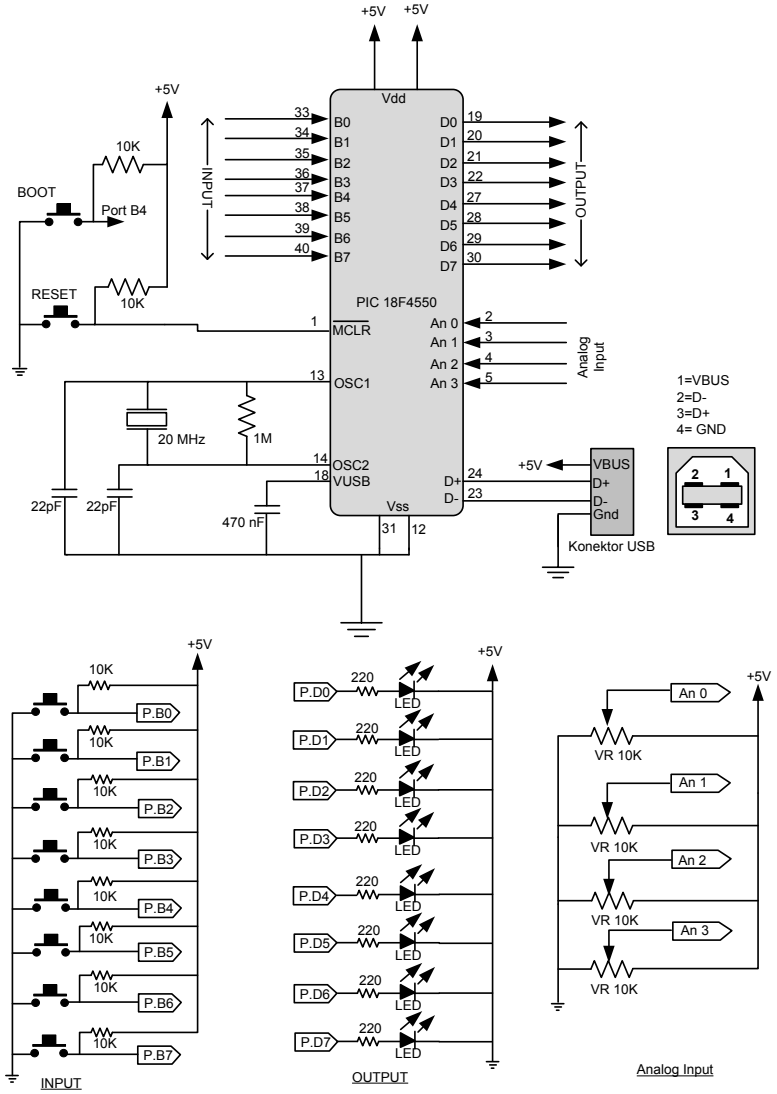
BAB IV
APLIKASI PIC 18F4550 DENGAN KOMPUTER MENGGUNAKAN *METODE EMULATING RS 232 OVER USB*

HARDWARE DAN DESAIN KOMUNIKASI HOST-DEVICE

Metode *Emulating RS 232 Over USB* pada aplikasi sehari-hari sering dikenal dengan **virtual COM**. Virtual COM memungkinkan komputer seolah-olah mempunyai port COM standar RS232 namun komunikasi data yang terjadi secara fisik melalui USB. Virtual COM inilah yang dijadikan dasar komunikasi dua arah antara host dengan device, sehingga dalam pemrograman aplikasi host (komputer), seolah-olah kita memprogram antarmuka serial standar RS 232 biasa. Apabila kita menggunakan software pemrograman Visual Basic 6.0 komponen inti untuk komunikasi serial adalah MSComm. Dari sisi device, Microchip telah menyediakan contoh project *CDC RS-232 Emulation* sebagai referensi bagi programmer.

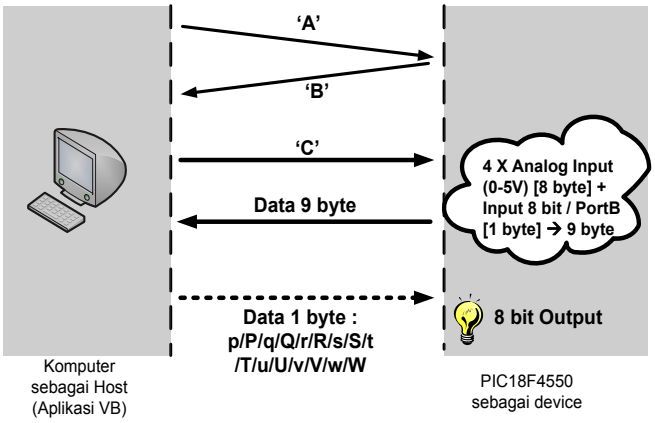
Pada prinsipnya, kelas USB ini mampu menangani komunikasi *line input/output* dengan kemampuan bit per-detiknya sama dengan pada komunikasi serial (bisa sampai 1 Kbyte per transaksi atau 1 ms). Namun kelemahan jenis USB ini adalah apabila komunikasi terputus ditengah jalan, virtual COM tidak mampu meneruskan paket data yang sempat tersendat, dan akhirnya aplikasi sisi host akan *crash*.
Dalam contoh kali ini kita akan membuat aplikasi yang memungkinkan kita memanfaatkan 4 analog input, 8 bit input dan 8 saluran untuk output (lihat Gambar 4.1) Pengontrolan Port pada PIC menggunakan program aplikasi komputer Visual Basic 6.



Gambar 4.1 Rangkaian Hardware Project "CDC_ku"

Aplikasi rangkaian ini banyak dipakai untuk keperluan akuisisi data yang nilai datanya dapat dipantau melalui komputer. Analog input berasal dari tegangan variable 0 sampai 5 volt, sedangkan inputan 8 saluran bisa berupa *bus data* 8 bit dan dalam hal ini disimulasikan dengan saklar *push button* biasa. Output PIC juga sebanyak 8 saluran yang bisa dijadikan aksi keluaran dari aplikasi yang dirancang.

Pada Gambar 4.1 terlihat output yang digunakan berada di Port D, sedangkan input menggunakan Port B. Analog input yang digunakan menggunakan Port A yang berjumlah 4 saluran input analog. Saklar BOOT dan RESET digunakan untuk meng*download* firmware menggunakan bootload yang telah disediakan oleh Microchip (lihat subbab mengenai bootloader di bab 9). Dengan beberapa tuntutan fungsi tersebut, maka kita harus menyusun atau mendesain sistem komunikasi antara host dan device.



Gambar 4.2 Desain komunikasi antara PC dengan PIC18F4550

Gambar 4.2 merupakan ilustrasi dari sistem komunikasi secara keseluruhan. Penjelasannya adalah sebagai berikut:

Pertama-tama host mengirimkan kode "**A**" ke device untuk mengecek komunikasi yang terjadi, apakah ada respon (balasan) atau tidak. Jika device eksis maka device (PIC) akan membalas dengan karakter "**B**". Jika karakter "**B**" tersebut diterima oleh oleh host, berarti host dapat menyimpulkan bahwa device telah siap untuk diajak bertransaksi data.

Kemudian host akan mengirimkan kode "**C**" secara *real time*. Respon dari device adalah mengirimkan byte hasil pembacaan analog input yang masing – masing 2 byte (ADC 10 bit). Karena ada 4 saluran input analog (4 channel ADC), maka jumlah keseluruhan data informasi ADC yang dikirim ke host adalah 8 byte. Data ADC sebesar 8 byte tersebut masih ditambah lagi dengan pembacaan dari input PortB sebesar 1 byte sehingga total keseluruhan adalah 9 byte. jadi pada akhirnya 9 byte data yang harus dikirim oleh device (PIC) ke host

Oleh host, kesembilan byte tersebut diolah untuk ditampilkan secara visual. Permintaan kesembilan byte tersebut merupakan permintaan *real time* dan berlangsung terus menerus. Dari sisi pemrograman host, permintaan dengan kode karakter "C" tersebut dilakukan dalam sebuah looping timer yang berjalan ketika device telah terdeteksi "konek".

Selain kode karakter ASCII "C", masih ada satu kode lagi yang berfungsi untuk mengatur ouput/keluaran Port D PIC, dan kode ini berasal dari host. Kode yang dimaksud adalah 1 karakter yang dikirim dari host untuk 8 bit (keluaran byte) pada Port D PIC18F4550 yaitu karakter (dalam ASCII) :

- "P" jika Port D0 bernilai 1 dan "p" jika Port D0 bernilai 0
- "Q" jika Port D1 bernilai 1 dan "q" jika Port D1 bernilai 0
- "R" jika Port D2 bernilai 1 dan "r" jika Port D2 bernilai 0
- "S" jika Port D3 bernilai 1 dan "s" jika Port D3 bernilai 0
- "T" jika Port D4 bernilai 1 dan "t" jika Port D4 bernilai 0
- "U" jika Port D5 bernilai 1 dan "u" jika Port D5 bernilai 0
- "V" jika Port D6 bernilai 1 dan "v" jika Port D6 bernilai 0
- "W" jika Port D7 bernilai 1 dan "w" jika Port D7 bernilai 0

Kode karakter untuk mengatur keluaran Port D tersebut dilakukan di atas *even* (kejadian) oleh user/pengguna dan tidak *real time.*

Kesimpulannya, PIC 18F4550 harus mampu melakukan aksi sebagai berikut (perhatikan antara karakter kapital dengan yang non kapital) :

1. Jika menerima karakter “A”, maka akan mengirimkan ke host karakter “B”
2. Jika menerima karakter “C”, maka akan mengirimkan 8 byte pembacaan keempat saluran analog input ditambah 1 byte nilai pembacaan PortB
3. Jika menerima karakter “P”, maka Port D0 akan bernilai 1
4. Jika menerima karakter “p”, maka Port D0 akan bernilai 0
5. Jika menerima karakter “Q”, maka Port D1 akan bernilai 1
6. Jika menerima karakter “q”, maka Port D1 akan bernilai 0
7. Jika menerima karakter “R”, maka Port D2 akan bernilai 1
8. Jika menerima karakter “r”, maka Port D2 akan bernilai 0
9. Jika menerima karakter “S”, maka Port D3 akan bernilai 1
10. Jika menerima karakter “s”, maka Port D3 akan bernilai 0
11. Jika menerima karakter “T”, maka Port D4 akan bernilai 1
12. Jika menerima karakter “t”, maka Port D4 akan bernilai 0
13. Jika menerima karakter “U”, maka Port D5 akan bernilai 1
14. Jika menerima karakter “u”, maka Port D5 akan bernilai 0
15. Jika menerima karakter “V”, maka Port D6 akan bernilai 1
16. Jika menerima karakter “v”, maka Port D6 akan bernilai 0
17. Jika menerima karakter “W”, maka Port D7 akan bernilai 1
18. Jika menerima karakter “w”, maka Port D7 akan bernilai 0

Apabila semua prinsip komunikasi host-device telah selesai dirancang, selanjutnya kita sudah bisa masuk ke ranah pemrograman. Pemrograman di sisi device menggunakan bahasa C standard ANSI sedangkan di sisi host menggunakan bahasa basic dengan program aplikasi Visual Basic 6.0. Untuk memulainya ketiga program yaitu MPLAB IDE, C18 Compiler, dan Visual Basic 6 harus sudah terinstall di komputer.

PROGRAMMING FIRMWARE

File yang dibutuhkan dalam pembuatan project ini adalah **CDC_RS232_Emulation.exe**, dan dapat diunduh dari situs Microchip (www.microchip.com). Instalasi secara default akan membuat directory baru di **C:¥MCHPFSUSB** yang berisi project contoh komunikasi kelas CDC.

Directory C:¥MCHPFSUSB sendiri masih terdiri dari beberapa subfolder dan di dalamnya terdapat beberapa file lagi yang merupakan sebuah kesatuan project contoh Microchip kelas CDC. Karena kita akan membuat project yang berdiri sendiri, kita perlu memilah file yang perlu dan yang tidak dalam pembuatan project kali ini. File yang dibutuhkan dalam project ini adalah seperti pada Tabel 4.1. Semua file dalam Tabel 4.1 dapat ditemukan di dalam directory **C:¥MCHPFSUSB.**

File-file tersebut dibedakan menjadi tiga macam yaitu *source, header* dan *linker. CDC_RS232_Emulation* sendiri disediakan oleh Microchip agar para programmer lebih mudah dalam menyusun program, khususnya kelas *CDC Serial Emulation*. Kita bisa mengedit secara langsung file-file tersebut (dalam Tabel 4.1) dan menyesuaikan dengan fungsi I/O PIC, namun langkah tersebut kurang efetif, karena kita semua file yang tidak diperlukan juga masih menjadi satu directory dengan file kerja yang kita buat.

Project yang akan kita buat adalah **CDC_ku.mcp** (mcp adalah ekstensi untuk project dari

mikrokontroler PIC). Project CDC_ku akan terdiri dari beberapa file *source, header*, dan *linker*. Untuk memudahkan dalam mengakses, memperbaiki seperti layaknya proyek software lainnya, kita buat sebuah directory yaitu: **C:¥CDC_ku** untuk meletakkan file CDC_ku.mcp.

Jenis File	Nama file	Asal directory :
File Source	cdc.c	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥class¥cd
	main.c	C:¥MCHPFSUSB¥fw¥Cdc
	usb9.c	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usb9
	usbctrltrf.c	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbctrltrf
	usbdrv.c	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbdrv
	usbdsc.c	C:¥MCHPFSUSB¥fw¥Cdc¥autofiles
	usbmmap.c	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb
	user.c	C:¥MCHPFSUSB¥fw¥Cdc¥user
File Header	cdc.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥class¥cdc
	io_cfg.h	C:¥MCHPFSUSB¥fw¥Cdc
	typedefs.h	C:¥MCHPFSUSB¥fw¥Cdc¥system
	usb9.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usb9
	usb.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb
	usb_compile_time_validation.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb
	usbcfg.h	C:¥MCHPFSUSB¥fw¥Cdc¥autofiles
	usbctrltrf.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbctrltrf
	usbdefs_ep0_buff.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbdefs
	usb_std_dsc.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbdefs
	usbdrv.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb¥usbdrv
	usbdsc.h	C:¥MCHPFSUSB¥fw¥Cdc¥autofiles
	usbmmap.h	C:¥MCHPFSUSB¥fw¥Cdc¥system¥usb
	user.h	C:¥MCHPFSUSB¥fw¥Cdc¥user
File Linker	rm18f4550.lkr	C:¥MCHPFSUSB¥fw¥Cdc

Tabel 4.1 Daftar file yang dibutuhkan dalam project CDC_ku.

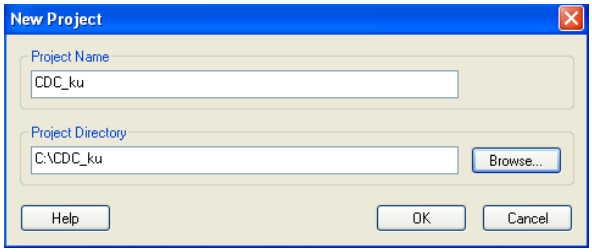
Setelah directory C:¥CDC_ku terbentuk, kita buat lagi subfolder di bawahnya sebagai berikut:

- *Header* (**C:¥CDC_ku¥Header**) untuk meletakkan file header (*.h),
- *Source* (**C:¥CDC_ku¥Source**) untuk meletakkan file source (*.c),
- *Linker* (**C:¥CDC_ku¥Linker**) untuk meletakkan file linker (*.rm),
- *Output* (**C:¥CDC_ku¥Output**) untuk meletakkan hasil akhir firmware (output).

Setelah subfolder–subfolder (di bawah C:¥CDC_ku) tersebut dibuat, salin semua file yang dibutuhkan (Tabel 4.1) ke directory yang bersesuaian. File yang di butuhkan untuk *source* diletakkan di **C:¥CDC_ku¥Source**, *Header* di **C:¥CDC_ku¥Header**, dan *Linker* di **C:¥CDC_ku¥Linker.**

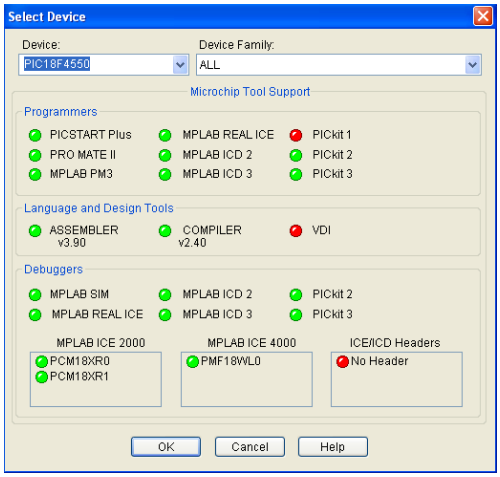
Selanjutnya langkah-langkah membuat project CDC_ku dengan **MPLAB IDE** adalah sebagai berikut:

1. Buka MPAB IDE, Klik :
   Project> New …
   Namakan project yang baru dengan nama CDC_ku dengan project directory di **C:\CDC_ku.** (Gambar 4.3)



Gambar 4.3. Pilihan penamaan Project dan peletakan directory (C:\CDC_ku)

2. Pemilihan device PIC18F4550 dapat dilakukan dengan cara:
Configure>Select Device …
Lalu pilih **PIC18F4550** pada pilihan Device. (Gambar 4.4)

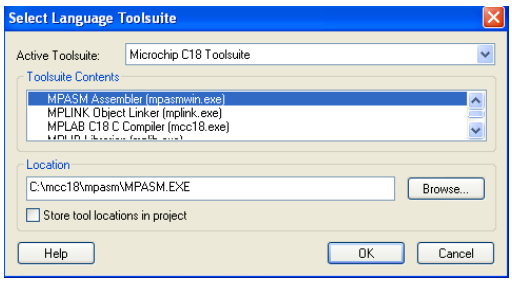


Gambar 4.4. Pemilihan jenis mikrokontroler PIC18F4550

3. Untuk memastikan bahwa MPLAB IDE menggunakan jenis kompiler C18 maka pilih :

Project>Select Languange Toolsuite > …

Sebelumnya pastikan bahwa Microchip C18 telah terinstall dan terdapat di directori **C:\mcc18**. Langkah-langkah intalasi bahasa C18 jenis ini dapat dilihat pada bab 9.



Gambar 4.5. Pemilihan jenis kompiler menggunakan Microchip C18

Sebuah jendela pilihan kompiler akan muncul seperti pada Gambar 4.5. Berikut adalah pemilihan letak directory bahasa dan kompiler yang digunakan (Gambar 4.5):

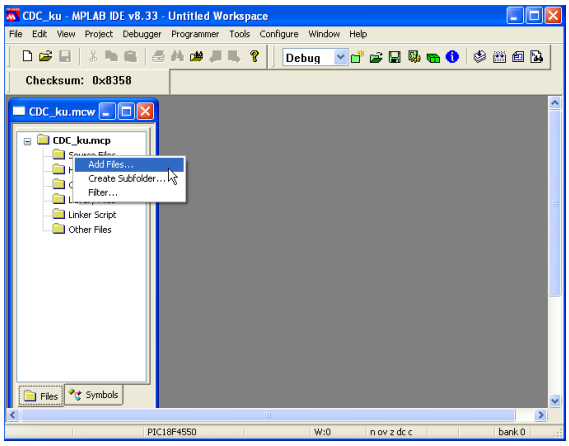
- MPASM Assembler (mpasmwin.exe): **C:\mcc18\mpasm\MPASM.EXE**

- MPLINK Object Linker (mplink.exe) : C**:\MCC18\bin\mplink.exe**
- MPLAB C18C Compiler (mcc18.exe): **C:\MCC18\bin\mcc18.exe**
- MPLAB Librarian (mplib.exe) : **C:\mcc18\bin\mplib.exe**

4. Sekarang project CDC_ku sudah terbuat. Untuk mengaktifkan jendela *editor file project*, dapat melalui:

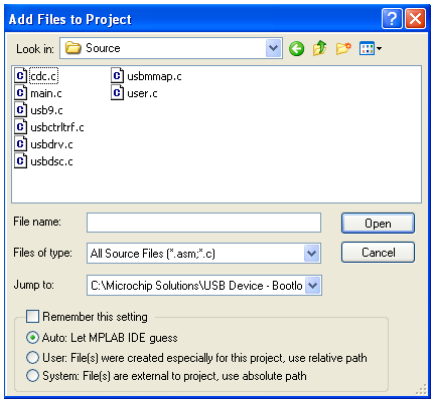
View>Project > ....

Kita dapat menambahan file source dengan cara menempatkan kursor mouse ke arah **Source Files** pada **Project View** lalu klik kanan , pilih **Add files …** (Gambar 4.6)

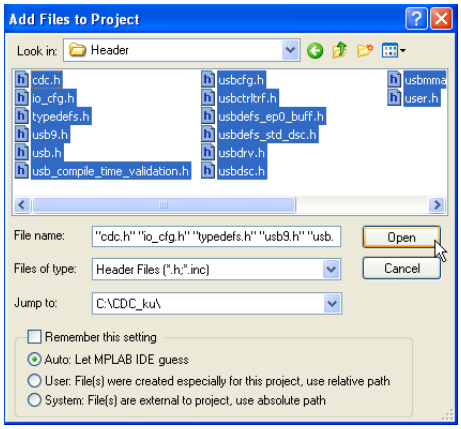


Gambar 4.6. Klik Kanan pada source Files dan pilihan untuk menambahakan file akan muncul.

Arahkan pilihan file yang akan kita tambahkan ke *Source Files* pada directory **C:\CDC_ku\Source.** Jendela pemilihan file yang akan ditambahkan ke *Source Files* akan muncul seperti Gambar 4.7. Pilih semua file C yang ada di **C:\CDC_ku\Source** dengan cara menyorot semua file pada file tersebut lalu klik **Open ...** (Gambar 4.8)



Gambar 4.7. Pemilihan File Source (C:\CDC_ku\Source)

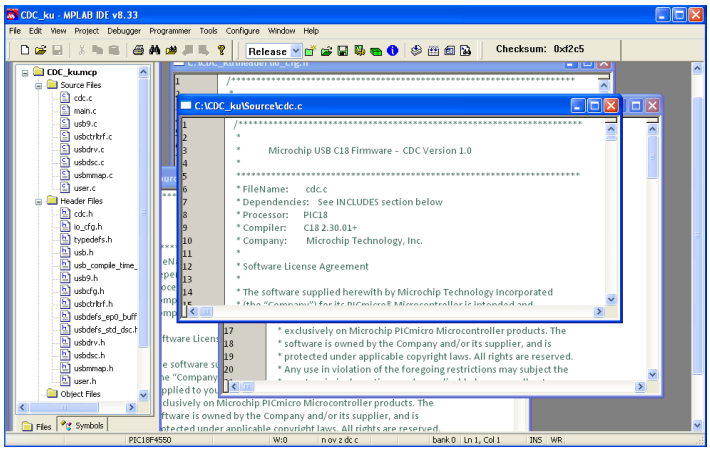


Gambar 4.8. Pemilihan File Header (C:\CDC_ku\Header)

Pada bagian Header Files lakukan hal yang serupa, pilih juga semua file yang ada di **C:\CDC_ku\Header**, klik tombol **Open.**

File Linker yang digunakan hanya satu buah yaitu **rm18f4550.rm** juga terletk di **C:\CDC_ku\Linker.**

Apabila seluruh file yang sudah ditambahkan dalam project CDC_ku, tampilan MPLAB akan tampak seperti pada Gambar 4.9.



Gambar 4.9. Tampilan akhir MPLAB IDE jika sudah selesai menambahkan file ke project CDC_ku.

5. Agar hasil kompiler dan pengaturan MPLAB IDE dalam mengeksekusi file tidak tersesat, perlu diatur nilai *set path* dari project CDC_ku. Pemilihan directory output project dan link dari project CDC_ku dilakukan dengan mengklik :

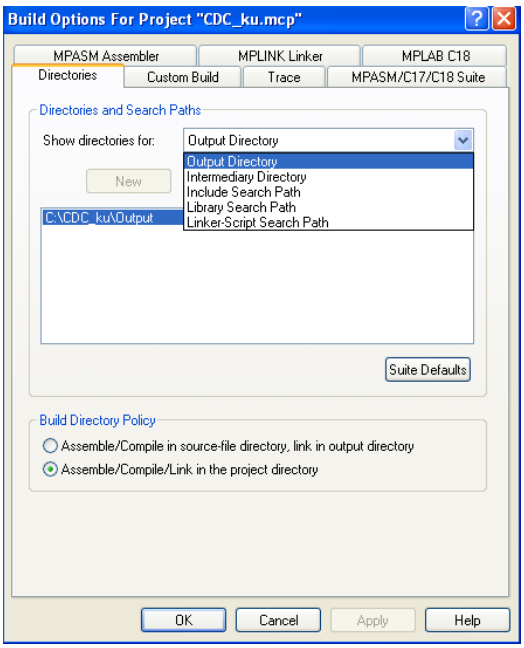
   Project > Build Option >Project > …

   Jendela seperti Gambar 4.10. akan muncul. Pengaturan directory ini akan menentukan MPLAB IDE dalam mencari file path yang digunakan, serta meletakkan file output hasil kompiler. Oleh karena itu, disarankan untuk membagi file-file dalam sebuh project ke dalam folder-folder sesuai dengan jenis file yang bersangkutan, seperti subbab di atas.

   Pemilihan path directory untuk project CDC_ku adalah sebagai berikut :

   - Output Directory: **C:\CDC_ku\Output**

- Intermediarry Directory : **C:\CDC_ku\Output**
- Include Search Path : **C:\CDC_ku**
- Library Search Path (3 macam): **C:\mcc18\lib ; C:\CDC_ku;** dan **C:\mcc18**
- Linker-Scripth Search Path : **C:\CDC_ku\Linker**



Gambar 4.10. Pengaturan Path MPLAB IDE.

Framework Usb Dan Modifikasi File Source (C:\CDC_ku\Source)

Framework Firmware USB yang disediakan oleh Microchip memungkinkan kita untuk memodifikasi file yang sudah tersedia sehingga memudahkan kita membangun sebuah aplikasi dengan USB. Di dalam framework juga telah tersedia file driver *.INF yang merupakan driver komunikasi antara aplikasi dengan device hardware.

Beberapa file harus diubah dan disesuaikan dengan keadaan lingkungan directory baru di: C:¥CDC_ku. Selain itu, disesuaikan pula dengan fungsi kerja PIC yang berhubungan dengan I/O dan ADC. Berikut ini merupakan bagian yang diubah dari masing-masing file dalam project CDC_ku.

1. **cdc.c**

   Modifikasi dilakukan pada baris ke-39:

   ```
   #include "system¥typedefs.h"
   #include "system¥usb¥usb.h"
   ```

   diganti menjadi:

   ```
   #include "header¥typedefs.h"
   #include "header¥usb.h"
   ```

2. **main.c**

   Modifikasi dilakukan pada baris ke-39:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
#include "io_cfg.h"
#include "system¥usb¥usb_compile_time_validation.h"
#include "user¥user.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
#include "header¥io_cfg.h"
#include "header¥usb_compile_time_validation.h"
#include "header¥user.h"
```

Pada baris ke-111:

```
{
    ADCON1 |= 0x0F;
    #if defined(USE_USB_BUS_SENSE_IO)
    tris_usb_bus_sense = INPUT_PIN;
    #endif

    #if defined(USE_SELF_POWER_SENSE_IO)
    tris_self_power = INPUT_PIN;
    #endif
```

diganti menjadi :

```
{
    #if defined(USE_USB_BUS_SENSE_IO)
    tris_usb_bus_sense = INPUT_PIN;
    #endif

    #if defined(USE_SELF_POWER_SENSE_IO)
    tris_self_power = INPUT_PIN;
    #endif
```

3. **usb9.c**

Modifikasi dilakukan pada baris ke-39:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
#include "io_cfg.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
#include "header¥io_cfg.h"
```

4. **usbctrltrf.c**

Modifikasi dilakukan pada baris ke-39

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
```

5. **usbdrv.c**

File usbdrv.c merupakan jantung dari interrupt dan pengaturan transisi proses tingkatan,

mulai dari device sebelum terdeteksi di root hub (detached), mulai dicolokkan (*attached*), lalu diberi power (*powered)*, sampai proses enumerasi selesai.
Modifikasi dilakukan pada baris ke--39:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
#include "io_cfg.h"
```

diganti menjadi :

```
#include "header¥typedefs.h"
#include "header¥usb.h"
#include "header¥io_cfg.h"
```

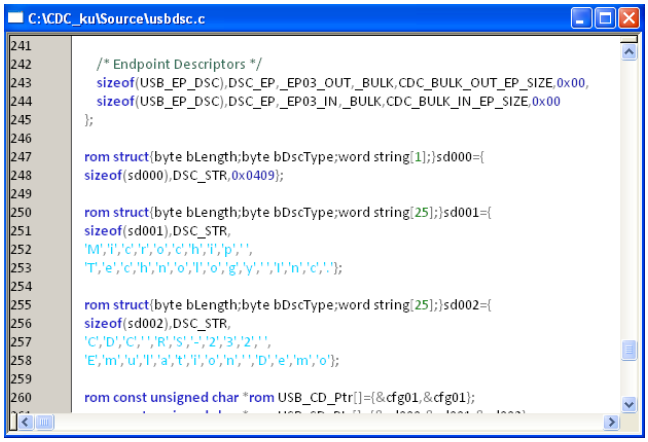
6. usbdsc.c

File usbdsc.c berisi informasi descriptor yang harus dilaporkan ke PC. Bagian yang penting di sini adalah penyetinggan nomor endpoint pada descriptor interface (*interface descritor*), serta penambahan dan modifikasi descriptor endpoint.
Modifikasi file ini terletak pada baris ke-173:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
```



Gambar 4.11. Standar Descriptor String yang disediakan Microchip

Terkadang sebagai pengguna sebuah mikrokontroler kita juga harus mengganti Vendor ID dan Produk ID (Gambar 4.12). Penggantian Vendor ID ini menjadi penting untuk menghindari kemungkinan produk yang sama-sama menggunakan produk Microchip, mempunyai descriptor yang sama, sehingga akan menyebabkan kesalahan identifikasi yang salah pada driver host. Walaupun keadaan ini kecil peluangnya, namun kemungkinan ini bisa saja terjadi. Penggantian Vendor ID dan Product ID juga berguna untuk meng*ekslusifkan* produk yang kita desain dari produk sejenis yang sama-sama menggunakan antarmuka USB.
Aturan dan lisensi tentang penggunaan Product ID dan Vendor ID ini di atur oleh USBIF (www.usb.org). Penggunaan Vendor ID (VID) sesuai dengan nomor yang dipunyai oleh perusahaan peripheral., sehingga satu VID bisa terdiri dari beberapa Product ID. Jika dalam praktiknya keperluan device adalah murni komersial, sangat disarankan untuk mempunyai lisensi PID dari USBIF sebagai vendor USB.

Modifikasi file usbdsc.c berguna untuk mengubah *string descriptor* serta nilai Product ID, Vendor ID dan dilakukan pada baris ke-255:

```
rom struct{byte bLength;byte bDscType;word string[25];}sd002={
sizeof(sd002),DSC_STR,
'C','D','C',' ','R','S','-','2','3','2',' ',
'E','m','u','l','a','t','i','o','n',' ','D','e','m','o'};
```

diganti menjadi:

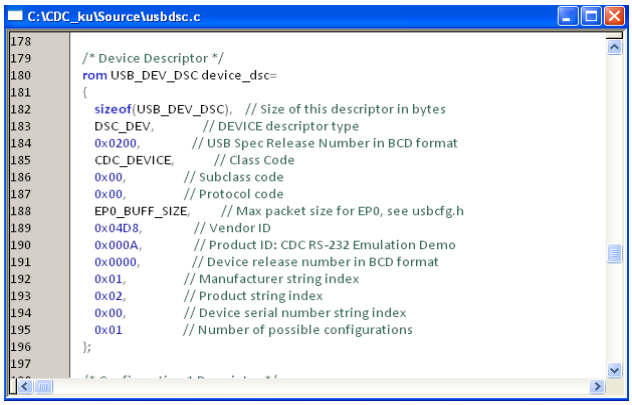
```
rom struct{byte bLength;byte bDscType;word string[12];}sd002={
sizeof(sd002),DSC_STR,
'M','y','_','F','i','r','s','t','_','C','D','C'};
```

Untuk ProductID dan Vendor ID pada baris ke-189:

```
0x04D8,        // Vendor ID
 0x000A,        // Product ID: CDC RS-232 Emulation Demo
```

diganti menjadi:

```
0x0899,        // Vendor ID milik kita yang baru
 0x0099,        // Product ID: My_First_CDC
```



Gambar 4.12. Standar Produk ID dan Vendor ID yang disediakan Microchip (baris ke-189 dan 190)

Dengan penggantian tersebut nilai Product ID dari device kita adalah 0099 dan Vendor ID 0899. Descriptor stringnya adalah "My_First_CDC".

7. usbmmap.c

Modifikasi dilakukan pada baris ke-152:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
```

8. user.c

Pengaturan I/O, ADC dan segala fungsi mikrokontroler dilakukan oleh file user.c. File user.c mengatur perintah atau data masukan dan mengembalikan (jika diperlukan), menyiapkan data, dan mengirim ke host. Modifikasi file ini dilakukan sesuai dengan rancangan awal kita dalam membuat project kelas CDC memanfaatkan 4 analog input, 8 saluran input dan 8 saluran output. Berikut merupakan modifikasi file user.c.

Pada baris ke-53:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
#include "io_cfg.h"                    // I/O pin mapping
#include "user¥user.h"
#include "user¥temperature.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
#include "header¥io_cfg.h"          // I/O pin mapping
#include "header¥user.h"
// Untuk ADC
#include <adc.h>
#include <stdlib.h>
#include <delays.h>
```

Pada baris ke-70 setelah :

```
rom char ansi_clrscr[]={"¥x1b[2J"};
```

ditambah:

```
//untuk adc dan transaksi data
int Result;
byte data_masuk;
byte AN0L,AN0H,AN1L,AN1H,AN2L,AN2H,AN3L,AN3H;
byte nilai_Port D,nilai_portB;
```

Pada baris ke-76 setelah :

```
BOOL Switch3IsPressed(void);
```

ditambah:

```
void konversiADC0 (void);
void konversiADC1 (void);
void konversiADC2 (void);
void konversiADC3 (void);
void inisialisasi_port (void);
```

Baris ke 77:

```
void Exercise_Example(void);
void Exercise_01(void);
void Exercise_02(void);
void Exercise_03(void);
void Exercise_04(void);
void Exercise_05(void);
```

dihilangkan (dihapus).

Selanjutnya pada baris ke-89 :

```
mInitAllLEDs();
mInitAllSwitches();
old_sw2 = sw2;
old_sw3 = sw3;
InitTempSensor();
```

diganti menjadi:

```
mInitAllLEDs();
mInitAllSwitches();
old_sw2 = sw2;
old_sw3 = sw3;
```

Baris ke-129 sampai baris ke-348 setelah kode:

```
void ProcessIO(void)
{
```

dan sebelum kode:

```
void BlinkUSBStatus(void)
```

disisipi dengan kode berikut:

```
// User Application USB tasks
    if((usb_device_state < CONFIGURED_STATE)||(UCONbits.SUSPND==1)) return;
if (getsUSBUSART(input_buffer, 1))
{
        data_masuk = input_buffer[0];

        switch(data_masuk)
        {
        case 'A' :
                output_buffer[0]= 'B';
                if (mUSBUSARTIsTxTrfReady())
                {
                mUSBUSARTTxRam((byte*)output_buffer,1);
                }
                break;
        case 'C':
                TRISB = 0b11111111;
                nilai_portB = PORTB ;
                konversiADC0();
                konversiADC1();
                konversiADC2();
                konversiADC3();
                //send via USB
                output_buffer[0]= AN0L;
                output_buffer[1]= AN0H;
                output_buffer[2]= AN1L;
                output_buffer[3]= AN1H;
                output_buffer[4]= AN2L;
                output_buffer[5]= AN2H;
                output_buffer[6]= AN3L;
                output_buffer[7]= AN3H;
                output_buffer[8]= nilai_portB;
                if (mUSBUSARTIsTxTrfReady())
                {
                mUSBUSARTTxRam((byte*)output_buffer,9);
                }
                break;
                case 'P':
                TRISDbits. TRISD0 = 0;
                PORT Dbits.RD0= 1;
                break;
                case 'p':
                TRISDbits. TRISD0 = 0;
                PORT Dbits.RD0= 0;
                break;
                case 'Q':
                TRISDbits. TRISD1 = 0;
                PORT Dbits.RD1= 1;
                break;
                case 'q':
                TRISDbits. TRISD1 = 0;
                PORT Dbits.RD1= 0;
                break;
                case 'R':
                PORT Dbits.RD2= 1;
```

```
                    break;
                    case 'r':
                    TRISDbits. TRISD2 = 0;
                    PORT Dbits.RD2= 0;
                    break;
                    case 'S':
                    TRISDbits. TRISD3 = 0;
                    PORT Dbits.RD3= 1;
                    break;
                    case 's':
                    TRISDbits. TRISD3 = 0;
                    PORT Dbits.RD3= 0;
                    break;
                    case 'T':
                    TRISDbits. TRISD4 = 0;
                    PORT Dbits.RD4= 1;
                    break;
                    case 't':
                    TRISDbits. TRISD4 = 0;
                    PORT Dbits.RD4= 0;
                    break;
                    case 'U':
                    TRISDbits. TRISD5 = 0;
                    PORT Dbits.RD5= 1;
                    break;
                    case 'u':
                    TRISDbits. TRISD5 = 0;
                    PORT Dbits.RD5= 0;
                    break;
                    case 'V':
                    TRISDbits. TRISD6 = 0;
                    PORT Dbits.RD6= 1;
                    break;
                    case 'v':
                    TRISDbits. TRISD6 = 0;
                    PORT Dbits.RD6= 0;
                    break;
                    case 'W':
                    TRISDbits. TRISD7 = 0;
                    PORT Dbits.RD7= 1;
                    break;
                    case 'w':
                    TRISDbits. TRISD7 = 0;
                    PORT Dbits.RD7= 0;
                    break;
                }
                }
}//end ProcessIO
void konversiADC0 (void)
{
  OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD,
                    ADC_CH0 & ADC_VREFPLUS_VDD & ADC_VREFMINUS_VSS&
ADC_INT_OFF, 11 );
  Delay10TCYx( 5 );
  ConvertADC();          // Start conversion
  while( BusyADC() );    // Wait for completion
  ReadADC();
  AN0H = ADRESH;
  AN0L = ADRESL;
  CloseADC();                    // Disable A/D converte
}
void konversiADC1 (void)
{
  OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD,
```

```
                    ADC_CH1     &     ADC_VREFPLUS_VDD&     ADC_VREFMINUS_VSS&
ADC_INT_OFF, 11 );
  Delay10TCYx( 5 );
  ConvertADC();          // Start conversion
  while( BusyADC() );    // Wait for completion
  ReadADC();
  AN1H = ADRESH;
  AN1L = ADRESL;
  CloseADC();                    // Disable A/D converte
}
void konversiADC2 (void)
{
  OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD,
                    ADC_CH2     &     ADC_VREFPLUS_VDD&     ADC_VREFMINUS_VSS&
ADC_INT_OFF, 11 );
  Delay10TCYx( 5 );
  ConvertADC();          // Start conversion
  while( BusyADC() );    // Wait for completion
  ReadADC();
  AN2H = ADRESH;
  AN2L = ADRESL;
  CloseADC();                    // Disable A/D converte
}
void konversiADC3 (void)
{
  OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD,
                    ADC_CH3     &     ADC_VREFPLUS_VDD&     ADC_VREFMINUS_VSS&
ADC_INT_OFF, 11 );
  Delay10TCYx( 5 );
  ConvertADC();          // Start conversion
  while( BusyADC() );    // Wait for completion
  ReadADC();
  AN3H = ADRESH;
  AN3L = ADRESL;
  CloseADC();                    // Disable A/D converte
}
void inisialisasi_port (void)
{
TRISA = 0b11111111;
TRISD = 0b00000000;
TRISBbits.TRISB0=1;
TRISBbits.TRISB1=1;
TRISBbits.TRISB2=1;
TRISBbits.TRISB3=1;
}
```

MODIFIKASI FILE HEADER (c:\CDC_ku\Header) dan LINKER (c:\CDC_ku\Linker)

Semua file header tidak memerlukan modifikasi khusus, kecuali untuk setingan kode agar kompatibel dengan directory C:¥CDC_ku, sehingga eksekusi program semua file *.h tidak tersesat. Berikut daftar file header yang perlu dimodifikasi beserta kode program modifikasi :

1. **cdc.h**

Modifikasi pada baris ke-40:

```
#include "system¥typedefs.h"
```

diganti menjadi:

```
#include "header¥typedefs.h"
```

2. io_cfg.h

Modifikasi pada baris ke-41:

```
#include "autofiles¥usbcfg.h"
```

diganti menjadi:

```
#include "header¥usbcfg.h"
```

3. typedefs.h

File ini tidak mengalami modifikasi.

4. usb.h

Modifikasi pada baris ke-47:

```
#include "autofiles¥usbcfg.h"
#include "system¥usb¥usbdefs¥usbdefs_std_dsc.h"
#include "autofiles¥usbdsc.h"

#include "system¥usb¥usbdefs¥usbdefs_ep0_buff.h"
#include "system¥usb¥usbmmap.h"

#include "system¥usb¥usbdrv¥usbdrv.h"
#include "system¥usb¥usbctrltrf¥usbctrltrf.h"
#include "system¥usb¥usb9¥usb9.h"

#if defined(USB_USE_CDC)
#include "system¥usb¥class¥cdc¥cdc.h"
#endif
```

diganti dengan:

```
#include "header¥usbcfg.h"
#include "header¥usbdefs_std_dsc.h"
#include "header¥usbdsc.h"

#include "header¥usbdefs_ep0_buff.h"
#include "header¥usbmmap.h"

#include "header¥usbdrv.h"
#include "header¥usbctrltrf.h"
#include "header¥usb9.h"
#if defined(USB_USE_CDC)
#include "header¥cdc.h"
#endif
```

5. usb9.h

Modifikasi pada baris ke-40:

```
#include "system¥typedefs.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
```

6. usbcfg.h

Modifikasi pada baris ke 46:

```
#define USE_SELF_POWER_SENSE_IO
#define USE_USB_BUS_SENSE_IO
```

dihilangkan atau dihapus, karena fasilitas pengaturan Power USB tidak dipakai.

7. usbctrltrf.h

Modifikasi pada baris ke-40:

```
#include "system¥typedefs.h"
```

diganti dengan

```
#include "header¥typedefs.h"
```

8. usbdrv.h

Modifikasi pada baris ke-40:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
```

9. usbdsc.h

Modifikasi pada baris ke-43:

```
#include "system¥typedefs.h"
#include "autofiles¥usbcfg.h"
#if defined(USB_USE_CDC)
#include "system¥usb¥class¥cdc¥cdc.h"
#endif
#include "system¥usb¥usb.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
#include "header¥usbcfg.h"
#if defined(USB_USE_CDC)
#include "header¥cdc.h"
#endif
#include "header¥usb.h"
```

10. usbmmap.h

Modifikasi pada baris ke-41:

```
#include "system¥typedefs.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
```

11. usb_compile_time_validation.h

Modifikasi pada baris ke-40:

```
#include "system¥typedefs.h"
#include "system¥usb¥usb.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
#include "header¥usb.h"
```

12. usbdefs_ep0_buff.h

Modifikasi pada baris ke-44:

```
#include "system¥typedefs.h"
#include "autofiles¥usbcfg.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
#include "header¥usbcfg.h"
```

13. usbdefs_std_dsc.h

Modifikasi pada baris ke-44:

```
#include "system¥typedefs.h"
```

diganti dengan:

```
#include "header¥typedefs.h"
```

14. user.h

Tidak ada modifikasi

File linker rm18f4550.lkr sama dengan file yang ada di directory **C:¥MCHPFSUSB¥fw¥Cdc**, tidak mengalami perubahan/modifikasi. File ini sudah dimodifikasi dari **18f4550.lkr** yang asli yang berada di **C:¥mcc18¥lkr¥18f4550.lkr**. Perubahan kode dimaksudkan agar alamat awal reset vektor berada di 0800h, sehingga dapat berjalan pada mode bootload. Berikut perubahan yang dimaksud:

Pada baris ke-11 file 18f4550.lkr yang ada di C:¥mcc18¥lkr¥18f4550.lkr

```
CODEPAGE   NAME=vectors   START=0x0                END=0x29
PROTECTED
CODEPAGE   NAME=page      START=0x2A               END=0x7FFF
```

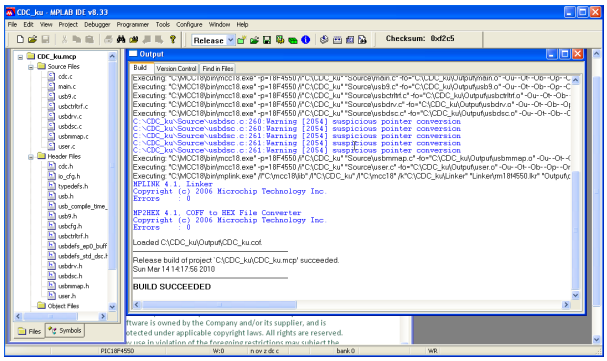
diganti dengan:

```
CODEPAGE   NAME=boot      START=0x0      END=0x7FF        PROTECTED
CODEPAGE   NAME=vectors   START=0x800    END=0x0x829      PROTECTED
CODEPAGE   NAME=page      START=0x82A    END=0x7FFF
```

Apabila modifikasi telah selesai, kita bisa melakukan kompile program pada MPLAB IDE dengan memilih:

Project> Build All ....

Sebuah jendela Output akan muncul dan menampilkan keterangan output dari project CDC_ku. (Gambar 4.13). Karena setting output project ada di C:¥CDC_ku¥Output, maka file **CDC_ku.hex** juga dihasilkan dalam directory tersebut.

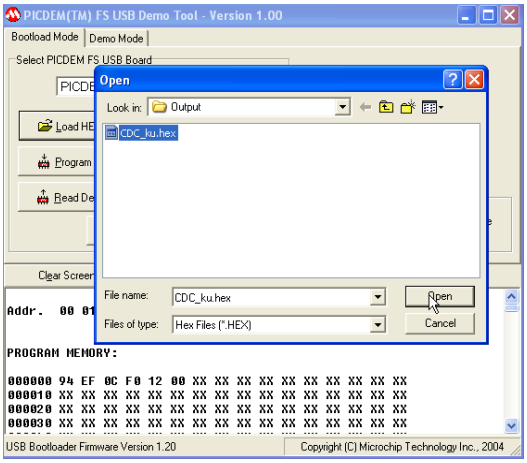


Gambar 4.13. Jendela Output project CDC_ku.

Mendownload File Hex Ke PIC 18F4550

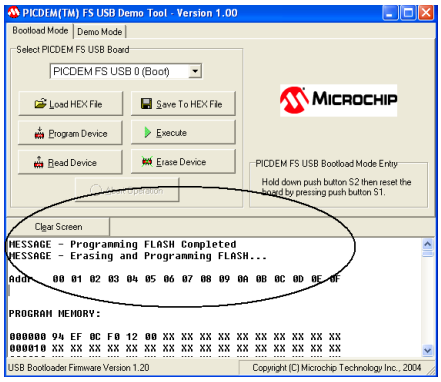
File **CDC_ku.hex** (C:¥CDC_ku¥Output) tidak mungkin berjalan pada keadaan vektor interupt normal pada PIC18F4550, melainkan di atas sistem bootload. Oleh karena itu, pertama kali kita harus mengisi PIC18F4550 kita dengan **MCHPUSB.hex**, sebuah bootload bawaan dari Microchip. Pada saat PIC telah terisi MCHPUSB.hex maka PIC sudah siap menjalankan program bootloader dan kita masuk mode *update firmware*, seperti yang telah dijelaskan pada Bab 9.

Identifikasi keberhasilan bahwa PIC sudah masuk ke menu *update firmware/bootload* adalah adanya keadaan blink LED pada D0 dan D1 secara bergantian, serta dari sisi aplikasi **PIC DEM PFSUSB TOOL** sudah mampu mendeteksi PIC dalam *update mode*.
Dengan menggunakan aplikasi PIC DEM PFSUSB TOOL upload program **CDC_ku.hex** (C:¥CDC_ku¥Output) yang telah kita buat. Jalankan PIC DEM PFSUSB TOOL, lalu masuk ke mode *update firmware PIC* dengan menekan BOOT pertama kali lalu disusul RESET, dan dilepaskan bersama-sama. Port D0 dan Port D1 akan blink bergantian sebagai tanda PIC masuk ke *mode update firmware*. Dari sisi PIC DEM PFSUSB TOOL ,sebuah jendela seperti Gambar 4.14 akan muncul, klik ***Open***, dan tampilan PIC DEM PFSUSB TOOL akan kembali ke menu utama.



Gambar 4.14. Pemilihan file CDC_ku.hex yang akan didownload ke PIC

Klik tombol **Program Device**, PIC DEM PFSUSB TOOL akan memrogram PIC sampai muncul pesan bahwa proses pemrograman telas selesai (Gambar 4.15)

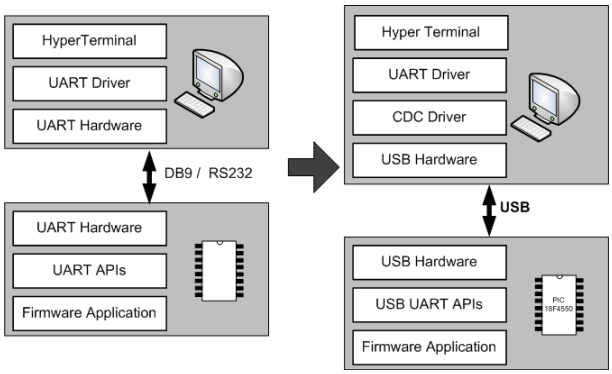


Gambar 4.15. Proses mentransfer file CDC_ku.hex ke PIC melalui bootloader sudah berhasil.

Setelah berhasil memrogram, tekan tombol RESET pada rangkaian /hardware untuk kembali ke *mode user*, dan menjalankan aplikasi. Kini PIC 18F4550 telah siap dijalankan.

Driver Untuk Aplikasi Host

Pada awalnya, penggunaan RS 232 sudah mencukupi untuk beberapa aplikasi. Namun penggunaan port ini semakin hari semakin ditinggalkan. Windows sendiri sebagai pembuat Sistem Operasi yang paling banyak dipakai oleh pengguna komputer masih mengusung kelas transfer jenis ini, walaupun dalam bentuk port USB. Perbedaan antara RS232 dengan USB yang menggunakan kelas CDC adalah seperti Gambar 4.16 berikut.
Sejak CDC dijadikan standar kelas USB, Microsost telah mengimplementasikan sebuah driver yang mendukung emulasi antarmuka RS232. Dari sisi PC, driver CDC menyediakan sebuah layer yang menghubungkan antara hardware USB dan UART Driver. Dengan adanya driver tersebut, aplikasi paling akhir akan tetap memandang seolah-olah Windows menggunakan standar RS232.



Gambar 4.16. Perbedaan RS232 murni dengan USB

Driver CDC telah disediakan oleh Microsoft, namun masih perlu dilakukan pemfilteran. Filter tersebut disediakan oleh Microchip sebagai vendor, dan kita bisa mendapatkan driver .INF tersebut di folder **C:¥MCHPFSUSB¥fw¥Cdc¥inf¥win2k_winxp**. Tentu saja setelah kita menginstall **CDC_RS232_Emulation.EXE**. Produk ID dan Vendor ID juga dapat ditemukan dalam .INF file tersebut dan harus disamakan dengan nilai di firmware.
Dilihat dari sisi user, sekali hardware dideteksi oleh Windows, maka Windows akan meminta driver (file .INF) tersebut. Namun setelah selesai, proses selanjutnya tidak perlu me*load* driver itu lagi. Nilai Vendor ID dari produk USB yang kita buat di sisi firmware maupun driver harus sama. Berikut modifikasi dari file mchpcdc.inf yang ada **di C:¥MCHPFSUSB¥fw¥Cdc¥inf¥win2k_winxp:**
Pada baris ke 25:

```
%DESCRIPTION%=DriverInstall, USB¥VID_04D8&PID_000A
```

diganti dengan:

```
%DESCRIPTION%=DriverInstall, USB¥VID_0899&PID_0099
```

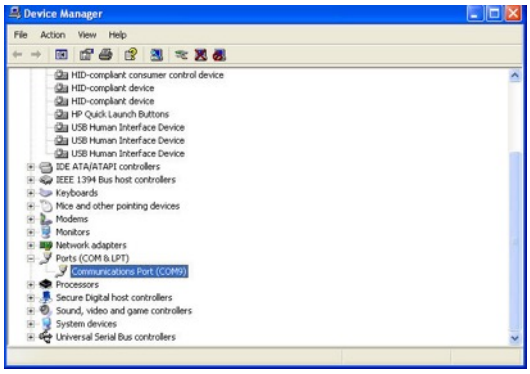
Setelah selesai diganti, ubah nama file **mchpcdc.inf** dengan nama **CDC_ku_driver.inf.**
Pada aplikasi 18F4550 yang kita buat, ketika windows akan meminta driver untuk PIC18F4550, maka kita tinggal mengarahkan ke **C:¥MCHPFSUSB¥fw¥Cdc¥inf¥win2k_winxp¥** dan pilih file **CDC_ku_driver.inf**

Gambar 4.17. Identifikasi device pertama kali

Pertama kali Windows mendeteksi keberadaan device diilustrasikan seperti pada Gambar 4.17. Kemudian sebuah jendela informasi mengenai driver akan muncul (10.18), dan selanjutnya akan menunjukkan informasi file .INF driver tersebut.

Gambar 4.18. Identifikasi model driver

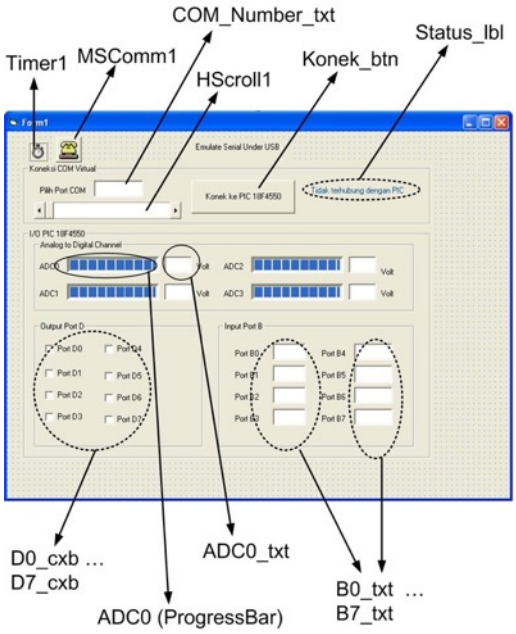
Pada jendela *Device Manager* (masuk *Device Manager* pada Windows : *Run*>ketik *devmgmt.msc*), ketika PIC18F4550 sudah tersambung dan driver **CDC_ku_driver.inf** sudah terpasang dengan benar, Port COM akan muncul seperti pada Gambar 4.19.
Faktor yang menentukan nilai port COM (COM1, COM2, dst) adalah jenis driver yang sudah tersimpan di register Windows. Device dengan nilai Product ID dan Vendor ID paling baru dideteksi dan di simpan di *registry Windows* akan menempati port COM yang paling akhir.



Gambar 4.19. Penambahan device baru berupa port COM pada *Device Manager*

Programming VB 6 (HOST)

Ada banyak bahasa pemrograman GUI (Graphical User Interface) yang bisa dipakai dan mendukung antarmuka serial/modem, mulai dari Dephi, Visual Basic dari versi 6 sampai platform .NET, serta Java. Semuanya tergantung dari kesenangan dan subyektifitas programmer. Kesempatan kali ini kita akan memakai Visual Basic 6. Programming Visual Basic di tingkat host untuk jenis USB ini sebenarnya juga bisa diimplementasikan untuk komunikasi jenis serial standar RS232. Komponen utama dalam membangun komunikasi ini adalam **MS Comm,** dan terdapat pada komponen bawaan aplikasi Visual Basic. Perlu diingat bahwa host-lah yang memegang peranan utama dalam setiap proses komunikasi dengan device. Dengan demikian, host memiliki peranan utama dalam menampilkan informasi data, dan mengatur kerja device.
Desain form (VB6) diilustrasikan seperti Gambar 4.20. Komponen yang dipakai dalam form ini adalah sebagai berikut:



Gambar 4.20. Gambar tata letak desain layout pada VB 6.

1) Form1 (Form)

Form1 merupakan form utama yang berjalan pertama kali ketika program jalan. Script untuk form ini adalah sebagai berikut:

```
Private Sub Form_Load()
Nilai_Port_COM = 1    'Mula mula nilai COM =1
Nilai_Port D = 0
B0_txt.Text = "0"
B1_txt.Text = "0"
B2_txt.Text = "0"
B3_txt.Text = "0"
B4_txt.Text = "0"
B5_txt.Text = "0"
B6_txt.Text = "0"
B7_txt.Text = "0"
COM_Number_txt.Text = 1
End Sub
```

2) Timer1 (Timer)

Timer1 berfungsi untuk menagih data sebesar 9 byte dari device, dengan mengirimkan karakter "C". Data tersebut diolah dengan cara 8 (delapan) byte pertama dikodekan menjadi 4 buah informasi analog, dan satu byte lagi yang merepresentasikan nilai Port B. Nilai ADC 10 bit dikirim sebanyak 2 kali sehingga menjadi 2 byte. Agar dapat dikonversi menjadi nilai tegangan 0 sampai 5 volt makan perhitungannya menjadi :

{([DataMasuk(byte kedua) AND 3] x 256) + (DataMasuk(byte pertama) / 204.6)}

Variable DataMasuk (array) merepresentasikan data yang diterima melalui port COM. Nilai byte tersebut di *and*kan dengan nilai 3 (biner: 00000011) agar 5 bit MSB (*Most Significant Bit*) tidak mempengaruhi nilai ADC. Ingat bahwa hasil pembacaan data 10 bit akan menghasilkan 2 byte: byte tinggi (3 byte) pada register ADRESH dan register ADRESL untuk byte rendah.

Representasi nilai Port B menggunakan metode modulus, dan pembagian. Hal itu dikarenakan data Port B bentuknya dalam satu byte. Kita harus mengidentifikasi nilai byte tersebut untuk mendeteksi Port B, bit mana saja yang berlogika 1 dan 0 (ON dan OFF)

Nilai properti untuk Timer 1 adalah sebagai berikut :

Interval = 100 , dan Enabled: True.

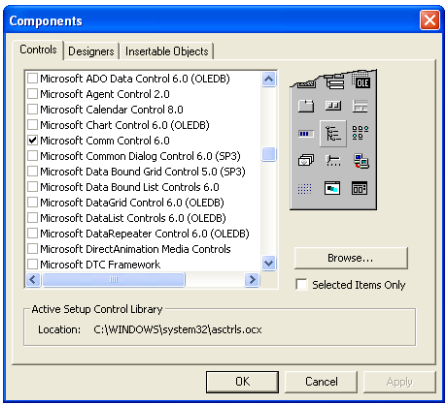
Kode untuk Timer1 mencakup pengolahan data ADC, dan satu byte pengolahan nilai Port B. Selengkapnya adalah sebagai berikut :

```
Private Sub Timer1_Timer()
If MSComm1.PortOpen = True Then
    MSComm1.Output = "C"
    If MSComm1.InBufferCount <> 0 Then
        DataMasuk() = MSComm1.Input
        Voltage0 = (((DataMasuk(1) And 3) * 256) + DataMasuk(0)) / 204.6
        ADC0_txt.Text = Format$(Voltage0, "#.00")
        ADC0.Value = Voltage0 * 4096 / 5
        Voltage1 = (((DataMasuk(3) And 3) * 256) + DataMasuk(2)) / 204.6
        ADC1_txt.Text = Format$(Voltage1, "#.00")
        ADC1.Value = Voltage1 * 4096 / 5
        Voltage2 = (((DataMasuk(5) And 3) * 256) + DataMasuk(4)) / 204.6
        ADC2_txt.Text = Format$(Voltage2, "#.00")
        ADC2.Value = Voltage2 * 4096 / 5
        Voltage3 = (((DataMasuk(7) And 3) * 256) + DataMasuk(6)) / 204.6
        ADC3_txt.Text = Format$(Voltage3, "#.00")
        ADC3.Value = Voltage3 * 4096 / 5
        Nilai_PortB = DataMasuk(8)
        B0_txt.Text = Nilai_PortB Mod 2
        Mod_B = Nilai_PortB Mod 2
        B0_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
        Nilai_PortB = Nilai_PortB / 2
        End If

        Mod_B = Nilai_PortB Mod 2
        B1_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
        Nilai_PortB = Nilai_PortB / 2
        End If
        Mod_B = Nilai_PortB Mod 2
        B2_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
        Nilai_PortB = Nilai_PortB / 2
        End If
        Mod_B = Nilai_PortB Mod 2
        B3_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
        Nilai_PortB = Nilai_PortB / 2
        End If
        Mod_B = Nilai_PortB Mod 2
        B4_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
        Nilai_PortB = Nilai_PortB / 2
        End If
        Mod_B = Nilai_PortB Mod 2
        B5_txt.Text = Mod_B
        If Mod_B = 1 Then
        Nilai_PortB = (Nilai_PortB / 2) - 0.5
        Else
```

```
            Nilai_PortB = Nilai_PortB / 2
            End If
            Mod_B = Nilai_PortB Mod 2
            B6_txt.Text = Mod_B
            If Mod_B = 1 Then
            Nilai_PortB = (Nilai_PortB / 2) - 0.5
            Else
            Nilai_PortB = Nilai_PortB / 2
            End If
            B7_txt.Text = Nilai_PortB
        End If
      End If
    End Sub
```

3) MsComm1 (MsComm)

MsComm merupakan komponen yang berfungsi untuk mengatur keberadan MODEM melalui PortCOM (standard RS232), sehingga kita juga bisa memanfaatkan keberadaannya untuk mengatur komunikasi data RS232. Komponen MsComm dapat dimunculkan jika komponen **Microsoft Comm Control 6.0** sudah terdaftar dalam lingkungan **VB ToolBox** kita. Jika belum, klik kanan pada menu **Toolbar** kemudian akan ada pilihan untuk memasukkan komponen, berilah tanda centang pada **Microsoft Comm Control 6.0** seperti Gambar 4.21. Klik tombol OK , maka di ToolBar akan muncullambang MSComm dan kita bisa menggunakannya untuk menyeting konfigurasi port COM/MODEM.



Gambar 4.21. Tap komponen MSComm

4) COM_Number_txt (textBox)

Berfungsi untuk menunjukkan nilai Port Com yang dipilih oleh *scrollbar*. Nilai yang ditampilkan pada TextBox ini akan tergantung dari nilai HScroll1.

5) HScroll1 (HScroll1)

HSroll1 merupakan komponen untuk menentukan Port Com yang akan dipakai.
Nilai minimal untuk HScroll1 1 dengan nilai maksimumnya 15 yang merepresentasikan nomor Port COM yang digunakan. Scipt kode untuk HScroll1 adalah sebagai berikut:

```
Private Sub HScroll1_Change()
Nilai_Port_COM = HScroll1.Value
COM_Number_txt.Text = Nilai_Port_COM
End Sub
```

6) Konek_btn (button)

Konek_btn merupakan button yang berfungsi untukmembangkitkan even (kejadian) agar

aplikasi mengirimkan karakter “A” ke devise melalui PortCOM. Dalam rutin komponen ini juga dicek karakter hasil balasan dari device. Jika “B” maka device dapat dikategorikan terhubung, jika selain “B” atau tidak ada balasan, maka device belum terhubung. Error handling diperlukan untuk menangani kasus yang berhubungan dengan PortCOM. Properti Caption untuk button ini adalah “ Konek ke PIC 18F4550”. Sedangkan kode untuk button ini adalah sebagai berikut :

```
Private Sub Konek_btn_Click()
On Error GoTo ErrorHandler

If MSComm1.PortOpen = True Then
HScroll1.Enabled = True
MSComm1.PortOpen = False
Status_lbl.Caption = "Diskonek dengan PIC"
Konek_btn.Caption = " Konek ke PIC "
Else
    MSComm1.CommPort = Nilai_Port_COM
    MSComm1.InputLen = 0
    MSComm1.InputMode = comInputModeBinary
    MSComm1.PortOpen = True                    MSComm1.Output = "A"
    Delay = Timer()
    While Timer() - Delay < 0.1
    Wend
    If MSComm1.InBufferCount <> 0 Then
        DataMasuk() = MSComm1.Input
        If DataMasuk(0) = &H42 Then 'PIC reply "B" ?
        Status_lbl.Caption = "Terhubung dengan PIC"
        Konek_btn.Caption = "Disconnect"
        HScroll1.Enabled = False
        D0_cxb_Click
        D1_cxb_Click
        D2_cxb_Click
        D3_cxb_Click
        D4_cxb_Click
        D5_cxb_Click
        D6_cxb_Click
        D7_cxb_Click
        End If
    Else
    HScroll1.Enabled = True
    Status_lbl.Caption = "Not Detected"
    MSComm1.PortOpen = False
    End If
End If
Exit Sub

ErrorHandler:
    MsgBox "Error :" & vbCrLf & Err.Description
End Sub
```

7) Status_lbl (label)

Status_lbl merupakan komponen label yang menunjukkan keterangan apakah aplikasi terhubung dengan device atau tidak, berdasarkan respon balasan karakter “B” dari host.

8) D0_cxb … D7_cxb (combobox)

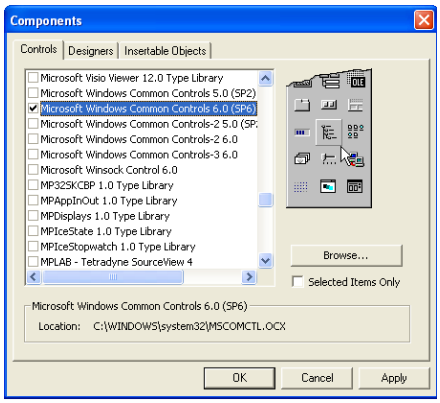
D0_cxb sampai D7_cxb merupakan combo box yangberjumlah 8 buah yag berfungsi untuk menyalakan dan mematikan LED pada Port D. Berikut kode untuk D0_cbx, D1_cbx , D2_cbx . . . sampai D7_cbx:

```
Private Sub D0_cxb_Click()
If MSComm1.PortOpen = True Then
If D0_cxb.Value = 1 Then
MSComm1.Output = "P"
Else
MSComm1.Output = "p"
End If
End If
End Sub
```

```
Private Sub D1_cxb_Click()
If MSComm1.PortOpen = True Then
If D1_cxb.Value = 1 Then
MSComm1.Output = "Q"
Else
MSComm1.Output = "q"
End If
End If
End Sub

Private Sub D2_cxb_Click()
If MSComm1.PortOpen = True Then
If D2_cxb.Value = 1 Then
MSComm1.Output = "R"
Else
MSComm1.Output = "r"
End If
End If
End Sub

Private Sub D3_cxb_Click()
If MSComm1.PortOpen = True Then
If D3_cxb.Value = 1 Then
MSComm1.Output = "S"
Else
MSComm1.Output = "s"
End If
End If
End Sub

Private Sub D4_cxb_Click()
If MSComm1.PortOpen = True Then
If D4_cxb.Value = 1 Then
MSComm1.Output = "T"
Else
MSComm1.Output = "t"
End If
End If
End Sub

Private Sub D5_cxb_Click()
If MSComm1.PortOpen = True Then
If D5_cxb.Value = 1 Then
MSComm1.Output = "U"
Else
MSComm1.Output = "u"
End If
End If
End Sub

Private Sub D6_cxb_Click()
If MSComm1.PortOpen = True Then
If D6_cxb.Value = 1 Then
MSComm1.Output = "V"
Else
MSComm1.Output = "v"
End If
End If
End Sub

Private Sub D7_cxb_Click()
If MSComm1.PortOpen = True Then
If D7_cxb.Value = 1 Then
MSComm1.Output = "W"
Else
MSComm1.Output = "w"
End If
End If
End Sub
```

9) ADC0 … ADC3 (ProgressBar)

Progress Bar berfungsi untuk mempercantik halaman aplikasi. Jika komponen progressBar belum ada, cara untuk menampilkan adalah dengan Klik kanan pada jendela **Tool** lalu masuk ke menu **Component** lalu beri tanda centang **Microsoft Windows Common Control 6.0 (SP6)** seperti Gambar 4.20. Setelah itu, klik OK dan Progreass Bar akan muncul di ToolBox VB 6. Nilai minimal properti komponen ini adalah 0 dan nilai maksimalnya 5120.



Gambar 4.20. Tap komponen ProgressBar

10) ADC0_txt … ADC3_txt (textBox)

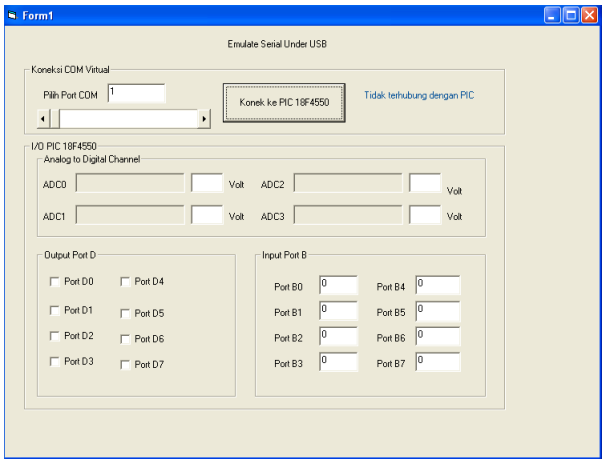
ADC0_txt, ADC1_txt, ADC2_txt, dan ADC3_txt merupakan komponen textBox yang menunjukkan berapa nilai ADC yang terbaca. Nilainya ditentukan pada even Timer1.

11) B0_txt … B7_txt (textBox)

B0_txt, B1_txt, sampai B7_txt merupakan Text Box yang menunjukkan apakah nilai portB yang bersangkutan bernilai 1 atau 0. Nilai kedelapannya ditentukan pada *even* Timer1. Nilai awal properti text untuk komponen ini dikosongkan.

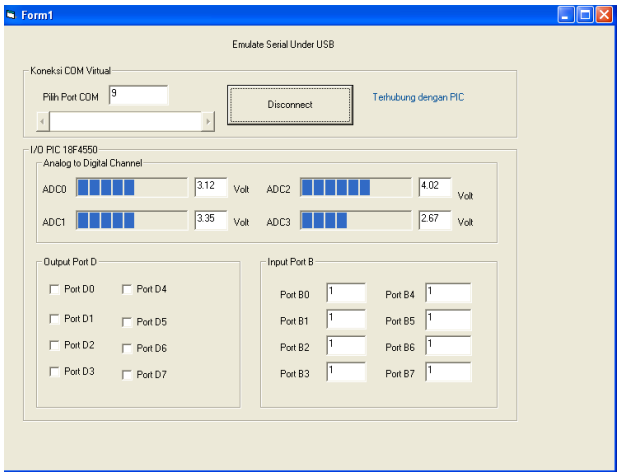
HASIL AKHIR APLIKASI

Tampilan akhir ketika program VB 6 dijalankan adalah seperti Gambar 4.21. Untuk mengkonekkan dengan device, user harus memilih nilai Port COM yang bersesuaian melalui *scroll bar*, lalu menekan tombol "Konek ke PIC 18F4550". Nilai Port COM dapat dilihat pada *Device Manager*. (dalam contoh kali ini adalah COM 9)



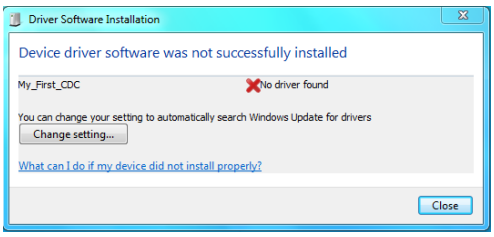
Gambar 4.21. Hasil akhir tampilan aplikasi sebelum konek dengan PIC18F4550

Proses koneksi yang sukses menghasilkan pembacaan nilai Analog 4 saluran, dan nilainya dapat diatur melalui simulasi VR (variable resistor) pada Port A0 sampai Port A3. Nilai PortB sebagai input juga tertampil bit per bit pada komputer, sehingga dapat dipantau langsung oleh user. Sebagai kontrol pada Port D sebanyak 8 saluran, user dapat memerintahkan Port D bit mana saja yang hidup dan mati melalui aplikasi Windows.

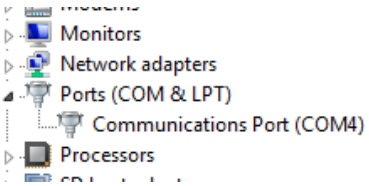


Gambar 4.22. Hasil akhir tampilan aplikasi setelah konek dengan PIC18F4550

Pada edisi Windows 7, identifikasi driver juga tidak berbeda jauh dengan versi Windows XP. Seperti Gambar 4.23, pertama kali Windows mendeteksi device dan meminta dukungan driver. Pada *Device Manager* klik kanan pada device yang masih diketahui sebagai device yang tidak dikenal, lalu lakukan update driver, arahkan ke folder **CDC_ku_driver.inf.** Proses identifikasi driver yang berhasil akan mengidentifikasi keterangan pada **CDC_ku_driver.inf** berupa Communication Port. Pada *Device Manager* PIC18F4550 juga diidentifikasi sebagai Port COM4 (Gambar 4.24).



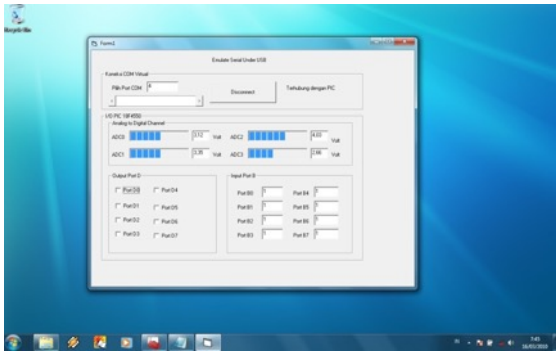
Gambar 4.23. Permintaan driver pada Windows 7



Gambar 4.24. Device Manager mendeteksi keberadaan Port COM (COM4)

Jika *Device Manager* telah berhasil mendeteksi keberadaan peripheral dengan benar, maka aplikasi yang akan berjalan di atas Windows 7 tinggal menyesuaikan dengan lingkungan Sistem

Operasi yang bersangkutan.



Gambar 4.25. Proses aplikasi yang berjalan di atas platform windows 7.

Pada Gambar 4.25 aplikasi berjalan komunikasi yang berjalan antara host-device tidak mengalami masalah yang berarti. Namun kita harus membuat agar komponen MSComm terdaftar di lingkungan Windows 7.