

4. MAGISTRALA USB

Această lucrare prezintă magistrala USB (*Universal Serial Bus*) și interfața care se bazează pe această magistrală. Sunt prezentate următoarele: noțiunile de bază ale interfeței USB, conectorii și cablurile utilizate, tipurile de transfer, modelul comunicației, protocolul magistralei, structura pachetelor, descriptorii USB și comenzile USB. De asemenea, lucrarea introduce clasa dispozitivelor HID și descrie modul în care se poate realiza comunicația între calculator și dispozitivele din această clasă.

4.1. Prezentare generală a magistralei USB

Elaborarea magistralei USB a început în anul 1995 de către un grup de firme care cuprindea Compaq, Digital, IBM, Intel, Microsoft, NEC și Northern Telecom. Aceste firme s-au unit în asociația *USB Implementers Forum* (USB-IF, <http://www.usb.org>), care a publicat prima versiune a standardului USB. Această asociație, care s-a extins cu un număr mare de firme, continuă să actualizeze standardele USB pentru controlerele USB și diferitele categorii de periferice care se pot conecta la magistrala USB.

Una din motivațiile elaborării magistralei USB a fost de a simplifica interconexiunile dintre calculator și periferice prin reducerea numărului de cabluri care se conectează la calculator și utilizarea aceluiași tip de conector pentru diferite categorii de periferice. Într-un sistem conținând o magistrală USB, perifericele se pot conecta în serie sau într-o topologie sub formă de stea pe mai multe nivele, un singur periferic fiind conectat la un port USB al calculatorului gazdă. A altă motivație pentru elaborarea magistralei USB a fost asigurarea unei rate de transfer mai ridicate decât cele permise de porturile seriale și paralele. Deși la primele versiuni (1.0 și 1.1) ale magistralei USB rata maximă de transfer era de numai 12 Mbiți/s, această rată a crescut în mod semnificativ (până la 480 Mbiți/s) la versiunea 2.0 a magistralei USB. Un alt scop a fost de a oferi posibilitatea adăugării perifericelor la calculator într-un mod simplu, fără deschiderea carcasei acestuia, fără deconectarea tensiunii de alimentare și fără reîncărcarea sistemului de operare.

Calculatorul echipat cu un port USB detectează adăugarea unui nou periferic și determină în mod automat resursele necesare acestuia, inclusiv driverul software și rata de transfer. Unul din perifericele calculatorului, de exemplu, tastatura sau monitorul, se conectează la un port USB al calculatorului. Alte periferice se pot conecta la un distribuitor ("hub") al tastaturii sau al monitorului, fiind posibilă realizarea unor conexiuni sub formă de arbore. Perifericele se pot afla la o distanță de până la 5 m unele față de altele sau față de un distribuitor. În total, se pot conecta până la 127 de periferice USB la un calculator, iar acestea sunt alimentate cu o tensiune de +5 V prin cablul USB. Toate perifericele USB utilizează un conector standard, eliminându-se necesitatea utilizării unor conectori diferiți pentru diferite tipuri de periferice.

Componentele principale ale unui sistem care utilizează magistrala USB sunt dispozitivele USB, cablurile USB și programele de sistem. Dispozitivele de pe magistrala USB sunt conectate fizic la calculatorul gazdă utilizând o topologie sub formă de stea pe mai multe nivele, după cum se ilustrează în figura 4.1.

Există două categorii de dispozitive USB: distribuitoare și funcții. Un *distribuitor* reprezintă o clasă specială de dispozitiv USB, care asigură puncte de conectare suplimentare pentru alte dispozitive USB. Aceste puncte de conectare se numesc porturi. Calculatorul gazdă conține un distribuitor rădăcină, care asigură unul sau mai multe puncte de conectare. În

plus, acest distribuitor conține controlerul magistralei USB; fiecare magistrală are un singur controler.

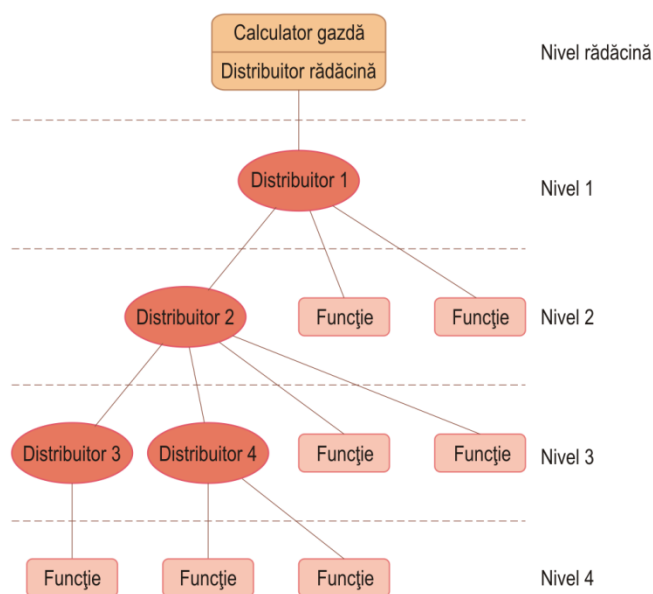


Figura 4.1. Topologia magistralei USB.

Figura 4.2 prezintă un distribuitor USB tipic. Unul din porturile distribuitorului permite conectarea la calculatorul gazdă sau la un distribuitor de pe nivelul superior al topologiei. Fiecare din celelalte șapte porturi permit conectarea la un distribuitor sau la o funcție de pe nivelul inferior. Distribuitorii pot fi conectați în cascadă până la cinci nivele. Distribuitorul detectează conectarea și deconectarea dinamică a unui periferic și asigură o putere de cel puțin 0,5 W pentru fiecare periferic în timpul inițializării. Sub controlul programului de sistem, distribuitorul poate asigura o putere suplimentară pentru funcționarea perifericelor, până la 2,5 W, 4,5 W sau 9 W (în funcție de versiunea USB). Unele periferice, cum este tastatura sau creionul optic, pot fi alimentate numai cu tensiunea furnizată de cablul magistralei, în timp ce altele pot avea o sursă proprie de alimentare.

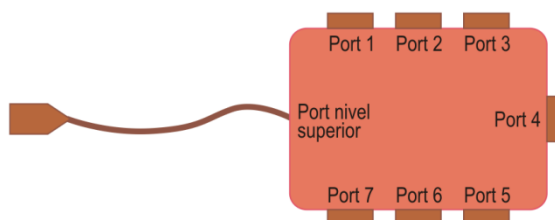


Figura 4.2. Porturile unui distribuitor USB tipic.

Un distribuitor constă din două părți: un controler și un repetor. Controlerul conține registre de interfață pentru comunicația cu calculatorul gazdă. Comenzile de stare și de control permit calculatorului gazdă configurarea distribuitorului, monitorizarea și controlul porturilor sale. Repetorul este un comutator controlat prin protocol între portul de nivel superior și porturile de nivel inferior. De asemenea, repetorul monitorizează semnalele de pe porturi și gestionează tranzacțiile care îi sunt adresate. Toate celelalte tranzacții sunt repetate la dispozitivele atașate. Fiecare port de nivel inferior poate fi validat individual și poate fi conectat la dispozitive cu viteză ridicată sau cu viteză redusă. Porturile cu viteză redusă sunt izolate de semnalele cu viteză ridicată.

Figura 4.3 ilustrează modul în care distribuitorii USB asigură conectarea într-un sistem de calcul tipic.

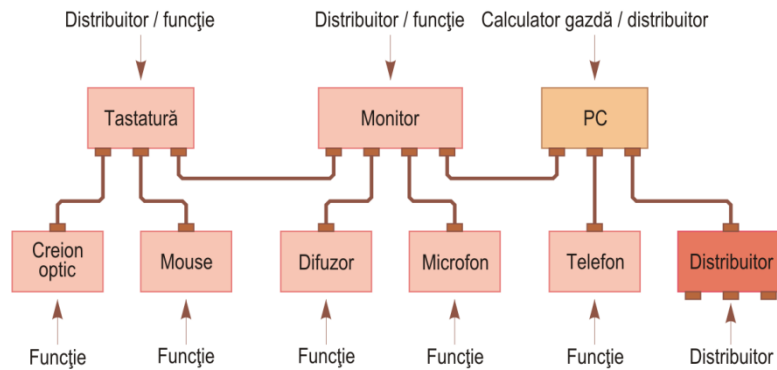


Figura 4.3. Utilizarea distribuitorilor USB într-un sistem de calcul tipic.

O *funcție* este un dispozitiv USB care poate transmite sau recepționa date sau informații de control pe magistrală. Acest dispozitiv trebuie să răspundă la cererile de tranzație transmise de calculatorul gazdă. O funcție este implementată în mod obișnuit ca un periferic separat conectat printr-un cablu la un port al unui distribuitor. Un singur dispozitiv fizic poate conține însă funcții multiple. De exemplu, o tastatură și un dispozitiv de trasare pot fi combinate într-un singur dispozitiv fizic. În cadrul unui asemenea dispozitiv compus, funcțiile individuale sunt atașate la un distribuitor, iar acest distribuitor intern este conectat la magistrala USB.

Fiecare funcție conține informații de configurație care descriu posibilitățile sale și resursele necesare. Aceste informații sunt transmise calculatorului gazdă ca răspuns la o tranzație de control. Înaintea utilizării unei funcții, aceasta trebuie configurată de calculatorul gazdă. Această configurare presupune alocarea unei lățimi de bandă în cadrul magistralei USB și selectarea opțiunilor specifice de configurație.

Programele de sistem asigură o tratare uniformă a sistemului de I/E de către toate programele de aplicații. Prin ascunderea detaliilor de implementare hardware, se asigură portabilitatea programelor de aplicații. Programele de sistem gestionează conectarea și deconectarea dinamică a perifericelor. Faza de conectare, numită *enumerare*, implică comunicarea cu un periferic pentru determinarea driverului de dispozitiv care trebuie încărcat și asignarea unei adrese unice acestuia. În timpul funcționării, calculatorul gazdă inițiază tranzații cu anumite periferice. Informațiile sunt transmise pe magistrală sub forma unor pachete, care sunt recepționate de fiecare periferic. Pachetele conțin adresa perifericului destinație; doar acest periferic va accepta o anumită tranzație și va răspunde în mod corespunzător.

Controlerul magistralei USB, aflat pe placa de bază a calculatorului gazdă, are propriile specificații. În cazul primelor versiuni ale magistralei USB, existau două specificații pentru aceste controlere. Prima dintre ele, *Universal Host Controller Interface* (UHCI), a fost elaborată de firma Intel și permitea simplificarea circuitelor, partea mai complexă fiind cea de software. A doua specificație, *Open Host Controller Interface* (OHCI), a fost elaborată de firmele Compaq, Microsoft și National Semiconductor; această specificație permitea simplificarea părții de software, partea mai complexă fiind cea de hardware. Odată cu introducerea versiunii 2.0 a magistralei USB, a fost necesară elaborarea unei noi specificații pentru controlerul de magistrală. Această specificație, numită *Enhanced Host Controller Interface* (EHCI), a fost elaborată de mai multe firme, printre care Intel, Compaq, NEC, Microsoft și Lucent Technologies. Specificația *Extensible Host Controller Interface* (xHCI) a fost definită de Intel cu contribuția unui număr mare de firme. Această specificație permite utilizarea tuturor vitezelor USB, inclusiv a vitezelor mai ridicate introduse de versiunea USB 3.0 și versiunile ulterioare, sub aceeași stivă de protocoale. Comparativ cu specificațiile anterioare, specificația xHCI îmbunătățește eficiența din punctul de vedere al puterii consumate, asigură suport pentru virtualizare și simplifică arhitectura driverelor.

Există specificații separate pentru diferite categorii (clase) de periferice USB. O clasă USB reprezintă un grup de periferice sau interfețe cu atribute sau servicii similare. De exemplu, două periferice sau interfețe sunt incluse în aceeași clasă dacă utilizează șiruri de date cu același format pentru comunicația cu calculatorul gazdă. Dintre clasele de periferice USB se

amintesc următoarele: dispozitive audio, dispozitive de comunicație (modemuri, telefoane analogice și digitale, adaptoare de rețea), dispozitive de interacțiune cu utilizatorul (HID – *Human Interface Device*), dispozitive de captare a imaginilor fixe (aparate foto digitale), imprimante, memorii de masă, dispozitive video.

4.2. Versiuni USB

Versiunea 1.0 a standardului USB a fost publicată în anul 1996, aceasta fiind urmată de versiunea 1.1, adoptată în anul 1998. Rata de transfer maximă specificată de aceste versiuni este de 12 Mbiți/s. Această rată de transfer este suficientă pentru periferice cum sunt telefonul sau difuzorul digital (care conține un convertor digital-analogic). Pentru periferice lente, cum este tastatura sau creionul optic, a fost prevăzut un canal de viteză redusă, cu rata de transfer de 1,5 Mbiți/s.

Versiunea 2.0 a standardului USB a fost publicată în anul 2000. Această versiune (numită și *Hi-Speed* USB) permite creșterea ratei de transfer cu un factor de 40 față de versiunea 1.1, de la 12 Mbiți/s până la 480 Mbiți/s. Această extensie a specificațiilor USB permite utilizarea aceluiași cabluri, conectori și drivere software. Utilizatorii pot beneficia însă de o varietate suplimentară de periferice, de exemplu, camere video digitale, scanere, imprimante, adaptoare de rețea, unități de discuri magnetice sau unități de discuri optice. Figura 4.4 ilustrează simbolul magistralei USB 2.0.



Figura 4.4. Simbolul magistralei USB 2.0.

Perifericele USB 2.0 cu viteze de transfer superioare sunt conectate la un distribuitor USB 2.0. Un distribuitor USB 2.0 acceptă tranzacții de viteză ridicată și furnizează datele cu ratele corespunzătoare perifericelor USB 2.0 și perifericelor USB 1.1. Posibilitatea de a utiliza transferuri cu viteză ridicată este negociată cu fiecare periferic, iar dacă un periferic nu permite aceste transferuri, legătura cu acest periferic va funcționa la o viteză mai redusă de 12 Mbiți/s sau 1,5 Mbiți/s. Aceasta implică o complexitate mai ridicată a distribuitorilor și necesită memorarea temporară a datelor recepționate. Un distribuitor USB 2.0 are porturi de ieșire pentru transferuri cu viteză ridicată și porturi de ieșire pentru transferuri cu viteză normală.

Specificațiile USB “On-The-Go” (OTG) au fost elaborate ca un supliment al specificațiilor USB 2.0 cu scopul de a permite conexiuni directe între echipamente mobile, fără utilizarea unui calculator. Magistrala USB standard utilizează o arhitectură *master/slave*, la care calculatorul gazdă are rolul de *master*, iar un periferic are rolul de *slave*. Doar calculatorul gazdă poate iniția transferuri de date pe magistrală. Perifericele pot doar răspunde la cererile de transfer inițiate de calculatorul gazdă. În cazul versiunii USB OTG, un periferic poate avea fie rolul de *master*, fie cel de *slave*. Rolurile de *master* și de *slave* se pot schimba în mod dinamic în timpul funcționării cu un protocol numit *Host Negotiation Protocol* (HNP). Astfel, orice periferic compatibil cu specificațiile USB OTG poate iniția transferuri de date pe magistrală. Condiția este ca cele două echipamente care comunică între ele să fie conectate direct și nu printr-un distribuitor.

Un exemplu de utilizare a versiunii USB OTG este cel al unei tablete sau telefon mobil, care poate avea rolul implicit de *slave* pentru un calculator PC (în scopul sincronizării datelor) sau rolul implicit de *master* pentru o imprimantă. Un alt exemplu este cel al unei imprimante, care poate avea rolul de *slave* pentru o tabletă și rolul de *master* pentru un aparat fotografic digital, dacă permite citirea fișierelor de la aparatul fotografic în scopul tipăririi lor. Figura 4.5 ilustrează simbolul magistralei USB OTG.



Figura 4.5. Simbolul magistralei USB OTG.

Specificațiile versiunii 3.0 a standardului USB au fost finalizate de *USB 3.0 Promoter Group* în anul 2008 și au fost transferate asociației *USB Implementers Forum*. Această versiune introduce magistrala *SuperSpeed*, care permite un nou mod de transfer cu viteza maximă de 5 Gbiți/s. În modul *SuperSpeed*, se utilizează două canale diferențiale unidirecționale (simplex) în plus față de canalul diferențial existent pentru modul convențional. Tehnologia este similară cu cea a versiunii 2.0 a magistralei *PCI Express*. Se utilizează aceeași codificare 8b/10b, ceea ce permite o rată maximă de transfer de 500 MB/s; este posibilă obținerea unei rate de transfer de 400 MB/s dacă se ia în considerare metoda de codificare a datelor. Figura 4.6 ilustrează simbolul magistralei USB *SuperSpeed*.



Figura 4.6. Simbolul magistralei USB *SuperSpeed*.

Specificațiile versiunii USB 3.1 au fost publicate în anul 2013. Această versiune introduce un mod de transfer mai rapid numit “*SuperSpeed+ USB 10 Gbps*”, care crește rata datelor până la 10 Gbiți/s în modul *USB 3.1 Gen 2*, dublă față de cea a versiunii USB 3.0 (care a fost redenumită *USB 3.1 Gen 1*). Metoda de codificare a datelor a fost schimbată de la 8b/10b la 128b/132b, ceea ce reduce volumul datelor suplimentare datorate codificării la doar 3%. Standardul USB 3.1 asigură compatibilitatea cu versiunile precedente USB 3.0 și USB 2.0. Figura 4.7 ilustrează simbolul magistralei USB *SuperSpeed+ 10 Gbps*.



Figura 4.7. Simbolul magistralei USB *SuperSpeed+ 10 Gbps*.

Specificațiile versiunii USB 3.2, publicate în 2017, introduc un mod de transfer numit “*SuperSpeed+ USB 20 Gbps*” utilizând cabluri și conectori USB 3.1 existenți. Dublarea ratei datelor este obținută prin folosirea a două benzi de comunicație în locul unei singure benzi, utilizând firele existente destinate pentru a oferi reversibilitate conectorului de tip C (acest conector este prezentat în secțiunea 4.3). Se utilizează aceeași codificare 128b/132b a datelor ca și în versiunea 3.1. Compatibilitatea cu versiunile anterioare este asigurată prin folosirea unuia din modulele de transfer următoare: USB 3.2 Gen 1x1 (o bandă de comunicație, *SuperSpeed* la 5 Gbiți/s), USB Gen 1x2 (două benzi de comunicație, *SuperSpeed* la 10 Gbiți/s) sau USB Gen 2x1 (o bandă de comunicație, *SuperSpeed+* la 10 Gbiți/s).

Specificațiile *USB Power Delivery* au fost dezvoltate ca o extensie a standardelor USB. Această extensie specifică utilizarea cablurilor USB certificate cu conectori USB standard pentru a furniza o putere crescută anumitor dispozitive. Dispozitivele pot solicita un curent mai mare și o tensiune de alimentare mai ridicată de la calculatoarele gazdă care sunt conforme cu aceste specificații, de până la 2 A la 5 V (10 W). Opțional, consumul de curent poate fi crescut până la 3 A sau 5 A la 12 V (36 W sau 60 W) sau 20 V (60 W sau 100 W). Aceasta permite încărcarea calculatoarelor laptop în mod similar cu tabletele sau telefoanele

mobile, prin porturile lor USB, ceea ce poate elimina în viitor diferitele porturi de încărcare proprietare.

4.3. Conectori și cabluri

Specificațiile inițiale ale magistralei USB au definit două tipuri de mufe (fișe), amplasate la capetele unui cablu USB, și două tipuri de socluri, amplasate într-un distribuitor sau periferic. Mufele și soclurile sunt denumite de tip A și de tip B.

Distribuitoarele (de exemplu, cele ale unui calculator gazdă) conțin un soclu rectangular de tip A. Perifericele se conectează la acest soclu printr-o mufă rectangulară de tip A (figura 4.8). Cablurile care sunt atașate permanent la periferice conțin o mufă de tip A. De obicei însă, perifericele sunt conectate printr-un cablu detașabil. Perifericele conțin un soclu pătrat de tip B, iar cablul care conectează perifericele la un distribuitor conține o mufă de tip B la capătul care se conectează la periferic și o mufă de tip A la capătul care se conectează la distribuitor. În acest fel, nu este posibilă conectarea incorectă a cablului.



Figura 4.8. Mufă și soclu de tip A (stânga); mufă și soclu de tip B (dreapta).

Specificațiile USB 2.0 au fost modificate după publicarea acestora pentru a include o mufă și un soclu de tip B de dimensiuni mai reduse. Acești conectori, numiți mini-B, conțin cinci pini și se utilizează pentru echipamente mobile cum sunt tablete, telefoane mobile și aparate foto digitale. Aceste echipamente conțin un soclu mini-B, iar cablurile utilizate pentru conectarea acestor echipamente la un calculator conțin o mufă mini-B la un capăt și o mufă de tip A la celălalt capăt. Dimensiunile conectorilor mini-B sunt de aproximativ 7x3 mm. Ulterior, a fost definit și un conector mini-A. Figura 4.9 ilustrează o mufă mini-B alăturată cu o mufă de tip A.



Figura 4.9. Mufă USB de tip mini-B alăturată cu o mufă de tip A.

În anul 2007 au fost definiți conectori de tip A și de tip B de dimensiuni și mai reduse, numiți micro-A și micro-B. Acești conectori au aceeași lățime ca și conectorii mini-B, dar grosimea este redusă la aproximativ jumătate. Noii conectori sunt destinați înlocuirii conectorilor mini-A și mini-B, care nu se mai utilizează la noile echipamente mobile.

Specificațiile USB OTG descriu noi tipuri de socluri, mini-AB și micro-AB. Aceste socluri permit, printr-o proiectare mecanică adecvată, conectarea fie a unei mufe de tip A, fie

a unei mufe de tip B cu dimensiunea corespunzătoare. Tipul mufei inserate este detectat cu ajutorul unui pin suplimentar ID, care este conectat la masă într-un conector de tip A și este neconectat într-un conector de tip B. Atunci când într-un soclu AB se inserează un conector de tip A, soclul va furniza tensiune de alimentare cablului și echipamentul cu soclul AB va avea rolul de *master*. Atunci când într-un soclu AB se inserează un conector de tip B, soclul va fi alimentat cu tensiunea cablului și echipamentul cu soclul AB va avea rolul de *slave*. Specificațiile USB OTG descriu și diferite tipuri de cabluri care utilizează conectori de dimensiuni reduse sau o combinație între un conector de dimensiuni reduse și unul de dimensiuni normale. Figura 4.10 ilustrează un soclu micro-AB și un soclu micro-B.



Figura 4.10. Socluri USB OTG micro-AB și micro-B.

Versiunea USB 3.0 definește noi conectori de tip A, de tip B și micro-B. Figura 4.11 ilustrează, de la stânga la dreapta, o mufă de tip A, o mufă de tip B și o mufă micro-B. Mufele USB 3.0 de tip A au aceeași formă ca și mufele USB 2.0 de tip A și sunt compatibile cu soclurile USB 2.0 de tip A. Mufele USB 3.0 conțin însă cinci pini suplimentari utilizați de cele două canale diferențiale suplimentare. Specificațiile USB 3.0 indică faptul că partea interioară a conectorilor trebuie să fie colorată în albastru (Pantone 300C). Mufele USB 3.0 de tip B au o formă diferită pentru a include cinci noi pini corespunzători celor cinci pini suplimentari din conectorii de tip A. Aceste mufe nu sunt compatibile cu soclurile USB 2.0 de tip B. O mufă USB 3.0 micro-B constă din doi conectori, un conector USB 2.0 micro-B și un conector suplimentar atașat lângă acesta. Conectorii USB 3.0 micro-B se utilizează mai ales pentru unitățile de discuri externe.



Figura 4.11. Mufe USB 3.0 de tip A, tip B și micro-B.

Conectorul USB de tip C este un nou conector ale cărui specificații au fost elaborate și publicate de *USB Implementers Forum* în anul 2014; specificațiile au fost actualizate în 2015 și 2017. Mufa de tip C se conectează atât la calculatoare gazdă cât și la dispozitive USB, fiind destinată înlocuirii mufelor de tip A și de tip B. Conectorul de tip C este reversibil și conține 24 de pini (figura 4.12), printre care patru pini de alimentare, patru pini de masă, patru pini pentru două canale diferențiale USB 2.0 (deși în cablul de tip C este implementat un singur canal), opt pini pentru patru canale *SuperSpeed* și doi pini de configurație pentru detectarea orientării cablului.



Figura 4.12. Pinii conectorului USB de tip C.

Dimensiunile unui port de tip C sunt de 8,4 x 2,6 mm, astfel încât se poate utiliza chiar și pentru dispozitive cu dimensiuni reduse. Conectorul permite un curent de 1,5 A sau 3 A la 5 V (7,5 W sau 15 W), sau poate fi conform cu specificațiile *USB Power Delivery* pentru a furniza o putere de până la 100 W. Dacă un dispozitiv conține un port de tip C, aceasta nu

înseamnă neapărat că este conform cu versiunea USB 3.1 sau 3.0 (de exemplu, poate fi conform doar cu versiunea USB 2.0) sau cu specificațiile *USB Power Delivery*. Pentru conectarea unui dispozitiv mai vechi la un calculator gazdă cu un soclu de tip C este necesar un adaptor sau un cablu cu o mufă de tip A sau de tip B la un capăt și o mufă de tip C la celălalt capăt. Conectorii și cablurile de tip C se utilizează și de către versiunea 3 a tehnologiei proprietare a firmei Intel, Thunderbolt, iar această versiune este compatibilă cu versiunea USB 3.1. Astfel, același dispozitiv poate utiliza ambele moduri de funcționare USB 3.1 și Thunderbolt.

Figura 4.13 ilustrează o mufă de tip C lângă o mufă de tip A.



Figura 4.13. Mufe USB de tip C și de tip A.

Pentru transferul semnalelor și a tensiunii de alimentare, magistrala USB 2.0 utilizează un cablu cu patru fire, ilustrat în figura 4.14. Semnalele diferențiale de date se transmit pe liniile $D+$ și $D-$, formate din două fire răsucite. Semnalul de ceas este codificat împreună cu datele. Metoda de codificare utilizată este numită NRZI (*Non-Return to Zero Inverted*). În cazul acestei metode, un bit de 1 este reprezentat printr-o lipsă a schimbării nivelului tensiunii, iar un bit de 0 este reprezentat printr-o schimbare a nivelului tensiunii, fără revenire la tensiunea de referință (zero) între biții codificați. Sunt inserați biți suplimentari în datele transmise pentru a asigura tranziții suficiente ale semnalelor, în scopul garantării unei sincronizări corecte. Este inserat un bit de 0 după fiecare șase biți consecutivi de 1 înainte de codificarea datelor, pentru a forța o tranziție în șirul de date. Fiecare pachet de date este precedat de un câmp de sincronizare pentru a permite receptorilor sincronizarea ceasurilor de recepție.

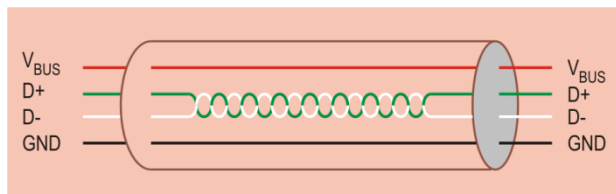


Figura 4.14. Cablu USB 2.0.

Un cablu USB 2.0 transmite și tensiunea de alimentare pentru periferice pe liniile V_{BUS} și GND . Tensiunea pe linia V_{BUS} este de +5 V la sursă. Pentru a asigura nivele de tensiune corecte la intrarea perifericelor și o impedanță de terminare corespunzătoare, se utilizează terminatori de magistrală la fiecare capăt al cablului. Terminatorii de magistrală permit, de asemenea, detectarea conectării sau deconectării unui periferic și diferențierea între perifericele cu viteză normală (12 Mbiți/s) și cele cu viteză redusă (1,5 Mbiți/s). La capătul cablului din partea distribuitorului, se utilizează terminatori rezistivi de 15 K Ω , prin intermediul cărora liniile $D+$ și $D-$ ale cablului sunt conectate la masă. La capătul cablului din partea perifericului, se utilizează ca terminator o rezistență de 1,5 K Ω , prin intermediul căreia una din liniile $D+$ sau $D-$ este conectată la o sursă de tensiune cuprinsă între 3 V și 3,6 V. Pentru un periferic cu viteză normală rezistența este conectată la linia $D+$, în timp ce pentru un periferic cu viteză redusă rezistența este conectată la linia $D-$.

Pentru identificarea simplă a firelor din cablurile USB 2.0, standardul specifică utilizarea culorilor indicate în tabelul 4.1 pentru aceste fire. Tabelul indică și asignarea pinilor conectorilor la semnalele magistralei.

Tabelul 4.1. Asignarea pinilor conectorilor la semnalele magistralei USB 2.0 și culorile firelor cablurilor.

Nr. pin	Semnal	Culoare
1	V _{BUS}	Roșu
2	D-	Alb
3	D+	Verde
4	GND	Negru

Cablurile USB 3.0, USB 3.1 și USB 3.2 conțin opt fire. Două fire se utilizează pentru semnalele de alimentare și masă, două fire se utilizează pentru canalul diferențial al modului convențional și patru fire se utilizează pentru cele două canale unidirecționale ale modului *SuperSpeed* sau *SuperSpeed+*.

Observație

- Nu este posibilă interconectarea a două calculatoare printr-un cablu USB obișnuit. Chiar dacă s-ar utiliza un cablu cu doi conectori de tip A, prin interconectarea calculatoarelor ar exista două controlere USB într-un sistem, ceea ce nu este permis. Există însă cabluri speciale care conțin o punte USB sub forma unui circuit integrat, prin intermediul căruia este posibilă comunicația între două calculatoare gazdă.

4.4. Tipuri de transfer

Arhitectura USB permite patru tipuri de transferuri de date: de control, de întrerupere, de date voluminoase și izocrone.

Transferurile *de control* se utilizează de driverele calculatorului gazdă pentru configurarea dispozitivelor care sunt atașate la sistem. Alte drivere pot utiliza transferuri de control în moduri specifice implementării.

Transferurile *de întrerupere* se utilizează pentru date care trebuie transferate cu o întârziere limitată. Transferul acestor date poate fi solicitat de un dispozitiv în orice moment, iar rata de transfer pe magistrală nu poate fi mai redusă decât cea specificată de dispozitiv. Datele pentru care se utilizează transferurile de întrerupere constau din notificarea unor evenimente, din caractere sau coordonate care sunt organizate pe unul sau mai mulți octeți. Un exemplu îl reprezintă coordonatele de la un dispozitiv indicator (mouse, tabletă grafică). Datele interactive pot avea anumite limite ale timpului de răspuns care trebuie garantate de magistrala USB.

Transferurile *de date voluminoase* (“*bulk*”) se utilizează cu periferice cum sunt memorii de masă, imprimante sau scanere. Aceste date sunt secvențiale. Fiabilitatea transferurilor este asigurată la nivel hardware prin utilizarea unui cod detector de erori și reluarea unui transfer cu erori de un număr de ori. Rata de transfer poate varia în funcție de alte activități de pe magistrală.

Transferurile *izocrone* (*isos* – egal, *chronos* – timp) se utilizează pentru datele care trebuie furnizate cu o anumită rată de transfer constantă și a căror sincronizare trebuie garantată. Izocron are semnificația “cu durată egală” sau “care apare la intervale regulate”. Datele izocrone sunt generate în timp real și trebuie furnizate cu rata cu care sunt recepționate pentru a păstra sincronizarea lor. Pe lângă rata de transfer impusă, pentru datele izocrone trebuie respectată și întârzierea maximă cu care acestea sunt furnizate. Furnizarea la timp a datelor izocrone este asigurată cu prețul unor pierderi potențiale în șirul de date. Deci, erorile de transmisie nu sunt corectate prin mecanisme hardware, cum ar fi retransmiterea lor. În concluzie, transferurile izocrone se caracterizează prin furnizarea la timp a datelor și prin lipsa retransmiterii lor în cazul unor erori, deoarece datele întârziate nu mai sunt utile. Spre deosebire de transferurile izocrone, transferurile asincrone se caracterizează prin faptul că fiabilitatea transmiterii datelor este mai importantă decât asigurarea sincronizării. De aceea, se utilizează retransmiterea datelor în cazul unor erori, chiar dacă apar întârzieri din această cauză.

Un exemplu tipic de date izocrone este reprezentat de imaginile video. Dacă rata de transfer a acestor șiruri de date nu este menținută, va avea loc pierderea unor date datorită depășirii capacității bufferelor. Chiar dacă datele sunt furnizate de magistrala USB cu rata

adecvată, întârzierile introduse de programe pot afecta negativ aplicațiile care utilizează aceste date, cum sunt aplicațiile pentru video-conferințe.

Șirurilor de date izocrone li se alocă o porțiune dedicată a lățimii de bandă a magistralei USB. Această magistrală este proiectată și pentru o întârziere minimă a transferurilor de date izocrone.

4.5. Modelul comunicației USB

Un sistem USB permite comunicația dintre un program de aplicație (client) rulat pe un calculator gazdă și unul sau mai multe dispozitive USB conectate la acest calculator. Un dispozitiv fizic USB conține o interfață cu magistrala USB, un dispozitiv logic USB și o funcție. O funcție USB poate avea cerințe diferite ale fluxului de comunicație pentru diferite interacțiuni între acea funcție și calculatorul gazdă. Prin separarea diferitelor fluxuri de comunicație cu o funcție USB, sistemul USB asigură o utilizare mai eficientă a magistralei. Fluxurile de comunicație utilizează magistrala USB pentru realizarea comunicației între programul de aplicație și funcția USB. Fiecare flux de comunicație se termină la un *punct terminal* (“*end-point*”) dintr-un dispozitiv USB.

Un punct terminal este o parte a unui dispozitiv USB, reprezentând punctul de sfârșit al fluxului de comunicație dintre calculatorul gazdă și dispozitiv. Fiecare dispozitiv logic USB este format din mai multe puncte terminale independente. Fiecare dispozitiv logic are o adresă unică, care este asignată de sistem în momentul conectării dispozitivului la magistrala USB. Fiecare punct terminal dintr-un dispozitiv USB este identificat printr-un număr unic, care este atribuit la proiectarea dispozitivului. De asemenea, fiecare punct terminal are o anumită direcție a fluxului de date: intrare (IN) pentru transferul datelor de la dispozitivul USB la calculatorul gazdă, sau ieșire (OUT) pentru transferul datelor de la calculatorul gazdă la dispozitivul USB. Combinația dintre adresa dispozitivului, numărul punctului terminal și direcția acestuia permite identificarea unică a fiecărui punct terminal.

Punctele terminale conțin buffere de intrare sau de ieșire prin intermediul cărora se realizează comunicația între calculatorul gazdă și dispozitivul USB. De exemplu, dacă programul de aplicație rulat pe calculatorul gazdă transmite un pachet adresat unui anumit dispozitiv USB, acest pachet va fi depus în bufferul unui punct terminal de ieșire al dispozitivului. Ulterior, controlerul dispozitivului va citi din buffer pachetul recepționat. Dacă dispozitivul trebuie să transmită date la calculatorul gazdă, nu poate depune datele direct pe magistrală, deoarece toate tranzacțiile sunt inițiate de calculatorul gazdă. Dispozitivul va depune datele în bufferul unui punct terminal de intrare, iar aceste date vor fi transferate la calculatorul gazdă atunci când calculatorul va transmite un pachet de intrare prin care solicită transferul unor date, dacă acestea sunt disponibile.

Fiecare dispozitiv trebuie să conțină cel puțin punctele terminale cu numărul 0 (atât punctul terminal de intrare, cât și cel de ieșire). Aceste puncte terminale se utilizează pentru transferul informațiilor de control și de stare în timpul fazei de enumerare și în timpul în care dispozitivul este operațional și conectat la magistrala USB. Dispozitivele cu viteză redusă (1,5 Mbiți/s) pot conține doar două puncte terminale suplimentare, pe lângă cele două cu numărul 0. Dispozitivele cu viteză normală (12 Mbiți/s) sau viteză ridicată pot conține un număr maxim de 15 puncte terminale de intrare și 15 puncte terminale de ieșire, pe lângă cele două puncte terminale cu numărul 0.

O colecție a unor puncte terminale dintr-un dispozitiv USB implementează o *interfață*. O asemenea interfață indică clasa dispozitivului USB, iar această clasă va determina driverul generic de dispozitiv care va fi utilizat de sistemul de operare pentru comunicația cu punctele terminale care implementează interfața respectivă.

Comunicația dintre programul de aplicație de pe calculatorul gazdă și un punct terminal dintr-un dispozitiv USB se realizează printr-o conexiune logică numită *conductă* (“*pipe*”). O conductă reprezintă o legătură între un buffer din memoria calculatorului gazdă și un punct terminal din dispozitivul USB. Fiecărei conducte i se asociază anumite informații, cum sunt rata de transfer necesară, tipul transferului și caracteristicile punctului terminal asociat, ca direcția și dimensiunea bufferului.

Există două moduri de comunicație printr-o conductă: șir sau mesaj. În *modul șir*, datele nu au o structură definită de specificațiile USB. Datele sunt transferate secvențial și au o direcție predefinită, de intrare sau de ieșire. Conducele de tip șir permit transferuri de întrerupere, de date voluminoase sau izocrone. Aceste conducte sunt controlate fie de calculatorul gazdă, fie de dispozitivul USB. În *modul mesaj*, datele au o anumită structură definită de specificațiile USB. Cu toate acestea, conținutul datelor transferate printr-o conductă de tip mesaj nu este interpretat de controlerul USB. Aceste conducte sunt controlate de calculatorul gazdă și permit doar transferuri de control, în ambele direcții.

Comunicația între punctele terminale de intrare și de ieșire cu numărul 0 și calculatorul gazdă se realizează printr-o conductă specială, numită *conductă implicită de control*. Această conductă de tip mesaj este disponibilă imediat după ce dispozitivul USB este conectat și resetat, asigurând o legătură bidirecțională pentru interogarea dispozitivului de către programele de sistem ale calculatorului gazdă și transmiterea informațiilor de configurație de la dispozitiv către calculatorul gazdă. După configurarea dispozitivului USB, vor fi disponibile alte conducte necesare pentru transferurile de date; conducta implicită de control va fi utilizată în continuare de programele de sistem ale calculatorului gazdă pentru transferurile de control.

4.6. Protocolul USB

Similar cu alte interfețe mai recente, interfața USB utilizează un protocol bazat pe pachete. Toate transferurile sunt inițiate de controlerul USB al calculatorului gazdă. Tranzacțiile de pe magistrală implică transmisia a patru tipuri de pachete:

- Pachet antet (simbol – “token”);
- Pachet de date;
- Pachet de confirmare (“handshake”);
- Pachet special.

Fiecare tranzacție începe în momentul în care controlerul USB transmite, pe baza unei planificări, un pachet antet care descrie tipul tranzacției, direcția acesteia, adresa dispozitivului USB și numărul punctului terminal. Sursa tranzacției transmite apoi un pachet de date conținând datele care trebuie transferate, sau poate indica faptul că nu are date de transmis prin faptul că pachetul de date nu conține informații utile. În general, destinația răspunde cu un pachet de confirmare indicând dacă transferul s-a terminat cu succes sau dacă punctul terminal nu este disponibil.

4.6.1. Câmpurile pachetelor USB

Principalele câmpuri ale pachetelor USB sunt descrise în continuare.

Câmpul de sincronizare

Toate pachetele încep cu un câmp de sincronizare (SYNC), care este utilizat de circuitele receptorului pentru sincronizarea cu ceasul transmițătorului. Câmpul de sincronizare are o lungime de 8 biți pentru viteza normală/joasă și 32 de biți pentru viteza ridicată. Acest câmp conține un număr de 6 sau 30 de tranziții succesive de la valoarea 1 la valoarea 0 sau invers, urmate de un marcaj de doi biți care este utilizat pentru a identifica sfârșitul câmpului de sincronizare.

Câmpul de identificare al pachetului

Câmpul de identificare al pachetului (PID) urmează imediat după câmpul de sincronizare. Câmpul PID conține patru biți care indică tipul pachetului urmați de patru biți de control care asigură decodificarea corectă a câmpului PID. Biții de control conțin complementul față de 1 al biților care reprezintă tipul pachetului. Formatul câmpului PID este ilustrat în continuare, unde $nPID_i$ reprezintă complementul față de 1 al bitului PID_i .

7	6	5	4	3	2	1	0
nPID ₃	nPID ₂	nPID ₁	nPID ₀	PID ₃	PID ₂	PID ₁	PID ₀

Calculatorul gazdă și toate funcțiile USB execută o decodificare completă a tuturor biților recepționați ai câmpului PID. Dacă un câmp PID este recepționat cu valori incorecte ale biților de control sau cu valori nedefinite ale tipului pachetului, se presupune că a fost alterat și restul pachetului este ignorat de receptor.

Tabelul 4.2 indică tipurile de pachete, codificarea și descrierea acestora. Pentru simplitate, pachetele speciale nu sunt detaliate.

Tabelul 4.2. Codificarea și descrierea tipurilor de pachete USB.

Tip pachet	Subtip pachet	PID [3..0]	Descriere
Antet (Token)	OUT	0001	Adresa și numărul punctului terminal într-o tranzacție de ieșire
	IN	1001	Adresa și numărul punctului terminal într-o tranzacție de intrare
	SOF	0101	Indicator de început al cadrului și numărul cadrului
	SETUP	1101	Adresa și numărul punctului terminal într-o tranzacție de control în faza de setare
Date	DATA0	0011	Identificator pentru pachetele de date cu număr par
	DATA1	1011	Identificator pentru pachetele de date cu număr impar
	DATA2	0111	Identificator pentru pachetele de date la tranzacțiile izocrone de intrare de viteză ridicată și cu lățimea de bandă ridicată
	MDATA	1111	Identificator pentru pachetele de date la tranzacțiile izocrone de ieșire de viteză ridicată și cu lățimea de bandă ridicată
Confirmare	ACK	0010	Confirmarea recepționării fără erori a pachetului de date
	NAK	1010	Dispozitivul receptor nu poate primi date sau dispozitivul transmițător nu poate transmite date
	STALL	1110	Punctul terminal este oprit
	NYET	0110	Nu s-a recepționat încă un răspuns de la receptor
Special		XY00	Identificator al unui pachet special; XY poate fi 01, 10 sau 11

Câmpul de adresă

Câmpul de adresă (ADDR) specifică adresa funcției USB care este sursa sau destinația unui pachet de date. Acest câmp are o lungime de 7 biți, permițând specificarea a până la 128 de adrese. Fiecare adresă definește o singură funcție. Adresa 0 este rezervată ca adresă implicită și nu poate fi asignată în mod explicit unei funcții. La punerea sub tensiune și resetarea unei funcții, adresa acesteia va avea valoarea implicită 0. Calculatorul gazdă trebuie să seteze adresa funcției în timpul procesului de enumerare.

Câmpul punctului terminal

Câmpul punctului terminal (ENDP) permite o adresare mai flexibilă a funcțiilor la care sunt necesare mai multe puncte terminale. Acest câmp are o lungime de 4 biți, ceea ce permite adresarea a până la 16 puncte terminale. Dispozitivele cu viteză redusă pot avea doar două puncte terminale suplimentare pe lângă punctul terminal cu numărul 0.

Câmpul de date

Câmpul de date poate conține între zero și 1024 de octeți, în funcție de tipul transferului. Biții de date din cadrul fiecărui octet sunt transmiși pe magistrală începând cu bitul cel mai puțin semnificativ.

Câmpurile de control ciclic redundant

Aceste câmpuri conțin codurile de control ciclic redundant (CRC) utilizate pentru verificarea integrității diferitelor câmpuri din pachetele de antet și de date. Câmpul PID nu este inclus în codul CRC al unui pachet. Codurile CRC pentru pachetele de antet și de date asigură detectarea tuturor erorilor de un singur bit și de doi biți. În cazul în care codul CRC calculat de receptor diferă de codul transmis într-un câmp CRC, receptorul va ignora câmpurile protejate și, în majoritatea cazurilor, întregul pachet. Standardul USB specifică polinoamele generatoare utilizate pentru calculul codurilor CRC. Pentru pachetele de antet se utilizează un câmp CRC de 5 biți (CRC5), iar pentru pachetele de date se utilizează un câmp CRC de 16 biți (CRC16).

Câmpul de sfârșit al pachetului

Câmpul de sfârșit al pachetului (EOP) indică sfârșitul unui pachet prin valoarea 0 pe durata a doi biți, urmată de valoarea 1 pe durata unui bit.

4.6.2. Formatul pachetelor USB

Această secțiune prezintă formatul pachetelor de antet, SOF, de date și de confirmare.

Pachete de antet

Aceste pachete sunt transmise doar de calculatorul gazdă. Structura unui pachet de antet este ilustrată în figura 4.15.



Figura 4.15. Structura unui pachet USB de antet.

Câmpul de identificare al pachetului, PID, poate specifica un pachet de antet cu subtipul IN, OUT sau SETUP. Pachetele cu subtipul IN sau OUT informează dispozitivul USB asupra direcției transferului care urmează, intrare (citire de către calculatorul gazdă), respectiv ieșire (scriere de către calculatorul gazdă). Un pachet cu subtipul SETUP se utilizează la începutul transferurilor de control. În cazul pachetelor cu subtipul OUT sau SETUP, câmpurile ADDR și ENDP identifică în mod unic punctul terminal care va recepționa următorul pachet de date. În cazul unui pachet cu subtipul IN, câmpurile ADDR și ENDP identifică punctul terminal care va transmite un pachet de date. Câmpul CRC5 conține codul CRC pentru câmpurile ADDR și ENDP.

Pachete SOF

Pentru sincronizarea întregului sistem USB, calculatorul gazdă transmite câte un pachet SOF (*Start Of Frame*) la fiecare interval de timp corespunzător începutului unui cadru sau micro-cadru. Un cadru reprezintă un interval de timp de $1 \text{ ms} \pm 0,0005 \text{ ms}$ și este definit pentru magistrala USB cu viteza normală (12 Mbiți/s). Un micro-cadru reprezintă un interval de timp de $125 \text{ } \mu\text{s} \pm 0,0625 \text{ } \mu\text{s}$ și este definit pentru magistrala USB cu viteza ridicată (480 Mbiți/s). Un pachet SOF constă dintr-un câmp de sincronizare, un câmp PID și un câmp de 11 biți reprezentând numărul cadrului, după cum se ilustrează în figura 4.16. În cazul magistralei USB cu viteza ridicată, numărul cadrului va fi același pentru opt pachete SOF consecutive, pe durata unei perioade de 1 ms.

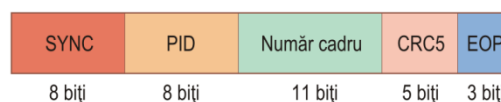


Figura 4.16. Structura unui pachet USB SOF.

Toate funcțiile și distribuitorii USB recepționează pachetele SOF. Recepția acestor pachete nu va iniția generarea unor pachete de confirmare de către receptor.

Pachete de date

Informațiile propriu-zise sunt transmise pe magistrala USB în pachetele de date. Un pachet de date constă din câmpul de sincronizare SYNC, un câmp de identificare al pachetului PID, un câmp de date, un câmp CRC de 16 biți și câmpul de sfârșit al pachetului EOP (figura 4.17). Codul CRC se calculează numai din câmpul de date. Datele sunt transmise într-un număr întreg de octeți. Pentru dispozitivele cu viteză redusă, lungimea maximă a câmpului de date este de 8 octeți. Pentru dispozitivele cu viteză normală (12 Mbiți/s) lungimea maximă a câmpului de date este de 1023 octeți, iar pentru cele de viteză ridicată (480 Mbiți/s) lungimea maximă este de 1024 octeți.

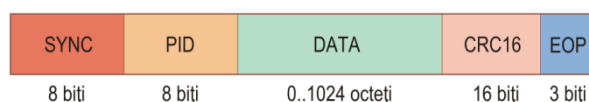


Figura 4.17. Structura unui pachet USB de date.

Pachete de confirmare

Pachetele de confirmare constau doar din câmpul de sincronizare SYNC, un câmp de identificare al pachetului PID și câmpul de sfârșit al pachetului EOP (figura 4.18). Aceste pachete se utilizează pentru a raporta starea unei tranzacții de date prin subtipul returnat în câmpul PID. Subtipul unui pachet de confirmare poate fi ACK (*Acknowledge*), NAK (*Negative Acknowledge*), STALL (*Stall*) sau NYET (*No Response Yet*). Aceste subtipuri sunt descrise în tabelul 4.2.



Figura 4.18. Structura unui pachet USB de confirmare.

4.7. Descriptori USB

Dispozitivele USB au o ierarhie de descriptori care descriu atributele lor. Acești descriptori sunt utilizați pentru raportarea atributelor dispozitivelor calculatorului gazdă. Un descriptor reprezintă o structură de date cu un format definit de standardele USB. Fiecare descriptor începe cu un octet care conține numărul total de octeți ai descriptorului, urmat de un octet care indică tipul descriptorului. Pe lângă descriptorii standard, dispozitivele USB pot returna și descriptori specifici unei clase de dispozitiv sau unui producător.

Principalele tipuri de descriptori standard sunt următoarele:

- Descriptori de dispozitiv;
- Descriptori de configurație;
- Descriptori de interfață;
- Descriptori ai punctelor terminale;
- Descriptori de tip și de caractere.

Ierarhia de descriptori are ca rădăcină la nivelul superior descriptorul de dispozitiv. La nivelul următor se află descriptorii de configurație; există câte un asemenea descriptor pentru fiecare configurație a dispozitivului. Pentru fiecare configurație pot exista unul sau mai mulți descriptori de interfață, în funcție de numărul de interfețe ale configurației respective. În sfârșit, pentru fiecare punct terminal al fiecărei interfețe există câte un descriptor al punctului terminal (figura 4.19).

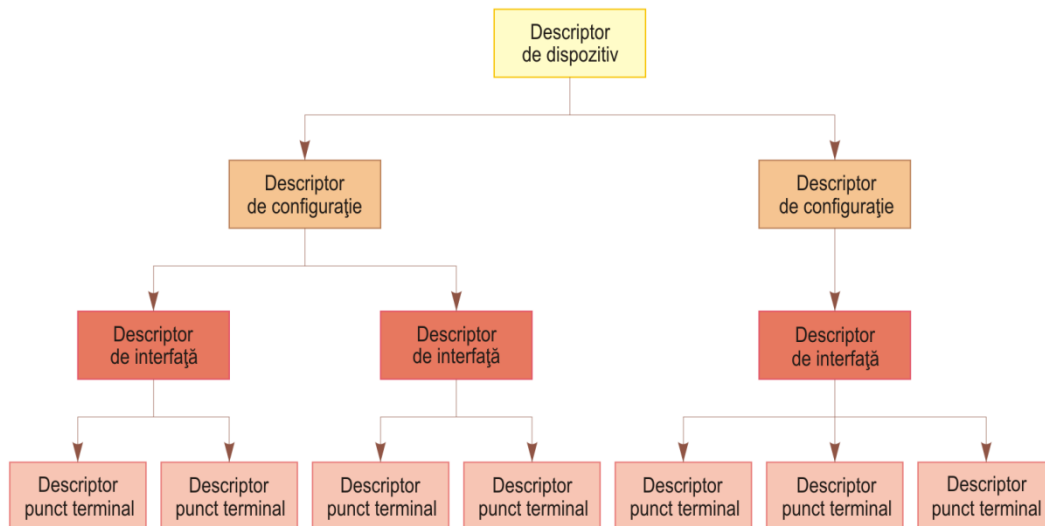


Figura 4.19. Ierarhia de descriptori ai unui dispozitiv USB.

Descriptorul de dispozitiv conține informații generale despre dispozitivul USB; aceste informații se referă la toate configurațiile dispozitivului. Un dispozitiv USB poate avea un singur descriptor de dispozitiv. Descriptorul de dispozitiv conține informații ca revizia standardului USB cu care este conform dispozitivul, clasa din care face parte dispozitivul, subclasa dispozitivului, identificatorul producătorului (asignat de organizația USB-IF), identificatorul produsului (asignat de producător), numărul de configurații posibile ale dispozitivului.

Un dispozitiv USB poate avea mai multe configurații, caracterizate prin atribute diferite. Un *descriptor de configurație* caracterizează o anumită configurație a dispozitivului. Acest descriptor conține informații ca numărul de interfețe ale configurației, dacă dispozitivul este alimentat de la magistrala USB sau de la o sursă de alimentare proprie și curentul maxim consumat de dispozitiv în cazul în care acesta este alimentat de la magistrala USB.

În faza de enumerare, calculatorul gazdă citește descriptorii de configurație și pe baza lor validează o anumită configurație a dispozitivului. La un moment dat, poate fi validată o singură configurație. De exemplu, un dispozitiv poate avea o configurație pentru alimentarea de la magistrală și o configurație pentru alimentarea de la o sursă proprie. Dacă dispozitivul este conectat la un calculator alimentat de la rețea, driverul dispozitivului poate selecta prima configurație, permițând alimentarea dispozitivului fără conectarea acestuia la o sursă de alimentare separată. Dacă dispozitivul este conectat la un calculator portabil, driverul poate selecta a doua configurație, care necesită conectarea dispozitivului la o sursă de alimentare proprie.

Un *descriptor de interfață* caracterizează un set de puncte terminale din cadrul unei configurații, care sunt grupate într-o interfață. De exemplu, considerăm un dispozitiv multifuncțional conținând un fax, un scanner și o imprimantă. Unul din descriptorii de interfață poate caracteriza funcția de fax a dispozitivului, un al doilea descriptor de interfață poate caracteriza funcția de scanner, iar un al treilea descriptor de interfață poate caracteriza funcția de imprimantă. Un dispozitiv poate avea mai mulți descriptori de interfață validați simultan. Un descriptor de interfață nu este accesibil direct, ci este returnat ca parte a unui descriptor de configurație.

O interfață poate conține setări alternative, care permit modificarea punctelor terminale asociate interfeței sau a caracteristicilor acestora după configurarea dispozitivului. În mod implicit, pentru o interfață este selectată setarea alternativă cu numărul zero. Setările alternative permit modificarea unei interfețe a dispozitivului în timp ce alte interfețe rămân în funcțiune, ceea ce este mai avantajos decât utilizarea unor configurații diferite. Dacă o configurație conține setări alternative pentru una sau mai multe din interfețele sale, pentru fiecare setare trebuie să se includă în descriptorul de configurație un descriptor de interfață separat și descriptorii asociați ai punctelor terminale.

Un descriptor de interfață conține informații ca numărul interfeței, numărul setării alternative a interfeței, numărul punctelor terminale utilizate de interfață, codul clasei și codul subclasei (asignate de organizația USB-IF). Numărul punctelor terminale utilizate de interfață nu include punctul terminal zero.

Fiecare punct terminal utilizat de o interfață are câte un descriptor propriu. În principiu, un *descriptor al unui punct terminal* conține informațiile necesare calculatorului gazdă pentru determinarea cerințelor lărimii de bandă a punctului terminal. Descriptorul unui punct terminal conține informații ca numărul (adresa) punctului terminal, direcția acestuia, tipul transferului utilizat pentru comunicarea cu punctul terminal, dimensiunea maximă a pachetelor pe care le poate transmite sau recepționa punctul terminal și intervalul de timp pentru interogarea punctului terminal de către calculatorul gazdă în vederea transferurilor de date. Descriptorii punctelor terminale nu sunt accesibili direct, fiind returnați ca parte a unui descriptor de configurație. Punctul terminal zero nu are un descriptor propriu.

Descriptorii de tip șir de caractere sunt opționali. Acești descriptori furnizează informații despre dispozitivul USB într-o formă care poate fi afișată direct. Referirea la descriptorii de tip șir de caractere se realizează prin valori de index din cadrul descriptorilor de dispozitiv, de configurație și de interfață. În cazul în care pentru un dispozitiv USB nu se utilizează descriptori de tip șir de caractere, toate referințele la asemenea descriptori trebuie setate la zero în descriptorii de dispozitiv, de configurație și de interfață.

Descriptorii de tip șir de caractere au o structură standard. Primul octet din fiecare descriptor indică dimensiunea în octeți a descriptorului, iar al doilea octet indică tipul descriptorului. În cadrul unui descriptor, șirul de caractere începe de la deplasamentul 2. Fiecare șir de caractere este codificat în formatul UNICODE, în modul definit de Consorțiul Unicode (<http://www.unicode.com>). Șirurile de caractere nu sunt terminate cu un octet NULL. Dimensiunea (în octeți) a unui șir se poate determina ca fiind $L-2$, unde L este dimensiunea în octeți a descriptorului. Descriptorii pot conține șiruri de caractere în diferite limbi. Atunci când se solicită un descriptor de tip șir de caractere, limba dorită trebuie specificată printr-un identificator al limbii (LANGID) de 16 biți.

4.8. Procesul de enumerare

Atunci când un dispozitiv USB este conectat la magistrala USB sau este deconectat de la magistrală, calculatorul gazdă execută un proces numit *enumerare* pentru a determina modificările apărute în configurația sistemului USB. La conectarea unui dispozitiv USB la magistrală, se execută următoarele operații:

1. Distribuitorul la care este conectat dispozitivul detectează conectarea acestuia cu ajutorul rezistenței utilizate ca terminator al magistralei de la capătul dispozitivului. Distribuitorul informează calculatorul gazdă asupra modificării apărute. În acest moment, dispozitivul USB este alimentat de la magistrală și portul la care este conectat este dezactivat.
2. Calculatorul gazdă determină tipul modificării apărute și portul la care a apărut modificarea prin interogarea distribuitorului.
3. Calculatorul gazdă așteaptă un timp de cel puțin 100 ms pentru ca tensiunea de alimentare a dispozitivului să devină stabilă, după care transmite o comandă de validare și de resetare a portului. În cazul unui dispozitiv cu viteză ridicată (480 Mbiți/s), acesta inițiază un protocol electric special pentru stabilirea unei legături cu această viteză. Dacă acest protocol electric nu este inițiat sau nu se termină cu succes, comunicația se va realiza cu viteza normală (12 Mbiți/s).
4. După terminarea procedurii de resetare, portul este validat. Dispozitivul se află acum în starea implicită și poate absorbi un curent de maxim 100 mA de pe linia V_{BUS} a magistralei. Dispozitivul va răspunde la tranzațiile cu adresa implicită zero.
5. Calculatorul gazdă solicită descriptorul de dispozitiv, iar dispozitivul transmite acest descriptor prin intermediul conductei implicite.

6. Calculatorul gazdă asignează o adresă unică dispozitivului.
7. Calculatorul gazdă solicită descriptorii de configurație, iar dispozitivul transmite calculatorului gazdă acești descriptori.
8. Pe baza informațiilor de configurație, calculatorul gazdă asignează o anumită configurație dispozitivului. Dispozitivul se află acum în starea configurată și toate punctele terminale ale acestei configurații sunt configurate conform caracteristicilor specificate în descriptorii acestora. Dispozitivul este pregătit pentru utilizare și poate absorbi de pe linia V_{BUS} a magistralei curentul specificat pentru configurația selectată.

Observație

- Detaliile operațiilor executate în timpul procesului de enumerare pot varia în funcție de sistemul de operare.

Atunci când un dispozitiv USB este deconectat de la un port USB, distribuitorul informează calculatorul gazdă asupra modificării apărute. Calculatorul gazdă dezactivează portul respectiv și actualizează informațiile sale despre topologia magistralei.

4.9. Comenzi USB

Fiecare dispozitiv USB trebuie să răspundă la comenzile USB transmise de calculatorul gazdă. Comenzile sunt transmise de calculatorul gazdă pe conducta implicită de control, iar răspunsul este transmis de dispozitiv pe aceeași conductă. Specificațiile USB definesc o serie de comenzi standard care trebuie implementate de fiecare dispozitiv USB. În plus, pot exista comenzi specifice pentru diferite clase de dispozitive. De asemenea, producătorii dispozitivelor pot defini comenzi proprii.

Transmiterea și execuția unei comenzi USB poate necesita două sau trei faze de transfer. În prima fază, calculatorul gazdă transmite comanda și parametrii acesteia într-un pachet SETUP utilizând un transfer de control. În faza a doua, care este opțională, se transferă date de la calculatorul gazdă la dispozitivul USB sau invers. În faza a treia, se transferă informații de stare de la dispozitivul USB la calculatorul gazdă sau invers. Direcția transferului din faza de stare este inversă direcției transferului din faza de date. Dacă faza de date lipsește, în faza de stare direcția transferului este de la dispozitivul USB la calculatorul gazdă.

Fiecare pachet SETUP conține opt octeți. Structura unui pachet SETUP este ilustrată în tabelul 4.3. Dimensiunea câmpurilor este indicată în octeți. Biții din cadrul octetului care descrie caracteristicile comenzii sunt indicați prin b7..b0.

Tabelul 4.3. Structura unui pachet SETUP.

Offset	Câmp	Dimensiune	Descriere
0	bmRequestType	1	Caracteristici ale comenzii: b7: Direcția transferului de date 0 = De la calculator la dispozitiv 1 = De la dispozitiv la calculator b6..b5: Tipul comenzii 0 = Standard 1 = Specific clasei 2 = Definit de producător 3 = Rezervat b4..b0: Destinația comenzii 0 = Dispozitiv 1 = Interfață 2 = Punct terminal 3 = Altă destinație 4..31 = Rezervat
1	bRequest	1	Codul comenzii
2	wValue	2	Parametru; variază în funcție de comandă
4	wIndex	2	Parametru; în mod tipic, numărul interfeței sau al punctului terminal
6	wLength	2	Numărul octeților de transferat în faza de date

Comenzile USB pot fi destinate dispozitivelor, interfețelor sau punctelor terminale. În funcție de destinație, aceeași comandă poate avea efecte diferite.

4.9.1. Comenzi standard destinate dispozitivelor

Comenzile standard care sunt destinate dispozitivelor sunt prezentate sintetic în tabelul 4.4. Coloana Data indică datele transferate în faza de date.

Tabelul 4.4. Comenzi standard destinate dispozitivelor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0000	GET_STATUS (0x00)	0	0	2	Stare dispozitiv
0000 0000	CLEAR_FEATURE (0x01)	Selector caracteristică	0	0	–
0000 0000	SET_FEATURE (0x03)	Selector caracteristică	0	0	–
0000 0000	SET_ADDRESS (0x05)	Adresă dispozitiv	0	0	–
1000 0000	GET_DESCRIPTOR (0x06)	Tip și index descriptor	0 sau LANGID	Lungime descriptor	Descriptor
0000 0000	SET_DESCRIPTOR (0x07)	Tip și index descriptor	0 sau LANGID	Lungime descriptor	Descriptor
1000 0000	GET_CONFIGURATION (0x08)	0	0	1	Număr configurație
0000 0000	SET_CONFIGURATION (0x09)	Număr configurație	0	0	–

Comanda *Get_Status* destinată unui dispozitiv returnează un cuvânt de stare de la dispozitiv la calculatorul gazdă în timpul fazei de date. Structura cuvântului returnat este ilustrată mai jos.

15	2	1	0
Rezervat		Remote Wakeup	Self Powered

Bitul Self Powered indică prin valoarea 1 faptul că dispozitivul este alimentat de la o sursă proprie. Dacă acest bit este 0, dispozitivul este alimentat de la magistrala USB. Bitul Self Powered nu poate fi modificat prin comenzile *Set_Feature* sau *Clear_Feature*. Bitul Remote Wakeup indică prin valoarea 1 faptul că dispozitivul are caracteristica *Remote Wakeup* validată. Atunci când această caracteristică este validată, dispozitivul poate determina trecerea calculatorului gazdă din starea inactivă (*Suspend*) în starea activă. Atunci când bitul Remote Wakeup este 0, caracteristica *Remote Wakeup* a dispozitivului este invalidată. Bitul Remote Wakeup poate fi modificat prin comenzile *Set_Feature* și *Clear_Feature* utilizând selectorul DEVICE_REMOTE_WAKEUP (cu valoarea 0x01).

Comenzile *Clear_Feature* și *Set_Feature* destinate unui dispozitiv permit invalidarea, respectiv validarea anumitor caracteristici ale dispozitivului. Câmpul wValue al comenzilor trebuie să conțină selectorul caracteristicii respective. Caracteristicile care pot fi invalidate sau validate sunt *Remote Wakeup*, cu selectorul DEVICE_REMOTE_WAKEUP (0x01), și *Test Mode*, cu selectorul TEST_MODE (0x02). Caracteristica *Test Mode* este implementată doar de dispozitivele USB cu viteză ridicată și de distribuitoare, permițând acestora să efectueze diferitele teste de conformitate cu standardul USB la nivelul interfeței electrice. Caracteristica *Test Mode* nu poate fi invalidată prin comanda *Clear_Feature*.

Comanda *Set_Address* este utilizată în timpul procesului de enumerare pentru asignarea unei adrese unice dispozitivului USB. Adresa trebuie specificată în câmpul wValue și poate avea valoarea maximă 127. Adresa dispozitivului va fi setată doar după terminarea cu succes a fazei de stare a comenzii *Set_Address*.

Comanda *Get_Descriptor* returnează un descriptor specificat, dacă acesta există. Câmpul wValue trebuie să specifice tipul descriptorului în octetul superior și indexul descrip-

torului în octetul inferior. Tipul descriptorului poate specifica un descriptor de dispozitiv (0x01), un descriptor de configurație (0x02) sau un descriptor de tip șir de caractere (0x03). Indexul descriptorului selectează un anumit descriptor în cazul în care dispozitivul implementează mai mulți descriptori de același tip. Acest index se utilizează numai pentru un descriptor de configurație sau de tip șir de caractere. Pentru un descriptor de dispozitiv, indexul trebuie setat la 0. În cazul descriptorilor de tip șir de caractere, câmpul *wIndex* trebuie să specifice identificatorul limbii (LANGID), iar în cazul altor descriptori acest câmp trebuie setat la 0. Câmpul *wLength* trebuie să conțină numărul de octeți care trebuie returnați. Dacă descriptorul are o lungime mai mare decât valoarea din câmpul *wLength*, dispozitivul returnează doar numărul specificat de octeți.

Dacă în câmpul *wValue* al comenzii *Get_Descriptor* se specifică un descriptor de configurație, dispozitivul va returna descriptorul de configurație, toți descriptorii de interfață pentru acea configurație și toți descriptorii punctelor terminale pentru toate interfețele. După descriptorul de configurație, se va returna primul descriptor de interfață. După acest descriptor, vor urma descriptorii punctelor terminale pentru prima interfață. Dacă există alte interfețe, se va returna descriptorul lor de interfață, urmat de descriptorii punctelor terminale ai acestora.

Comanda *Set_Descriptor* este opțională și se poate utiliza pentru actualizarea unor descriptori existenți sau adăugarea unor noi descriptori. Semnificația câmpurilor comenzii este aceeași ca și la comanda *Get_Descriptor*.

Comanda *Get_Configuration* returnează numărul configurației curente a dispozitivului. Valoarea este returnată într-un octet în timpul fazei de date. Dacă valoarea returnată este 0, dispozitivul nu este configurat.

Comanda *Set_Configuration* permite validarea unei anumite configurații a dispozitivului. Numărul configurației dorite trebuie specificat în octetul inferior al câmpului *wValue*. Acest număr trebuie să corespundă cu numărul unei configurații dintr-un descriptor de configurație. Octetul superior al câmpului *wValue* este rezervat.

4.9.2. Comenzi standard destinate interfețelor

Tabelul 4.5 prezintă comenzile standard care sunt destinate interfețelor.

Pentru toate comenzile destinate interfețelor, câmpul *wIndex* trebuie să conțină în octetul inferior numărul interfeței la care se referă comanda.

Tabelul 4.5. Comenzi standard destinate interfețelor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0001	GET_STATUS (0x00)	0	Interfață	2	Stare interfață
0000 0001	CLEAR_FEATURE (0x01)	Selector caracteristică	Interfață	0	–
0000 0001	SET_FEATURE (0x03)	Selector caracteristică	Interfață	0	–
1000 0001	GET_INTERFACE (0x0A)	0	Interfață	1	Setare alternativă
0000 0001	SET_INTERFACE (0x0B)	Setare alternativă	Interfață	0	–

Comanda *Get_Status* destinată unei interfețe returnează doi octeți cu valoarea 0 în timpul fazei de date. Acești octeți sunt rezervați pentru versiunile viitoare ale standardului USB.

Comenzile *Clear_Feature* și *Set_Feature* destinate unei interfețe permit invalidarea, respectiv validarea anumitor caracteristici ale interfeței. Versiunea 2.0 a standardului USB nu permite invalidarea sau validarea nici uneia din caracteristicile interfețelor.

Comanda *Get_Interface* returnează setarea alternativă selectată pentru interfața specificată. Setarea alternativă este returnată într-un octet în timpul fazei de date. Principiul setărilor alternative a fost descris în secțiunea 4.7.

Comanda *Set_Interface* permite selectarea unei setări alternative pentru interfața specificată. Câmpul *wValue* trebuie să conțină setarea alternativă care trebuie selectată.

4.9.3. Comenzi standard destinate punctelor terminale

Comenzile standard care sunt destinate punctelor terminale sunt prezentate în tabelul 4.6.

Tabelul 4.6. Comenzi standard destinate punctelor terminale.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0010	GET_STATUS (0x00)	0	Punct terminal	2	Stare punct terminal
0000 0010	CLEAR_FEATURE (0x01)	Selector caracteristică	Punct terminal	0	–
0000 0010	SET_FEATURE (0x03)	Selector caracteristică	Punct terminal	0	–
1000 0010	SYNCH_FRAME (0x0C)	0	Punct terminal	2	Număr cadru

În cazul comenzilor destinate punctelor terminale, câmpul *wIndex* trebuie să conțină numărul punctului terminal la care se referă comanda și direcția acestuia. Formatul câmpului *wIndex* este ilustrat mai jos.

15	8	7	6	4	3	0
Rezervat			Dir	Rezervat		Nr. punct terminal

Bitul *Dir* indică prin valoarea 0 un punct terminal de ieșire și prin valoarea 1 un punct terminal de intrare. În cazul unui punct terminal de control, bitul *Dir* trebuie setat la 0. Biții 3..0 specifică numărul punctului terminal.

Comanda *Get_Status* destinată unui punct terminal returnează doi octeți de stare în timpul fazei de date. Bitul 0 al octetului inferior returnat indică prin valoarea 1 faptul că punctul terminal este oprit (are caracteristica *Halt* setată la 1).

Comenzile *Clear_Feature* și *Set_Feature* destinate unui punct terminal permit invalidarea, respectiv validarea unor caracteristici ale punctului terminal respectiv. Câmpul *wValue* al comenzilor trebuie să conțină selectorul caracteristicii respective. Versiunea 2.0 a standardului USB permite invalidarea (resetarea la 0) și validarea (setarea la 1) unei singure caracteristici care se referă la un punct terminal, și anume caracteristica *Halt*, pentru care trebuie să se utilizeze selectorul *ENDPOINT_HALT* (0x00). Această caracteristică va fi resetată la 0 în mod automat după o comandă *Set_Configuration* sau *Set_Interface*.

Comanda *Synch_Frame* determină returnarea de către un punct terminal a numărului unui cadru de sincronizare. Atunci când un punct terminal permite transferuri izocrone, poate fi necesar ca dimensiunea transferurilor să varieze conform unui model repetitiv. Prin această comandă, punctul terminal returnează calculatorului gazdă numărul cadrului de la care începe modelul repetitiv.

4.10. Clasa de dispozitive USB HID

4.10.1. Prezentare generală a clasei de dispozitive HID

Clasa de dispozitive USB HID (*Human Interface Device*) constă în principal din dispozitive utilizate de operatori pentru controlul funcționării unui sistem de calcul. Exemple tipice de dispozitive din această clasă sunt următoarele:

- Tastaturi și dispozitive indicatoare: mouse, trackball;
- Comenzi de pe panouri frontale: comutatoare, butoane, glisoare;
- Comenzi aflate pe dispozitive cum sunt jocuri sau simulatoare: volane, pedale;
- Dispozitive de afișare: diode electroluminiscente, afișaje alfanumerice;
- Instrumente medicale: aparate cu ultrasunete;

- Dispozitive care nu necesită intervenție umană, dar care pot transmite date într-un format similar cu dispozitivele din clasa HID: cititoare pentru coduri de bare, termometre, aparate de măsură.

Clasa HID constă deci dintr-o categorie largă de dispozitive, cu caracteristici variate. În această clasă se pot încadra și multe dispozitive care nu sunt prevăzute cu o interfață umană. Prin utilizarea modelului clasei HID, este posibilă comunicarea unitară între calculatorul gazdă și diferite dispozitive, ceea ce permite dezvoltarea mai simplă a unor aplicații diverse. Diferitele sisteme de operare și, în particular, sistemele de operare *Windows*, pun la dispoziție drivere pentru dispozitivele din clasa HID, care se pot utiliza pentru comunicația cu aceste dispozitive. Utilizarea acestor drivere este avantajoasă, deoarece astfel se elimină necesitatea scrierii unor drivere pentru comunicația cu dispozitivele respective.

Ca orice dispozitiv USB, un dispozitiv din clasa HID poate fi sursa sau destinația unei tranzacții în fiecare cadru (1 ms) sau micro-cadru (125 μ s). O tranzacție poate conține mai multe pachete, dar este limitată ca dimensiune la 8 octeți pentru dispozitive cu viteză redusă, 64 octeți pentru dispozitive cu viteză normală și 1024 octeți pentru dispozitive cu viteză ridicată. Pentru dispozitivele cu viteză ridicată, este posibil să se execute două sau trei tranzacții în fiecare micro-cadru. Un transfer reprezintă mai multe tranzacții care creează un set de date cu o anumită structură și semnificație pentru dispozitiv. În cazul dispozitivelor din clasa HID, un transfer se numește *raport*. Datele rapoartelor au o anumită structură, care este specificată în descriptorii de raport (secțiunea 4.10.2).

Există trei tipuri de rapoarte: de intrare, de ieșire și de caracteristici. Un *raport de intrare* este transmis de un dispozitiv și conține date destinate aplicațiilor rulate pe calculatorul gazdă. Asemenea date sunt, de exemplu, coordonatele x și y de la un dispozitiv indicator. Un *raport de ieșire* este transmis de calculatorul gazdă unui dispozitiv și conține date de la aplicații care sunt destinate unor comenzi sau afișaje. Un *raport de caracteristici* conține date destinate unui dispozitiv sau date reprezentând starea unui dispozitiv. Spre deosebire de datele din rapoartele de intrare sau de ieșire, datele dintr-un raport de caracteristici sunt destinate utilizării de către utilitarele de configurare ale dispozitivelor și nu de către aplicații. De exemplu, valoarea ratei de repetare a unei taste poate fi o dată dintr-un raport de caracteristici.

Anumite dispozitive pot avea mai multe structuri de rapoarte. De exemplu, o tastatură cu un dispozitiv indicator integrat poate raporta în mod independent date referitoare la tastele apăsată și date referitoare la coordonate prin același punct terminal. Pentru diferențierea acestor structuri, se utilizează un identificator al raportului, care este o valoare de un octet care precedă fiecare raport. Pentru exemplul anterior, identificatorul permite driverului clasei de dispozitive HID să deosebească datele referitoare la taste de datele referitoare la coordonate.

Dezavantajul utilizării clasei HID este că driverele tipice ale acestei clase permit numai transferuri de întrerupere și rata maximă de transfer este limitată la o valoare inferioară lărimii de bandă a magistralei USB. Rata de transfer este limitată la 8.000 B/s (8 B/msec) pentru dispozitivele cu viteză redusă și la 64.000 B/s (64 B/msec) pentru dispozitivele cu viteză normală. Pentru dispozitivele cu viteză ridicată, presupunând că se execută trei tranzacții în fiecare micro-cadru, rata de transfer este limitată la aproximativ 23,4 MB/s ($3 \cdot 1024$ B/micro-cadru, sau $3 \cdot 1024 \cdot 8 = 24.576$ B/msec).

4.10.2. Descriptori specifici pentru clasa HID

Pe lângă descriptorii standard utilizați pentru toate clasele de dispozitive USB, există descriptori specifici utilizați pentru clasa de dispozitive HID. Astfel de descriptori sunt descriptorul HID, descriptorul de raport și descriptorul fizic. Figura 4.20 ilustrează poziția acestor descriptori în ierarhia descriptorilor USB standard.

Un descriptor HID conține numărul, lungimea și tipul descriptorilor subordonați specifici clasei HID pentru un dispozitiv. Numărul descriptorilor subordonați ai unui descriptor HID trebuie să fie cel puțin 1, deoarece trebuie să existe întotdeauna un descriptor de raport. Descriptorul HID mai conține informații ca versiunea specificațiilor clasei HID cu care este conform dispozitivul și codul țării pentru care este localizat dispozitivul.

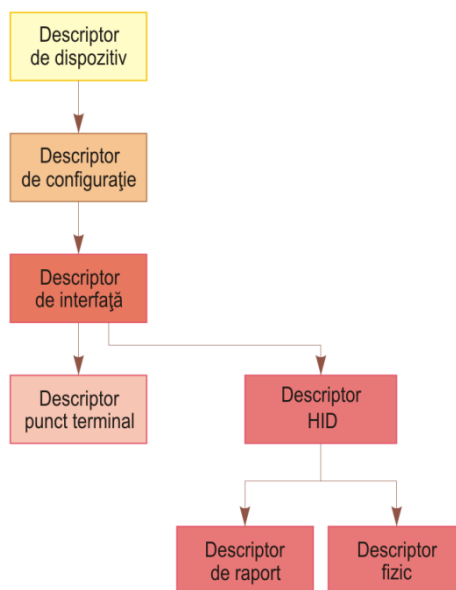


Figura 4.20. Poziția descriptorilor specifici clasei HID în ierarhia descriptorilor USB standard.

Un descriptor de raport conține articole care descriu dimensiunea și structura datelor raportate. Un asemenea descriptor furnizează informații despre datele raportate de dispozitiv pentru fiecare din funcțiile sale și despre datele care sunt destinate funcțiilor dispozitivului. Exemple de informații sunt dimensiunea datelor returnate, dacă datele sunt absolute sau relative, valorile minime și maxime ale datelor individuale. De asemenea, un descriptor de raport indică natura datelor raportate, de exemplu, dacă ele reprezintă coordonatele x și y .

Descriptorii fizici sunt opționali. Acești descriptori furnizează informații despre partea sau părțile corpului uman care se utilizează pentru activarea comenzilor dispozitivului. În plus, un descriptor fizic poate conține valori care cuantifică efortul care trebuie depus de utilizator pentru activarea diferitelor comenzi.

4.10.3. Comenzi USB specifice pentru clasa HID

Dispozitivele din clasa HID trebuie să implementeze comenzile USB standard *Get_Descriptor* și *Set_Descriptor*. Pe lângă aceste comenzi, dispozitivele din clasa HID pot implementa diferite comenzi specifice acestei clase. Aceste comenzi inițiază tranzacții care permit calculatorului gazdă determinarea posibilităților și a stării unui dispozitiv și setarea stării articolelor de ieșire și de caracteristici. Principalele comenzi specifice clasei HID sunt *Get_Report* și *Set_Report*. Implementarea comenzii *Get_Report* este obligatorie.

Tabelul 4.7 prezintă comenzile *Get_Report* și *Set_Report*.

Tabelul 4.7. Comenzile *Get_Report* și *Set_Report* specifice dispozitivelor din clasa HID.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1010 0001	GET_REPORT (0x01)	Tip raport și ID raport	Interfață	Lungime raport	Raport
0010 0001	SET_REPORT (0x09)	Tip raport și ID raport	Interfață	Lungime raport	Raport

Comanda *Get_Report* permite calculatorului gazdă recepționarea unui raport de la un dispozitiv prin conducta implicită de control. Câmpul *wValue* trebuie să conțină tipul raportului în octetul superior și identificatorul raportului (ID raport) în octetul inferior. Tipul raportului specifică un raport de intrare (0x01), un raport de ieșire (0x02), sau un raport de caracteristici (0x03). Dacă nu se utilizează identificatori ai rapoartelor, octetul inferior al câmpului *wValue* trebuie setat la 0.

Comanda *Get_Report* este utilă în momentul inițializării dispozitivului pentru determinarea stării caracteristicilor acestuia. Comanda nu este destinată pentru a interoga starea

dispozitivului la intervale regulate. Pentru rapoartele de intrare repetate, trebuie să se utilizeze o conductă de întrerupere de intrare. În mod opțional, pentru rapoartele de ieșire se poate utiliza o conductă de întrerupere de ieșire (secțiunea 4.10.4).

Comanda *Set_Report* permite calculatorului gazdă să transmită un raport la un dispozitiv pentru setarea stării unor comenzi. Semnificația câmpurilor acestei comenzi este aceeași ca și pentru comanda *Get_Report*.

4.10.4. Interfața cu dispozitivele din clasa HID

Un dispozitiv din clasa HID comunică cu driverul acestei clase fie prin conducta implicită de control, fie printr-o conductă de întrerupere. Dispozitivul utilizează conducta implicită de control pentru următoarele operații:

- Recepționarea comenzilor USB transmise de calculatorul gazdă și transmiterea răspunsului la aceste comenzi;
- Transmiterea datelor atunci când dispozitivul este interogată de driverul clasei HID prin comanda *Get_Report*;
- Recepționarea datelor de la calculatorul gazdă.

Driverul clasei HID utilizează o conductă de întrerupere pentru următoarele operații:

- Recepționarea unor date de la dispozitiv în mod asincron (date care nu au fost solicitate în mod explicit);
- Transmiterea cu o întârziere redusă a unor date la dispozitiv.

Utilizarea unei conducte de întrerupere de ieșire este opțională. Dacă un dispozitiv inițializează un punct terminal de întrerupere cu direcția de ieșire, atunci rapoartele de ieșire sunt transmise de calculatorul gazdă la dispozitiv prin această conductă. Dacă nu este disponibil un punct terminal de întrerupere cu direcția de ieșire, atunci rapoartele de ieșire sunt transmise de calculatorul gazdă prin punctul terminal de control, utilizând comenzi *Set_Report*.

4.11. Comunicația cu dispozitivele din clasa HID

La conectarea unui dispozitiv din clasa HID la un port USB al calculatorului, sistemul de operare execută procesul de enumerare, în modul descris în secțiunea 4.8. Acest proces nu implică utilizarea unor drivere specifice clasei HID sau specifice dispozitivului. În prima parte a procesului de enumerare, driverul USB solicită dispozitivului transmiterea descriptorului de dispozitiv, după care preia descriptorul și asignează o adresă unică dispozitivului. În continuare, driverul USB solicită dispozitivului transmiterea descriptorului de configurație și preia acest descriptor (care include descriptorii de interfață și descriptorii punctelor terminale). Din descriptorul de interfață, driverul USB determină clasa dispozitivului, iar dacă această clasă este HID, driverul USB predă controlul driverului clasei HID. Dacă descriptorul de interfață nu specifică o anumită clasă, atunci driverul USB selectează un driver corespunzător pentru dispozitiv pe baza identificatorului producătorului (VID – *Vendor Identifier*) și a identificatorului produsului (PID – *Product Identifier*), sau solicită utilizatorului specificarea unui driver specific dispozitivului.

După ce dispozitivul este recunoscut de sistemul de operare ca un dispozitiv din clasa HID, se pot utiliza funcții de sistem pentru scrierea unei aplicații de comunicație cu dispozitivul. Aceste funcții utilizează driverul dispozitivelor din clasa HID, astfel încât nu este necesară scrierea unui driver specific dispozitivului. În această secțiune se descriu operațiile necesare pentru comunicația cu un dispozitiv din clasa HID pentru un sistem de operare *Windows*. Aceste operații sunt diferite pentru alte sisteme de operare, cum sunt *Linux* sau *MacOS*.

Înainte de utilizarea unui dispozitiv din clasa HID, este necesară stabilirea comunicației cu acel dispozitiv, ceea ce presupune mai multe etape. Aceste etape sunt descrise în continuare.

1. Se apelează funcția `HidD_GetHidGuid()` pentru a obține identificatorul unic global GUID (*Globally Unique Identifier*) pentru dispozitivele din clasa HID. Parametrul acestei funcții este un pointer la o structură de tip `_GUID`, reprezentând un buffer în care va fi returnat identificatorul GUID. Această funcție este declarată în fișierul `hidsdi.h`, care face parte din pachetul WDK (*Windows Driver Kit*) al Microsoft. Pentru linkeditarea statică este necesară utilizarea fișierului de bibliotecă `hid.lib`. Pentru apelul funcției `HidD_GetHidGuid()`, adăugați următoarele linii în fișierul sursă:

```
struct _GUID GUID;
HidD_GetHidGuid(&GUID);
```

2. Se apelează funcția `SetupDiGetClassDevs()` pentru a obține informații despre toate dispozitivele din clasa HID care sunt atașate la sistem. Această funcție returnează un indicator (variabilă de tip `HANDLE`) la setul de informații despre lista de dispozitive. Primul parametru al acestei funcții este un pointer la identificatorul global GUID obținut în etapa anterioară. Această funcție este declarată în fișierul `SetupAPI.h`, iar pentru linkeditarea statică trebuie să se utilizeze fișierul de bibliotecă `SetupAPI.lib`. Pentru apelul funcției `SetupDiGetClassDevs()`, adăugați următoarele linii:

```
HANDLE PnPHandle;
PnPHandle = SetupDiGetClassDevs(&GUID, NULL, NULL,
                                DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);
```

Dacă funcția returnează valoarea `INVALID_HANDLE_VALUE`, afișați un mesaj de eroare. În acest caz, nu se poate stabili comunicația cu dispozitivul și operația este terminată.

3. Etapele 3-9 trebuie repetate într-o buclă pentru fiecare dispozitiv din clasa HID; conținutul de iterații al buclei poate fi setat, de exemplu, la 30. Se apelează funcția `SetupDiEnumDeviceInterfaces()` pentru a obține informații despre interfața unui dispozitiv din lista dispozitivelor din clasa HID. Primul parametru al acestei funcții este indicatorul la setul de informații conținând dispozitivele pentru care se solicită informații despre interfață; acest indicator s-a obținut în etapa 2. Al doilea parametru este opțional și poate fi `NULL`. Al treilea parametru este pointerul la identificatorul global GUID. Dispozitivul despre care se solicită informații este specificat prin cel de-al patrulea parametru reprezentând indexul (începând cu 0) în lista de dispozitive. Al cincilea și ultimul parametru este un pointer la o variabilă reprezentând o structură de tip `SP_DEVICE_INTERFACE_DATA` care va fi completată de funcție cu informații despre interfața dispozitivului. Înaintea apelului funcției, membrul `cbSize` al acestei structuri trebuie setat la `sizeof(SP_DEVICE_INTERFACE_DATA)`. În caz de succes, funcția returnează valoarea `TRUE`.

Observație

- După apelul funcției `SetupDiEnumDeviceInterfaces()`, determinați codul ultimei erori prin apelul funcției `GetLastError()`. În cazul în care codul ultimei erori este `ERROR_NO_MORE_ITEMS`, terminați bucla cu o instrucțiune `break` și continuați cu etapa 10.

4. Dacă apelul funcției din etapa 3 s-a realizat cu succes, se apelează funcția de sistem `SetupDiGetDeviceInterfaceDetail()` pentru a afla informații detaliate despre interfața dispozitivului selectat în etapa 3. Primul parametru al acestei funcții este indicatorul la setul de informații, care a fost obținut în etapa 2. Al doilea parametru este un pointer la structura cu informațiile despre interfața dispozitivului, structură care a fost completată în etapa 3. Al treilea parametru este un pointer la o variabilă reprezentând o structură de tip `SP_DEVICE_INTERFACE_DETAIL_DATA` care va fi completată de funcție cu informații detaliate despre interfața dispozitivului. Această structură conține doi membri, `cbSize` și `DevicePath`, al doilea membru fiind un șir de caractere de lungime variabilă terminat cu un octet zero. Înaintea apelului funcției, mem-

brul `cbSize` al acestei structuri trebuie setat la `sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA)`. Membrul `DevicePath` va fi completat de funcția apelată cu calea de acces la dispozitiv. Al patrulea parametru trebuie să specifice dimensiunea totală a structurii amintite anterior. Al cincilea parametru, *RequiredSize*, este un pointer la o variabilă de tip `DWORD` în care funcția va depune dimensiunea necesară a bufferului care va conține structura cu informațiile detaliate. Al șaselea și ultimul parametru este opțional și poate fi `NULL`.

Observație

- Funcția `SetupDiGetDeviceInterfaceDetail()` trebuie apelată de două ori, în modul descris pe pagina de documentație a funcției. Primul apel este efectuat pentru a determina dimensiunea necesară a bufferului în care funcția va depune informații detaliate despre interfața dispozitivului; pentru acest apel, se setează al treilea parametru la `NULL` și al patrulea parametru la zero. După primul apel, trebuie alocată o memorie cu dimensiunea corespunzătoare pentru acest buffer și funcția trebuie apelată din nou cu parametrii setați în mod normal.

În caz de succes, funcția returnează valoarea `TRUE`. Dacă funcția returnează valoarea `FALSE`, se afișează un mesaj de eroare, se eliberează memoria alocată pentru informațiile detaliate și se continuă cu următoarea iterație de la etapa 3.

5. Se apelează funcția `CreateFile()` pentru a deschide comunicația cu dispozitivul. Primul parametru al acestei funcții este numele dispozitivului, reprezentat prin calea de acces care a fost obținută în etapa 4 (membrul `DevicePath` al structurii completate de funcția `SetupDiGetDeviceInterfaceDetail()`). Al doilea parametru este modul de acces, care poate fi setat la `GENERIC_READ | GENERIC_WRITE`. Al treilea parametru este modul de partajare al dispozitivului, care poate fi setat la `FILE_SHARE_READ | FILE_SHARE_WRITE`. Al patrulea parametru este un pointer la o structură cu atributele de securitate; acest parametru este opțional și poate fi `NULL`. Al cincilea parametru reprezintă acțiunea de executat asupra unui dispozitiv care există sau nu există; acest parametru poate fi setat la `OPEN_EXISTING`. Al șaselea parametru reprezintă atributele dispozitivului și poate fi zero, iar al șaptelea și ultimul parametru este opțional și poate fi `NULL`. Dacă funcția `CreateFile()` returnează valoarea `INVALID_HANDLE_VALUE`, se afișează un mesaj de eroare, se eliberează memoria alocată pentru informațiile detaliate și se continuă cu următoarea iterație de la etapa 3.

Observație

- Pentru unele dispozitive (de exemplu, mouse), modul de acces trebuie setat la 0, iar modul de partajare trebuie setat la `FILE_SHARE_READ|FILE_SHARE_WRITE`.

6. Se eliberează memoria alocată în pasul 4 pentru informațiile detaliate despre interfața dispozitivului.
7. Dacă funcția `CreateFile()` a returnat un indicator valid de fișier, se determină șirul de identificare al dispozitivului HID apelând funcția `HidD_GetProductString()`. Parametrii acestei funcții sunt indicatorul returnat de funcția `CreateFile()`, pointerul la un buffer alocat de utilizator în care funcția va depune șirul de identificare și lungimea în octeți a bufferului alocat. Adăugați următoarele linii pentru apelul funcției `HidD_GetProductString()`:

```
CHAR cBuffer [256];
bRet = HidD_GetProductString(hFile, cBuffer, sizeof(cBuffer));
```

unde `hFile` este indicatorul returnat de funcția `CreateFile()`, iar `bRet` este o variabilă booleană.

8. Se compară șirul de identificare obținut în etapa 7 cu șirul de identificare al dispozitivului cu care trebuie să se realizeze comunicația. Trebuie să se țină cont că șirul de identificare al dispozitivului este reprezentat în codul Unicode. Dacă șirurile sunt identice, se termină bucla cu o instrucțiune `break` și se continuă cu etapa 10. Dacă șirurile sunt diferite, se continuă cu etapa 9.
9. Se închide fișierul deschis în etapa 5 apelând funcția `CloseHandle()`, se incrementează indexul dispozitivului și se continuă cu etapa 3 pentru a obține informații despre interfața următorului dispozitiv.
10. Se apelează funcția `SetupDiDestroyDeviceInfoList()` pentru eliberarea memoriei alocate pentru informațiile despre dispozitive. În acest moment, operația de stabilire a comunicației cu dispozitivul este terminată.

După stabilirea comunicației cu dispozitivul, aplicația poate prelua rapoarte de intrare de la dispozitiv prin apelul funcției `ReadFile()` sau `ReadFileEx()` și poate transmite rapoarte de ieșire la dispozitiv prin apelul funcției `WriteFile()`. Pentru apelul acestor funcții, trebuie utilizat indicatorul de fișier returnat de funcția `CreateFile()`.

Funcțiile `ReadFile()` și `ReadFileEx()` preiau rapoartele de intrare de la dispozitiv utilizând transferuri de întrerupere. Aceasta înseamnă că dispozitivul trebuie să transmită aceste rapoarte utilizând transferuri de întrerupere printr-un punct terminal de intrare. Primul octet din fiecare raport reprezintă identificatorul raportului. Este posibilă preluarea rapoartelor de intrare și prin utilizarea transferurilor de control. Pentru aceasta se poate utiliza funcția `HidD_GetInputReport()`, care va transmite dispozitivului o comandă `Get_Report`.

Funcția `WriteFile()` transmite rapoartele de ieșire prin comenzi `Set_Report`. Dacă dispozitivul nu are un punct terminal de întrerupere cu direcția de ieșire, această funcție utilizează transferuri de control, iar în caz contrar utilizează transferuri de întrerupere. Primul octet din fiecare raport reprezintă identificatorul raportului. Se poate utiliza și funcția `HidD_SetOutputReport()` pentru transmiterea unui raport de ieșire printr-un transfer de control.

4.12. Aplicații

4.12.1. Răspundeți la următoarele întrebări:

- a. Care este rolul terminatorilor utilizați la magistrala USB?
- b. Care este deosebirea dintre transferurile asincrone și transferurile izocrone pe magistrala USB?
- c. Ce informații conține un descriptor de dispozitiv și un descriptor de configurație?
- d. Care sunt avantajele și dezavantajele utilizării modelului clasei HID pentru comunicația cu perifericele?

4.12.2. Creați o aplicație *Windows* pentru afișarea șirurilor de identificare ale dispozitivelor din clasa HID conectate la sistem. Ca model pentru aplicația *Windows*, utilizați aplicația *AppScroll* disponibilă pe pagina laboratorului în arhiva *AppScroll.zip*. Executați următoarele operații pentru a crea proiectul aplicației:

1. În mediul de programare *Visual Studio 2022*, creați un nou proiect gol de tip *Windows Desktop* cu utilitarul *Windows Desktop Wizard*. Bifați opțiunea *Place solution and project in the same directory* pentru a evita crearea unui alt director pentru soluția creată.
2. Verificați dacă platforma activă a soluției este setată la *x64*.
3. Modificați proprietatea *Character Set* a proiectului deschizând fereastra de dialog *Properties*. În această fereastră, expandați opțiunea *Configuration Properties*, expandați opțiunea *Advanced*, selectați linia *Character Set* în panoul din dreapta și alegeți opțiunea *Not Set*.

4. Copiați în directorul proiectului fișierele din arhiva AppScroll.zip și adăugați la proiect toate fișierele copiate.
5. Copiați în directorul proiectului fișierele din arhiva HID8.1.zip, disponibilă pe pagina laboratorului. Adăugați la proiect fișierele antet SetupAPI.h și hidsdi.h.
6. Specificați fișierele SetupAPI.lib și hid.lib ca dependențe suplimentare pentru linker.
7. Deschideți fișierul sursă AppScroll.cpp, ștergeți directiva `#include "Hw.h"` și adăugați directive `#include` pentru a include fișierele antet SetupAPI.h și hidsdi.h.
8. În funcția `AppScroll()`, ștergeți secvențele pentru inițializarea bibliotecii HW cu funcția `HwOpen()` și pentru închiderea bibliotecii HW cu funcția `HwClose()`.
9. Selectați *Build* → *Build Solution* și verificați ca aplicația să fie construită fără erori.

În fișierul sursă `AppScroll.cpp`, scrieți o funcție pentru afișarea șirurilor de identificare ale dispozitivelor din clasa HID conectate la sistem. Parametrul de intrare al funcției este indicatorul la fereastra aplicației `hWnd`; funcția returnează o valoare întreagă. Pentru scrierea funcției, parcurgeți etapele descrise în secțiunea 4.11, cu următoarea modificare: în etapa 8, afișați șirul de identificare obținut în etapa 7 și apoi continuați cu etapa 9. Pentru detalii despre parametrii unei funcții, accesați documentația Windows Hardware Developer prin poziționarea cursorului pe numele funcției și apăsarea tastei F1.

După scrierea funcției, includeți apelul acestei funcții în funcția `AppScroll()` și apoi verificați funcționarea sa.

4.12.3. Extindeți aplicația 4.12.2 prin scrierea unei funcții pentru stabilirea comunicației cu placa de dezvoltare STM32L496 Discovery prin magistrala USB. Parametrul de intrare al funcției este indicatorul la fereastra aplicației `hWnd`. Funcția este similară cu funcția scrisă pentru aplicația 4.12.2, cu excepția faptului că în etapa 8 compară șirul de identificare obținut în etapa 7 cu șirul de identificare al plăcii de dezvoltare în locul afișării șirului. Placa este configurată ca un dispozitiv HID cu șirul de identificare “Keil MCB2140 HID”.

Observații

- În etapa 5, la apelul funcției `CreateFile()`, al doilea parametru reprezentând modul de acces trebuie setat la `GENERIC_READ | GENERIC_WRITE`, iar al treilea parametru reprezentând modul de partajare trebuie setat la `FILE_SHARE_READ | FILE_SHARE_WRITE`.
- Modificați variabila `hFile` reprezentând indicatorul returnat de funcția `CreateFile()` la o variabilă globală, în scopul utilizării acesteia de către funcțiile care vor fi scrise pentru aplicațiile următoare.

Funcția returnează valoarea `TRUE` în cazul în care comunicația cu placa a fost stabilită și valoarea `FALSE` în caz contrar. După scrierea funcției, includeți apelul acestei funcții în funcția `AppScroll()` și afișați un mesaj indicând dacă a fost stabilită comunicația cu placa. Rulați aplicația fără ca placa de dezvoltare STM32L496 Discovery să fie conectată la calculator. Conectați apoi placa la calculator prin două cabluri USB. Rulați din nou aplicația și verificați dacă a fost stabilită comunicația cu placa de dezvoltare.

4.12.4. Extindeți aplicația 4.12.3 prin scrierea unei funcții pentru citirea și afișarea stării butoanelor de pe placa de dezvoltare STM32L496 prin magistrala USB. Funcția nu are parametri de intrare și utilizează indicatorul returnat de funcția `CreateFile()` din aplicația 4.12.3 pentru a apela funcția `HidD_GetInputReport()` sau `ReadFile()` în scopul preluării unui raport de intrare de doi octeți. Primul octet al raportului de intrare este identificatorul raportului (0x00), iar al doilea octet conține starea butoanelor dispozitivului joystick de pe placă. Placa transmite în mod repetat octetul de stare utilizând transferuri de întrerupere. Cele

cinci butoane ale dispozitivului joystick sunt asignate biților octetului de stare după cum urmează:

Bitul 0: stânga;
 Bitul 1: dreapta;
 Bitul 2: selecție (butonul central);
 Bitul 3: sus;
 Bitul 4: jos.

Atunci când un buton este apăsat, bitul său de stare este 1. Funcția afișează octetul de stare și returnează valoarea 0 dacă operația s-a terminat cu succes sau valoarea 1 dacă execuția funcției `HidD_GetInputReport()` sau `ReadFile()` s-a terminat cu o eroare.

După scrierea funcției, includeți apelul acestei funcții în funcția `AppScroll()` și verificați corectitudinea sa conectând placa de dezvoltare STM32L496 Discovery la calculator.

4.12.5. Extindeți aplicația 4.12.4 cu o funcție pentru setarea stării celor două diode LED controlate de utilizator, LD1 și LD2, de pe placa de dezvoltare STM32L496 Discovery. Parametrul de intrare al funcției este un octet reprezentând starea dorită a diodelor LED. Funcția utilizează indicatorul returnat de funcția `CreateFile()` din aplicația 4.12.3 pentru a apela funcția `HidD_SetOutputReport()` sau `WriteFile()` pentru transmiterea unui raport de ieșire de doi octeți. Primul octet al raportului de ieșire trebuie să fie identificatorul raportului (0x00), iar al doilea octet trebuie să fie starea dorită a diodelor LED. Dioda LED LD1 (portocalie) este asignată bitului 0, iar dioda LED LD2 (verde) este asignată bitului 1 al octetului reprezentând starea dorită. Pentru aprinderea unei diode, bitul corespunzător trebuie setat la 1. Funcția returnează valoarea 0 dacă operația s-a terminat cu succes sau valoarea 1 dacă execuția funcției `HidD_SetOutputReport()` sau `WriteFile()` s-a terminat cu o eroare.

După scrierea funcției, includeți apelul acestei funcții în funcția `AppScroll()`, starea dorită a diodelor LED fiind definită ca o constantă.

4.12.6. Extindeți funcția pentru afișarea șirurilor de identificare ale dispozitivelor din clasa HID, scrisă pentru aplicația 4.12.2, pentru a afișa informații suplimentare despre aceste dispozitive. Executați operațiile descrise în continuare pentru determinarea și afișarea informațiilor suplimentare. Consultați paginile Windows Hardware Developer pentru detalii despre funcțiile care trebuie apelate.

Observație

- Operațiile următoare trebuie executate pentru fiecare dispozitiv detectat din clasa HID, după afișarea șirului său de identificare.
1. Apelați funcția `HidD_GetAttributes()` pentru determinarea atributelor dispozitivului. Înaintea apelului funcției, declarați o variabilă de tip `HIDD_ATTRIBUTES` și inițializați membrul `Size` al acesteia cu dimensiunea variabilei. Primul parametru al funcției este indicatorul returnat de funcția `CreateFile()`, iar al doilea parametru este un pointer la variabila de tip `HIDD_ATTRIBUTES`. Funcția va completa ceilalți membri ai acestei variabile (`VendorID`, `ProductID` și `VersionNumber`) cu identificatorul producătorului, identificatorul produsului și numărul reviziei dispozitivului. În caz de succes, când funcția returnează valoarea `TRUE`, afișați identificatorul producătorului, identificatorul produsului și numărul reviziei dispozitivului.
 2. Apelați funcția `HidD_GetPreparsedData()` pentru a obține un pointer la un buffer alocat de funcție, conținând date din descriptorul de raport al dispozitivului. Înaintea apelului funcției, declarați o variabilă de tip `PHIDP_PREPARSED_DATA`; acesta este un pointer la un buffer care va fi înscris de funcție cu datele din descriptorul de raport. Primul parametru al funcției este indicatorul returnat de funcția `CreateFile()`, iar al doilea parametru este adresa pointerului de tip `PHIDP_PREPARSED_DATA`.

Funcția va inițializa pointerul la bufferul conținând datele descriptorului. În caz de succes, funcția returnează valoarea TRUE.

3. Dacă apelul funcției `HidD_GetPreparedData()` s-a realizat cu succes, apelați funcția `HidP_GetCaps()` pentru a determina informații despre posibilitățile dispozitivului. Înaintea apelului funcției, declarați o variabilă de tip `HIDP_CAPS`; aceasta este o structură care va fi completată de funcție cu informații despre posibilitățile dispozitivului. Primul parametru al funcției este pointerul de tip `PHIDP_PREPARED_DATA` care a fost definit în etapa 2, iar al doilea parametru este un pointer la variabila de tip `HIDP_CAPS`. În caz de succes, când funcția returnează valoarea `HIDP_STATUS_SUCCESS` de tip `NTSTATUS`, afișați următoarele informații din structura `HIDP_CAPS`: pagina de utilizare (membrul `UsagePage`); identificatorul de utilizare (membrul `Usage`); lungimea rapoartelor de intrare; lungimea rapoartelor de ieșire; lungimea rapoartelor de caracteristici.
4. Apelați funcția `HidD_FreePreparedData()` pentru eliberarea memoriei alocate pentru bufferul conținând datele din descriptorul de raport al dispozitivului. Parametrul acestei funcții este pointerul de tip `PHIDP_PREPARED_DATA` care a fost definit în etapa 2.

Bibliografie

- [1] Allman, S., "Using the HID class eases the job of writing USB device drivers", EDN, September 19, 2002, <http://m.eet.com/media/1138623/18538-243218.pdf>.
- [2] Axelson, J., *USB Complete. The Developer's Guide*, Fifth Edition, 2015, <http://janaxelson.com/usbc.htm>.
- [3] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, "Universal Serial Bus Specification", Revision 2.0, 2000, http://www.usb.org/developers/docs/usb20_docs/usb_20_0702115.zip.
- [4] Crider, M., "USB 3.1 have you confused? Here's everything you need to know about the standard", Digital Trends, 2015, <http://www.digitaltrends.com/computing/what-is-usb-3-1-when-will-it-be-released-and-what-will-it-do-for-pcs/>.
- [5] DataPro International Inc., "USB 3.0 Explained", 2015, http://www.datapro.net/techinfo/usb_3_explained.html.
- [6] Dong, N., "USB Type-C: One cable to connect them all", CNET, 2015, <http://www.cnet.com/news/usb-type-c-one-cable-to-connect-them-all/>.
- [7] Hyde, J., "Learning USB by Doing", Intel Corporation., <http://www.devasys.com/PD11x/JHWP.pdf>.
- [8] Microsoft Corp., MSDN Library, 2015, <http://msdn.microsoft.com/library/>.
- [9] Microsoft Corp., WDK and WinDbg downloads, 2015, <https://msdn.microsoft.com/en-us/windows/hardware/hh852365>.
- [10] Peacock, C., "On-The-Go Supplement – Point-to-Point Connectivity for USB", Beyond Logic, 2005, <http://retired.beyondlogic.org/usb/otghost.htm>.
- [11] Peacock, C., "USB in a Nutshell – Making Sense of the USB Standard", Beyond Logic, 2010, <http://www.beyondlogic.org/usbnutshell/>.
- [12] Rosch, W. L., *Hardware Bible*, Sixth Edition, Que Publishing, 2003.
- [13] USB Implementers Forum, Inc., "A Technical Introduction to USB 2.0", http://www.usb.org/developers/usb20/developers/whitepapers/usb_20g.pdf.

- [14] USB Implementers Forum, Inc., “Device Class Definition for Human Interface Devices (HID)”, Version 1.11, 2001, http://www.usb.org/developers/hidpage/HID1_11.pdf.
- [15] USB Implementers Forum, Inc., “HID Usage Tables”, Version 1.12, 2004, http://www.usb.org/developers/hidpage/Hut1_12v2.pdf.
- [16] USB Implementers Forum, Inc., “Introduction to USB On-The-Go”, http://www.usb.org/developers/othego/USB_OTG_Intro.pdf.
- [17] USB Implementers Forum, Inc., “On-The-Go and Embedded Host Supplement to the USB 2.0 Specification”, Revision 2.0, 2009, http://www.usb.org/developers/othego/USB_OTG_and_EH_2-0.pdf.
- [18] USB Implementers Forum, Inc., “SuperSpeed USB”, <http://www.usb.org/developers/ssusb>.
- [19] USB Implementers Forum, Inc., “USB 2.0 Specification Engineering Change Notice (ECN) #1: Mini-B Connector”, 2000, <http://www.usb.org/developers/docs/ecn1.pdf>.
- [20] USB Implementers Forum, Inc., “USB Frequently Asked Questions”, <http://www.usb.org/developers/usbfaq/>.
- [21] USB Implementers Forum, Inc., “USB On-The-Go”, <http://www.usb.org/developers/othego/>.