IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Antar Muka USB

Aman Subanjar
amansubanjar@gmail.com

Lisensi Dokumen:
Copyright © 2003-2007 IlmuKomputer.Com
Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Sejak diperkenalkan tahun 1998 silam, USB (Universal Serial Bus) mengalami perkembangan yang pesat, terutama dari sisi kecepatan transfer data dan kuantitas perangkat keras yang menggunakan sistem antarmuka ini. Sayangnya keunggulan-keunggulan yang ditawarkan oleh antarmuka USB (USB interface) harus dibayar oleh keruwetan transfer data baik dari sisi host (komputer) maupun dari sisi device (peripheral). Sebagai pengembangan dari antarmuka port serial, USB mempunyai nilai kesulitan tersendiri dari sisi pengembang karena pembuatan device yang menggunakan antarmuka USB tidak bisa dilepaskan dari desain hardware dan software. Port USB memang dapat dipastikan ada di setiap PC keluaran tahun 2000 ke atas. Namun demikian rancangan alat yang sifatnya desain personal menggunakan port ini jarang ditemui. Hal itu dikarenakan keruwetan yang diusung teknologi USB dari pada pendahulunya paralel dan serial. Buku ini disusun untuk para designer sistem embedded, hobbiest, dan siapapun yang ingin atau bahkan sudah terjun ke dunia interface USB. Berdasarkan pengalaman penulis sendiri, pertama kali merencanakan penggunaan jenis interface ini adalah ketika bingung memulai dari mana. Membaca referensi tentang protokol USB sangatlah membingungkan dan terlalu banyak rule protokol yang harus dipelajari.

Tulisan ini selesai pada tahun 2010 sehingga perkembangan USB versi 3.0 belum dibahas. Beberapa istilah dalam buku ini menggunakan istilah versi bahasa Inggris. Penggunaan istilah yang dipaksa untuk seratus persen berubah ke bahasa Indonesia akan terkesan janggal. Misalnya : register, endpoint, pipe, descriptor, video streaming, dan sebagainya. Penggunaan istilah-istilah dalam tulisan ini mengacu pada nama standard universal.

BAB I
TINJAUAN UNIVERSAL SERIAL BUS SECARA UMUM

Keunggulan Universal Serial Bus

Sebuah peripheral atau device yang terhubung dengan sebuah komputer, idealnya memenuhi beberapa kreteria sebagai berikut :

- Mudah digunakan, tanpa konfigurasi yang memusingkan pengguna (user)
- Mempunyai kecepatan transfer data yang tinggi sehingga tidak ada proses menunggu yang terlalu lama dari pengguna
- Data yang tersaji dapat dipercaya
- Satu jenis interface yang secara fisik kompatibel dengan berbagai peripheral, dan dapat dipakai dimana saja
- Tidak mahal
- Konsumsi daya listrik tidak boros
- Support dengan sistem operasi yang banyak dipakai pengguna (Ms. Windows)

Semua kreteria di atas dapat dipenuhi oleh sistem antarmuka USB (Universal Serial Bus). USB didesain untuk memudahkan user (pengguna) agar mampu mengatasi keruwetan hardware antarmuka generasi sebelumnya (serial dan paralel port).

Universal Serial Bus (USB) secara perlahan-perlahan menggantikan protokol komunikasi RS232 sebagai solusi protokol komunikasi antara PC dengan peripheral. USB menyediakan bandwidth yang lebih lebar untuk keperluan pertukaran data, serta power supply 5 Volt 500 mA yang pada praktiknya arus ini mampu sampai 2 Ampere. Di jaman sekarang sebuah PC sudah dilengkapi dengan port USB minimal 2 buah port. Kabel yang digunakan pun hanya 4 buah yang secara fisik lebih kecil dibandingkan dengan standar DB9 (serial COM) apalagi DB25. Panjang kabel maksimum yang digunakan juga bisa menjangkau area 5 meter dari sisi PC (host). Dengan hub jarak ini bisa mencapai 30 meter.

Semua PC baik yang bersistem operasi Mac OS, Linux, ataupun Windows sudah dilengkapi dengan port USB yang menghungkan dengan peripheral standar seperti mouse, keyboard, kamera, scanner, printer dan beberapa periperal khusus yang menggunakan interface USB. Pada sistem operasi Ms Windows, ketika pemakai mencolokkan periperal pada konektor USB, Windows mendeteksi peripheral tersebut dan mencari driver yang cocok. Pertama kali device USB konek, Windows mungkin meminta driver software, namun setelah itu instalasi akan berjalan secara otomatis. Dengan USB, pemakai juga lebih mudah melakukan koneksi peripheral yang bersangkutan ke PC dibanding dengan jenis peripheral dengan interface non USB seperti PCI, serial ataupun DB25.

Peripheral yang memakai USB juga tidak memerlukan settingan khusus yang rumit dari sisi pemakai, seperti alamat port, sistem interrupt yang dipakai, apalagi harus mensetting secara hardware (setting jumper). Port USB juga memungkinkan lebih dari sebuah peripheral terhubung langsung pada port yang sama secara seri. Hal tersebut dikarenakan adanya USB host controller di tingkat host. Dengan begitu, para programmer embedded (tingkat device) tidak perlu repot-repot memikirkan bagaimana kontrol yang ada di sisi PC (host) jika harus menghandel lebih dari sebuah peripheral, karena cukup menuliskan kode komunikasi pada device yang bersangkutan.

Dibandingkan dengan teknik antarmuka sebelumnya, USB menawarkan berbagai keunggulan dari sisi pemakai yaitu:

- Mudah digunakan. Hanya dengan satu jenis antarmuka (bentuk konektor yang simpel) USB dapat digunakan berbagai peripheral
- Power 5 volt yang tersedia memungkinkan beberapa perangkat USB menggunakan self-

Copyright © 2003-2007 IlmuKomputer.Com

powering sehingga tidak perlu tambahan Power Supply dari luar

- Dari sisi aplikasi tertinggi untuk pengguna pada sistem operasi, tidak memerlukan settingan yang rumit untuk mengatur pertukaran data, bahkan mendukung sistem plug and play

Karateristik Dan Jenis Antarmuka USB

Memang selain USB masih banyak jenis interface lain yang dapat dimanfaatkan untuk keperluan yang sama (termasuk port serial RS232 dan port paralel). Tabel 1.1 menunjukkan perbandingan berbagai jenis antarmuka, dilihat dari kecepatan transfer data, panjang kabel maksimum, serta kegunaan yang biasanya dipakai dari berbagai jenis antarmuka.
USB versi 1.0 dirilis pada Januari 1996, dan setelah diadopsi oleh banyak perusahan, kini protokol komunikasi USB menjadi standar komunikasi hampir semua peripheral komputer. USB menjadi populer dikarenakan yang versi 1.0 saja, bandwidth yang mampu diusung oleh protokol ini mencapai 1,5 Mbits/s, lebih cepat daripada standar RS232, dan 12 Mbit/s (fullspeed). Selain itu sebuah host USB mampu berkomunikasi dengan 127 peripheral yang berbeda. Sejak dipublikasikan versi 1.0 nya spesifikasi USB mengalami 2 kali revisi sehingga menjadi versi 1.1 pada bulan September 1998 dan berkembang ke versi 2.0 pada April tahun 2000.
Pada tahun 2009 sudah dirilis peripheral dengan mengusung versi 3.0 yang juga dikenal dengan nama superspeed, tentu saja dengan kecepatan transfer data yang lebih besar daripada versi 2.0.

Jenis Interface	Format Komunikasi Data	Maksimal Device yang mampu terhubung	Panjang kabel (jarak) (max, meter)	Kecepatan transfer (maximum, bits/sec.)	Contoh aplikasi
USB	Serial Asinkron	127	4.8m (bisa 30 m dengan 5 hub)	1.5M, 12M, 480M	Mouse, keyboard, drive, audio, printer, dan peripheral standard ataupun khusus.
Ethernet	Serial	1024	480	10G	Komunikasi jaringan
IEEE-1394b (FireWire 800)	Serial	64	90	3.2G	Video, media penyimpanan
IEEE-488 (GPIB)	Parallel	15	18	8M	Instrumentasi
IrDA	Serial Asinkron Infra Merah	2	1.8	16M	Printers, *handheld*
I2C	Serial Sinkron	40	5	3.4M	Komunikasi Mikrokontroler
Microwire	Serial sinkron	8	3	2M	Komunikasi Mikrokontroler
MIDI	Serial Current loop	2	15	31.5K	Music, *show control*
Parallel Printer Port	Parallel	2	3-9	8M	Printers, scanners, *disk drives*
RS-232 (EIA/TIA-232)	Serial Asinkron	2	15-30	115K	Modem, mouse, Instrumentasi

Copyright © 2003-2007 IlmuKomputer.Com

Jenis Interface	Format Komunikasi Data	Maksimal Device yang mampu terhubung	Panjang kabel (jarak) (max, meter)	Kecepatan transfer (maximum, bits/sec.)	Contoh aplikasi
RS-485 (TIA/EIA-485)	Serial Asinkron	32	1200	10M	Akuisi data dan sistem kontrol
SPI	Serial sinkron	8	3	2.1M	Komunikasi Mikrokontroler

Tabel 1.1 Perbandingan transfer data USB dengan jenis transfer data yang lain

Versi USB (setidaknya sampai tahun 2009) dibagi menjadi 3 macam, yaitu versi 1.0 , versi 2.0 serta versi 3.0. **USB 1.0** terbagi menjadi **Low Speed** dan **Full Speed**, sedangkan **High Speed** masuk ke dalam kategori **versi 2.0**. Penggunaan USB (yang berkaitan dengan kelas USB) disesuaikan dengan fungsi peripheral, dan tidak serta merta menggunakan jenis yang sama untuk peripheral yang berbeda. Misalnya sebuah mouse/keyboard membutuhkan kabel yang lentur dan fleksibel. USB Low Speed tidak terlalu dipengaruhi oleh kualitas kabel dan juga pembuatan Low Speed lebih murah. Oleh sebab itu USB jenis Low Speed cocok digunakan pada jenis peripheral seperti mouse. Full Speed ditujukan untuk mengganti peripheral yang menggunakan standar RS 232 (serial) dan paralel port. High Speed menjadi pilihan yang bagus untuk periperal yang sangat tergantung dari kecepatan transfer data seperti *mass-storage*, akuisi data, streaming data, dan sebagainya.

Jenis USB	Contoh Aplikasi
LOWspeed • Interactive Devices • 10 – 100 kb/s	Keyboard, Mouse Stylus *Game Peripherals* *Virtual Reality Peripherals*
FULLspeed • Phone, Audio, Compressed Video 500 kb/s – 10 Mb/s	POTS *Broadband* Audio Microphone
HIGHspeed • Video, Storage • 25 – 400 Mb/s	Video *Storage* *Imaging* Broadband

Tabel 1.2 Taksonomi USB pada aplikasi pemakaian.

Bagaimana dengan reabilitas USB? Seperti RS 232, USB dibangun atas hardware dan sekumpulan software di sisi device (biasa dinamakan **firmware**). Hardware yang bersangkutan harus mampu menjamin bahwa tidak ada *noise* yang dapat mengganggu transfer data. Selain itu protokol USB mensyaratkan adanya *error checking* dari sisi software, sehingga satu paket data dapat ditransfer ulang. Proses deteksi, pengecekan hardware, lalu pentransferan kembali dilakukan di sisi hardware dan tidak diperlukan program khusus atau interferensi dari pengguna awam atau end user.

Tidak seperti jenis antarmuka (interface) yang lain, USB tidak mampunyai jalur-jalur transfer data khusus pada kabel, seperti halnya serial port dan paralel port. Untuk berkomunikasi dengan peripheral standar seperti printer, keyboard, USB harus mendefiniskan kelas dari masing-masing hardware tersebut (dilakukan di tingkat software). Kelas tersebut berisi identifikasi dari peripheral yang bersangkutan

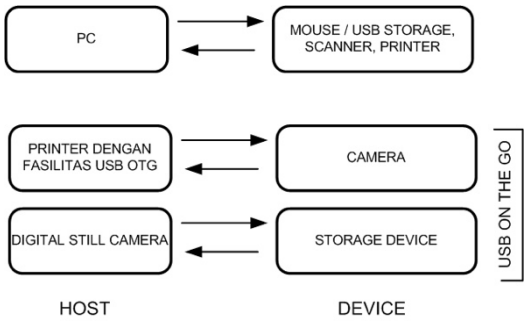
Copyright © 2003-2007 IlmuKomputer.Com

Dari sisi komputer yang berfungsi sebagai host, Windows 98 merupakan sistem operasi pertama buatan Microsoft yang mendukung sistem antarmuka USB, dan sistem operasi setelah Windows 98 semua sistem operasi yang diproduksi oleh Microsoft sudah mendukung antarmuka jenis ini. Disamping itu, Apple computer/Machintosh dan Linux juga telah mendukung interface USB pada tingkat kernel sistem operasi yang bersangkutan, dan sudah menjadi kebutuhan standar. Dukungan USB pada tingkat sistem operasi sebuah komputer menuntut sistem operasi tersebut harus melakukan 3 hal sebagai berikut:

1. Mendeteksi ketika device/ peripheral dimasukkan atau dihubungkan maupun diputus dari sistem.
2. Melakukan komunikasi dengan device yang terpasang dan melakukan pertukaran data dengan device.
3. Menyiapkan sebuah mekanisme rutin software yang mampu mengatur komunikasi antara aplikasi host dengan device yang menggunakan interface USB.

Pada level tertinggi sistem operasi (tingkat aplikasi), kita diberi keleluasaan untuk mengakses device yang kita buat. Jika sistem operasi yang kita gunakan tidak tersedia driver untuk device (peripheral) yang kita buat, maka kita harus membuat driver dari device yang kita buat.
Microsoft sendiri sebagai pembuat sistem operasi terbesar, telah banyak menambahkan kelas driver yang mendukung berbagai macam device. Diantaranya Human Interface Device (device yang menggunakan kelas ini misalnya keyboard, mouse, game controller), audio devices, modem, scanner, printer, card reader, dan kamera. Bahkan kita dapat membuat driver yang memungkinkan kita menentukan fungsi device yang kita rancang melalui windows API (Application Programming Interface).

Dari sisi peripheral (atau kalau dalam sebuah sistem embedded dikenal dengan sebutan device) hardware yang terhubung harus mampu mengatur setiap detil komunikasi USB. Dengan begitu adalah tidak mungkin ketika kita menyamakan sebuah sistem interface USB dengan interface sebelumnya contohnya port paralel. Penggunaan kontroller yang berbasis mikroprosesor sangat mutlak diperlukan untuk membangun jenis interface ini. Beberapa jenis kontroler yang sudah dikemas dalam sebuah chip mikrokontroler sudah banyak yang mendukung antarmuka USB. Fungsi dari mikrokontroler ini adalah merespon setiap data yang akan dikirim dan diterima, mengidentifikasi dan mengkonfigurasi device, serta membaca dan menulis data yang diperlukan ke dalam bus data. Memang ada beberapa seri mikrokontroler yang proses komunikasi USB (enumerasi) sudah tertanam secara hardware. Namun jika belum, mau tidak mau kita harus menulis kode-kode yang diperlukan agar komunikasi USB dapat terbentuk. Kode-kode tersebut biasanya sudah disediakan oleh vendor produsen mikrokontroler yang bersangkutan.



Gambar 1.1 Bentuk hubungan komunikasi Host – Device

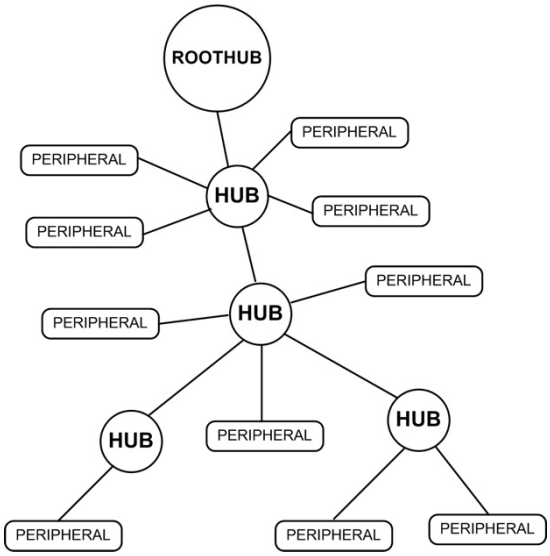
Komunikasi yang terjadi dalam antarmuka USB adalah host ke atau dari device (PC dan peripheral). Komunikasi tersebut dirasakan para developer masih kurang. Oleh karena itu, pada

Copyright © 2003-2007 IlmuKomputer.Com

tahun 2001 diciptakanlah sebuah komunikasi USB antara host dan device yang mempunyai kapasitas terbatas. USB ini umumnya dikenal dengan **USB On-The-Go** (OTG).

Contoh implementasi dari USB OTG adalah printer yang mampu mencetak langsung dari kamera. Kamera dalam hal ini berfungsi sebagai device, printer dalam hal ini berfungsi sebagai host. USB OTG mempunyai beberapa keterbatasan, dibandingkan dengan USB murni, dan didesain atas pertimbangan fungsi dari device saja. Kecepatan transfer dari USB OTG inipun juga sudah setara dengan USB versi 2.0. Selain USB OTG dikenal pula **USB Wireless** yang menggunakan jalur radio sebagai media komunikasi host dengan device.

Secara fisik koneksi dari sebuah Universal Serial Bus terdiri atas sirkuit, konektor, dan kabel yang menghubungkan sebuah host dengan satu atau lebih device. Sebuah host seperti yang telah disinggung di muka, berupa sebuah PC atau sistem komputer yang mengandung sebuah host kontroler dan sebuah root hub. Kedua komponen tadi bekerja bersama-sama untuk mengaktifkan sistem operasi untuk berkomunikasi dengan device dalam sebuah bus. Format data yang diterima oleh host kontroler juga harus dimengerti oleh program tingkat aplikasi.
Root hub pada kenyataannya berupa konektor USB yang bisa terdiri lebih dari satu buah. Root hub ini terhubung oleh beberapa hub dan yang berhubungan langsung dengan pripheral seperti paga Gambar 1.2. Setiap peripheral harus mampu berkomunikasi dengan host tanpa konflik dengan peripheral yang lain.



Gambar 1.2. Topologi sistem antarmuka USB. Hub menjadi pusat koneksi peripheral

Topologi dari susunan koneksi USB dalam sebuah bus menyerupai bintang (star), bertingkat dari satu peripheral ke peripheral yang lain. Jumlah koneksi di dalam sebuah hub sangat bervariasi, bahkan sebuah hub dapat dihubungkan ke dalam hub lagi secara bertingkat. Untuk berkomunikasi dengan device yang bermacam-macam dan jumlahnya lebih dari satu, host tidak mempedulikan berapa jumlah device yang konek. Hanya 1 device saja yang berhubungan dalam satu waktu tertentu saja, dan untuk meningkatkan bandwidth yang tersedia untuk device, sebuah PC bisa mempunyai banyak *host controler*. Beberapa device ada yang mengandung peripheral sekaligus hub (contohnya card reader).
Sebuah hub mempunyai sebuah jalur *upstream* untuk berkomunikasi dengan host, serta satu atau lebih jalur *downstream* untuk komunikasi ke device. Setiap jalur downstream atau koneksi internal inilah yang dikenal dengan port USB.

Copyright © 2003-2007 IlmuKomputer.Com

Tinjauan Kerja Host

Antara host dan device masing-masing mempunyai kewajiban berbeda. Host memegang peranan penting dalam berkomunikasi, dan device juga harus mempuyai kemampuan merespon data yang masuk, agar komunikasi dapat terus berlangsung.

Berikut ini merupakan tugas dari sebuah host kontroler ditingkat PC :

1. Mendeteksi device
   Pada waktu sebuah device dicolokkan ke port USB, hub memberi tahu host bahwa ada device yang terhubung. Di dalam proses enumerasi (pertukaran sata awal, untuk mengidentifikasi kelas device), host menyiapkan sebuah alamat dan meminta informasi dari device yang bersangkutan. Ketika device tersebut dicabut dari port (remove) , peristiwa tersebut harus diketahui oleh host dan memberitahu aplikasi bahwa device tidak tersambung lagi. Proses tersebut terjadi pada saat "power up" dan setiap proses pertukaran data selalu didahului oleh proses ini.

2. Mengatur aliran data
   Host kontroler membagi waktu yang tersedia menjadi segmen-segmen yang dikenal dengan frame atau mikroframe. Setiap frame atau mikroframe tersebut berisi setiap transaksi dengan setiap device. Setiap transfer data yang terjadi memerlukan waktu untuk menyelesaikan transaksi data dan dalam satu frame (mikroframe) tersebut harus cukup tersedia cukup data yang diperlukan.
   Selama terjadi enumerasi, driver dari device meminta bandwidth yang akan mengatur semua transaksi data (timing). Jika bandwidth tidak tersedia, host tidak mengijinkan proses awal komunikasi terjadi lalu driver akan meminta jumlah bandwidth yang lebih kecil, atau menunggu lebar bandwidth sudah cukup tersedia untuk bertransaksi data.

3. *Checking error*
   Bit *Checking error* ditambahkan oleh host ketika proses transfer data berlangsung. Di device, data yang masuk dikalkulasi dan hasilnya dikomparasi dengan nilai bit checking error. Jika hasilnya tidak sesuai, device tidak melanjutkan penerimaan data, dan mengirimkan sinyal ke host supaya host mentransfer ulang data yang tadi dikirimkan. Proses tersebut terjadi bergantian. Artinya sinyal respon yang dikirim oleh device terjadi setelah sinyal dari host selesai dikirim dalam satu frame/mikroframe. Bisa jadi pula host menerima indikasi bahwa device tidak dapat mengirim atau menerima data. Jika hal itu terjadi, host menginformasikan driver problem yang ada, dan driver dapat memberitahukan apikasi, sehingga dapat terdeteksi masalah transfer pada level aplikasi yang berjalan pada sebuah sistem operasi.

4. Menyedikan power
   Jika device belum ada power supply, USB mampu menyediakan power sebesar +5 volt.

5. Proses pertukaran data dengan peripheral
   Pertukaran data yang umum terjadi antara host dan device terjadi tanpa ada interval waktu yang tetap. Artinya host mengirim data jika ada permintaan dari device maupun sebaliknya Namun untuk beberapa kasus, driver dari device meminta host untuk mengirim atau menerima data pada interval waktu yang telah ditentukan.

Copyright © 2003-2007 IlmuKomputer.Com

Tinjauan Kerja Device / Peripheral

Kebanyakan cara kerja dari peripheral merupakan kebalikan dari kerja host. Ketika host menginisialisasi pertama kali sebuah komunikasi, peripheral harus mampu merespon. Namun demikian peripheral juga mempunyai tugas khusus. Peripheral memang tidak bisa memulai komunikasi USB sendiri, melainkan harus menunggu dan merespon komunikasi dari host. (kecuali ada fasilitas remote wake up di sisi peripheral).
Setiap device memonitor alamat device yang dipakai di setiap komunikasi di dalam bus. Jika alamat tidak sesuai dengan alamat yang yang ada di device, maka device akan membatalkan komunikasi. Jika alamat pada device sesuai, maka selanjutnya device menyimpan data yang diterima di memori buffer dan membangkitkan sebuah sinyal interupsi yang mengindikasikan bahwa data telah diterima melalui USB port.

Kebanyakan pada chip yang mendukung USB, kode yang kita buat tidak perlu menginisialisasikan memori USB yang sudah ada, karena sudah tertanam secara hardware sejak dari pabrik. Firmware yang ada di dalam chip tidak perlu bereaksi atau merespon sampai chip mendeteksi komunikasi yang mengandung informasi alamat device. Selama enumerasi (pertukaran data awal) berlangsung device harus merespon permintaan standar host. Permintaan standar juga dikirim oleh host sewaktu-waktu setelah proses enumerasi selesai.

Dalam menerima sebuah permintaan data, device hanya merespon saja permintaan dari host. Ketika host meminta sebuh konfigurasi, dan device tidak support, maka device akan mengirimkan kode yang mengatakan bahwa konfigurasi data tersebut tidak support.
Checking error pada sisi device dilakukan dengan mengkalkulasi data yang diterima dan dicocokkan dengan *error-checking bit*. Fungsi error detect pada sisi device kebanyakan sudah ada di sisi hardware dan tidak perlu untuk diprogram ulang. Jika diperlukan, device juga mendeteksi sinyal dari host bahwa data telah diterima oleh host. Kontroler device bisa jadi support satu atau lebih kecepatan, dan semua hub harus mendukung low dan Full Speed, kecuali jika hub yang bersangkutan hanya mempunyai sebuah satu kecepatan transfer data saja. Pengguna tidak perlu mengetahui apakah device yang digunakan fullspeed atau Low Speed. Host yang sudah mendukung kecepatan Full Speed (ver 2.x) harus mendukung kecepatan Low Speed (versi 1.x).

Kecepatan data transfer yang sebenarnya antara peripheral dan host kurang dari kecepatan bus pada chip dan tidak bisa ditentukan dengan pasti. Hal itu karena beberapa bit yang dikirim digunakan untuk mengidentifikasi, mensinkronkan, dan *checking error*, serta rata-rata kecepatan data juga tergantung dari jenis transfer dan seberapa sibuk bus data pada saat itu.
Untuk beberapa data yang sensitif terhadap pewaktuan, USB mendukung jenis transfer yang menjamin jumlah data terkirim dengan benar. Pada transfer asinkron host dapat meminta jumlah byte yang dikirim dari atau ke peripheral pada interval tertentu.

Copyright © 2003-2007 IlmuKomputer.Com

BAB II
KARATERISTIK FISIK DAN PENSINYALAN USB

Lambang Dan Ikon USB

Sesuai dengan **USB-IF (**USB Implemented Forum yaitu forum kumpulan para vendor pengembang perangkat USB) ada beberapa logo USB yang masing-masing tidak bisa saling ditukar keberadaannya pada device. Gambar 2.1a. merupakan lambang dasar USB versi 1.x dan digunakan jika device menggunakan versi Lowspeed (Kecepatan Rendah). Apabila device yang digunakan adalah versi High Speed (Kecepatan Tinggi) atau Full Speed (Kecepatan Penuh) logo yang digunakan seperti pada Gambar 2.1b. Ada kalanya device yang menggunakan Highspeed atau Fullspeed dan mendukung pula versi sebelumnya, sehinga dalam device yang bersangkutan menggunakan kedua simbol. Disamping kedua logo tersebut masih ada pula logo untuk USB dengan komunikasi wireless dan OTG versi 1.0 dan yang terakhir USB versi 3.0 atau Superspeed. Buku ini membahas versi 1.x dan versi 2.0, dengan beberapa protokol baik fisik, kelistrikan, maupun transfer data.

Gambar 2.1.a. Logo USB versi Basic-speed

Gambar 2.1.b. Logo USB versi Hi-speed

Gambar 2.1.c. Logo USB versi On-The-Go Basic-speed

Gambar 2.1.d. Logo USB versi On-The-Go Hi-speed

Gambar 2.1.e. Logo USB versi Wireless USB

Gambar 2.1.f. Logo USB versi 3.0 (Superspeed)

Tinjauan Fisik

Secara fisik spesifikasi koneksi USB meliputi kabel dan konektor untuk menghubungkan device ke host. Spesifikasi keduanya meliputi dimensi, material (bahan), dan tingkat kehandalan. Untuk beberapa device yang digunakan oleh orang awam, barangkali spesifikasi kabel tidak terlalu

Copyright © 2003-2007 IlmuKomputer.Com

berpengaruh, apalagi belum versi superspeed. Namun, untuk keperluan koneksi di lingkungan yang ekstrim, di mana banyak pengaruh harmonik jaringan listrik, medan magnet, atau jarak ke host yang lebih dari 2 meter, jenis kabel sangat berperan penting dalam kelangsungan koneksi. Gambar 2.3 menunjukkan jenis konektor dan jenis kabel secara standar yang disyaratkan oleh USB-IF.

KONEKTOR USB

Topologi USB (hirarki hubungan device ke device maupun device ke host) secara fisik menyangkut hubungan port hub yang berfungsi sebagai media downstream ataupun upstream dari dan ke hub atau device. USB dengan Highspeed (480Mb/s) atau Fullspeed (12Mb/s) membutuhkan kabel jenis shielded dengan dua jalur konduktor power dan twisted pair untuk konduktor sinyal (D+ dan D-). Lowspeed (1,5 Mb/s) tidak menuntut kabel dengan jenis twisted pair. Selain itu, konektor yang terhubung harus didesain hotpug (mudah dilepas dan ditancapkan). Icon USB yang mengindikasikan jenis USB yang dipakai harus tertera pada konektor plug (jantan).

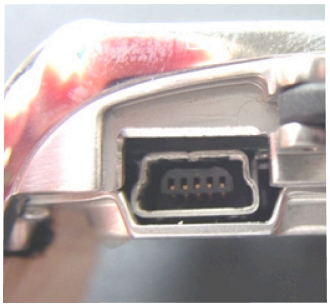
Untuk memudahkan pemakaian dan meminimalisir kesalahan pemakai akibat kesalahan bentuk fisik dari interface USB, protokol USB mempunyai bentuk baku yang sudah distandarkan. Protokol merupakan aturan-aturan dasar yang harus ditaati pengembang device. Bentuk standar dari konektor USB ada 2 macam yaitu tipe A dan tipe B. Konektor tipe A digunakan untuk menghubungkan secara langsung device ke host atau port downstream dari hub. Semua device USB harus mempunyai standar konektor seri A.

Konektor seri B mengijinkan vendor pembuat device untuk menyediakan kabel yang dapat dilepas-pasang oleh pengguna device. Gambar 2.2 berikut menunjukkan contoh jenis konektor USB.

Gambar 2.2.a. Seri A plug

Gambar 2.2.b. Seri B plug

Gambar 2.2.c. Seri A plug mini



Gambar 2.2.d. Seri A receptacle mini

Copyright © 2003-2007 IlmuKomputer.Com

Gambar 2.2.e. Seri A receptacle

Gambar 2.2.f. Seri B receptacle

Dari kedua tipe konektor tersebut, dibedakan lagi menjadi 2 jenis yaitu tipe receptacle (betina) dan tipe plug (jantan). Sesuai dengan standar pemakaian kabel USB pada USB IF:

- Tipe "A" receptacle harus berpasangan dengan tipe "A" plug. Tipe "A" receptacle berfungsi sebagai output host /hub
- Tipe "A" plug harus berpasangan dengan tipe "A" receptacle. Tipe "A" plug selalu berfungsi untuk mengirimkan data ke host.
- Tipe "B" receptacle berpasangan dengan dengan "B" plug (male). Tipe "B" receptacle berfungsi sebagai input ke hub/ device.
- Tipe "B" plug berpasangan dengan tipe "B" receptacle. Tipe "B" plug selalu berorientasi menuju ke hub USB atau device

Selain itu, masih ada tipe konektor mini USB yang secara ukuran lebih kecil dari pada tipe standar. Konektor ini biasanya digunakan untuk device yang secara ukuran fisik mengharuskan portabel.

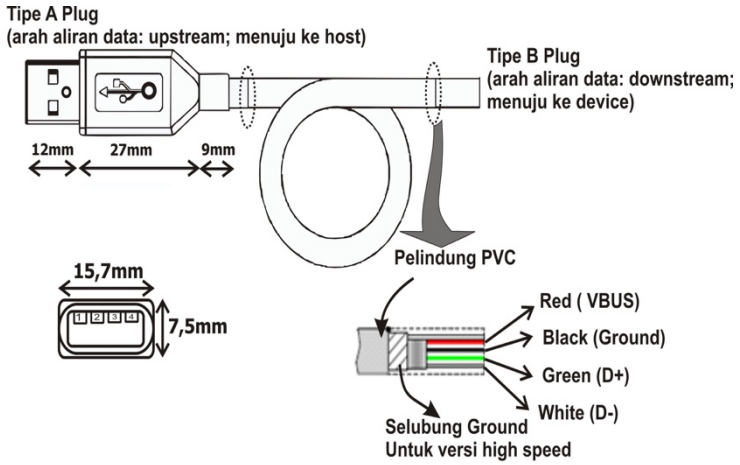
SPESIFIKASI KABEL

Kabel USB terdiri atas 4 konduktor, 2 dipakai untuk keperluan power 5 volt dan 2 kabel lagi untuk keperluan signal data. High Speed ataupun Full Speed menggunakan jenis kabel twisted pair. Jalur VBUS berguna untuk menyalurkan tegangan +5 Volt, dan Ground membungkus secara keseluruhan penampang kabel. Jenis kabel ini juga kompatibel dengan Low Speed. Pada kecepatan rendah/Low Speed, jenis kabel tidak terlalu berpengaruh karena tingkat gangguan datanya rendah, walaupun memang akan lebih baik jika menggunakan kabel yang kompatibel dengan Full Speed.

Kabel USB untuk menghubungkan perangkat device dengan host harus memiliki kriteria sebagai berikut:

- Ujung kabel harus terhubung dengan tipe A plug dan ujung lainnya terhubung dengan perangkat. Jika di dalam perangkat tersebut masih diperlukan konektor, maka di ujung kabel yang terhubung dengan vendor dipasang konektor tipe B plug. Sehingga di setiap ujung kabel mempunyai konektor A dan B tipe plug.
- Kabel yang dipakai harus mendukung standar kecepatan highspeed dan fullspeed ataupun low–speed.
- Impedansi kabel harus sesuai dengan impedansi yang disyaratkan oleh USBIF.

Copyright © 2003-2007 IlmuKomputer.Com



Gambar 2.3. Ukuran Konektor (Tipe A) dan kabel USB

Tabel 2.1 menunjukkan keterangan masing–masing pin konektor USB, baik tipe A maupun tipe B.

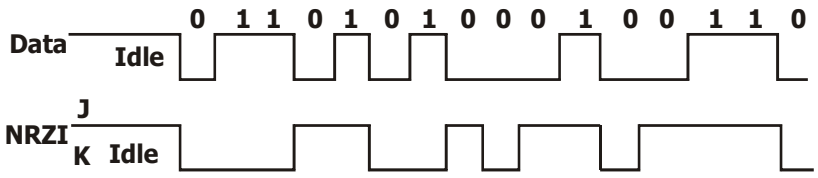
Nomor pin	Nama	Warna kabel
1	V BUS	Merah
2	D-	Putih
3	D+	Hijau
4	GND	Hitam

Tabel 2.1 Deskripsi pin kaki USB

TINJAUAN KELISTRIKAN

1. ENCODING/DECODING DATA

Sesuai dengan protokol USB, transfer data USB menggunakan sebuah encoding data NonReturn to Zero Inverted (NRZI). NRZI sendiri merupakan suatu sistem pengkodean dimana suatu transisi (low ke high atau high ke low) pada awal bit time akan dikenal sebagai biner '0' untuk bit time tersebut. Jika biner 1 maka tidak ada transisi. Sehingga di dalam NRZI akan lebih tepat jika menggunakan kata differensial 1 atau differensial 0 karena memang pembacaan logika berasal dari perubahan transisi, dan bukan semata-mata karena nilai data bit awal.



Gambar 2.4 Pengkodean sistem NRZI

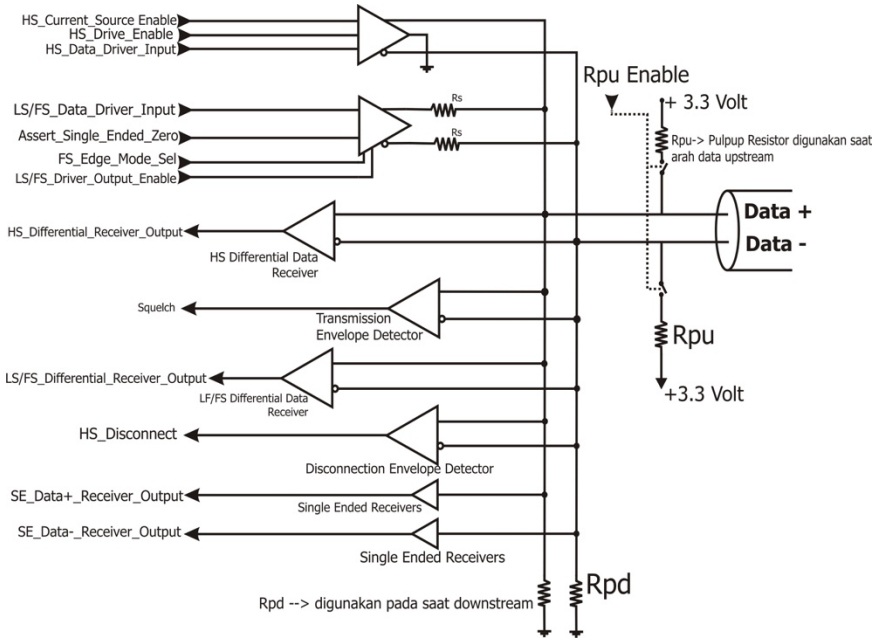
Bagaimana nilai differensial 1 dan differensial 0 dihasilkan oleh konektor D+ dan D- ? Pada kecepatan Low dan Full Speed, differensial 1 ditransmisikan dengan menarik D+ sampai 2,8 volt melalui resistor 15 Ohm ke ground serta D- di bawah 0,3 volt dengan resistor 1,5 K Ohm yang terhubung ke 3,6 volt. Dengan kata lain differensial 0 diindikasikan dengan nilai

Copyright © 2003-2007 IlmuKomputer.Com

D- yang lebih besar dari 2,8 volt dan D+ kurang dari 0,3 volt.
Dari sisi receiver (penerima) differensial 1 sama dengan nilai D+ 200 mV lebih besar daripada D-. Differensial 0 mempunyai nilai D+ 200mV lebih kecil daripada D-. Nilai 200mV merupakan nilai minimal yang mampu diterima. Polaritas dari D+ dan D- akan membalik (inverse) tergantung dari kecepatan bus. Kondisi differensial 0 akan lebih dikenal dengan konsisi 'K' dan differensial 1 dengan 'J'.
Tranceiver (tranmitter dan receiver) USB sendiri akan menerima sinyal J atau K serta sebuah sinyal yang mengindikasikan proses telah selesai yang dikenal dengan Single Ended Zero (SE0). Sebuah sinyal SE0 ditengarai dengan nilai level tegangan D- dan D+ lebih kecil dari 0,3 volt.

2. LEVEL SINYAL UNTUK LOW / FULL SPEED

Gambar 2.5 di bawah ini merupakan bagan tranceiver USB versi 1.1 yang sudah dilengkapi komponen untuk mendukung operasi High Speed. Dinamakan tranceiver karena memang mempunyai untuk trasmit (mengirim data) dan receive (menerima data).
Tabel 2.2 berikut ini menunjukkan level pensinyalan USB. Kolom pertama menunjukkan keadaan bus, kolom kedua menujukkan keadaan ujung/sumber pada konektor, dan kolom ketiga menunjukkan keadaan yang diterima target.



Gambar 2.5 Tranceiver Sirkuit USB versi 1.1

Keadaan bus	Level Sinyal		
	Ujung Konektor /sumber (dihitung dari bit terakhir)	Target Konektor	
		Standar yang diperlukan	Keadaan yang diterima
Differensial 1	D+ > V_{OH} min D- < V_{OL} mak	(D+) - (D-) > 200mV D+ > V_{IH} min	(D+) – (D-) > 200mV
Differensial 0	D- > V_{OH} min D+ < V_{OL} mak	(D-) - (D+) > 200mV D- > V_{IH} min	(D-) – (D+) > 200mV
Single Ended 0 (SE0)	D+ danD- < V_{OL}(mak)	D+ dan D- < V_{IL}(mak)	D+danD- < V_{IH}(min)
Single ended 1 (SE1)	D+danD- >V_{OSE1}(min)	D+ dan D- > V_{IL}(mak)	

Copyright © 2003-2007 IlmuKomputer.Com

Keadaan bus	Level Sinyal		
Keadaan "J" Low Speed Full Speed	*Differensial* "0" *Differensial* "1"	*Differensial* "0" *Differensial* "1"	
Keadaan "K" Low Speed Full Speed	*Differensial* "1" *Differensial* "0"	*Differensial* "1" *Differensial* "0"	
Keadaan "Idle" Low Speed Full Speed	-	D-> VIHZ(min) dan D+ < VIL (mak) D+ > VIHZ(min) dan D- < VIL (mak)	D- > VIHZ(min) dan D+ < VIL (mak) D+ > VIHZ(min) dan D- < VIL (mak)
Keadaan Resume (mengulang)	Data 'K'	Data "K"	
Start of packet (SOP)	Perubahan data dari "Idle" ke "K"		
End of Packet (EOP)	SE0 selama 2t bit yang diikuti oleh J 1t bit	SE0 ≥ 1 t bit diikuti oleh j selama 1 t bit	SE0 untuk waktu ≥ 1 bit time diikuti oleh "J"
Diskonek (Port Downstream)	-	SE0 ≥ 2,25 us	
Konek (Port Downstream)	-	Idle ≥ 2ms	Idle ≥ 2,5 us
Reset	D+ dan D- < VOL (mak) selama ≥ 10ms	D+ dan D- <VIL (mak) selama ≥ 10ms	D+ dan D- <VIL (mak) selama ≥ 2,5us

Tabel 2.2 Level Pensinyalan USB Low/ Full Speed

Nilai J dan K adalah dua logika keadaan untuk menggambarkan perbedaan data di dalam sistem. Nilai differensial (perbedaan) tersebut diukur dari poin dimana keadaan sinyal saling berseberangan. Pada bagaian device, diagram komunikasi USB sebagai penerima data maupun pengirim data terdiri atas beberapa bagian seperti Gambar 2.5. Semua piranti (device) mempunyai sebuah koneksi untuk upstream ke host dan host sendiri mempunyai sebuah koneksi downstream ke device. Berdasarkan Gambar 2.5 beberapa elemen tranceiver USB dapat dijabarkan sebagai berikut:

- *Low/ Full Speed Driver*
  Low/ Full Speed driver digunakan sebagai buffer/penyangga setiap logika data yang masuk.
- *Low/ Full Speed Differensial Receiver*
  Penerima data baik Low Speed maupun Full Speed yang sudah berupa differensial data (J atau K).
- *Single Ended Receiver*
  Digunakan untuk Low Speed maupun High Speed yang mendeteksi kode *single ended*.
- *High Speed Current Driver*

  Hanya digunakan untuk High Speed. Nominal arus masukan untuk data pada D+ atau D- adalah 17,78 mA. Jika melalui impedansi kabel 45 Ohm, nilai VHSOH kurang lebih 400mV, sehingga nilai differensial antara J dan K (antara (D- dan D+) adalah 400mV. Pada Gambar 2.5 ada *HS_Current_Source_Enable* berfungsi untuk meng*enable* arus On lalu *HS_Drive_Enable* untuk menghubungkan arus dengan sumber sehingga seolah-olah terhubung langsung ke jalur data. Sinyal Diskonek diindikasikan ketika amplitudo differensial sinyal lebih besar atau sama dengan 625 mVolt. Level ini tidak akan terdeteksi sebagai "diskonek" jika ampiltudo sinyal hanya mencapai 525m Volt.
- *High Speed Differensial Data Receiver*

  Berfungsi untuk menerima data, dan memilah data yang masuk jika menggunakan High Speed.
- *Transmission Envelope Detector*

  Berfungsi untuk mengindikasikan bahwa data tidak valid, ketika data yang masuk di

Copyright © 2003-2007 IlmuKomputer.Com

bawah treshosd (batas bawah) / VHSSQ.

- *Disconnection Envelope Detector*

  Dibutuan untuk arah aliran downstream ke port untuk mendeteksi *High Speed disconnec*t di line (VHSDSC). "Diskonek" diindikasikan dengan adanya amplitudo dari differensial sinyal pada downstream yang lebih besar atau sama dengan 625 mV.

- *Pullup resistor (Rpu)*

  Berfungsi untuk indikator speed transaksi data upstream. Ketika transaksi data menggunakan kecepatan High Speed/fullspeed, nilai 1,5 KOhm harus dihilangkan. Bagian yang mengatur keberadaan resistor ini adalah RPU *Enable*.

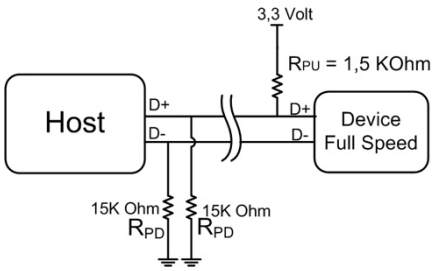
- *Pulldown resistor (Rpd)*

  Digunakan untuk arah downstream dan harus disesuaikan dengan spesifikasi hih speed atau Low Speed.

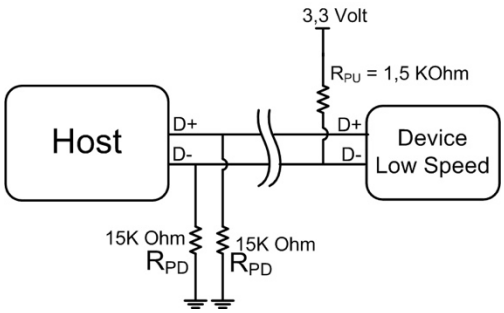
Seperti telah disinggung di muka, USB menggunakan differensial output untuk mendrive sinyal data ke kabel. Untuk kecepatan rendah dan tinggi output statis maksimal adalah 0,3 volt dengan 1,5 KOhm pull-up ke 3,6 volt. Untuk High Speed harus di atas VOH 2,8 volt dengan resistor 15 KOhm pull-down. Pergerakan sinyal antara diferensial tinggi dan rendah harus seimbang untuk meminimalkan efek skew. Output dari driver harus mendukung tiga jenis keadaan operasi untuk membuat sebuah komunikasi half duplex dua arah.

3. Identifikasi Kecepatan USB

Selain untuk mengirim data berupa nilai J dan K, konektor D+ dan D- juga mempunyai fungsi untuk mengirimkan indikator kecepatan yang sedang dipakai, apakah High Speed atau Low Speed. Seperti terlihat pada Gambar 2.6 dan 2.7, keberadaan High Speed dilakukan dengan menaruh resistor pull-up ke D+. Resistor pull-up ini juga digunakan host/hub untuk mendeteksi keberadaan device untuk pertama kali tersambung ke port. Tanpa adanya resistor pull-up tadi, port USB akan diasumsikan kosong, tidak ada yang terhubung ke bus. Resistor pull-up sudah terintergerasi dalam sebuah keping chip. Kita sebagai pemakai tinggal mengatur di bagian firmwarenya saja.



Gambar 2.6 Identifikasi Full Speed Device dengan resistor pull up yang terhubung ke D+



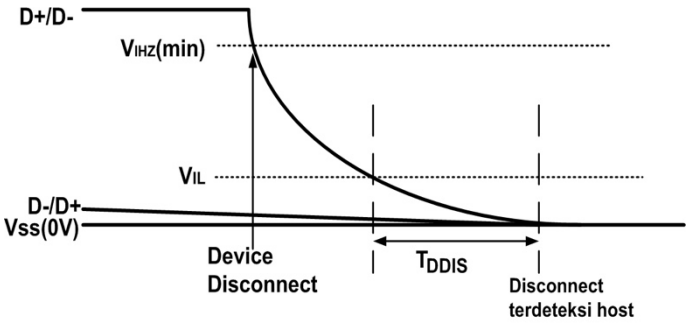
Copyright © 2003-2007 IlmuKomputer.Com

Gambar 2.7. Identifikasi Low Speed Device dengan resistor pull up yang terhubung ke D-

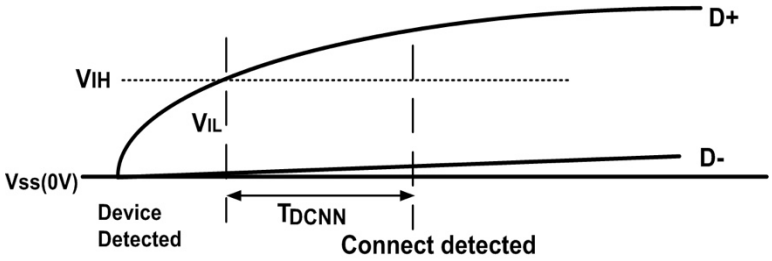
Jika kondisi yang terjadi bukanlah Full Speed, melainkan High Speed, device akan berlaku seperti pada Full Speed (1,5KOhm ke 3,3 volt). Setelah device terhubung dengan host, komunikasi yang terjadi akan menggunakan High Speed jika hub juga mampu mendukung High Speed. Jika operasi device berada pada mode High Speed, resistor pull-up tadi dihilangkan untuk menyeimbangkan jaringan.
Device (piranti) USB versi 2.0 tidak harus mendukung mode High Speed. Sedangkan hub/host harus mendukung arah aliran downstream dengan 3 mode kecepatan High Speed, Full Speed serta Low Speed.

4. Sinyal Konek , Diskonek, Dan Delay

Jika tidak ada sesuatu yang tersambung ke arah downstream dari host/ hub, resistor pull down akan mengakibatkan sebuah kondisi SE0 (Single Ended Zero). Selama kondisi ini melebihi waktu T_{DDIS} (waktu untuk mendeteksi sebuah device diskonek) maka host akan mendeteksi keadaan diskonek.
Kondisi konek terjadi ketika hub mendeteksi salah satu D+ atau D- di atas batas V_{IH} (Tegangan yang merepresentasikan differensial 1) selama lebih dari TDCNN (waktu mendeteksi sebuah device terkoneksi) . Setelah device terdeteksi, hub baru mendeteksi kecepatan yang digunakan oleh device sebelum mengirim sinyal SE0 ke device. Proses konek dan diskonek dilustrasikan oleh Gambar 2.8 dan Gambar 2.9.



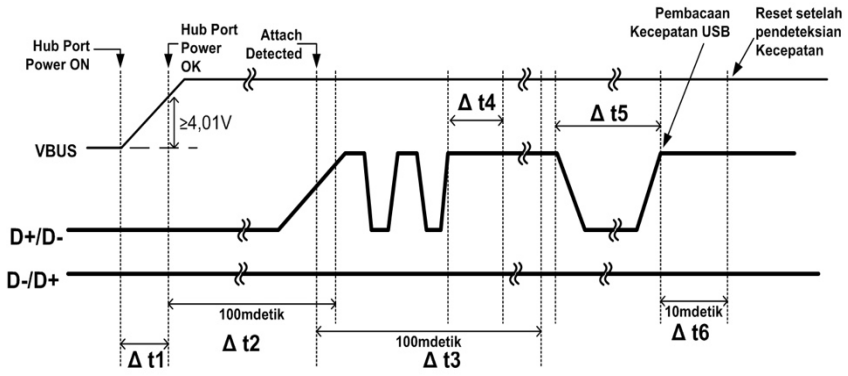
Gambar 2.8. Deteksi proses Diskonek Low/Full Speed



Gambar 2.9. Deteksi proses Konek Full/High Speed

Device juga harus mampu mendukung *plug* dan *remove*, serta proses *switching* yang dilakukan oleh hub. Selama proses dicolokkan pertama kali sampai terdeteksi oleh host, ada 6 delay yang terjadi selama proses tersebut, seperti ditunjukkan oleh Gambar 2.10.

Copyright © 2003-2007 IlmuKomputer.Com



Gambar 2.10. Grafik urutan kejadian per waktu saat device disambungkan (attach) sampai dimulainya proses enumerasi

Δt1 : Total waktu yang diperlukan port hub untuk menswitch power. Jika device sudah tercolok dalam port hub USB dan tidak perlu ada switch lagi maka nilai dari Δt1 ini adalah 0.

Δt2 :(T_{SIGATT}), yaitu waktu maksimum sejak dari VBUS (4,01V) sampai device mendeteksi sinyal host. Δt2 merupakan waktu yang diperlukan device untuk stabil, dan untuk D+ maupun D- merupakan kesempatan untuk mencapai V_{IH} (min). Δt2 harus kurang dari 100ms untuk semua implementasi hub dan device.

Δt3 :(T_{ATTDB}) nilai waktu yang menunjukkan interval dengan durasi minimum 100ms yang diatur dalam sofware. Hal ini dilakukan ketika device telah terdeteksi. Interval ini diperlukan untuk memastikan koneksi ke device sudah stabil. Jika device diskonek maka interval ini akan restart.

Δt4 : (T_{2SUSP}) Terjadi jika tidak ada aktivitas transaksi data di dalam bus, maka device akan memasuki kondisi "suspend"

Δt5:(T_{DRST}) . Perode hub untuk mereset device.

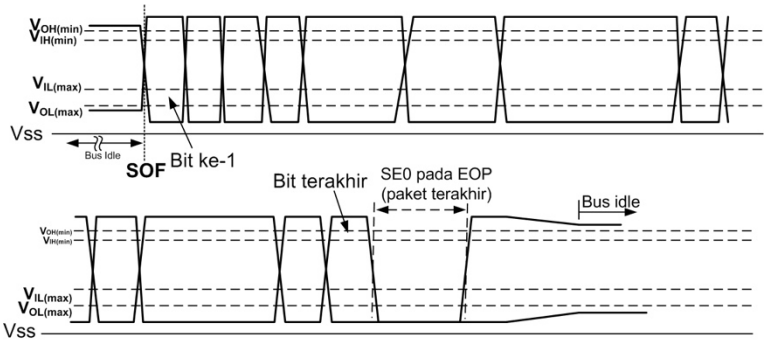
Δt6:(T_{RSTRCY}) Waktu minimum 10ms untuk mereset kembali device.

5. SINYAL DATA UNTUK LOW/ FULL SPEED

Transmisi data USB dilakukan dalam sebuah paket-paket yang berisi nilai differensial data. *Start of Packet* (SOP) diindikasikan dengan mengubah nilai D+ dan D- dari kondisi idle ke level logika K. Proses ini dikenal dengan proses SYNC (synchronisasi). Hub harus membatasi lebar perubahan bit pertama dari SOP kurang dari 5ns. SE0 (*Singe Ended Zero*) digunakan untuk menunjukkan proses EOP (*End Of Packet*). EOP sendiri dilakukan dengan membuat D+ dan D- ke SE0 untuk 2 kali bit waktu lamanya diikuti dengan J untuk satu satuan waktu 1 bit. Setelah itu kedua konektor D+ dan D- disetel pada kondisi impedansi tinggi (lihat Gambar 2.11 yang menunjukkan proses dari SOP dan EOP), sehingga kondisi device akan menjadi idle.

Kondisi idle untuk kecepatan tinggi/High Speed diketahui jika kedua jalur terhubung ke ground. SOP dibuat dengan mengubah dari kondisi idle ke K. Nilai K inilah yang yang menjadi bit pertama dari SYNC. EOP dimulai dari transisi simbol terakhir sebelum EOP menuju ke kebalikan kode awal. Setelah proses EOP, driver akan mengeset D+ dan D- kembali ke kondisi High Speed idle.

Copyright © 2003-2007 IlmuKomputer.Com



Gambar 2.11. Proses dari SOP dan EOP Low/ Full Speed

1. SINYAL RESET
   Sinyal reset ke arah downstream dilakukan dengan memberi sinyal SE0 (Kedua jalur D+ dan D- logika 0). Setelah sinyal SE0 diterima, device akan berada pada kondisi default. Sinyal reset dibangkitan dari software, dan diberikan selama minimum 10ms (T_{DRST}).
2. SINYAL "REMOVE" DEVICE (SUSPEND)
   Apapun jenis device yang digunakan, device tersebut harus mendukung sebuah proses pemisahan diri dari host. Kondisi ini lazim disebut dengan suspend. Jika kondisi suspend sudah terpenuhi, device USB baru bisa dipisahkan dari host secara aman. Proses menuju ke suspend mode ini juga tidak mempedulikan dari posisi mana kondisi sinyal yang tengah berjalan. Device harus mampu mengkondisikan ke mode suspend jika menemui sinyal idle di jalur data lebih dari 3 milidetik. Device harus benar-benar berhenti, dan akan memulai transaksi data lagi setelah 10ms setelah tidak ada aktivitas apapun di dalam bus. Pada saat kondisi suspend, host tetap memberi power D+ (Full/High Speed) atau D0 (lowspeed), agar hub dapat selalu memantau kondisi idle dari device.

Copyright © 2003-2007 IlmuKomputer.Com

BAB III
TRANSFER DATA USB

Jenis Transfer Data

Komunikasi data pada USB dikategorikan menjadi 2 macam yaitu komunikasi untuk proses **enumerasi** (pengenalan device) dan **komunikasi aplikasi** yang berjalan sesuai dengan jenis device yang terhubung. Selama enumerasi, firmware merespon setiap permintaan data yang dikirimkan host. Host akan menganalisa dan mempelajari jenis device yang terhubung. Jika sistem operasi yang gunakan adalah Microsoft Windows, pada saat enumerasi, Windows harus menempatkan file *.INF yang merupakan driver dari device yang bersangkutan. Setelah host selesai bertukar informasi (proses enumerasi selesai), komunikasi aplikasi dapat dimulai. Aplikasi dari sisi host dapat memanfaatkan standard Windows API untuk membaca dan menulis ke device. Dari sisi device, pengiriman data membutuhkan rutin penempatan byte data di buffer *USB controller transmit*. Sedangkan penerimaan data berasal dari buffer *USB controller receive*. Semua jenis transfer data yang terjadi menggunakan salah satu jenis transfer : **kontrol**, **interrupt**, **bulk**, atau **asinkron**. Masing-masing mempunyai format transfer dan kegunaan yang berbeda.

Host mengatur trafik data dengan membagi waktu transaksi data menjadi pecahan-pecahan yang dikenal dengan **frame** (low/ Full Speed) atau **mikroframe** (High Speed). Satu frame mempunyai durasi waktu 1 milidetik. Untuk High Speed, host membagi setiap frame menjadi delapan bagian yang masing masing 125 mikrodetik atau singkatnya 8 mikroframe. Host melakukan sebuah proses transfer pada setiap frame/mikroframe. Setiap frame/mikroframe dimulai dengan proses Start of Frame. Setiap transfer data mengandung satu atau lebih proses transaksi data. Setiap transaksi dimulai ketika host mengirimkan sebuah blok informasi yang mengandung informasi alamat device yang dituju (device ke-n) dengan cara memanggil endpoint device yang bersangkutan.

Kecepatan Host Dan Kecepatan Bus

Ketika kita berbicara tentang versi USB, secara otomatis kita membicarakan tentang versi kontroler hub yang pada akhirnya akan berujung pada kecepatan yang mampu diusung oleh hub tersebut. Berdasarkan hukum kompabilitas, maka USB 2.0 harus mensuport USB versi 1.1. Hal itu berarti bahwa pada sebuah USB High Speed, host kontrolernya harus mendukung kecepatan High Speed dan Low Speed (versi 1.x) juga. Sedangkan hub versi 1.x jelas tidak bisa menkonvert kecepatan yang berbeda. Hub versi 2.0 seolah-olah berfungsi seperti prosesor yang mampu me*remote* kompabilitas antara Low Speed dan High Speed.

Endpoint

Semua data yang ada di bus mengalir menuju dari dan ke endpoint. Endpoint merupakan buffer yang menyimpan byte data. Dalam kenyataannya, endpoint ini merupakan sebuah blok memori data atau register di dalam chip kontroler. Register ini hanya terdapat pada jenis chip kontroler yang sudah mendukung antarmuka USB. Data yang disimpan di dalam endpoint sendiri bisa berupa data yang diterima device ataupun bisa jadi data yang akan dikirim ke host. Host juga mempunyai buffer untuk menampung data yang diterima maupun data yang akan dikirim ke device, namun bukan merupakan endpoint. Sekali lagi, endpoint hanya dipunyai oleh device. Endpoint mempunyai alamat yang unik pada device yang mampu mengeluarkan sumber data ataupun menerima data. Sebuah endpoint hanya membawa data pada satu arah saja (kecuali jenis endpoint kontrol yang mampu dua arah). Alamat endpoint mengandung sebuah nomor endpoint yang menyatakan jenis arah data yang menyertainya. Nomor endpoint berkisar antara 0

Copyright © 2003-2007 IlmuKomputer.Com

sampai 15.

Arah endpoint diperspektifkan dari sisi host. **Endpoint IN** berarti bertugas menyiapkan data yang akan dikirim ke host (arah data: device ke host). **Endpoint OUT** berfungsi untuk menyimpan dan menerima data yang diterima dari host (arah data: dari host ke device). Endpoint yang dikonfigurasi sebagai **endpoint kontrol** harus mampu memenuhi keduanya yaitu antara IN dan OUT. Endpoint kontrol sebenarnya terdiri dari sepasang endpoint OUT dan IN yang masing-masing menshare nomor endpoint yang sama. Setiap device harus mempunyai endpoint 0 (zero endpoint) yang dikonfigurasi sebagai endpoint kontrol. Sebuah endpoint juga harus bisa berfungsi sebagai endpoint OUT maupun IN. Misalnya sebuah device mungkin mempunyai sebuah alamat endpoint 1 IN untuk mengirim data ke host dan endpoint 1 OUT untuk menerima data dari host. Sebagai tambahan endpoint 0, sebuah device Full Speed atau High Speed bisa mempunyai lebih dari 30 alamat endpoint. (1 sampai 15 yang masing-masing mendukung IN dan OUT). Sedangkan Low Speed hanya mempunyai 2 endpoint tambahan.

Transaksi data di dalam bus dimulai dengan paket yang mengandung sebuah nomor endpoint dan sebuah kode yang mengindikasikan arah aliran data serta indikator perlu tidaknya transaksi data menginisialisasi transfer kontrol. Kode yang yang dimaksud adalah IN, OUT atau Setup.
Transaksi Setup seperti OUT karena data mengalir dari host ke device, tetapi transaksi Setup selalu menggunakan jenis transfer kontrol, dengan endpoint kontrol sebagai buffernya. Jenis endpoint kontrol merupakan jenis transaksi kontrol yang mengharuskan device selalu harus menerima serta harus diidentifikasi lalu respon permintaan data yang terkandung dalam transaki data Setup. Ketika device menerima sebuah paket OUT atau Setup yang mengandung alamat device, hardware sendiri mengeluarkan sebuah interrupt (interupsi). Sebuah *Interrupt Service Rutine* lalu memproses data yang sudah diterima dan mengambil aksi yang diperlukan.

PIPE : Penghubung Antara Host Dan Endpoint

Sebelum sebuah transfer host-device dapat terjadi, antara host dan device harus terjalin sebuah jalur komunikasi yang dinamakan ***pipe*** (pipa). Dinamakan demikian karena memang pada kenyataannya merupakan jalur data yang dibentuk oleh software di tingkat host kontroler.
Host menyediakan pipe selama proses enumerasi. Jika device tidak lagi terhubung ke device, host juga memindahkan pipe yang tidak dibutuhkan lagi. Host mungkin juga meminta pipe baru atau pipe yang sudah tidak diperlukan namun diperlukan pada saat device terhubung kembali. Setiap device mempunyai sebuah **Default Control Pipe** yang menggunakan Endpoint 0.
Konfigurasi informasi yang diterima host mencakup sebuah *endpoint descriptor* untuk setiap endpoint yang akan digunakan oleh device. Setiap *endpoint descriptor* merupakan blok informasi yang menyuruh host untuk mencari tahu apa yang dibutuhkan host untuk berkomunikasi dengan endpoint. Informasi ini mencakup alamat endpoint, jenis transfer yang digunakan, ukuran paket data maksimum, kapan waktu pentransferan dilakukan, dan interfal waktu transfer.

TIPE / JENIS TRANSFER

USB didesain untuk menghandel beberapa peripheral dengan berbagai macam *transfer rate*, *respon time*, dan *error checking*. Empat tipe transfer dari transfer data masing masing mempunyai fungsi dan kegunaan yang berbeda, dan device dapat menggunakan salah satu jenis transfer yang paling sesuai dengan fungsi dari device yang bersangkutan. Seperti telah disinggung dimuka keempat tipe transfer tersebut adalah kontrol, bulk, asinkron, serta interrupt.

Copyright © 2003-2007 IlmuKomputer.Com

	Jenis transfer			
	Kontrol	**Bulk**	**Interrupt**	**Asinkron**
Contoh kegunaan secara umum	Konfigurasi dan identifikasi device	Printer, scanner, Drive	Mouse, Keyboard, Game Kontroller	Streaming audio, video
Memperbolehkan Low Speed	Ya	Tidak	Ya	Tidak
Byte/milidetik transfer data, nilai maksimum transfer per pipe (High Speed)	15872 byte/ milidetik. (31X 64 byte transaksi/ mikroframe)	53248 byte/ milidetik. (13X 512 byte transaksi/ mikroframe)	24576 byte/ milidetik. (3X 1024 byte transaksi/ mikroframe)	24576 byte/ milidetik. (3X 1024 byte transaksi/ mikroframe)
Byte/milidetik transfer data, nilai maksimum transfer per pipe (Full Speed)	832 byte/ milidetik. (13X 64 byte transaksi/ frame)	1216 byte/ milidetik. (19X 64 byte transaksi/ frame)	64 byte/ milidetik. (1 X 64 byte transaksi/ frame)	1023 byte/ milidetik. (1X 1023 byte transaksi/ frame)
Byte/milidetik transfer data, nilai maksimum transfer per pipe (Low Speed)	24 byte/ milidetik. (3X 8 byte transaksi)	(tidak mendukung Low Speed)	0.8 byte/ milidetik. (8 byte / 10 transaksi)	(tidak mendukung Low Speed)
Arah aliran data	IN dan OUT	IN atau OUT	IN atau OUT (versi 1.0 hanya IN saja)	IN atau OUT
Error Checking	Ya	Ya	Ya	Tidak
Messege data / Stream Data	Message	Stream	Stream	Stream
Jaminan data sampai tujuan	Tidak	Tidak	Tidak	Ya
Jaminan waktu pengiriman	Tidak	Tidak	Ya	Ya

Tabel 3.1. Keterangan berbagai Jenis Transfer (Jan Axelson)

Di antara keempat tipe transfer tersebut, transfer kontrol merupakan satu-satunya transfer yang wajib harus bisa dilakukan oleh semua jenis device. Transfer Kontrol meng*enable* host untuk membaca informasi mengenai device, mengeset alamat device, serta memilih konfigurasi dan beberapa setting yang diperlukan.

Transfer jenis Bulk digunakan dimana jumlah byte per-transaksi bukanlah sesuatu yang terlalu penting, melainkan jaminan akan keakuratan data yang diterima. Jenis Bulk ditemui pada jenis antarmuka USB seperti pada printer, scanner, atau pengaksesan file pada drive eksternal. Hanya full dan High Speed yang dapat melakukan transfer Bulk.
Transfer Interrupt digunakan pada device yang harus menerima interrupt (interupsi) host/device secara berkala. Transfer ini merupakan salah satu jalan yang mengijinkan transfer data pada kecepatan Low Speed. Keyboard, Mouse, Game Kontroller menggunakan jenis transfer interrupt.

Transfer asinkron dapat menjamin waktu yang digunakan untuk bertransaksi data, tetapi tidak ada *error checking*. Jenis transfer asinkron banyak digunakan untuk transfer data dengan jumlah byte yang besar, seperti pada streaming audio, atau video, dan hanya didukung oleh High Speed dan Full Speed.

"STREAM PIPE" DAN "MESSAGE PIPE"

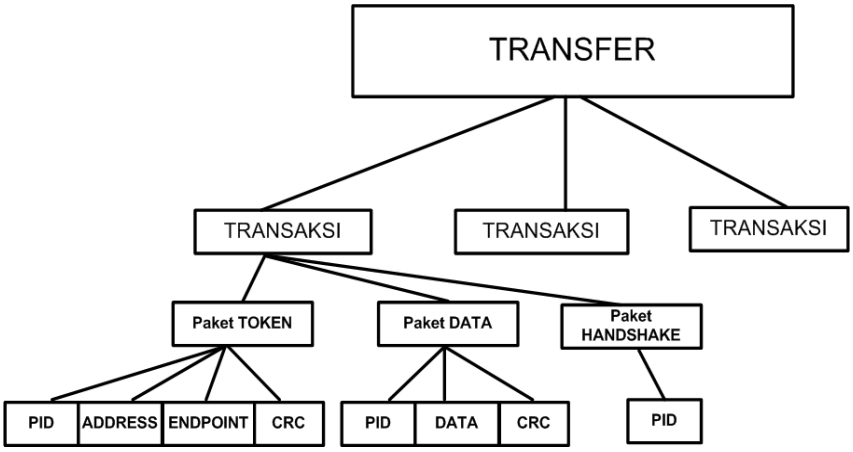
Jenis pipe dibedakan menjadi dua jenis yaitu **stream pipe** (aliran) dan **message pipe** (pesan) . Jenis transfer juga mempengaruhi jenis pipe yang dibangun oleh host kontroler. Message Pipe digunakan untuk Transfer Kontrol, sedangkan ketiga jenis transfer yang lain mengunakan Stream pipe. Seperti analogi kata Message pipe, pipe jenis ini digunakan melayani data yang hanya sebatas informasi (pesan) device. Berbeda dengan stream pipe yang melayani data yang berbentuk aliran, tentu saja dengan jumlah data yang jauh lebih banyak dari pada message pipe. Didalam message pipe setiap transfer dimulai transaksi Setup yang mengandung permintaan data ke device

Copyright © 2003-2007 IlmuKomputer.Com

menggunakan jenis transfer kontrol. Stream pipe, jenis pipe ini tidak mempunyai struktur standard. Host maupun device menerima data apapun yang datang. Firmware maupun aplikasi host hanya akan mengolah data yang masuk, hanya jika berhubungan dengan aplikasi yang bersangkutan.

TRANSAKSI DATA

Setiap transfer data USB terdiri dari beberapa (bisa satu atau lebih) transaksi data. Setiap transaksi bisa mengandung paket **token**, dan mungkin masih ditambah lagi dengan paket **data**, serta paket **handshake** (jabat tangan). Setiap paket-paket tersebut terdiri dari sebuah PID (Packet Identifier) dan atau ditambah lagi dengan informasi tambahan serta CRC (error checking).



Gambar 3.1. Bagian-bagian dari sebuah transfer data secara umum

Ketiga jenis transaksi tersebut ditentukan oleh tujuan dan arah aliran data. Setup mengirimkan transfer kontrol ke device. **Transaksi OUT** mengirimkan informasi status ke device. Sedangkan **transaksi IN** mengirimkan data/status informasi data ke host. Setiap transaksi bisa merupakan sebuah proses identifikasi device, check error, cek status, atau informasi kontrol. Sehingga transfer yang komplit mungkin membutuhkan beberapa frame atau mikroframe dan selama transaksi sedang berlangsung tidak boleh ada interupsi.

SESI (Bagian-bagian) TRANSAKSI

Setiap transaksi (Gambar 3.1) terdiri atas 3 bagian: TOKEN, DATA, dan HANDSHAKE. Setiap fase sesi terdiri atas satu atau dua paket, masing-masing merupakan blok informasi dengan format data tertentu. Semua paket tersebut selalu dimulai dengan Packet ID (PID) yang berguna untuk mengidentifikasi informasi mengenai device. Di dalam sebuah transaksi, PID bisa jadi diikuti oleh alamat endpoint, data, informasi status, atau nomor frame tertentu disertai dengan bit error checking.

Pada sesi TOKEN, host memulai dengan mengirimkan paket token. PacketID (PID) sendiri mengindikasikan jenis transaksi (Setup, IN, OUT) atau Start Of Frame. Sesi TOKEN mempunyai kegunaan tambahan yaitu sebagai pembawa SOF (Start Of Frame), dimana dikirim oleh host dengan interval 1 mili detik jika Full Speed, dan 125 mikrodetik pada High Speed. Paket TOKEN juga mengandung nomor frame yang selalu bertambah dan balik lagi jika sudah mencapai nilai maksimum. Penomoran frame berguna untuk indikator urutan frame, sehingga dalam 8 mikroframe (Full Speed) akan mempunyai nomor yang sama.
Sedangkan pada sesi DATA, host atau device mengirimkan informasi dalam sebuah paket-paket data. PID berisi tentang urut urutan nilai paket data yang bersangkutan, untuk menjaga agar data

Copyright © 2003-2007 IlmuKomputer.Com

yang mengikuti PID tidak hilang dan tidak saling tertukar jika paket data yang dikirimkan lebih dari satu paket.

Pada tingkat HANDSHAKE / jabat tangan, host atau device saling mengirimkan informasi status melalui paket-paket handshake. PID dalam hal ini mengandung kode status (ACK, NAK, STALL, atau NYET). Sering juga sesi handshake ini dinamakan Paket Status karena memang dalam sesi ini antara host dan device saling bertukar status.

Tipe paket	Nama PID	Nilai	Jenis transfer	Sumber	Kecepa-tan Bus	Keterangan
Token (untuk mengiden-tifikasi jenis transaksi)	OUT	0001	Control, Bulk, Asinkron, interrupt	Host	High, Full, Low Speed	Nomor device dan endPoint untu transaksi OUT (host ke device)
	IN	1001	Control, Bulk, Asinkron, interrupt	Host	High, Full, Low Speed	Nomor Device dan endPoint untuk transaksi IN (device ke host)
	SOF (Start Of Frame)	0101	Start Of Frame	Host	High, Full, Low Speed	Indikator SOF dan nomor frame
	Setup	1101	Kontrol	Host	High, Full, Low Speed	Nomor Device dan endpoint untu transaksi setup.
Data	DATA0	0011	Control, Bulk, Asinkron, interrupt	Host, Device	High, Full, Low Speed	Togle Data , urut-urutan data PID
	DATA1	1011	Control, Bulk, Asinkron, interrupt	Host, Device	High, Full, Low Speed	Togle Data , urut-urutan data PID
	DATA2	0111	Asinkron	Host, Device	High Speed	Urutan data PID
	MDATA	1111	Asinkron	Host, Device	High Speed	Urutan data PID
Handshake	ACK	0010	Control, Bulk, Asinkron, Interrupt	Host, Device	High, Full, Low Speed	Receiver menerima paket data yang bebas dari error
	NAK	1010	Control, Bulk, interrupt	Device	High, Full, Low Speed	Receiver tidak dapat menerima data atau pengirim tidak dapat mengirim data, atau tidak ada data yang dikirim
	STALL	1110	Control, Bulk interrupt	Device	High, Full, Low Speed	Permintaan Kontrol tidak didukung oleh endpoint
	NYET	0110	Control Write, Bulk OUT, Split	Device	High Speed	Device menerima paket data yang bebas error tetapi belum siap untuk menerima data yang lain.
Tambahan	PRE	1100	Kontrol, Interrupt	Host	Full Speed	Indikator yang dikeluarkan host untuk menandai bahwa paket berikutnya adalah Low Speed
	ERR	1100	Control, Bulk, Asinkron, Interrupt	Hub	High Speed	Error pada Transaksi split (gabungan) .
	SPLIT	1000	Control, Bulk, Asinkron, Interrupt	Host	High Speed	Indikator transaksi split (sebelum TOKEN)
	PING	0100	Control Write, Bulk OUT	Host	Hish speed	Check BUSY untuk bulk OUT dan Control Write (setelah NYET)

Tabel 3.2. *Packet Identifier* yang merupakan informasi tentang sebuah transaksi dalam sebuah paket.

Copyright © 2003-2007 IlmuKomputer.Com

Low Speed tidak menggunakan paket SOF, tetapi menggunakan sinyal End Of Packet (EOP) yang disebut sinyal Lowspeed Keep Alive dan dikirimkan sekali per-frame. Seperti yang dilakukan SOP pada Full Speed, sinyal Lowspeed Keep Alive menjaga device dari keadaan Suspend State. PID yang digunakan untuk Low Speed adalah PRE, kepanjangan dari preambule yang berarti pembuka. Paket PID ini dikirimkan oleh device yang menginformasikan hub jika paket yang mengikutinya merupakan Low Speed. Dalam menerima PID PRE, hub membuka komunikasi dengan seluruh device Low Speed. PID PRE sendiri akan mendahului semua paket TOKEN, DATA, dan HANDSHAKE dan langsung ke device.

PID yang digunakan dalam device High Speed adalah PING. Dalam transfer bulk atau kontrol dengan beberapa paket data, sebelum mengirim paket DATA kedua atau data paket tambahan, host akan mengirimkan PING untuk mendeteksi jika endpoint telah siap untuk menerima data. Device akan merespon dengan kode status. PID SPLIT mengidentifikasi paket TOKEN yang dikirim sebagai transaksi yang dipecah-pecah. PID ERR digunakan hanya di transaksi SPLIT. Hub 2.0 menggunakan PID ERR ini untuk melaporkan error ke host pada transaksi Low sampai Full Speed. PID ERR dan PRE mempunyai nilai yang sama tetapi tidak akan saling tertukar karena hub tidak akan pernah mengirimkan PRE ke host atau ERR ke device. Dan juga ERR hanya digunakan pada High Speed sedangkan PRE tidak digunakan pada segmen transaski High Speed.

URUT-URUTAN PAKET

Setiap transaksi mempunyai paket TOKEN yang dikirimkan oleh host. Paket ini akan berisi identifikasi jenis paket, device yang menerima dan endpoint di dalamnya, serta arah transfer data. Untuk Low Speed yang berada di bus Full Speed, paket PRE akan mendahului paket TOKEN. Sedangkan untuk transaksi yang dipecah pecah, paket SPLIT akan mendahului paket token. Sedangkan paket DATA akan selalu mengikuti paket TOKEN. Pada semua jenis transfer kecuali asinkron, penerima paket DATA akan mengembalikan paket HANDSHAKE yang mengandung kode yang menandakan sukses tidaknya transaksi.

TRANSAKSI YANG DIPECAH (SPLIT)

Sebuah hub 2.0 berkomunikasi dengan host 2.0 dengan kecepatan High Speed pula, kecuali jika sebuah hub 1.x ikut terhubung dengan keduanya. Hub 2.0 harus bisa mengkonversi kecepatan sesuai dengan yang dibutuhkan, padahal High Speed mempunyai kecepatan 320 kali lebih cepat dari pada Low Speed dan 40 kali lebih cepat dari pada Full Speed. Solusi untuk kejadian ini adalah dengan transaksi data yang dipecah (split). Host 2.0 menggunakan transaksi yang dipecah ketika dalam sebuah bus data ditemukan device full dan Low Speed. Transaksi split membagi menjadi satu transaksi Low atau Full Speed setidaknya menjadi 2 jenis tipe transaksi : satu untuk transaksi **split-start** yang mengirimkan informasi ke device dan satu lagi transaksi **split-complete** yang menerima informasi dari device. Namun untuk Transaksi Asinkron OUT tidak menggunakan slit-complete karena tidak ada yang data dikirimkan device.

Walaupun transaksi split kelihatannya lebih ribet dan memerlukan banyak transaksi, transaksi split membuat efisien penggunaan waktu khususnya untuk Low dan Full Speed. Host kontroler 2.0 dan hub 2.0 merupakan komponen yang bertugas untuk membuat transaksi split, dan device sama sekali tidak perlu tahu jika host menggunakan metode transaksi split. Transaksi data yang terjadi di device antara menggunakan transaksi split dengan tidak menggunakan adalah sama. Sedangkan pada sisi host, driver untuk device juga tidak mempedulikan untuk terjadinya transaksi split, karena transaksi split terjadi pada level yang lebih rendah lagi, yakni di sisi hardware host.

Copyright © 2003-2007 IlmuKomputer.Com

HANDSHAKING (JABAT TANGAN)

Transfer USB menggunakan handshaking dan check error untuk memastikan bahwa data sampai ke tujuan dengan cepat dan tanpa error. Seperti antarmuka yang lain, USB menggunakan status dan kontrol atau handshaking/jabat tangan untuk membantu mengatur transfer data. Bedanya dengan versi serial DB9, pada USB semua handshaking ditulis dalam kode software bukan sebuah fisik port yang bisa dilihat. Handshaking pada USB mengindikasikan berhasil atau tidaknya semua transaksi kecuali pada transfer asinkron. Sinyal handshaking dikirimkan dalam paket data atau apaket handshake. Kode status yang digunakan adalah ACK, NAK, STALL, NYET, dan ERR.

Jenis transaksi (PING)	Sumber paket data	Isi paket data	Sumber Paket handshake	Isi paket handshake
Setup	Host	Data	Device	ACK
IN	Host	Data	Devive	ACK, NAK, STALL, NYET (High Speed), ERR (*split complete*)
OUT	Device	Data, NAK, STALL, ERR(hub dengan paket *split- complete*)	Host	ACK
PING (High Speed)	-	-	Device	ACK, NAK, STALL

Tabel 3.3. Hubungan siyal-sinyal Jabat Tangan (*Handshake*) dengan arah transfer.

1. **ACK**

   ACK (Acknowledge) mengindikasikan bahwa host atau device telah menerima data tanpa error. Device harus mengeluarkan sinyal ACK dalam paket handshake pada transaksi Setup ketika paket TOKEN dan paket DATA diterima tanpa error.
2. **NAK**

   NAK (Negative Acknowledge) mengandung arti bahwa device sedang sibuk atau tidak ada data untuk dikembalikan ke host. Jika host mengirim data pada pada waktu device sedang sibuk, maka device akan mengeluarkan kode NAK pada paket handshake. Selain itu jika host meminta data dari device tetapi tidak ada yang harus dikirimkan oleh device, maka device akan mengeluarkan sinyal NAK pada paket data. Dengan kata lain NAK merupakan kode sementara agar host mengulang lagi permintaan data ke device. Host tidak pernah mengirimkan sinyal NAK.
3. **STALL**

   STALL mempunyai beberapa arti : permintaan kontrol yang tidak didukung oleh device, permintaan kontrol gagal, atau endpoint yang gagal. Jika device menerima permintaan transfer kontrol tetapi device tidak mendukung permintaan itu, maka device akan mengirimkan kode STALL ke host. Untuk meng"clear"kan STALL, host hanya perlu mengirimkan paket Setup untuk memulai transfer kontrol dengan nilai konfigurasi yang lain. Kegunaan lain dari STALL adalah jika mikrokontroler memberhentikan fungsi endpoint karena suatu interupsi sehingga tidak bisa menerima dan mengirim data, maka kondisi ini juga dinamakan kondisi STALL. Transfer Bulk dan Interupt mendukung fungsi STALL namun untuk transfer asinkron tidak menggunakan STALL karena tidak menggunakan paket HANDSHAKE. Sinyal kondisi STALL juga hanya dikirim oleh device, host tidak pernah mengirimkan sinyal STALL.
4. **NYET**

Copyright © 2003-2007 IlmuKomputer.Com

NYET kepanjangan dari *Not Yet* dan hanya digunakan oleh device High Speed. Transfer Kontrol dan Bulk pada High Speed mempunyai sebuah protokol yang memungkinkan host untuk mengecek kesiapan device sebelum mengirimkan data. NYET digunakan untuk menghemat waktu yang diperlukan selama proses yang terjadi low atau Full Speed, karena kode NAK harus dikirim berulang-ulang untuk memberitahu host bahwa device belum siap.
Cara kerja NYET adalah sebagai berikut: transaksi kontrol dan bulk pada High Speed dengan beberapa paket DATA datang, lalu device mengembalikan handshake NYET yang mengatakan bahwa endpoint telah menerima data tetapi belum siap menerima paket data yang lain. Apabila device telah siap, host akan mengirimkan paket token PING dan endpoint akan mengembalikan sebuah kode ACK untuk memberitahu bahwa device talah siap untuk menerima data lagi, atau kode NAK/STALL jika belum siap. Mengirimkan PING jauh lebih efisien daripada semua paket data yang hanya untuk memberitahu bahwa device telah siap untuk menerima data.

5. **ERR**

Kode Handshake ERR hanya digunakan oleh hub High Speed untuk melengkapi transaksi data yang dipecah. ERR mengindikasikan bahwa device tidak tidak merespon sejumlah paket handshake dalam hal berkomunikasi dengan host melalui hub.

Jenis transfer	Arah aliran	Paket Data	Paket Handshake
Kontrol - Write (Host mengirimkan data ke device)	IN	Device mengirimkan status: ZLP (sukses), NAK (busy), STALL (gagal)	Host mengembalikan ACK
Kontrol - Read (Device mengirimkan data ke host)	OUT	Host mengirimkan ZLP	Device mengirimkan status: ACK(sukses), NAK(busy), STALL (gagal)

Tabel 3.4 Indikator status dari transfer kontrolter masuk di dalamnya ZLP (paket dengan lebar 0 byte).

Untuk melaporkan status dari transaksi, sebuah tambahan paket status ditambahkan berupa *Zero Length Data Packet (*ZLP), yang berisi laporan lengkap tidaknya transfer kontrol. ZLP berisi data PID dan bit error checking, tetapi tidak ada paket data.

CHECKING ERROR

Checking error digunakan untuk mencegah terjadinya kesalahan data yang diakibatkan oleh noise baik internal maupun eksternal. Noise internal bisa berupa kondisi penghantar yang kurang bagus atau kurang memenuhi spesifikasi standar. Gangguan dari luar misalnya adanya harmonik sumber listrik, induksi medan magnet di sekitar kabel penghantar. Dalam hal ini koreksi galat (error checking) sangat diperlukan yaitu dengan adanya bit error-checking.

Semua paket TOKEN, DATA, dan SOF mengandung bit untuk digunakan sebagai error-checking. Nilai bit tersebut dihitung berdasarkan nilai CRC (*Cyclic Redundancy Check*). CRC sendiri dilakukan ditingkat hardware. Pertama-tama data yang akan dikirim setelah dikalkulasi dengan metode CRC, lalu dikirim bersama-sama dengan data itu pula. Dari sisi penerima, data tersebut akan dikalkulasi. Jika hasil antara bit CRC dengan isi data maka akan dikirimkan sinyal ACK, jika belum akan dikirimkan sinyal paket HANDSHAKE yang mengisyaratkan agar data dikirim ulang. Penentuan pengulangan diisyaratkan oleh host sampai n kali (tergantung pemrogram). Apabila gagal, host akan menginformasikan driver tentang kegagalan transaksi data.

Copyright © 2003-2007 IlmuKomputer.Com