Vogelaar Electronics

Dorpsstraat 90
3751 ES Bunschoten Netherlands
Telefoon +31 (0)33 2980727
Fax +31 (0)847 115096
E-mail info@vogelaar-electronics.com

Use_08201.doc
16-10-2005

Using the

DelphiStamp VE08201

and evaluation board VE09201

Bunschoten, Netherlands
16 – October - 2005

Contents:

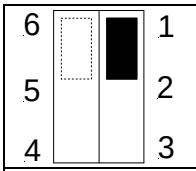
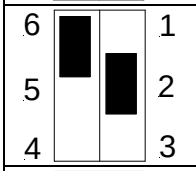
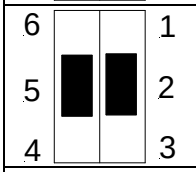
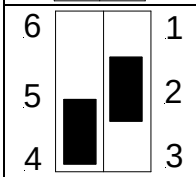
1. Introduction

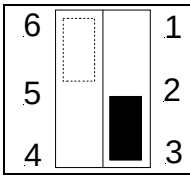
The DelphiStamp is a miniature plug-in controller sizing 52 x 20 x 20 mm. Its firmware can be written in Pascal using the Delphi IDE including features like auto-completion etc. Delphi is also used for testing, simulating and debugging. As the resulting code is compiled by the included PasAvr cross compiler, fast and compact code is generated for the Atmel ATMega128 RISC processor running at 14.7 MHz.

For program storage the DelphiStamp contains 128 kBytes of Flash memory. This type of memory can be reprogrammed up to 1000 times. A protected section of 8 kBytes is set aside to contain the BIOS. The BIOS contains four sections i.e.:

1. **Loader.** Firmware / applications for the DelphiStamp are written and compiled on an external PC. The generated object code is uploaded into the DelphiStamp using a RS232 connection. The Loader program resident in the BIOS, reads this data and writes it into flash memory. When all code is loaded execution can be started.
2. **M485 Server.** Connecting to a running M485 server RAM, Flash, EeProm and I/O memory can be monitored in real time. Also facilities are available to modify the memory contents. Communication with the server is through the RS232 port using the M485 protocol. The DelphiStamp kit contains a client program for this server named Mon485. This is an invaluable tool in debugging a DelphiStamp program.
3. **Interface Server.** A running interface server provides control of standard hardware by reading and writing into RAM variables. Now the standard hardware can be controlled by using the Mon485 client program of from within a Delphi program by using the M485.dll as supplied with the DelphiStamp kit. This technique enables remote control of the DelphiStamp.
4. **Drivers.** For study purposes the BIOS contains drivers for standard hardware. This allows a quick start in writing software for the DelphiStamp. The drivers are available through a jump table made accessible in Delphi through the supplied template unit UDrivers.pas.

Controlling the BIOS.
On the DelphiStamp two three position dip-switches (jumpers) are available. These switches select the functioning of the BIOS.

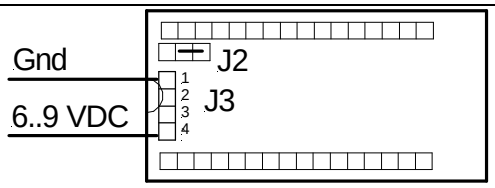
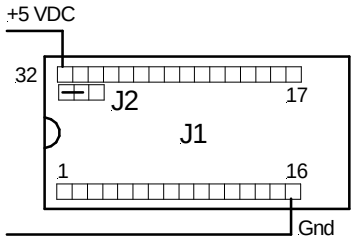
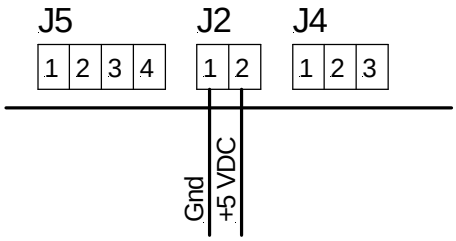
	Right switch: 1-2 Left switch: don't care The DelphiStamp is in Reset
	Right switch: 2 middle Left switch: 5 – 6 The M485 server is running
	Right switch: 2 middle Left switch: 5 middle Both, M485 Server and Interface Server are running
	Right switch: 2 midden Left switch: 4 - 5 A test program for standard hardware is executed. Both, M485 Server and Interface Server are running

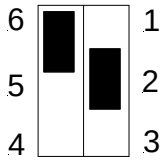
	Right switch: 2 – 3 Left switch: don't care (can be used in a user application) Executing a user application

2. Power supply

Three configurations are available to energize the DelphiStamp.

WARNING: Incorrect connections or voltage supply levels can degrade or even destroy the DelphiStamp.

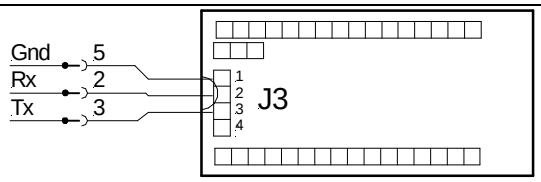
	**A. DelphiStamp communication header J3.** This connection is protected against polarity cross-over errors and tolerate a voltage level between 6 and 9 Volt DC. An on-board low drop regulator provides for 5 VDC for the DelphiStamp only (ther is no capacity to energize external components). On header J2 the connection 1-2 should be made.
	**B. DelphiStamp main header J1.** In this configuration the DelphiStamp is energized from the circuit it is plugged in. The voltage between pin 16 and 32 shoul be 5 VDC +/- 5%. On header J2 the connection 2-3 should be made.
	**C. Evaluation board screw terminal J2.** When using evaluation board VE09201 a screw terminal is available to energize the Evaluation board and the DelphiStamp with a voltage of +5 VDC +/- 5%. On header J2 the connection 2-3 should be made.



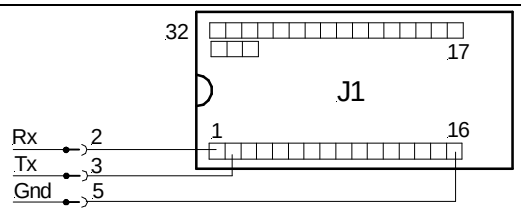
Setting the BIOS switches in the M485 server position before power-on the DelphiStamp can be tested on proper functioning. Yellow LED L1 will flash in a 0.5 Hertz rithm i.e. 1 second on, then 1 second off.

3. RS232 connection

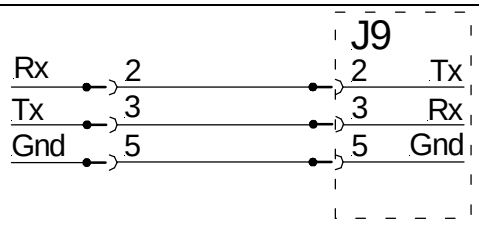
A RS232 connection is required between the DelphiStamp and a remote PC to be able to upload and program firmware and for implementing a remote control connection. Three configurations are available to connect a Com port with the DelphiStamp.



A. Communication header J3.
In stand-alone or plugged-in configurations header J3 provides a RS232 connection. A cable wired as shown in the diagram on the left connects the DelphiStamp with the PC,

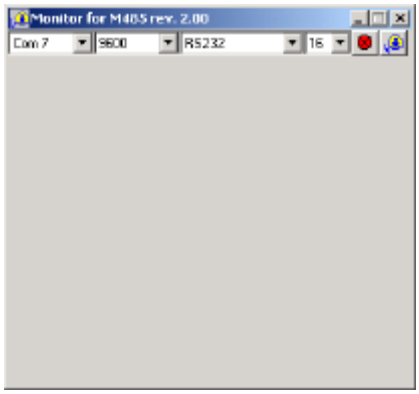


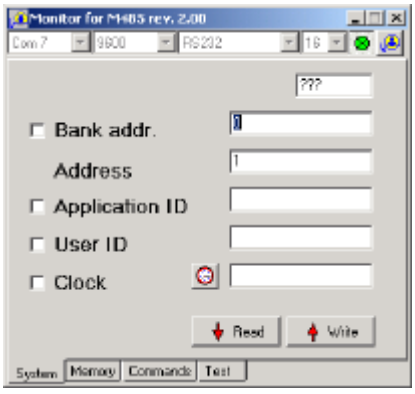
B. Main header J1.
The connections of J3 are duplicated on J1 to inter connect with an application connector / header.

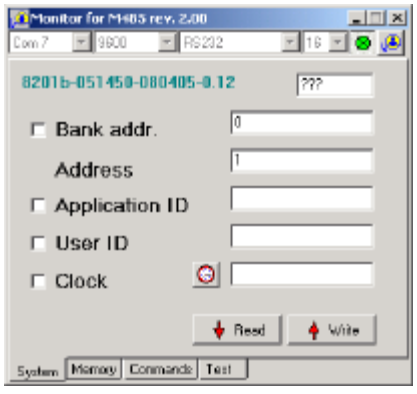


C. Evaluation board connector J9.
The VE09201 evaluation board is fitted with a standard 9 pin female connector. A standard RS232 cable is used for a connection with a PC.

Proper functioning is tested with the M485 client program Mon485.exe. Reset the DelphiStamp and enable the M485 server by using the on-board switches.







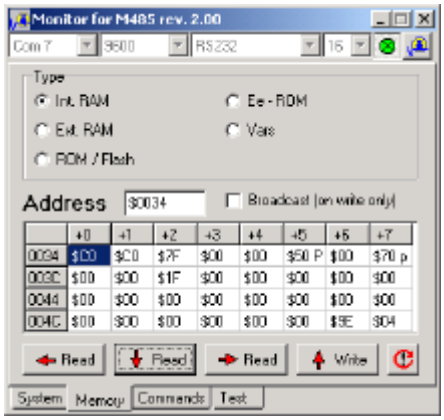
The M485 server in the DelphiStamp communicates using the M485 protocol. This protocol allows for up to 65000 devices. Each device is identified by an address and bank number, both ranging from 0 .. 255. Each device will respond to bank = 0 and address = 1.

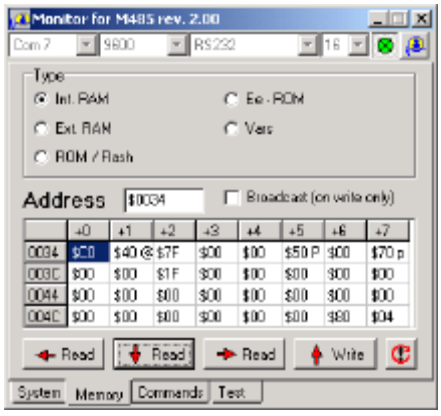
1. Start program Mon485.exe on the PC and select the used Com port. (ex. Com 1), select the used baud rate of 9600 baud, communicate through RS232 and specify a block size of 16 Byte.
2. Create a connection by pressing the button with the red lamp. If succeeded the lamp turns green. Specify Bank address 0 en Address 1 as communication address.
3. Press the button Read to read the serial number of the DelphiStamp. It is visualized as bbbbb-ssssss-ddmmyy-r.rr where bbbbb is the board number, ssssss the serialnumber, ddmmyy the date of manufacturing and r.rr the firmware revision number.

4. Monitoring the DelphiStamp

A running M485 server allows for monitoring and editing RAM, Flash, EeProm and I/O memory in the DelphiStamp. Be careful not to change the stack memory as it is likely that this will cause a processor crash making a reset necessary. An example of monitoring and editing is given below.
Make a connection as described under (3) RS232 connection and connect to the M485 server using Mon485.

Example 1. Monitoring flashing LED L1.





Led L1 is connected to I/O port C on bit 7.
The data direction register of I/O Port C i.e. DDRC is located on RAM location $0034 ($ = Hexa-Decimal) and the data output register PortC is located on RAM location $0035.

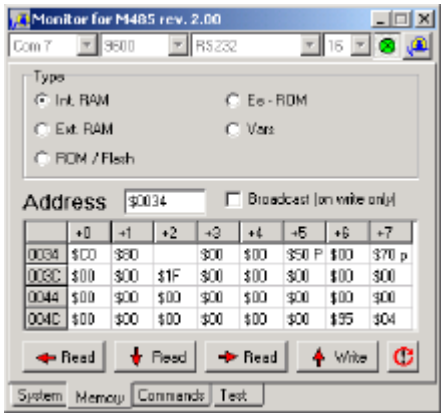
Open sheet Memory in monitor program Mon485 and specify memory Type: Internal RAM. Specify address to read on $0034.

By repeatedly pressing the Read (Down) button the contents of addresss $0034 and $0035 can be observed. $0034 always reads $C0 and $0035 reads alternating $C0 and $40.

Register DDRC reads $C0 = %11000000 ($ is the prefix for Hexa decimal and % for binary numbers), the most left bit, representing DDRC.7 always reads High, meaning this I/O wire is programmed as output.

Register PortC reads alternating $C0 = %11000000 and $40 = %01000000, the most left bit, representing PortC.7 alternates between High and Low. As the LED is connected between +5Volt and ground a low output sets 5 Volt across the LED and makes it emitting light.

Example 2. Setting Led L2 on and off.



Led L2 is connected to port C on bit 6.

As Data Direction Register DDRC reads $C0 = %11000000 both D7 and D6 are high and set as output.

Led L2 is on when PortC.6 is low and Led L2 is off when PortC.6 is High.

Setting PortC is done by writing to internal RAM address $0035. Writing is executed by clicking the write button from the lowest address until an empty field is found. Leave $0034 on $C0 to keep PortC.7 and PortC.6 as output. Write $00 to address $0035 to switch Led L2 on or $40 to switch it off.

5. Connecting standard hardware

The BIOS of the DelphiStamp contains drivers and a test program for standard hardware.

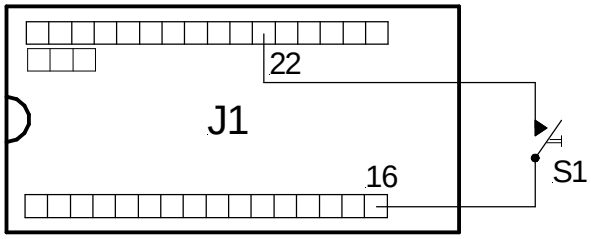
Standard hardware present on the DelphiStamp:

- ?? Led L1 and Led L2
- ?? Switches / Jumpers
- ?? RS485 port

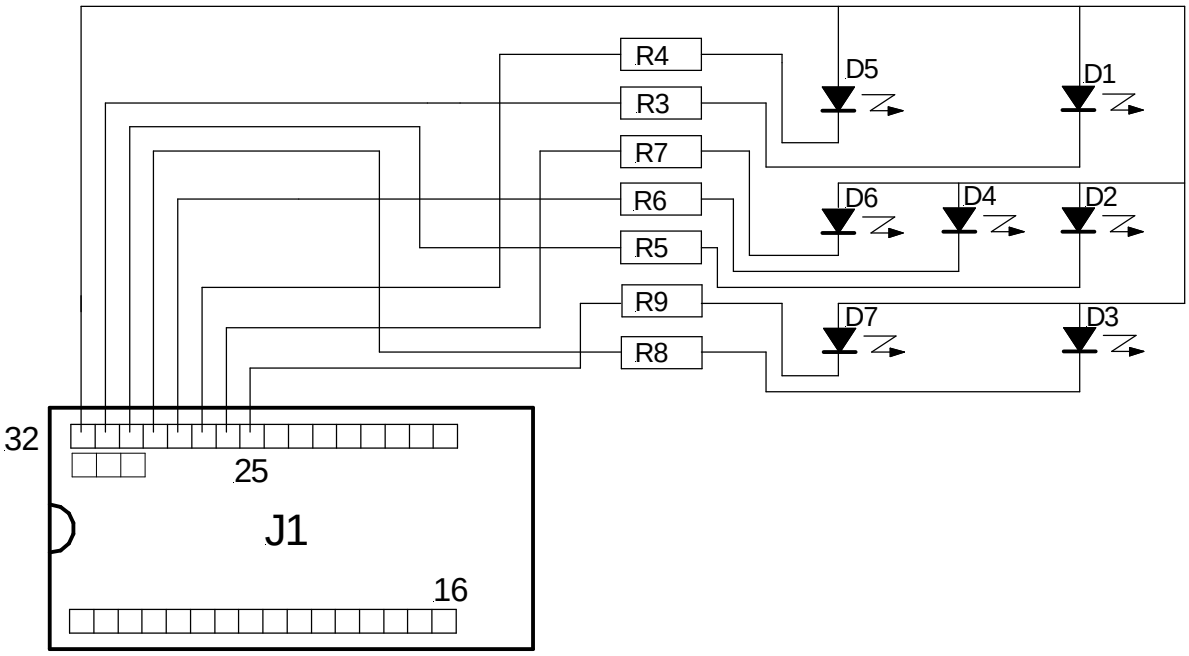
External hardware, as used on the VE09201 evaluation board:

- ?? Push button
- ?? 7 Led's configured as a dice
- ?? Potentiometer and auxiliary analogue input
- ?? LCD with 16 characters
- ?? Stepper motor with 400 half steps per revolution equals 0.7 degrees

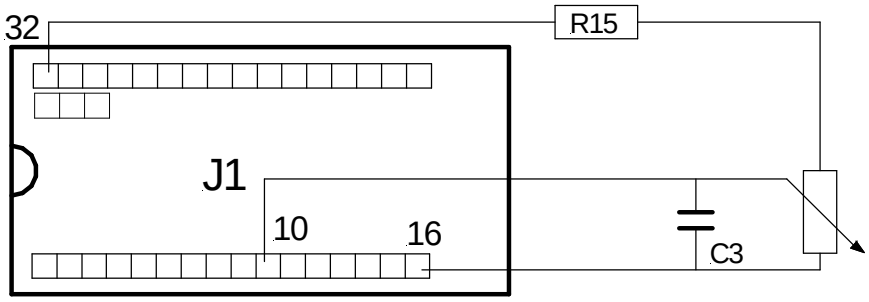
Push button. A normally open push button is connected between PD0 = pin 22 and Gnd = pin 16. The AVR controller provides for a pull-up resistor. This makes PD0 normally high and low when pressed.



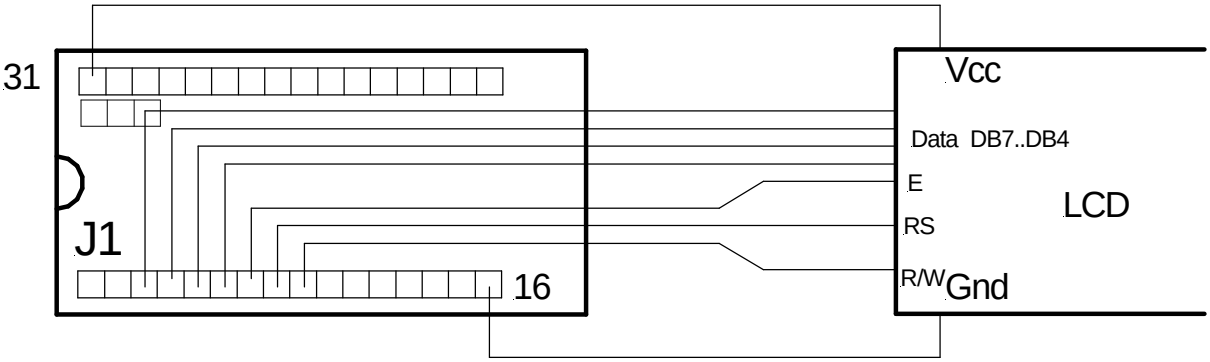
Led's as dice. The dice is made from 7 Led's connected to PB0 till PB6. All anodes are connected to +5 Volt; the cathodes are connected through a current limiting resistor to the controller. As the controller can sink higher currents than it can source this configuration gives minimum heat dissipation in the controller. However the data is inverse, a low for Led on and a High for Led off.



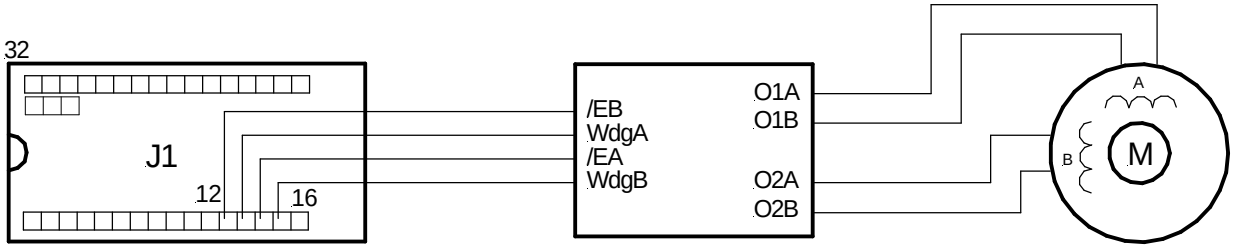
Potentiometer. As the analogue input range of the DelphiStamp is rated as 0 .. 2.56 Volt R15 is dimensioned to result in 0 .. 2.56 Volt over the 300 degrees rotating angle of the potentiometer. The dynamic range is 10 bits i.e. 0 .. 300 degrees is translated into a number of 0 .. 1023. Capacitor C3 is added in parallel of the input to suppress possible EMC interference.



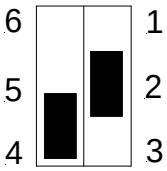
Liquid Crystal Display. The DelphiStamp is able to control LCD's up to 4 lines with 80 characters per line. The resident driver is for a one line by 16 characters model. To preserve I/O lines it is controlled in the 4 bit data bus mode requiring a total of 7 data wires. A connected potentiometer provides for a bias voltage to set for optimal contrast.



Stepper motor. To control a stepper motor four wires and a suitable amplifier are required. The VE09201 evaluation board is equipped with a Unifilar type of motor with a step size of 1.8 degrees. To let the motor run in half step mode is is necessary to be able to switch te motor current on and off and to determine the direction of the current. The /EA and /EB wires are current enable inputs for motor windings A and B. The logic level on the WdgA and WdgB inputs determines the direction of the current through the A and B windings, provided they are enabled. A driver IC is used as energy amplifier.



6. Testing standard hardware



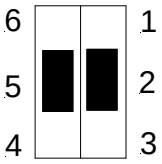
The BIOS contains a test program for standard hardware as described above. This program is started by setting the on-board jumpers in the position as shown on the left and a power-on reset (switch the DelphiStamp off, then on).

The test program includes:

Push button	Led L2 on the DelphiStamp visualizes pressing (led on) and releasing (led off) the push button.
Dice	The Dice counts every second, 1, 2 .. 6, off, 1, 2 etc.
Potentiometer	Turning the potentiometer between 0 and 300 degrees provides 0 .. 2.56 Volt. After digitalization by the DelphiStamp the numeric result is shown on the LCD.
LCD	The LCD shows two numbers. On the left the up-time in seconds and on the right the potentiometer value.
Stepper motor	The stepper motor rotates in half step mode alternately one revolution CW and one revolution CCW.

7. The Interface Server

The easiest way to write your first program for the DelphiStamp is using the Interface Server for controlling standard hardware. A controller program written in Delphi and running on a PC communicates through RS232 and M485.dll with the in RAM located InfDta data record of the DelphiStamp. The Interface Server also communicates with this record and keeps the standard hardware in synchrony.



By setting both switches on the DelphiStamp in the middle position and performing a power-on reset both the M485 Server and the Interface Server become active.

The InfDta record is located in internal RAM from address $10C and is defined as follows:

```
Var     InfData     : Packed Record
                          Leds          : Byte;
                          LcdStr1       : String [8];
                          LcdStr2       : String [8];
                          MotorSts      : Byte;
                          MotorSol      : Word;
                          MotorIst      : Word;
                          Potm          : Word;
                          Analog2       : Word;
                          Dice          : Byte;
                          Dta485        : Byte;
                          Tx485Req      : ByteBool;
                          Button        : ByteBool;
                          Jumpers       : Byte;
                      End;
```

The active M485 Server communicates through RS232 using the M485 protocol at 9600 Baud, 1 stop bit and no parity. The M485 protocol is supported by both the monitor client program Mon485, for manual interactions, and the library M485.dll to be used in programs as written in Delphi. See document "DelphiStamp as interface" for an example in using M485.dll. The interface server synchronizes this record with standard hardware at a rate of 12 Hz.

Field name	Type	W/R	Addr	Description
Leds	Boolean	W/R	$10C	Status of two on-board Led's. Bit 0 : status of Led1 1=On Bit 1 : status of Led2 1=On Bit 7 : Write request.
LcdStr1	String [8]	W/R	$10D	Caption of LCD display. The used LCD contains one line of 16 characters. The internal hardware is built as two attached 8 character strings. LcdStr1 is the left half of the caption. LcdStr1 [0] contains the length of the string and LcdStr1 [1..8] contains the text to display. Strings shorter then 8 characters are right padded with spaces. A write command overwrites the present shown text. LcdStr [0] Bit 7 : Write request.
LcdStr2	String [8]	W/R	$116	Same as LcdStr1 for the right half of the caption.
MotorSts	Byte	W/R	$11F	Status of stepper motor. Bit 0 : Enable power. Bit 1 : CW motor turns clock wise Bit 2 : CCW motor turns counter clock wise Bit 7 : Write request
MotorSol	Word	W/R	$120	Motor setpoint in half steps. Range : -$8000..$7FFF
MotorIst	Word	R	$122	Actual motor position in half steps.
Potm	Word	R	$124	Digitalized value of potentiometer position. Range : 0..1023
Analog2	Word	R	$126	Digitalized value of analogue input #2. Range : 0 .. 2.56 Volts corresponds with 0..1023
Dice	Byte	W/R	$128	Setting of Dice leds. Bit 210 : $0 : Off, $1..$6 show 1 .. 6 in dice format Bit 7 : Write request
Dta485	Byte	W/R	$129	Data byte to transmit through the RS485 port at 9600 baud, no parity and 1 stop bit. When transmission complete this byte is reset to $00. Remark : it is not possible to send $00.
Button	ByteBool	R	$12A	Status of push button. Value is $FF when pressed and $00 when released.
Jumpers	Byte	R	$12B	Position of on-board jumper. Bit0=S1 Bit1=S3 Bit2=S4 Bit3=S6

For many writable locations Bit 7 is used as write command.

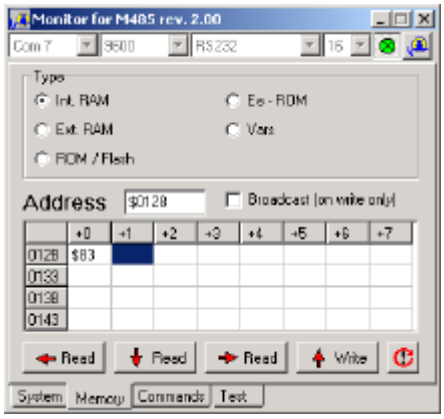
Example : switch on-board Led1 on and Led2 off

Write to record field Leds on address $10C the value $81 = %10000001

Reading this field back after processing by the interface server : $01 = $00000001

A few examples in communicating with the interface server by using the M485 client Mon485.

1. Setting the Dice.

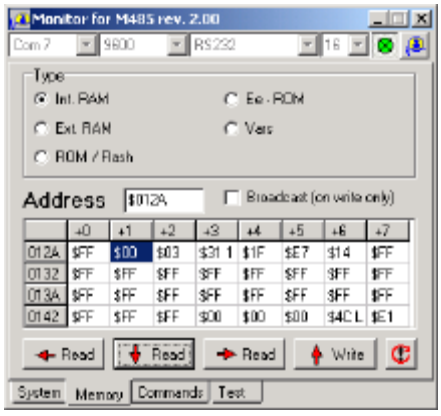


Write $83 to field Dice on address $83 to request the value 3 including bit 7 for the write request.

Mon485 actions:

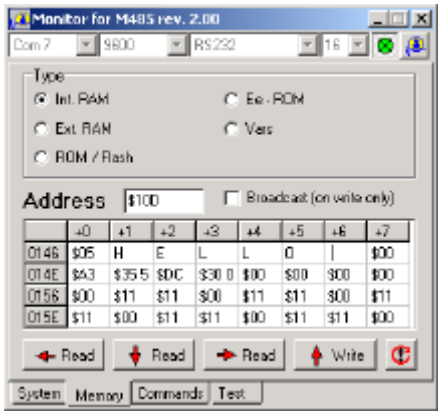
1. Select Memory sheet
2. Select memory type Int. RAM
3. Specify address $0128
4. Provide $83 as byte to write followed by an empty field
5. Press the write button

2. Reading the Push button.



Read field Button on internal RAM address $012A.
Press repeatedly the Read – down button.
See $FF while the button is pressed and $00 while released.

3. Writing the LCD.



Write in field LcdStr1 of internal RAM at address $10D the string 'HELLO' starting with the value $85 on address $10D. This byte includes a write request on bit 7 and a string length of 5 characters. Press the write button to write it into the DelphiStamp.

Reading from internal RAM address $10D shows the present displayed string.

8. Resident drivers

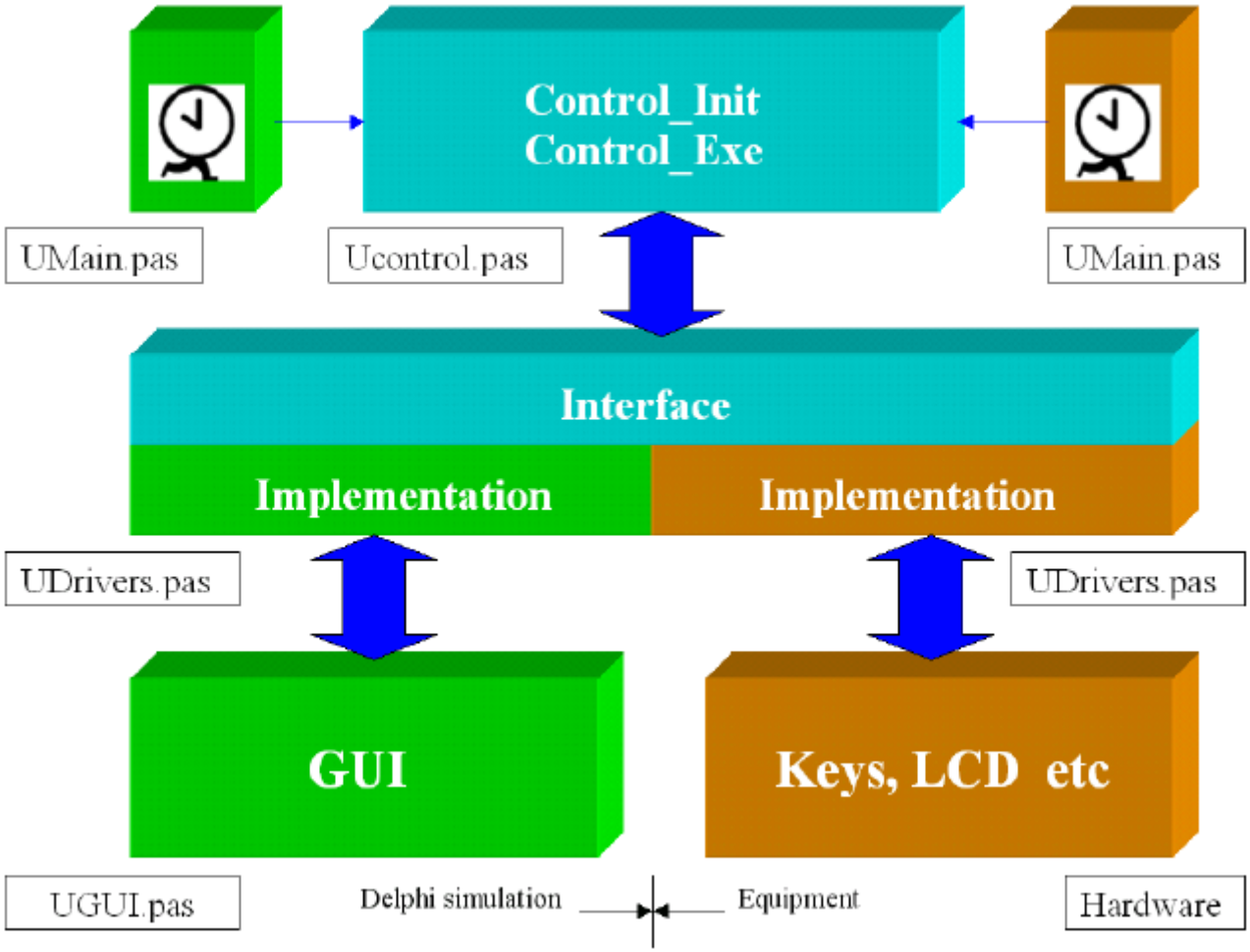
To assist in writing your first DelphiStamp application the BIOS contains resident drivers to control standard hardware. The driver routines are accessible through a jump table. This table is declared in the template unit UDrivers.pas. Available are:

Syntax	Address	Description
`Procedure ServerInit (UBBR : Word)`	$1E08C	Initialisation of the M485 Server. UBBR = (921600 / Baud rate) – 1
`Procedure ServerTest (UBBR : Word)`	$1E090	Restart the M485 server after a comm. error
`Procedure ServerExe`	$1E094	M485 Server execute procedure. Call this procedure repeatedly for example in ControlIdle
`Procedure InitVE09201` `Procedure IoInit (Alias)`	$1E0CC	Initialize hardware on DelphiStamp including Evaluation board VE09201.
`Procedure Led1On`	$1E0D0	DelphiStamp Led L1 on
`Procedure Led1Off`	$1E0D4	DelphiStamp Led L1 off
`Procedure Led1Toggle`	$1E0D8	DelphiStamp Led L1 toggle
`Procedure Led2On`	$1E0DC	DelphiStamp Led L2 on
`Procedure Led2Off`	$1E0E0	DelphiStamp Led L2 off
`Procedure Led2Toggle`	$1E0E4	DelphiStamp Led L2 toggle
`Function GetJumper : Byte`	$1E0E8	Read position of DelphiStamp jumpers $yx x : position 123 y : position 456 0=Up 1=Middle 2=Down
`Procedure LcdInit`	$1E0EC	Initialization and clear LCD
`Procedure LcdWrite1 (Var S : TStr20)`	$1E0F0	Write S to line 1. On a single line LCD this represent the left half of the caption..
`Procedure LcdWrite2 (Var S : TStr20)`	$1E0F4	Write S to line 2. On a single line LCD this represent the right half of the caption..
`Procedure StepperOpen`	$1E0F8	Energize the stepper motor
`Procedure StepperClose`	$1E0FC	De-energize the stepper motor
`Procedure StepperGo (Pos : Integer)`	$1E100	Set setpoint of stepper motor in half steps. Range : -$8000 .. $0000 .. $7FFF
`Function StepperDir : Byte`	$1E104	Read stepper motor dorection $FF : moving CCW $00 : Halt $01 : moving CW
`Procedure StepperExe`	$1E108	Execute stepper motor algorithm. Call continuously @ F > 256 Hz
`Function GetButton : Boolean`	$1E10C	Read push button. $FF = pressed $00 = released
`Function Potmeter : Word`	$1E110	Potentiometer-angle $000 .. $3FF 0 .. 1023
`Function GetAnalog2 : Word`	$1E114	Analogue input #2 $000 .. $3FF 0 .. 1023
`Procedure Tx485 (Ch : Char)`	$1E118	Write CH through the RS485 port at 9600 Baud, no parity and 1 stopbit
`Procedure DoDice (X : Byte)`	$1E11C	Set Dice Led's $00..$06 Dice code. Off, 1 .. 6 $80..$FF binaire code
`Procedure ClkUpd`	$1E120	One second passed test (UpdFlg) and update RTC record
`Procedure ISInit`	$1E124	Initialize Interface Server
`Procedure ISExe`	$1E128	Interface Server execute procedure. Call this procedure repeatedly for example in ControlIdle

For an example in using the resident drivers see document "Stand alone applications"

9. Programming structures

It is well known that in Europe many roads lead to Rome. There are at least that many alternatives in writing programs. Defining a modular structure and providing templates for the units used in this structure dramatically simplifies the design of DelphiStamp applications.



A design starts with a simulation on a PC using Delphi. Delphi provides for a good editor with code completion, keyboard macros, Pascal help files etc. Delphi also has a rich set of VCL (Visual Components Library) components to simulate hardware normally connected to the DelphiStamp.

In this design stage at least for units are required:

UGUI.pas	Unit containing visual components replacing the hardware console of the controller application.
UDrivers.pas	Drivers to control UGUI.pas.
UControl.pas	The actual control algorithm.
UMain.pas	Containing the simulated operating system of the DelphiStamp.

Units required for the generation of DelphiStamp firmware by the PasAvr cross compiler:

Hardware	Console components like : push buttons, switches, Led's, LCD, motors etc.
UDrivers.pas	Drivers to control above mentioned hardware
UControl.pas	The actual control algorithm. A copy of UControl.pas as used in the simulation.
UMain.pas	Operating system for the DelphiStamp

UDrivers.pas
UDrivers.pas for Delphi is a different unit then UDrivers.pas for PasAvr. This because the Delphi unit controls visual components while the PasAvr unit has to handle physical hardware. The Interface section of both units is identical, the implementation differs. When drivers are designed for particular hardware they can be included in a library for re-use in future projects.

UControl.pas
Because the PasAvr Cross-Compiler is compatible with the Delphi compiler (no objects) both units are identical. This property makes it so convenient for testing end debugging AVR firmware under Delphi.

UMain.pas
This unit handles power-on initialization, timers, event handling (IRQ) and background processing. These units are different for both platforms. However, they are supplied as templates and can be used unmodified.

10. Installation of PasAvr

The PasAvr Cross-Compiler as supplied with the DelphiStamp kit compiles code written in Delphi into a ROM hex file. This file contains the binary code required by the AVR processor on the DelphiStamp. It is assumed that the compiler will be installed in the directory C:\PasAvr.

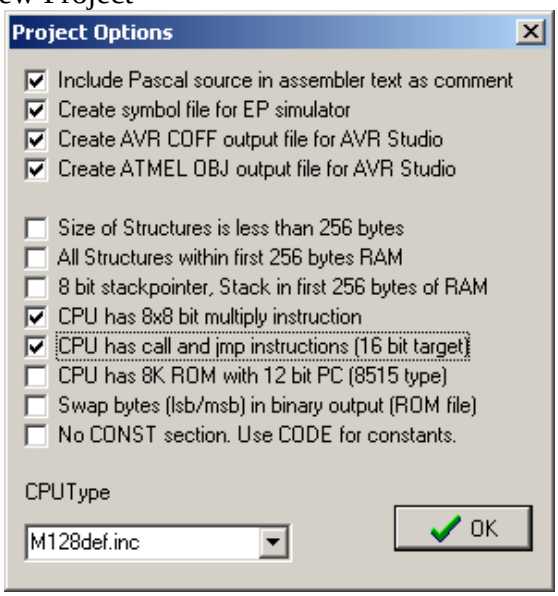
1. Copy the directory CD\PasAvr to C:\
2. Execute the installer C:\PasAvr\AvrPas.exe

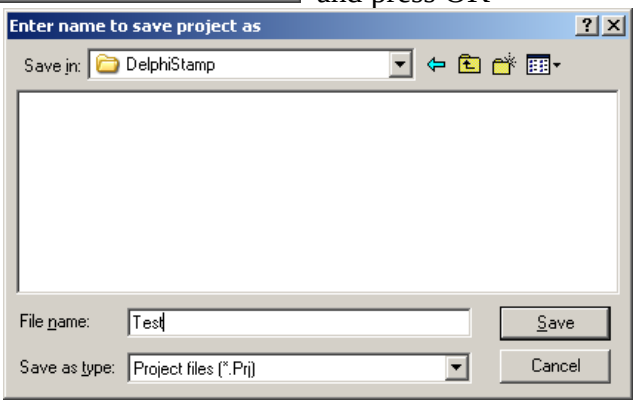
After installation the compiler is in the demo mode. In this mode the maximum size of units is 4000 characters and the resulting binary file is limited to 2000 bytes. The author Rainier Lamers from South Africa is in the process of converting this compiler into a public domain version. Until this is done you have the authors permission to use my personal license to unlock the demo limits.

3. In the menu go to : Help | About
4. Press Ctrl plus KLWQ
5. Username = Anthony J Vogelaar
6. Userkey = 3C2E820BPL7W
7. Press on Register – OK - Close
8. Start C:\PasAvr\AvrSim.exe
9. In the menu go to : Help | About
10. Press Ctrl plus UTRD
11. Username = Anthony J Vogelaar
12. Userkey = 9F083400KLWQ
13. Press on Register – OK - Close

11. Compiling a PasAvr project

A typical DelphiStamp application requires two templates (UMain.pas and UDrivers.pas) and the in Delphi written and debugged file UControl.pas. To compile this project:

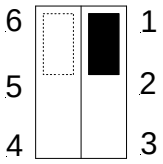
1. Create a new directory ex. C:\DelphiStamp
2. Copy the Delphi file UControl.pas into this directory
3. Copy the template file ~\PasAvr Start\UMain.pas into this directory
4. Copy the template file ~\PasAvr Start\UDrivers.pas into this directory
5. Start AvrPas.exe
6. Menu ✍ Project | New Project
7. Set project options 

and press OK
8. Save this project in the created directory 

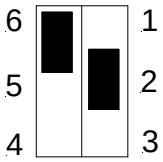
9. Project | Main file | UMain.pas
10. Project | Add to project | UMain.pas
11. File | Open | UDrivers.pas
12. File | Open | UControl.pas
13. Click on the UDrivers.pas tab. Compile | Current editor file
14. Click on the UControl.pas tab. Compile | Current editor file
15. Compile | Main project file
16. The file C:\DelphiStamp\Test.ROM is created ready for uploading into the DelphiStamp.

12. Program upload and execute.

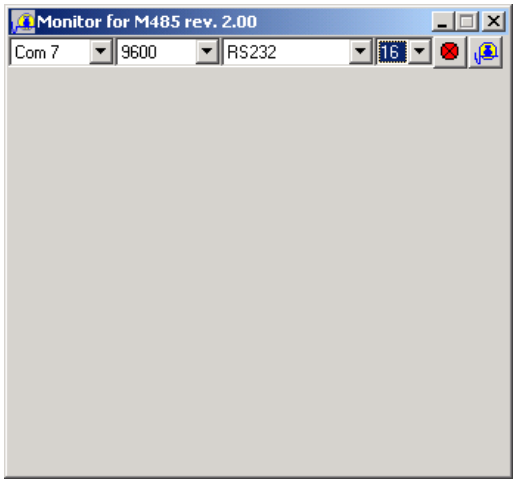
To upload the by the PasAvr cross-compiler generated *.ROM file into the DelphiStamp the monitor program CD:\Mon485\Mon485.exe is used. Remark : copy it to hard disk first.



Reset de DelphiStamp by using the on-board jumpers

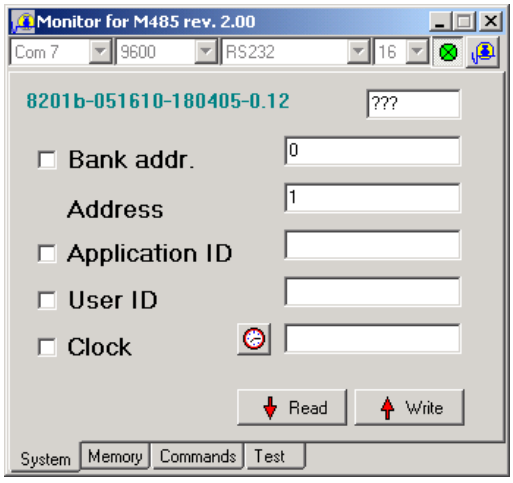


Activate the M485 Server by using the on-board switches. Now yellow Led L1 will flash in a rithm of 0.5 Hz. (1 second on and 1 second off).



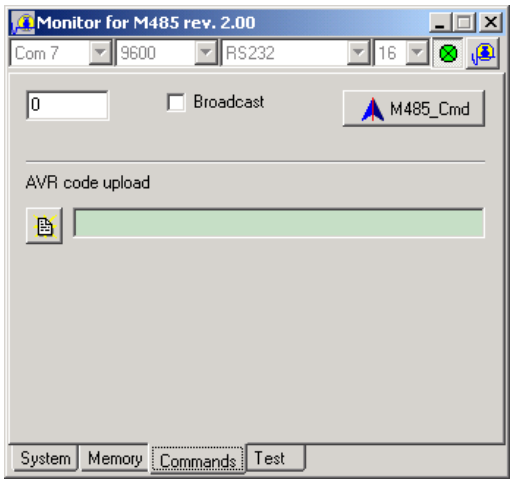
Establish a RS232 connection between the DelphiStamp and the PC.
Start the monitor program Mon485.exe

Pressing the button with the red lamp will open the connection between the DelphiStamp and the PC.



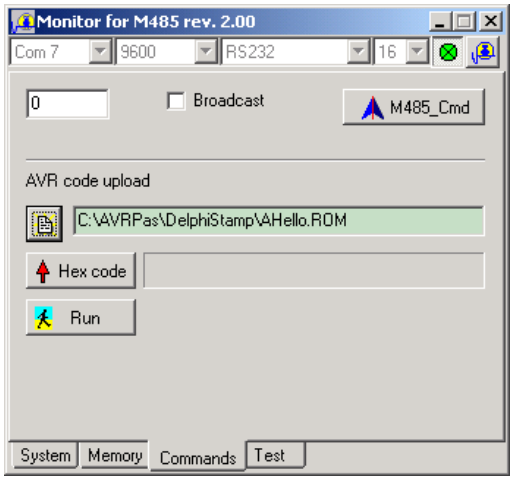
Pressing the Read button will present the DelphiStamp hardware identifier. It contains its serial number, manufacturing date and software revision number.

This procedure is done to check for correct communication between the DelphiStamp and the PC.

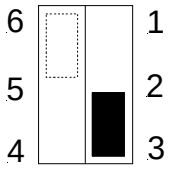


Click on tab Commands to open the upload code dialog.

Press the "AVR code upload button" and specify the *.ROM file to upload.



Click on button "Upload Hex code". This will transfer the binary into the DelphiStamp and store it into Flash memory.
To start execution press the "Run" button.



To start this user program at power-on jumpers 2-3 should be connected on the DelphiStamp. Jumpers 4-5-6 are free to use by the application program.

We wish as much as pleasure in programming and using the DelphiStamp as it was for us in designing it.

Best regards, Anthony J. Vogelaar
Vogelaar Electronics.