PEMANFAATAN PIC18F4550 SEBAGAI ANTARMUKA KOMUNIKASI USB UNTUK PENCACAH FREKUENSI

Tugas Akhir

Sebagai salah satu syarat untuk memperoleh gelar sarjana sains
dalam bidang Fisika

Oleh :
MUH. ALI SYAHBANA
0810930049-93

JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA

2013


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN TUGAS AKHIR
FREKUENSI KONTER DENGAN KOMUNIKASI USB HID

Oleh :
MUH. ALI SYAHBANA
0810930049-93

Setelah dipertahankan didepan majelis penguji
Pada tangal: ……..
dinyatakan memenuhi syarat untuk memperoleh gelar sarjana Sains
dalam bidang Fisika

Pembimbing I	Pembimbing II
Dr. Ing. Setyawan P.S., M.Eng	Dr. Hari Arief D., M.Eng
NIP. 196508251 999002 1 001	NIP. 19690920 199412 1 001

Mengetahui
Ketua Jurusan Fisika
Fakultas MIPA Universitas Brawijaya

Drs. Adi Susilo, M.Si Ph.D
NIP. 19631227 199103 1 002

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Muh. Ali Syahbana
NIM : 0810930049
Jurusan : FISIKA
PenulisanTugas Akhir Berjudul :

PEMANFAATAN PIC18F4550 SEBAGAI ANTAR MUKA KOMUNIKASI USB UNTUK PENCACAH FREKUENSI

Dengan ini menyatakan bahwa:

1. **Isi dari tugas akhir yang saya buat adalah benar-benar karya sendiri.**
2. **Apabila dikemudian hari tugas Akhir yang Saya tulis terbukti hasil jiplakan, maka saya bersedia menanggung segala resiko yang saya terima.**

Malang, Juni 2013
Yang menyatakan,

(MUH ALI SYAHBANA)
NIM. 0810930049-93


UNIVERSITAS BRAWIJAYA

PEMANFAATAN PIC18F4550 SEBAGAI ANTAR MUKA KOMUNIKASI USB UNTUK PENCACAH FREKUENSI

ABSTRAK

Komunikasi data dapat dilakukan dengan menggunakan serial atau parallel.Komunikasi serial banyak digunakan karena menggunakan kabel yang sedikit.Komunikasi USB merupakan komunikasi serial yang sering digunakan dalam perangkat instrumentasi. Dengan memanfaatkan komunikasi USB pengiriman data instrument ke komputer dapat berjalan cepat dan mudah. Dalam penelitian ini telah dilakukan pembuatan pencacah frekuensi dengan mennggunakan mikrokontroller, pengiriman data hasil pencacahan dilakukan menggunakan komunikasi USB. Untuk memudahkan dalam penggunaan, sistem komunikasi menggunakan kelas HID dan ditangani secara langsung oleh mikrokontroller PIC18F4550. Hasil percobaan menunjukkan pengiriman data hasil pencacahan dapat dilakukan dengan baik dan diterima oleh program pada komputer tanpa memerlukan tambahan driver khusus pada komputer.Data pencacahan frekuensi dikirimkan ke komputer setiap 1 detik.Dengan mengikuti standar protokol format data, data yang dikirimkan dapat diterima oleh komputer sebagai data frekuensi pencacah dan selanjutnya dapat diolah oleh komputer.

UTILIZATION OF COMMUNICATION PIC18F4550 USB INTERFACE AS FOR FREQUENCY COUNTER

ABSTRACT

Data communication can be acomplished by using either serial or parallel means. Serial communication is widely used because it requires only a fewcables. USB communication is a serial communication which is often used in instrumentation device. By incorporating USB communication, instrument data transfer to computer can be accomplished in easy and fast way. During this research, a microcontroller based-frequency counter device was constructed. The data result from the counter ware then transfered using USB communication. To obtain the ease of use of the operation, communication system was done by using HID class and handled directly by PIC18F4550 microcontroller. The results from the experiment showed that the data transfer has been done correctly and recieved by the program on the computer without the requirements of additional driver. The frequency counter data was sent to the computer for every 1 second. By following the data format standard protocol, the data that was sent were recieved by computer as a frequencies counter data and can be processed for further purposes.

KATA PENGANTAR

Segala puji dan syukur dipanjatkan kehadirat ALLAH SWT, yang selalu mencurahkan kasih, sayang, ridho, dan rahmatNYA, sehingga penulis dapat menyelesaikan tugas akhir ini yang berjudul **PEMANFAATAN PIC18F4550 SEBAGAI ANTAR MUKA KOMUNIKASI USB UNTUK PENCACAH FREKUENSI.**

Dengan selesainya penelitian dan penulisan skripsi ini, penulis mengucapkan terimakasih pada semua pihak yang telah membantu baik secara material maupun perbuatan, baik secara langsung maupun tidak langsung. Pada kesempatan ini penulis ingin mengucapkan terimakasih yang sebesar-besarnya kepada :

1. Ibu, Ayahdan Adek tercinta yang sealu mendo'akan, menyayangi sipenulis tercinta.
2. Dr. Ing.Setiawan P.S., M. Eng dan Drs. Hari Arief Dharmawan, M.Eng selaku dosen pembimbing yang telah meluangkan waktu untuk membimbing, memberikan masukan, mengarahkan dan dorongan kepada penulis, hingga penulis menyelasaikan dengan baik.
3. قارئ الدنس الذي أحسن خلقه و جميلة
4. Para dosen jurusan Fisika UB Pak Adi (selaku kajur), Didik, Sunaryo(selaku desen PA), Arinto, Joko, Johan, Nabayang telah mengajarkan ilmunya.
5. Sasi, Yesika, Zarnuji (untuk bantuan di panggungnya), Mas Aul, kevin, kruger, angkatan 2009 dan teman yang tidak sempat saya sebutkan, terimakasihku untuk tragedi tanggal 24 mei 2013.
6. Anggota 911 Atok, Ali, Winartoy, Ratih, Iin.
7. Kakak sepupu Mas Den, Arip, Habib, Kholis, Mbak Is, Likah, Lidia, Waroh, dan yang lainNya yang tidak bisa disebutkan.
8. Elok Farida, Ladima, Eny, dan Lidia.
9. Ayu, Dewi, Wahyu, Barika, danTeman-teman satu angkatan 2008 yang tidak dapat saya sebutkan satu-satu.
10. Teman ngelab Mas Vani + Mbak Risa, Sigit, Mbak Bibi, Veni.
11. Team Phy_Ro_C Mas ulin, Lalu, Imam, Muklis.
12. Adek tingkat Phy_Ro_C Ike & Acem (trims Kuenya), Nur, Adwi, Tosu, Lia, Vina, Tasi, dan lainnya yang tidak bisa saya sebut satu-satu.

13. Keluarga besar Sukorejo-situbondo, Mas Alimin,Najib, Madid, Habibi, Nanda dan yang tidak dapat saya sebutkan satu-satu.
14. Para staf dan karyawan Fisika UB, Mas Adi, Dani, dan yang lainnya.
15. Mas Yus, Bagus yang mengajari aku mikrokontroler.
16. Teman satu geng P4 Vero, Panji, Fathul
17. Para senpai dan kohe kempo khususnya Dojo Universitas Brawijaya, sekaligus anak2 UKM.

Penulis menyadari bahwa dalam tugas akhir ini masih belum sempurna, sehingga kritik dan saran pembaca dapat membuat lebih baik. Penulis memohon maaf apabila ada kesalahan selama pengerjaan dan penulisan skripsi ini.Dan semoga tugas akhir ini dapat bermanfaat bagi pembaca, khususnya mahasiswa jurusan Fisika UB.

Malang, Juni 2013

Penulis

DAFTAR PUSTAKA

LEMBAR PERNYATAAN v
ABSTRAK vii
ABSTRACT ix
KATA PENGANTAR xi
DAFTAR PUSTAKA xiii
DAFTAR GAMBAR xvi
DAFTARTABEL xvii
DAFTAR LAMPIRAN xviii
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan Penelitian 2
1.4 Manfaat Penelitian 2
BAB II DASAR TEORI 3
2.1 PIC18F4550 3
2.1.1 Memori Program 4
2.1.2 ALU (Arithmetic and Logic Unit) 4
2.1.3 Pembangkit *Clock-Oscillator* 4
2.1.4 Interupsi 5
2.1.5 Timer 5
2.2 USB 6
2.2.1 Konektor dan Kabel 7
2.2.2 Pengenalan pada USB 9
2.2.3 SYNC 9

2.2.4 ENDPOINT 9
2.2.5 Paket Token 10
2.2.6 Komunikasi Data USB 10
2.3 HID 13
2.4 Counter 13
2.5 Frekuensi 14
BAB III METODOLOGI PENELITIAN 15
3.1 Tempat dan Waktu penelitian 15
3.2 Alat dan Bahan 15
3.3 Tahap Penelitian 17
3.3.1 Persiapan 17
3.3.2 Perakitan Hardware 18
3.3.3 Pembuatan Program 18
3.3.4 Pengujian Alat 19
3.4 Diagram Blok 19
3.5 Komunikasi USB-HID 20
3.6 Flowchart 22
3.7 Pengambilan Data 23
BAB IV HASIL DAN PEMBAHASAN 25
4.1 Hasil 25
4.1.1 Pengujian komunikasi USB 25
4.1.2 Pengujian Pencacah Frekuensi 25
4.2 Pembahasan 28
4.2.1 Protokol USB 29
4.2.2 Pengiriman Data 30

4.3 Penampilan Display 30
BAB 5 KESIMPULAN 33
5.1 Kesimpulan 33
5.2 Saran 33
DAFTAR PUSTAKA 35
Lampiran 37

DAFTAR GAMBAR

Gambar 2.1 PIC18F4550 3
Gambar 2.2 Konektor USB 7
Gambar 2.3 USB Hub 8
Gambar 3.1 PICkit 3 15
Gambar 3.2 MPLAB 16
Gambar 3.3 Tahap Penelitian 17
Gambar 3. 4 Skematik 18
Gambar 3.5 Diagram blok pengiriman data 19
Gambar 3.6 Flowchart USB-HID 20
Gambar 3.7 Flowchart Program Mikrokontroler 22
Gambar 4.1 Perangkat USB Pencacah Frekuensi 25
Gambar 4.2HID Terminal 25
Gambar 4.3 Flowchart USB 29
Gambar 4.4Terminal USB HID 31

DAFTARTABEL

Tabel 2.1 Pin USB 8
Tabel 2.2 Format ENDP 9
Tabel 2.3 Format Pengirimsan PID 11
Tabel 2.4 Nilai PID 11
Tabel 4.1 Nilai Cacahan 10-90 Hz 26
Tabel 4.2 NIlai Cacahan 100-900 Hz 26
Tabel 4.3 NIlai Cacahan 1- 9 KHz 26
Tabel 4.4 NIlai Cacahan 10-90 KHz 27
Tabel 4.5 NIlai Cacahan 100 - 900 KHz 27
Tabel 4.6 Pengiriman Data USB 30

DAFTAR LAMPIRAN

Lampiran 1Sourcode Pencacah Frekuensi.....37
Lampiran 2 Sourcode Komunikasi USB HID 41

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pengiriman data dari suatu perangkat ukur atau kendali ke komputer dapat dilakukan dengan mengunakan cara komunikasi serial atau parallel. Cara komunikasi serial banyak dipergunakan terutama karena penggunaan kabel yang sedikit, noise yang rendah dan interferensi antar jalur data yang rendah.Dengan sifat ini maka komunikasi data dengan jarak koneksi yang panjang dapat dilakukan.Hal ini tidak bisa dilakukan dengan menggunakan komunikasi parallel.Namun demikian untuk komunikasi data dengan jarak pendek dan kecepatan pengiriman data yang sangat tinggi, komunikasi parallel masih banyak dipergunakan.

Standar komunikasi serial ada beberapa seperti: RS232, RS485, SPI (*Serial Peripheral INterface*), I^2C (*Inter Integrated Circuit*), dan USB. Komunikasi RS232 digunakan untuk komunikasi suatu perangkat dengan komputer, dan membutuhkan driver untuk mengenali informasi seperti nomor port com, baud rate, parity, data bit, dan stop bit.Dalam penggunaan komunikasi RS232 harus memerlukan driver pada komputer,membutuhkan IC MAX232 untuk mengubah level tegangan agar dapat dibaca oleh komputer, dan DB9.Komunikasi serial juga membutuhkan sumber tegangan sendiri.

Komunikasi serial telah dikembangkan menjadi komunikasi USB. Denganmenggunakan komunikasi USB komunikasi data lebih cepat, dan yang terpenting tidak memerlukan sumber tegangan dari luar (sumber tegangan langsung dari komputer). Komunikasi USB pada saat ini sudah memiliiki kecepatan transfer yang tinggi.

Komputer maupun laptop saat ini semua sudah dilengkapi komunikasi USB. kelas USB yang sering digunakan untuk peralatan adalah kelas HID, dimana USB kelas HID ini tidak memerlukan hardware dan driver tambahan, karena *protocol* dari USB kelas HID sudah terpasang pada OS (*operating system*) seperti Windows, Linux, maupun MAC. peralatan yang memanfaatkan komunikasi USB kelas HID ini seperti mouse, keyboard, kamera digital, dan peralatan lainya yang mendukung komunikasi ini.Dengan memanfaatkan protokol HID dapat mempermudah sistim kerja dari suatu alat instrumentasi. Alat

instrumentasi yang dapat didukung komuniksi USB-HID antara lain pencacah frekuensi, dimana pencacah frekuensi banyak dibutuhkan dalam bidang penelitian seperti penelitian QCM.

Nilai frekuensi yang dicacah dapat ditampilkan ke komputer dengan cara mengirimkan data cacahan ke komputer melalui komunikasi data. Dengan menggirimkan data dari mikrokontroler PIC18F4550 ke komputer dapat dilakukan dengan menggunakan komunikasi USB, karena mikrokontroler PIC18F4550 mendukung komunikasi USB. Agar tidak membutuhkan software tambahan maka dapat memanfaatkan komunikasi USB kelas HID dimana komputer sudah mendukung komunikasi ini tanpa harus tambahan software.

1.2 Rumusan Masalah

- Bagaimanaperangkat keras sistem pencacah frekuensi dengan menggunakan komunikasi data melalui USB ke komputer ?
- Bagaimana pengiriman datamelalui protokol HID untuk pencacah frekuensi ?

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk membuat alat pencacah frekuensi dengan pengiriman data menggunakan komunikasi USB kelas HID.

1.4 Manfaat Penelitian

Penelitian ini diharapkan dapat menghasilkan alat pencacah frekuensi yang dapat diterapkan pada alat yang membutuhkan pencecah frekuensi dan dapat disambungkan ke alat yang lebih modern dengan melalui komunikasi USB dengan lebih sederhana.

BAB II
DASAR TEORI

2.1 PIC18F4550

Mikrokontroler PIC18F4550 merupakan mikrokontroler yang diproduksi Microchip Inc, yang mengimplementasikan arsitektur Hardvard dimana memori data dan memori program diakses dengan bus yang berbeda, sehingga eksekusi program secara keseluruhan dapat berlangsung lebih cepat. Mikrokontroler PIC18F4550 dilengkapi dengan beberapa kebutuhan seperti 10 bit ADC, timer, I/O, ISP, usart, port ICSP, dan juga mendukung komunikasi USB(Microchip, 2006).

Gambar 2.1PIC18F4550

Mikrokontroler 18F4550 dilengkapi port I/O, yang dapat disambungkan dengan alat intrumentasi lainnya. Untuk pengaturan input/otputterdapat pada pengaturan program. Port I/O ini terdiri dari port A sebanyak 6 bit, port B sebanayak 8 bit, port C sebanyak 7 bit, port D sebanyak 8 bit, port E 3 bit. Register TRIS pada mikrokontroler digunakan untuk mengatur pin yang ada pada mikrokontroler PIC18F4550, apakah sebagai masukan atau keluaran. TRIS dengan nilai 0 menjadikan pin pada mikrokontroler sebagai keluaran sedangkan nilai 1 menjadikan masukan(Microchip, 2006).

PIC18F4550 memiliki fitur khusus seperti bisa menggunakan kompiler C, 100 ribu baca tulis untuk flas memori, 1 juta baca tulis data untuk memori EEPROM, flas/data EEPROM bisa bertahan lebih dari 40 tahun, vector interrupt, WTD (watchdog timer), proteksi kode program,32 KB program memory, 2048 byte SRAM, 256 byte data EEPROM, 20 sumber interupsi, 16-bit timer, 8-bit timer, 13 chanel 10

bit ADC, USART, serial, parallel, SPI, I2C interface, dan range tegangan 2V sampai 5,5 V sekaligus komunikasi USB yang sudah tidak membutuhkan komponen tambahan(Microchip, 2006).

2.1.1 Memori Program

Memori program digunakan untuk menyimpan program. Satu kata berukuran 32 bit dan 16384 kata dapat disimpan. Memori program dapat dibaca, ditulis, dan dihapus.Dalam proses memasukan atau menghapus program pada mikrokontroler instruksi akan berhenti sampai operasi selesai (Microchip, 2006).

2.1.2 ALU (Arithmetic and Logic Unit)

ALU merupakan bagian dari mikrokontroler yang bertanggung jawab terhadap operasi aritmatika seperti penjumlahan, pengurangan, pembagian, dan perkalian.Selain itu juga terdapat operasi logika seperti AND, OR dan lainnya. ALU bekerja sama dengan memori, dimana hasil dari perhitungan disimpan pada memori.

Rangkaian ALU yang digunakan untuk penjumlahan merupakan rangkaian adder. Terdapat 3 macam adder yaitu half adder (penjumlahan bilangan biner yang terdiri satu bit), full adder, dan parallel adder (Microchip, 2006).

2.1.3 Pembangkit *Clock-Oscillator*

Clock yang digunakan untuk masukan mikrokontroler PIC18F4550 terdiri dari dua jenis yaitu clock yang berasal dari dalam (*internal clock*) dan clock yang berasal dari luar (*external clock*).

Penggunakan clock internal diset pada program, karena didalam mikrokontroler PIC18F4550 dilengkapi pembangkit clock tersendiri. Untuk clock dari luar dapat menggunakan 3 jenis clock yaitu: oscillator, crystal, dan rangkaian RC. Penggunaan *crystal* sebagai *clock* PIC18F4550 memerlukan penambahan kapasitor, sedangkan untuk oscillator tidak membutuhkan komponen tambahan.

Clock diperlukan mikrokontroker untuk mensinkronkan proses yang berlangsung dalam mikrokontroler. Sumber *clock*untuk mikrokontroler PIC18F4550 dibuatdengan menambahkan rangkaian *crystal* dan kapasitor atau dengan menggunakan komponen *oscillator* secara langsung. PIC18F4550 memiliki12 konfigurasi untuk *clock* yaitu:

1. XT (crystal/resonator)

2. XTPLL (crystal/resonator dengan PLL)
3. HS (high-speed cristal/resonator)
4. HSPLL (high-speed crystal/resonator dengan PLL)
5. EC (exterlan clock dengan keluaran $\frac{Fosc}{4}$)
6. ECIO (external clock dengan I/O pada RA6)
7. ECPLL (external clock dengan PLL dan keluran $\frac{Fosc}{4}$ pada RA6
8. ECPIO (external clock dengan PLL dan I/O pada RA6
9. INTHS (internal osilator digunakan untuk clock mikrokontroler dan HS digunakan sumber clock USB)
10. INTXT (internal osilator sebagai clock mikrokontroler dan XT digunakan sebagai sumber clock USB)
11. INTIO (internal osilator sebagai sumber clock mikrokontroler dan EC digunakan sebagai sumber clock USB).
12. INTCKO (menggunakan osilator internal untuk sumber clock mikrokontroler dan EC digunakan untuk sumber clock USB, dan keluaran osilator $\frac{Fosc}{4}$ pada RA6

(Microchip, 2006).

2.1.4 Interupsi

Interupsi adalah suatu kejadian atau peristiwa yang terjadi pada mikrokontroler yang mengharuskan mikrokontroler tersebutberhenti melakukan alur instruksi dan berganti mengerjakan instruksi lain. Program yang dijalankan ketika melakukan*interrupt*(instruksi) disebut ***Interrupt Service Routine***. Pada saat terjadi instruksi program akan berhenti sesaat untuk menjalankan program instruksi dan kembali ke program utama setelah program instruksi selesai(Steiner, 2005).

2.1.5 Timer

Timer merupakan fitur yang ada disetiap mikrokontroler, Peranan timer sangat banyak dalam perancangan sebuah sistim.Timer dapat digunakan untuk menunda eksekusi dari suatu perintah, sehingga sistem mikrokontroler yang berjalan cepat dapat sinkron denga operator.Dengan timer pula dapat menghitung lebar pulsa/frekuensi dari sinyal, menghitung kecepatan jatuh suatu benda atau kecepatan suatu kendaraan.Karena fungsi timer yang banyak dalam suatu sistem, maka banyak *vendor*memasang timer kedalam mikrokontroler lebih dari satu, dengan panjang data 8sampai 16 bit. Dalam mikrokontroler

PIC18F4550 ada 4 timer antara lain: Timer 0, timer 1, dan timer 3 memiliki 8 bit low dan 8 bit high yang jumlah totalnya 16 bit yang dapat diatur penggunaannya apakah 8bit ataukah 16 bit, sedangkan untuk timer 2 hanya memiliki 8 bit(Microchip, 2006).

2.2 USB

USB (*Universal Serial Bus*) adalah standar komunikasi serial bus yang digunakan untuk menghubungkan suatu peralatan (komputer ke perangkat USB).Pada awalnya USB didesain untuk komputer.karena kemudahan dalam komunikasimaka dalam aplikasi perangkat ini dikembangkan keperangkat lain seperti konsole video game, telpon seluler, dan lainnya. USB juga memerlukan sumber daya yang besarnya sekitar 500mA (Reznik dan Hershenhoren, 2010).

Komunikasi USB tidak menggunakan istilah transmitter dan reciver, tetapi menggunakan tiga istilah yaitu:

a. Paket token adalah header yang mendefinisikan apa yang mengikuti.
b. Optional paket data adalah mengandung data payload.
c. Paket status adalah digunakan untuk status paket atau status kesalahan (Reznik dan Hershenhoren, 2010).

USB merupakan *host-centric* bus dimana host/terminal induk memulai semua transaksi. Paket pertama/penanda (token) awal dihasilkan oleh host untuk menjelaskan apakah paket yang mengikuti akan dibaca atau ditulis dan apa tujuan dari perangkat dan titik akhir. Paket berikutnya adalah paket yang diikuti oleh *handshaking* packet yang melaporkan apakah data atau penanda sudah diterima dengan baik atau pun titik akhir gagal menerima data dengan baik.

Saat ini terdapat 4 jenis kecepatan komunikasi dalam USB yaitu:

1. Low speed dengan kecepatan pengiriman data 1,5Mbps dan toleransi pensinyalan data antara 15.000ppm
2. Full speed dengan kecepatan pengiriman data 12Mbps dengan toleransi pensinyalan data sekitar 2.500ppm
3. High speed dengan kecepatan pengiriman data 480Mbps dengan toleransi pensinyalan data sekitar 500ppm
4. Super speed dengan kecepatan pengiriman 4.800Mbps.

Komunikasi USB pada dasarnya dikembangkan dari komunikasi serial.Komunikasi USB hanya membutuhkan kabel yang sedikit dan mudah diterapkan dalam penggunaannya, dengan memasang dan

melepas tanpa ada *booting* lagi walaupun PC dalam keadaan masih bekerja.

USBsering dijumpai pada peralatan elektronik pada jaman sekarang ini seperti pada printer, komputer, scanner, hanphone, mouse, kamera digital, flash driver. Sebelum ada USB komunikasi data lebih dikenal dengan menggunakan port serial dan parallel. Namun untuk saat ini, port USB telah menjadi port komunikasi standar yang banyak dimanfaatkan.

Komunikasi USB juga dimanfaatkan pada perangkat bio sensor, dimana data yang dikirim ke komputer merupakan nilai ADC yang telah diubah nilainya ke bentuk ASCII(Loncaric dkk., 2011).

Komponen CH372 dapat digunakan untuk komunikasi USB. Dari suatu mikrokontroler yang memliki data (semisal data ADC) dapat mengirimkan data ke komputer dengan hanya memanfaatkan IC CH372, dimana CH372 difungsikan sebagai antarmuka(Niu, 2012).

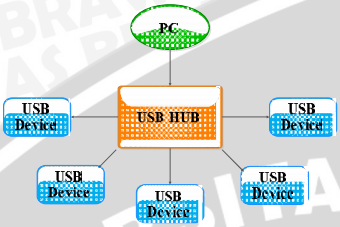
2.2.1 Konektor dan Kabel

Konektor USB terdapat dua tipe, yaitu tipe A dan tipe B yang seperti terlihat pada gambar 2.2 dimana kedunya memeliki fungsi yang sama untuk komunikasi data, pada umumnya konektor tipe A dihubungkan pada terminal USB yang ada di komputer sedangkan konektor tipe B dihubungkan pada peralatan USB (Brey, 2002).

Gambar 2.2 Konektor USB

USB hub seperti pada gambar 2.3 digunakan untuk menyambungkan banyak peralatan USB dengan cara membuat percabangan, Bagian alamat dari peralatan dimana paket digunakan. Dengan lebar 7 bit akan terdapat 127 peralatan dapat disambungkan. Alamat 0 tidak sah, peralatan yang belum terdaftar harus merespon paket yang dikirim ke alamat 0. Terdapat 2 jenis USB Hub, yang

pertama USB Hub dengan sumber daya sendiri dan yang kedua USB Hub dengan sumber tegangan dari kabel USB sendiri(Axelsion, 1999).



Gambar 2.3 USB Hub

Jalur USB terdapat 4 pin untuk komunikasi seperti terlihat pada gambar 2.4. Pin nomor 1 digunakan untuk VCC atau tegangan 5 volt, Pin nomor 4 digunakan untuk ground, Pin nomor 2 diberi nama D- sedangkan pin nomor 3 diberi nama D+. pin D-/D+ digunakan untuk tempat transfer data atau untuk komunikasi, tegangan pada pin 2 dan 3 antara 0-3,3 Volt. Sinyal digital yang dikirimkan melalui pin 2 dan 3 tidak dinyatakan dengan besarnya tegangan pada saluran tersebut terhadap ground melainkan dinyatakan dengan perbedaan tegangan antara dua kabel. Jika tegangan pada saluran D+ lebih tinggi dari tegangan pada saluran D- maka informasi yang dikirim adalah sinyal 1, sebaliknya sinyal digital 0 dinyatakan dengan tegangan D+ lebih kecil dari tegangan D- (Axelsion, 1999).

Tabel 2.1Pin USB

Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

Hubungan piranti masukan/keluaran dengan PC melalui USB dilakukan secara asimatrik yang berarti konektor pada kedua ujung kabel tidak sama sehingga harus diketahui ujung mana yang dipasang pada master dan ujung mana yang dipasang pada slave. Terminologi yang diadopsi oleh spesifikasi USB adalah *upstream* dan *downstream*.Ujung *upstream* mengatur protokol dan menginstrusikan

ujung yang ditentukan. Konektor tipe A dipasang pada PC sedang tipe B dipasang pada piranti masukan/keluaran.

2.2.2 Pengenalan pada USB

Peralatan USB bekerja secara*Plug & Play* (dipasang langsung bekerja), hal tersebut merupakan kelebihan dari komunikasi USB dibandingkan dengan komunikasi yang lainnya.Agar peralatan USB dapat *Plug & Play*, komputer harus melakukan pengenalan setiap saat kepada perangkat USB yang terpasang pada komputer. Jika pengenalan berhasil dilakukan maka komputerakan mengendalikan perangkat USB tersebut.

Setiap perangkat USB yang terpasang pada komputer memiliki perlakuan yang berbeda-beda, hal inilah yang menyebabkan komputer membutuhkan driver agar dapat mengontrol perangkat USB yang terpasang(Brey, 2002).

2.2.3 SYNC

SYNC (*Synchronization*) merupakan awal dari paket data USB. semua paket USB dimulai denga Sync. Untuk USB dengan kecepatan *low speed* dan *full speed* memiliki field 8 bit sedangkan untuk *high speed* memiliki fild 32 bit.Field Sync ini digunakan untuk mengsinkronisasi pulsa pada receiver (*Host* controller atau komputer) dengan pulsa pada transmitter (perangkat USB). Dua bit terakhir pada field Sync ini menunjukkan dimana field PID dimulai.

2.2.4 ENDPOINT

Field endpoint menunjukkan peralatan akhir USB yang dituju, yang Terdiri dari 4 bit sehingga memungkinkan terdapat 16 endpoint dengan Format ENDP.

Tabel 2.2 Format ENDP

ENDP0	ENDP1	ENDP2	ENDP3

Pada peralatan low speed hanya memiliki maksimal 4 endpoint.Endpoint dapatdideskripsikan sebagai sumber data.endpoint menunjukkan akhir salurankomunikasi dari USB.Semua peralatan harus dapat menerima kondisi endpoint zero, dimana endpointyang menerima

permintaan status dan kontrol perhitungan serta durasi darioutput yang dihasilkan selama peralatan beroperasi dalam sistem bus.

2.2.5 Paket Token

Terdapat tiga tipe dari paket token, yaitu yang pertama adalah sinyal **IN**, yangdigunakan untuk memberitahukan kepada peralatan USB bahwa *Host Controller* meminta untuk membaca informasi/data, jika peralatan USB tidakmemiliki data yang akan dikirim maka Host Controller akan menerima sinyalNAK. Kedua adalah sinyal **OUT** yang digunakan untuk memberitahukankepada peralatan USB bahwa Host Controller akan mengirim informasi/data.Data yang dikirimkan oleh Host Controller mengikuti sinyal OUT Tokentersebut, jika peralatan USB belum siap menerima data maka host controllerakan menerima sinyal NAK dan akan mengirim kembali sinyal OUT Token diikuti data ketika peralatan USB telah siap. Ketiga adalah sinyal **SETUP** yangdigunakan untuk mengendalikan transfer data pada transfer kontrol.

2.2.6Komunikasi Data USB

Komunikasi USB disebut juga master tunggal, dimana komputer (HC/*Host controller*) mengatur semua aktifitas komunikasi data dari USB. Sekitar 8 byte samapi 256 byte data yang dikirim melalui saluran USB yang dikemas dalam bentuk paket (paket data). Komputer meminta data ke peralatan USB dan peralatan USB harus memberi data ke komputer (HC/*Host controller*). Waktu yang dibutuhkan untuk pengiriman data sekitar 1 ms (satu mili detik) satu kali pengiriman.

Pengiriman data terjadi secara asinkron, oleh sebab itu peralatan USB yang terhubung dengan komputer (HC) harus menyesuaikan kecepatan dengan komputer untuk penerimaan data. Pengiriman data dengan kecepatan 1,5 Mbps (sekitar 666.7 ns untuk pengiriman 1 bit). Sedangkan untuk kecepatan 12 Mbps (sekitar 88,3 ns untuk pengiriman 1 bit) merupakan kecepatan *full speed*.

Setiap proses transaksi pada USB terdiri paket token/sinyal penanda (menjelaskan data yang mengikutinya) pilihan paket data (temasuk tingkat muatan) dan status paket (pemberitahuan hasil transaksi dan untuk koreksi kesalahan). Pengiriman data pada komunikasi USB (bus USB) berlangsung dengan cara mendahulukan data LSB.

Paket-paket USB terdiri dari data-data sync, Dua bit terakhir PID (*packet indentity*), ADDR dan ENDP Semua paket harus diawali dengan data sync. Sync dengan data 8 bit untuk low dan full speed sedangkauntuk data 32 bit untuk high speed yang digunakan untuk mensinkronkan antara *Host ontroller* dengan perangkat USB(Reznik dan Hershenhoren, 2010).

PID adalah singkatan dari Packet ID atau identitas paket, field ini digunakanuntuk mengidentifikasikan jenis paket yang akan dikirimkan, baik oleh USB*host controller* atau oleh peralatan USB. Field PID ini dibagi menjadi empatgroup jenis paket yaitu : Token, Data, Handshake, dan Spesial.

PID adalah field untuk menandakan tipe dari paket yang sedang dikirim. Terdapat 8 bit yang pada dasarnya terdapat 4 bit untuk penanda tipe dari paket yang dikirim. Sedangakan 4 bit yang lain hanya berfungsi sebagai pengecekan agar tidak ada kesalahan dalam pengiriman. Untuk formatnya sebagai berikut:

Tabel 2.3 Format Pengirimman PID

PID0	PID1	PID2	PID3	nPID0	nPID1	nPID2	nPID3

Tabel 2.4 Nilai PID

GROUP	Nilai PID	Paket dentifikasi
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	Setup Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Heandshake	0010	ACK Handshake
	1110	NAK Handshake
	0101	STALL Handshake
	0110	NYET(belum merespom)

Special	1100	PREamble
	1100	EER
	1000	Split
	0100	Ping

(Reznik dan Hershenhoren, 2010).

CRC (cyclic redundancy check) dijalankan pada data didalam paket yang dikirim. Pada paket penanda (token) mempunyai5 bit CRC sedangkan pada paket data mempunyai 16 bit CRC.

EOP (End Of Packet) merupakan akhir dari paket yang disinyalkan dengan satu angka akhir 0 (single Ended Zero/SEO) untuk sekitar 2 kali bit diikuti oleh sebuah J 1 kali. Data yang dikirim dalam bus USB adalah salah satu dari 4 bentuk yaitu control, interrupt, bulk, atau *isochronus*(Reznik dan Hershenhoren, 2010).

Untuk mengurangi noise, USB menggunakan sinyal diferensial. Pengiriman data menggunakan USB dilakukan secara half duplex, yang berarti pengiriman dilakukan secara bergantian.Kabel data USB tidak memiliki sinyal *clock*, sehingga komunikasi dengan USB dapat dikatakan sebagai asinkron. Sebuah bus keceepatan penuh memiliki kondisi D+ tinggi dan D- rendah kondisi ini biasanya dikatakan logika 1 dari bus, namun untuk mencacah kerancauan dan mempertahankan konsistensi dengan spesifikasi USB, kondisi ini kemudian disebut kondisi J. Kondisi yang lain disebut kondisi K yang dikatakan logika 0 dari bus yaitu saat D+ lebih rendah dan D- lebih tinggi.

Dalam acuan buku ditentukan persyaratan yang sangat ketat untuk kabel USB, tidak semua kabel bisa dipakai, khususnya untuk USB dengan kecepatan transfer data sampai 1.2 Mega bps (full speed). Sehingga kabel USB selalu dijual dalam bentuk sudah jadi, ujung yang satu terpasang konektor tipe A dan ujung satunya terpasang konektor tipe B, tidak ada yang menjual konektor USBsecara lepas. Pada komputer biasanya terdapat dua buah atau lebih terminal untuk konektor tipe A jadi dengan mudah bisa dipasang dua buah atau lebih peralatan USB(Reznik dan Hershenhoren, 2010).

Panjang kabel untuk USB dibatasi sampai maksimum lima meter untuk antarmuka kecepatan penuh (*full speed*) dan maksimum tiga meter untuk antarmuka kecepatan rendah (*low speed*). Daya maksimum yang tersedia melalui kabel-kabel USB adalah 100 mA arus maksimum pada 5 V (Brey, 2002).

2.3 HID

HID merupaka singkatan dari *Human Interface* Device. USB HID sendiri merupakan sebuah *class* perangkat USB yang menggambarkan perangkat antarmuka seperti keyboard, mouse, game kontroler, dll. Intinya USB *class*HID *class*mendefinisikan perangkat atau device yang digunaka hampir disetiapkomputer modern(Axelsion, 1999).

Sebagian besar *operating system* (OS) akan mengenali standar USB HID suatu perangkat seperti keyboard dan mouse tanpa perlu driver khusus.Sebagai perbandingan, pesan ini biasanya tidak muncul untuk perangkat yang terhubung melalui pin 6-PS/2 (Anonimous, 2001).

Protokol HID ada 2 identitas yaitu *Host* (tuan) dan *Device* (perangkat). Device (Perangkat) adalah identitas yang secara langsung berinteraksi dengan manusia, seperti keyboard atau mouse. Host berkomunikasi dengan perangkat dan menerima data masukan dari perangkat pada tindakan yang dilakukan oleh manusia. Output data mengalir dari host ke perangkat dan kemudian ke manusia.

Tuan rumah (*Host Controller*) perlu untuk mengambil descriptor HID dari perangkat dan mengurangi sebelum sepenuhnya dapat berkomuniksasi dengan perangkat(Anonimous, 2001).

Keyboard setiap 256 kode kunci pertama (penggunaan) dedefinisikan dalam table penggunaan HID, penggunaan halaman 7 dapat dilaporkan oleh keyboard menggunakan protokol boot, tetapi kebanyakan sistim hanya menangani subset dari kunci-kunci(Anonimous, 2001).

2.4 Counter

Counter merupakan perangkat yang tersusun atas lebih dari satu rangkaian flip-flop dengan kombinasi ataupun tanpa kombinasi dari rangkaian-rangkaian logika. Fisik dari counter merupakan sekumpulan rangkaian – rangkaian logika *sequensial* yang dapat berinteraksi menjadi sebuah IC.

Meskipun fungsi utama dari *counter* adalah untuk aplikasi pencacahan, *counter* juga difungsikan untuk mengukur interval waktu antara dua kejadian yang terjadi, dan dapat juga digunakan untuk mencacah frekuensi suatu sinyal.

Ada dua macam tipe counter yaitu *asynchronous counter* dan *synchronous counter*. *Asynchronous counter* merupakan konter yang terdiri dari susunana rangkaian flip-flop yang keluaran satu sama lain

mengendalikan *clock*keluaran dari flip-flop berikutnya. Konsekuensi ilmiah dari susunan ini akan menyebabkan perubahan status dari sejumlah n flip-flop tidak semuanya terjadi secara serentak. Flip-flop urutan ke-2 mengalami perubahan status setelah jeda waktu tertentu terhadap perubahan status pada *clock input*. Begitu juga pada flip-flop berikutnya akan mengalamiperubahan status setelah kelipatan n jeda waktu antara perubahan status pada setiap flip-flop. Jeda waktu ini akan menjalar dan bertambah besar jika urutan flip-flop bertambah jauh.

Sedangkan *synchronous counter* semua flip-flop yang ada didalamnya mengalami perubahan status secara serentak terhadap perubahan status *clock input*.

2.5 Frekuensi

Frekuensi beresal dari bahasa inggris *frequency* yang diartikan kedalam bahasaIndonesia menjadi keseringan. Frekuensi mengandung arti suatu bilangan (angka0 yang menunjukan beberapa kalikah suatu variabel (yang dilambangkan dengan angka-angka itu) berulang dalam deretan angka tersebut, atau beberapa kalikah suatu variable (yang dilambangkan dengan angka itu) muncul dalam deratan angka tersebut.

BAB III
METODOLOGI PENELITIAN

3.1 Tempat dan Waktu penelitian

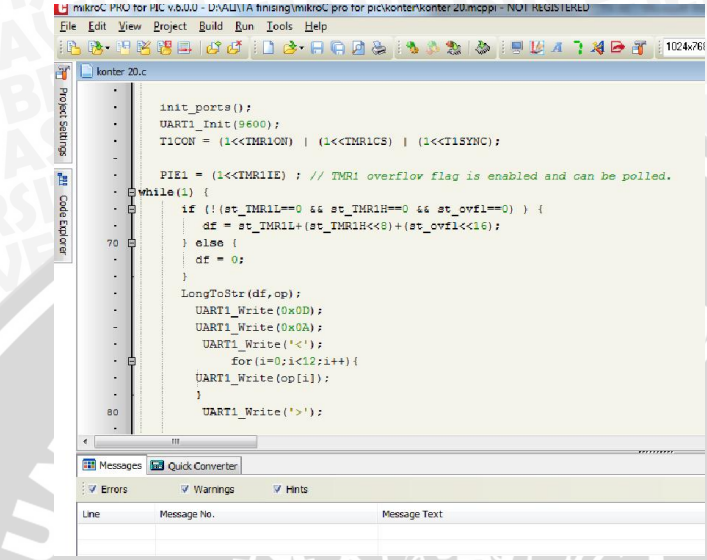
Penelitian ini dilaksanakan di laboratorium Elektronika dan Instrumentasi jurusan Fisika Universitas Brawijaya pada tanggal 20 Juni 2012 sampai dengan 3 Januari 2013.

3.2 Alat dan Bahan

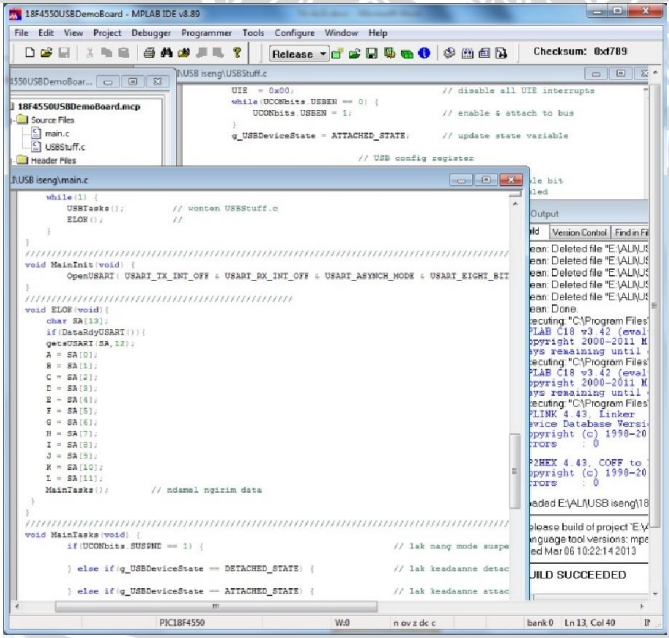
Penelitian ini membutuhkan perangkat keras dan perangkat lunak, antara lain adalah, timah, solder, PCB komponen listrik baik komponen aktif maupun yang pasif, dan alat-alat bantu elektronika yang lainnya.

Untuk mendownload program ke IC mikrokontroler digunakan PICkit3 yang terlihat pada gambar 3.1. Sedangkan untuk software pada penelitian ini digunakan 2 macam software yaitu MPLAB (gambar 3.3) yang digunakan untuk membuat program komunikasi USB sedangkan MikroC Pro For Pic (gambar 3.2) digunakan untuk membuat program pencacah.

Gambar 3.1PICkit 3



Gambar 3.2MikroC Pro For PIC



Gambar 3.2MPLAB

3.3Tahap Penelitian

Tahap penelitian in terdiri dari beberapa tahapan seperti terlihat pada diagram blok dibawah ini.

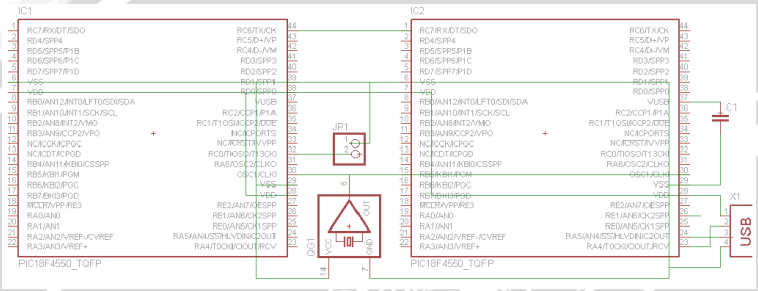
Gambar 3.3 Tahap Penelitian

3.3.1 Persiapan

Sebelum melakukan penelitian diperlukan persiapan terlebih dahulu. Pertama mempersiapkanrefrensi penelitian dan kedua mempersiapkanketersediaan alat-alat dan bahan yang diperlukan. Persiapanreferensi penelitian dilakukan dengan mencari literatur. Dari literatur tersebut didapatkan pemahaman tentang landasan teori yang digunakan alat-alat dan bahan yang diperlukan dalam penelitian. Literature utama dalam penelitian ini bersumber dari datasheet

PIC18F4550 karena fokus kajian ini terpusat pada mikrokontroler PIC18F4550. Bahasa yang digunakan dalam pemrograman ini menggunakan bahasa C baik di dalam MPLAB maupun dalam mikroCPRO for PIC.

3.3.2 Perakitan Hardware



Gambar 3.4Skematik

Dalam perancangan hardware agar terbentuk dengan bagus maka terlebih dahulu membuat rangkaian dengan membuat skematiknya terlebih dahulu, yang selanjutnya dicetak menjadiPCB.

Dalam perakitan *hardware* (sistim minimum) percobaan ini dibutuhkan beberapa komponen seperti PIC18F4550, PCB, soket, osilator, kapasitor, resistor, konektor USB, soket dan beberapa hider.Rangakaian sistim minimum mengikuti pola yang sesuai pada gambar 3.5.Pembuatan PCB dapat dilakukan dengan pemesanan di tokoelektronik yang menyediakan jasa pembuatan PCB.

Osilator yang dipasang hanya satu yang dapat memberi masukan berupa clock pada kedua mikrokontroler dengan cara diparallel. Konector USB digunakan sebagai sumber daya dan komunikasi dengan PC (komputer).

3.3.3 Pembuatan Program

Dalam pembuatanprogram diperlukan *software* khusus yang terdiri dari dua macam software yaitu mikroCPRO for PIC dan MPLAB. Software mikroC PRO for PIC digunakan untuk membuat program pencacah karena penggunaan software ini sangat mudah. MPLAB di gunakan untuk memasukan program yang sudah berbentuk HEX yang sebelumnya diprogram menggunakan mikroC PRO for PIC.Selain itu

MPLAB juga digunakan untuk membuat program USB yang berbasis HID.MPLAB dipilih karena penggunaannya gratis.

Rancangan untuk pencacah frekuensidalam penelitian ini menggunakan sistem yang sama dengan yang sudah dikembangkan sebelumnya. Desain sistem pada (pencacah frequensi resolusi 0.1 ppm Menggunakan IC tunggal mikrokontroler) menggunakan PIC16F877A, namun pada penelitian ini diimplementasikan dalam PIC18F4550. Hasil pencacahan selanjutnya dikirimkan secara serial ke mikrokontroller PIC18F4550 ke dua yang berperan sebagai piranti komunikasi ke komputer melalui komunikasi USB(Sakti, 2012).

3.3.4 Pengujian Alat

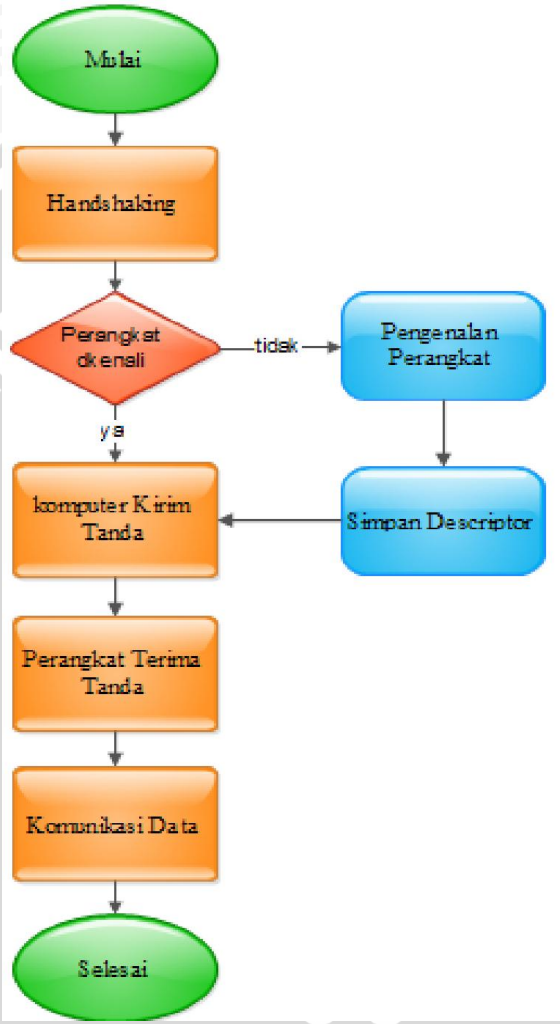
Pengujian alat dilakukan setelah programdimasukkan kedalam mikrokontroler. Yang diuji pertama kali adalah program pencacah, dengan menggunakan komunikasi serial terlebih dahulu yang ditampilkan melalui USART terminal. Pengujian program berikutnya untuk komunikasi USB, dimana komunikasi USB harus mengirimkan data yang diberikan pada mikrokontroler pencacah ke PC (penampilan yang sudah tersedia pada mikroCPRO PIC).Setelah semua selesai, yang terakhir menyambungkan masukan pencacah ke signal generator dan nilai frekuensi diubah-ubah pada signal generator sampai sesuai yang dinginkan. Jika peralatan sudah benar, maka alat dikalibrasi dengan menggunakan osilator yang keluarannya 10MHz.

3.4 Diagram Blok

Gambar 3.5 Diagram blok pengiriman data

Mikrokontroler yang digunakan ada 2, yang pertama untuk mencacah data sekaligus mengolah data atau mengubah data dari biner ke ASCII dan mengirim data berupa ASCII ke mikrokontroler USB melalui komunikasi serial (*Transmite* pada mikrokontroler pencacah dan *Receive*pada mikrokontroler USB).Mikrokontroler USB mengirimkan data yang diterima dari mikrokontroler pencacah ke PC (komputer).

3.5 Komunikasi USB-HID



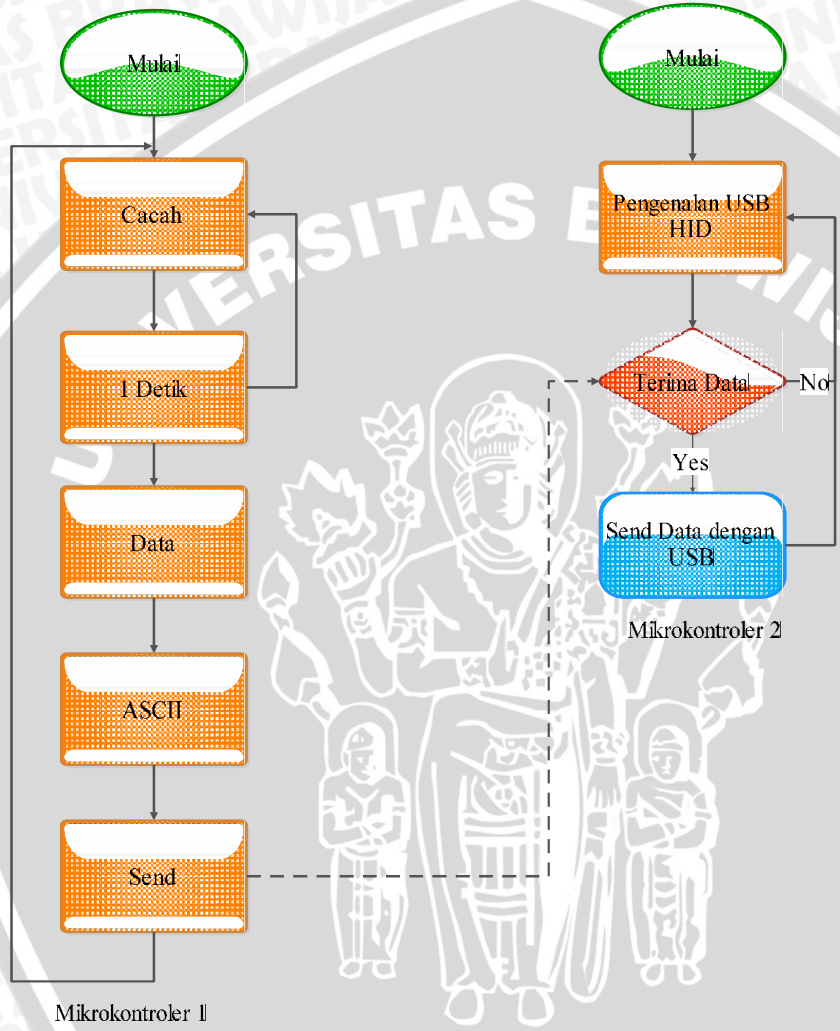
Gambar 3.6 Flowchart USB-HID

Komunikasi USB kelas HID dimulai dari komputer dengan perangkat melalui*handshaking*.Komputer memberikan tanda pada perangkat USB Ketika peralatan USB disambungkan dengan komputer. Jika komputer tidak mengenali suatu perangkat USB maka perangkat

USB melakukan pengenalan dengan komputer, menyimpan descriptor perangkat USB ke dalam memori komputer.

Setelah pengenalan selesai, komputer mengirim tanda (token) ke perangkat dan perangkat menerima tanda untuk komunikasi.Dalam komunikasi, perangkat USB maupun komputer dapat mengirim dan menerima data secara bergantian sampai perangkat USB diputus dengan komputer (dicabut dari komputer).

3.6Flowchart



Gambar 3.7 Flowchart Program Mikrokontroler

Mikrokontroler 1 mencacah sinyal frekuensi selama satu detik, data yang dicacah disimpan didalam register (register low, high, dan

overflow) dalam bentuk biner. Pada mikrokontroler PIC18F4550 hanya tersedia 16 bit timer yang terdiri 8 bit low dan 8 bit high. Untuk counter overflow menggunakan register tambahan sebesar 32 bit. Jika timer high overflow maka counter overflow akan bertambah nilai padanya.

Pencacahan dikerjakan mikrokontroler 1 selama satu detik, setelah satu detik register low, high, dan overflow dinolkan (clear). Data yang tersimpan pada register yang masih bentuk biner diubah (konversi) dalam bentuk ASCII.Data yang sudah menjadi ASCII dikirimkan melalui komunikasi serial (pin Rx) ke mikrokontroler 2 (pin Tx).

Mikrokontroler 2 melakukan pengenalan terhadap *Host Controler* (HC), dan mengirimkan data penanda agar komunikasi mikrokontroler 2 dengan PC tidak terputus.Mikrokontroler menerima data dalam bentuk ASCII dari mikrokontroler 1 setiap satu detik. Data yang diterima disimpan dalam register. Data yang disimpan dalam mikrokontroler 2 dikirimkan ke *host controller* (komputer) sebanyak satu kali. Hal tersebut akan mengakibatkan mikrokontroler 2 mengirimkan data setiap satu detik.

3.7Pengambilan Data

Setelah semua tahapan sebelumnyaberhasil dilakukan, selanjutnya mengamati hasil sekaligus mengambil data. Data yang diambil adalah data hasil pembacaan alat pencacah frekuensi yang dibuat dan nilai dari signal generator.

Halaman ini sengaja dikosongkan

BAB IV
HASIL DAN PEMBAHASAN

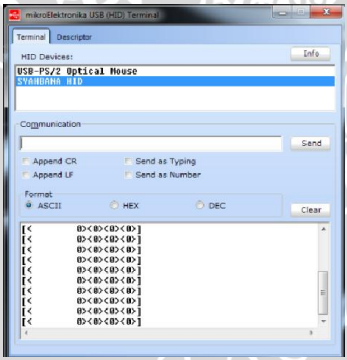
4.1 Hasil

Perangkat pencacah frekuensi menggunakan komunikasi USB tampak pada gambar 4.1. Perangkat ini mencacah frekuensi dan tada cacahan dikirim ke komputer melalui komunikasi USB kelas HID.

Gambar 4.1 Perangkat USB Pencacah Frekuensi

4.1.1 Pengujian komunikasi USB

Pengujian komunikasi USB, dilakukan dengan mengirimkan karakter dari mikrokontroler ke PC.Karakter yang dikirim ditampilkan menggunakan mikroElektronika USB (HID) Terminal yang terlihat pada gambar 4.2.Gambar tersebut menunjukan beberapa karakter.



Gambar 4.2HID Terminal

4.1.2 Pengujian Pencacah Frekuensi

Bagian ini akan membahas tentang pengujian alat dan nilai frekuensi yang terukur oleh alat pada masing-masing rentang frekuensi sumber yang sudah ditentukan. Pengujian ini menggunakan sinyal

masukan dari signal generator yang bentuk keluarannya berupa sinyal TTL dan sinus.

Tabel 4.1 Nilai Cacahan 10-90 Hz

Signal Generator (Hz)	Frekuensi Counter (Hz)
10	9/10
20	19/20
30	29/30
40	39/40
50	49/50
60	59/60
70	69
80	79/80
90	89/90

Tabel 4.2 Nilai Cacahan 100-900 Hz

Signal Generator (Hz)	Frekuensi Counter (Hz)
100	99/100
200	199/200
300	299/300
400	399/400
500	499/500
600	599/600
700	699/700
800	799/800
900	899/900

Tabel 4.3 Nilai Cacahan 1- 9 KHz

Signal Generator (Hz)	Frekuensi Counter (Hz)
1000	999/1000
2000	2000
3000	3000/3001
4000	4000/4001
5000	5000/5001

6000	6000/6001
7000	7001
8000	8001/8002
9000	9001/9002

Tabel 4.4 Nilai Cacahan 10-90 KHz

Signal Generator (Hz)	Frekuensi Counter (Hz)
10000	10001/10002
20000	20004/20005
30000	30006/30007
40000	40008/40009
50000	50011/50012
60000	60013/60014
70000	70015/70016
80000	80018/80019
90000	90021

Tabel 4.5 Nilai Cacahan 100 - 900 KHz

Signal Generator (Hz)	Frekuensi Counter (Hz)
100000	100022/100023
200000	200046/20047
300000	300069/30007
400000	400093/40001
500000	500117
600000	600140/600141
700000	700164/700165
800000	800188/800189
900000	900212/900213

Pencacahan frekuensi dilakukan selama satu detik. Pemrograman untuk menghasilkan satu detik tidak menggunakan fungsi delay atau timer, tetapi memanfaatkancara kerja dari mikrokontroler PIC18F4550, hal tersebut dikarenakan lebih mendekati satu detik dengan menggunakan perumusan dibawah ini.

$$1\ siklus = \frac{4}{osilator}$$

$$1\ siklus = \frac{4}{20MHz} = 200nS$$

$$\frac{1\ S}{2nS} = 5 \times 10^9 siklus$$

Untuk mendapatkan hasil satu detik digunakan penambahannilai siklus sampai 1 detik, melalui program yang jumlah siklusnya mencapai 5×10^9 siklus.

Nilai yang tercacah mengalami perubahan pada digit yang terakhir, hal ini dikarenakan osilator memberikan masukan *clock* tidak tepat 1 detik, tetapi memiliki selisih orde nano detik.

4.2 Pembahasan

Komunikasi USB-HIDmemiliki keunggulan dibandingkan dengan komunikasi USB yang lain seperti halnya menggunakan IC FTDI.

FTDI merupakan IC yang dapat berkomunikasi menggunakan USB akan tetapi jika menggunakan FTDI harus memerlukan software tambahan agar dapat terdeteksi oleh PC, haltersebut dikarenakan FTDI tidak menggunakan komunikasi kelas HID. Cara kerja FTDIadalah mengubah komunikasi serial ke komunikasi USB akan tetapi data yang dibaca komputer berupa data serial. Untuk masukan yang diterima FTDI hanya bisa satu tidak bisa lebih. Berbeda denganalat ini, yang dapat langsung disambungkan ke PC dan tidak membutuhkan software tambahan, karena sudah mendukung komunikasi USB yang berbasis HID dimana setiap PC pada saat ini sudah tersedia komunikasi USB kelas HID, juga memiliki masukan pin lebih dari satu.

4.2.1 Protokol USB

Gambar 4.3 Flowchart USB

Host Controller (CPU/komputer) mengirimkan token ke perangkat USB. Mikrokontroler (USB Device) menunggu paket token yang dikirim oleh *Host Controller*(HC) yang berupa PID 8 bit, ADDR 7 bit, ENDP 4 bit, dan CRC5 5 bit.

Data yang diterima oleh Mikrokontroler (USB device) berupa PID, ADDR, dan CRC5. Bit PID yang dikirim bernilai 1001, yang berarti *identifikasi paket***IN Token**. **IN Token** menandakan mikrokontroler (USB Device) diperintah *Host* Controller mengirimkan paket data untuk dibaca (HC / komputer) dan mengirimkan nilai nol SE0 (*single ended zero*) untuk memberi tanda kalau pengiriman data berakhir.

Mikrokontroler (USB Device) mengirimkan paket data kekomputer (HC)jika telah menerima PID yang nilainya 1001 (*IN Token*). Format Pengiriman data mikrokontroler ke komputer sebagai berikut:

Tabel 4.6 Pengiriman Data USB

Field	PID	DATA	CRC16
Bit	8	192	16

Setelah pengiriman token (penanda) sekaligus diikuti bit data yang jumlahnya 24 Byte (192 bit) yang merupakan membentuk suatu string, data yang dikirim akan ditampilkan komputer (HC) dengan mengunakan interface yang sudah tersedia. Agar tidak ada kesalahan dalam pengirimanmaka data dicek oleh CRC16.

Jika pengiriman data dari mikrokontroler (USB Device) terjadi *eror*, maka HC (PC) akan mengirimkan sinyal NAK mikrokontroler yang menandakan mikrokontroler mengirim data kembali.

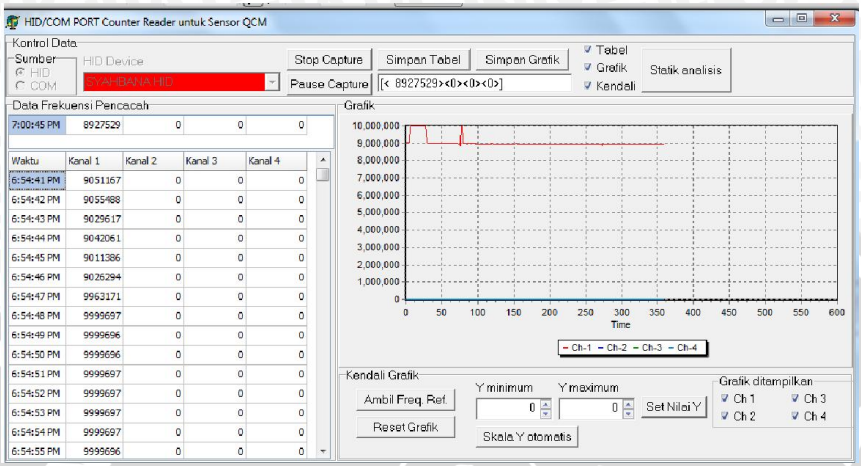
Data akan diterima oleh HC (CPU) dan HC akan mengirimkan sinyal ACK ke mikrokontroleryang menandakan bahwa data sudah diterima oleh *Host* (komputer).

4.2.2Pengiriman Data

Pengiriman data yang dikirimkan ke PC yang melalui USB dilakukan dalam bentuk start bit dan diakhiri dengan stop bit, dan setiap data yang akan dikirim ditempatkan dalam host yang berada dalam USB, satu paket host memiliki lebar data 64 byet. Berapapun data yang dikirimkan USB akan mengirimkan 64 byet dalam satu paket pengiriman.

4.3 Penampilan Display

Penampilan data menggunakan HID/COM PORT Counter Reader. Nilai yang ditampilkan sudah berupa ASCII yang terlihat pada gambar 4.4 dibawah.



Gambar 4.4Terminal USB HID

Halaman ini sengaja dikosongkan

BAB 5
KESIMPULAN

5.1 Kesimpulan

Penelitian tentang pemanfaatan PIC18F4550 sebagai antar muka komunikasi USB untuk pencacah frekuensi dapat disimpulkan sebagai berikut :

1. Perangkat keras pencacah frekuensi dengan menggunakan komunikasi USB untuk pengiriman data ke komputersudah selesai dibuat.
2. Nilai cacahan frekuensi dapat dikirim ke komputer dengan menggunakan protokol USB-HID.
3. Pengiriman data ke komputer dengan menggunakan komunikasi USB kelas HID dapat diterima oleh komputer dengan baik.
4. Nilai cacahan frekuensi pencacah dikirim ke komputer melalui komunikasi USB kelas HID.

5.2 Saran

Untuk penelitian berikutnya datadiharapkan dapat ditampilkan pada HP android.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

Anonimous. 2001. Device Class Definition for Human Interface Device (HID).

Axelsion, J. 1999. USB Complete. 3, Lakeview.

Brey, B. B. 2002. Mikroprosesor Intel. Erlangga. jakarta.

Loncaric, C., Y. Tang, C. Ho, M. A. Parameswaran dan H.-Z. Yu. 2011. AUSB-based electrochemical biosensor prototype for point-of-care diagnosis.

Microchip. 2006. PIC18F2455/2550/4455/4550 Data Sheet. USA, Microchip Technology Incorporated.

Niu, J. 2012. Design of USB-CAN Controller Based on PIC18F4580.

Reznik, O. dan R. Hershenhoren. 2010. USB2.0 Protocol Engine.

Sakti, S. P. 2012. Pencacah Frekuensi Resolusi 0.1ppm Menggunakan IC tunggal mikrokontroler. *Proseding Seminar Nasional Fisika Terapan III*.

Steiner, C. 2005. The 8051/8052 Microcontroller Architecture, Assembly Language, and Hardware Interface Boca Raton. USA.

Halaman ini sengaja dikosongkan

Lampiran

Lampiran 1Sourcode Pencacah Frekuensi

```
/*********************************************************/
School of Physics
University of Brawijaya
Written by        : 1. Dr. Ing. Setyawan P.S., M.Eng
                    2.Muh. Ali Syahbana
ID                : 0810930049
Last Modified     : 25 Desember 2012
File nime         : Pencacah Konter.C
 /*********************************************************/
unsigned long  ldmy=0;
unsigned short sdmy=0;

void init_ports(void) {
  ADCON1 = 0x06; // output kanggo digital
  PORTA = 0;
  TRISA = 0xFC;
  PORTB = 0;
  TRISB = 0;
}

unsigned long gerbang_1_detik(unsigned int jml) org 180 {
unsigned long  ALI = 0;
do {
jml--;
sdmy = PIR1 & (1<<TMR1IF);
asm {
nop ;
        }
if (sdmy!=0) {          // lak overflow
        PIR1 &= ~(1<<TMR1IF);  // ngilangi nilai Low Counter
        ALI += 1;              // nambah nilaine High Counter
      } else {
sdmy  &= ~(1<<TMR1IF);
ldmy += 1;
      }
Delay_us(990);   // ngepasno teko perhitungan
asm {
```

```
nop
nop
nop
nop
nop
nop
   }
    } while (jml>0);
asm {                          // kanggo ngepasno sak detik
nop
nop
nop
nop
nop
nop
   }

return ALI;
}

void main() org 500 {
char op[12];                // kango ngrubah kode ASCII
unsigned short st_TMR1L;     // kanggo nyimpen 1 low byte
unsigned short st_TMR1H;     // kanggo nyempen konter high
unsigned long  st_ovfl= 0;   // kanggo nyimpen lak overflow
unsigned long  df = 0;
unsigned short i;
unsigned short dx = 0;

char lcdop[4];

init_ports();
  UART1_Init(9600);
  T1CON = (1<<TMR1ON) | (1<<TMR1CS) | (1<<T1SYNC);

PIE1 = (1<<TMR1IE) ;     // TMR1 overflow flag is enabled and can be polled.
while(1) {
if (!(st_TMR1L==0 && st_TMR1H==0 && st_ovfl==0) ) {
df = st_TMR1L+(st_TMR1H<<8)+(st_ovfl<<16);
    } else {
df = 0;
    }
```

```
LongToStr(df,op);
      UART1_Write(0x0D);
      UART1_Write(0x0A);
       UART1_Write('<');
for(i=0;i<12;i++){
      UART1_Write(op[i]);
      }
       UART1_Write('>');

    st_TMR1L    = 0;
    st_TMR1H    = 0;
    st_ovfl     = 0;

    TMR1H = 0;                  // clear Timer 1 High
PIR1  &= ~(1<<TMR1IF);       // clear overflow bit.
    TMR1L = 0;                  // clear timer1 low count.

    st_ovfl = gerbang_1_detik(1000);

    st_TMR1L = TMR1L;
    st_TMR1H = TMR1H;

if ( PIR1 & (1<<TMR1IF) ) {
       PIR1     &= ~(1<<TMR1IF);
       st_ovfl += 1;         // munggahno nilai counter overflow
    }
  }
}
```

Halaman ini sengaja dikosongkan

Lampiran 2Sourcode Komunikasi USB HID

```
/*******************************************************/
School of Physics
University of Brawijaya
Written by      : Muh. Ali Syahbana
ID              : 0810930049
Last Modified   : 28 Januari 2013
File nime       : Komunikasi USB HID.C
/*******************************************************/
// includes//////////////////////////////////////////////////////////////////
#include<p18f4550.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include<math.h>
#include<timers.h>
#include<delays.h>
#include<usart.h>
#include"USBStuff.h"

//  config ///////////////////////////////////////////////////////////////////

#pragma config PLLDIV = 5
#pragma config CPUDIV = OSC1_PLL2
#pragma config USBDIV = 2
#pragma config FOSC = HS
#pragma config FCMEN = OFF
#pragma config IESO = OFF
#pragma config PWRT = OFF
#pragma config BOR = OFF
#pragma config BORV = 3
#pragma config VREGEN = ON
#pragma config WDT = OFF
#pragma config WDTPS = 32768
#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config LPT1OSC = ON
#pragma config MCLRE = OFF
#pragma config STVREN = ON
```

```
#pragma config LVP = OFF
#pragma config ICPRT = OFF
#pragma config XINST = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF
#pragma config CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF,WRT3 = OFF
#pragma config WRTC = OFF
#pragma config WRTB = OFF
#pragma config WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF
#pragma config EBTRB = OFF

// panggone variables ////////////////////////////////////////////////////////////////////////
#pragma udata
extern volatile BDT_ENTRY g_buffDescTable[4];
extern BYTE g_USBDeviceState;
extern BYTE g_fromHostToDeviceBuffer[65];
extern BYTE g_fromDeviceToHostBuffer[65];
long int A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,SA,DA;
// panggone prototypes ///////////////////////////////////////////////////////////////////////
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();

void MainInit(void);
void MainTasks(void);
void ELOK(void);

#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
0x1008
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
0x1018

extern void _startup(void);

////////////////////////////////////////////////////////////////////////////////////////////
#pragma code REMAPPED_RESET_VECTOR = 0x1000
void _reset(void) {
      _asm goto _startup _endasm
}
////////////////////////////////////////////////////////////////////////////////////////////
```

```
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR =
REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR(void) {
      _asm goto YourHighPriorityISRCode _endasm
}
//////////////////////////////////////////////////////////////////////////////////////////
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR =
REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR(void) {
      _asm goto YourLowPriorityISRCode _endasm
}

//////////////////////////////////////////////////////////////////////////////////////////
#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR(void) {
      _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
_endasm
}

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

//////////////////////////////////////////////////////////////////////////////////////////
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode() {
}

//////////////////////////////////////////////////////////////////////////////////////////
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode() {
}

#pragma code
//////////////////////////////////////////////////////////////////////////////////////////
void main(void) {
        USBInit();              // kanggo komunikasine USB
        MainInit();             // inisialisasi USART ae
        while(1) {
```

```
                USBTasks();         // wonten USBStuff.c
                ELOK();
        }
}
////////////////////////////////////////////////////////////////////////////////////////////
void MainInit(void) {
                OpenUSART( USART_TX_INT_OFF &
USART_RX_INT_OFF & USART_ASYNCH_MODE &
USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH, 129
);
}
////////////////////////////////////////////////
void ELOK(void){
        char SA[13];
        if(DataRdyUSART()){
        getsUSART(SA,12);
        A = SA[0];
        B = SA[1];
        C = SA[2];
        D = SA[3];
        E = SA[4];
        F = SA[5];
        G = SA[6];
        H = SA[7];
        I = SA[8];
        J = SA[9];
        K = SA[10];
        L = SA[11];
        MainTasks();            // ndamel ngirim data
         }
}
////////////////////////////////////////////////////////////////////////////////////////////
void MainTasks(void) {
if(UCONbits.SUSPND == 1) {  // lak nang mode suspend

} else if(g_USBDeviceState == DETACHED_STATE) {
} else if(g_USBDeviceState == ATTACHED_STATE) {
} else if(g_USBDeviceState == POWERED_STATE) {
} else if(g_USBDeviceState == DEFAULT_STATE) {
} else if(g_USBDeviceState == ADDRESS_STATE) {
} else if(g_USBDeviceState == CONFIGURED_STATE) {
```

```
                g_fromDeviceToHostBuffer[1] = 0X0D;
                g_fromDeviceToHostBuffer[2] = 0X0A;
                g_fromDeviceToHostBuffer[3] = A;
                g_fromDeviceToHostBuffer[4] = B;
                g_fromDeviceToHostBuffer[5] = C;
                g_fromDeviceToHostBuffer[6] = D;
                g_fromDeviceToHostBuffer[7] = E;
                g_fromDeviceToHostBuffer[8] = F;
                g_fromDeviceToHostBuffer[9] = G;
                g_fromDeviceToHostBuffer[10] = H;
                g_fromDeviceToHostBuffer[11] = I;
                g_fromDeviceToHostBuffer[12] = J;
                g_fromDeviceToHostBuffer[13] = K;
                g_fromDeviceToHostBuffer[14] = L;

transferFromDeviceToHostViaEP1((BYTE*)&g_fromDeviceToHostBuffer[1],
64);            // ngirim paket nang host

        }
}
```

```
// USBStuff.c
// includes ////////////////////////////////////////////////////////////////////////////////
#include<p18f4550.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include"USBStuff.h"

// global variables
//////////////////////////////////////////////////////////////////////////////
#pragma udata USB_BDT = 0x400
volatile BDT_ENTRY g_buffDescTable[4];

volatile CTRL_TRF_SETUP g_ctrlTrfSetupPkt;
volatile BYTE g_ctrlTrfData[USB_EP0_BUFF_SIZE];

#pragma udata

#pragma udata USB_VARS
BYTE g_fromHostToDeviceBuffer[65];
BYTE g_fromDeviceToHostBuffer[65];

#pragma udata
OUT_PIPE g_outPipe;
IN_PIPE  g_inPipe;

BYTE g_USBDeviceState;
BYTE g_controlTransferState;
BYTE g_USBActiveConfiguration;
BYTE g_shortPacketStatus;


#pragma romdata
// device descriptor
rom USB_DEVICE_DESCRIPTOR g_userDeviceDescriptor = {0x12,
                                    USB_DESCRIPTOR_DEVICE,
                                    0x0200,
```

```
                                0x00,
                                0x00,
                                0x00,


        USB_EP0_BUFF_SIZE,
                                0x04D8,
                                0x003F,
                                0x0002,
                                0x01,
                                0x02,
                                0x00,
                                0x01    };
// config 1 descriptor
rom BYTE g_configDescriptor1[] = {  0x09,

        USB_DESCRIPTOR_CONFIGURATION,
                                0x29,
                                0x00,
                                1,
                                1,
                                0,
                                _DEFAULT | _SELF,
                                50,
                                0x09,

        USB_DESCRIPTOR_INTERFACE,
                                0,
                                0,
                                2,
                                HID_INTF,
                                0,
                                0,
                                0,
                                0x09,   // HID descriptor
                                HID_DESCRIPTOR,
                                0x11,
                                0x01,
```

```
                                    0x00,
                                    HID_NUM_OF_DSC,
                                    REPORT_DESCRIPTOR,
                                    HID_RPT01_SIZE, 0x00,

                                    0x07,    // endpoint descriptor

        USB_DESCRIPTOR_ENDPOINT,
                                    1 | _EP_IN,
                                    _INT,
                                    0x40,
                                    0x00,
                                    0x01,

                                    0x07,    // endpoint descriptor

        USB_DESCRIPTOR_ENDPOINT,
                                    1 | _EP_OUT,
                                    _INT,
                                    0x40,
                                    0x00,
                                    0x01     };

//kode bahasa string descriptor
rom struct{
BYTE bLength; BYTE bDscType; WORD string[1];}
g_stringDescript000 = {
sizeof(g_stringDescript000), USB_DESCRIPTOR_STRING, { 0x0409
} };

// mfg. string descriptor
rom struct{BYTE bLength; BYTE bDscType; WORD string[4];}
g_stringDescript001 = {sizeof(g_stringDescript001),
                       USB_DESCRIPTOR_STRING,
        {'E','L','O','K'} };
```

```
// produk string descriptor
rom struct{BYTE bLength; BYTE bDscType; WORD string[12];}
g_stringDescript002 = {sizeof(g_stringDescript002),
        USB_DESCRIPTOR_STRING,
        {'S','Y','A','H','B','A','N','A',' ','H','I','D'} };

rom struct{BYTE report[HID_RPT01_SIZE];}
g_HIDReport01 = { {    0x06, 0x00, 0xFF,
                                    0x09, 0x01,
                       0xA1, 0x01,
                       0x19, 0x01,
                       0x29, 0x40,
                       0x15, 0x00,
                       0x26, 0xFF, 0x00,
                       0x75, 0x08,
                       0x95, 0x40,
                       0x81, 0x02,
                       0x19, 0x01,
                       0x29, 0x40,
                       0x91, 0x02,
                       0xC0 } };

// array teko config descriptors
rom BYTE *rom g_userConfigDescriptorPtr[] = {
(rom BYTE *rom)&g_configDescriptor1 };

// array teko string descriptors
rom BYTE *rom g_USBStringDescPtr[] = {
        (rom BYTE *rom)&g_stringDescript000,
        (rom BYTE *rom)&g_stringDescript001,
        (rom BYTE *rom)&g_stringDescript002 };

#pragma udata
#pragma code
////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void USBInit(void) {
        BYTE i;

        UEIR = 0x00;
        UIR = 0x00;

        UEIEbits.BTSEE   = 1;
        UEIEbits.BTOEE   = 1;
        UEIEbits.DFN8EE  = 1;
        UEIEbits.CRC16EE = 1;
        UEIEbits.CRC5EE  = 1;
        UEIEbits.PIDEE   = 1;

                // UIE -> USB interrupt enable register
        UIEbits.SOFIE   = 1;
        UIEbits.STALLIE = 1;
        UIEbits.IDLEIE  = 1;
        UIEbits.TRNIE   = 1;
        UIEbits.ACTVIE  = 0;
        UIEbits.UERRIE  = 1;
        UIEbits.URSTIE  = 1;

        UCONbits.PPBRST = 1;
        UCONbits.PPBRST = 0;
        UADDR = 0x00;

for(i=0; i<(sizeof(g_buffDescTable) / sizeof(BDT_ENTRY)); i++) {

        g_buffDescTable[i].STAT.STATVal = 0x00;
        g_buffDescTable[i].CNT = 0x00;
        g_buffDescTable[i].ADR = 0x0000;
        }

                // USB endpoint 0 configuration
        UEP0bits.EPHSHK   = 1;          // endpoint 0 handshake aktif
        UEP0bits.EPCONDIS = 0;
        UEP0bits.EPOUTEN  = 1;          // endpoint 0 output aktif
        UEP0bits.EPINEN   = 1;          // endpoint 0 input aktif
```

```
        UEP0bits.EPSTALL  = 0;

        while(UIRbits.TRNIF == 1) {
        UIRbits.TRNIF = 0;
                                }

        g_inPipe.busy = 0;
        g_outPipe.busy = 0;
        g_outPipe.wCount = 0;

        UCONbits.PKTDIS = 0;

        g_USBActiveConfiguration = 0;
        g_USBDeviceState = DETACHED_STATE;

        for(i=0;i<65;i++) {
                g_fromHostToDeviceBuffer[i] = 0x00;
                g_fromDeviceToHostBuffer[i] = 0x00;
        }
}

////////////////////////////////////////////////////////////////////////////////
void USBTasks(void) {
        BYTE i;

if(g_USBDeviceState == DETACHED_STATE) {
                UCON = 0x00;
                UIE  = 0x00;     // disable UIE interrupts kabeh
                while(UCONbits.USBEN == 0) {
                        UCONbits.USBEN = 1;
                }
        g_USBDeviceState = ATTACHED_STATE;
                // USB config register
        UCFGbits.UTEYE  = 0;
        UCFGbits.UOEMON = 0;
        UCFGbits.UPUEN  = 1;
        UCFGbits.UTRDIS = 0;            // nang chip ngirime aktif
        UCFGbits.FSEN   = 1;            // full speed clock (48 MHz)
```

```
        UCFGbits.PPB1   = 0;
        UCFGbits.PPB0   = 0;
        }

if(g_USBDeviceState == ATTACHED_STATE) {

if(UCONbits.SE0 == 0) {
        UIR = 0x00;                 // clear interrupt register
        UIE = 0x00;
        UIEbits.URSTIE = 1;
        UIEbits.IDLEIE = 1;
        g_USBDeviceState = POWERED_STATE;
                }
        }

if(UIRbits.ACTVIF &&            // lak D+ / D- terdeteksi aktif
UIEbits.ACTVIE) {
USBWakeFromSuspend();
}

if(UCONbits.SUSPND == 1) {
        return;
}

if(UIRbits.URSTIF &&  UIEbits.URSTIE) {
        USBInit();
        g_USBDeviceState = DEFAULT_STATE;
        g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfSetupPkt;
        g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
        g_buffDescTable[0].STAT.DTS    = 0;
        g_buffDescTable[0].STAT.KEN    = 0;
        g_buffDescTable[0].STAT.INCDIS = 0;
        g_buffDescTable[0].STAT.DTSEN  = 1;
        g_buffDescTable[0].STAT.BSTALL = 1;
        g_buffDescTable[0].STAT.BC9    = 0;
        g_buffDescTable[0].STAT.BC8    = 0;
        g_buffDescTable[0].STAT.UOWN   = 1;
        }
```

```
if(UIRbits.IDLEIF &&  UIEbits.IDLEIE) {
USBSuspend();
}

if(UIRbits.SOFIF &&
UIEbits.SOFIE) {
UIRbits.SOFIF = 0;
}

if(UIRbits.STALLIF &&
UIEbits.STALLIE) {
USBStallHandler();
}

if(UIRbits.UERRIF && UIEbits.UERRIE) {
  UEIR = 0x00;
}

if(g_USBDeviceState < DEFAULT_STATE) return;
if(UIEbits.TRNIE) {
for(i=0;i<4;i++) {
        if(UIRbits.TRNIF) {
                if(USTATbits.ENDP3 == 0 &&
                        USTATbits.ENDP2 == 0 &&
                        USTATbits.ENDP1 == 0 &&
                        USTATbits.ENDP0 == 0) {
                        USBEP0ControlTransfer();
                }
                UIRbits.TRNIF = 0;
                } else {
                        break;
                        }
                }
        }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
void USBWakeFromSuspend(void) {
        UCONbits.SUSPND = 0;
        UIEbits.ACTVIE = 0;
        while(UIRbits.ACTVIF) {
        UIRbits.ACTVIF = 0;
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////
void USBSuspend(void) {
        UIEbits.ACTVIE = 1;
        UIRbits.IDLEIF = 0;
        UCONbits.SUSPND = 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////
void USBStallHandler(void) {
        if(UEP0bits.EPSTALL == 1) {
        if((g_buffDescTable[0].STAT.UOWN   == 1)
        &&(g_buffDescTable[1].STAT.UOWN   == 1)) {
        g_buffDescTable[0].STAT.DTS    = 0;
        g_buffDescTable[0].STAT.KEN    = 0;
        g_buffDescTable[0].STAT.INCDIS = 0;
        g_buffDescTable[0].STAT.DTSEN  = 1;
        g_buffDescTable[0].STAT.BSTALL = 1;
        g_buffDescTable[0].STAT.BC9    = 0;
        g_buffDescTable[0].STAT.BC8    = 0;
        g_buffDescTable[0].STAT.UOWN   = 1;
}
UEP0bits.EPSTALL = 0;
}
UIRbits.STALLIF = 0;
}

////////////////////////////////////////////////////////////////////////////////////////////////
void USBEP0ControlTransfer(void) {
        if(USTATbits.DIR == 0) {
```

```
                if(g_buffDescTable[0].STAT.PID == SETUP_TOKEN)
{
                        USBSetupControlTransfer();
                } else {
                        USBOutControlTransfer();
                }
        } else if(USTATbits.DIR == 1) {
                USBInControlTransfer();
        }
}

////////////////////////////////////////////////////////////////////////////////////////////
void USBSetupControlTransfer(void) {
        g_buffDescTable[1].STAT.STATVal = 0x00;
        g_shortPacketStatus = SHORT_PKT_NOT_USED;
        g_controlTransferState = CTRL_TRF_WAIT_SETUP;
        g_inPipe.wCount = 0;
        g_inPipe.busy = 0;
        USBCheckStandardRequest();
        USBCheckHIDRequest();
        USBFinishControlTransferStuff();
}

////////////////////////////////////////////////////////////////////////////////////////////
void USBCheckStandardRequest(void) {
        if(g_ctrlTrfSetupPkt.RequestType != STANDARD) return;

        if(g_ctrlTrfSetupPkt.bRequest == SET_ADDRESS) {
                g_inPipe.busy = 1;
                g_USBDeviceState = ADR_PENDING_STATE;
        } else if(g_ctrlTrfSetupPkt.bRequest == GET_DESCRIPTOR)
{
                if(g_ctrlTrfSetupPkt.bmRequestType == 0x80) {
                        g_inPipe.busy = 1;
        if(g_ctrlTrfSetupPkt.bDescriptorType ==
USB_DESCRIPTOR_DEVICE) {
        g_inPipe.bRom = (rom BYTE*)&g_userDeviceDescriptor;
        g_inPipe.wCount = sizeof(g_userDeviceDescriptor);
```

```
        } else if(g_ctrlTrfSetupPkt.bDescriptorType ==
        USB_DESCRIPTOR_CONFIGURATION) {
g_inPipe.bRom = *(g_userConfigDescriptorPtr +
g_ctrlTrfSetupPkt.bDscIndex);
                g_inPipe.wRom = (rom WORD*)g_inPipe.bRom;
                g_inPipe.wCount = *(g_inPipe.wRom + 1);
        } else if(g_ctrlTrfSetupPkt.bDescriptorType ==
USB_DESCRIPTOR_STRING) {
g_inPipe.bRom = *(g_USBStringDescPtr +
g_ctrlTrfSetupPkt.bDscIndex);
g_inPipe.wCount = *g_inPipe.bRom;
} else {
g_inPipe.busy = 0;
        }
}
} else if(g_ctrlTrfSetupPkt.bRequest == SET_CONFIGURATION) {
g_inPipe.busy = 1;
g_USBActiveConfiguration = g_ctrlTrfSetupPkt.bConfigurationValue;
if(g_ctrlTrfSetupPkt.bConfigurationValue == 0) {
g_USBDeviceState = ADDRESS_STATE;
} else {
g_USBDeviceState = CONFIGURED_STATE;
UEP1bits.EPHSHK = 1;            // endpoint 1 handshake aktif
UEP1bits.EPCONDIS = 1;
UEP1bits.EPOUTEN = 1;           // endpoint 1 output aktif
UEP1bits.EPINEN = 1;            // endpoint 1 input aktif
UEP1bits.EPSTALL = 0;
g_buffDescTable[2].STAT.UOWN = 0;
g_buffDescTable[2].STAT.DTS  = 1;
g_buffDescTable[3].STAT.UOWN = 0;
g_buffDescTable[3].STAT.DTS  = 1;

transferFromHostToDeviceViaEP1((BYTE*)&g_fromHostToDeviceBu
ffer[1], 64);
                }
        }
}
```

```
////////////////////////////////////////////////////////////////////////////
void USBCheckHIDRequest(void) {
        if(g_ctrlTrfSetupPkt.Recipient != RECIPIENT_INTERFACE)
return;

        if(g_ctrlTrfSetupPkt.bRequest == GET_DESCRIPTOR) {
                if(g_ctrlTrfSetupPkt.bDescriptorType ==
HID_DESCRIPTOR) {
                if(g_USBActiveConfiguration == 1) {
        g_inPipe.bRom = (rom BYTE*)&g_configDescriptor1 + 18;
        g_inPipe.wCount = sizeof(USB_HID_DSC) + 3;
                        g_inPipe.busy = 1;
                }
                }
else if(g_ctrlTrfSetupPkt.bDescriptorType ==
REPORT_DESCRIPTOR) {
if(g_USBActiveConfiguration == 1) {
        g_inPipe.bRom = (rom BYTE*)&g_HIDReport01;
        g_inPipe.wCount = sizeof(g_HIDReport01);
        g_inPipe.busy = 1;
                        }
} else if(g_ctrlTrfSetupPkt.bDescriptorType == PHY_DESCRIPTOR) {
                g_inPipe.busy = 1;
                }
        }
}

////////////////////////////////////////////////////////////////////////////
void USBFinishControlTransferStuff(void) {
        UCONbits.PKTDIS = 0;
        if(g_inPipe.busy == 0) {
                if(g_outPipe.busy == 1) {
                }
else {
g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfSetupPkt;
g_buffDescTable[0].STAT.DTS    = 0;
g_buffDescTable[0].STAT.KEN    = 0;
```

```
g_buffDescTable[0].STAT.INCDIS = 0;
g_buffDescTable[0].STAT.DTSEN  = 1;
g_buffDescTable[0].STAT.BSTALL = 1;
g_buffDescTable[0].STAT.BC9    = 0;
g_buffDescTable[0].STAT.BC8    = 0;
g_buffDescTable[0].STAT.UOWN   = 1;

g_buffDescTable[1].STAT.DTS    = 0;
g_buffDescTable[1].STAT.KEN    = 0;
g_buffDescTable[1].STAT.INCDIS = 0;
g_buffDescTable[1].STAT.DTSEN  = 0;
g_buffDescTable[1].STAT.BSTALL = 1;
g_buffDescTable[1].STAT.BC9    = 0;
g_buffDescTable[1].STAT.BC8    = 0;
g_buffDescTable[1].STAT.UOWN   = 1;
                }
        }
else {
if(g_outPipe.busy == 0) {
if(g_ctrlTrfSetupPkt.DataDir == DEV_TO_HOST) {
if(g_ctrlTrfSetupPkt.wLength < g_inPipe.wCount) {
g_inPipe.wCount = g_ctrlTrfSetupPkt.wLength;
        }
USBControlTransferTransmit();
g_controlTransferState = CTRL_TRF_TX;
g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfSetupPkt;
g_buffDescTable[0].STAT.DTS    = 0;
g_buffDescTable[0].STAT.KEN    = 0;
g_buffDescTable[0].STAT.INCDIS = 0;
g_buffDescTable[0].STAT.DTSEN  = 0;
g_buffDescTable[0].STAT.BSTALL = 0;
g_buffDescTable[0].STAT.BC9    = 0;
g_buffDescTable[0].STAT.BC8    = 0;
g_buffDescTable[0].STAT.UOWN   = 1;

g_buffDescTable[1].ADR = (BYTE*)&g_ctrlTrfData;
g_buffDescTable[1].STAT.DTS    = 1;
```

```
g_buffDescTable[1].STAT.KEN    = 0;
g_buffDescTable[1].STAT.INCDIS = 0;
g_buffDescTable[1].STAT.DTSEN  = 1;
g_buffDescTable[1].STAT.BSTALL = 0;
g_buffDescTable[1].STAT.BC9    = 0;
g_buffDescTable[1].STAT.BC8    = 0;
g_buffDescTable[1].STAT.UOWN   = 1;
        }
else {
        g_controlTransferState = CTRL_TRF_RX;
        g_buffDescTable[1].CNT = 0;
        g_buffDescTable[1].STAT.DTS    = 1;
        g_buffDescTable[1].STAT.KEN    = 0;
        g_buffDescTable[1].STAT.INCDIS = 0;
        g_buffDescTable[1].STAT.DTSEN  = 1;
        g_buffDescTable[1].STAT.BSTALL = 0;
        g_buffDescTable[1].STAT.BC9    = 0;
        g_buffDescTable[1].STAT.BC8    = 0;
        g_buffDescTable[1].STAT.UOWN   = 1;

        g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
        g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfData;

        g_buffDescTable[0].STAT.DTS    = 1;
        g_buffDescTable[0].STAT.KEN    = 0;
        g_buffDescTable[0].STAT.INCDIS = 0;
        g_buffDescTable[0].STAT.DTSEN  = 1;
        g_buffDescTable[0].STAT.BSTALL = 0;
        g_buffDescTable[0].STAT.BC9    = 0;
        g_buffDescTable[0].STAT.BC8    = 0;
        g_buffDescTable[0].STAT.UOWN   = 1;
                        }
                }
        }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
void USBOutControlTransfer(void) {
        if(g_controlTransferState == CTRL_TRF_RX) {
                USBControlTransferReceive();
        }
else {
USBPrepareForNextSetupTransfer();
        }
}

////////////////////////////////////////////////////////////////////////////////////////////////
void USBInControlTransfer(void) {
        BYTE lastDTS;

        lastDTS = g_buffDescTable[1].STAT.DTS;

        if(g_USBDeviceState == ADR_PENDING_STATE) {
                UADDR = g_ctrlTrfSetupPkt.bDevADR;
                if(UADDR > 0) {
                        g_USBDeviceState = ADDRESS_STATE;
        } else {
                        g_USBDeviceState = DEFAULT_STATE;
                }
        }

if(g_controlTransferState == CTRL_TRF_TX) {
        g_buffDescTable[1].ADR = (BYTE*)g_ctrlTrfData;
        USBControlTransferTransmit();
if(g_shortPacketStatus == SHORT_PKT_SENT) {
        g_buffDescTable[1].STAT.DTS    = 0;
        g_buffDescTable[1].STAT.KEN    = 0;
        g_buffDescTable[1].STAT.INCDIS = 0;
        g_buffDescTable[1].STAT.DTSEN  = 0;
        g_buffDescTable[1].STAT.BSTALL = 1;
        g_buffDescTable[1].STAT.BC9    = 0;
        g_buffDescTable[1].STAT.BC8    = 0;
        g_buffDescTable[1].STAT.UOWN   = 1;
        }
```

```
else {
if(lastDTS == 0) {
        g_buffDescTable[1].STAT.DTS    = 1;
        g_buffDescTable[1].STAT.KEN    = 0;
        g_buffDescTable[1].STAT.INCDIS = 0;
        g_buffDescTable[1].STAT.DTSEN  = 1;
        g_buffDescTable[1].STAT.BSTALL = 0;
        g_buffDescTable[1].STAT.BC9    = 0;
        g_buffDescTable[1].STAT.BC8    = 0;
        g_buffDescTable[1].STAT.UOWN   = 1;
}
else {
g_buffDescTable[1].STAT.DTS    = 0;
g_buffDescTable[1].STAT.KEN    = 0;
g_buffDescTable[1].STAT.INCDIS = 0;
g_buffDescTable[1].STAT.DTSEN  = 1;
g_buffDescTable[1].STAT.BSTALL = 0;
g_buffDescTable[1].STAT.BC9    = 0;
g_buffDescTable[1].STAT.BC8    = 0;
g_buffDescTable[1].STAT.UOWN   = 1;
                        }
                }
        }
else {
        USBPrepareForNextSetupTransfer();
        }
}

////////////////////////////////////////////////////////////////////////////////
void USBControlTransferReceive(void) {
        BYTE byteToRead;
        BYTE i;

        byteToRead = g_buffDescTable[0].CNT;
        if(byteToRead > g_outPipe.wCount) {
                byteToRead = g_outPipe.wCount;
        } else {
                g_outPipe.wCount = g_outPipe.wCount - byteToRead;
```

```
        }
        for(i=0;i<byteToRead;i++) {
                *g_outPipe.bRam++ = g_ctrlTrfData[i];
        }
        if(g_outPipe.wCount > 0) {
                g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
                g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfData;
        if(g_buffDescTable[0].STAT.DTS == 0) {
                g_buffDescTable[0].STAT.DTS    = 1;
                g_buffDescTable[0].STAT.KEN    = 0;
                g_buffDescTable[0].STAT.INCDIS = 0;
                g_buffDescTable[0].STAT.DTSEN  = 1;
                g_buffDescTable[0].STAT.BSTALL = 0;
                g_buffDescTable[0].STAT.BC9    = 0;
                g_buffDescTable[0].STAT.BC8    = 0;
        g_buffDescTable[0].STAT.UOWN   = 1;
                }
else {
g_buffDescTable[0].STAT.DTS    = 0;
g_buffDescTable[0].STAT.KEN    = 0;
g_buffDescTable[0].STAT.INCDIS = 0;
g_buffDescTable[0].STAT.DTSEN  = 1;
g_buffDescTable[0].STAT.BSTALL = 0;
g_buffDescTable[0].STAT.BC9    = 0;
g_buffDescTable[0].STAT.BC8    = 0;
g_buffDescTable[0].STAT.UOWN   = 1;
                }
        }
else {
g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfSetupPkt;

g_outPipe.busy = 0;
        }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////
void USBControlTransferTransmit(void) {
        WORD_STRUCT byteToSend;
        BYTE *pDestination;

if(g_inPipe.wCount < USB_EP0_BUFF_SIZE) {
        byteToSend.wordVal = g_inPipe.wCount;
        if(g_shortPacketStatus == SHORT_PKT_NOT_USED) {
        g_shortPacketStatus = SHORT_PKT_PENDING;
}
else if(g_shortPacketStatus == SHORT_PKT_PENDING) {
g_shortPacketStatus = SHORT_PKT_SENT;
                }
        }
else {
        byteToSend.wordVal = USB_EP0_BUFF_SIZE;
        }
g_buffDescTable[1].STAT.BC9 = 0;
g_buffDescTable[1].STAT.BC8 = 0;
g_buffDescTable[1].STAT.STATVal |= byteToSend.byte.HB;
g_buffDescTable[1].CNT = byteToSend.byte.LB;

g_inPipe.wCount = g_inPipe.wCount - byteToSend.wordVal;

pDestination = (BYTE*)g_ctrlTrfData;

while(byteToSend.wordVal) {
        *pDestination = *g_inPipe.bRom;
        *pDestination++;
        *g_inPipe.bRom++;
        byteToSend.wordVal--;
        }
}

////////////////////////////////////////////////////////////////////////////////////////////
void USBPrepareForNextSetupTransfer(void) {
        if((g_controlTransferState == CTRL_TRF_RX) &&
          (UCONbits.PKTDIS == 1) &&
```

```
        (g_buffDescTable[0].CNT == sizeof(CTRL_TRF_SETUP))
&&
        (g_buffDescTable[0].STAT.PID == SETUP_TOKEN) &&
        (g_buffDescTable[0].STAT.UOWN == 0)) {
                BYTE setup_cnt;
                g_buffDescTable[0].ADR =
(BYTE*)&g_ctrlTrfSetupPkt;
        for(setup_cnt = 0; setup_cnt < sizeof(CTRL_TRF_SETUP);
setup_cnt++) {
*(((BYTE*)&g_ctrlTrfSetupPkt) + setup_cnt) =
*(((BYTE*)&g_ctrlTrfData) + setup_cnt);
        }
        }
else {
g_controlTransferState = CTRL_TRF_WAIT_SETUP;
g_buffDescTable[0].CNT = USB_EP0_BUFF_SIZE;
g_buffDescTable[0].ADR = (BYTE*)&g_ctrlTrfSetupPkt;
g_buffDescTable[0].STAT.DTS    = 0;
g_buffDescTable[0].STAT.KEN    = 0;
g_buffDescTable[0].STAT.INCDIS = 0;
g_buffDescTable[0].STAT.DTSEN  = 1;
g_buffDescTable[0].STAT.BSTALL = 1;
g_buffDescTable[0].STAT.BC9    = 0;
g_buffDescTable[0].STAT.BC8    = 0;
g_buffDescTable[0].STAT.UOWN   = 1;

g_buffDescTable[1].STAT.UOWN   = 0;
g_buffDescTable[1].STAT.DTS    = 0;
g_buffDescTable[1].STAT.KEN    = 0;
g_buffDescTable[1].STAT.INCDIS = 0;
g_buffDescTable[1].STAT.DTSEN  = 0;
g_buffDescTable[1].STAT.BSTALL = 0;
g_buffDescTable[1].STAT.BC9    = 0;
g_buffDescTable[1].STAT.BC8    = 0;

        }
if(g_outPipe.busy == 1) {
        g_outPipe.busy = 0;
```

```
        }
}

////////////////////////////////////////////////////////////////////////////////////////////
void transferFromHostToDeviceViaEP1(BYTE* pPacketData, BYTE
packetLength) {
g_buffDescTable[2].STAT.DTS    = !g_buffDescTable[2].STAT.DTS;

g_buffDescTable[2].STAT.KEN    = 0;
g_buffDescTable[2].STAT.INCDIS = 0;
g_buffDescTable[2].STAT.DTSEN  = 1;
g_buffDescTable[2].STAT.BSTALL = 0;
g_buffDescTable[2].STAT.BC9    = 0;
g_buffDescTable[2].STAT.BC8    = 0;
g_buffDescTable[2].STAT.UOWN   = 1;

g_buffDescTable[2].CNT = packetLength;
g_buffDescTable[2].ADR = pPacketData;
}

////////////////////////////////////////////////////////////////////////////////////////////
void transferFromDeviceToHostViaEP1(BYTE* pPacketData, BYTE
packetLength) {
g_buffDescTable[3].STAT.DTS    = !g_buffDescTable[3].STAT.DTS;

g_buffDescTable[3].STAT.KEN    = 0;
g_buffDescTable[3].STAT.INCDIS = 0;
g_buffDescTable[3].STAT.DTSEN  = 1;
g_buffDescTable[3].STAT.BSTALL = 0;
g_buffDescTable[3].STAT.BC9    = 0;
g_buffDescTable[3].STAT.BC8    = 0;
g_buffDescTable[3].STAT.UOWN   = 1;

g_buffDescTable[3].CNT = packetLength;
g_buffDescTable[3].ADR = pPacketData;
}
```

```
/************************************************/
School of Physics
University of Brawijaya
Written by      : Muh. Ali Syahbana
ID              : 0810930049
Last Modified   : 25 Desember 2012
File nime       : USBStuff.h
/************************************************/

#define LED0 PORTBbits.RB0
#define LED1 PORTBbits.RB1
#define LED2 PORTBbits.RB2
#define LED3 PORTBbits.RB3

#define LED4  PORTDbits.RD7
#define LED5  PORTDbits.RD6
#define LED6  PORTDbits.RD5
#define LED7  PORTDbits.RD4
#define LED8  PORTDbits.RD3
#define LED9  PORTDbits.RD2
#define LED10 PORTDbits.RD0
#define LED11 PORTDbits.RD1

#define SWITCH1 PORTAbits.RA1
#define SWITCH2 PORTAbits.RA2
#define SWITCH3 PORTAbits.RA3

#define TURN_LED3_OFF      0x80
#define TURN_LED3_ON       0x81

#define DO_NOT_USE_DEBUG_LEDS          0x00
#define USE_DEBUG_LEDS                 0x01

#define SWITCH1_NOT_PRESSED     0x00
#define SWITCH1_PRESSED         0x01

#define SWITCH2_NOT_PRESSED     0x00
```

```
#define SWITCH2_PRESSED             0x01

#define SWITCH3_NOT_PRESSED         0x00
#define SWITCH3_PRESSED             0x01

#define DETACHED_STATE              0x00
#define ATTACHED_STATE              0x01
#define POWERED_STATE               0x02
#define DEFAULT_STATE               0x04
#define ADR_PENDING_STATE           0x08
#define ADDRESS_STATE               0x10
#define CONFIGURED_STATE            0x20

// g_controlTransferState
#define CTRL_TRF_WAIT_SETUP         0
#define CTRL_TRF_TX                 1
#define CTRL_TRF_RX                 2

// kango digaweg_shortPacketStatus
#define SHORT_PKT_NOT_USED          0
#define SHORT_PKT_PENDING           1
#define SHORT_PKT_SENT              2

// kango digaweg_ctrlTrfSetupPkt.bDescriptorType
#define USB_DESCRIPTOR_DEVICE               0x01
#define USB_DESCRIPTOR_CONFIGURATION        0x02
#define USB_DESCRIPTOR_STRING               0x03
#define USB_DESCRIPTOR_INTERFACE            0x04
#define USB_DESCRIPTOR_ENDPOINT             0x05
#define USB_DESCRIPTOR_DEVICE_QUALIFIER     0x06
#define USB_DESCRIPTOR_OTHER_SPEED          0x07
#define USB_DESCRIPTOR_INTERFACE_POWER      0x08
#define USB_DESCRIPTOR_OTG                  0x09

typedef enum _BOOL { FALSE = 0, TRUE = 1 } BOOL;
```

```
#define NULL 0

typedef unsigned char       BYTE;       // unsigned 8-bit int
typedef unsigned short      WORD;       // unsigned 16-bit int
typedef unsigned long       UINT32;     // unsigned 32-bit int
typedef unsigned long long  UINT64;     // unsigned 64-bit int

typedef union _WORD_VAL {           // 2 bytes
        WORD wordVal;
        struct {
                BYTE LB;
                BYTE HB;
        } byte;
} WORD_STRUCT;

typedef struct _USB_DEVICE_DESCRIPTOR {
        BYTE bLength;
        BYTE bDescriptorType;
        WORD bcdUSB;
        BYTE bDeviceClass;
        BYTE bDeviceSubClass;
        BYTE bDeviceProtocol;
        BYTE bMaxPacketSize0;
        WORD idVendor;
        WORD idProduct;
        WORD bcdDevice;
        BYTE iManufacturer;
        BYTE iProduct;
        BYTE iSerialNumber;
        BYTE bNumConfigurations;
} USB_DEVICE_DESCRIPTOR;

typedef struct _USB_HID_DSC {
        BYTE bLength;
        BYTE bDescriptorType;
        WORD bcdHID;
        BYTE bCountryCode;
        BYTE bNumDsc;
```

```
} USB_HID_DSC;

typedef struct {
        union {
                struct {
                        unsigned BC8:1;
                        unsigned BC9:1;
                        unsigned BSTALL:1;
                        unsigned DTSEN:1;
                        unsigned INCDIS:1;
                        unsigned KEN:1;
                        unsigned DTS:1;
                        unsigned UOWN:1;
                };
                struct {
                        unsigned BC8:1;
                        unsigned BC9:1;
                        unsigned PID:4;
                        unsigned :1;
                        unsigned UOWN:1;
                };
                BYTE STATVal;
        } STAT;
        BYTE CNT;
        BYTE *ADR;
} BDT_ENTRY;

#define USB_EP0_BUFF_SIZE       8

#define HID_INTERFACE_ID        0x00

#define HID_NUM_OF_DSC          1
#define HID_RPT01_SIZE          29
#define _EP_IN          0x80
#define _EP_OUT         0x00

// konfigurasine attributes
#define _DEFAULT    (0x01 << 7)
```

```
#define _SELF           (0x01 << 6)
#define _RWU            (0x01 << 5)

// endpoint transfer
#define _CTRL 0x00
#define _ISO    0x01
#define _BULK 0x02
#define _INT    0x03

// kodene USB HID
#define HID_INTF                0x03

typedef union _CTRL_TRF_SETUP {
        struct {
                BYTE bmRequestType;
                BYTE bRequest;
                WORD wValue;
                WORD wIndex;
                WORD wLength;
        };
        struct {
                unsigned Recipient:5;
                unsigned RequestType:2;
                unsigned DataDir:1;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
        };
        struct {
                unsigned :8;
                unsigned :8;
                BYTE bDscIndex;
                BYTE bDescriptorType;
                WORD wLangID;
```

```
                unsigned :8;
                unsigned :8;
        };
        struct {
                unsigned :8;
                unsigned :8;
                BYTE bDevADR;
                BYTE bDevADRH;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
        };
        struct {
                unsigned :8;
                unsigned :8;
                BYTE bConfigurationValue;
                BYTE bCfgRSD;
                unsigned :8;
                unsigned :8;
                unsigned :8;
                unsigned :8;
        };
} CTRL_TRF_SETUP;

typedef struct {
        rom BYTE *bRom;
        rom WORD *wRom;
        BYTE busy;
        WORD wCount;
} IN_PIPE;

typedef struct {
        BYTE *bRam;
        BYTE busy;
        WORD wCount;
} OUT_PIPE;
```

```
#define GET_STATUS              0
#define CLEAR_FEATURE           1
#define SET_FEATURE             3
#define SET_ADDRESS             5
#define GET_DESCRIPTOR          6
#define SET_DESCRIPTOR          7
#define GET_CONFIGURATION       8
#define SET_CONFIGURATION       9
#define GET_INTERFACE           10
#define SET_INTERFACE           11
#define SYNCH_FRAME             12

// kango digaweg_buffDescTable.STAT.PID
#define SETUP_TOKEN         0b1101
#define OUT_TOKEN           0b0001 //  kode HC ngirim nang USB
#define IN_TOKEN            0b1001 //  nerimo data teko USB

// kango digawe g_ctrlTrfSetupPkt.DataDir
#define HOST_TO_DEV             0
#define DEV_TO_HOST             1

// kango digawe CTRL_TRF_SETUP.RequestType
#define STANDARD    0x00
#define CLASS       0x01
#define VENDOR              0x02

// kango digawe karo g_ctrlTrfSetupPkt.Recipient
#define RECIPIENT_DEVICE        0
#define RECIPIENT_INTERFACE         1
#define RECIPIENT_ENDPOINT          2
#define RECIPIENT_OTHER             3

// kango digawe karo g_ctrlTrfSetupPkt.bRequest
#define GET_REPORT          0x01
#define GET_IDLE            0x02
#define GET_PROTOCOL        0x03
#define SET_REPORT          0x09
#define SET_IDLE            0x0A
```

```
#define SET_PROTOCOL        0x0B

// kanggo g_ctrlTrfSetupPkt.bDescriptorType, karo deklarasikno
deskriptor
#define HID_DESCRIPTOR              0x21
#define REPORT_DESCRIPTOR           0x22
#define PHY_DESCRIPTOR              0x23

#define CLK_FREQ 48000000

// function prototypes
/////////////////////////////////////////////////////////////////////////////
void USBInit(void);
void USBTasks(void);
void USBWakeFromSuspend(void);
void USBSuspend(void);
void USBStallHandler(void);
void USBEP0ControlTransfer(void);
void USBSetupControlTransfer(void);
void USBOutControlTransfer(void);
void USBInControlTransfer(void);
void USBCheckStandardRequest(void);
void USBCheckHIDRequest(void);
void USBFinishControlTransferStuff(void);
void USBControlTransferReceive(void);
void USBControlTransferTransmit(void);
void USBPrepareForNextSetupTransfer(void);
void transferFromHostToDeviceViaEP1(BYTE* pPacketData, BYTE
packetLength);
void transferFromDeviceToHostViaEP1(BYTE* pPacketData, BYTE
packetLength);
void showByteOnLEDs(BYTE byteToShow);
void delay(void);
```