laitteeseen saadaan lisää ominaisuuksia. Myös laiteohjelmistoa on mahdollista kehittää vielä pitemmälle.

Asiasanat: usb, mikrokontrolleri, sulautetut järjestelmät, PIC, C#

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Embedded Systems

JORI LOIKKANEN:
Embedded systems in USB-based solutions

Bachelor's thesis 78 pages, appendices 36 pages
May 2015

The purpose of this thesis was to gain an understanding of USB-technology, the designing and building of a working USB-device and to get introduced to USB firmware. A decision was made to build the device around a PIC18F2550 microcontroller, largely due to the integrated USB-circuitry. Also, an attempt was made to create a simple host program, which connected with the designed USB device would enable the programmatical controlling of the functionality of the operating system by using the buttons on the USB device. Knowledge to enable these plans was acquired from several sources including USB documentation, device documentation and various finished USB projects. The firmware and host program were based on a finished project, so the amount of work would not prove too daunting. Both were modified to add features and to achieve wanted functionality.

The inner workings of the USB bus were researched both from an electronic and a programming point of view. The requirements set by the USB standard and how they affect the values of pull-up resistors and capacitors were used as a basis for building the device. Different packet types used in USB communication were also researched and the significance of pull-up resistors in choosing the speed of a USB device was understood. The device was successfully designed and built on a breadboard, which enabled the uploading of the firmware onto the device microcontroller. The report descriptor of the firmware was modified to meet the goals of the thesis. A feature was added to the host program which prevented errors when using multiple buttons on the console. Functions were created for three buttons so that the volume controls of the host could be changed by pressing the mentioned buttons on the USB device.

Due to the complexity of the USB bus and the programming involved it was not possible to produce as much original code as originally planned. In addition, the USB device ended up a really simple HID control console with only four buttons and a controllable LED. However, the additions made to both the firmware and the host program worked as intended and created a possibility to do a lot of things already in a Windows operating system. In the future an LCD screen might be added to the device in addition to trying to add features by using one of the integrated A/D converters of the microcontroller. Also, the firmware has potential to be developed further.

Key words: usb, microcontroller, embedded systems, PIC, C#

SISÄLLYS

LYHENTEET JA TERMIT

API	Application Programming Interface, ohjelmointirajapinta
C#	C-sharp, Microsoftin luoma ohjelmointikieli
Descriptor	Kuvailutietue
DLL	Dynamic Link Library, Windowsissa käytettävä funktiokirjastotyyppi
Endpoint	Päätepiste
Enumeration	Luettelu
Firmware	Laiteohjelmisto
GUID	Globally Unique Identifier, tietokoneohjelmissa käytettävä tunniste
HID	Human Interface Device, ihmisen ohjattavalla käyttöliittymällä varustettu laite
HID-report	HID-raportti, käytetään HID-laitteen datan siirtämiseen isännälle ja isännältä
Host	Isäntä, USB-väylässä usein tietokone
ICSP	In-Circuit Serial Programming, piirin sisäinen sarjamuotoinen ohjelmointi
PDA	Personal Digital Assistant, kämmentietokone
PIC	Peripheral Interface Controller, Mirochipin valmistama mikrokontrollerityyppi
PID	Product ID, tuotetunnus
SIE	Serial Interface Engine, sarjaliitäntämoottori
Transceiver	Lähetin/vastaanotin
Usage	Käyttötunnus HID-laitteelle
Usage Page	Käyttösivu HID-laitteelle
USB	Universal Serial Bus, yleinen sarjaväyläliitäntä
VID	Vendor ID, toimittajatunnus
WFF	Windows Workflow Foundation, Microsoftin ohjelmointityökalu

1 JOHDANTO

USB on sarjaväyläliitäntätyyppi, joka alunperin luotiin helpottamaan lisälaitteiden liittämistä tietokoneisiin. Nykyään USB-liitäntä on muodostunut yleisimmäksi liitäntätyypiksi laitteiden ja tietokoneiden välille. Lisäksi monet matkapuhelimien laturit käyttävät USB-liitäntää hyväkseen. Se on lähes kokonaan korvannut perinteiset sarja- ja rinnakkaisväyläliitännät, joita perinteisesti käytettiin muun muuassa tulostimien ja ohjainten liittämiseen.

Koska melkein jokaisessa modernissa tietokoneessa on vähintään muutama USB-portti, on se hyvä lähtökohta omavalmisteiselle lisälaitteelle. Monesti tietokoneiden ergonomiaa ja käyttömukavuutta yritetään parantaa erilaisilla ohjelmilla tai laitteilla, mutta aina markkinoilta ei löydy käyttäjän tarpeisiin tai budjettiin sopivaa laitetta. Yksi ratkaisu on suunnitella ja ohjelmoida oma USB-lisälaite, joka on edullisempi vaihtoehto, ja jonka voi suunnitella täysin käyttökohteeseen sopivaksi.

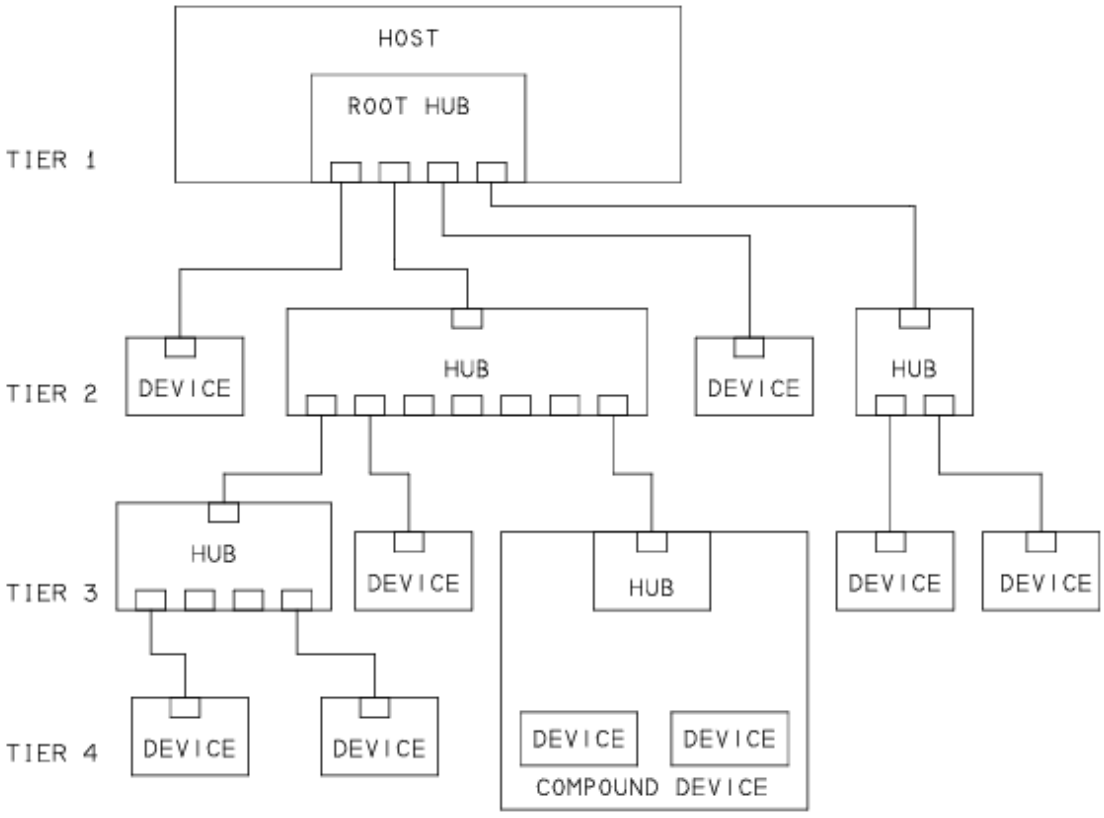
Tässä työssä esitellään USB-teknologia laite- ja ohjelmointinäkökulmasta ja samalla suunnitellaan ja rakennetaan yksinkertainen ohjauskonsoli, joka voidaan liittää mihin tahansa tietokoneeseen, jossa on Windows-käyttöjärjestelmä. Laitteelle asennetaan valmis laiteohjelmisto, jota muokataan opinnäytetyön tavoitteiden saavuttamiseksi. Lisäksi isännälle eli tietokoneelle kirjoitetaan yksinkertainen ohjelma, joka mahdollistaa tietokoneen ja USB-laitteen välisen kommunikoinnin.

USB-laite rakennetaan PIC18F2550-mikrokontrollerin ympärille, jossa on integroituna tarvittava piiristö USB-väylän käyttöä varten. Kytkentä rakennetaan koekytkentäalustalle ja se koostuu mikrokontrollerin lisäksi Ledistä, painikkeista ja vaadittavista kondensaattoreista ja vastuksista. Isäntäohjelmiston runkona käytetään myös valmista ohjelmaa, jossa on yksinkertainen graafinen käyttöliittymä ja joka on kirjoitettu C#-kielellä. Lopullisena tavoitteena on mahdollisuus lähettää vapaamuotoista dataa laitteelta ja laitteelle ja lisäksi muokata isäntäsovelluksesta sellainen, että isännän eli tietokoneen toimintaa voidaan ohjata USB-laitteen painikkeilla.

2 USB-STANDARDI

USB on 90-luvulla kehitetty ja julkaistu sarjaväyläliitäntäteknologia, jonka tarkoituksena oli korvata vanhat sarja- ja rinnakkaisliitäntätyypit ja helpottaa muun muuassa lisälaitteiden kuten hiirten ja näppäimistöjen kytkemistä tietokoneisiin. USB-portit toimivat sekä datan että lisälaitteen käyttövirran lähteinä (Boston Globe, 1999). Väylä toimii host-device-periaatteella (isäntä-laite), jossa perinteisesti tietokone toimii isäntänä. Nykyään isäntänä voi toimia myös esimerkiksi pelikonsoli tai matkapuhelin On-The-Go-spesifikaation ansiosta (USB in a nutshell 2010).

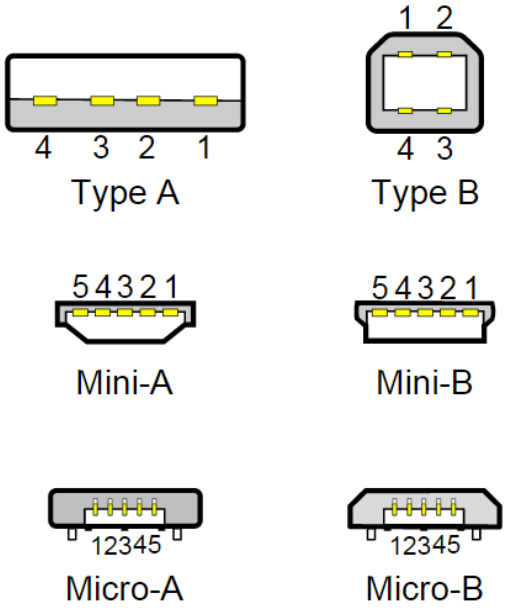
USB-väylässä käytetään monitasoista tähtitopologiaa, jossa jokaisen tähden keskipisteenä toimii isäntä tai USB-toistin. Toistimella on yksi ylävirtaliitäntä isännän kanssa kommunikointia varten ja kaksi tai useampi liitäntää laitteita varten. USB-väylää käytettäessä fyysinen kytkentätaso on loogisesti erillään ohjelmallisesta kytkennästä, eli ohjelmoidessa isännän ohjelmaa tai laiteohjelmaa ei tarvitse välittää, onko yhteys luotu suoraan isännästä laitteeseen vai yhden tai usean toistimen kautta (Axelson, 2009, 15).



KUVA 1. Esimerkki USB-topologiasta (Axelson, 2009, 16)

2.1 Liitäntä

USB-liittimet suunniteltiin alun perin siten, että samassa kaapelissa virtaa syöttävä liitin on A-tyyppiä ja virtaa käyttävä liitin on B-tyyppiä. Tällä varmistettiin alkuaikoina se, että kahta tietokonetta (isäntää) ei pystytty liittämään toisiinsa. Datapinnit ovat syvemmällä liittimessä kuin tehopinnit, jotta laite ehtii kytkeytyä päälle ennen kuin se yrittää luoda datayhteyden. Version 2.0 myötä lisättiin miniliitin, jonka leveys ja paksuus olivat noin puolet alkuperäisestä A-liittimestä (USB implementers forum 2000, 99, 101). Yleiseen käyttöön vakiintui mini-B-tyyppinen liitin, jota käytettiin paljon pienemmissä laitteissa kuten digitaalisissa kameroissa, varhaisissa älypuhelimissa ja PDA-laitteissa. Vuonna 2007 esiteltiin nykyisistä liitännöistä uusin ja ohuin micro-malli. Se on suunniteltu kestämään 10000 kytkemis- ja irrottamissykliä, huomattavasti enemmän kuin sitä edeltävät liitintyypit (USB implementers forum 2007, 6). Micro-liitin korvasi melkein kokonaan muut liitintyypit älypuhelimissa sen jälkeen, kun useat suuret matkapuhelinvalmistajat tekivät siitä yleisen liitinpään älypuhelimien latureihin (Engadget 2010).



KUVA 2. USB-liittimien eri tyypit (USB Implementers Forum, 2007)

2.1.1 Kaapelit

USB-liittimissä on joko neljä tai viisi pinniä, riippuen onko kyseessä perinteinen vai mini- tai micro-liitin (kuva 2). Kaksi hoitavat käyttöjännitteen syöttämisen laitteelle ja toiset kaksi datan kuljettamisen isännän ja laitteen välillä. Taulukosta 1 nähdään mitä mitäkin kuvassa 2 oleva pinni vastaa. Taulukosta nähdään myös, minkä värinen kaapeli vastaa kutakin pinniä perinteisesti. Mini- ja micro-liittimissä neljäs pinni on ID-pinni, jonka avulla tunnistetaan, onko liitin kytketty isäntään vai laitteeseen, jolloin viides pinni on maa. Muuten pinnien järjestys vastaa perinteistä liitintä. USB-standardin mukaan kaapeli saa olla enintään 5 metriä pitkä high-speed- ja 3 metriä low-speed-sovelluksissa (Cables Plus, USB cable length limitations).

TAULUKKO 1. USB-liittimien kaapelointi

Pin	Nimi	Väri	Kuvaus
1	Vbus	Punainen	+5V DC
2	D-	Valkoinen	Data-
3	D+	Vihreä	Data+
4	GND	Musta	Maa

2.2 Versiot

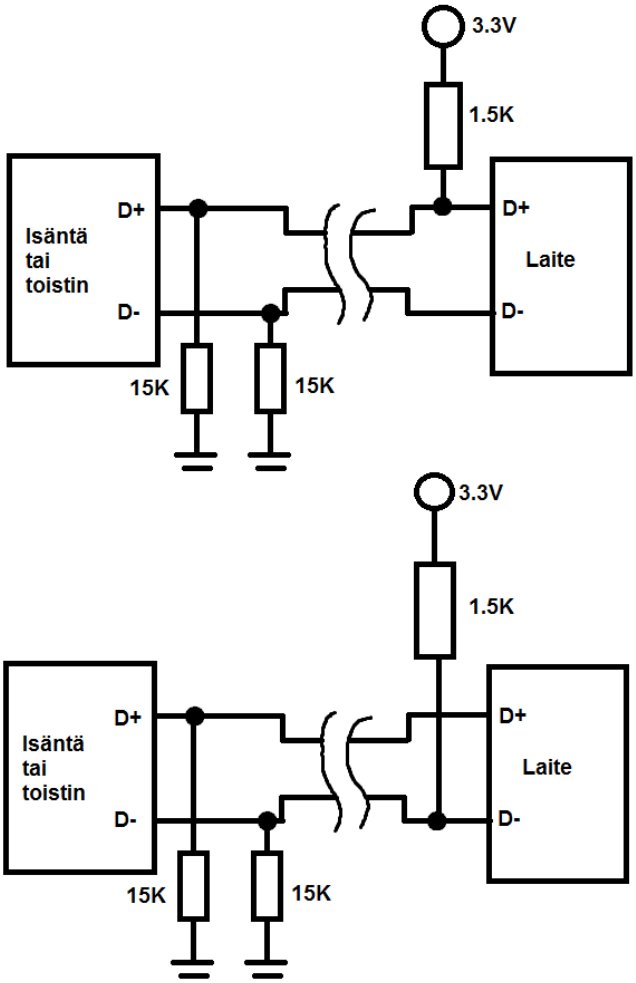
USB-standardista on julkaistu tähän mennessä viisi versiota: 1.0, 1.1, 2.0, 3.0 ja 3.1. Koska työssä käytettiin ainoastaan versioita 1.x ja 2.0, keskitytään niiden esittelemiseen. Ensimmäinen versio eli 1.0 julkaistiin jo vuonna 1996, mutta ensimmäinen yleiseen käyttöön vakiintunut versio oli vuonna 1998 julkaistu 1.1-versio. 1.x-versioissa esiteltiin 1,5 Mbit/s low-speed- ja 12 Mbit/s full-speed-nopeudet. Low-speed-nopeus oli tarkoitettu pientä kaistaa vaativia lisälaitteita, kuten näppäimistöjä ja hiiriä varten ja lisäksi oli vähemmän herkkä sähkömagneettisille häiriöille, mikä teki siitä halvemman vaihtoehdon. Full-speed-nopeus oli tarkoitettu vaativimpiin laitteisiin kuten tulostimiin ja levyasemiin.

2.3 Elektroniikka

USB-laitteissa käyttöjännite täytyy pysyä 5 ± 0.25 V arvossa ja laitteen käyttämä enimmäisvirta on rajoitettu 500 milliampeeriin. Laitteen tarvitsema virta ilmoitetaan isännälle 2 milliampeerin yksiköissä (USB Implementers Forum 2000, 178, 266). USB:n yksi parhaimpia puolia onkin mahdollisuus käyttöjännitteen- ja virran ottamisessa suoraan USB-portista, jolloin ulkoisia virtalähteitä ei välttämättä tarvita. Käyttövirta on jaettu viiteen yksikkökuormaan, joista laite voi käyttää vain yhtä käynnistyksen yhteydessä. Konfiguroinnin yhteydessä ilmoitetaan isännälle vaadittu virtamäärä (USB in a nutshell 2010). Laitteistollisesti on myös huomattava, että USB-laitetta irrotettaessa kaapeliin saattaa indusoitua suuri jännite, jonka takia Vbus:n ja maan välille on hyvä asentaa kondensaattori. Standardissa suositellaan vähintään 1 µF:n suuruista kondensaattoria (USB Implementers Forum 2000, 179).

2.3.1 Nopeuden valinta

Ylösvetovastuksen asetuksesta riippuen laite siirtyy joko low-speed- tai full-speed-tilaan. Full-speed-tila saadaan aikaan vetämällä dataparin plus-puolen jännite ylös ja low-speed-tila vastaavasti vetämällä miinus-puolen jännite ylös (kuva 3). 3.3 voltin jännite saadaan usein piirin sisältämästä regulaattorista. Joissain piireissä tai laitteissa vastukset saattavat olla valmiiksi asennettuina ja joihinkin sovelluksiin ne täytyy asentaa. USB-standardissa on määritelty laitteen ylösvetovastuksen kooksi 1.5 kΩ (USB Implementers Forum 2000, 123). Jos laite haluaa siirtyä high-speed-tilaan, joutuu se ensin kytkemään itsensä full-speed-tilassa, jonka jälkeen tehdään high-speed-kutsu resetoinnin yhteydessä, jolloin high-speed-tilaan siirrytään, jos sitä tuetaan (USB in a nutshell 2010). Kun high-speed-tilaan on siirrytty, on ylösvetovastukset poistettava aktiivisesta piiristä.



KUVA 3. Full-speed- ja low-speed-tilan valitseminen ylösvetovastuksilla

3 USB-KOMMUNIKOINTI

Isäntä aloittaa aina kommunikointitapahtuman. Jokainen tapahtuma koostuu paketeista ja jokainen paketti koostuu kentistä. Jokainen tapahtuma sisältää ainakin token- ja handshake-paketin, joiden lisäksi transaktioon voidaan lisätä myös datapaketti. Monissa nykyisissä USB-väylällä varustetuissa piireissä on USB-teknologian alhaisimmat kerrokset hoidettu jo laitteistotasolla, jolloin kehittäjä voi keskittyä ylempiin kerroksiin ilman, että joutuu suunnittelemaan pakettien pakkaamista ja purkamista. Paketit lähetetään aina aloittaen vähiten merkitsevästä bitistä (USB in a nutshell 2010).

3.1 USB-pakettien rakenne

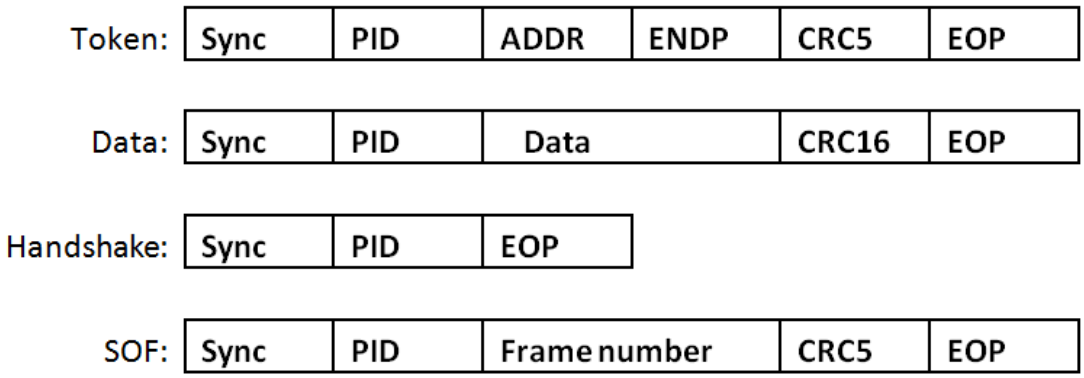
3.1.1 Kentät

Jokainen paketti alkaa sync-kentällä. Kentällä synkronoidaan lähetin ja vastaanotin datan siirtämistä varten. Low-speed- ja full-speed-sovelluksissa sync-kentän pituus on 8 bittiä, high-speed sovelluksissa 32 bittiä (FTDI 2009, 6). Sync-kenttää seuraa aina PID-kenttä, joka tässä tapauksessa ei viittaa tuotetunnisteeseen vaan pakettitunnisteeseen (Packet ID). PID-kenttä on neljäbittinen, mutta tarkastuksen vuoksi se lähetetään 8-bittisenä, joissa neljä viimeistä bittiä ovat neljän ensimmäisen komplementteja. PID-kentällä kerrotaan, mitä paketilla yritetään tehdä, esimerkiksi token-pakettia käsitellessä, yritetäänkö kirjoittaa vai lukea tietoa.

Osoitekentällä (ADDR) määritetään laite, jolle paketti on tarkoitettu. Osoitekentän pituus on 7 bittiä, ja koska nolla-arvo on aina varattu osoitettaan odottaville laitteille, saadaan 127 mahdollista osoitetta. Tämä tarkoittaa käytännössä sitä, että yhdessä USB-väylässä voi olla enimmillään 127 laitetta kerrallaan (USB in a nutshell 2010). ENDP-kenttä kertoo, mihin päätepisteeseen (endpoint) transaktion täytyy päätyä. Päätepisteitä on high-speed- ja full-speed-laitteissa 16, low-speed-laitteissa 4 (USB in a nutshell 2010). CRC-kentät ovat virheentarkastusta varten, jotka ovat token-paketeilla 5-bittisiä ja datapaketeilla 16-bittisiä. Jokaisen paketin lopussa sijaitsee EOP-kenttä, jolla merkitään paketin päättyminen.

3.1.2 Paketit

USB-transaktioissa käytetään neljää erilaista pakettityyppiä kommunikointiin. Jokainen lähetys alkaa token-paketilla, joka sisältää tiedot siitä, mille laitteelle lähetetään ja minkälaista transaktiota ollaan suorittamassa. Token-paketteja on kolmea eri tyyppiä: IN, OUT ja SETUP. IN-tyyppisillä paketeilla yritetään lukea tietoa laitteelta, OUT-tyyppisillä kirjoittaa tietoa laitteelle ja SETUP-tyyppiset paketit on varattu ohjaustoimintoja varten. Token-paketti kertoo myös, mitä päätepistettä transaktiossa halutaan käyttää. Kuvasta 4 nähdään sekä token- että muiden pakettien rakenne.



KUVA 4. Pakettien rakenteet

Datapaketteja on neljä erilaista. Jokaiselle USB-nopeudelle on määritelty Data0- ja Data1-tyypit, jotka voivat molemmat enimmillään sisältää 1024 tavua dataa. High-speed-tilassa on määritelty myös Data2- ja MDATA-tyypit. Low-speed-laitteet eivät voi lähettää kahdeksaa tavua enempää kerrallaan, kun taas full-speed- ja high-speed-laitteet voivat lähettää 1023 ja 1024 tavua kerrallaan. Datakentän bittilukumäärä on aina kahdeksalla jaollinen.

Handshake-paketit eivät sisällä muuta tietoa kuin PID-koodinsa. Handshake-paketeilla ilmoitetaan, mitä paketin siirron yhteydessä tapahtui. ACK-koodilla ilmoitetaan onnistuneesta vastaanotosta, NAK-koodilla laite voi raportoida isännälle, jos se ei voi lähettää tai vastaanottaa dataa. STALL-koodi tarkoittaa, että päätepiste on seisahtunut ja isännän täytyy suorittaa korjaustoimenpide, ennen kuin kommunikaatiota kyseiseen päätepisteeseen voidaan jatkaa. NYET-koodia käytetään ainoastaan high-speed-tilassa, jos vastaanottimelta ei saada vastausta (USB made simple). SOF-pakettia käytetään

ainoastaan full-speed- ja high-speed-tiloissa ilmaisemaan kehyksen alkua, jota käytetään datalähetysten aikataulujen järjestämiseen.

3.2 Päätepisteet

USB-väylässä kaikki laitteelle saapuva ja sieltä lähtevä liikenne kulkee päätepisteiden kautta. Käytännössä päätepisteitä voidaan ajatella puskureina, jotka varastoivat saapuvaa ja lähtevää dataa laitteen muistiin, kunnes on aika lukea data tai siirtää se eteenpäin. Päätepisteet ovat yksinomaan laitteen ominaisuus, vaikka isännällä on myös omat puskurinsa dataa varten (Axelson 2009). Kuten jo mainittu päätepisteen tunniste muodostuu full-speed- ja high-speed-laitteissa numeroista 0-15, ja ne voivat olla IN- tai OUT-tyyppisiä (lähetettävä ja saapuva data). Ohjauspäätepisteet muodostuvat IN- ja OUT-osoiteparista, jolloin ohjauspäätepisteen kautta voidaan sekä vastaanottaa että lähettää dataa. Jokaisessa laitteessa ohjauspäätepiste täytyy sijoittaa nollaosoitteeseen. Myös muilla päätepistenumeroilla voi olla sekä IN- että OUT-osoite.

3.2.1 Putket

Jotta päätepistettä voidaan käyttää isännän ja laitteen väliseen kommunikointiin, täytyy isännän ohjelmiston ja laitteen päätepisteen välille luoda looginen kanava, jota USB-väylissä kutsutaan putkeksi. Isännän ohjelmisto luo jokaiselle päätepisteelle putken, jonka kanssa se haluaa kommunikoida. Putket luodaan, kun laite kiinnitetään isännän USB-porttiin ja laite luetellaan. Luettelemalla isäntä saa laitteesta tarvittavat tiedot operointia ja kommunikointia varten. Putkia voidaan myös luoda tai poistaa luettelun jälkeen lähettämällä ohjauspäätepisteeseen uudet asetukset (Axelson 2009, 36). Tällaisen control-siirron lisäksi putkia käytetään bulk-, interrupt- ja isochronous-siirtoja varten. Jokainen siirtomuoto on luotu erilaisia tarpeita varten, kuten vaihtelevat datamäärät tai parempi virheensietokyky. Jokaista voidaan käyttää sekä IN- että OUT-tyyppisiin lähetyksiin.

3.2.2 Control-siirto

Control- eli ohjaussiirto on ainoa siirtomuoto, jonka funktiot on määritelty USB-spesifikaatiossa ja jonka täytyy olla tuettu kaikissa USB-laitteissa. Ohjaussiirrolla

mahdollistetaan tiedon kulku laitteelta isännälle, laitteen osoitteen asetus ja muiden asetusten määrittäminen, kuten tarvittava virta, lähetteiden koko tavuina jne. Ajureita muokkaamalla on myös mahdollista lähettää luokka- tai valmistajakohtaisia pyyntöjä ohjaussiirroilla (Axelson 2009, 36).

3.2.3 Bulk-siirto

Bulk-siirto on ensisijaisesti tarkoitettu sellaisen datan siirtoon, joka ei riipu ajoituksesta tai kiinteästä siirtonopeudesta. Useimmiten sitä käytetään esimerkiksi tulostimissa, skannereissa ja levyasemissa. Bulk-siirron nopeus riippuu paljon siitä, kuinka paljon muuta liikennettä väylällä on. Väylän ollessa tyhjä yhtä laitetta lukuun ottamatta voi bulk-siirto toimia suhteellisen nopeasti. Bulk-siirrot sisältävät virheenkorjausmahdollisuuden.

3.2.4 Interrupt-siirto

Suurin ero interrupt- eli keskeytyssiirrossa muihin siirtomuotoihin nähden on se, että sillä on taattu latenssi (siirtoyritysten välinen maksimiaika) kaikilla nopeuksilla. Low-speed-tilassa voidaan lähettää 8 tavua per 10 ms, full-speed-tilassa 64 tavua per ms ja high-speed-tilassa 3072 tavua per 125 µs (kolme 1024 tavun pakettia). Myös keskeytyssiirroissa on virheenkorjausmahdollisuus.

3.2.5 Isochronous-siirto

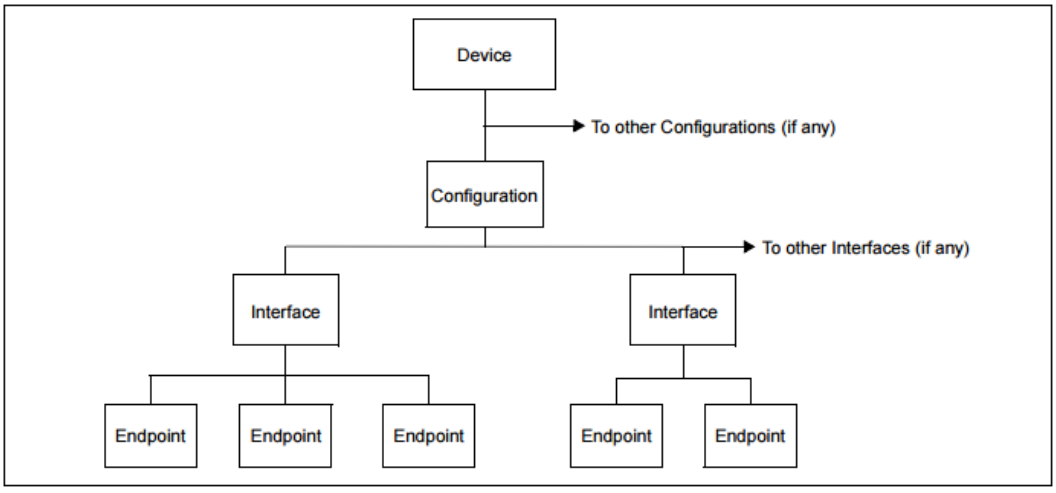
Isochronous eli isokrooninen siirto eroaa monella tavalla muista siirtotyypeistä. Se on ainoa siirtotyyppi, jolla on taattu siirtonopeus ja se on ainoa, joka ei sisällä virheenkorjausmahdollisuutta. Isokrooninen siirto on tarkoitettu suurten datamäärien nopeaan lähettämiseen, kuten audio- ja videosignaalien. Näissä käyttökohteissa myös ajoittaiset virheet eivät vaikuta suuresti lopputulokseen, toisin kuin muissa lähetysmuodoissa.

4 ENUMERATION

Ennen kuin USB-laite voi toimia isännän kanssa, sille pitää suorittaa enumeration-toimenpide (suomeksi luettelu). Luettelulla tarkoitetaan USB-laitteen laiteohjelmistossa sijaitsevien kuvailutietueiden (descriptors) lukemista, jotka sisältävät laitteen eri ominaisuudet ja vaatimukset. Lisäksi luetteloinnissa asetetaan laitteelle osoite ja valitaan ja ladataan ajuri laitteelle. Jotta luettelu onnistuisi, täytyy laiteohjelmistoa kirjoittaessa ottaa huomioon, että luetteluprosessit eivät tapahdu tietyssä järjestyksessä ja kuvailutietueiden rakenne täytyy tuntea, jotta ne voidaan määrittää oikein (Axelson 2009, 89-90).

4.1 USB descriptors

Kommunikaationäkökulmasta jokaisessa USB-laitteessa täytyy olla ainakin seuraavat kuvailutietueet: laite-, asetus-, rajapinta- ja päätepistekuvailutietueet. Kyseisiä kuvailutietueita voidaan ajatella hierarkkisesti, jossa laitekuvailutietue on ylimmällä ja päätepistekuvailutietue alimmalla tasolla. Laitteella voi olla monta konfiguraatiota, joista jokaisella on omat rajapintansa. Asiaa on helpompi ajatella videokameraesimerkillä: jos USB-porttiin liitettävässä kamerassa on video-ominaisuuden lisäksi mikrofoni, voisi yksi rajapinta olla videolle ja toinen äänelle. Aktiivisia konfiguraatioita voi olla vain yksi kerrallaan käytössä, mutta rajapintoja voi yhdessä konfiguraatiossa olla enemmän kuin yksi (USB in a nutshell 2010). Usein laitteella on kuitenkin olemassa vain yksi konfiguraatio. Kuvasta 5 voidaan nähdä esimerkki yhden USB-laitteen kuvailutietuehierarkiasta.



KUVA 5. Kuvailijahierarkia (Microchip 2004, 184)

4.1.1 Laitekuvailutietue

Laitekuvailutietue on ensimmäinen joka luetaan USB-laitteen kytkemisen yhteydessä. Laitekuvailutietue mahdollistaa tarvittavien lisätietojen hankkimisen laitteelta. Laitekuvailutietueella määritetään laitteen luokka, jos sitä ei tehdä rajapintakuvailutietueessa. Vaihtoehdot on määritelty USB-standardissa ja niitä ovat mm. USB-toistin, langaton ohjain ja kommunikaatiolaite (Axelson 2009, 101). Tällä tasolla ilmoitetaan myös laitteen Vendor ID (VID) ja Product ID (PID). VID saadaan USB Implementers Forumilta ja VID:n omistaja eli laitteen valmistaja valitsee PID-tunnisteen. PID-tunnistetta voidaan käyttää mm. Windows-käyttöjärjestelmässä määrittämään laitteelle ajuri.

4.1.2 Asetuskuvailutietue

Jokaisella laitteella on ainakin yksi asetuskuvailutietue, joka määrittää laitteen ominaisuudet ja kyvyt. Vaikka usein yhdet asetukset riittävät, voidaan useampaa käyttää esimerkiksi vaihdettaessa virtalähdettä USB-väylästä ulkoiseen tai päinvastoin. Asetuskuvailutietueet ovat voimassa yksi kerrallaan ja sisältävät tiedot eri rajapinnoista ja laitteen virtavaatimuksista. Virtavaatimukset ilmoitetaan 2 mA:n yksiköissä (Axelson 2009, 104).

4.1.3 Rajapintakuvailutietue

Rajapintakuvailutietue sisältää tietoa laitteen funktioista tai ominaisuuksista. Näihin tietoihin kuuluu muun muuassa luokka-, alaluokka- ja protokollatietoa ja rajapinnan käyttämien päätepisteiden lukumäärä. Erilaisiin luokkiin kuuluu muun muuassa audio-, HID-, tulostin- ja videolaitteet. Tässä huomataan, että laiteluokka voidaan laitetason sijasta ilmoittaa vasta rajapintatasolla. Jokaisella rajapinnalla voi olla useita konfiguraatioita ja jokaisella konfiguraatiolla voi olla useita rajapintoja käytössään samanaikaisesti. Rajapinta saattaa olla yhteydessä tiettyyn funktioon, tai molemmat voivat olla toisistaan riippumattomia (Microchip 2004, 184; Axelson 2004, 107). Jokaisella rajapinnalla on oman kuvailutietueensa lisäksi alikuvailutietueet ja mahdollisuus käyttää useampia asetuksia esimerkiksi silloin, kun halutaan käyttää isokroonista siirtoa.

4.1.4 Päätepistekuvailutietue

Jokaisella rajapintakuvailutietueessa määritetyllä päätepisteellä on myös oma kuvailutietueensa lukuun ottamatta päätepiste nollaa, joka on aina varattu ohjaustoimintoja varten. Päätepistekuvailutietueilla on aina sama määritelty koko lukuun ottamatta audio-luokkaa, jossa kuvailija sisältää kaksi ylimääräistä tavua tietoa. Jos päätepistekuvailutietueisiin täytyy lisätä tietoa, se tehdään alaisten kautta. Päätepistekuvailutietueet lähetetään vastaamalla konfiguraatiopyyntöön (Axelson 2009, 110). Päätepistekuvailutietueilla määritetään käytettävä siirtomuoto kyseisessä päätepisteessä, pakettien koko ja lähetysintervalli keskeytys- ja isokroonisille lähetyksille.

4.2 HID-laitteet

Yksi yleisimmistä ja valmiimmin tuetuista laitemuodoista on Human Interface Device- tai HID-luokkaa käyttävät laitteet. Yleisimmät HID-luokkaan kuuluvat laitteet ovat näppäimistöt, hiiret ja peliohjaimet. HID-laitteiden on tarkoitus toimia isännän kanssa sen verran nopeasti, että käyttäjä ei havaitse viivettä tehtyään jotain laitteella, kuten painiketta painaessaan. Lisäksi laitteelle ei tarvitse kirjoittaa omaa ajuria, vaan voidaan käyttää Windows-käyttöjärjestelmään integroituja HID-ajureita. Kaikki data HID-laitteissa kulkee raportteina, jotka ovat muodoltaan määriteltyjä rakenteita. HID-laitteita suunniteltaessa täytyy muistaa, että ne voivat käyttää vain ohjaus- ja keskeytyssiirtoja. Raporteille luodaan omat kuvailijansa, jotka kertovat raportin koon, ja mitä se saa sisältää ja missä muodossa. Yleisin tapa on käyttää keskeytyssiirtoa IN-päätepisteestä. Isännän täytyy kysellä dataa tarpeeksi usein, jotta viivettä ei muodostu. Useimmissa laitteissa low-speed-tila riittää normaaliin toimintaan ilman havaittavaa viivettä. Low-speed-tilan käyttö vähentää myös laitekustannuksia (Axelson 2009, 182).

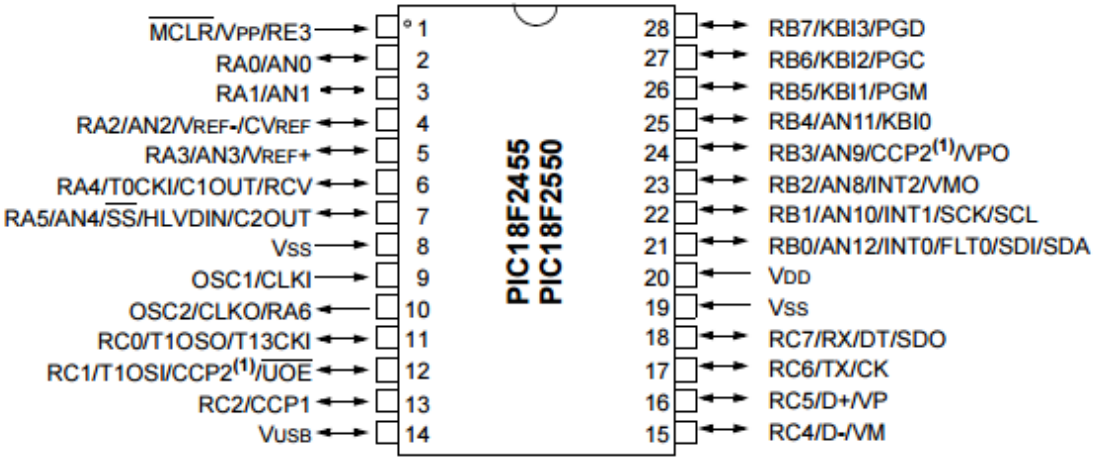
4.2.1 HID API

Isännän USB-ohjelmisto voi käyttää käyttöjärjestelmän ohjelmointirajapintoja (API) hyödykseen kommunikoidessaan HID-laitteen kanssa. API:lla ei kuitenkaan ole suoraa pääsyä esimerkiksi näppäimistöjen raportteihin, vaan sillä luetaan käyttöjärjestelmän lukemista raporteista dataa. Raportteja voidaan lukea ja kirjoittaa muun muuassa

funktioilla ReadFile ja WriteFile, ja joillekin HID-kohtaisille toiminnoille on omat funktionsa kuten HidD_SetFeature. Näiden funktioiden lisäksi voidaan käyttää muita rajapintoja kuten DirectInput tai raw input, kun reagoidaan käyttäjän syötteisiin (Introduction to HID concepts; Axelson 2009, 184).

5 PIC18F2550-MIKROKONTROLLERI

PIC18F2550 on Microchip-yrityksen valmistama mikrokontrolleri, jossa on integroitu USB-piiristö. PIC:issä on mahdollisuus käyttää pinnejä analogiseen ja digitaaliseen tuloon ja lähtöön. Tässä työssä PORTA-rekisterin pinnejä tullaan käyttämään painikkeiden painallusten havaitsemiseen ja yhtä pinniä käytetään lähtönä ledille, joka kertoo käyttäjälle, että laite on lueteltu ja konfiguroitu. Kuva 6 esittelee työssä käytetyn 2550-mallin pinnit.



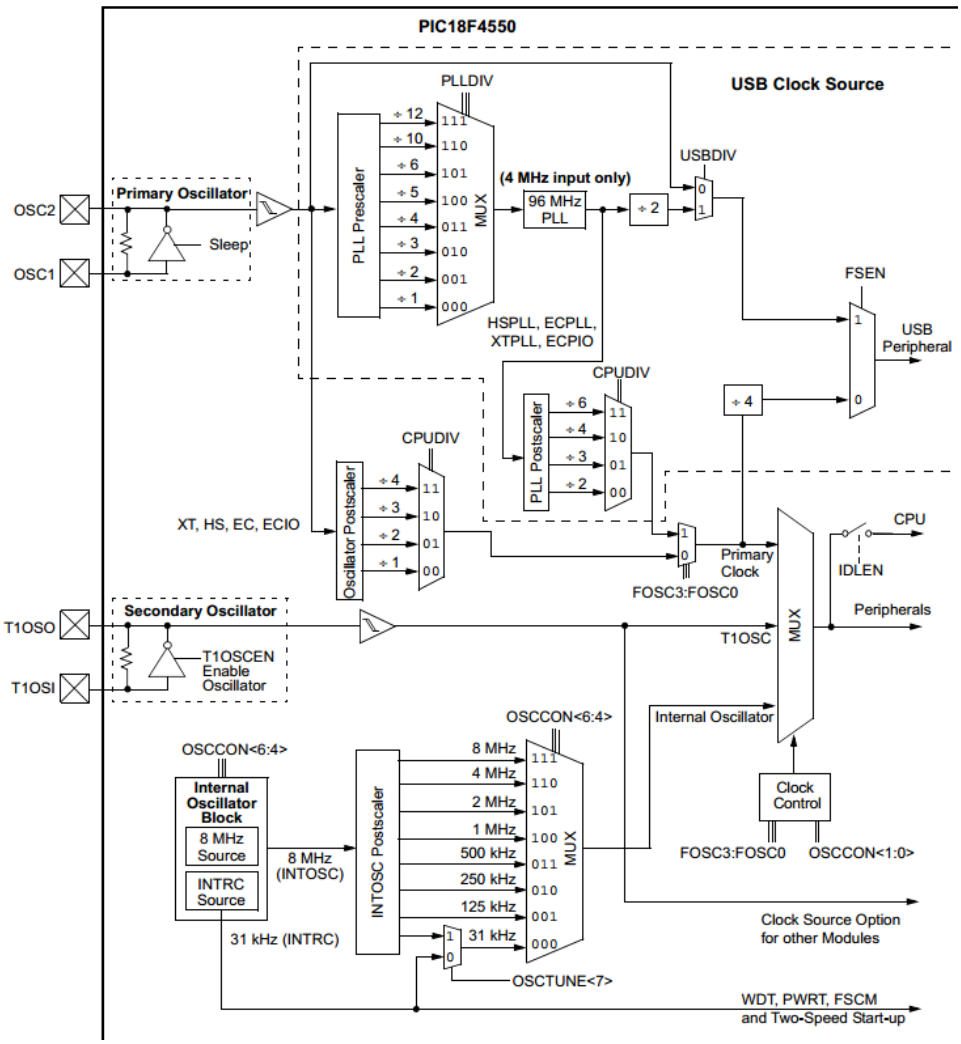
KUVA 6. PIC18F2550-mikrokontrollerin pinnit (Microchip 2004)

5.1 USB-piiristö

PIC kykenee sekä low-speed- että full-speed-nopeuksiin. Kommunikointi isännän ja mikrokontrollerin välillä tapahtuu SIE:n eli sarjaliitäntämoottorin kautta. Sitä voidaan käyttää joko PIC:in sisäisen lähetin-vastaanottimen kautta tai vaihtoehtoisesti se voidaan kytkeä ulkoiseen lähetin-vastaanottimeen. Sisäisen lähetin-vastaanottimen käyttöjännitettä ohjataan sisäisellä 3.3 V:in regulaattorilla. Sisäinen lähetin-vastaanotin kytketään päälle makrolla USB_TRANSCEIVER_OPTION. Nopeus valitaan FSEN-bitillä määrittämällä se USB_SPEED_OPTION-makrossa. Nopeuden valinta edellyttää sisäisten ylösvetovastusten käyttöä, jotka kytketään UPUEN-bitillä käyttäen makroa USB_PULLUP_OPTION (liite 1). Lähetin-vastaanotin saa käyttöjännitteensä V_{usb}-pinnistä (Microchip 2004, 163-165). Sisäinen regulaattori vaatii vakaasti toimiakseen V_{usb}:n ja maan välille vähintään 220 nF:n suuruisen kondensaattorin (Microchip 2004, 167).

5.1.1 Oskillaattorin asennus

Edellä mainittujen komponenttien lisäksi USB-piiri vaatii toimiakseen kiteen, joka takaa sen toiminnan vaatiman 48 tai 6 MHz:n sisäisen kellotaajuuden. PIC:in ensisijainen oskillaattori toimii kellolähteenä, jolloin toissijainen ja sisäinen oskillaattori jäävät itse mikrokontrollerin ja sen lisäosien käyttöön. Oskillaattorikonfiguraatio valitaan biteillä FOSC3:FOSC0, ja vaihtoehtoja on 12. Microchipin datalehdessä ilmoitetaan ensisijaisen oskillaattorin vaihtoehdoiksi keraamisia resonaattoreita ja kiteitä, jotka kytketään OSC1- ja OSC2-pinneihin (Microchip 2004, 23-26). Ensisijaisen oskillaattorin avulla saatu kello jaetaan siten, että vaihelukittua silmukkaa varten saadaan 4 MHz tulo. Tuloksena saatu 96 MHz taajuus jaetaan kahdella ennen syöttämistä USB-piiristölle asettamalla USBDIV-bitti yhdeksi (full-speed-nopeutta käytettäessä) (kuva 7).

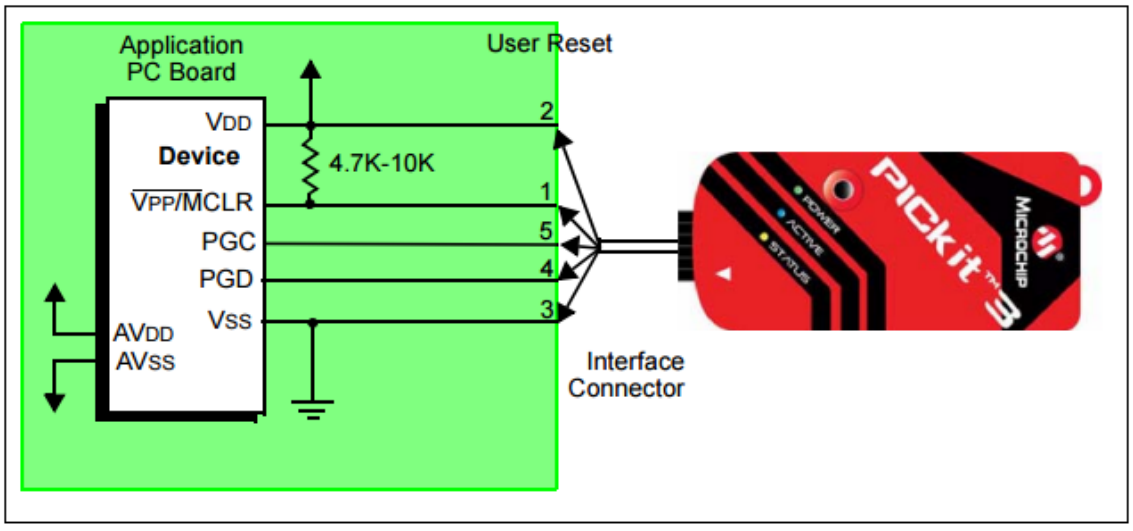


KUVA 7. Oskillaattorien toiminta PIC18F2550:ssa (Microchip 2004, 24)

Kun kide on valittu ja kytketty, kaikki tarvittavat bitit täytyy asettaa. Kiteen käyttö ensisijaisessa oskillaattorissa ilmoitetaan asettamalla FOSC-konfiguraation parametriksi HSPLL_HS (liite 2). Ensisijaista oskillaattoria käytettäessä kiteelle täytyy etsiä sopivat kytkentäkondensaattorit. Esimerkiksi työssä käytetty 20 MHz kide toimii ainakin 15 pikofaradin kondensaattoreilla, vaikkakaan ei optimaalisesti (Microchip 2004, 25). Kondensaattorit kytketään kiteen jalkojen ja maan välille.

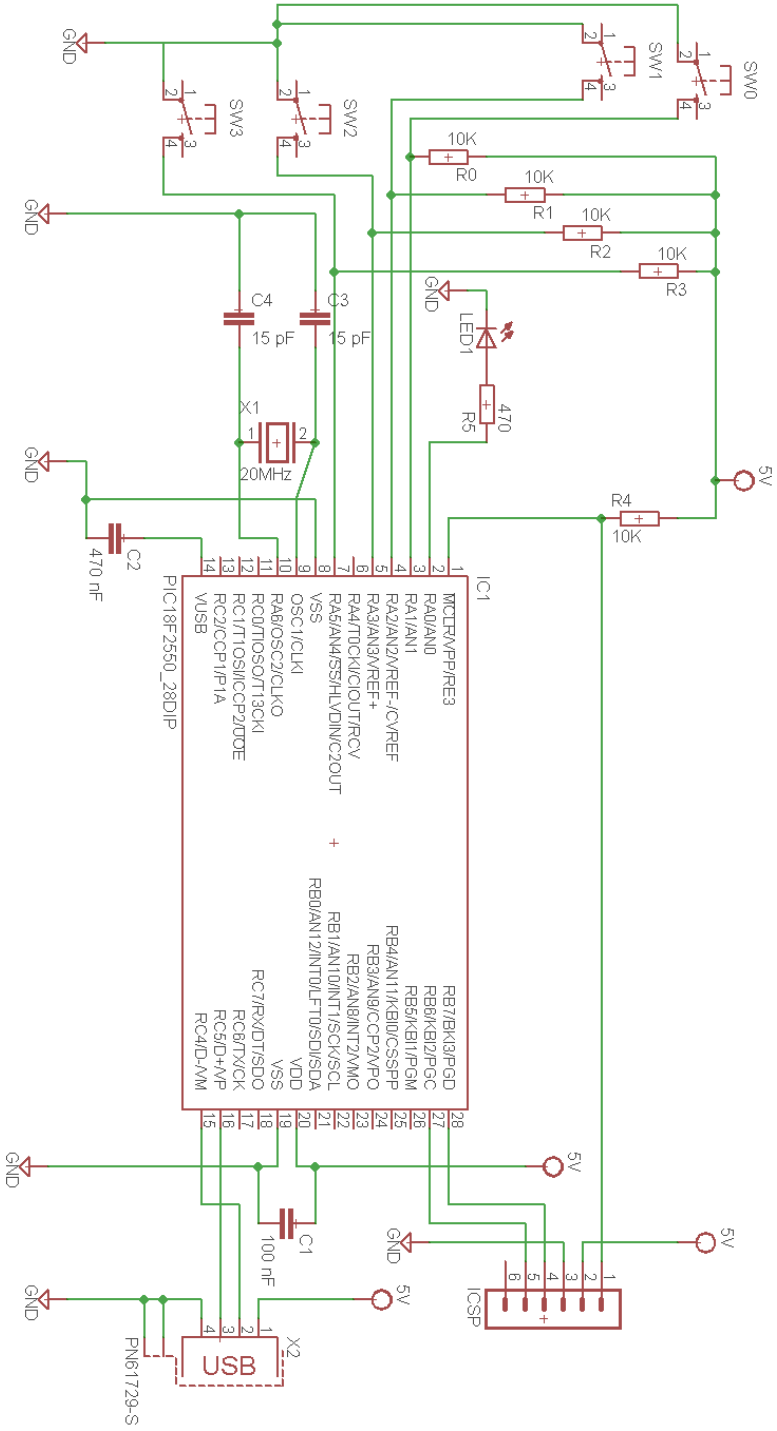
5.1.2 Piirin suunnittelu

Piirin suunnittelu aloitettiin vastusten, kiteen, V_{usb}:n kondensaattorin, LED-lampun, mikrokontrollerin ja painikkeiden sijoittamisella koekytkentäalustaan. Edellä mainitun kondensaattorin lisäksi käyttöjännitenapojen väliin kytkettiin 100 nF kondensaattori suojelemaan piiriä esimerkiksi virrankatkaisun aiheuttamilta piikeiltä. Laitteelle rakennettiin yksinkertainen USB-B-adapteri, jotta USB-johto voitaisiin irrottaa tarvittaessa sen sijaan, että johdot kuorittaisiin USB-kaapelista ja upotettaisiin koekytkentäalustaan. Kiteen jalat yhdistettiin OSC1- ja OSC2-pinneihin. Jotta mikrokontrolleria voitaisiin ohjelmoida, asennettiin koekytkentäalustaan kuuden pinnin adapteri, johon PicKIT3-mallinen ICSP-ohjelmointilaite voidaan liittää. Adapterin pinnit yhdistettiin mikrokontrolleriin kuvassa 8 osoitetulla tavalla, jolloin ohjelmoija on mahdollista asettaa ja poistaa helposti piiristä tarvittaessa.



KUVA 8. Ohjelmointilaitteen liittäminen mikrokontrolleriin (Microchip 2009)

Painikkeet kytkettiin siten, että signaali on 0-tilassa, kun painike on painettuna (kuva 9). Painikkeet toimivat siis active-low-periaatteella. Jokaiseen painikkeelliseen pinniin on kytketty myös ylösvetovastukset, jotka estävät tuntemattoman tilan syntymistä jännitteen kellumisen vuoksi. Ensimmäiseen PORTA-pinniin sijoitettu valoa emittoiva diodi syttyy, kun laite saa virtaa ja on lueteltu. Käyttöjännite kytkennälle saadaan suoraan USB-portista.



KUVA 9. Valmiin laitteen kytkentäkaavio

6 PIC18F2550-LAITEOHJELMISTO

6.1 Ohjelmalliset asetukset

USB-väylän käyttö mikrokontrollerien kanssa vaatii usein USB-pinon. PIC18F2550:n valmistaja Microchip on luonut tarkoitusta varten ohjelmistopaketin, johon kuuluu myös pino kirjastoineen. Pinon tehtävänä on helpottaa monimutkaisia tai alhaisen tason toimintoja, kuten luettelua ja pakettien kasaamista ja purkamista. Kyseiset toiminnot voidaan suorittaa kutsumalla niille tarkoitettuja funktioita, esimerkkinä USBDeviceTasks-funktio, jota kutsutaan tietyin väliajoin pakettien vastaanottamiseksi ja lähettämiseksi (liite 2). Jotta USB-piiristö ja itse mikrokontrolleri toimisivat, täytyy molemmille tehdä omat konfiguraatiot ja USB-rajapinnalle lisäksi kuvailutietuetiedostot.

6.1.1 Laiteprofiili

Laiteprofiililla tarkoitetaan tämän työn yhteydessä sellaisia asetusparametrejä, jotka liittyvät laitteen pinnien mahdolliseen alustukseen ja laitteen Windowsissa näkyvään nimeen. Kyseiset asetukset ovat lähdetiedostossa HardwareProfile.h. Kyseisessä tiedostossa määritellään käytetty kellotaajuus (48 MHz full-speed-tilaa varten), USB-laitteen PID ja VID, laitteen nimi, Windows-kuvaaja ja sarjanumero ja kytkimien ja ledin käyttöön liittyvät rekisterit (liite 3). PIC-mikrokontrollereissa yksi portti sisältää kahdeksan pinniä, joita ohjaillaan kolmella rekisterillä: TRIS-, PORT- ja LAT-rekistereillä. TRIS-rekistereillä määritetään datan suunta (tulo vai lähtö), PORT-rekistereillä luetaan yksittäisten bittien tilat ja LAT-rekistereillä ohjataan menokiikkujen toimintaa (Microchip 2004, 111). Kuten kuvasta 9 nähdään, kaikki kytkimet on liitetty PORTA:n pinneihin. Tällöin tarvitaan TRISA-, PORTA- ja LATA-rekistereitä. Jotta pinneistä voidaan lukea kytkimien tilat, täytyy niitä vastaavat bitit asettaa 1-tilaan TRISA-rekisterissä. RA0, johon LED on kytketty, vaatii 0-tilan vastaavaan bittiin TRISA-rekisterissä. Ledin kytkemiseksi päälle ja pois LATA0-bittiin kirjoitetaan 1-tila, kun LED halutaan päälle ja 0-tila, kun LED halutaan sammuttaa. Kytkinbiteille ja LED-bitille kirjoitettiin tunnisteet makroilla. Myös rekisterien alustamista varten luotiin mInitAllSwitches- ja mInitStatusLeds-makrot, joilla bitit asetetaan halutuiksi (liite 3).

6.1.2 USB:n konfigurointi

Tiedostossa usb_config.h määritetään yleiset USB-asetukset laitteelle, kuten päätepisteiden määrä, ohjaussiirtojen koko tavuina ja muun muuassa kyselläänkö dataa polling- vai interrupt-metodilla. Suurin ero näiden kahden välillä on USBDeviceTasks-funktion kutsu: polling-metodilla jää käyttäjän vastuulle kutsua kyseistä funktiota tarpeeksi usein, kun taas interrupt-metodilla ajoitukset hoidetaan tarpeeksi usein automaattisesti (Neuroscience and robotics lab 2012). Makrolla USB_PULLUP_OPTION määritetään, käytetäänkö sisäistä vai ulkoista ylösvetovastusta. USB_TRANSCEIVER_OPTION-makron parametri kertoo, käytetäänkö sisäistä vai ulkoista lähetinvastaanotinpiiriä. Lisäksi usb_config.h-tiedostossa kerrotaan, halutaanko käyttää low-speed- vai full-speed-nopeutta tiedonsiirrossa, ja tässä projektissa valittiin full-speed-tila (liite 1). HID-laiteluokkaa käytettäessä on myös luotava asetukset HID-päätepisteille koskien niiden siirtomuotoja. Ohjaussiirroille se ei ole tarpeellista.

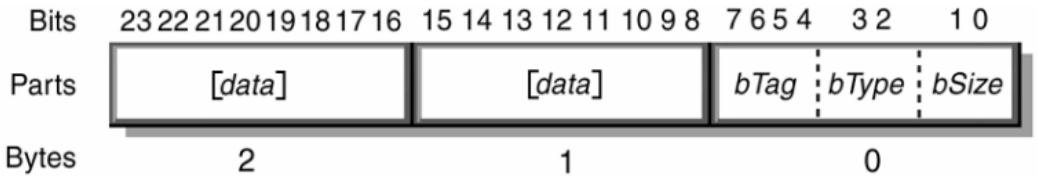
6.1.3 USB-kuvailutietueet

Usb_descriptors.c-tiedostolla määritetään kaikki laitteen kuvailutietueet, joita isäntä tarvitsee pystyäkseen kommunikoimaan laitteen kanssa. Tiedoston pohjana käytettiin Microchip-yrityksen luoman USB-pinon mukana tulleita esimerkkiprojekteja, joista erityisesti USB HID Joystick koettiin hyödylliseksi HID-kohtaisten kuvailutietueiden luonnissa. Tyypillisimmät asetukset, kuten laiteluokka voitiin säilyttää melkein sellaisenaan, kuitenkin VID- ja PID-tunnisteet piti muuttaa mikrokontrollerille sopivaksi. Suurimmat muutokset tulivat päätepisteiden ja raportin kuvailutietueisiin. Vaikka laite käyttää full-speed-tilaa, päätettiin ohjaussiirtojen kokoa rajoittaa kahdeksaan tavuun, jotta ajoitusrajoitukset varmasti saavutettaisiin. Laitteen maksimivirtavaatimukseksi ilmoitettiin 100 milliampeeria (liite 4). Vaikka pienempikin virtamäärä olisi riittänyt, katsottiin tarpeelliseksi ilmoittaa se riittävän suureksi siltä varalta, että projektiin haluttaisiin tulevaisuudessa lisätä esimerkiksi LCD-näyttö.

6.2 HID- ja raporttikuvailutietueet

HID-laitteissa on määriteltävä erikseen HID- ja raporttikuvailutietueet. HID-kuvailutietueella määritellään mm. laitteen maakoodi mahdollisia maakohtaisia asetuksia varten (0x00 jos ei käytössä) ja laitteen käyttämä HID-versio, joka tässä työssä on 1.11. HID-kuvailijassa lisäksi määritellään, kuinka monta luokkakuvailutietuetta laitteella on sekä niiden koot. Raporttikuvailutietue on aina pakollinen. Raporttikuvailutietue kuvailee laitteen luomaa dataa ja mitä on datan sisällön merkitys, esimerkiksi tieto vipujen tai painikkeiden asennosta. Koska jokainen laite on erilainen, on niillä myös vaihteleva määrä mahdollisia syötteitä, joiden tyypit ja arvot vaihtelevat laitteesta laitteeseen. Tästä syystä raporttikuvailutietue on poikkeuksellinen siten, että se ei ole vain arvoista muodostuva taulukko. Jokainen raporttikuvailutietue sisältää useita "alkioita", jotka ovat yksittäisiä informaatiopaloja.

Jokaisella alkiolla on sitä edeltävä määrittelytavu, josta saadaan selville alkion tunniste, tyyppi ja koko (kuva 10). Määritystavu sisältää aina bTag-, bType- ja bSize-ominaisuudet. bTag-bitit määräävät alkion tarkoituksen (esimerkiksi syöte tai lähtö). Bitit bType-kohdassa määrittelevät alkion tyypin, joita on kolme: main (päätuote), global (yleinen) ja local (paikallinen). Alkion bSize-bitit kertovat minkä kokoinen alkio on (0, 1, 2 tai 4 tavua). Arvon ollessa nolla alkio ei sisällä muuta kuin bTag-, bType- ja bSize-bitit. Alkiolla voi olla myös dataa koon ollessa suurempi kuin nolla, tosin se ei ole pakollista. Data voi kertoa esimerkiksi minkälaisia arvoja raportissa voidaan käyttää. Alkiot kerätään kokoelmiksi, ja kokoelmat voivat olla sisäkkäin raporttikuvailijassa. Raporttikuvailutietueet parsitaan HID-luokan purkajalla, joka analysoi yksittäiset alkiot kuvailijan sisällä lineaarisesti. Jokaisen alkion tila tallennetaan alkiotilataulukkoon (USB implementers forum 2001, 15).



KUVA 10. Lyhyen paketin määritysrakenne (USB implementers forum, 2001)

6.2.1 Main-alkiot

Raporttikuvailijat koostuvat aina kolmesta edellä mainitusta alkiotyypistä ja yhdessä kuvailijassa voi olla useita main-alkioita (USB implementers forum 2001, 25). Main-alkiot kertovat käsiteltävän datan koon ja määrittelevät, onko data absoluuttista vai suhteellista muun olennaisen tiedon lisäksi. Main-alkioille on viisi eri tunnistetta: input, output, feature, collection ja end collection. Input-tunniste viittaa alkioihin, joilla luetaan esimerkiksi laitteen akselien, nappien, vipujen jne. tila- ja asentodataa. Output-tunniste viittaa alkioihin, joilla asetetaan esimerkiksi akselien tai vipujen asento tiettyyn arvoon tai sytytetään yksi tai useampi LED. Feature-tunnisteiset alkiot kuvaavat epäsuoria ohjauksia tuloille ja lähdöille, kuten ohjelmisto-ominaisuuksia. Collection-tunnisteella aloitetaan input-, output- ja feature-tunnisteisten alkioiden kokoaminen ryhmäksi, kuten osoittimet tai fyysiset ohjaimet. Tunnisteella end collection kerrotaan kuvailijalle, että tällaisen kokoelman määritys on lopetettu.

6.2.2 Local- ja global-alkiot

Global-alkioilla kuvaillaan dataa. Main-alkiot perivät ominaisuutensa alkiotilataulukosta, joka vuorostaan määritellään global-alkioilla. Käytännössä tämä tarkoittaa sitä, että global-alkioiden ominaisuudet koskevat kaikkia sitä seuraavia alkioita kuvailutietueessa, ellei ominaisuuksia muuteta toisella global-alkiolla. Esimerkkinä alkiot logical minimum ja maximum, joilla määritellään jonkun alkion pienin ja suurin mahdollinen arvo. Jos logical minimum määritellään kuvailutietueen alussa nollaksi, kaikki kuvailutietueessa sen jälkeen luodut arvoalkiot voivat olla pienimmillään nolla, ellei logical minimum –alkiota määritetä uudelleen. Usage page on myös global-alkio.

Local-alkioilla määritetään laitteen ohjaimien ominaisuuksia (napit, akselit, vivut jne.). Local-alkioiden ominaisuudet eivät ”periydy” samalla tavalla kuin global-tyyppisten, vaan koskevat vain niitä main-alkioita, joiden yhteydessä ne on ilmoitettu (USB implementers forum 2001, 39-40). Jos ohjaimia määritellään useita main-alkiossa, voidaan jokaiselle tehdä tarvittaessa omat local-alkiot. Vaikka local-alkiot eivät periydy, voidaan yhteen ohjaimeen liittää useampi local-alkio. Local-alkioihin kuuluvat mm. tunnisteet usage, usage minimum/maximum ja delimiter. Etenkin tunnisteet usage minimum ja maximum ovat hyödyllisiä, koska niillä voidaan määrittää käyttötunnisteet

useille ohjaimille kerrallaan. Hyvä esimerkki on näppäimistöt, joissa sekä laitteen palauttama arvo (mitä painiketta painettiin) että käyttösivu ja käyttötunniste voidaan määrittää luomalla logical minimum/maximum- ja usage minimum/maximum –arvot kuvailutietueeseen.

6.2.3 Käyttösivut ja käyttötunnisteet

Raporttikuvailutietueen tärkeimpiin alkioihin kuuluvat käyttösivut ja käyttötunnisteet. Käyttösivulla tarkoitetaan laiteosaluokkaa, johon laitteen raporttialkio kuuluu, ja se sisältää toimintoja, joita kyseisellä käyttösivulla voidaan toteuttaa. Esimerkkejä käyttösivuista ovat napit, monitorit ja kameraohjaus. Käyttötunnisteella (Usage) selvitetään, mikä käyttösivun toiminnoista yritetään toteuttaa. Esimerkki käyttösivu-käyttötunniste yhdistelmästä voisi olla perinteinen joystick. Käyttösivuksi ilmoitettaisiin Generic Desktop, joka kattaa työpöytälaitteet, ja käyttötunnisteeksi valittaisiin kyseiseltä sivulta joystick ja samalla luotaisiin joystick-kokoelma. Kyseisen joystick-kokoelman sisälle lisättäisiin tämän jälkeen joystickin painikkeet ja akselit omaan kokoelmaansa, joka sisältäisi painikkeiden ja akselien käyttösivut ja täten niiden määritelmät (USB Implementers Forum 2001, 64).

6.2.4 Ohjauskonsolin raporttikuvailijan suunnittelu

Ohjauskonsolin raporttia suunniteltaessa aloitettiin seuraavista vaatimuksista:
-Raportissa täytyy olla tilaa sekä kaikkien porttien bittien tallentamista omiin alkioihinsa että mahdollista ylimääräistä dataa varten.
-Koska data saattaa tulevaisuudessa sisältää tekstiä, yhden alkion kooksi valittiin 8 bittiä eli yksi tavu.
-Samasta syystä yhden alkion loogisen arvon rajat asetetaan nollasta 255:een.
-Alkiot kootaan taulukoksi ja sekä syötteille että ulostulolle luodaan omat taulukkonsa.
-Laitteen käyttösivun täytyy olla sellainen, että se ei vastaa mitään olemassaolevaa laiteluokkaa, jotta käyttötunniste voidaan vapaasti määrittää.

Raporttikuvailutietueen luominen on hyvä aloittaa HID-laitteen vaatimuksista. Jokaisessa raporttikuvailutietueessa on oltava ainakin seuraavat alkiot: input (output tai feature), usage, usage page, logical minimum ja maximum, report size ja report count (USB implementers forum 2001, 25). HID-laitteiden käyttösivutaulukossa on

valmistajakohtaiset FF00-FFFF-arvoiset käyttösivut, jotka ovat ohjelmoijien määriteltävissä. Käyttötunniste valitaan yleislaitekokoelmakategoriasta, joka on mikä vain arvo väliltä 0x01 – 0x1F (USB implementers forum 2004, 15). Raporttikuvailijan määritys aloitetaan siis käyttösivun ja käyttötunnisteen luomisella. Lisäksi avataan kokoelma tulevia input- ja output-taulukoita varten (kuva 11). Kaikki tarvittavat heksadesimaalilukuarvot alkioille ovat HID-standardin määrittävässä dokumentissa (USB implementers forum 2001, 28, 35, 40).

```
// Class specific (report) descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{
    0x06, 0x00, 0xFF,       // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,             // Usage (Vendor Usage 1)
    0xA1, 0x01,             // Collection (Application)
```

KUVA 11. Raporttikuvailijan määritys

Käyttötunnisteet luotiin seuraavaksi alkioilla usage minimum ja usage maximum. Global-alkioina logical minimumilla ja maximumilla voitiin määrittää molempien taulukkojen arvorajat. Samalla päivitettiin alkiotilataulukkoon molempia arvotaulukkoja varten alkioiden koko bitteinä (report size) ja alkioiden määrä taulukossa (report count). Syötetaulukko luotiin input-tyyppisellä main-alkiolla, jossa data ilmoitettiin olevan taulukkomuodossa ja absoluuttinen (kuva 12).

```
// Class specific (report) descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{
    0x06, 0x00, 0xFF,       // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,             // Usage (Vendor Usage 1)
    0xA1, 0x01,             // Collection (Application)
    0x19, 0x01,             //      Usage Minimum
    0x29, 0x40,             //      Usage Maximum   //64 input usages total (0x01 to 0x40)
    0x15, 0x01,             //      Logical Minimum
    0x25, 0x40,             //      Logical Maximum (unsigned 255)
    0x75, 0x08,             //      Report Size: 8-bit field size
    0x95, 0x40,             //      Report Count: Make sixty-four 8-bit fields
    0x81, 0x00,             //      Input (Data, Array, Abs)
```

KUVA 12. Raporttikuvailijan syötetaulukon luonti

Lopuksi määriteltiin lähtötaulukko. Koska alkiot, joilla määritettiin loogiset arvorajat, taulukon koko ja taulukon alkioiden bittien määrä ovat global-tyyppisiä, voidaan lähtötaulukko luoda yksinkertaisesti määrittämällä vain käyttötunnisteet uudelleen (usage minimum ja maximum) ja käyttämällä output-alkiota. Output-alkiossa käytetään samoja parametrejä kuin input-taulukossa (absoluuttista dataa taulukkomuodossa).

Lopuksi laitekokoelma suljetaan end collection –tavulla, jolloin raporttikuvailijan luonti on valmis ja laite voi aloittaa datan vastaanottamisen ja lähettämisen isännältä ja isännälle (kuva 13; liite 4).

```
// Class specific (report) descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{
    0x06, 0x00, 0xFF,       // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,             // Usage (Vendor Usage 1)
    0xA1, 0x01,             // Collection (Application)
    0x19, 0x01,             //      Usage Minimum
    0x29, 0x40,             //      Usage Maximum   //64 input usages total (0x01 to 0x40)
    0x15, 0x01,             //      Logical Minimum
    0x25, 0x40,             //      Logical Maximum (unsigned 255)
    0x75, 0x08,             //      Report Size: 8-bit field size
    0x95, 0x40,             //      Report Count: Make sixty-four 8-bit fields
    0x81, 0x00,             //      Input (Data, Array, Abs)
    0x19, 0x01,             //      Usage Minimum
    0x29, 0x40,             //      Usage Maximum   //64 output usages total (0x01 to 0x40)
    0x91, 0x00,             //      Output (Data, Array, Abs)
    0xC0}                   // End Collection
};
```

KUVA 13. Raporttikuvailijan viimeistely

6.3 Laiteohjelmiston viimeistely

Laiteohjelmiston toiminnan ohjelmointi aloitettiin Microchipin USB-esimerkkiohjelmien ja Simon Innsin WFF-demon pohjalta (Simon Inns 2011, WFF generic HID demo 3 firmware). Sekä Microchipin ohjelmissa että Innsin demossa on käytetty hyväksi USB-pinon apufunktioita USB-laitteiston ja siirtojen alustamiseen. Käyttäjän tehtäväksi jää alustaa PIC-mikrokontrollerin rekisterit ja USB-toimintoihin vaadittavat muuttujat sekä tietenkin määrittää, mitä raporteilla ja siirroilla halutaan tehdä.

6.3.1 Muuttujien alustus

Jotta dataa voidaan käsitellä ohjelmallisesti, täytyy se pystyä tallentamaan laitteen muistiin. Tarkoitusta varten on muistista varattu 64-tavuiset taulukot ReceivedDataBuffer ja ToSendDataBuffer, joista ensimmäisellä vastaanotetaan ja toisella lähetetään isännälle data. Microchipin USB-pinoa käytettäessä tietoa pakettien siirtotilanteesta voidaan lukea USB_HANDLE-osoittimella, jolla varmistetaan muun muuassa, että dataa ei yritetä lukea tai kirjoittaa, ennen kuin se on mahdollista. Sekä

saapuville että lähteville siirroille luotiin omat USB_HANDLE-osoittimensa (liite 2). Esimerkkiohjelma sisältää PIC:in oskillaattorirekisterien ym. asetukset.

PIC:in porttien ja niihin liittyvien rekisterien alustamista varten luotiin initialisePIC-aliohjelma, jossa kaikki laitteen TRIS- ja PORT-rekisterit aluksi nollataan. Aliohjelmassa myös kutsutaan applicationInit-funktiolla mInitStatusLeds- ja mInitAllSwitches-makroja, joilla LED ja kytkimet saadaan toimintavalmiiksi. Lisäksi USB_HANDLE-osoittimet nollattiin. Tässä vaiheessa on tärkeää kutsua Microchipin USB-pinon USBDeviceInit-aliohjelmaa, joka hoitaa käyttäjän puolesta lukuisat tiedonsiirtoon ja toiminnan aloittamiseen vaadittavat asetukset. InitialisePIC-aliohjelman viimeiseksi tehtäväksi jää while-silmukan suorittaminen, joka sisältää laiteohjelmiston ”sydämenä” toimivan processUsbCommands-aliohjelman (liite 2).

6.3.2 USB-komentojen käsittely

Aliohjelman alussa tarkastetaan, että kaikki tarvittavat USB-asetukset ovat määritetty USBDeviceState-muuttujalla. Tarkastuksen onnistuessa luetaan datapuskurin tila ja tarkistetaan, onko laitteelle saapunut isännältä käskyä. Tarkastus suoritetaan USBOutHandle-muuttujaa vasten, johon on aina ennen tarkistusta tallennettu HIDRxPacket-makron tiedot. Kyseisellä makrolla kopioidaan vastaanotettu HID-paketti mikrokontrollerin muistiin (ReceivedDataBuffer), jos sellainen on isännältä lähetetty. Kun isännältä tulee käsky, se tallennetaan taulukon ensimmäiseen alkioon, jonka tavu luetaan, jos tarkistuksessa huomataan taulukon sisältävän jotain. Laiteohjelmistodemossa on valmiiksi koodi debug-viestin lähettämiseksi 0x10-käsky luettaessa. Komennolla 0x80 kytketään PA0-pinniin kytketty LED päälle, jos se on sammunut ja pois, jos se on päällä. Kytkeminen tapahtuu makrolla mStatusLED0_Toggle, joka suorittaa PA0-bitille XOR-operaation. Tällä voidaan varmistaa, että laite vastaanottaa komentoja ja/tai ei ole lukkiutunut (liite 2).

Viimeinen komento 0x81 käskee laitetta lukemaan niiden pinnien tilat, joihin on kytketty painike. Pinnien tilat kirjoitetaan ToSendDataBuffer-taulukkoon, ja jokaisen pinnin tila tallennetaan omaan alkioonsa. USB-pinon komennolla HIDTxHandleBusy tarkastetaan, että USBInHandle-muuttuja ei sisällä tietoa keskeneräisestä siirrosta. Jos lähteviä siirtoja ei ole käynnissä, USBInHandle-muuttujaan kirjoitetaan HIDTxPacket-funktiolla ToSendDataBuffer-taulukon tiedot, jotka lähetetään isännälle seuraavan

kerran kun se pyytää laitteelta dataa (liite 2). Data saapuu isännälle samanlaisessa taulukkomuodossa, kun se on tallennettu.

7 ISÄNTÄOHJELMISTO

Isäntäohjelmisto toimii laitteen ja käyttöjärjestelmän välisenä kommunikointityökaluna. Isäntäohjelmisto on vastuussa laitteen loogisesta liittämisestä, kommunikointikanavien luomisesta ja avaamisesta ja kommunikointitapahtumien aloituksesta. Jos isäntä ei kysy laitteelta dataa, se ei voi sitä lähettää. Isäntäohjelmiston runkona toimii Simon Innsin WFF Demo, jossa on valmiina luokat HID-kommunikoinnille ja yksinkertainen käyttöliittymä, jonka debug-kenttä kertoo, mitä laitteessa tapahtuu (Simon Inns 2011).

7.1 Laitteen liittäminen

Laitteen liittäminen Windows-käyttöjärjestelmässä tehdään usein sen omalla API:lla. Keskeisin funktio laitteen liittämiselle on CreateFile (William Novak 2013). Simon Innsin demossa laitteen liittäminen aloitetaan luomalla lista kaikista tietokoneeseen liitetyistä laitteista, joilla on HID-ajurin kanssa yhteensopiva GUID (Globally Unique Identifier) (Simon Inns 2011, WFF Generic HID demo 3). Toiminto suoritetaan aliohjelmalla findHidDevices ja se palauttaa boolean-arvon, joka kertoo löydettiinkö laitteita (tosi, jos löydettiin). Aliohjelmaa kutsutaan findTargetDevice-aliohjelmassa, joka tarkastaa kaikkien listan laitteiden VID- ja PID-arvot (liite 5). Tarkoituksena on löytää opinnäytetyössä luotu laite sen VID- ja PID-arvojen perusteella, jotka ovat tunnettuja. Jos laite löydetään, sen ominaisuudet listataan queryDeviceCapabilities-aliohjelmalla, jonka jälkeen sille luodaan luku- ja kirjoituskahvat Windows API:n CreateFile-aliohjelmalla (liite 5). Kahvojen luomisen onnistumista kysellään käyttöjärjestelmältä, ja jos ne luotiin onnistuneesti, voidaan aloittaa kommunikointi laitteen kanssa.

7.1.1 Laiteilmoitukset

Jokaisen HID-laitetta ohjaavan isäntäohjelman täytyy ottaa huomioon, että laite voidaan ohjelman ollessa käynnissä liittää tai irroittaa milloin vain. Windows-käyttöjärjestelmässä ohjelman täytyy rekisteröidä laitteen tapahtumat lähettämällä pääikkunakahvansa RegisterDeviceNotification-funktiolle (Developer Fusion, 2010). Tämä ohjeistaa käyttöjärjestelmää lähettämään ohjelman pääikkunaan

WM_DEVICECHANGE-viestin aina, kun USB-laite liitetään koneeseen. Ohjelma voi tämän jälkeen tarkastaa, onko liitetty laite sama kuin minkä se on rekisteröinyt. Samanlainen tarkastus voidaan tehdä irrotetulle laitteelle, jolloin ohjelman täytyy sulkea kaikki laitteeseen liitetyt kommunikointikahvat ja lopettaa laitteeseen liittyvä toiminta, kunnes se liitetään uudestaan tietokoneeseen. Isäntäohjelmassa laitetapahtumat rekisteröidään registerForDeviceNotifications-aliohjelmassa, ja viestit prosessoidaan handleDeviceNotificationMessages-aliohjelmassa. Aliohjelmalla findTargetDevice tarkastetaan, onko irrotettu tai liitetty laite sama, kuin opinnäytetyössä käytetty (liite 6).

7.2 Kommunikoinnin aloitus

7.2.1 Laitteen tuloraportin lukeminen

Kun laite on loogisesti liitetty ja kaikki tarvittavat kommunikointikahvat on luotu, voidaan aloittaa kommunikointi isännän ja laitteen välillä. Jokaista kommunikaatioyritystä edeltää tarkistus, jossa tarkistetaan onko laite vielä liitettynä. Jos laite on liitettynä, voidaan aloittaa luku- tai kirjoitustoimenpide. Kirjoitustoimenpide suoritetaan readRawReportFromDevice-aliohjelmassa. Laitteelta lukemista varten luodaan tapahtumaolio, jolla käsitellään ReadFile-funktiota. Lisäksi muistista varataan tilaa tulopuskuria varten, joka hoidetaan Marshal-oliolla (liite 7). Marshal-oliolla mahdollistetaan kaikkien sellaisten ohjelmointikielien kanssa työskentely, jotka eivät ole ns. ”hallittuja” (käytännössä kaikki muut kielet kuin Visual Basic ja C#).

Funktiolla ReadFile pyritään lukemaan laitetta varten luotu lukukahva, ja liittämään sen välittämät tiedot Marshal-oliolla varattuun puskuriin. Lukuoperaation onnistuminen tarkastetaan. Jos lukeminen onnistui, tallennetaan luettujen tavujen määrä numberOfBytesRead-muuttujaan (liite 7). Tämän jälkeen kopioidaan Marshal-olion puskurin tiedot hallittuun puskuriin, jolloin laitteen lähettämä raportti on luettavissa C#-kielellä.

Muuttujalla numberOfBytesRead välitetään tieto, kuinka monta tavua kopioidaan. Jos lukeminen epäonnistuu, lukuoperaatio peruutetaan ja laite irrotetaan ohjelmallisesti isännästä. Laitteen irrottamisella yritetään palauttaa laitteelle tunnettu tila. Lukuoperaation onnistumisen tai epäonnistumisen jälkeen vapautetaan Marshal-oliolla

varatut muistialueet ja suljetaan lukukahva (liite 7). Aliohjelmaa readRawReportFromDevice kutsutaan readSingleReportFromDevice-aliohjelmassa, jossa välitetään ensimmäiselle aliohjelmalle luettelussa saatu tieto laitteen raportin koosta tavuina. Lisäksi tarkistetaan, että tulopuskurin koko on asetettu ainakin yhtä suureksi kuin laitteelta saapuvan tuloraportin.

7.2.2 Laitteen lähtöraportin kirjoittaminen

Datan lähettäminen laitteelle tapahtuu aliohjelmalla writeReportToDevice. Ensiksi tarkastetaan, että laite on vielä liitettynä. Laitteen ollessa kiinni pyritään lähettämään lähtöraportti laitteelle. Tämä suoritetaan laitetta varten luodulla kirjoituskahvalla käyttäen WriteFile-funktiota. Lähetykseen käytetään interrupt-siirtoa päätepisteeseen yksi. Kirjoitettava data välitetään aliohjelman outputReportBuffer-puskuriin (liite 7).

7.3 Pääohjelman toiminta

Pääohjelman suorittaminen aloitetaan luomalla USB-laiteolio, joka saa parametreinä työssä käytetyn laitteen PID- ja VID-arvot. Kyseiseen olioon lisätään laitekuuntelija, joka ilmoittaa sille kaikista tietokoneessa tapahtuvista USB-laitemuutoksista (liite 8). Laite etsitään, ja jos se on liitetty, aktivoidaan painike pääikkunassa, jolla voidaan sytyttää ja sammuttaa laitteen LED. Tällä voidaan varmistaa, että isännän käskyt menevät perille ja laite on liitetty ohjelmallisesti oikein. Ohjelma käyttää ajastinta laskemaan, milloin laitteelta kysytään tai sille lähetetään dataa. Yhden millisekunnin välein suoritetaan painikkeiden lukukäsky, jos pääikkunassa ei ole painettu LED-painiketta (jolloin LED-käsky lähetetään ensin).

Käskyt lähetetään laitteelle kutsumalla luodun usbDemoDevice-olion aliohjelmia. Esimerkiksi jos halutaan lukea painikkeiden tila, kutsutaan readPushButtonState-aliohjelmaa (liite 9). Aliohjelmassa varataan tulo- ja lähtötaulukko, ja lähtötaulukon toiseen alkioon lisätään komentotavu 0x81. Komento lähetetään writeRawReportToDevice-aliohjelmalla laitteelle ja laitteen vastaus vastaanotetaan readSingleReportFromDevice-aliohjelmalla. Koska työssä rakennetussa konsolissa on useampi painike, ohjelmoitiin for-silmukka, joka estää useamman samanaikaisen painikkeen painalluksen aiheuttamat ongelmat. Jos useampi painike on painettuna, for-

silmukka palauttaa vain sen painetun painikkeen tilan, joka ensimmäisenä havaitaan (liite 9). Käytännössä tämä tarkoittaa sitä, että vain vasemman puoleisin painike katsotaan painetuksi useamman painikkeen ollessa painettuna. For-silmukassa tallennetaan myös olion previousState-muuttujaan tieto siitä, oliko painike painettuna viime kerralla, kun painikkeiden tilaa kysyttiin. Jos kahden tai useamman kyselyn välissä huomataan, että yksi tai useampi painike on vielä pohjassa, ei painalluksia rekisteröidä. Tällä estetään ohjelmaa tulkitsemasta painikkeen pohjassa pitämistä useammaksi painallukseksi.

7.4 Painikefunktiot

Kun nappia on painettu ja painetun painikkeen numero palautettu sitä kutsuneelle timer1_Tick-aliohjelmalle, tarkastetaan switch-lauseella painikkeen numero ja suoritetaan sitä vastaava aliohjelma (liite 8). Yksi idea oli toteuttaa audio- tai videolaitteiden lähtöjen ja tulojen vaihto ohjelmallisesti, mutta idea osoittautui toistaiseksi mahdottomaksi. Kyseiset toiminnot ovat käyttöjärjestelmän piilottamia ja suojelemia ominaisuuksia, joihin on ohjelmallisesti vaikea päästä käsiksi, ja joihin ei ole olemassa minkäänlaista dokumentaatiota. Tästä johtuen päätettiin koemielessä ohjelmoida funktiot, joilla nostetaan ja lasketaan äänenvoimakkuutta, ja kolmannella vaimennetaan käyttöjärjestelmän kaikki äänet.

7.4.1 Windows-API:n käyttö

Windows-käyttöjärjestelmän toimintojen ohjaaminen vaatii sille tarkoitetun ohjelmointirajapinnan käyttöä. Kyseinen rajapinta muodostuu useista jaetuista DLL-kirjastoista, jotka sisältävät tarvittavat funktiot käyttöjärjestelmän ohjaamiseen. Komennolla WM_APPCOMMAND on mahdollista välittää viestejä käyttöjärjestelmälle, ja sen mahdollisiin arvoihin sisältyy muun muuassa äänenvoimakkuuden säätöarvot (Keyboard Input Notifications:WM_APPCOMMAND). Komento lähetetään arvoineen käyttämällä User32-kirjaston SendMessage-funktiota (SendMessage function; Pinvoke.NET). 64-bittisissä käyttöjärjestelmissä funktion kokonaisluvuista muodostuvat parametrit täytyy välittää funktiolle osoittimilla, muutoin ohjelma kaatuu (Pinvoke.NET). Tästä syystä isäntäohjelman pääikkunan luokkaan lisättiin sekä WM_APPCOMMAND-viestinumero että tarvittavat viestin

parametriarvot. Esimerkiksi jos äänenvoimakkuutta halutaan laskea, käytetään APPCOMMAND_VOLUME_DOWN-parametriarvoa, joka on yhdeksän (Keyboard Input Notifications:WM_APPCOMMAND).

```
private void button1Pressed()
    {
        SendMessageW(
            this.Handle,                    // Pääikkunan kahva
            WM_APPCOMMAND,                  // Viestiarvo (0x319)
            this.Handle,                    // Mahdollinen lisäinfo
            (IntPtr)APPCOMMAND_VOLUME_MUTE  // Komennon parametriarvo (0x80000)
            );
    }
```

KUVA 14. Esimerkki SendMessage-funktion käytöstä

Kuvassa 14 nähdään esimerkki komennon käytöstä painikefunktiossa (liite 8). Painiketta 1 painettaessa kutsutaan button1Pressed-funktiota, jonka sisällä puolestaan kutsutaan SendMessage-funktiota, joka tässä tapauksessa saa viestiarvokseen 319 (WM_APPCOMMAND-komento) ja komennon parametriarvoksi 0x80000 (APPCOMMAND_VOLUME_MUTE). Tällä komennolla vaimennetaan isännän kaikkien audiolaitteiden tuottamat äänet. Vastaavat funktiot tehtiin äänenvoimakkuuden nostamiselle ja laskemiselle.

8 POHDINTA

Työssä asetettiin tavoitteiksi USB-teknologian ja ohjelmoinnin ymmärtäminen ja yksinkertaisen USB-laitteen toteuttaminen. USB-väylän toimintaa tutkittiin sekä elektroniikka- että ohjelmointinäkökulmasta. Työssä saavutettiin hyvin laaja ymmärrys siitä, kuinka USB-väylä toimii ja kuinka esimerkiksi erilaiset standardin asettamat vaatimukset vaikuttavat USB-laitteen suunnitteluun. Konsoli saatiin rakennettua toimintakuntoiseksi ja opittiin etsimään datalehdistä ja USB-standardin määrittelevistä dokumenteista tarvittavat arvot esimerkiksi kondensaattoreille. Konsolin rakentamisessa ja suunnittelussa ei ajauduttu suurempiin ongelmiin. Konsolin kehitystä voitaisiin tulevaisuudessa jatkaa mahdollisesti lisäämällä siihen esimerkiksi LCD-näyttö diagnooseja varten.

Vaikkakin laiteohjelmisto perustui valmiiseen koodiin, katsottiin tärkeäksi ymmärtää mihin erilaisten ohjausbittien asettaminen perustuu, ja miten kuvailutietueet oli määritelty. Etenkin kuvailutietueiden ymmärtämisessä oli paljon vaikeuksia liian yleistetyn dokumentoinnin vuoksi. Niitä pyrittiinkin ymmärtämään muista projekteista haetuilla esimerkeillä, joka helpotti huomattavasti raporttikuvailutietueen suunnittelua. Myöhemmin huomattiin, että monissa projekteissa raportit muistuttavat usein toisiaan ja niin myös tässä työssä, koska yksinkertaisten laitteiden tapauksessa ei ole tarpeellista ”keksiä pyörää uudelleen”.

Painikkeiden toiminnan suunnittelussa otettiin huomioon sekä miten tieto saadaan laitteelta isännälle että miten data käsitellään isännän ohjelmassa. Kuvailutietueessa luodulla tavutaulukolla oli helppo siirtää kaikkien painikkeiden tilat omissa alkioissaan isännälle, joka puolestaan pystyi helposti käsittelemään taulukon alkioita silmukalla. Ainoa vaikeus oli ohjelmoida ominaisuus, joka estäisi useamman painikkeen samanaikaisen painamisen tai painikkeen pohjassa pitämisen aiheuttamat virhetilanteet. Ongelma saatiin ratkaistua vartijamuuttujalla ja if-lausekkeella.

Painikkeita vastaavat funktiot ohjelmoitiin Windows-käyttöjärjestelmän ohjelmointirajapintoja käyttäen. Toteutetussa versiossa lähetetään viestejä käyttöjärjestelmälle, joka ohjeistaa sitä muuttamaan äänenvoimakkuusasetuksia. Tulevaisuudessa voitaisiin mahdollisesti yrittää muita toimintoja, kuten tietokoneeseen liitettyjen audio- ja videolaitteiden ohjaamista. Lisäksi voitaisiin harkita mahdollisuutta käyttää mikrokontrollerin muita ominaisuuksia hyväksi laitteessa, kuten A/D-muuntimia. Lisäksi laiteohjelmistoa voitaisiin kehittää pitemmälle, jotta esimerkiksi edellä mainittua LCD-näyttöä olisi mahdollista käyttää laitteen kanssa.

LÄHTEET

Axelson, J. 2009. USB complete: the developer's guide 4th edition. Madison, WI : Independent Publishers Group

Boston Globe. 1999. USB deserves more support. Julkaistu 20.5.1999. Luettu 3.4.2015. http://simson.net/clips/1999/99.Globe.05-20.USB_deserves_more_support+.shtml

Cables Plus. USB cable length limitations. Luettu 8.4.2015. http://www.cablesplususa.com/pdf/USB_Cable_Length_Limitations.pdf

Developer Fusion. 2010. Working with USB devices in .NET and C#. Julkaistu 7.7.2010. Luettu 9.5.2015. http://www.developerfusion.com/article/84338/making-usb-c-friendly/

Engadget. 2010. European standardization bodies formalize micro-USB cellphone charger standard. Julkaistu 29.12.2010. Luettu 5.5.2015. http://www.engadget.com/2010/12/29/european-standardization-bodies-formalize-micro-usb-cellphone-ch/

FTDI. 2009. USB data packet structure. Julkaistu 11.11.2009. Luettu 10.4.2015. http://www.ftdichip.com/Support/Documents/TechnicalNotes/TN_116_USB%20Data%20Structure.pdf

Introduction to HID concepts. Microsoft. Luettu 17.4.2015. https://msdn.microsoft.com/enus/library/windows/hardware/jj126202%28v=vs.85%29.aspx

Keyboard Input Notifications:WM_APPCOMMAND. Microsoft. Luettu 12.5.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms646275(v=vs.85).aspx

Microchip. 2004. PIC18F2455/2550/4455/4550 Data Sheet. Luettu 16.4.2015. http://ww1.microchip.com/downloads/en/DeviceDoc/39632b.pdf

Microchip. 2009. PICkit™ 3 Programmer/Debugger User's Guide. Luettu 20.4.2015. ww1.microchip.com/downloads/en/DeviceDoc/PICkit_3_User_Guide_51795A.pdf

Neuroscience and robotics lab. 2012. USB communication using PIC microcontrollers. Luettu 23.4.2015. http://hades.mech.northwestern.edu/index.php/USB_Communication_using_PIC_microcontrollers

Pinvoke.NET. SendMessage (user32). Muokattu 10.5.2013. Luettu 12.5.2015. http://www.pinvoke.net/default.aspx/user32/SendMessage.html

SendMessage function. Microsoft. Luettu 12.5.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms644950%28v=vs.85%29.aspx

Simon Inns. 2011. WFF generic HID demo 3 firmware. Julkaistu 21.11.2011. Luettu 18.3.2015.
http://www.waitingforfriday.com/images/a/a6/WFF_Generic_HID_Demo_3_firmware.zip

Simon Inns. 2011. WFF Generic HID Demo. Julkaistu 21.11.2011. Luettu 18.3.2015.
http://www.waitingforfriday.com/images/3/36/WFF_Generic_HID_Demo_3.zip

USB Implementers Forum. 2000. Universal serial bus specification revision 2.0. Julkaistu 27.4.2000. Luettu 8.4.2015. http://www.usb.org/developers/docs

USB Implementers Forum. 2001. Device class definition for Human Interface Devices (HID). Julkaistu 27.6.2001. Luettu 26.4.2015.
http://www.usb.org/developers/hidpage/HID1_11.pdf

USB Implementers Forum. 2004. HID usage tables. Julkaistu 28.10.2004. Luettu 23.4.2015.
http://www.usb.org/developers/hidpage/Hut1_12v2.pdf

USB Implementers Forum. 2007. Universal Serial Bus Micro-USB Cables and Connectors Specification. Julkaistu 4.4.2007. Luettu 4.4.2015.
http://mgvs.org/public/shema/datasheet/usb_20/Micro-USB_final/Micro-USB_1_01.pdf

USB in a nutshell. 2010. Beyond Logic. Julkaistu 17.9.2010. Luettu 4.4.2015.
http://www.beyondlogic.org/usbnutshell/usb1.shtml

William Novak. 2013. Real-time data-transfer with a PIC18F4550 microcontroller via USB with C#.NET. Julkaistu 4.1.2013. Luettu 9.5.2015.
https://drive.google.com/file/d/0BwD6p8UHlw6GcHBRRHV2b2djNVk/view

USB made simple. Luettu10.4.2015. http://www.usbmadesimple.co.uk/ums_7.htm

LIITTEET

Liite 1. usb_config.h-tiedosto (Simon Inns 2011, WFF generic HID demo 3 firmware)

1(2)

```
/********************************************************************
   usb_config.c

   WFF USB Generic HID Demonstration 3
    usbGenericHidCommunication reference firmware 3_0_0_0
    Copyright (C) 2011 Simon Inns

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

   Email: simon.inns@gmail.com

********************************************************************/

#ifndef USBCONFIG_H
#define USBCONFIG_H

// Definitions
#define USB_EP0_BUFF_SIZE     8  // Valid Options are 8, 16, 32, or 64 bytes.

#define USB_MAX_NUM_INT       1
#define USB_MAX_EP_NUMBER      1

// USB device descriptor
#define USB_USER_DEVICE_DESCRIPTOR &device_dsc
#define USB_USER_DEVICE_DESCRIPTOR_INCLUDE extern ROM USB_DEVICE_DESCRIPTOR
device_dsc

// Configuration descriptors
#define USB_USER_CONFIG_DESCRIPTOR USB_CD_Ptr
#define USB_USER_CONFIG_DESCRIPTOR_INCLUDE extern ROM BYTE *ROM USB_CD_Ptr[]

// Set the USB ping pong mode
#define USB_PING_PONG_MODE USB_PING_PONG__FULL_PING_PONG

// Uncomment either USB_POLLING or USB_INTERRUPT to set how the USB
// device is updated
#define USB_POLLING
//#define USB_INTERRUPT

// Set the USB pullup option for usb_device.h
#define USB_PULLUP_OPTION USB_PULLUP_ENABLE
//#define USB_PULLUP_OPTION USB_PULLUP_DISABLED

// Use the internal USB transceiver module
```

2(2)

```
#define USB_TRANSCEIVER_OPTION USB_INTERNAL_TRANSCEIVER

// Set either full-speed or low-speed USB device mode
#define USB_SPEED_OPTION USB_FULL_SPEED
//#define USB_SPEED_OPTION USB_LOW_SPEED

// Option to enable auto-arming of the status stage of control transfers
#define USB_ENABLE_STATUS_STAGE_TIMEOUTS

// Section 9.2.6 of the USB 2.0 specifications state that:
// Control transfers with no data stage must complete within 50ms of the start of
the control transfer.
// Control transfers with (IN) data stage must complete within 50ms of sending
the last IN data packet in fullfilment of the data stage.
// Control transfers with (OUT) data stage have no specific status stage timing
(but must not exceed 5 seconds)
#define USB_STATUS_STAGE_TIMEOUT     (BYTE)45

#define USB_SUPPORT_DEVICE

#define USB_NUM_STRING_DESCRIPTORS 3

// Enable/disable event handlers
//#define USB_INTERRUPT_LEGACY_CALLBACKS
#define USB_ENABLE_ALL_HANDLERS
//#define USB_ENABLE_SUSPEND_HANDLER
//#define USB_ENABLE_WAKEUP_FROM_SUSPEND_HANDLER
//#define USB_ENABLE_SOF_HANDLER
//#define USB_ENABLE_ERROR_HANDLER
//#define USB_ENABLE_OTHER_REQUEST_HANDLER
//#define USB_ENABLE_SET_DESCRIPTOR_HANDLER
//#define USB_ENABLE_INIT_EP_HANDLER
//#define USB_ENABLE_EP0_DATA_HANDLER
//#define USB_ENABLE_TRANSFER_COMPLETE_HANDLER

// Set device type to USB HID
#define USB_USE_HID

// HID endpoint allocation
#define HID_INTF_ID             0x00
#define HID_EP              1
#define HID_INT_OUT_EP_SIZE     3
#define HID_INT_IN_EP_SIZE      3
#define HID_NUM_OF_DSC          1
#define HID_RPT01_SIZE          28

#endif
```

Liite 2. Main.c-tiedosto (Simon Inns 2011, WFF generic HID demo 3 firmware)


```
/********************************************************************
   main.c

   WFF USB Generic HID Demonstration 3
    usbGenericHidCommunication reference firmware 3_0_0_0
    Copyright (C) 2011 Simon Inns

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

   Email: simon.inns@gmail.com

********************************************************************/

#ifndef MAIN_C
#define MAIN_C

// Global includes
// Note: string.h is required for sprintf commands for debug
#include <string.h>

// Local includes
#include "HardwareProfile.h"
#include "debug.h"

// Microchip Application Library includes
// (expects V2.9a of the USB library from "Microchip Solutions v2011-07-14")
//
// The library location must be set in:
// Project -> Build Options Project -> Directories -> Include search path
// in order for the project to compile.
#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"

// Ensure we have the correct target PIC device family
#if !defined(__18F4550) && !defined(__18F2550)
   #error "This firmware only supports either the PIC18F4550 or PIC18F2550 microcontrollers."
#endif

// Define the globals for the USB data in the USB RAM of the PIC18F*550
#pragma udata
#pragma udata USB_VARIABLES=0x500
unsigned char ReceivedDataBuffer[64];
unsigned char ToSendDataBuffer[64];
#pragma udata

USB_HANDLE USBOutHandle = 0;
USB_HANDLE USBInHandle = 0;
BOOL blinkStatusValid = FLAG_TRUE;

// PIC18F4550/PIC18F2550 configuration for the WFF Generic HID test device
#pragma config PLLDIV   = 5         // 20Mhz external oscillator
#pragma config CPUDIV   = OSC1_PLL2
#pragma config USBDIV   = 2         // Clock source from 96MHz PLL/2
#pragma config FOSC     = HSPLL_HS
#pragma config FCMEN    = OFF
#pragma config IESO     = OFF
#pragma config PWRT     = OFF
#pragma config BOR      = ON
#pragma config BORV     = 3
#pragma config VREGEN   = ON
#pragma config WDT      = OFF
```

```
#pragma config WDTPS    = 32768
#pragma config MCLRE    = ON
#pragma config LPT1OSC  = OFF
#pragma config PBADEN   = OFF
// #pragma config CCP2MX   = ON
#pragma config STVREN   = ON
#pragma config LVP      = OFF
// #pragma config ICPRT    = OFF
#pragma config XINST    = OFF
#pragma config CP0      = OFF
#pragma config CP1      = OFF
// #pragma config CP2      = OFF
// #pragma config CP3      = OFF
#pragma config CPB      = OFF
// #pragma config CPD      = OFF
#pragma config WRT0     = OFF
#pragma config WRT1     = OFF
// #pragma config WRT2     = OFF
// #pragma config WRT3     = OFF
#pragma config WRTB     = OFF
#pragma config WRTC     = OFF
// #pragma config WRTD     = OFF
#pragma config EBTR0    = OFF
#pragma config EBTR1    = OFF
// #pragma config EBTR2    = OFF
// #pragma config EBTR3    = OFF
#pragma config EBTRB    = OFF

// Private function prototypes
static void initialisePic(void);
void processUsbCommands(void);
void applicationInit(void);
void USBCBSendResume(void);
void highPriorityISRCode();
void lowPriorityISRCode();

// Remap vectors for compatibilty with Microchip USB boot loaders
#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
    #define REMAPPED_RESET_VECTOR_ADDRESS         0x1000
    #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS   0x1008
    #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS 0x1018
#elif defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
    #define REMAPPED_RESET_VECTOR_ADDRESS         0x800
    #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS   0x808
    #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS 0x818
#else
    #define REMAPPED_RESET_VECTOR_ADDRESS         0x00
    #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS   0x08
    #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS 0x18
#endif

#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER) ||
defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
    extern void _startup (void);
    #pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
    void _reset (void)
    {
        _asm goto _startup _endasm
    }
#endif

#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void)
{
     _asm goto highPriorityISRCode _endasm
}

#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void)
{
     _asm goto lowPriorityISRCode _endasm
}
```

```
#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER) ||
defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void)
{
     _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}

#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void)
{
     _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}
#endif

#pragma code

// High-priority ISR handling function
#pragma interrupt highPriorityISRCode
void highPriorityISRCode()
{
   // Application specific high-priority ISR code goes here

   #if defined(USB_INTERRUPT)
      // Perform USB device tasks
      USBDeviceTasks();
   #endif

}

// Low-priority ISR handling function
#pragma interruptlow lowPriorityISRCode
void lowPriorityISRCode()
{
   // Application specific low-priority ISR code goes here
}

// String for creating debug messages
char debugString[64];

// Main program entry point
void main(void)
{
   // Initialise and configure the PIC ready to go
    initialisePic();

   // If we are running in interrupt mode attempt to attach the USB device
    #if defined(USB_INTERRUPT)
        USBDeviceAttach();
    #endif

   // Initialise the debug log functions
    debugInitialise();

    // Show that we are up and running
    mStatusLED0_on();

   sprintf(debugString, "USB Generic HID Demonstration 3");
   debugOut(debugString);

   sprintf(debugString, "(C)2011 Simon Inns - http://www.waitingforfriday.com");
   debugOut(debugString);

   sprintf(debugString, "USB Device Initialised.");
   debugOut(debugString);

   // Main processing loop
    while(1)
    {
        #if defined(USB_POLLING)
          // If we are in polling mode the USB device tasks must be processed here
          // (otherwise the interrupt is performing this task)
            USBDeviceTasks();
        #endif
```

```
        // Process USB Commands
          processUsbCommands();

          // Note: Other application specific actions can be placed here
      }
}

// Initialise the PIC
static void initialisePic(void)
{
    // PIC port set up -----------------------------------------------------

    // Default all pins to digital
     ADCON1 = 0x0F;

    // Configure ports as inputs (1) or outputs(0)
    TRISA = 0b00000000;
    TRISB = 0b00000000;
    TRISC = 0b00000000;
#if defined(__18F4550)
    TRISD = 0b00000000;
    TRISE = 0b00000000;
#endif

    // Clear all ports
    PORTA = 0b00000000;
    PORTB = 0b00000000;
    PORTC = 0b00000000;
#if defined(__18F4550)
    PORTD = 0b00000000;
    PORTE = 0b00000000;
#endif

    // If you have a VBUS sense pin (for self-powered devices when you
    // want to detect if the USB host is connected) you have to specify
    // your input pin in HardwareProfile.h
     #if defined(USE_USB_BUS_SENSE_IO)
        tris_usb_bus_sense = INPUT_PIN;
     #endif

     // In the case of a device which can be both self-powered and bus-powered
     // the device must respond correctly to a GetStatus (device) request and
     // tell the host how it is currently powered.
     //
     // To do this you must device a pin which is high when self powered and low
     // when bus powered and define this in HardwareProfile.h
     #if defined(USE_SELF_POWER_SENSE_IO)
        tris_self_power = INPUT_PIN;
     #endif

     // Application specific initialisation
     applicationInit();

     // Initialise the USB device
     USBDeviceInit();
}

// Application specific device initialisation
void applicationInit(void)
{
    // Initialise the status LEDs
    mInitStatusLeds();

    // Initialise the switches
    mInitAllSwitches();

     // Initialize the variable holding the USB handle for the last transmission
     USBOutHandle = 0;
     USBInHandle = 0;
}

// Process USB commands
void processUsbCommands(void)
```

```
{
    // Check if we are in the configured state; otherwise just return
    if((USBDeviceState < CONFIGURED_STATE) || (USBSuspendControl == 1))
    {
        // We are not configured
        return;
   }

   // Check if data was received from the host.
    if(!HIDRxHandleBusy(USBOutHandle))
    {
      // Command mode
       switch(ReceivedDataBuffer[0])
      {
          case 0x10:   // Debug information request from host
              // Copy any waiting debug text to the send data buffer
              copyDebugToSendBuffer((BYTE*)&ToSendDataBuffer[0]);

               // Transmit the response to the host
                 if(!HIDTxHandleBusy(USBInHandle))
              {
                  USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
              }
              break;

         // Place application specific commands here:

           case 0x80:  // Toggle the LED

              // Toggle the LED0
              mStatusLED0_Toggle();
              break;

           case 0x81:  // Read the push switch status
              ToSendDataBuffer[0] = sw0;
                 ToSendDataBuffer[1] = sw1;
                 ToSendDataBuffer[2] = sw2;
                 ToSendDataBuffer[3] = sw3;

              // Transmit the response to the host
                 if(!HIDTxHandleBusy(USBInHandle))
              {
                  USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
              }
              break;

           case 0x82:  // Read the LED status
              // Get the LED state and put it in the send buffer
              ToSendDataBuffer[0] = mStatusLED0_Get();

              // Transmit the response to the host
                 if(!HIDTxHandleBusy(USBInHandle))
              {
                  USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
              }
              break;

           default:   // Unknown command received
               break;
      }

       // Re-arm the OUT endpoint for the next packet
       USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
   }
}

// USB Callback handling routines ------------------------------------------------------------

// Call back that is invoked when a USB suspend is detected
void USBCBSuspend(void)
{
}

// This call back is invoked when a wakeup from USB suspend is detected.
```

```
void USBCBWakeFromSuspend(void)
{
}

// The USB host sends out a SOF packet to full-speed devices every 1 ms.
void USBCB_SOF_Handler(void)
{
    // No need to clear UIRbits.SOFIF to 0 here. Callback caller is already doing that.
}

// The purpose of this callback is mainly for debugging during development.
// Check UEIR to see which error causes the interrupt.
void USBCBErrorHandler(void)
{
    // No need to clear UEIR to 0 here.
    // Callback caller is already doing that.
}

// Check other requests callback
void USBCBCheckOtherReq(void)
{
    USBCheckHIDRequest();
}

// Callback function is called when a SETUP, bRequest: SET_DESCRIPTOR request arrives.
void USBCBStdSetDscHandler(void)
{
    // You must claim session ownership if supporting this request
}

//This function is called when the device becomes initialized
void USBCBInitEP(void)
{
    // Enable the HID endpoint

USBEnableEndpoint(HID_EP,USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP
);

    // Re-arm the OUT endpoint for the next packet
    USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
}

// Send resume call-back
void USBCBSendResume(void)
{
    static WORD delay_count;

    // Verify that the host has armed us to perform remote wakeup.
    if(USBGetRemoteWakeupStatus() == FLAG_TRUE)
    {
        // Verify that the USB bus is suspended (before we send remote wakeup signalling).
        if(USBIsBusSuspended() == FLAG_TRUE)
        {
            USBMaskInterrupts();

            // Bring the clock speed up to normal running state
            USBCBWakeFromSuspend();
            USBSuspendControl = 0;
            USBBusIsSuspended = FLAG_FALSE;

            // Section 7.1.7.7 of the USB 2.0 specifications indicates a USB
            // device must continuously see 5ms+ of idle on the bus, before it sends
            // remote wakeup signalling.  One way to be certain that this parameter
            // gets met, is to add a 2ms+ blocking delay here (2ms plus at
            // least 3ms from bus idle to USBIsBusSuspended() == FLAG_TRUE, yeilds
            // 5ms+ total delay since start of idle).
            delay_count = 3600U;
            do
            {
                delay_count--;
            } while(delay_count);

            // Start RESUME signaling for 1-13 ms
            USBResumeControl = 1;
```

```
            delay_count = 1800U;

            do
            {
                delay_count--;
            } while(delay_count);
            USBResumeControl = 0;

            USBUnmaskInterrupts();
        }
    }
}

// USB callback function handler
BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_TRANSFER:
            // Application callback tasks and functions go here
            break;
        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            USBCBSuspend();
            break;
        case EVENT_RESUME:
            USBCBWakeFromSuspend();
            break;
        case EVENT_CONFIGURED:
            USBCBInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            USBCBStdSetDscHandler();
            break;
        case EVENT_EP0_REQUEST:
            USBCBCheckOtherReq();
            break;
        case EVENT_BUS_ERROR:
            USBCBErrorHandler();
            break;
        default:
            break;
    }
    return FLAG_TRUE;
}

#endif
```

Liite 3. HardwareProfile.h-tiedosto (Simon Inns 2011, WFF generic HID demo 3 firmware)


```
/********************************************************************
   HardwareProfile.h

   WFF USB Generic HID Demonstration 3
    usbGenericHidCommunication reference firmware 3_0_0_0
    Copyright (C) 2011 Simon Inns

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

   Email: simon.inns@gmail.com

********************************************************************/

#ifndef HARDWAREPROFILE_H
#define HARDWAREPROFILE_H

// USB stack hardware selection options ---------------------------------------
----------------------

// (This section is the set of definitions required by the MCHPFSUSB framework.)

// Uncomment the following define if you wish to use the self-power sense feature
// and define the port, pin and tris for the power sense pin below:
// #define USE_SELF_POWER_SENSE_IO
#define tris_self_power     TRISAbits.TRISA2
#if defined(USE_SELF_POWER_SENSE_IO)
   #define self_power          PORTAbits.RA2
#else
   #define self_power          1
#endif

// Uncomment the following define if you wish to use the bus-power sense feature
// and define the port, pin and tris for the power sense pin below:
//#define USE_USB_BUS_SENSE_IO
#define tris_usb_bus_sense  TRISAbits.TRISA1
#if defined(USE_USB_BUS_SENSE_IO)
   #define USB_BUS_SENSE       PORTAbits.RA1
#else
   #define USB_BUS_SENSE       1
#endif

// Uncomment the following line to make the output HEX of this project work with
the MCHPUSB Bootloader
//#define PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER
```

```
// Uncomment the following line to make the output HEX of this project work with
the HID Bootloader
#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
```



```
// Application specific hardware definitions --------------------------------
-----------------------

// Oscillator frequency (48Mhz with a 20Mhz external oscillator)
#define CLOCK_FREQ 48000000

// Device Vendor Indentifier (VID) (0x04D8 is Microchip's VID)
#define USB_VID  0x04D8

// Device Product Indentifier (PID) (0x0042)
#define USB_PID  0x0042

// Manufacturer string descriptor
#define MSDLENGTH  10
#define MSD   'S','i','m','o','n',' ','I','n','n','s'

// Product String descriptor
#define PSDLENGTH  20
#define PSD   'W','F','F',' ','G','e','n','e','r','i','c',' ','H','I','D','
','d','e','m','o'

// Device serial number string descriptor
#define DSNLENGTH  7
#define DSN   'W','F','F','_','3','.','0'

// Common useful definitions
#define INPUT_PIN 1
#define OUTPUT_PIN 0
#define FLAG_FALSE 0
#define FLAG_TRUE 1

// Comment out the following line if you do not want the debug
// feature of the firmware (saves code and RAM space when off)
//
// Note: if you use this feature you must compile with the large
// memory model on (for 24-bit pointers) so that the sprintf()
// function will work correctly.  If you do not require debug it's
// recommended that you compile with the small memory model and
// remove any references to <strings.h> and sprintf().
#define DEBUGON

// PIC to hardware pin mapping and control macros

// Led control macros
#define mInitStatusLeds()     LATA &= 0b00000001; TRISA &= 0b00000001;
#define mStatusLED0         LATAbits.LATA0
#define mStatusLED0_on()    mStatusLED0 = 1;
#define mStatusLED0_off()     mStatusLED0 = 0;
#define mStatusLED0_Toggle()  mStatusLED0 = !mStatusLED0;
#define mStatusLED0_Get()       mStatusLED0

// Switch macros
#define mInitAllSwitches()  TRISA=0b00101110;
#define mInitSwitch0()      TRISAbits.TRISA1=1;
#define sw0                 PORTAbits.RA1
#define sw1                 PORTAbits.RA2
#define sw2                 PORTAbits.RA3
#define sw3                 PORTAbits.RA5   //No TTL input in RA4
#endif
```

Liite 4. usb_descriptors.c-tiedosto (Simon Inns 2011, WFF generic HID demo 3 firmware)


```
/********************************************************************
   usb_descriptors.c

   WFF USB Generic HID Demonstration 3
    usbGenericHidCommunication reference firmware 3_0_0_0
    Copyright (C) 2011 Simon Inns

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

   Email: simon.inns@gmail.com

********************************************************************/

#ifndef USB_DESCRIPTORS_C
#define USB_DESCRIPTORS_C

// Local includes
#include "HardwareProfile.h"

// Microchip Application Library includes
#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"

// Device Descriptor
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,               // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,  // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,      // Max packet size for EP0, see usb_config.h
    USB_VID,            // Vendor ID
    USB_PID,            // Product ID
    0x0002,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x03,                   // Device serial number string index
    0x01                    // Number of possible configurations
};

// Configuration 1 Descriptor
ROM BYTE configDescriptor1[]=
{
    // Configuration Descriptor
    0x09,                       // Size of this descriptor in bytes (sizeof(USB_CFG_DSC))
    USB_DESCRIPTOR_CONFIGURATION,   // CONFIGURATION descriptor type
    0x29,0x00,                  // Total length of data for this cfg
    1,                          // Number of interfaces in this cfg
    1,                          // Index value of this configuration
    0,                          // Configuration string index
    _DEFAULT | _SELF,           // Attributes, see usb_device.h
    50,                         // Max power consumption (2X mA)

    // Interface Descriptor
    0x09,                       // Size of this descriptor in bytes (sizeof(USB_INTF_DSC))
    USB_DESCRIPTOR_INTERFACE,    // INTERFACE descriptor type
```

```
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
```

2(3)

```
    HID_INTF,               // Class code
    0,                      // Subclass code
    0,                      // Protocol code
    0,                      // Interface string index

    // HID Class-Specific Descriptor
    0x09,                   // Size of this descriptor in bytes (sizeof(USB_HID_DSC)+3)
    DSC_HID,                // HID descriptor type
    0x11,0x01,                  // HID Spec Release Number in BCD format (1.11)
    0x00,                   // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC,             // Number of class descriptors, see usbcfg.h
    DSC_RPT,                // Report descriptor type
    HID_RPT01_SIZE,0x00,        // Size of the report descriptor (sizeof(hid_rpt01))

    // Endpoint Descriptor
    0x07,                   // sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,    // Endpoint Descriptor
    HID_EP | _EP_IN,            // Endpoint Address
    _INTERRUPT,                 // Attributes
    0x40,0x00,                  // size
    0x01,                   // Interval

    // Endpoint Descriptor
    0x07,                   // sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,    // Endpoint Descriptor
    HID_EP | _EP_OUT,           // EndpointAddress
    _INTERRUPT,                 // Attributes
    0x40,0x00,                  // size
    0x01                    // Interval
};

// Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
}};

// Manufacturer string descriptor (set in HardwareProfile.h)
ROM struct{BYTE bLength;BYTE bDscType;WORD string[MSDLENGTH];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{MSD}};

// Product string descriptor (set in HardwareProfile.h)
ROM struct{BYTE bLength;BYTE bDscType;WORD string[PSDLENGTH];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{PSD}};

// Device serial number string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[DSNLENGTH];}sd003={
sizeof(sd003),USB_DESCRIPTOR_STRING,
{DSN}};

// Class specific (report) descriptor - HID
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
{
    0x06, 0x00, 0xFF,       // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,             // Usage (Vendor Usage 1)
    0xA1, 0x01,             // Collection (Application)
    0x19, 0x01,             // Usage Minimum
    0x29, 0x40,             // Usage Maximum  //64 input usages total (0x01 to 0x40)
    0x15, 0x01,             // Logical Minimum (bytes in the report may have min value = 0x00)
    0x25, 0x40,            //  Logical Maximum (bytes in the report may have max value = 0x00FF
= unsigned 255)
    0x75, 0x08,             // Report Size: 8-bit field size
    0x95, 0x40,             // Report Count: Make sixty-four 8-bit fields (the next time the
parser hits an "Input", "Output", or "Feature" item)
    0x81, 0x00,             // Input (Data, Array, Abs): Instantiates input packet fields based
on the above report size, count, logical min/max, and usage.
    0x19, 0x01,             // Usage Minimum
    0x29, 0x40,             // Usage Maximum  //64 output usages total (0x01 to 0x40)
```

```
    0x91, 0x00,             // Output (Data, Array, Abs): Instantiates output packet fields.
Uses same report size and count as "Input" fields, since nothing new/different was specified to
the parser since the "Input" item.
    0xC0}                   // End Collection
};
```



```
// Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

// Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002,
    (ROM BYTE *ROM)&sd003
};

#endif
```

Liite 5. deviceDiscovery.cs (Simon Inns 2011, WFF Generic HID Demo)


```
//-------------------------------------------------------------------------
//
//  deviceDiscovery.cs
//
//  USB Generic HID Communications 3_0_0_0
//
//  A class for communicating with Generic HID USB devices
//  Copyright (C) 2011 Simon Inns
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
//  Web:    http://www.waitingforfriday.com
//  Email:  simon.inns@gmail.com
//
//-------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Runtime.InteropServices;
using Microsoft.Win32.SafeHandles;

// The following namespace allows debugging output (when compiled in debug mode)
using System.Diagnostics;

namespace usbGenericHidCommunications
    {
    public partial class usbGenericHidCommunication
        {
        /// <summary>
        /// Find all devices with the HID GUID attached to the system
        /// </summary>
        /// <remarks>This method searches for all devices that have the correct HID GUID and
        /// returns an array of matching device paths</remarks>
        private bool findHidDevices(ref String[] listOfDevicePathNames, ref int
numberOfDevicesFound)
            {
            Debug.WriteLine("usbGenericHidCommunication:findHidDevices() -> Method called");
            // Detach the device if it's currently attached
            if (isDeviceAttached) detachUsbDevice();

            // Initialise the internal variables required for performing the search
            Int32 bufferSize = 0;
            IntPtr detailDataBuffer = IntPtr.Zero;
            Boolean deviceFound;
            IntPtr deviceInfoSet = new System.IntPtr();
            Boolean lastDevice = false;
            Int32 listIndex = 0;
            SP_DEVICE_INTERFACE_DATA deviceInterfaceData = new SP_DEVICE_INTERFACE_DATA();
            Boolean success;

            // Get the required GUID
            System.Guid systemHidGuid = new Guid();
            HidD_GetHidGuid(ref systemHidGuid);
            Debug.WriteLine(string.Format("usbGenericHidCommunication:findHidDevices() ->
Fetched GUID for HID devices ({0})", systemHidGuid.ToString()));

            try
                {
```

```
                // Here we populate a list of plugged-in devices matching our class GUID
(DIGCF_PRESENT specifies that the list
                // should only contain devices which are plugged in)
                Debug.WriteLine("usbGenericHidCommunication:findHidDevices() -> Using
SetupDiGetClassDevs to get all devices with the correct GUID");
                deviceInfoSet = SetupDiGetClassDevs(ref systemHidGuid, IntPtr.Zero, IntPtr.Zero,
DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

                // Reset the deviceFound flag and the memberIndex counter
                deviceFound = false;
                listIndex = 0;

                deviceInterfaceData.cbSize = Marshal.SizeOf(deviceInterfaceData);

                // Look through the retrieved list of class GUIDs looking for a match on our
interface GUID
                do
                    {
                    //Debug.WriteLine("usbGenericHidCommunication:findHidDevices() ->
Enumerating devices");
                    success = SetupDiEnumDeviceInterfaces
                        (deviceInfoSet,
                        IntPtr.Zero,
                        ref systemHidGuid,
                        listIndex,
                        ref deviceInterfaceData);

                    if (!success)
                        {
                        //Debug.WriteLine("usbGenericHidCommunication:findHidDevices() -> No
more devices left - giving up");
                        lastDevice = true;
                        }
                    else
                        {
                        // The target device has been found, now we need to retrieve the device
path so we can open
                        // the read and write handles required for USB communication

                        // First call is just to get the required buffer size for the real
request
                        success = SetupDiGetDeviceInterfaceDetail
                            (deviceInfoSet,
                            ref deviceInterfaceData,
                            IntPtr.Zero,
                            0,
                            ref bufferSize,
                            IntPtr.Zero);

                        // Allocate some memory for the buffer
                        detailDataBuffer = Marshal.AllocHGlobal(bufferSize);
                        Marshal.WriteInt32(detailDataBuffer, (IntPtr.Size == 4) ? (4 +
Marshal.SystemDefaultCharSize) : 8);

                        // Second call gets the detailed data buffer
                        //Debug.WriteLine("usbGenericHidCommunication:findHidDevices() ->
Getting details of the device");
                        success = SetupDiGetDeviceInterfaceDetail
                            (deviceInfoSet,
                            ref deviceInterfaceData,
                            detailDataBuffer,
                            bufferSize,
                            ref bufferSize,
                            IntPtr.Zero);

                        // Skip over cbsize (4 bytes) to get the address of the devicePathName.
                        IntPtr pDevicePathName = new IntPtr(detailDataBuffer.ToInt32() + 4);

                        // Get the String containing the devicePathName.
                        listOfDevicePathNames[listIndex] =
Marshal.PtrToStringAuto(pDevicePathName);
```

```
//Debug.WriteLine(string.Format("usbGenericHidCommunication:findHidDevices() -> Found matching
device (memberIndex {0})", memberIndex));
                        deviceFound = true;
                        }
                    listIndex = listIndex + 1;
                    }
                while (!((lastDevice == true)));
                }
            catch (Exception)
                {
                // Something went badly wrong... output some debug and return false to indicated
device discovery failure
                Debug.WriteLine("usbGenericHidCommunication:findHidDevices() -> EXCEPTION:
Something went south whilst trying to get devices with matching GUIDs - giving up!");
                return false;
                }
            finally
                {
                // Clean up the unmanaged memory allocations
                if (detailDataBuffer != IntPtr.Zero)
                    {
                    // Free the memory allocated previously by AllocHGlobal.
                    Marshal.FreeHGlobal(detailDataBuffer);
                    }

                if (deviceInfoSet != IntPtr.Zero)
                    {
                    SetupDiDestroyDeviceInfoList(deviceInfoSet);
                    }
                }

            if (deviceFound)
                {
                Debug.WriteLine(string.Format("usbGenericHidCommunication:findHidDevices() ->
Found {0} devices with matching GUID", listIndex - 1));
                numberOfDevicesFound = listIndex - 2;
                }
            else Debug.WriteLine("usbGenericHidCommunication:findHidDevices() -> No matching
devices found");

            return deviceFound;
            }

        /// <summary>
        /// Find the target device based on the VID and PID
        /// </summary>
        /// <remarks>This method used the 'findHidDevices' to fetch a list of attached HID
devices
        /// then it searches through the list looking for a HID device with the correct VID and
        /// PID</remarks>
        public void findTargetDevice()
            {
            Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> Method called");

            bool deviceFoundByGuid = false;
            String[] listOfDevicePathNames = new String[128]; // 128 is the maximum number of
USB devices allowed on a single host
            int listIndex = 0;
            bool success = false;
            bool isDeviceDetected;
            int numberOfDevicesFound = 0;

            try
                {
                // Reset the device detection flag
                isDeviceDetected = false;

                // Get all the devices with the correct HID GUID
                deviceFoundByGuid = findHidDevices(ref listOfDevicePathNames, ref
numberOfDevicesFound);

                if (deviceFoundByGuid)
                    {
```

```
                    Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> Devices
with matching GUID found...");
                    listIndex = 0;

                    do
                        {

Debug.WriteLine(string.Format("usbGenericHidCommunication:findTargetDevice() -> Performing
CreateFile to listIndex {0}", listIndex));
                        deviceInformation.hidHandle =
CreateFile(listOfDevicePathNames[listIndex], 0, FILE_SHARE_READ | FILE_SHARE_WRITE, IntPtr.Zero,
OPEN_EXISTING, 0, 0);

                        if (!deviceInformation.hidHandle.IsInvalid)
                            {
                            deviceInformation.attributes.size =
Marshal.SizeOf(deviceInformation.attributes);
                            success = HidD_GetAttributes(deviceInformation.hidHandle, ref
deviceInformation.attributes);

                            if (success)
                                {

Debug.WriteLine(string.Format("usbGenericHidCommunication:findTargetDevice() -> Found device
with VID {0}, PID {1} and Version number {2}",
                                    Convert.ToString(deviceInformation.attributes.vendorId, 16),
                                    Convert.ToString(deviceInformation.attributes.productId,
16),
                                    Convert.ToString(deviceInformation.attributes.versionNumber,
16)));

                                //  Do the VID and PID of the device match our target device?
                                if ((deviceInformation.attributes.vendorId ==
deviceInformation.targetVid) &&
                                    (deviceInformation.attributes.productId ==
deviceInformation.targetPid))
                                    {
                                    // Matching device found

Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> Device with matching VID and
PID found!");
                                    isDeviceDetected = true;

                                    // Store the device's pathname in the device information
                                    deviceInformation.devicePathName =
listOfDevicePathNames[listIndex];
                                    }
                                else
                                    {
                                    // Wrong device, close the handle

Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> Device didn't match...
Continuing...");
                                    isDeviceDetected = false;
                                    deviceInformation.hidHandle.Close();
                                    }
                                }
                            else
                                {
                                //  Something went rapidly south...  give up!
                                Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -
> Something bad happened - couldn't fill the HIDD_ATTRIBUTES, giving up!");
                                isDeviceDetected = false;
                                deviceInformation.hidHandle.Close();
                                }
                            }

                        //  Move to the next device, or quit if there are no more devices to
examine
                        listIndex++;
                        }
                    while (!((isDeviceDetected || (listIndex == numberOfDevicesFound + 1))));
                    }
```

```
                // If we found a matching device then we need discover more details about the
attached device
                // and then open read and write handles to the device to allow communication
                if (isDeviceDetected)
                    {
                    // Query the HID device's capabilities (primarily we are only really
interested in the
                    // input and output report byte lengths as this allows us to validate
information sent
                    // to and from the device does not exceed the devices capabilities.
                    //
                    // We could determine the 'type' of HID device here too, but since this
class is only
                    // for generic HID communication we don't care...
                    queryDeviceCapabilities();

                    if (success)
                        {
                        // Open the readHandle to the device
                        Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() ->
Opening a readHandle to the device");
                        deviceInformation.readHandle = CreateFile(
                            deviceInformation.devicePathName,
                            GENERIC_READ,
                            FILE_SHARE_READ | FILE_SHARE_WRITE,
                            IntPtr.Zero, OPEN_EXISTING,
                            FILE_FLAG_OVERLAPPED,
                            0);

                        // Did we open the readHandle successfully?
                        if (deviceInformation.readHandle.IsInvalid)
                            {
                            Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() ->
Unable to open a readHandle to the device!");
                            }
                        else
                            {
                            Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() ->
Opening a writeHandle to the device");
                            deviceInformation.writeHandle = CreateFile(
                                deviceInformation.devicePathName,
                                GENERIC_WRITE,
                                FILE_SHARE_READ | FILE_SHARE_WRITE,
                                IntPtr.Zero,
                                OPEN_EXISTING,
                                0,
                                0);

                            // Did we open the writeHandle successfully?
                            if (deviceInformation.writeHandle.IsInvalid)
                                {
                                Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -
> Unable to open a writeHandle to the device!");

                                // Attempt to close the writeHandle
                                deviceInformation.writeHandle.Close();
                                }
                            else
                                {
                                // Device is now discovered and ready for use, update the status
                                isDeviceAttached = true;
                                onUsbEvent(EventArgs.Empty); // Throw an event
                                }
                            }
                        }
                    }
                else
                    {
                    //  The device wasn't detected.
                    Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> No
matching device found!");
                    }
                }
            catch (Exception)
```

```
                {
                Debug.WriteLine("usbGenericHidCommunication:findTargetDevice() -> EXCEPTION:
Unknown - device not found");
                isDeviceDetected = false;
                }
            }

        /// <summary>
        /// queryDeviceCapabilities - Used the hid.dll function to query the capabilities
        /// of the target device.
        /// </summary>
        private void queryDeviceCapabilities()
            {
            IntPtr preparsedData = new System.IntPtr();
            Int32 result = 0;
            Boolean success = false;

            try
                {
                // Get the preparsed data from the HID driver
                success = HidD_GetPreparsedData(deviceInformation.hidHandle, ref preparsedData);

                // Get the HID device's capabilities
                result = HidP_GetCaps(preparsedData, ref deviceInformation.capabilities);
                if ((result != 0))
                    {
                    Debug.WriteLine("usbGenericHidCommunication:queryDeviceCapabilities() ->
Device query results:");

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Usage: {0}",
                        Convert.ToString(deviceInformation.capabilities.usage, 16)));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->     Usage
Page: {0}",
                        Convert.ToString(deviceInformation.capabilities.usagePage, 16)));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->     Input
Report Byte Length: {0}",
                        deviceInformation.capabilities.inputReportByteLength));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Output Report Byte Length: {0}",
                        deviceInformation.capabilities.outputReportByteLength));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Feature Report Byte Length: {0}",
                        deviceInformation.capabilities.featureReportByteLength));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Link Collection Nodes: {0}",
                        deviceInformation.capabilities.numberLinkCollectionNodes));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Input Button Caps: {0}",
                        deviceInformation.capabilities.numberInputButtonCaps));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Input Value Caps: {0}",
                        deviceInformation.capabilities.numberInputValueCaps));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Input Data Indices: {0}",
                        deviceInformation.capabilities.numberInputDataIndices));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Output Button Caps: {0}",
                        deviceInformation.capabilities.numberOutputButtonCaps));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Output Value Caps: {0}",
                        deviceInformation.capabilities.numberOutputValueCaps));
```

```
Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Output Data Indices: {0}",
                        deviceInformation.capabilities.numberOutputDataIndices));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Feature Button Caps: {0}",
                        deviceInformation.capabilities.numberFeatureButtonCaps));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Feature Value Caps: {0}",
                        deviceInformation.capabilities.numberFeatureValueCaps));

Debug.WriteLine(string.Format("usbGenericHidCommunication:queryDeviceCapabilities() ->
Number of Feature Data Indices: {0}",
                        deviceInformation.capabilities.numberFeatureDataIndices));
                    }
                }
            catch (Exception)
                {
                // Something went badly wrong... this shouldn't happen, so we throw an exception

Debug.WriteLine("usbGenericHidCommunication:queryDeviceCapabilities() -> EXECEPTION: An
unrecoverable error has occurred!");
                throw;
                }
            finally
                {
                // Free up the memory before finishing
                if (preparsedData != IntPtr.Zero)
                    {
                    success = HidD_FreePreparsedData(preparsedData);
                    }
                }
            }
        }
    }
```

Liite 6. deviceNotifications.cs (Simon Inns 2011, WFF Generic HID Demo)


```
//-------------------------------------------------------------------------
//
//  deviceNotifications.cs
//
//  USB Generic HID Communications 3_0_0_0
//
//  A class for communicating with Generic HID USB devices
//  Copyright (C) 2011 Simon Inns
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
//  Web:    http://www.waitingforfriday.com
//  Email:  simon.inns@gmail.com
//
//-------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Windows.Forms;
using System.Runtime.InteropServices;

// The following namespace allows debugging output (when compiled in debug mode)
using System.Diagnostics;

namespace usbGenericHidCommunications
    {
    /// <summary>
    /// This partial class contains the methods required for detecting when
    /// the USB device is attached or detached.
    /// </summary>
    public partial class usbGenericHidCommunication : Control
        {

        /// <summary>
        /// Create a delegate for the USB event handler
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        public delegate void usbEventsHandler(object sender, EventArgs e);

        /// <summary>
        /// Define the event
        /// </summary>
        public event usbEventsHandler usbEvent;

        /// <summary>
        /// The usb event thrower
        /// </summary>
        /// <param name="e"></param>
        protected virtual void onUsbEvent(EventArgs e)
            {
            if (usbEvent != null)
                {
                Debug.WriteLine("usbGenericHidCommunications:onUsbEvent() -> Throwing a USB
event to a listener");
                usbEvent(this, e);
                }
```

```
            else Debug.WriteLine("usbGenericHidCommunications:onUsbEvent() -> Attempted to throw
a USB event, but no one was listening");
            }

        /// <summary>
        /// isNotificationForTargetDevice - Compares the target devices pathname against the
        /// pathname of the device which caused the event message
        /// </summary>
        private Boolean isNotificationForTargetDevice(Message m)
            {
            Int32 stringSize;

            try
                {
                DEV_BROADCAST_DEVICEINTERFACE_1 devBroadcastDeviceInterface = new
DEV_BROADCAST_DEVICEINTERFACE_1();
                DEV_BROADCAST_HDR devBroadcastHeader = new DEV_BROADCAST_HDR();

                Marshal.PtrToStructure(m.LParam, devBroadcastHeader);

                // Is the notification event concerning a device interface?
                if ((devBroadcastHeader.dbch_devicetype == DBT_DEVTYP_DEVICEINTERFACE))
                    {
                    // Get the device path name of the affected device
                    stringSize = System.Convert.ToInt32((devBroadcastHeader.dbch_size - 32) /
2);
                    devBroadcastDeviceInterface.dbcc_name = new Char[stringSize + 1];
                    Marshal.PtrToStructure(m.LParam, devBroadcastDeviceInterface);
                    String deviceNameString = new String(devBroadcastDeviceInterface.dbcc_name,
0, stringSize);

                    // Compare the device name with our target device's pathname (strings are
moved to lower case
                    // using en-US to ensure case insensitivity accross different regions)
                    if ((String.Compare(deviceNameString.ToLower(new
System.Globalization.CultureInfo("en-US")),
                        deviceInformation.devicePathName.ToLower(new
System.Globalization.CultureInfo("en-US")), true) == 0)) return true;
                    else return false;
                    }
                }
            catch (Exception)
                {
                Debug.WriteLine("usbGenericHidCommunication:isNotificationForTargetDevice() ->
EXCEPTION: An unknown exception has occured!");
                return false;
                }
            return false;
            }

        /// <summary>
        /// registerForDeviceNotification - registers the window (identified by the
windowHandle) for
        /// device notification messages from Windows
        /// </summary>
        public Boolean registerForDeviceNotifications(IntPtr windowHandle)
            {
            Debug.WriteLine("usbGenericHidCommunication:registerForDeviceNotifications() ->
Method called");

            // A DEV_BROADCAST_DEVICEINTERFACE header holds information about the request.
            DEV_BROADCAST_DEVICEINTERFACE devBroadcastDeviceInterface = new
DEV_BROADCAST_DEVICEINTERFACE();
            IntPtr devBroadcastDeviceInterfaceBuffer = IntPtr.Zero;
            Int32 size = 0;

            // Get the required GUID
            System.Guid systemHidGuid = new Guid();
            HidD_GetHidGuid(ref systemHidGuid);

            try
                {
                // Set the parameters in the DEV_BROADCAST_DEVICEINTERFACE structure.
                size = Marshal.SizeOf(devBroadcastDeviceInterface);
```

```
                devBroadcastDeviceInterface.dbcc_size = size;
                devBroadcastDeviceInterface.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
                devBroadcastDeviceInterface.dbcc_reserved = 0;
                devBroadcastDeviceInterface.dbcc_classguid = systemHidGuid;

                devBroadcastDeviceInterfaceBuffer = Marshal.AllocHGlobal(size);
                Marshal.StructureToPtr(devBroadcastDeviceInterface,
devBroadcastDeviceInterfaceBuffer, true);

                // Register for notifications and store the returned handle
                deviceInformation.deviceNotificationHandle =
RegisterDeviceNotification(windowHandle, devBroadcastDeviceInterfaceBuffer,
DEVICE_NOTIFY_WINDOW_HANDLE);
                Marshal.PtrToStructure(devBroadcastDeviceInterfaceBuffer,
devBroadcastDeviceInterface);

                if ((deviceInformation.deviceNotificationHandle.ToInt32() ==
IntPtr.Zero.ToInt32()))
                    {
                    Debug.WriteLine("usbGenericHidCommunication:registerForDeviceNotifications()
-> Notification registration failed");
                    return false;
                    }
                else
                    {
                    Debug.WriteLine("usbGenericHidCommunication:registerForDeviceNotifications()
-> Notification registration succeded");
                    return true;
                    }
                }
            catch (Exception)
                {
                Debug.WriteLine("usbGenericHidCommunication:registerForDeviceNotifications() ->
EXCEPTION: An unknown exception has occured!");
                }
            finally
                {
                // Free the memory allocated previously by AllocHGlobal.
                if (devBroadcastDeviceInterfaceBuffer != IntPtr.Zero)
                    Marshal.FreeHGlobal(devBroadcastDeviceInterfaceBuffer);
                }
            return false;
            }

        /// <summary>
        /// handleDeviceNotificationMessages - this method examines any windows devices messages
that are
        /// received to check if they are relevant to our target USB device.  If so the method
takes the
        /// correct action dependent on the message type.
        /// </summary>
        /// <param name="m"></param>
        public void handleDeviceNotificationMessages(Message m)
            {
            //Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() ->
Method called");

            // Make sure this is a device notification
            if (m.Msg != WM_DEVICECHANGE) return;

            Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() ->
Device notification received");

            try
                {
                switch (m.WParam.ToInt32())
                    {
                    // Device attached
                    case DBT_DEVICEARRIVAL:

Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() -> New device
attached");
                        // If our target device is not currently attached, this could be our
device, so we attempt to find it.
```

4(4)

```
                        if (!isDeviceAttached)
                            {
                            findTargetDevice();
                            onUsbEvent(EventArgs.Empty); // Generate an event
                            }
                        break;

                    // Device removed
                    case DBT_DEVICEREMOVECOMPLETE:

Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() -> A device has
been removed");

                        // Was this our target device?
                        if (isNotificationForTargetDevice(m))
                            {
                            // If so detach the USB device.

Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() -> The target USB
device has been removed - detaching...");
                            detachUsbDevice();
                            onUsbEvent(EventArgs.Empty); // Generate an event
                            }
                        break;

                    // Other message
                    default:

Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() -> Unknown
notification message");
                        break;
                    }
                }
            catch (Exception)
                {
                Debug.WriteLine("usbGenericHidCommunication:handleDeviceNotificationMessages() -
> EXCEPTION: An unknown exception has occured!");
                }
            }

        /// <summary>
        /// WndProc - This method overrides the WinProc in order to pass notification messages
        /// to the base WndProc
        /// </summary>
        /// <param name="m"></param>
        protected override void WndProc(ref Message m)
            {
            handleDeviceNotificationMessages(m);

            // Pass the notification message to the base WndProc
            base.WndProc(ref m);
            }
        }
    }
```

Liite 7. deviceCommunication.cs-tiedosto (Simon Inns 2011, WFF Generic HID Demo)


```
//--------------------------------------------------------------------------
//
//  deviceCommunication.cs
//
//  USB Generic HID Communications 3_0_0_0
//
//  A class for communicating with Generic HID USB devices
//  Copyright (C) 2011 Simon Inns
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
//  Web:    http://www.waitingforfriday.com
//  Email:  simon.inns@gmail.com
//
//--------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Runtime.InteropServices;
using Microsoft.Win32.SafeHandles;
using System.Threading;

// The following namespace allows debugging output (when compiled in debug mode)
using System.Diagnostics;

namespace usbGenericHidCommunications
    {
    public partial class usbGenericHidCommunication
        {
        #region outputToDeviceRegion

        /// <summary>
        /// writeRawReportToDevice - Writes a report to the device using interrupt transfer.
        /// Note: this method performs no checking on the buffer.  The first byte must
        /// always be zero (or the write will fail!) and the second byte should be the
        /// command number for the USB device firmware.
        /// </summary>
        protected bool writeRawReportToDevice(Byte[] outputReportBuffer)
            {
            bool success = false;
            int numberOfBytesWritten = 0;

            // Make sure a device is attached
            if (!isDeviceAttached)
                {
                Debug.WriteLine("usbGenericHidCommunication:writeReportToDevice(): -> No device
attached!");
                return success;
                }

            try
                {
                // Set an output report via interrupt to the device
                success = WriteFile(
                    deviceInformation.writeHandle,
                    outputReportBuffer,
                    outputReportBuffer.Length,
```

```
                    ref numberOfBytesWritten,
                    IntPtr.Zero);

                if (success) Debug.WriteLine("usbGenericHidCommunication:writeReportToDevice():
-> Write report succeeded");
                else Debug.WriteLine("usbGenericHidCommunication:writeReportToDevice(): -> Write
report failed!");

                return success;
                }
            catch (Exception)
                {
                // An error - send out some debug and return failure
                Debug.WriteLine("usbGenericHidCommunication:writeReportToDevice(): -> EXCEPTION:
When attempting to send an output report");
                return false;
                }
            }
        #endregion

        #region inputFromDeviceRegion
        /// <summary>
        /// readRawReportFromDevice - Reads a raw report from the device with timeout handling
        /// Note: This method performs no checking on the buffer.  The first byte returned is
        /// usually zero, the second byte is the command that the USB firmware is replying to.
        /// The other 63 bytes are (possibly) data.
        ///
        /// The maximum report size will be determind by the length of the inputReportBuffer.
        /// </summary>
        private bool readRawReportFromDevice(ref Byte[] inputReportBuffer, ref int
numberOfBytesRead)
            {
            IntPtr eventObject = IntPtr.Zero;
            NativeOverlapped hidOverlapped = new NativeOverlapped();
            IntPtr nonManagedBuffer = IntPtr.Zero;
            IntPtr nonManagedOverlapped = IntPtr.Zero;
            Int32 result = 0;
            bool success;

            // Make sure a device is attached
            if (!isDeviceAttached)
                {
                Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice(): -> No device
attached!");
                return false;
                }

            try
                {
                // Prepare an event object for the overlapped ReadFile
                eventObject = CreateEvent(IntPtr.Zero, false, false, "");

                hidOverlapped.OffsetLow = 0;
                hidOverlapped.OffsetHigh = 0;
                hidOverlapped.EventHandle = eventObject;

                // Allocate memory for the unmanaged input buffer and overlap structure.
                nonManagedBuffer = Marshal.AllocHGlobal(inputReportBuffer.Length);
                nonManagedOverlapped = Marshal.AllocHGlobal(Marshal.SizeOf(hidOverlapped));
                Marshal.StructureToPtr(hidOverlapped, nonManagedOverlapped, false);

                // Read the input report buffer
                Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice(): ->
Attempting to ReadFile");
                success = ReadFile(
                    deviceInformation.readHandle,
                    nonManagedBuffer,
                    inputReportBuffer.Length,
                    ref numberOfBytesRead,
                    nonManagedOverlapped);

                if (!success)
                    {
                    // We are now waiting for the FileRead to complete
```

```
                    Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice(): ->
ReadFile started, waiting for completion...");

                    // Wait a maximum of 3 seconds for the FileRead to complete
                    result = WaitForSingleObject(eventObject, 3000);

                    switch (result)
                        {
                        // Has the ReadFile completed successfully?
                        case (System.Int32)WAIT_OBJECT_0:
                            success = true;

                            // Get the number of bytes transferred
                            GetOverlappedResult(deviceInformation.readHandle,
nonManagedOverlapped, ref numberOfBytesRead, false);


Debug.WriteLine(string.Format("usbGenericHidCommunication:readReportFromDevice(): -> ReadFile
successful (overlapped) {0} bytes read", numberOfBytesRead));
                            break;

                        // Did the FileRead operation timeout?
                        case WAIT_TIMEOUT:
                            success = false;

                            // Cancel the ReadFile operation
                            CancelIo(deviceInformation.readHandle);
                            if (!deviceInformation.hidHandle.IsInvalid)
deviceInformation.hidHandle.Close();
                            if (!deviceInformation.readHandle.IsInvalid)
deviceInformation.readHandle.Close();
                            if (!deviceInformation.writeHandle.IsInvalid)
deviceInformation.writeHandle.Close();

                            // Detach the USB device to try to get us back in a known state
                            detachUsbDevice();

                            Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice():
-> ReadFile timedout! USB device detached");
                            break;

                        // Some other unspecified error has occurred?
                        default:
                            success = false;

                            // Cancel the ReadFile operation

                            // Detach the USB device to try to get us back in a known state
                            detachUsbDevice();

                            Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice():
-> ReadFile unspecified error! USB device detached");
                            break;
                        }
                    }
                    if (success)
                        {
                        // Report receieved correctly, copy the unmanaged input buffer over to
the managed buffer
                        Marshal.Copy(nonManagedBuffer, inputReportBuffer, 0, numberOfBytesRead);

Debug.WriteLine(string.Format("usbGenericHidCommunication:readReportFromDevice(): -> ReadFile
successful {0} bytes read", numberOfBytesRead));
                        }
                }
            catch (Exception)
                {
                // An error - send out some debug and return failure
                Debug.WriteLine("usbGenericHidCommunication:readReportFromDevice(): ->
EXCEPTION: When attempting to receive an input report");
                return false;
                }

            // Release non-managed objects before returning
```

```
            Marshal.FreeHGlobal(nonManagedBuffer);
            Marshal.FreeHGlobal(nonManagedOverlapped);

            // Close the file handle to release the object
            CloseHandle(eventObject);

            return success;
            }

        /// <summary>
        /// readSingleReportFromDevice - Reads a single report packet from the USB device.
        /// The size of the passed inputReportBuffer must be correct for the device, so
        /// this method checks the passed buffer's size against the input report size
        /// discovered by the device enumeration.
        /// </summary>
        /// <param name="inputReportBuffer"></param>
        /// <returns></returns>
        protected bool readSingleReportFromDevice(ref Byte[] inputReportBuffer)
            {
            bool success;
            int numberOfBytesRead = 0;

            // The size of our inputReportBuffer must be at least the same size as the input
report.
            if (inputReportBuffer.Length !=
(int)deviceInformation.capabilities.inputReportByteLength)
                {
                // inputReportBuffer is not the right length!
                Debug.WriteLine(
                    "usbGenericHidCommunication:readSingleReportFromDevice(): -> ERROR: The
referenced inputReportBuffer size is incorrect for the input report size!");
                return false;
                }

            // The readRawReportFromDevice method will fill the passed readBuffer or return
false
            success = readRawReportFromDevice(ref inputReportBuffer, ref numberOfBytesRead);

            return success;
            }

        /// <summary>
        /// readMultipleReportsFromDevice - Attempts to retrieve multiple reports from the
device in
        /// a single read.  This action can be block the form execution if you request too much
data.
        /// If you need more data from the device and want to avoid any blocking you will have
to make
        /// multiple commands to the device and deal with the multiple requests and responses in
your
        /// application.
        /// </summary>
        /// <param name="inputReportBuffer"></param>
        /// <param name="numberOfReports"></param>
        /// <returns></returns>
        protected bool readMultipleReportsFromDevice(ref Byte[] inputReportBuffer, int
numberOfReports)
            {
            bool success = false;
            int numberOfBytesRead = 0;
            long pointerToBuffer = 0;

            // Define a temporary buffer for assembling partial data reads into the completed
inputReportBuffer
            Byte[] temporaryBuffer = new Byte[inputReportBuffer.Length];

            // Range check the number of reports
            if (numberOfReports == 0)
                {
                Debug.WriteLine(
                    "usbGenericHidCommunication:readMultipleReportsFromDevice(): -> ERROR: You
cannot request 0 reports!");
                return false;
                }
```

```
            if (numberOfReports > 128)
                {
                Debug.WriteLine(
                    "usbGenericHidCommunication:readMultipleReportsFromDevice(): -> ERROR:
Reference application testing does not verify the code for more than 128 reports");
                return false;
                }

            // The size of our inputReportBuffer must be at least the same size as the input
report multiplied by the number of reports requested.
            if (inputReportBuffer.Length !=
((int)deviceInformation.capabilities.inputReportByteLength * numberOfReports))
                {
                // inputReportBuffer is not the right length!
                Debug.WriteLine(
                    "usbGenericHidCommunication:readMultipleReportsFromDevice(): -> ERROR: The
referenced inputReportBuffer size is incorrect for the number of input reports requested!");
                return false;
                }

            // The readRawReportFromDevice method will fill the passed read buffer or return
false
            while (pointerToBuffer != ((int)deviceInformation.capabilities.inputReportByteLength
* numberOfReports))
                {
                Debug.WriteLine(
                    "usbGenericHidCommunication:readMultipleReportsFromDevice(): -> Reading from
device...");
                success = readRawReportFromDevice(ref temporaryBuffer, ref numberOfBytesRead);

                // Was the read successful?
                if (!success)
                    {
                    return false;
                    }

                // Copy the received data into the referenced input buffer
                Array.Copy(temporaryBuffer, 0, inputReportBuffer, pointerToBuffer,
(long)numberOfBytesRead);
                pointerToBuffer += (long)numberOfBytesRead;
                }

            return success;
            }

        #endregion
        }
    }
```

Liite 8. Form1.cs-tiedosto (Simon Inns 2011, WFF Generic HID Demo)


```
//-------------------------------------------------------------------------
//
//  Form1.cs
//
//  USB Generic HID Demonstration 3_0_0_0
//
//  A reference test application for the usbGenericHidCommunications library
//  Copyright (C) 2011 Simon Inns
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
//  Web:    http://www.waitingforfriday.com
//  Email:  simon.inns@gmail.com
//
//-------------------------------------------------------------------------

//
//  Current reference class library version:
//  usbGenericHidCommunications class library version 3.0.0.0
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace WFF_Generic_HID_Demo_3
    {
    public partial class Form1 : Form
        {

        private const int APPCOMMAND_VOLUME_MUTE = 0x80000;
        private const int APPCOMMAND_VOLUME_UP = 0xA0000;
        private const int APPCOMMAND_VOLUME_DOWN = 0x90000;
        private const int WM_APPCOMMAND = 0x319;

        [DllImport("user32.dll")]
        public static extern IntPtr SendMessageW(IntPtr hWnd, int Msg,
            IntPtr wParam, IntPtr lParam);

        public Form1()
            {

            InitializeComponent();

            // Create the USB reference device object (passing VID and PID)
            theUsbDemoDevice = new usbDemoDevice(0x04D8, 0x0042);

            // Add a listener for usb events
            theUsbDemoDevice.usbEvent += new usbDemoDevice.usbEventsHandler(usbEvent_receiver);

            // Perform an initial search for the target device
            theUsbDemoDevice.findTargetDevice();
            }

        // Create an instance of the USB reference device
```

```
private usbDemoDevice theUsbDemoDevice;

// Listener for USB events
private void usbEvent_receiver(object o, EventArgs e)
    {
    // Check the status of the USB device and update the form accordingly
    if (theUsbDemoDevice.isDeviceAttached)
        {
        // Device is currently attached

        // Update the status label
        this.toolStripStatusLabel1.Text = "USB Device is attached";

        // Update the form
        this.toggleLedStateButton.Enabled = true;
        }
    else
        {
        // Device is currently unattached

        // Update the status label
        this.toolStripStatusLabel1.Text = "USB Device is detached";

        // Update the form
        this.toggleLedStateButton.Enabled = false;
        }
    }

// Link label to website clicked
private void linkLabel1_LinkClicked_1(object sender, LinkLabelLinkClickedEventArgs e)
    {
    // Specify that the link was visited.
    this.linkLabel1.LinkVisited = true;

    // Navigate to a URL.
    System.Diagnostics.Process.Start("http://www.waitingforfriday.com");
    }

// Toggle LED state button pressed
private void toggleLedStateButton_Click(object sender, EventArgs e)
    {
    if (theUsbDemoDevice.isDeviceAttached)
        {
        theUsbDemoDevice.toggleLedState();
        }
    }

// Timer 1 has ticked, poll the USB device for status
private void timer1_Tick(object sender, EventArgs e)
    {
    if (theUsbDemoDevice.isDeviceAttached)
        {
        // Read the push button state
        int buttonState = theUsbDemoDevice.readPushButtonState();

        if (buttonState > 0)
            {
                this.pushButtonStateLabel.Text = "Button " + buttonState + " pressed";

                switch (buttonState)
                {
                    case 1:
                        button1Pressed();
                        break;
                    case 2:
                        button2Pressed();
                        break;
                    case 3:
                        button3Pressed();
                        break;
                    case 4:
                        button4Pressed();
                        break;
                    default:
                        break;
```

```
                    }
                }
            if (buttonState == 0) this.pushButtonStateLabel.Text = "Button not pressed";

            // Read the LED state
            bool ledState = theUsbDemoDevice.readLedState();

            if (ledState == true) this.ledStateLabel.Text = "LED is on";
            else this.ledStateLabel.Text = "LED is off";
            }
        }

    // Collect debug timer has ticked
    private void debugCollectionTimer_Tick(object sender, EventArgs e)
        {
        String debugString;

        // Only collect debug if the device is attached
        if (theUsbDemoDevice.isDeviceAttached)
            {
            // Collect the debug information from the device
            debugString = theUsbDemoDevice.collectDebug();

            // Display the debug information
            if (debugString != String.Empty)
                {
                this.debugTextBox.AppendText(debugString);
                }
            }
        else
            {
            // Clear the debug window
            this.debugTextBox.Clear();
            }
        }

    private void button1Pressed()
        {
            SendMessageW(
                this.Handle,                    // Pääikkunan kahva
                WM_APPCOMMAND,                  // Viestiarvo (0x319)
                this.Handle,                    // Mahdollinen lisäinfo
                (IntPtr)APPCOMMAND_VOLUME_MUTE  // Komennon parametriarvo (0x80000)
                );
        }

    private void button2Pressed()
        {
            SendMessageW(
                this.Handle,
                WM_APPCOMMAND,
                this.Handle,
                (IntPtr)APPCOMMAND_VOLUME_DOWN
                );
        }

    private void button3Pressed()
        {
            SendMessageW(
                this.Handle,
                WM_APPCOMMAND,
                this.Handle,
                (IntPtr)APPCOMMAND_VOLUME_UP
                );
        }

    private void button4Pressed()
    {

    }

    }
}
```

Liite 9. usbDemoDevice.cs-tiedosto (Simon Inns 2011, WFF Generic HID Demo)


```
//---------------------------------------------------------------------------
//
//  usbDemoDevice.cs
//
//  USB Generic Demonstration 3_0_0_0
//
//  A demonstration application for the usbGenericHidCommunications library
//  Copyright (C) 2011 Simon Inns
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
//  Web:    http://www.waitingforfriday.com
//  Email:  simon.inns@gmail.com
//
//---------------------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// The following namespace allows debugging output (when compiled in debug mode)
using System.Diagnostics;

namespace WFF_Generic_HID_Demo_3
    {
    using usbGenericHidCommunications;

    /// <summary>
    /// This class performs several different tests against the
    /// reference hardware/firmware to confirm that the USB
    /// communication library is functioning correctly.
    ///
    /// It also serves as a demonstration of how to use the class
    /// library to perform different types of read and write
    /// operations.
    /// </summary>
    class usbDemoDevice : usbGenericHidCommunication
        {
        int prevState;
        /// <summary>
        /// Class constructor - place any initialisation here
        /// </summary>
        /// <param name="vid"></param>
        /// <param name="pid"></param>
        public usbDemoDevice(int vid, int pid) : base(vid, pid)
            {
                prevState = 0;
            }

        public bool toggleLedState()
            {
            // Command 0x80 - Toggle LED state

            // Declare our output buffer
            Byte[] outputBuffer = new Byte[65];

            // Byte 0 must be set to 0
            outputBuffer[0] = 0;

            // Byte 1 must be set to our command
```

```
            outputBuffer[1] = 0x80;

            // Perform the write command
            bool success;
            success = writeRawReportToDevice(outputBuffer);

            // We can't tell if the device receieved the data ok, we are
            // only indicating that the write was error free.
            return success;
            }

        public int readPushButtonState()
            {
            // Command 0x81 - Read the push button state

            // Declare our output buffer
            Byte[] outputBuffer = new Byte[65];

            // Declare our input buffer
            Byte[] inputBuffer = new Byte[65];

            // Byte 0 must be set to 0
            outputBuffer[0] = 0;

            // Byte 1 must be set to our command
            outputBuffer[1] = 0x81;

            // Perform the write command
            bool success;
            success = writeRawReportToDevice(outputBuffer);

            // Only proceed if the write was successful
            if (success)
                {
                // Perform the read
                success = readSingleReportFromDevice(ref inputBuffer);
                }

            // Check report for all button states, starting with left-most one
            for (int i = 1; i < 5; i++)
            {
                // Only return button state if it wasn't held during last poll
                if (inputBuffer[i] == 0 && prevState == 0)
                {
                    prevState = i;
                    return i;
                }
            }

            //
            if (inputBuffer[prevState] == 1 && prevState > 0)
            {
                prevState = 0;
                return prevState;
            }

            // Note the push button is active low, so we send false if the reponse was 1
            /*if (inputBuffer[1] == 1 && inputBuffer[2] == 1 && inputBuffer[3] == 1 &&
inputBuffer[4] == 1)
                return 0;
            else return 1;*/
            return 0;
            }

        public bool readLedState()
            {
            // Command 0x82 - Read the LED state

            // Declare our output buffer
            Byte[] outputBuffer = new Byte[65];

            // Declare our input buffer
            Byte[] inputBuffer = new Byte[65];
```

```
            // Byte 0 must be set to 0
            outputBuffer[0] = 0;

            // Byte 1 must be set to our command
            outputBuffer[1] = 0x82;

            // Perform the write command
            bool success;
            success = writeRawReportToDevice(outputBuffer);

            // Only proceed if the write was successful
            if (success)
                {
                // Perform the read
                success = readSingleReportFromDevice(ref inputBuffer);
                }

            if (inputBuffer[1] == 1) return true; else return false;
            }

        // Collect debug information from the device
        public String collectDebug()
            {
            // Collect debug information from USB device
            Debug.WriteLine("Reference Application -> Collecting debug information from
device");

            // Declare our output buffer
            Byte[] outputBuffer = new Byte[65];

            // Declare our input buffer
            Byte[] inputBuffer = new Byte[65];

            // Byte 0 must be set to 0
            outputBuffer[0] = 0;

            // Byte 1 must be set to our command
            outputBuffer[1] = 0x10;

            // Send the collect debug command
            writeRawReportToDevice(outputBuffer);

            // Read the response from the device
            readSingleReportFromDevice(ref inputBuffer);

            // Byte 1 contains the number of characters transfered
            if (inputBuffer[1] == 0) return String.Empty;

            // Convert the Byte array into a string of the correct length
            string s = System.Text.ASCIIEncoding.ASCII.GetString(inputBuffer, 2,
inputBuffer[1]);

            return s;
            }
        }
    }
```