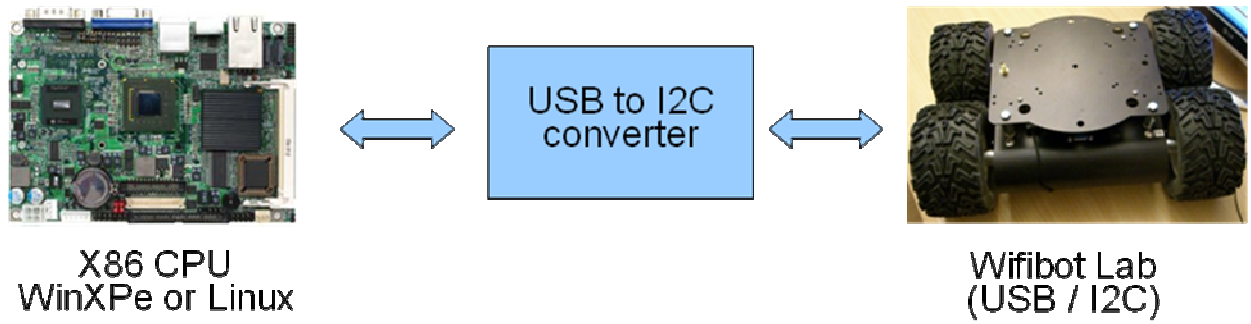
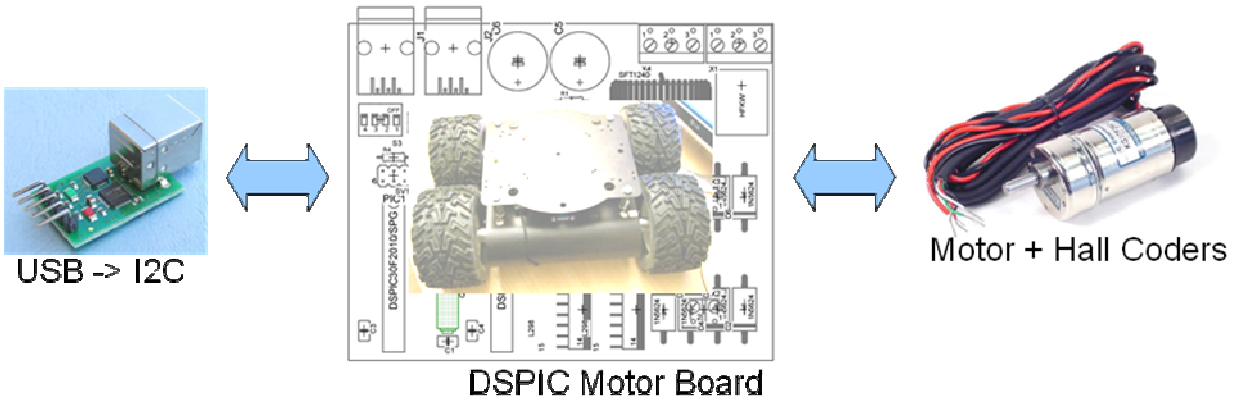
LOW LEVEL PROTOCOLE DUAL DSPIC 30F

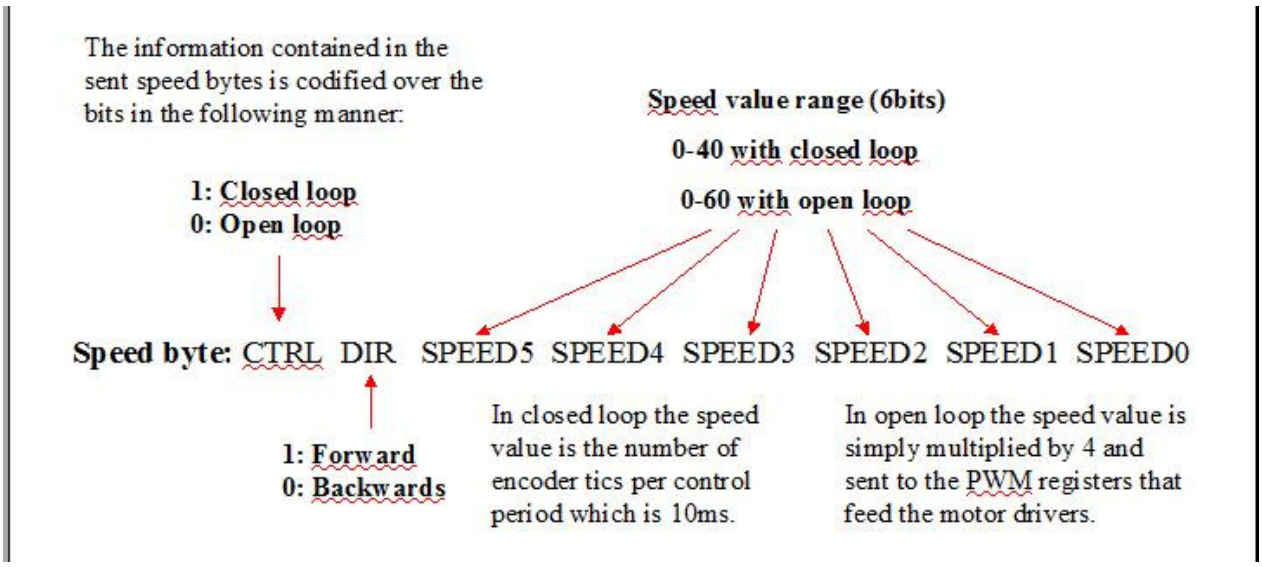
To control the chassis you need to send to USB/I2C adapter some special char that will be forwarded to the DSPICS that controls the motors. To get the data from the encoders wheel, it is the same you need to receive from the DSPICS via the USB/I2C adapter some char to read the IR sensors value, the battery level, the speed (in tics) and the accumulated Odometry (in tics).

Sending or receiving data from the USB/I2C adapter is only a matter of sending and receiving chars on a RS232 serial port at 19200 bauds. This task is achieved on the embedded CPU running Linux or windows.



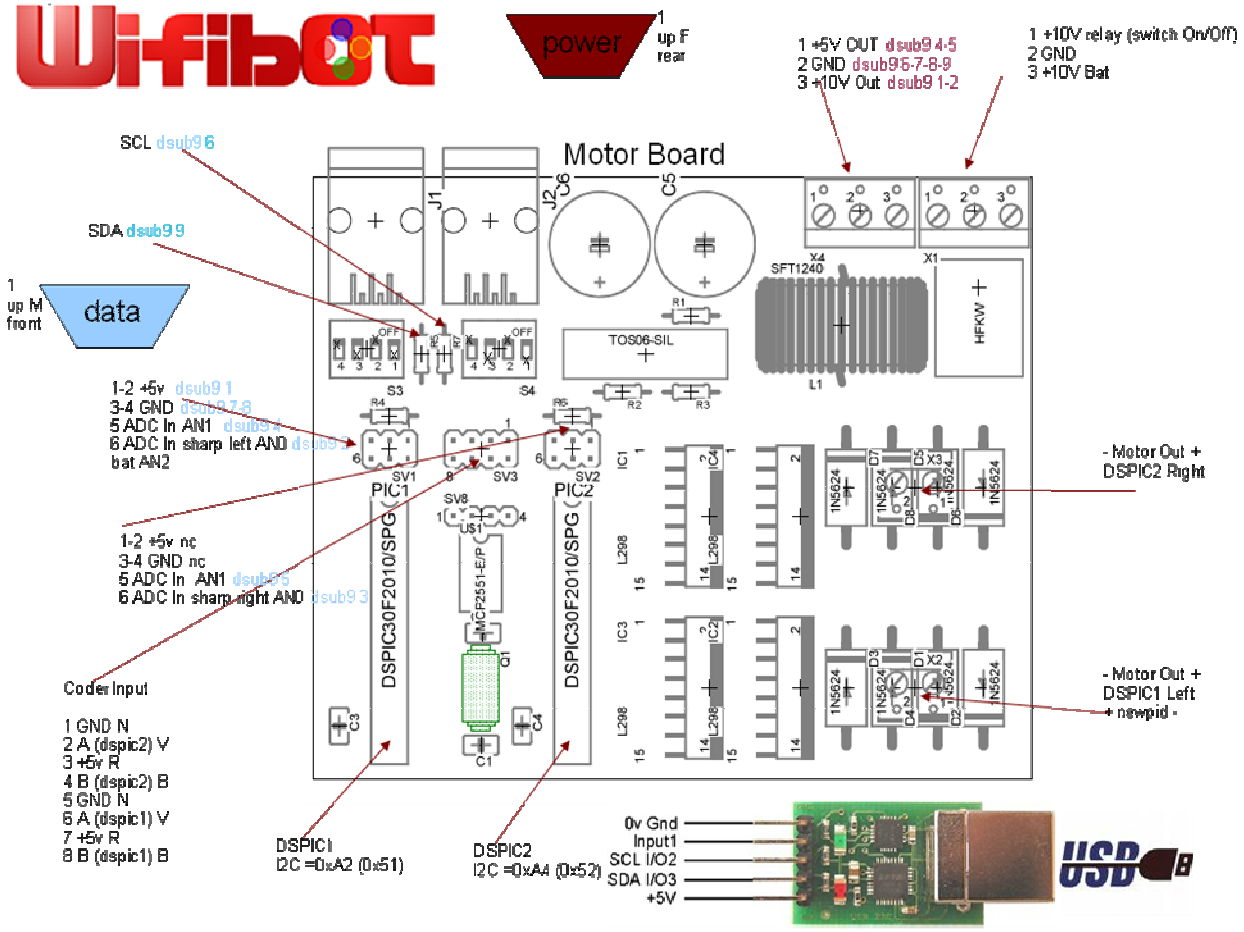


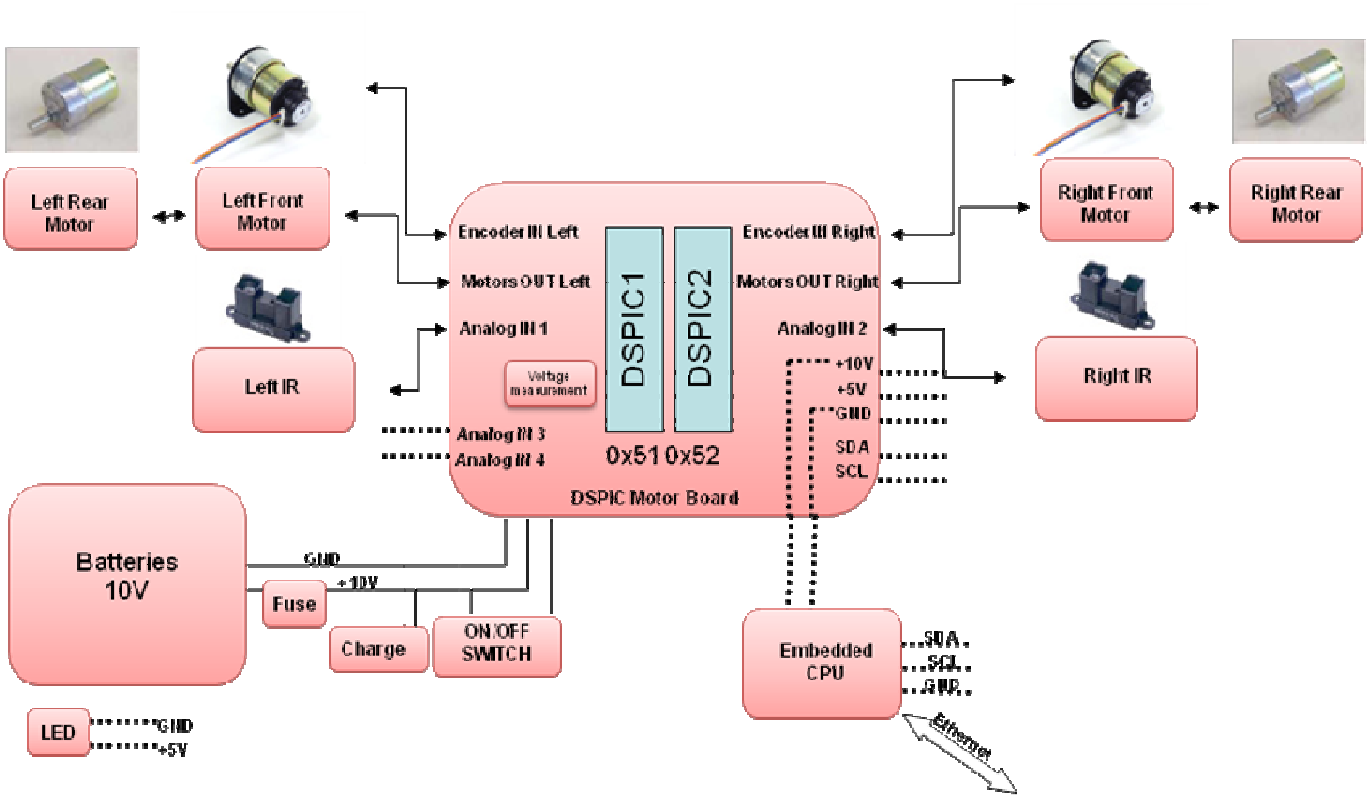
Controlling the motors:



The DSPIC 30F board has 2 dspics that controls each side of the robot:

DSPIC Left I2C address: 0xA2
DSPIC Right I2C address: 0xA4





SET SPEED or PID:

So to control for example the left side (Left 2 motors, 1 encoders) :

The speed is between 0-60 (0-40 effective)

If we want the left side to move at speed 20 forward without motor control we send:
128*0 + 64*1 + 20*1 = 84

```
SetI2cMotor(hUSB,0xA2,84);
```

If we want the left side to move at speed 20 backward with motor control we send:

128*1 + 64*0 + 20*1 = 148

```
SetI2cMotor(hUSB,0xA2,148);
```

```
//hUSB is the serial port handle opened at 19200 baud
int SetI2cMotor(HANDLE hUSB, BYTE add,BYTE speed)
{
	DWORD n;
	BYTE sbuf[5];

	sbuf[0] = I2C_CMD;
	sbuf[1] = add;
	sbuf[2] = 0x00;
	sbuf[3] = 0x01;
	sbuf[4] = speed;
	WriteFile(hUSB, &sbuf, 5, &n, NULL);
	ReadFile(hUSB, &sbuf, 1, &n, NULL);
	return sbuf[0];
}
```

If we want to set a new PID on the left side at P=0.55 I=0.01 D=0.28

```
SetI2cMotorPID(hUSB,0xA2,0x00,55,1,28);

//hUSB is the serial port handle of I2C USB module opened at 19200 baud
int SetI2cMotorPID(HANDLE hUSB, BYTE add,BYTE speed,BYTE pp,BYTE ii,BYTE
dd)
{
      DWORD n;
      BYTE sbuf[8];

      sbuf[0] = I2C_CMD;
      sbuf[1] = add;
      sbuf[2] = 0x00;
      sbuf[3] = 0x04;
      sbuf[4] = speed;
      sbuf[5] = pp;
      sbuf[6] = ii;
      sbuf[7] = dd;
      WriteFile(hUSB, &sbuf, 8, &n, NULL);
      ReadFile(hUSB, &sbuf, 1, &n, NULL);
      return sbuf[0];
}
```

GET Chassis DATA:

Each DSPICS side sends you 8 chars:

```
bufsend[0]=(unsigned char)speedlab;//speed = tics / 10 ms
bufsend[1]=(unsigned char)(AN_Bat/10);//10,9v = 109
bufsend[2]=(unsigned char)(AN_Sharp/10); //2,4v->50
bufsend[3]=(unsigned char)(AN_Sharp2/10); //2,4v->50
bufsend[4]=bufposition[0];//odometry data in long format 4 char
bufsend[5]=bufposition[1];//12ppr x 4 x 51 gerar box = 2448 tics/wheel turn
bufsend[6]=bufposition[2];
bufsend[7]=bufposition[3];
```

If you want to receive for the dspic 0xA2 :

```
SensorData dataLeft = GetI2cMotor(hUSB,0xA2);

struct SensorData
      {
      int SpeedFront;
      int BatLevel;
      int IR;
      int IR2;
      long odometry;
      };
```

```
struct SensorData GetI2cMotor(HANDLE hUSB, BYTE add)
{
	DWORD n;
	BYTE sbuf[8];//unsigned char for LINUX
	struct SensorData data;

	sbuf[0] = I2C_CMD;// send command
	sbuf[1] = add+1;
	sbuf[2] = 0x00;
	sbuf[3] = 0x08;
	WriteFile(hUSB, &sbuf, 4, &n, NULL);
	ReadFile(hUSB, &sbuf, 8, &n, NULL);
	data.SpeedFront=sbuf[0];
	data.SpeedRear=sbuf[1];
	data.IR=sbuf[2];
	data.IR2=sbuf[3];
	data.odometry=((((long)sbuf[7] << 24))+(((long)sbuf[6] <<
	16))+(((long)sbuf[5] << 8))+((long)sbuf[4]));
	return data;
}
```

OPEN SERIAL PORT (WINDOWS):

```
HANDLE SetupI2CCommPort( LPCSTR comport)
{
	HANDLE hCom;
	DCB dcb;
	COMMTIMEOUTS ct;

	hCom = CreateFileA( comport, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, 0, 0);
	GetCommState(hCom, &dcb);
	dcb.BaudRate = RS232_SPEED;//19200 bauds
	dcb.fParity = FALSE;
	dcb.fOutxCtsFlow = FALSE;
	dcb.fOutxDsrFlow = FALSE;
	dcb.fDtrControl = DTR_CONTROL_DISABLE;
	dcb.fDsrSensitivity = FALSE;
	dcb.fOutX = FALSE;
	dcb.fInX = FALSE;
	dcb.fRtsControl = RTS_CONTROL_DISABLE;
	dcb.fAbortOnError = FALSE;
	dcb.ByteSize = 8;
	dcb.Parity = NOPARITY;
	dcb.StopBits = TWOSTOPBITS;
	SetCommState(hCom, &dcb);

	GetCommTimeouts(hCom, &ct);
	ct.ReadIntervalTimeout = 500;
	ct.ReadTotalTimeoutMultiplier =500;
	ct.ReadTotalTimeoutConstant = 500;
	SetCommTimeouts(hCom, &ct);
	SetCommMask(hCom, EV_RXCHAR);
	return hCom;
}
```