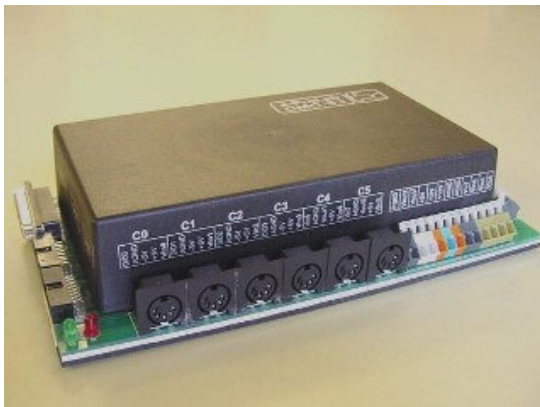


Programming CMC-S3 and H-bridge systems in Borland Delphi

Author:
Slavko Kocijancic
University of Ljubljana, Faculty of Education

20th December 2004

Prepared within the e-ProLab initiative and EU project ComLab:
<http://e-prolab.com/comlab>



CMC-S3 data acquisition system



H-bridge system

Contents

CMC-S3 data acquisition system and H-bridge control system	2
Programming libraries for the CMC-S3 and H-bridge systems	3
First application with Borland Delphi	3
Procedures for programming the IO systems	3
Starting a new project	7

CMC-S3 data acquisition system and H-bridge control system

The CMC-S3 and H-bridge systems are electronics devices for PCs with input and output (IO) functions.

The CMC-S3 system is more complex since it has 24 digital IO lines, 8 analogue inputs (12 bit resolution) and 2 analogue outputs (8-bit resolution). More details are available at:

<http://e-prolab.com/comlab/lowcdaq/lowcdaq.htm>

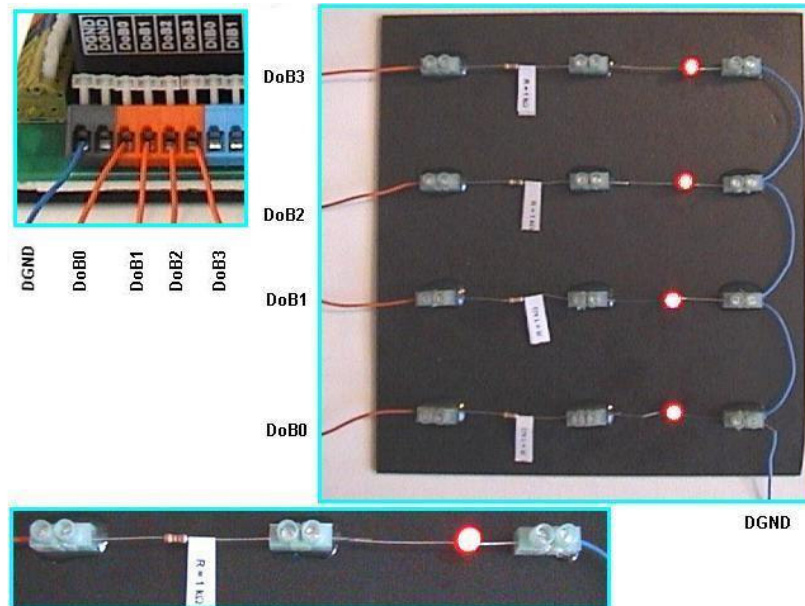
The H-bridge system has 8-bit digital output and 4-bit digital input. Digital output has amplified voltage for logic level 1 (about 10 V or more). It can source and sink current up to 0.5 A. It can be used therefore to control small DC motors, stepper motors, bulbs, et. The H-bridge can be directly connected to the PC's parallel port as well as to the CMC-S3 system via a special flat cable. More details are available at:

<http://e-prolab.com/comlab/hbridge/hbridge.htm>

It is recommended to have the PC switched-off before connecting the IO system to it.

The H-bridge system needs to be first connected with the PC with the enclosed 25-pin parallel port cable. Then connect it to the power supply (AC or DC adaptor) when the red LED at the left-front corner starts to light.

The CMC-S3 system also needs to be first connected with the PC with the enclosed 25-pin parallel port cable. Then connect it to the power supply (12 V AC adaptor) when the red and green LEDs at the left-front corner starts to light. For the examples below it is assumed, that H-bridge system is connected to Dig C connector of the CMC-S3 or that H-bridge system is "stand-alone", that is connected directly to the PC. If having the CMC-S3 and not the H-bridge, one can use digital inputs and outputs available at the right side of the system marked with DoBx (outputs) and DiBx (inputs). Connect LEDs to digital output as shown below.



Programming libraries for the CMC-S3 and H-bridge systems

Besides Borland Delphi, versions 3 and above, one needs to install drivers and programming library named BDLibePL available free for downloading at the *ComLab* website:

<http://e-prolab.com/comlab/download.htm>.

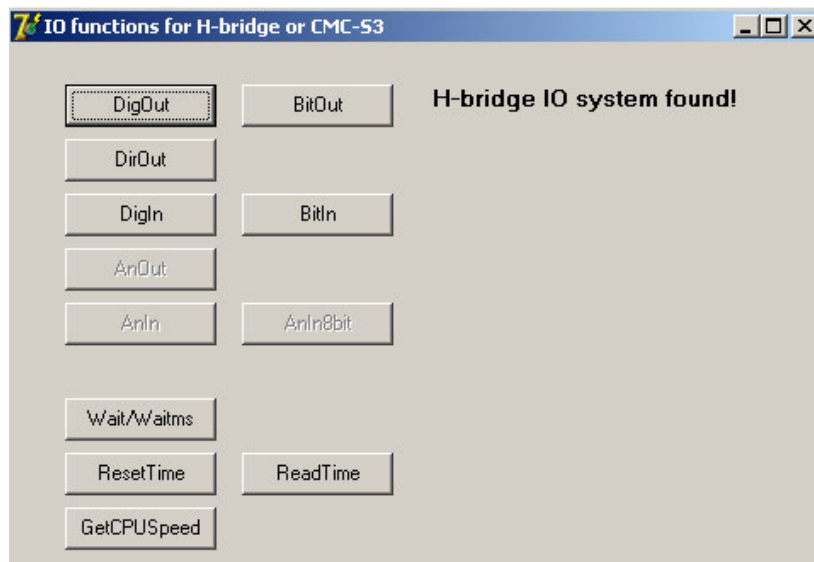
Save the setup file BDLibePL-v1_0-en.exe to a temporary folder and open it afterwards. The standard setup process starts. The default destination path for the installed files is

C:\eProLab\BDLibePL.

First application with Borland Delphi

Run Borland Delphi.

For a start, open an example project File->Open Project and browse for BDLibePLExample.dpr. Copmpile and run the project through Run-> Run, shortcut is F9. If everything goes fine, the window entitled **IO functions for...** appears. If H-bridge system is connected to the PC, in the right-upper portion of the window there should be a label "H-bridge IO system found". The AnOut, AnIn and AnIn8bit buttons are disabled since H-bridge has no analogue functions. If CMC-S3 system is connected to the PC, the label "CMC-S3 IO system found" should appear. If none of both IO systems are connected to the PC, the label displays "No IO system found!"



The following test should work only in the case that H-bridge system is directly connected to the PC or the CMC-S3 system is connected to the PC and H-bridge is connected to CMC-S3. After clicking the DigOut button, two LEDs should be turned-on, Do0 and Do2. Click then BitOut and the LEDs Do6 in Do7 should alternatively light ten times while Do0 in Do2 LEDs still light.

In any case, if either CMC-S3 or H-bridge system are connected to the PC and the power, and the label for IO system is not "No IO system found!", one can continue with the following examples. If not, the one should run the e-ProLab software and check the test module. The e-proLab software is also available at <http://e-prolab.com/comlab/download.htm>.

Procedures for programming the IO systems

For the following instructions it is assumed, that the user is familiar with the Borland Delphi (BD) at least on the basic level. There are several books as well as websites about BD.

Procedures developed especially for the H-bridge and CMC-S3 system are described in details. The source code for the procedures are collected in the module BDLibePL.pas. How to include the module to the new project is shown in the next section.

The use of the most important procedures is illustrated in the example project named BDLibePLExample.

Procedure InitIOePL;

The procedure needs to be called before using any procedure from the library. The best choice is to invoke it when the application is run!

The procedure has no parameters. Basically it checks which or the two IO systems is connected to the parallel port and determines the speed of PC's CPU clock for time related procedures. Type of the IO system is written to an enumerated variable IOePL, which gets one of the three values: CMC_S3, H_bridge or None.

It is suggested to call the InitIOePL within the FormCreate () procedure. One can create it by a mouse double click at the main form where no other object is placed. In this case, the only code that needs to be entered by the user is InitIOePL:

```
procedure TForm1.FormCreate(Sender: TObject);
  LibePL.InitIOePL;
End;
```

Procedure FindIOePL;

This procedure is called within the InitIOePL. It checks the type of the IO system connected to the PC. Type of the IO system is written to an enumerated variable IOePL, which gets one of the three values: CMC_S3, H_bridge or None (the last one is used when none of the both systems is connected or the power is not switched on).

Since the procedure is called within the InitIOePL, it is usually not needed to call it separately.

Procedure DigOut (Dig:TDig;nData:integer) ;

Procedure for the digital output..

The Dig parameter can be either DigA, DigB, DigC or DigD. For example, if H-bridge circuit is connected to Dig C of the CMC-S3, the Dig parameter is DigC. If H-bridge is connected directly to the PC, any choice of TDig would do. However, either DigC or DigD is recommended.

nData is 8-bit data (numbers from 0 to 255).

Example code:

```
LibePL.DigOut (DigC, 202);
```

On the digital output, the following bits will be switched on D1, D3, D6 in D7. 202 is binary written as 11001010, where D0 is on the right, and D7 on the left. The data that was on the digital output before the call is overwritten.

procedure BitOut (Dig:TDig;Dout:TDout;BitState:TBitState) ;

Also a procedure for the digital output, only that it influences only the logic level of one single bit, the same bits do not change – they keep their level according the same as they had before the call.

Dig can be either DigA, DigB, DigC ali DigD (se the DigOut procedure).

Bit is a mark of the bit which is addressed: Do0, Do1,... to Do7.

BitState marked with BitOn sets the bit to level 1 while BitOff sets it to 0.

Example code:

```
LibePL.BitOut (DigC, Do5, BitOn);
```

turns bit Do5 on Dig C to logic level 1.

procedure DirOut (Dig:TDig;DoutA:TDout;DoutB:TDout;DirState:TDirState) ;

The procedure is also related to the digital output. It influences the state of two bits, the rest of the bits keep the previous logic levels. It is designed to control DC and stepper motors, etc.

Dig is either DigA, DigB, DigC or DigD (see DigOut).
 DoutA is the first of the two bits concerned, marked with Do0, Do1,... to Do7.
 DoutB is the second bit marked with Do0, Do1,... to Do7.
 DirState determines one of the three possible values: DirPlus, DirMinus and DirOff.
 DirPlus sets level 0 to DoutA and 1 to DoutB (for example, a DC axis would rotate clockwise), DirMinus sets 1 to DoutA and 0 to DoutB (the DC axis would rotate anticlockwise) and DirOff sets both bits to 0 (the axis does not rotate).

Example code:

```
LibePL.DirOut (DigC, Do4, Do5, DirPlus);
```

sets bit Do5 to 1, Do4 to 0 and does not change the levels of other bits.

function DigIn(Dig:TDig):integer;

The function to apply the 4-bit digital input.
 Dig is either DigA, DigB, DigC or DigD (see DigOut).
 It returns a 4-bit decimal number from 0 do 15.

Example code:

```
nX := LibePL.DigIn(DigC)
```

The variable nX gets the value read from the figital input, port DigC. For example, if the bit Di2 ic connected to +5V, the nX would be 4.

function BitIn(Dig:TDig;Din:TDin):TBitState;

The function returns the level of chosen bit at the 4-bit digital input.
 Dig is either DigA, DigB, DigC ali DigD.
 Bit indicates which of the four bits is verified (Di0 to Di3).
 The function returns the logic level of the chosen bit which can be either BitOn or BitOff.

Example code:

```
If LibePL.BitIn(DigC, Di2) = BitOn Then
  Labell.Caption := 'ON'
Else
  Labell.Caption := 'OFF'
```

The code writes ON if bit Di1 on Dig C is logic 1 and OFF, if it is 0.

procedure AnOutI (Vout:TVout;nData:Integer) ;

The procedure for analogue output makes sense only for the CMC-S3 system since the H-bridge does not support analogue output.

Vout is one of the two channels, either Vout1 or Vout2.

nData is integer value from 0 do 255. The voltage on Vout1 is linearly dependent to the number within the interval from 0 to 5 V, while for Vout2 it is between -9.8 V and +9.8 V.

Example code:

```
LibePL.AnOutI (Vout1,100);
LibePL.AnoutI (Vout2,100);
```

would generate 1.96 V (=100*5V/255) at Vout1, while Vout2 would be -2.11 V (= -9.8+100*19.6/255).

procedure AnOut (Vout:TVout; dbVolt : double);

The procedure also makes sense only for the CMC-S3 system.

dbVolt is voltage represented as a real number. The value for Vout1 can be between 0 to 5 V, and for Vout2 between -9.8 V and +9.8 V.

Example code:

```
LibePL.AnOut (Vout1,2.5);
LibePL.Anout (Vout2,-3.5);
```

generates about 2.5 V at Vout1 and about -3.5 V at Vout2. Note that certain tolerance needs to be tolerated due to DA resolution and limited analogue accuracy.

function AnInI (Vin:TVin) : integer;

The function for analogue input can be used only with the CMC-S3, since the H-bridge system does not support the analogue input.

Vin is the input channel marked with Vin0 to Vin7.

The function returns an integer value from 0 to 4095, since the CMC-S3 has 12-bit analogue to digital converter. The obtained number is linearly dependent on the input voltage. The range of channels Vin0 to Vin5 is from 0 to 5V, the channels Vin6 and Vin7 the range is -9.8 V to +9.8 V.

Example code:

```
nX1 := LibePL.AnInI (Vin0);  
nX2 := LibePL.AnInI (Vin6);
```

say that the input voltage on channel Vin0 would be 1,520 V, the nX1 ideal value would be 1245 ($=1.520V * 4095/5V$). With -3,50 V on channel Vin6, the nX2 ideal value would be 1316 ($=(-3.50V+9.8V)*4095/19.6V$). Note that the accuracy of channels Vin6 and Vin7 is not that good as it is at the rest of the channels.

function AnIn (Vin:TVin) : double;

The function works with the CMC-S3 system.

Vin is the input channel marked with Vin0 to Vin7.

The function returns the voltage as real number. At Vin0 to Vin5 the value can be between 0 to 5V, channels Vin6 and Vin7 return from -9.8V to+9.8V.

Example code:

```
dbX1 := LibePL.AnIn (0);  
dbX2 := LibePL.AnIn (6);
```

dbX1 is a real number representing the voltage at Vin0, while dbX2 at Vin6.

function AnIn8bitI (Vin:TVin) : integer;

The function works with the CMC-S3 system.

Vin is the input channel marked with Vin0 to Vin7.

The function returns an integer value from 0 to 255, since the CMC-S3 also has 8-bit analogue to digital converter which is faster as the 12-bit. The number obtained is linearly dependent to the input voltage. The range of channels Vin0 to Vin5 is from 0 to 5V, and for Vin6 and Vin7 it is from -9.8V and +9.8V.

Example code:

```
nX1 := LibePL.AnIn8bitI (Vin0);  
nX2 := LibePL.AnIn8bitI (Vin6);
```

Say that input voltage at Vin0 would be 1,520 V, the nX1 ideal value would be 78 ($=1,520V * 255/5V$). At -3.50 V at Vin6, the nX2 would be 82 ($=(-3.50V+9.8V)*255/19,6V$).

function AnIn8bit (Vin:TVin) : real;

The function works with the CMC-S3 system.

Vin is the input channel marked with Vin0 to Vin7.

The function returns a real number for the input voltage and is based on faster 8-bit AD converter. The range of channels Vin0 to Vin5 is from 0 to 5V, the channels Vin6 and Vin7 the range is -9.8 V to +9.8 V.

Example code:

```
dbX1 := LibePL.AnIn8bit (Vin0);  
dbX2 := LibePL.AnIn8bit (Vin6);
```

dbX1 is a real number representing the voltage at Vin0, while dbX2 at Vin6.

procedure Wait (dbWait:double);

Function is not directly related to the IO system. It is used to achieve wait operation with the resolution between 8 to 16 milliseconds (ms).

snTime is real number in seconds.

The function returns actual time of waiting (what may not be of and interest at all).

Example code:

```
LibePL.Wait (2.5);
```

Waits for 2.5 seconds.

While the wait operation, the other events are still registered. Like a mouse click on some other button for example.

function GetCPUSpeed : Double;

The function is a part of a dynamic link library (dll) named TimeHiRes.dll, which was developed within the Delphi development environment (author S. Kocijancic). Its call is required before the rest of the high resolution time related functions, namely Waitms, ResetTime and GetTime. The GetCPUSpeed procedure is evoked within the InitIOePL, so it does not need to be explicitly called if InitIOePL was called while starting the application.

The function returns the frequency of the PC's processor (CPU) clock. It is a real number denoting the frequency in MHz (megahertz).

Example code:

```
Label1.Caption := FloatToStrF(LibePL.GetCPUSpeed, ffFixed, 6, 2) + ' MHz';
```

writes the clock frequency in MHz.

Procedure Waitms (dbWaitms:double);

The procedure is a part of TimeHiRes.dll.

dbWaitms is waiting time in mms (milliseconds).

Example code:

```
For nI := 1 To 10 do begin
  LibePL.BitOut (DigC, Do6, BitOn);
  LibePL.Waitms (100);
  LibePL.BitOut (DigC, Do6, BitOff);
  LibePL.Waitms (50)
end;
```

The following is repeated ten times: bit Do6 is set to level 1 for 100 ms and to level 0 for 50 ms.

A time resolution of the procedure is few microseconds (μ s). During the Waitms procedure, nothing else can be registered by the PC, even the mouse cursor would "freeze". It is recommended to use the procedure for no longer than 100 ms.

Procedure ResetTime;

The procedure is a part of TimeHiRes.dll. and is used in a combination with the ReadTime function. The procedure resets the internal clock to 0.

Example code:

```
LibePL.ResetTime;
```

function ReadTime:Double;

The procedure is a part of TimeHiRes.dll and is used in a combination with the ResetTime procedure. The function returns time in seconds (a real number) since the last call of the ResetTime procedure.

Example code:

```
LibePL.ResetTime;
For nI := 1 To 1000 do dbX := LibePL.AnIn(0);
Label1.Caption := FloatToStrF(LibePL.ReadTime, ffFixed, 6, 6) + ' s';
Writes time in seconds needed to sample 1000 times from the analogue input.
```

Starting a new project

On the PC, Borland Delphi and BDLibePL need to be installed first.

Before starting a new project, one needs to know, what is the path to the file BDLibePL.pas. The file is required for programming IO system. One possibility is to copy the file to a particular folder where all projects will use the same file. The second choice is to copy the BDLibePL.pas file in the new folder created specially for the new project. In the following example, the second choice is assumed. Say that a new folder named IOPrvi is created on drive C:. Copy BDLibePL.pas to it.

Run Borland Delphi (BD).

Run BD. Save the new project File -> Save All, browse for the IOPrvi folder (there is BDLibePL.pas) and rename the unit – say to IOPrviUnit1.pas. Then BD expects to give a name to the project.– say it is IOPrvi.dpr (do not change the default extensions).

The task is to include the library into our project. Open menu under Project and chose option Add to project and browse for BDLibePL.pas. After including the libraries, pres F12 (or in menu View chose Toggle Form/Unit). In the line Uses at the top of the code add BDLibePL. Example line is like this:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, BDLibePL;
```

After doing this, press F12.

By a double click anywhere on the form we come back to the code editor. Between

```
procedure TForm1.FormCreate(Sender: TObject);
begin

end;
```

```
write LibePL.InitIOePL;
```

```
so:
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  LibePL.InitIOePL;
end;
```

Place then two buttons to Form1. Double click to Button1 and insert

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  LibePL.DigOut(DigC, 15);
end;
```

Return back to the Form1 (F12), double click to Button2 and insert a bit different line

```
LibePL.DigOut (DigC, 15*16);.
```

Compile (Run -> Run ali F9) and test. After clicking to the first button, the first four LEDs are on, after clicking to the second buttons should be on.

Save the changes to the project (Save All) in close BD.