

Becker & Hickl GmbH
Nahmitzer Damm 30
12277 Berlin
Germany
Tel.: +49 / 30 / 787 56 32
Fax: +49 / 30 / 787 57 34
Email: support@becker-hickl.de
<http://www.becker-hickl.de>



26.10.99

Arbitrary Pulse Generator / PC Pattern Generator PPG-100

Operating Manual V 1.2



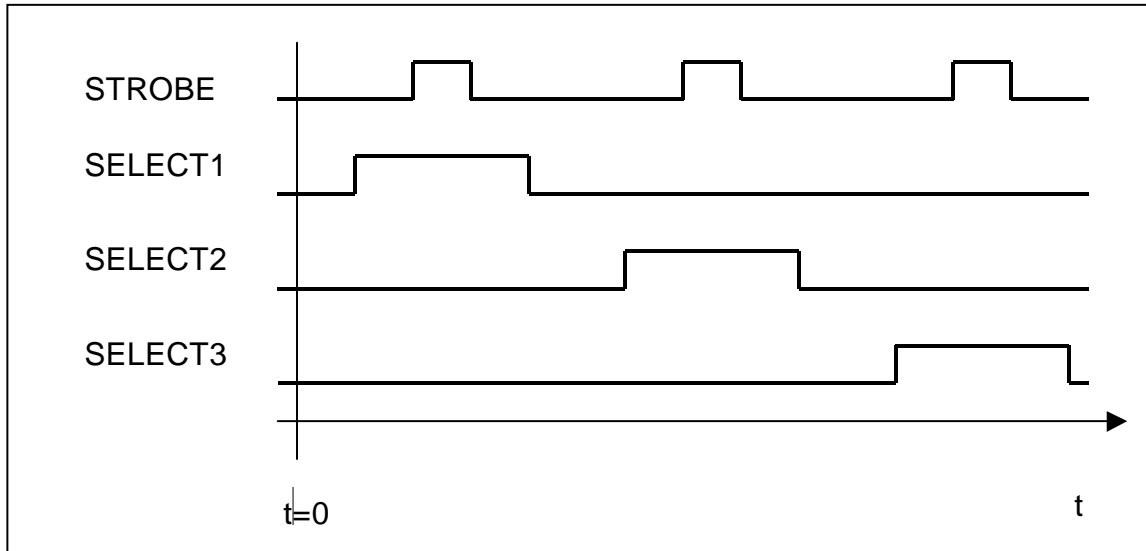
Contents

Introduction.....	1
Technical Data.....	3
Hardware Description	4
Structure of the PPG-100 Module	4
Connector Layout	5
I/O Base Address Setup.....	5
Setup of the Voltage Range of the Analog Output.....	6
Software Support.....	6
Installation	6
Installation Procedure	6
Requirements to the PC	7
Software Installation.....	7
PPG_WIN	7
PPG_DOS and function libraries	8
Hardware Installation	8
Changing the Module Address	8
Program Description PPG_WIN	10
Status	10
Running / Stopped.....	10
Wait for Trigger	10
Status Line.....	10
Next Pattern Number	11
Menu Bar	11
Main	11
Production Data	11
About	11
Exit.....	11
Buttons.....	11
Reset (F6).....	11
Start (F7)	11
Stop (F8)	11
Step (F9).....	12
Load (F10).....	12
Input Fields.....	12
Clock Source	12
Reset Pattern	12
Set Pattern No	12
Program Description PPG_DOS	13
Reset	13
Load	13
Start.....	13
Terminate (Stop).....	13

PPG-100 Libraries.....	14
Installation.....	14
Description of the PPG Library Functions	14
PPG_init	15
PPG_get_parameters	15
PPG_set_clk_source.....	15
PPG_get_clk_source.....	16
PPG_set_res_pattern.....	16
PPG_get_res_pattern	16
PPG_get_status.....	16
PPG_start.....	17
PPG_stop	17
PPG_reset	17
PPG_get_eeprom_data	17
PPG_write_eeprom_data.....	18
PPG_load_pattern.....	18
PPG_write_pattern	18
PPG_read_pattern.....	19
PPG_fill_pattern	19
PPG_set_memptr	20
PPG_get_memptr	20
PPG_get_id.....	20
PPG_test_id.....	20
PPG_get_version	21
PPG_set_mode	21
PPG_get_mode.....	21
PPG_get_error_string.....	21
Appendices	22
Appendix A: Configuration File (PPG100.INI)	22
Appendix B: Pattern File (EXAMPLE.PAT)	23
Pattern Types.....	24
Time.....	24
Trigger Condition	24
Labels	24
Patterns	24
Analog Value.....	25
Appendix C: Contents of the Pattern Memory	26
Trigger Condition.....	27
Technical Support	27

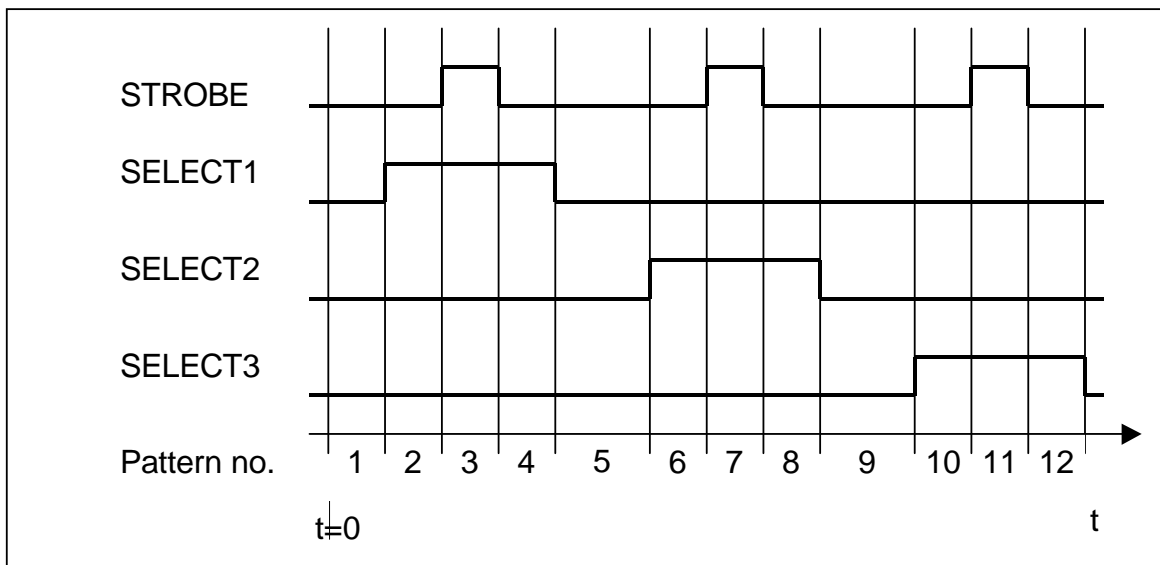
Introduction

Experiment Control frequently requires several logical control signals. All these signals must be switched on and off in a defined sequence. The following diagram shows a simple example of an experiment control with four signals:



The Arbitrary Pulse Generator / Pattern Generator PPG-100 can generate up to 32 logical control signals.

The state of all 32 outputs at a certain time is called "pattern". Further can the time during which the pattern doesn't change (the "pattern time") be treated as a characteristic of the pattern. Our example divided into patterns looks like this:



The PPG-100 can generate a sequence of up to 64 k patterns. The pattern time is programmable for each individual pattern in the range of 100 ns to > 400 s with a resolution of 100 ns.

Pattern sequences are defined by the user in a text file (ASCII) with a special format (see *Appendix B: Pattern File*). The PPG Operating Software can load such a pattern file, interpret the data, store the information into the pattern memory on the PPG-100 module and starts the execution on user command.

There is a set of commands available, which enables extended features like waiting for a trigger condition or programming of loops.

The PPG-100 has four trigger inputs. The execution of the pattern sequence can be suspended at any pattern until a specified pattern occurs at the trigger inputs. Unused trigger inputs can be specified as "don't care".

Following software is delivered with the PPG-100:

- PPG_WIN Control Program for PPG-100; runs under Windows 3.1 or higher.
This program allows loading of pattern files, starting, stopping and single stepping of the pattern sequence and displays the actual status.
- PPG_DOS Simple MS-DOS program for PPG-100; runs under MS-DOS 5.0 or higher.
This program allows loading of pattern files, starting and stopping of pattern sequences.
- PPG_LIB Function Libraries and Dynamic Link Libraries for Windows;
contains all necessary functions to control the PPG-100.

Technical Data

Inputs

External Clock 2 Inputs 74HCT compatible

Trigger 4 Inputs 74HCT compatible

Digital Outputs

Quantity 32

Voltage Levels CMOS compatible (74ACT574 through 50 Ω)

Analog Output

Output Impedance 50 Ω

Output Voltage Range DIP Switch selectable:

@ 50 Ω load:	@open output:
0,00 V ... 2,55 V	0,00 V ... +5,10 V
-2,56 V ... +2,55 V	-5,12 V ... +5,10 V
-1,28 V ... +1,27 V	-2,56 V ... +2,55 V

Resolution 8 bit = 256 steps

Connectors

Digital Signals with 16 2 1 2 x 37 pin female Sub-D Connector, utilised

Digital Outputs,
Trigger Inputs
External Clock Input

Analog Output SMA, female, 50 Ω

Timing Control

Clock Sources Internal Oscillator,

2 External Clock Inputs,
software selectable

Time per Pattern 2 • (1 ... 4.227.858.432) Clocks
100 ns ... > 420 s @ Internal Oscillator,
programmable individually for each pattern

Pattern Memory

Size 64k Patterns

External Trigger

Method Pattern Generation is suspended until trigger condition is true. A trigger condition may be specified for each pattern.

Trigger Conditions Pattern of the four trigger inputs. Each input can be specified as '0' or '1' or 'X' (= 'don't care').

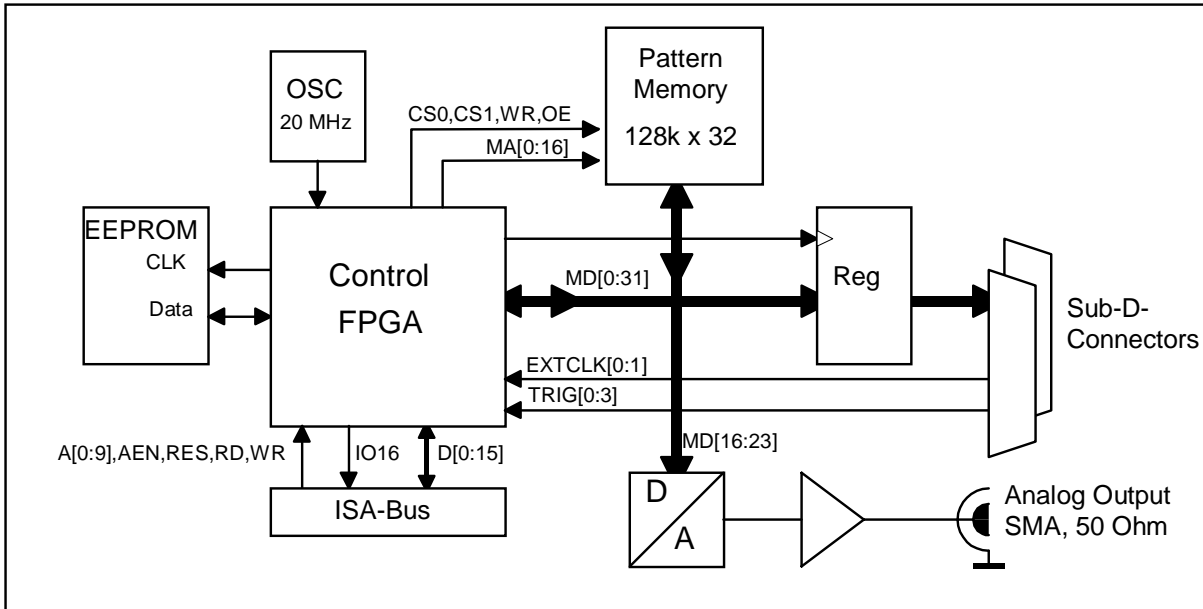
Miscellaneous

Computer Bus ISA 16 bit

Dimensions 160 x 106 mm

Hardware Description

Structure of the PPG-100 Module



Block Diagram of the PPG-100

The **Control FPGA** controls the generation of the pattern sequence and all accesses to the PPG-100 from the ISA bus.

The **Pattern Memory** can keep up to 64 k pattern definitions. There are various types of definitions:

- *Time* – Time for which an output pattern should remain stable + Output Pattern,
- *Wait* – Trigger Condition + Output Pattern,
- *Jump* – Target Pattern Number + Trigger Condition + Output Pattern

Details see *Appendix B: Pattern File*.

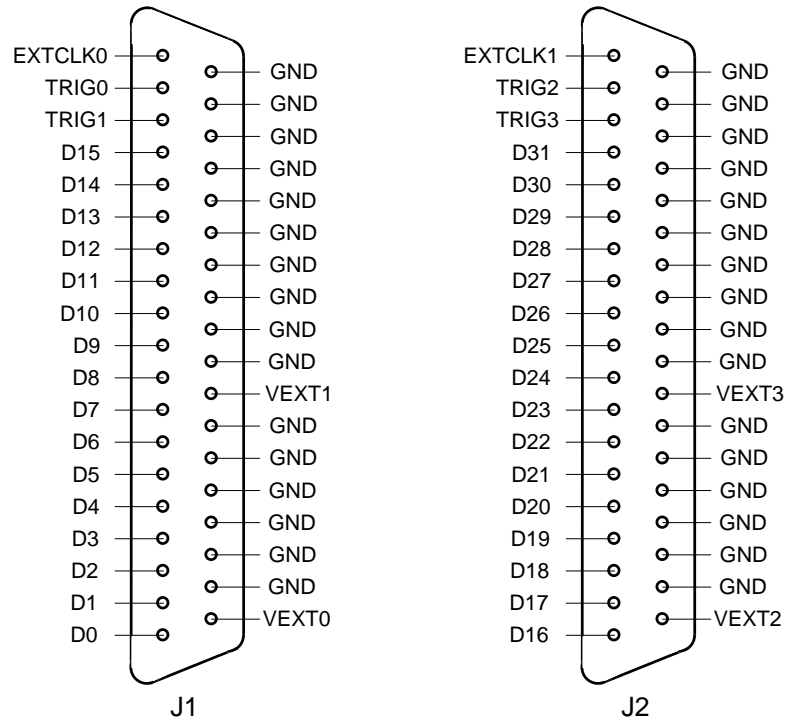
The desired pattern sequence must be defined in a text file, the Pattern File. The format of this file is described in *Appendix B: Pattern File*.

If necessary an external clock can be used. The PPG-100 requires two clock pulses for one time unit. Therefore the input clock frequency must be twice the maximum pattern frequency. The minimum pulse distance must not be less than 50 ns.

The EEPROM (24C04) contains manufacturing data as product name, serial number, date of manufacturing and configuration data like size of the pattern memory.

Connector Layout

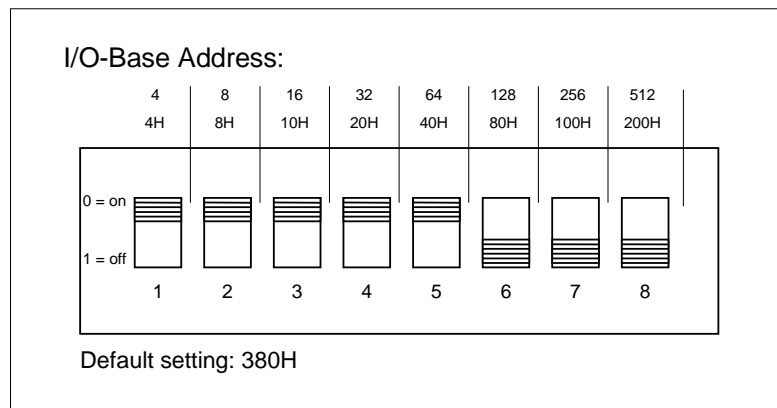
The digital I/O signals can be accessed through the female 37 pin Sub-D connectors J1 and J2. J1 is soldered directly onto the PPG-100 PCB. J2 mounted on a separate bracket and is connected to the PPG-100 via flat cable.



The signals VEXT[0:3] are currently not used. Do not connect!

I/O Base Address Setup

The I/O Base Address is set up with DIP switch SW1:



Setup of the Voltage Range of the Analog Output

The Voltage Range of the Analog Output is selected by DIP switch SW2:

Voltage Range of Analog Output:		switch		Voltage Range	
0 = on	1 = off	1	2	50 Ohm load	open
		OFF	OFF	0 ... +2,55 V	0 ... +5,10 V
		OFF	ON	-1,28 ... +1,27 V	-2,56 ... +2,55 V
		ON	ON	-2,56 ... +2,55 V	-5,12 ... +5,10 V
		ON	OFF	----- invalid -----	

Default: OFF OFF

Software Support

Following Software Packages are delivered with the PPG-100 Module:

- PPG_WIN, a Windows Application for Operation of the PPG-100-Module.
- PPG_LIB, a set of function libraries, which can be linked with user programs developed under Borland C++ (V4.5) and Borland Pascal and a Dynamic Link Library (DLL) for Windows Applications.
- PPG_DOS, a simple DOS Application which controls the PPG-100 using command line parameters.

Installation

Installation Procedure

Zur Installation des PPG-100-Moduls und der zugehörigen Software gehen Sie am besten wie folgt vor:

To install the PPG-100 Module and the software the following procedure is recommended:

- Ensure that your PC complies with the *Requirements to the PC* mentioned below.
- Install PPGWIN or PPGDOS as described under *Installation - Software*. Only one of these programs is required. For Windows Users PPGWIN is recommended.
- Switch off your computer and install the PPG-100-Module as described under *Installation - Hardware*.
- Switch your PC on again and start PPG_WIN or PPG_DOS. If there is no error message the installation was successful and you can start to generate your first patterns. Some simple example pattern files are enclosed.

Requirements to the PC

Your PC must be IBM-AT compatible. The processor must be 80386 or higher. At least one empty 16-bit ISA slot is required. Up to 16 digital Output Signals, the Analog Output, two Trigger Inputs and one External Clock Input can be connected directly to the PPG-100 Module. The second set of these signals (no second Analog Output) can be connected to an extension bracket, which is connected to the PPG-100 through a flat cable. Another free slot of either type is required for installation of the extension bracket.

The Operating System must be either MS-DOS 5.0 or higher, with or without Windows 3.1 or Windows for Workgroups 3.11, or Windows 95 or Windows 98.

A 3,5"-HD Floppy Disk Drive is required for Software Installation.

A Mouse or Trackball is required for Windows Users.

PPG_WIN requires 800 KB hard disk space, PPG_DOS and the function libraries together require 800 KB too.

Software Installation

PPG_WIN

It is assumed that the PC is running and Windows is started.

Put the PPG_WIN disk into the floppy disk drive.

Choose *Execute* from the *File* menu of the program manager and enter `a:\setup.exe` or `b:\setup.exe` depending on the drive letter of your Floppy Disk Drive. All necessary files are then transferred to your hard disk.

PPG_WIN is based on LabWindows/CVI of National Instruments. Therefore it requires the "CVI Runtime Engine". This program is loaded to the memory at the start of PPG_WIN and contains library functions of LabWindows/CVI. During installation of PPG_WIN a separate directory is recommended for the Runtime Engine. In this way other Labwindows/CVI applications can share the same Runtime Engine.

To enable PPG_WIN to find the Runtime Engine after installation of PPG_WIN the following entry has to be added to WIN.INI:

```
[cvirt3]
cvirt3=c:\cvi\cvirt3.exe
```

WIN.INI is located in the Windows directory (normally `c:\windows`). If you changed the recommended path for the Runtime Engine during installation please use your path instead of `c:\cvi`. PPG_WIN checks at its start whether it can access the Runtime Engine. If it can't access the runtime Engine it will give some hints what to do.

If you want to present PPG_WIN in a special program group e.g. "BECKER & HICKL" on the desktop, you can create this group by choosing *File – New* within the program manager. Choose *group* in the next dialog. Next you will be prompted for a *description* of the group and a *group file name*. The description will be used as headline of the group. The file name is not important. Choose any name e.g. "B&H". The group is created then. If you want to put PPG_WIN to an existing group select this group by clicking on it with the left mouse button.

Now choose *File – New – Program* and enter the values of PPG_WIN:

Description = PPG-100

Program Path = C:\PPG_WIN\PPG_WIN.EXE

Directory = C:\PPG_WIN.

Finally you can choose after clicking on *Other Symbol* the PPG Icon which is presented then. After a final *OK* you can start PPG_WIN by double clicking on the icon.

PPG_DOS and function libraries

Copy all files from the PPG_LIB disk to a directory of your choice to your hard disk. If you want to use PPG_DOS from other directories you should add the PPG_DOS path to the path statement in the file C:\AUTOEXEC.BAT. A restart of the PC is required to update the path information.

Hardware Installation

The PPG-100 Module is designed in a way that all communication is done through a set of IO ports. DMA and Interrupts are not used to avoid configuration problems as far as possible.

The PPG-100 Module occupies as default the IO addresses 0x380-0x38F which is always free in PCs which are equipped with standard modules only (graphics card, FD-, HD-controller, serial and parallel Interfaces).

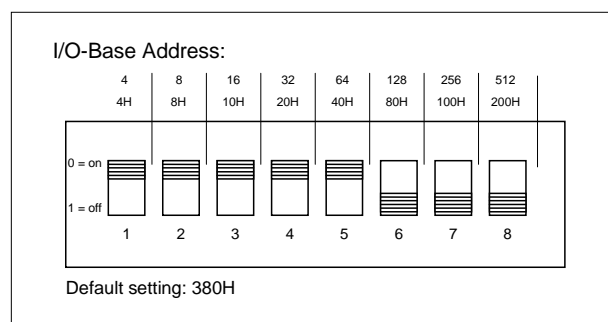
Be sure to switch off the PC before mounting the PPG-100 Module into an unused 16-bit ISA slot. The extension bracket for the second set of 16 Output Signals should be mounted into a slot of any type adjacent to the PPG-100 Module.

When the PPG-100 Module and the extension bracket are mounted switch on the computer start Windows and start PPG_WIN. If no errors are reported everything is okay and you can use the PPG-100.

If errors are reported and/or malfunctions at the PC appear there is probably an address conflict between the PPG-100 Module and some other hardware. In this case the IO base address of the PPG-100 Module must be changed (see *Changing the Module Address*).

Changing the Module Address

The PPG-100 Module is accessed through a set of 4 subsequent IO addresses. The lowest of these addresses which must be dividable through 16 without remainder is the IO base address of the module or the Module Address. The upper 8 bits of this 10 bit address is selectable by DIP switch SW1:



The library software which is used also by PPG_WIN takes the Module Address from a configuration file (default: *PPG100.INI*). Therefore the setting of the DIP switch and the entry in the configuration file must be equal. The configuration file is an ASCII file which can be edited with any ASCII editor as EDIT under MS-DOS or NOTEPAD under Windows. Details see *Appendix A: Configuration File*.

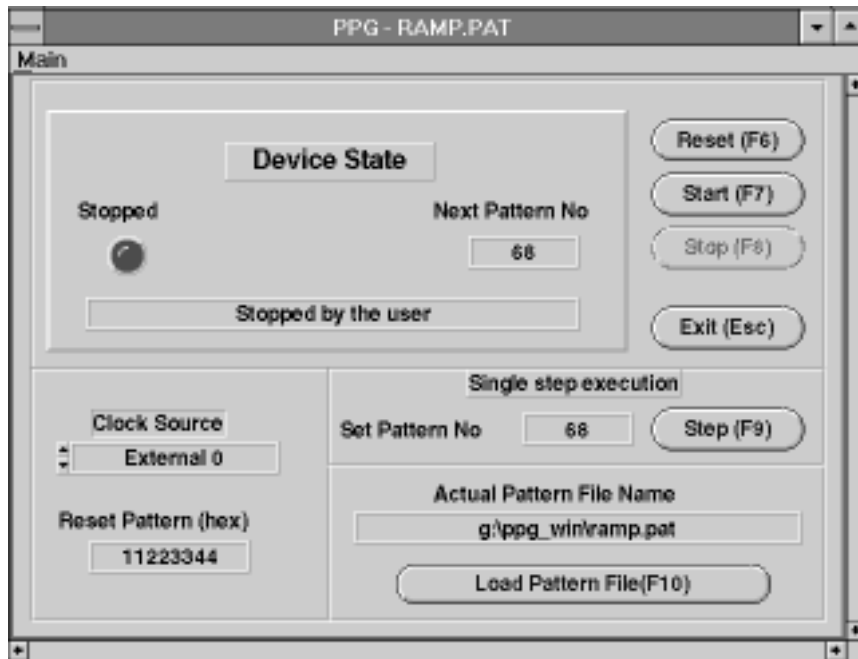
PPG100.INI contains the sections [base] and [device]. The Module Address (IO base address) is located in section [base]:

```
[base]
baseadr = 0x380 ; base IO address (0 ... 0x3F0, increment 0x10)
```

Program Description PPG_WIN

PPG_WIN is used to control the PPG-100 Module and to analyse its status. PPG_WIN is available for Windows 3.1 and for Windows 95/98/NT. It requires a PC with a 486, Pentium or Pentium II processor.

PPG_WIN allows setup of Module Parameters, loading of pattern files, starting, stopping and single stepping of pattern generation.



Status

The status of the PPG-100 Module is displayed with two "lamps" and a status line:

Running / Stopped

If the lamp is green currently a pattern sequence is executed (*Running*). In case of a red lamp no pattern sequence is executed (*Stopped*).

Wait for Trigger

This yellow lamp indicates that pattern sequence execution is suspended until a trigger condition is met.

Status Line

In the status line additional information is displayed e.g. reasons for stopping of execution of pattern sequence.

Next Pattern Number

Here the number of the next pattern to be executed is displayed. This is especially useful to identify the pattern which is to be executed next in case of "Wait for Trigger".

Menu Bar

Nearly all functions are available through buttons. So the menu bar contains only one item: *Main*.

Main

The menu item *Main* contains the sub menus *Production Data*, *About* and *Exit*.

Production Data

Production data are: module type, serial no., production data. These data can be inspected here.

About

The menu item *About* gives information about PPG_WIN as Software Version and creation date.

Exit

One way to leave PPG_WIN. Other ways are the *Exit* button and the ESC key.

Buttons

Reset (F6)

Pushing the *Reset* button (or the F6 key) will reset the PPG-100 hardware. At the outputs will appear the *Reset-Pattern*, the pattern pointer will be set to 0. The contents of the pattern memory are not changed.

Start (F7)

Pushing the button *Start* (or the F7 key) starts the execution of the pattern sequence. Pattern sequence execution starts at the pattern which is displayed at *Next Pattern Number*.

Stop (F8)

The execution of the pattern sequence is stopped by pushing *Stop* (or the F8 key). The actual output pattern is frozen. The sequence can be continued by pushing *Start* (or F7).

Step (F9)

Step (or F9) executes the next pattern: the output pattern is fed to the outputs, times and trigger conditions are ignored. This function is useful for verification of the pattern sequence.

Load (F10)

Load (or F10) opens the load menu. Within this menu a pattern file can be selected and loaded to the pattern memory.

Input Fields

Clock Source

Here the Clock Source can be selected. Possible choices are *Internal Oscillator*, *EXTCLK0* and *EXTCLK1*. The external clocks are fed through the 37 pin connectors to the PPG-100.

The default setting of *Clock Source* is taken from the configuration file (see *Appendix A: Configuration File*). The *Clock Source* setting may be loaded from the pattern files too (see *Appendix B: Pattern File*).

External clocks must deliver two clock pulses for one pattern. The minimum time between two clock cycles is given in *Technical Data*.

Reset Pattern

Here the output pattern can be specified which appears at the outputs when Reset (or F6) is pushed or the PC is reset (by reset button or CTRL-ALT-DEL).

The default setting of *Reset Pattern* is taken from the configuration file (see *Appendix A: Configuration File*). The *Reset Pattern* setting may be loaded from the pattern files too (see *Appendix B: Pattern File*). In this case the Reset function is executed and the pattern will appear at the outputs directly after loading.

Set Pattern No

The number of the next pattern to be executed may be specified here. This is useful for test or if more than one pattern sequences are defined within one pattern file and you want to select one of the sequences which do not start at 0.

Program Description PPG_DOS

PPG_DOS is a simple tool to control the PPG-100 module for those who do not want to use Windows or who cannot use Windows for any reason.

PPG_DOS controls the PPG-100 Module through command line parameters

Pattern files and configuration files for PPG_DOS and PPG_WIN are fully compatible.

PPG_DOS requires (as PPG_WIN) a configuration file. The default file name is PPG100.INI the default path is the current directory. If another configuration file should be used the command line parameter `-I<configuration file>` must be used.

The command `PPGDOS -?` Displays a help text which describes the command syntax and the command line parameters.

Reset

Command: `PPG_DOS [-I<Konfigurationsdatei>] -R`

The PPG-100 hardware will be reset. At the outputs will appear the *Reset-Pattern*, the pattern pointer will be set to 0. The contents of the pattern memory are not changed.

Load

Command: `PPG_DOS [-I<configuration file>] [-S] <pattern file>`

The pattern definitions contained in `<pattern file>` are transferred to the pattern memory, the Reset pattern (if specified in the file) is set, a Reset of the PPG-100 is executed and the pattern sequence execution is started if the `-S` switch is included.

The pattern file may be specified with a full path. If only a file name is specified this file is taken from the current directory.

Start

Command: `PPG_DOS [-I<Konfigurationsdatei>] [-S]`

The execution of the pattern sequence will be started / continued.

Terminate (Stop)

Command: `PPG_DOS [-I<Konfigurationsdatei>] -T`

The execution of the pattern sequence will be stopped. The outputs will remain stable until the next *Reset*, *Load* or *Start* occurs.

PPG-100 Libraries

To support the user to create his own software, libraries for the basic module control functions are available. Different libraries are provided for DOS and WINDOWS applications and for applications under LabWindows CVI.

- ppg_d31.lib - DOS library, created in Borland C++ v.3.1
- ppg_d45.lib - DOS library, created in Borland C++ v.4.5
- ppg_w45.dll - DLL for Windows, created in Borland C++ v.4.5
- ppg_lw45.dll - DLL for LabWindows CVI, created in Borland C++ v.4.5
- ppg_p45p.dll - DLL for protected mode Pascal applications, created in Borland Pascal
- ppg_p45c.dll - DLL for protected mode C applications, created in Borland C++ V.4.5

The libraries have been created under Borland C++ 3.1 and Borland C++ 4.5. Basically the DLLs should work also with other compilers. The compatibility is, however, not tested for these cases.

To facilitate the first steps of the library application we deliver the source code of sample programs with the libraries. The program *USE_PPG.EXE* works under Windows and uses *ppg_w45.dll*. The program *PPG_DOS.EXE* is a DOS application (uses *ppd_d45.lib*) which enables initialization, start, stop, reset of PPG-100 module as well as loading PPG pattern files.

Installation

There is no special installation procedure required. Simply copy the contents of the PPG-LIB disk to a directory of your choice on the hard disk.

Description of the PPG Library Functions

To be able to control more than one PPG-100 module within one PC *PPG_init* returns a *handle* which will be different for each different IO-adress, that is defined in the respective configuration file in use. All other functions select with *handle* on which PPG-100 module the required action should be executed.

In case of errors each function returns an error code and writes an error message to the file *ppg_err.out* in the same directory as the configuration file in use. The file *ppg_err.out* will be created, if it doesn't exist otherwise the error message will be appended at the end of the file.

It is also possible to get error message from inside the application by using library function *PPG_get_error_string*.

PPG_init

short PPG_DECL PPG_init (short far *mod_hndl,char far *config_file);

Input parameters: *mod_hndl pointer to module handle variable
 config_file: pointer to string containing configuration file name
 and path

Return value: 0 - on success,
 <0 - on error ,returns error code

PPG_init initialises the library internal structures using parameters from the configuration file config_file (recommended extension *.ini*), and initialises the internal registers of the module. At the beginning function checks the state of PPG module (with base address taken from config_file). If the module is actually generating patterns or is stopped (not at 0 as it is after reset), the function performs simplified initialisation without terminating current PPG action otherwise full initialisation is done. During full initialisation PPG memory is tested and next filled with STOP commands. Loading pattern file and starting patterns generation is also possible during full initialisation.

The function sets mod_hndl to the module handle value. Handle value is required as a input parameter for other PPG library functions.

The function must be the first PPG function called in user application.

PPG_get_parameters

short PPG_DECL PPG_get_parameters(short hndl,PPGdata far * data);

Input parameters: hndl requested module's handle
 *data pointer to resulting structure

Return value: 0 - on success,
 <0 - on error ,returns error code

PPG_get_parameters fills "data" structure with the contents of internal library data structure of the module with handle equal to hndl.

Definition of PPGdata structure is given in ppg_def.h file.

PPG_set_clk_source

short PPG_DECL PPG_set_clk_source(short hndl,short source);

Input parameters: hndl requested module's handle
 source clock source :
 0 - internal
 1 - external EXTCLK0
 2 - external EXTCLK1

Return value: 0 - on success,
 <0 - on error ,returns error code

The function sets the source of the clock which will be used during pattern generation.

PPG_get_clk_source

short PPG_DECL PPG_get_clk_source(short hndl,short *source);

Input parameters: hndl requested module's handle
*source pointer to result variable

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets the source variable according to the clock source value used by module with handle 'hndl' during generating patterns.

PPG_set_res_pattern

short PPG_DECL PPG_set_res_pattern(short hndl,unsigned long pattern);

Input parameters: hndl requested module's handle
pattern new reset pattern value

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets the reset pattern of the module hndl to the value 'pattern'. Reset pattern is sent on the module outputs after making module reset (by software or by hardware).

PPG_get_res_pattern

short PPG_DECL PPG_get_res_pattern(short hndl,unsigned long *pattern);

Input parameters: hndl requested module's handle
*pattern pointer to result variable

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets the pattern variable according to the actual value of reset pattern of the module hndl.

PPG_get_status

short PPG_DECL PPG_get_status(short hndl,short far *status);

Input parameters: hndl requested module's handle
*status pointer to result variable

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets the pattern variable according to the actual value of the module hndl status. Status bits are defined in ppg_def.h file.

PPG_start

short PPG_DECL PPG_start(short hndl,unsigned short start_adr);

Input parameters: hndl requested module's handle
start_adr start pattern number

Return value: 0 - on success,
<0 - on error ,returns error code

The function loads initial pattern number with value 'start_adr' and starts generation of patterns by the module hndl.

PPG_stop

short PPG_DECL PPG_stop(short hndl,unsigned short far *stop_adr);

Input parameters: hndl requested module's handle
stop_adr pointer to stop pattern number variable

Return value: 0 - on success,
<0 - on error ,returns error code

The function stops generation of the patterns by the module hndl. The function sets 'stop_adr' according to the actual pattern number. This value can be next used to restart pattern generation using function PPG_start.

PPG_reset

short PPG_DECL PPG_reset(short hndl);

Input parameters: hndl requested module's handle

Return value: 0 - on success,
<0 - on error ,returns error code

The function stops generation of the patterns by the module hndl (if it is in progress) , set actual pattern number to 0 and sends 'reset pattern' on the module outputs.

PPG_get_eeprom_data

short PPG_DECL PPG_get_eeprom_data(short hndl,PPG_EEP_Data far *eep_data);

Input parameters: hndl requested module's handle
*eep_data pointer to result structure

Return value: 0 - on success,
<0 - on error ,returns error code

The function fills eep_data structure with the data read from PPG module's EEPROM.

See ppg_def.h file for the definition of PPG_EEP_DATA structure.

PPG_write_eeprom_data

short PPG_DECL PPG_write_eeprom_data(short hndl,unsigned short write_enable, PPG_EEP_Data far *eep_data);

Input parameters: hndl requested module's handle
write_enable value which enables writing
*eep_data pointer to result structure

Return value: 0 - on success,
<0 - on error, returns error code

The function writes the contents of eep_data structure to PPG module's EEPROM. The function is not intended to use by the normal user.

See ppg_def.h file for the definition of PPG_EEP_DATA structure.

PPG_load_pattern

short PPG_DECL PPG_load_pattern(short hndl,char far *pat_file);

Input parameters: hndl requested module's handle
*pat_file pattern file name

Return value: 0 - on success,
<0 - on error ,returns error code

The function interprets the contents of 'pat_file' line by line and writes subsequent patterns found in the file to PPG memory.

After finding syntax error in the file interpretation is stopped and function returns with error code.

After reading the whole file the second phase is executed in which correct jump addresses are written to PPG module memory. The function reads PPG memory next and checks, whether patterns were written correctly.At the end function PPG_reset is called.

Description of PPG pattern files language and structure can be found in 'PPG-100 Bedienungsanleitung' manual.

PPG_write_pattern

short PPG_DECL PPG_write_pattern(short hndl,unsigned short adr, unsigned short count,unsigned long *buffer);

Input parameters: hndl requested module's handle
adr first pattern number in PPG memory which will be overwritten
count number of patterns to write
*buffer buffer which contains patterns

Return value: 0 - on success,
<0 - on error ,returns error code

The function writes the contents of the 'buffer' interpreted as PPG patterns (two 32-bits words per one pattern = Control word and Output pattern) to PPG memory.

Adr + count shouldn't be greater than PPG memory size. Buffer should contain correct number of patterns.

Writing to PPG memory is not possible during pattern generation.

PPG_read_pattern

```
short PPG_DECL PPG_read_pattern(short hndl,unsigned short adr, unsigned short count,unsigned long *buffer);
```

Input parameters:	hndl	requested module's handle
	adr	first pattern number in PPG memory which will be read
	count	number of patterns to read
	*buffer	buffer which will be filled with patterns

Return value: 0 - on success,
<0 - on error ,returns error code

The function reads 'count' patterns from the PPG memory starting from pattern number 'adr' and writes them to 'buffer'.

Adr + count shouldn't be greater than PPG memory size. 'Buffer' should have enough space to contain 'count' patterns (two 32-bits words).

Reading PPG memory is not possible during pattern generation.

PPG_fill_pattern

```
short PPG_DECL PPG_fill_pattern(short hndl,unsigned short adr, unsigned short count,unsigned long *buffer);
```

Input parameters:	hndl	requested module's handle
	adr	first pattern number in PPG memory which will be written
	count	number of patterns to write
	*buffer	buffer which contains pattern to be written

Return value: 0 - on success,
<0 - on error ,returns error code

The function overwrites 'count' patterns in the PPG memory starting from pattern number 'adr' with value taken from 'buffer'.

Adr + count shouldn't be greater than PPG memory size. 'Buffer' should contain one pattern (two 32-bits words).

Filling PPG memory is not possible during pattern generation.

PPG_set_memptr

```
short PPG_DECL PPG_set_memptr(short hndl,unsigned short adr);
```

Input parameters: hndl requested module's handle
 adr pattern number in PPG memory

Return value: 0 - on success,
 <0 - on error ,returns error code

The function sets actual pattern number (memory pointer) in the PPG memory.

It is low level function not intended for normal use.

Adr shouldn't be greater than PPG memory size.

Setting PPG memory pointer is not possible during pattern generation.

PPG_get_memptr

```
short PPG_DECL PPG_get_memptr(short hndl,unsigned short *adr);
```

Input parameters: hndl requested module's handle
 *adr pointer to result variable

Return value: 0 - on success,
 <0 - on error ,returns error code

The function sets 'adr' according to the actual pattern number (memory pointer) in the PPG memory.

Reading PPG memory pointer during pattern generation can result in not correct value.

PPG_get_id

```
short PPG_DECL PPG_get_id (short hndl,short far *id);
```

Input parameters: hndl requested module's handle
 *id pointer to result variable

Return value: 0 - on success,
 <0 - on error ,returns error code

The function sets 'id' according to the PPG module identification code.

PPG_test_id

```
short PPG_DECL PPG_test_id (short hndl);
```

Input parameters: hndl requested module's handle

Return value: 0 - on success,
 <0 - on error ,returns error code

The function reads PPG module identification code and compares it with predefined value.

The function returns error code when id is not OK , otherwise it returns 0.

PPG_get_version

short PPG_DECL PPG_get_version (short hndl,short far *version);

Input parameters: hndl requested module's handle
*version pointer to result variable

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets 'version' according to the PPG module FPGA version.

PPG_set_mode

short PPG_DECL PPG_set_mode (short hndl,short mode);

Input parameters: hndl requested module's handle
mode 0 - simulation mode ,1 - hardware mode

Return value: 0 - on success,
<0 - on error ,returns error code

The function sets mode of PPG library operation.

The function can be used to set the simulation mode of the PPG library operation after hardware test error. In this mode all accesses to the PPG module are replaced by a simulation sequence. Buffer in PC memory is also allocated to simulate PPG memory.

When 'mode' = 1 , hardware test is executed. Mode is changed only, when test's result is OK.

PPG_get_mode

short PPG_DECL PPG_get_mode (short hndl);

Input parameters: hndl requested module's handle

Return value: 1 - hardware mode,
0 - simulation mode

The function returns mode of PPG library operation.

PPG_get_error_string

short PPG_DECL PPG_get_error_string(char far *string);

Input parameters: *string pointer to error string

Return value: 0

The function copies library error string contents to 'string' and next clears library error string.

Appendices

Appendix A: Configuration File (PPG100.INI)

The IO base address as well as basic settings are stored in the configuration file (normally *PPG100.INI*)

If more than one PPG-100 Modules are installed in one PC for each module a separate configuration file is necessary. PPG_INIT must be called one time for every module (with the appropriate configuration file name)

The structure of the configuration file is described below. The text of the configuration file is printed in type writer letters, commenting text is printed like this text.

Contents of PPG100.INI:

```
;      PPG100 initialization file
;-----
;      PPG parameters have to be included in .ini file only when parameter
;      value is different from default.
```

Comments are preceded by a semicolon. Sections are enclosed in brackets as below:

```
[base]
```

```
baseadr      = 0x380      ; base I/O address   (0 ... 0x3F0, increment 0x10)
```

Within a section values are assigned to the parameters. In the section [base] the value 0x380 is assigned to the parameter baseadr. If for any reason the IO base address must be changed the value 0x380 must be changed to the new address. See (*Changing the Module Address*)

```
[device]
```

```
clk_source   = 0          ; clock source: 0 - internal (default),
;                  1 - EXTCLK0
;                  2 - EXTCLK1

res_pattern  = 0x00000000 ; reset pattern (32 bits)(default = 0x00000000)
; This pattern will be written to the outputs when
; the PPG-100-module is reset.
```

Within the section [device] the default values of Clock Source and Reset Pattern can be defined. The values can be adjusted to the requirements of the user.

Appendix B: Pattern File (EXAMPLE.PAT)

The following pattern file shows the file structure and describes the various methods to define patterns.

```
PPG-100_PATTERN          ; this item identifies the file as a pattern
                        ; file for PPG_WIN or PPG_DOS. The file must
                        ; start with this key word.

; Comments are preceded by ";". The text from ";" to the end of the line
; is ignored (not interpreted).

; The various elements of the pattern definitions are separated by white
; spaces. White spaces are one or more <blank>, <TAB>, <CR>, <LF>.

; General Parameters which are valid for the whole pattern sequence are
; preceded by "#". Only the following two parameters are of this type:

#res_pattern 0x00000000   ; this pattern is fed to the outputs on a
                        ; RESET of the PPG-100 or a PC RESET.
                        ; Directly after loading a RESET of the
                        ; PPG-100 is forced. "0x...." is used for
                        ; hexadecimal values.

#clock_source 0          ; Clock Source: 0 internal Oscillator
                        ;
                        ;           1 EXTCLK0
                        ;           2 EXTCLK1

:start                  ; Labels are preceded by ":". Labels may be
                        ; as jump targets. Label names must start
                        ; with a letter.

; pattern definitions:
;-----
;      Time          !digital pattern    %analog output
$Time 15000000         !0x5555          ; 1.5 sec Digital Outputs: 0x00005555
$Time 1                !0x80005555       ; 100 ns Digital Outputs: 0x80005555
$Time 10000            !0x80005555 %0.9 ; 10 µs Digital Outputs: 0x80??5555
                        ; Analog Output 0,9 V
$Time 10000            %1.765           ; 10 µs Digital Outputs: 0x80??5555
                        ; Analog Output 1,765 V
;-----
;
;      !digital pattern
$Stop                  !0xaaaa          ; stops pattern sequence execution.
                        ; Digital Outputs: 0x0000AAAA
                        ; The sequence may be continued with
                        ; START.
;-----
;      Trigger-      !digital pattern
;      condition
$Wait 1???             !0                ; Wait until TRIG3 = 1
                        ; Ausgänge: 0x00000000
$Wait 0000             !0x80000000       ; Wait until TRIG3-TRIG0 = 0
                        ; Ausgänge: 0x80000000
```

```

; Digital patterns may be defined in binary : "0b..." , White spaces are
allowed; therefore all 32 bits must be defined)
;-----
;      Zeit      !Bitmuster

$Time  10      !0b 0000 0 000 11 00 00000000 0000 0000 1000
$Time  10      !0b 0000 1 000 11 00 00000000 0000 0000 1001
$Time  10      !0b 0000 0 000 11 10 00000000 0000 0000 1010

$Jump  ???? start !0      ; Sprung zur Marke "start"
                        ; (ohne Triggerbedingung)

; End of file
;-----

```

Pattern Types

Following Pattern Types are defined:

```

$Time <time>          [!<digital pattern>] [%<analog value>]
$Wait <trigger condition> [!<digital pattern>] [%<analog value>]
$Jump <trigger condition> <label>          [!<digital pattern>] [%<analog value>]
$Stop                [!<digital pattern>] [%<analog value>]

```

If the digital pattern is not given the previous digital pattern is used. If the analog value is defined bits 16..23 of the digital pattern are overwritten with the analog value.

Time

The time is expressed in multiples of the minimum pattern time (100 ns @ internal oscillator). In case of external clock two clock pulses are required for one pattern. So if the time 10 is specified it will require 20 clock pulses to expire. The time is given in decimal.

Trigger Condition

If a trigger condition is specified the execution of the pattern sequence is suspended until the specified trigger condition is met. A trigger condition is a 4 digit number which specifies the state of the four trigger inputs TRIG3, TRIG2, TRIG1, TRIG0 (in this sequence). Each digit may be "0", "1" or "?" for "don't care"

Labels

The targets for pattern type *Jump* must be defined as labels. A label starts with ":" followed by the label name e.g. ":LABEL1"

Patterns

Digital Patterns may be defined binary, hexadecimal or decimal:

```
$Time 100    ! 0b 0001 0 010 10001101 00 0101 01 10 0111 1000    ; binary
$Time 100    ! 0x12345678                                          ; hexadecimal
$Time 100    ! 305419896                                          ; decimal
```

Of course the result is in all three cases the same. Depending on the task one of the formats is most suitable.

The binary format allows insertion of white spaces to separate groups of bits. Therefore in the binary format all 32 bits must be explicitly defined.

Hexadecimal and decimal values are right adjusted, i.e. 0x1 is the same as 0x00000001.

To make the patterns unreadable for everybody they can be entered in decimal too!

Analog Value

The analog output is coded into bits [23:16] of the digital pattern. The value 0 represents 0.0 V the value 0xFF 2.0 V at the unloaded output or 1.0 V at a load of 50 Ω .

The voltage level can be given directly in V within the pattern definition:

```
$Time 100    ! 0x12345678 %0.7    ; 0.7 V at the (unloaded) analog output
```

Bits [23:16] of the digital pattern are overwritten in this case with the analog value.

Appendix C: Contents of the Pattern Memory

The contents of the pattern memory is interpreted by the control hardware as follows:

memory address MA[16:0]	data D[31:0]
00000	control information pattern 0
00001	output pattern pattern 0
00002	control information pattern 1
00003	output pattern pattern 1
:	:

D[23:16] of the output pattern are fed to the DAC which generates the analog output voltage. The output impedance of the output buffer is 50 Ω .

The control information contains either the time for which a pattern should remain stable or a command:

D[31:26]	D[25:24]	D[23:16]	D[15:0]	Description
111111	00	TRIGM[3:0] TRIGV[3:0]	don't care	WAIT FOR TRIGGER. First the output pattern is transferred to the outputs. Then the execution of the pattern sequence is suspended until the trigger condition is met. The WAIT bit in the status register is set. Definition of trigger conditions see below.
111111	01	TRIGM[3:0] TRIGV[3:0]	PA[15:0]	JUMP to pattern no. (PA[15:0]), when the trigger condition is met.
111111	10	don't care	don't care	Reserved, currently not used.
111111	11	don't care	don't care	STOP. First the output pattern is transferred to the outputs. Then the execution of the pattern sequence is stopped and the STOP bit in the status register is set.
TIM[31:26] if ≠111111	TIM[25:24]	TIM[23:16]	TIM[15:0]	Time for which the output pattern should remain stable. (TIM[31:0]+1) * 100 ns @ internal oscillator

Trigger Condition

For all four external trigger inputs (TRIG[3:0]) there is a mask bit (TRIGM[3:0]) and a value bit (TRIGV[3:0]). If the mask bit is 1 the respective input must be equal to the value bit to meet the trigger condition for this input. To meet the whole trigger condition the trigger conditions of all four inputs must be met.

Layout of the trigger condition field:

D23	D22	D21	D20	D19	D18	D17	D16
TRIGM[3]	TRIGM[2]	TRIGM[1]	TRIGM[0]	TRIGV[3]	TRIGV[2]	TRIGV[1]	TRIGV[0]

Technical Support

If you have any questions, problems or comments regarding our products please feel free to contact your dealer where you purchased the product or contact us directly:

Becker & Hickl GmbH
Nahmitzer Damm 30
12277 Berlin
Germany
Tel. +49 / 30 / 787 56 32
Fax +49 / 30 / 787 57 34
Email support@becker-hickl.de
WWW <http://www.becker-hickl.de>