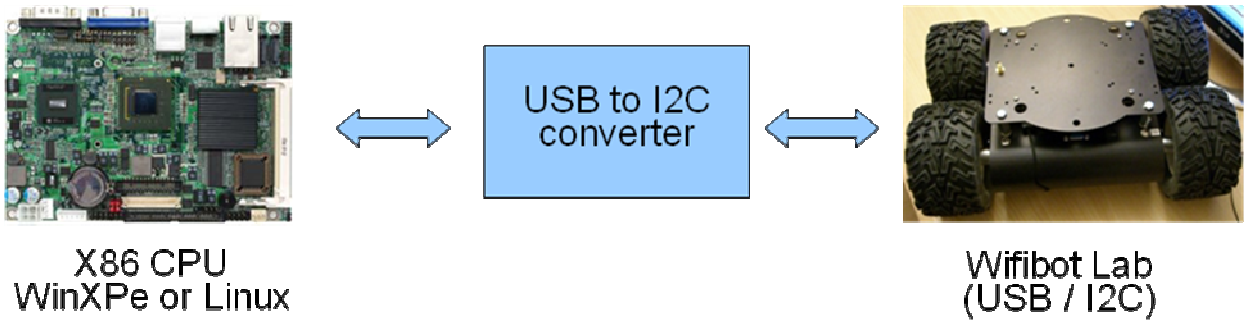
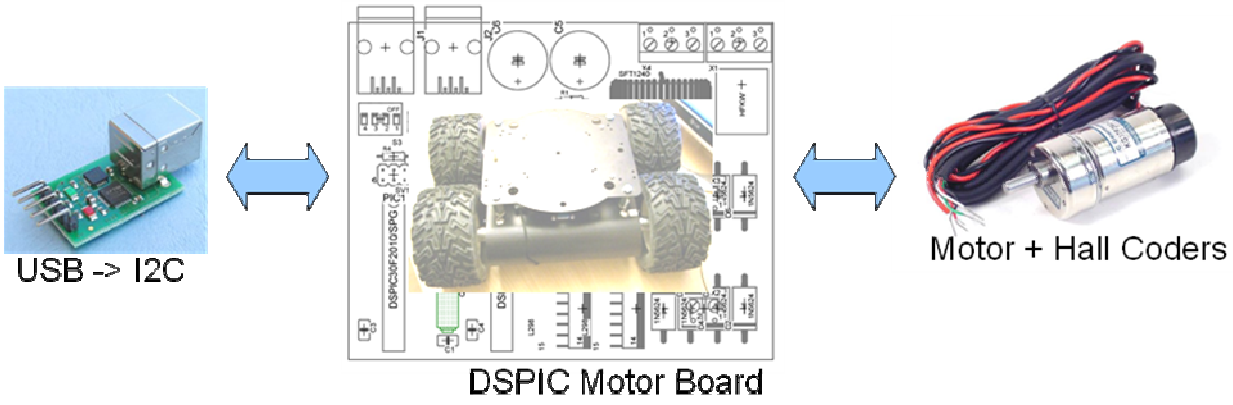
LOW LEVEL PROTOCOLE Single DSPIC 33F using I2C

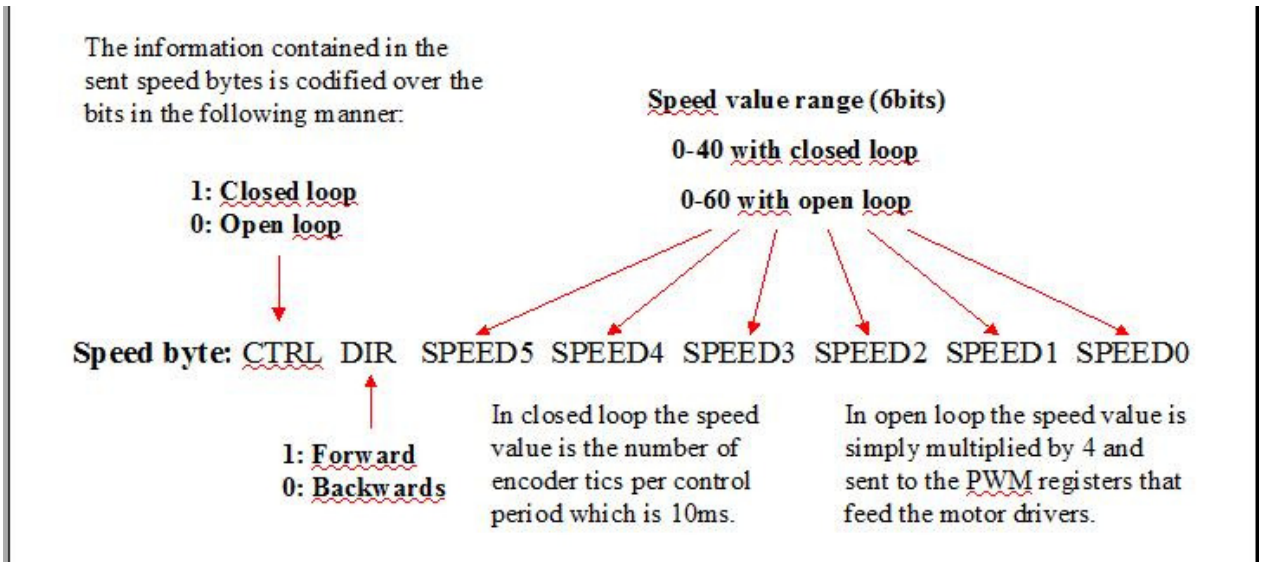
To control the chassis you need to send to the USB/I2C adapter some special char that will be forwarded to the DSPIC 33F that controls the motors. To get the data from the encoders wheel, it is the same you need to receive from the DSPICS via the USB/I2C adapter some char to read the IR sensors value, 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.

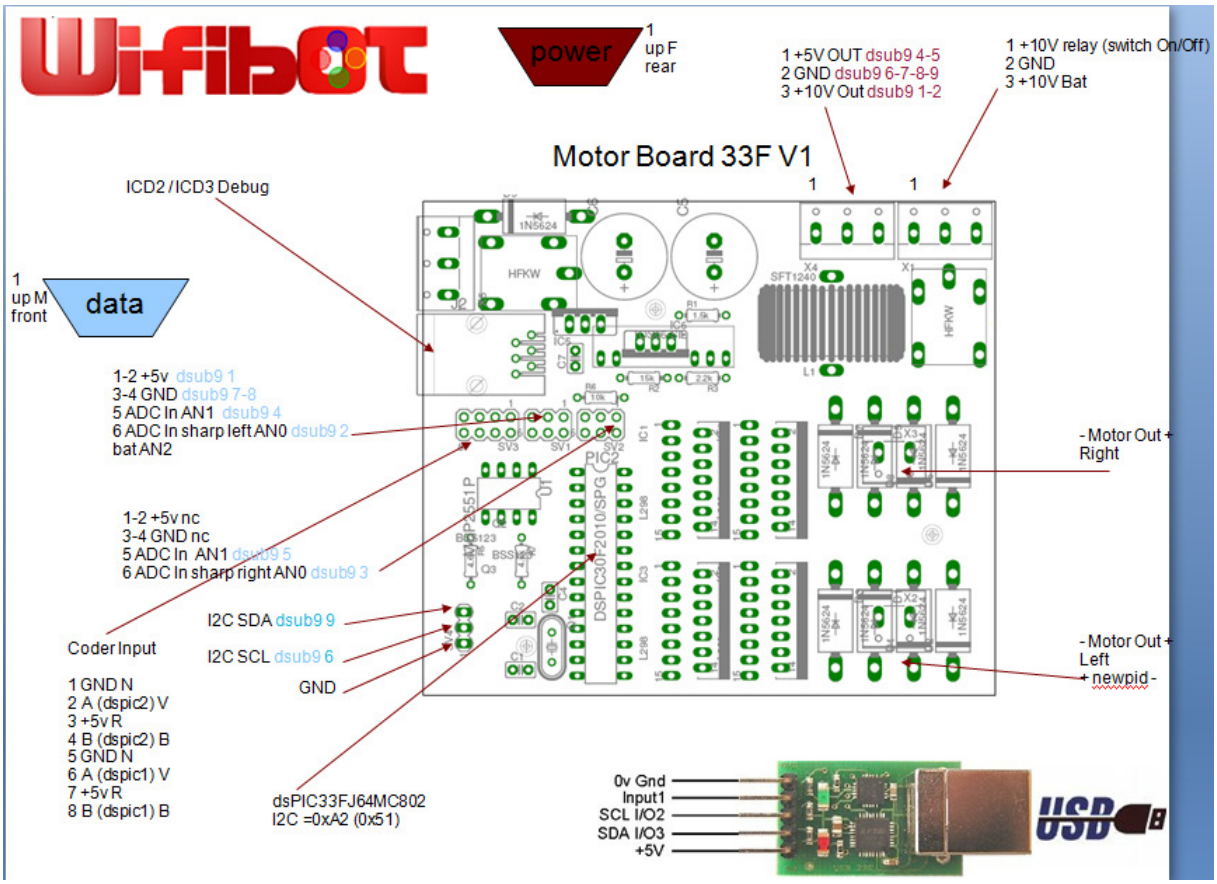


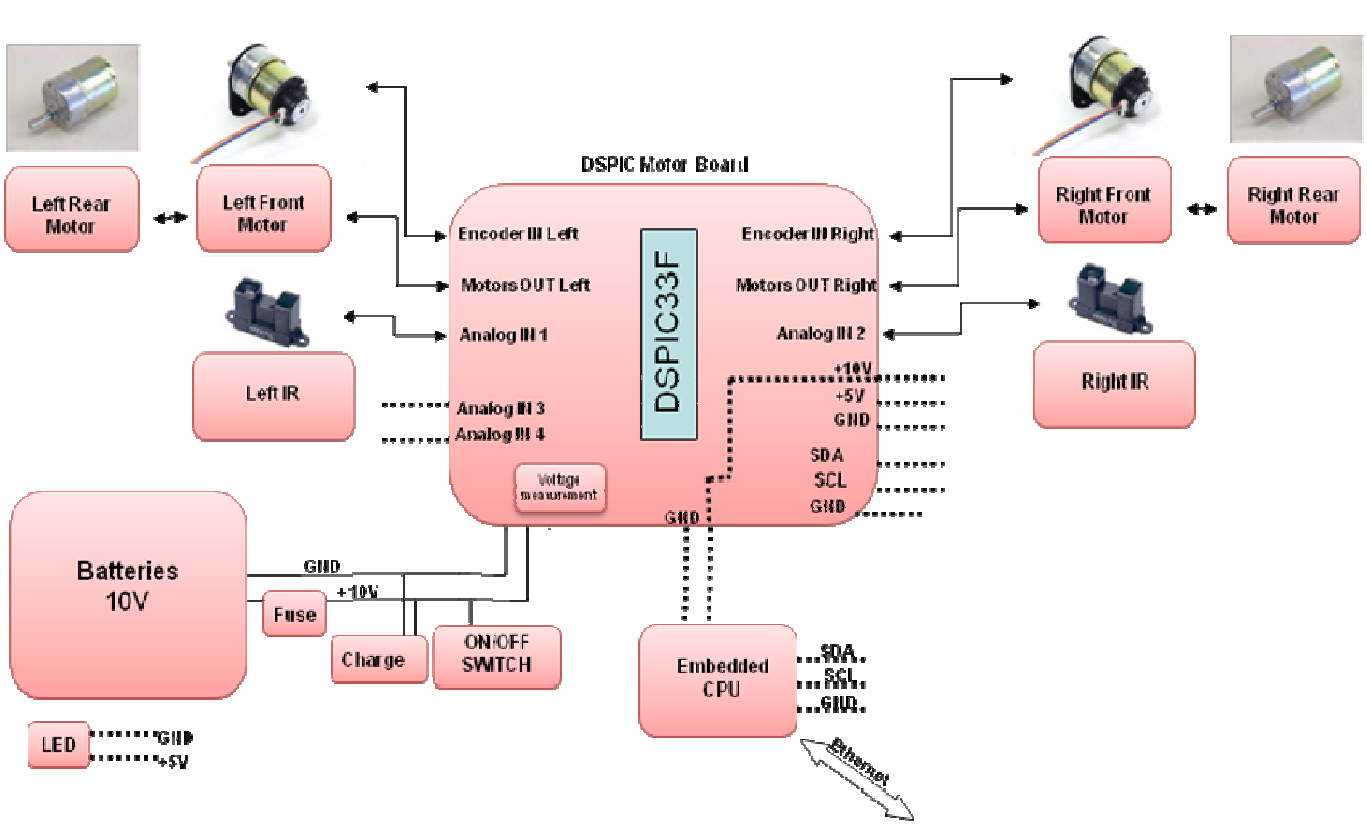


Controlling the motors:



DSPIC 33F: I2C address = 0xA2





SET SPEED or PID:

So to control for example the left side and the right side (Left & Right 2 motors, 2 encoders):

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

If we want the left side & right side to move at speed 20 forward without motor control we send:

128*0 + 64*1 + 20*1 = 84

```
SetI2cMotor33f(hUSB,0xA2,84,84);
```

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

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

```
SetI2cMotor33f(hUSB,0xA2,148,148);
```

```
//hUSB is the serial port handle opened at 19200 baud
int SetI2cMotor33f(HANDLE hUSB, BYTE add,BYTE speed1,BYTE speed2)
{
	DWORD n;
	BYTE sbuf[10];

	sbuf[0] = I2C_CMD;
	sbuf[1] = add;
	sbuf[2] = 0x00;
	sbuf[3] = 0x02;
	sbuf[4] = speed1;
	sbuf[5] = speed2;
	WriteFile(hUSB, &sbuf, 6, &n, NULL);
	ReadFile(hUSB, &sbuf, 1, &n, NULL);
	return sbuf[0];
}
```

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

```
SetI2cMotorPID33f(hUSB,0xA2,0x00,0x00,55,1,28);
int SetI2cMotorPID33f(HANDLE hUSB, BYTE add,BYTE speed1,BYTE speed2,BYTE pp,BYTE ii,BYTE dd)
{
	DWORD n;
	BYTE sbuf[10];

	sbuf[0] = I2C_CMD;
	sbuf[1] = add;
	sbuf[2] = 0x00;
	sbuf[3] = 0x05;
	sbuf[4] = speed1;
	sbuf[5] = speed2;
	sbuf[6] = pp;
	sbuf[7] = ii;
	sbuf[8] = dd;
	WriteFile(hUSB, &sbuf, 9, &n, NULL);
	ReadFile(hUSB, &sbuf, 1, &n, NULL);
	return sbuf[0];
}
```

GET DATA:

The DSPIC sends you 15 chars:

```
bufsend[0]=(unsigned char)(-speedlab); //tics / 10 ms
bufsend[1]=(unsigned char)(tmpadc2 >> 2);//10.1V 1.28V 404 /4 -> 101
bufsend[2]=(unsigned char)(tmpadc4 >> 2);//3.3v->255 2v->156
bufsend[3]=(unsigned char)(tmpadc3 >> 2);//3.3v->255 2v->156
bufsend[4]=bufposition[0];// acumulated odometrie
bufsend[5]=bufposition[1];//12 ppr x 4 x 51 gear box = 2448 tics/wheel turn
bufsend[6]=bufposition[2];
bufsend[7]=bufposition[3];
bufsend[8]=(unsigned char)speedlab2;
bufsend[9]=(unsigned char)(tmpadc0 >> 2);//3.3v->255 2v->156
bufsend[10]=(unsigned char)(tmpadc1 >> 2);//3.3v->255 2v->156
bufsend[11]=bufposition2[0];
bufsend[12]=bufposition2[1];
bufsend[13]=bufposition2[2];
bufsend[14]=bufposition2[3];
```

You receive:

```
GetI2cMotor33f(hUSB,0xA2,&dataL,&dataR);

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

```
int GetI2cMotor33f(HANDLE hUSB, BYTE add,SensorData *dataL,SensorData
*dataR)
{
	DWORD n;
	BYTE sbuf[20];

	sbuf[0] = I2C_CMD;// send command
	sbuf[1] = add+1;
	sbuf[2] = 0x00;
	sbuf[3] = 0x0f;
	WriteFile(hUSB, &sbuf, 4, &n, NULL);
	ReadFile(hUSB, &sbuf, 15, &n, NULL);
	dataL->SpeedFront=sbuf[0];
	dataL->BatLevel=sbuf[1];
	dataL->IR=sbuf[2];
	dataL->IR2=sbuf[3];
	dataL->odometry=((((long)sbuf[7] << 24))+(((long)sbuf[6] <<
16))+(((long)sbuf[5] << 8))+((long)sbuf[4]));

	dataR->SpeedFront=sbuf[8];
	dataR->BatLevel=0;
	dataR->IR=sbuf[9];
	dataR->IR2=sbuf[10];
	dataR->odometry=((((long)sbuf[14] << 24))+(((long)sbuf[13] <<
16))+(((long)sbuf[12] << 8))+((long)sbuf[11]));
	return 1;
}
```

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;
}
```