

LTC and VITC timecode reader board for IBM-PC's

PCL-5

A1 Copyright

Copyright <c> Alpermann+Velte Electronic Engineering GmbH 1997. All rights reserved.

Information in this publication supersedes that in all previously published material. Specifications and price change privileges reserved.

For further information, contact or your local dealer or

Alpermann + Velte

Electronic Engineering GmbH
D-42369 Wuppertal, Otto-Hahn-Str. 42
Tel.: ++49 - (0)202 - 2441110
Fax: ++49 - (0)202 - 2441115

A2 CE Declaration

We,

Alpermann + Velte

Electronic Engineering GmbH
D-42369 Wuppertal, Otto-Hahn-Str. 42

declare under our sole responsibility that the

PCL-5

to which this declaration relates is in conformity with the following standards:

1. EN 55022, Class B
2. IEC 801-2
3. IEC 801-3 / ENV 50140
4. EN 61000-4-4

A3 General hints for safe operation

Please only use the equipment in dry rooms and according to the directions.

**Damage due
to transportation:**

In case of obvious damage caused during transportation, please inform the responsible forwarding agency. Please also get directly in touch with your dealer.

Repairs:

As electronic state-of-the-art components have been used in your equipment, no maintenance is required. The unit does not contain any parts which might be repaired by yourself. For this reason, any intervention should only be performed by an authorized service partner.

Content	Page
A1 COPYRIGHT	
A2 CE DECLARATION	
A3 GENERALS HINTS FOR SAFE OPERATION	
B1 GENERAL	1
B2 CONNECTIONS AND TECHNICAL SPECIFICATIONS	1
Technical specifications	1
Connections	2
Configuration of input mode (SW3)	2
Addressing (SW1)	2
Interrupt configuration (SW2)	3
B3 GENERAL DATA ACCESS	4
B4 TIMECODE READING	5
B5 CTL FUNCTIONS	7
B6 LTC FUNCTION	7
B7 TIMECODE COMPARATOR	7
C1 PCL-5 PROTOCOL	9
Description of registers	9
Description of commands	13
C2 DLL FOR WINDOWS OPERATING SYSTEMS	18
File README.TXT	18
File AVPCL.H	23

B1 General

PCL-5 is a PC board for the 8 bit ISA bus with a VITC and a LTC reader each and a CTL counter.

32 I/O addresses are assigned which permit to activate the 256 bytes of a dual-ported RAM. The data may be interchanged by polling or interrupt.

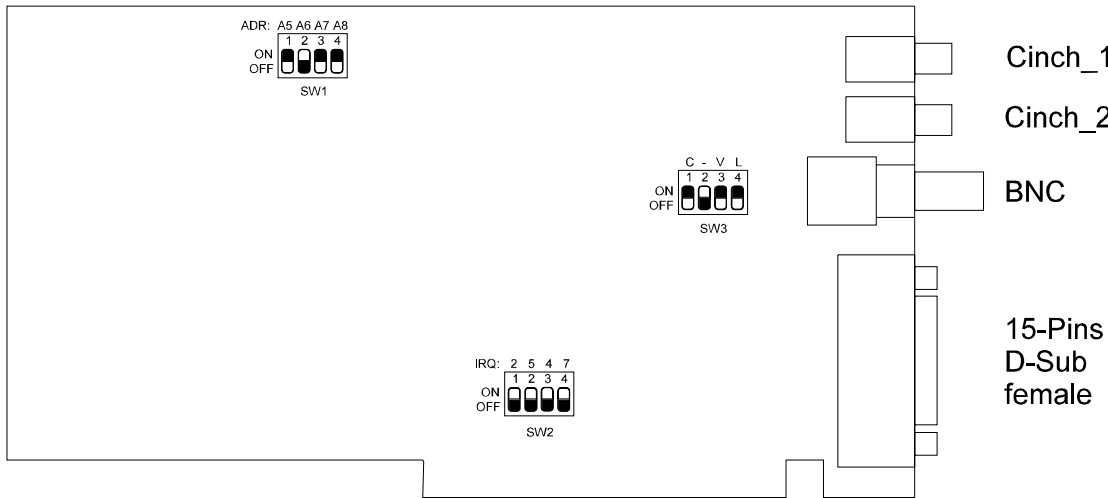
PCL-5 is fully compatible with the PCL-10 predecessor model. Therefore existing software has not to be changed. The only exception is the configuration of interrupts: IRQ3 cannot be selected, for that IRQ5 has to be selected.

B2 Connections and technical specifications

Technical specifications

Format	ISA 8 Bit; PC bus driver: 74 HCTxx
Operating voltage	+5V ± 5% / 12V ± 5%
Power consumption	approx. 250 mA (5V) / approx. 25 mA (+12V)
Acceptable ambient temperature	5° - 40° C
VITC	Format: SMPTE/EBU Data level: 0,38 - 1,1 Vpp VITC noise level: < 30mV
LTC	Input level: 30 mV - 5V Input impedance: 47 K ¹ / ₁₅ - 60times nominal speed
CTL	Input level: „high“: +2,5 V - 15 V / „low“: < +1,5V Frequency: 0 Hz - 15 kHz

Connections



- LTC-Input: Cinch_1 and Cinch_2 if balanced signal
+ SW3.4=off
Cinch_2 only if unbalanced signal +SW3.4=on
- VITC-Input: at BNC, SW3.3 switches 75 Ohm termination on/off.
- CTL-Input: at 15-Pins D-Sub:
Pin 7 = CTL pulse
Pin 8 = CTL direction
Pin 13 = GND

Configuration of input mode (SW3)

- 1 = C: Reserved for PCL-6
- 2: not connected
- 3 = V: Video termination
ON 75 Ohm, termination on
OFF termination off
- 4 = L: LTC input
ON unbalanced LTC input at Cinch_2
OFF balanced LTC input at Cinch_1 and Cinch_2

Addressing (SW1)

The board is designed for the 8 bit PC slot. 32 I/O addresses of the PC are used. The basic address of the board is selected with the SW-1 switch. The address position should be adjusted to avoid any overlapping with other boards.

Switch SW1				Basic address (I/O)	
1 = A5	2 = A6	3 = A7	4 = A8	hexa	decimal
on	on	on	on	\$200	512
off	on	on	on	\$220	544
on	off	on	on	\$240	576
off	off	on	on	\$260	608
on	on	off	on	\$280	640
off	on	off	on	\$2A0	672
on	off	off	on	\$2C0	704
off	off	off	on	\$2E0	736
on	on	on	off	\$300	768
off	on	on	off	\$320	800
on	off	on	off	\$340	832
off	off	on	off	\$360	864
on	on	off	off	\$380	896
off	on	off	off	\$3A0	928
on	off	off	off	\$3C0	960
off	off	off	off	\$3E0	992

\$240 or \$340 are recommended. The factory setting is \$240.

Interrupt configuration (SW2)

If the board should operate with interrupts, one of the interrupt lines must be selected by switch SW2. Any overlapping with other boards should be avoided.

Switch SW2				Interrupt
1 = IRQ2	2 = IRQ5	3 = IRQ4	4 = IRQ7	
on	off	off	off	IRQ-2
off	on	off	off	IRQ-5
off	off	on	off	IRQ-4
off	off	off	on	IRQ-7
off	off	off	off	Polling

Care should be taken to set only one switch to "ON" !

B3 General data access

A dual-ported RAM with 256 bytes controls the exchange of data between PCL-5 and the PC. The RAM is subdivided into 16 registers with 16 bytes each. To access to a specific byte, it is at first necessary to set the register address into the address register (to the basic address + \$10), then a transparent access is possible to 16 bytes via the basic address + the low nibble of the register.

Register	Description
\$0	Mixed timecode
\$1	CTL counter
\$2	LTC timecode
\$3	VITC timecode
\$4	Reserved
\$5	Reserved
\$6	Reserved
\$7	Reserved
\$8	Timecode comparator (event)
\$9	Reserved
\$A	General data transfer
\$B	Status
\$C	Serial control register
\$D	Serial receiver
\$E	Reserved
\$F	Command register

A detailed description is given in the protocol (please see appendix).

A read or write access to a specific byte consists of two steps:

1. Write operation: Data = address of the register
 Address = basic address + \$10
2. Read or write: Data = read or write data
 Address = basic address + byte address (within the register)

Example: Read byte \$B of register \$1; the basic address being set to \$240:
 1. Write data = \$1 to I/O address \$250
 2. Read data of I/O address \$24B

Step 1 may be omitted, if the same register is repeatedly accessed to.

Commands to PCL-5 can be transmitted via register \$F. The procedure is as follows:

- If data shall also be transmitted with the command, these data must be written into the bytes \$0..\$7 of register \$F (see appendix).
- Write the command byte (see appendix) into byte \$F of register \$F.
- PCL-5 handles the command and then sets the command byte \$F in register \$F to zero. Any eventually resulting data are placed into byte \$0..\$7 of register \$F by PCL-5.

B4 Timecode reading

The VITC register (\$3) contains the data of the VITC reader, the LTC register (\$2) contains the data of the LTC reader, and the CTL register (\$1) contains the data of the CTL counter which counts pulses irrespective of the timecode.

The mixed register (\$0) can contain VITC or LTC timecode or a time counted by CTL pulses, if no timecode can be read.

The independent CTL counter of register \$1 does not appear in the mixed register. Therefore if the CTL is switched on or off for the mixed register, the timecode bypass with CTL pulses is switched on or off (drop-out).

The 16 bytes of the timecode register are organized as follows:

Byte	Description		
\$0	BCD frames		
\$1	BCD seconds		
\$2	BCD minutes		
\$3	BCD hours		
\$4	Flags:		
	Bits for VITC	for LTC	for CTL
	0 Bit 14 (drop)	Bit 10 (drop)	-
	1 Bit 15 (colorframing)	Bit 11 (colorfr.)	-
	2 Bit 35 (field NTSC)	Bit 27	-
	3 Bit 55	Bit 43	-
	4 Bit 74	Bit 58	-
	5 Bit 75 (field PAL)	Bit 59	-
	6 -	-	sign (0=+)
	7 -	direction (0=forw.)	direction (0=forw.)
\$5			
\$6			
\$7			
\$8	User 1 (low nibble) and 2 (high nibble)		
\$9	User 3 (low nibble) and 4 (high nibble)		
\$A	User 5 (low nibble) and 6 (high nibble)		
\$B	User 7 (low nibble) and 6 (high nibble)		
\$C			
\$D			

Byte	Description	
\$E	Request:	\$80 = asynchronous \$40 = synchronous
\$F	New data flag:	\$20 = permanent \$80 = VITC data \$40 = LTC data \$20 = CTL data

Timecode reading in the polling mode:

Asynchronous and synchronous request:

1. Store \$80 or \$40 in the request byte of the register. \$80 (asynchronous) means that the timecode currently stored in the memory shall be transmitted to the register. \$40 (synchronous) means that the next read timecode value shall be transmitted to the register.
2. Read the request byte until it appears as \$00; then the timecode values can be read out. The new data flag indicates, if VITC, LTC or CTL have currently been transmitted into the mixed register.
3. Start again with 1.

Permanent request:

1. Set the new data flag to \$00.
2. Store \$20 into the request byte of the register. \$20 (permanent) means that every newly read timecode shall automatically be transmitted into the register.
3. Read the new data flag until \$00 is discovered, then read the timecode values. Set the new data flag again to \$00 and start with 3.
4. Set the new data flag to \$00 again and start with 3.

Reading timecode with interrupts:

1. Select an interrupt line (see chapter 3) and activate the interrupt of the desired register with command \$03 or \$04 (see protocol).
2. Store a \$40 in the request byte of the register which shall activate the interrupt.
3. As soon as the interrupt line is active, read byte \$E of register \$F to detect who activated the interrupt.
4. Read the data of the register and set the request byte again to \$40, if further interrupts are desired.
5. Set byte \$E of register \$F to \$00 to release the interrupt line.
6. Start with 3.

B5 CTL functions

The independent counter:

The counting values of this counter are written into register \$1. The CTL counter counts pulses as sub-seconds, seconds, minutes and hours. It is an up/down counter; when counting up, the carryover of the hours is made from BCD 23 to 00. When counting down, the carryover depends on the mode (command \$40):

If CTL was selected with polarity, the carryover is from +0 to -1 sub-second (the polarity changes). If CTL was selected without polarity, the carryover is from 0 to 23 hours.

After the reset the sub-seconds correspond to the frames of the timecode with framerate = 25. It is possible to set a different rate with the command \$32. If p.e. one pulse per second is delivered at normal speed, the counting rate must be set to 0 or 1. If the seconds shall be subdivided into hundreds, the counting rate must be set to \$9A.

The timecode bypass (drop-out):

To bypass timecode drop-outs with CTL, CTL must be selected for the mixed register with command \$20. This counter operates without polarity, it counts up/down like the timecode (00:00:00:00 - max. 23:59:59:59), and the subseconds correspond always to the frames of the timecode. However, if the CTL counting rate differs from the frame rate, the CTL counting rate must be set with command \$31 to avoid any deviations from the timecode. Example: If CTL pulses with 18 pulses per second are applied (at normal speed), but a bypass rate of 25 is used, cueing to a specific position of an audio-tape recorder in fast-forward mode will always be too short.

Command \$41 (reset CTL counter) always acts upon both counters, but the drop-out counter would immediately be overwritten by any timecode value. Command \$42 (preset CTL counter) only acts upon the independent counter.

B6 LTC function

The LTC reader reads time and user bits simultaneously in the up/down mode at any operative speed. The read time data are checked for their plausibility. Additionally a check is performed, that the timecode values appear in continuous order (increasing for the up-counting code, decreasing for the down-counting code). If both checks are o.k., the LTC data are transmitted into the registers. The continuity check may be switched on/off with command \$11.

B7 Timecode comparator

PCL-5 permits to set a timecode event, i.e. a timecode value and/or user data which are compared with every value which is newly transmitted to the mixed reg-

ister. In case of identical values, a flag is set which can be used as a comparator in polling mode. It is also possible to activate an interrupt in case of identical values.

Additionally the time difference between the event time and the time in the mixed register is calculated and stored in register \$8. The difference is transmitted to register \$8 simultaneously with the new values in the mixed register. The difference is expressed in three different ways (see also remark *1 of the protocol):

Difference 1 = (event time) - (time of the mixed register), i.e. the simple arithmetic difference.

Difference 2 = the amount of difference 1 or the time distance, respectively, i.e.:
Actual (event-time) \geq (time of the mixed register) \implies difference 2 = difference 1,
Actual (event time) $<$ (time of the mixed register) \implies difference 2 = (time of the mixed register) - (event time)

Difference 3 = same as difference 2, but modulo 12 hours, i.e. the distance is always \leq 12 hours:

Actual difference 2 \leq 12:00:00:00 \implies difference 3 = difference 2,
Actual difference 2 $>$ 12:00:00:00 \implies
Difference 3 = amount (00:00:00:00 - difference 2)

Difference 3 is the time distance from tape positions extending over a timecode carryover of 00:00:00:00 or 24:00:00:00, respectively. This assumes that it is not possible to cover a distance of $>$ 12 hours on a tape.

C1 PCL-5 protocol

Description of registers

Register	Byte	Bit	PC read/write	Description		
\$0	\$0		r	Mixed register frames		
	\$1		r	Mixed register seconds		
	\$2		r	Mixed register minutes		
	\$3		r	Mixed register hours		
	\$4		r	Depends on VITC/LTC/CTL		
	\$5					
	\$6					
	\$7					
	\$8			r	User 1/2	
	\$9			r	User 3/4	
	\$A			r	User 5/6	
	\$B			r	User 7/8	
	\$C					
	\$D			r	BCD framerate, Bit7=1 ==> drop mode	
	\$E			w/r	Request : \$80 = asynchronous \$40 = synchronous \$20 = permanent	*2
\$F			r/w	New data flag: \$80 = VITC data \$40 = LTC data \$20 = CTL data	*2	
\$1	\$0		r	CTL register frames		
	\$1		r	CTL register seconds		
	\$2		r	CTL register minutes		
	\$3		r	CTL register hours		
	\$4		r	CTL status		
			6		Sign : 0 = pos., 1 = neg.	
			7		Direction : 0 = forward, 1 = reverse	
	\$5					
	\$6					
	\$7					
	\$8					
	\$9					
	\$A					
	\$B					
	\$C					
\$D						
\$E			w/r	Request: \$80 = asynchronous \$40 = synchronous \$20 = permanent	*2	
\$F			r/w	New data flag : <> 0 = new data	*2	

Operating Instructions PCL-5

Register	Byte	Bit	PC r/ w	Description		
\$2	\$0		r	LTC register frames		
	\$1		r	LTC register seconds		
	\$2		r	LTC register minutes		
	\$3		r	LTC register hours		
	\$4		0	r	Bit 10 of LTC (drop)	
			1		Bit 11 of LTC (CF)	
			2		Bit 27 of LTC	
			3		Bit 43 of LTC	
			4		Bit 58 of LTC	
			5		Bit 59 of LTC	
			7		Direction: 0 = forward, 1 = reverse	
	\$5					
	\$6					
	\$7					
	\$8			r	User 1/2	
	\$9			r	User 3/4	
	\$A			r	User 5/6	
\$B			r	User 7/8		
\$C						
\$D						
\$E			w/r	Request :	*2	
				\$80 = asynchronous \$40 = synchronous \$20 = permanent		
\$F			r/w	New data flag:	*2	
				<> 0 = new data		
\$3	\$0		r	VITC register frames		
	\$1		r	VITC register seconds		
	\$2		r	VITC register minutes		
	\$3		r	VITC register hours		
	\$4		0	r	Bit 14 of VITC (drop)	
			1		Bit 15 of VITC (CF)	
			2		Bit 35 of VITC (NTSC field mark)	
			3		Bit 55 of VITC	
			4		Bit 74 of VITC	
	\$5				Bit 75 of VITC (PAL field mark).	
	\$6					
	\$7					
	\$8			r	User 1/2	
	\$9			r	User 3/4	
	\$A			r	User 5/6	
	\$B			r	User 7/8	
\$C						
\$D						
\$E			w/r	Request :	*2	
				\$80 = asynchronous \$40 = synchronous \$20 = permanent		
\$F			r/w	New data flag:	*2	
				<> 0 = new data		
\$4				Reserved		
\$5				Reserved		
\$6				Reserved		
\$7				Reserved		

Register	Byte	Bit	PC r/w	Description		
\$8	\$0		r	Event difference 1 frame	*1	
	\$1		r	Event difference 1 second	*1	
	\$2		r	Event difference 1 minute	*1	
	\$3		r	Event difference 1 hours	*1	
	\$4		r	Event difference 2 frames	*1	
	\$5		r	Event difference 2 seconds	*1	
	\$6		r	Event difference 2 minutes	*1	
	\$7		r	Event difference 2 hours	*1	
	\$8		r	Event difference 3 frames	*1	
	\$9		r	Event difference 3 seconds	*1	
	\$A		r	Event difference 3 minutes	*1	
	\$B		r	Event difference 3 hours	*1	
	\$C		r	Status event time:	*1	
			0		Units frames of distance 3 <> 0	
			1		Tens frames of distance 3 <> 0	
			2		Units seconds of distance 3 <> 0	
		3		Tens seconds of distance 3 <> 0		
		4		Minutes of distance 3 <> 0		
		5		Hours of distance 3 <> 0		
		6		Sign modulo 12 hours		
		7		Sign absolute		
	\$D		r	Status event user :		
		0		User 1 of event <> mixed register user		
		1		User 2 of event <> mixed register user		
		2		User 3 of event <> mixed register user		
		3		User 4 of event <> mixed register user		
		4		User 5 of event <> mixed register user		
		5		User 6 of event <> mixed register user		
		6		User 7 of event <> mixed register user		
		7		User 8 of event <> mixed register user		
	\$E		r/w	Flag „event found“	*12	
				Low nibble <> 0 ==> event user found		
				High nibble <> 0 ==> event time found		
	\$F		r/w	New data flag : <> 0 = new data	*2	
				(Is set like the flag of register \$0, request of data via flag \$E of register \$0)		
\$9				Reserved		
\$A	\$0 ... \$E		r	Data, inquired by command \$02		
	\$F		r/w	New data flag : <> 0 = new data	*2	
\$B	\$C		r	Software identification; normal = 0		
	\$D		r	Software version number		
\$C				Free		
\$D				Free		
\$E				Free		

Operating Instructions PCL-5

Page 12

Register	Byte	Bit	PC r/w	Description		
\$F	\$0		w	Data 0 of command		
	\$1		w	Data 1 of command		
	\$2		w	Data 2 of command		
	\$3		w	Data 3 of command		
	\$4		w	Data 4 of command		
	\$5		w	Data 5 of command		
	\$6		w	Data 6 of command		
	\$7		w	Data 7 of command		
	\$8					
	\$9					
	\$A					
	\$B					
	\$C					
	\$D					
	\$E			r/w	Interrupt flag: Bit 0 = 1 : mixed register interrupt Bit 1 = 1 : CTL register interrupt Bit 2 = 1 : LT register interrupt Bit 3 = 1 : VITC register interrupt	*11
	\$F			w	Command	

Description of commands

Command	Data	Default
\$01 Reset	Data 0: Mode \$00 Resets PCL-5 to all defaults (no recorder control). \$01 Sony 9-Pin RS422, timecode via RS422. \$02 As \$01, but adapted to JVC PR 900. \$05 Panasonic AG-7330 RS232, timecode VITC reader. \$06 Sony 9-Pin RS422, timecode LTC/VITC reader. \$07 As \$06, but adapted to JVC PR 900. \$09 JVC BR-S 605/ 622/ 822 (SR-S368) RS232, timecode LTC/VITC reader. \$0A Panasonic IA232TC Interface, for AG-7350, AG-7150, AG-7355, timecode via RS232. \$0B As \$0A, timecode LTC/VITC reader. \$0C Simulation Sony protocol, e.g. „PCL-5 as recorder“.	*15 \$00
\$02 Data request	Data 0: number of register (\$0 - \$9, \$B -\$E); command transfers data of register-no. to register \$A	*6
\$03 Interrupts	Bit 0 : 0/1 = mixed interrupt disable/enable Bit 1 : 0/1 = CTL interrupt disable/enable Bit 2 : 0/1 = LTC interrupt disable/enable Bit 3 : 0/1 = VITC interrupt disable/enable Bit 4 : 0/1 = event time interrupt disable/enable Bit 5 : 0/1 = event user interrupt disable/enable	\$00 *12 *12
\$04 Enable interrupts	Data as command \$03, but only bits = will enable interrupts.	
\$05 Disable interrupts	Data as command \$03, but only bits = 1 will disable interrupts.	
\$10 Reader on/off	Bit 0 : =0 CTL off, =1 CTL on Bit 1 : =0 LTC off, =1 LTC on Bit 2 : =0 VITC off, =1 VITC on	\$07
\$11 LTC sequence	\$00: sequence check on <>\$00: sequence check off	\$00
\$12 Reader source	\$01 = source 1 \$02 = source 2 (only special design; 2x LTC Input)	\$01
\$14 VITC setup	From revision 1.D (7/97) on: the VITC reader can be adjusted to block mode or 2-lines mode. The lines can be selected within the range of 4 - 26. The 1 st line must be equal to or smaller than the 2 nd line: Data 0: 1 st line, hexadecimal values within \$04 - \$1A Data 1: 2 nd line, hexadecimal values within \$04 - \$1A Data 2: = 0 2-lines mode <> 0 block mode After power on the block mode is valid for the complete lines range.	
\$20 Mixed register	Bit 0 : =0 CTL off, =1 CTL on Bit 1 : =0 LTC off, =1 LTC on Bit 2 : =0 VITC off, =1 VITC on The reader, which are linked to the bits, are considered in mixed register (=1) resp. are not considered (=0). This means for CTL: timecode drop out bypass on/off.	\$07
\$21 Priority	\$01 = VITC - LTC (- CTL) \$02 = LTC - VITC (- CTL) This command defines the priority according to which the three signals VITC, LTC and CTL are transmitted to the mixed register. For example: if all signals are available at priority \$01, VITC will be transmitted to mixed register. If VITC fails, LTC will be transmitted, then CTL.	\$01

Command	Data	Default
---------	------	---------

Operating Instructions PCL-5

Page 14

Command	Data	Default
\$30	Framerate \$00 = automatically \$01 = 24 frames \$02 = 25 frames \$03 = 30 frames non drop mode \$04 = 30 frames drop mode	\$00
\$31	CTL rate for time- code drop outs \$00 = same as timecode framerate \$01 - \$FF = pulses per second at normal play speed	\$00 *20
\$32	CTL rate for inde- pendent counter Data 0 : rate of sub-seconds, if = \$00 or \$01 a whole second is counted each pulse. Range : BCD 00 - 99 and \$9A, to count from 00 to 99 in sub-seconds.	\$25
\$40	CTL mode Data 0 : Bit 0 = 0 : CTL without sign Bit 0 = 1 : CTL with sign	\$01
\$41	CTL reset	
\$42	CTL preset Data 0 = frames Data 1 = seconds Data 2 = minutes Data 3 = hours	
\$44	CTL direction \$00 : H = reverse, L = forward \$01 : L = reverse, H = forward	\$01
\$50	Set event time Data 0 = frames Data 1 = seconds Data 2 = minutes Data 3 = hours High nibble of flag "found event" is reset.	
\$51	Set event user Data 0 = user 1/2 Data 1 = user 3/4 Data 2 = user 5/6 Data 3 = user 7/8 Low nibble of flag "found event" is reset.	

*1 Event difference

Difference 1 = (event time) - (mixed reg. time)

Difference 2 : (event time) >= (mixed reg. time) ==>
 difference 2 = difference 1
 (Event-time) < (mixed reg. time) ==>
 difference 2 = (mixed reg. time) - (event time)

Difference 3 : difference 2 <= 12:00:00:00 ==>
 difference 3 = difference 2
 difference 2 > 12:00:00:00 ==>
 difference 3 = |(12:00:00:00 - difference 2)|

Sign absolute (Bit 7 of status event time) :
 (event time) >= (mixed reg. time) ==> sign absolute = 0
 (event time) < (mixed reg. time) ==> sign absolute = 1

Sign modulo 12 hours (Bit 6 of status event time) :
 difference 2 <= 12:00:00:00 ==>
 sign modulo 12 hours = sign absolute
 difference 2 > 12:00:00:00 ==>
 sign Modulo 12 hours = inverse to sign absolute

Example :

Event time	Mixed register	Difference 1	Difference 2	Difference 3	Status event time, bits 76543210
00:00:00:00	00:00:00:01	23:59:59:24	00:00:00:01	00:00:00:01	11000001
00:00:00:01	00:00:00:00	00:00:00:01	00:00:00:01	00:00:00:01	00000001
00:00:00:00	12:00:00:00	12:00:00:00	12:00:00:00	12:00:00:00	11100000
00:00:00:00	11:59:59:24	12:00:00:01	11:59:59:24	11:59:59:24	11111111
00:00:00:00	23:00:00:00	01:00:00:00	23:00:00:00	01:00:00:00	10100000
00:00:00:00	01:00:00:00	23:00:00:00	01:00:00:00	01:00:00:00	11100000
23:00:00:00	01:00:00:00	22:00:00:00	22:00:00:00	02:00:00:00	01100000
23:59:59:24	00:00:00:00	23:59:59:24	23:59:59:24	00:00:00:01	01000001
11:11:11:11	22:22:21:21	12:48:49:15	11:11:10:10	11:11:10:10	11111010
22:22:21:21	11:11:11:11	11:11:10:10	11:11:10:10	11:11:10:10	00111010
13:00:00:00	01:00:00:00	12:00:00:00	12:00:00:00	12:00:00:00	00100000

***2 Data transfer (register \$0-\$8), PC part**

via synchronous/asynchronous request :

1. Set request flag.
2. Read request flag until \$0 is detected.
3. Transfer data.

via permanent request :

1. Set new data flag to zero.
2. Read new data flag until it is <> zero.
3. Transfer data

***6 Command \$02, register data**

1. Reset the new data flag of register \$A (\$3AF).
2. Write the number of the desired register to byte 0 of command data, then write \$2 to command.
3. Read the new data flag (\$3AF) until it is <>0, then read the data \$0 - \$E of register \$A.

***11 Data transfer via interrupt**

1. Set the interrupt bit of the desired register with the command \$3 or \$4 (the corresponding reader should not be switched off by command \$10).
2. Set the request byte (synchronous/asynchronous) of the timecode register.
3. The interrupt occurs at the same time as the resetting of the request byte.
4. Detect the timecode register, take over the data and set the request byte again.
5. Reset the interrupt flag.

Remark:	Only one bit of the interrupt flag is set at one time. If several interrupts are applied at the same time, they are batched and activated one after the other. As long as the interrupt flag \$E of register \$F is not zero, no new interrupt is enabled.
---------	---

***12 "event found" flag**

The event comparator uses the data of the mixed register, i.e. the calculations and comparisons for the event register \$8 are performed with the new values of the mixed register. In case of identical values (time or user), the "event found" flag is set. If the interrupt "on" was selected for the events, it becomes active at this moment of time. The "event found" flag is only reset with the command \$50 or \$51 from PCL-5.

***20 CTL rate**

This CTL rate does not act upon the independent CTL counter. If "double CTL" is applied, set data = \$32 (50 Hz) for frame rate = 25. If frame pluses are applied, data = frame rate can be set (p.e. \$19, in case of frame rate = 25), or data = \$00 can be selected.

C2 DLL for Windows Operating Systems

File README.TXT

AVPCL: Driver for the Alpermann+Velte PCL Family

=====

Functions

AVPCL comprises drivers for the PC boards PCL1, PCL2, PCL3, PCL5, PCL6, PCL10 and PCL12 by Alpermann+Velte. These drivers enable the boards to be run with the operating systems MS-DOS, Windows 3.1x, Windows 95, and Windows NT 4.0 (i386).

Installation

Add a new directory named AVPCL onto the hard disk. Then copy the content of the floppy disk including all subdirectories into this directory.

If you don't have that floppy disk but only the AVPCL.ZIP file, then unpack the file as follows:

```
PKUNZIP -d AVPCL
```

If you wish to run your PCL card with Windows NT, log to Windows NT as administrator. Then start the SETUP.EXE program in the WINNT directory to install the Kernel mode driver AVPCLNT.SYS.

Survey

The floppy disk contains a particular subdirectory for each one of the following operating system families: DOS, Windows 16 bit, Windows 32 bit, and Windows NT. In addition to the actual drivers, they dispose of the two demonstration programs PCLTEST and LIBTEST. And you will find a MAKE.BAT file for demonstrating the demonstration program shave to be compiled.

The last of the subdirectories is COMMON containing all files which are identical in the different operating systems. The other subdirectories have a reference to these files.

Definitions

The COMMON\AVPCL.H file contains all declarations required to run the drivers. This file is commented extensively, serving also as a programmer's reference. AVPCL.H is identical for all operating systems, enabling the same C source text to be used in different configurations by re-compiling, at least as far as the

access to the PCL hardware is concerned. Both of the demonstration programs make use of this facility.

Borland Pascal and Delphi programmers find a unit in the file COMMON\AVPCL.PAS. This file is the same for all operating systems, too.

The files COMMON\AVPCL.TXT and COMMON\AVPCL32.TXT are for Visual Basic programmers. They contain all declarations for 16 and 32 bit Visual Basic programs. You may use the API-Reference (APILOAD.EXE) from the Visual Basic package to browse this files.

User programs compiled for the 32-bit operating systems Windows 95 and Windows NT are also binary compatible, with a DLL determining whether the hardware access will be executed directly (Windows 95) or via a Kernel mode driver (Windows NT).

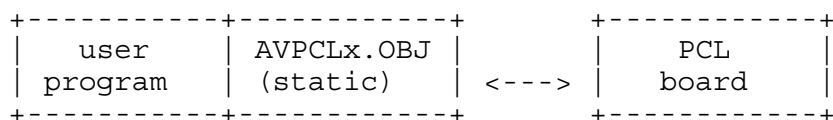
Demonstration programs

The COMMON\PCLTEST.C program demonstrates how the drivers can be used to address the PCL board. Different timecode registers are read out and displayed. Moreover, the remote control functions of the PCL6 serve to control a VCR connected. This way, the program can also be used to install the PCL.

COMMON\LIBTEST.C successively invokes all driver functions available and indicates the results. This means that the program tests the driver rather than the functions of the PCL.

DOS

The MS-DOS drivers are statically linked to the user program:



The memory models Small, Medium, Compact, Large and Huge have a particular OBJ file:

```

DOS\AVPCLS.OBJ  DOS driver for the 'Small' memory model.
DOS\AVPCLM.OBJ DOS driver for the 'Medium' memory model.
DOS\AVPCLC.OBJ DOS driver for the 'Compact' memory model.
DOS\AVPCLL.OBJ DOS driver for the 'Large' memory model.
DOS\AVPCLH.OBJ DOS driver for the 'Huge' memory model.

```

The C function prototypes are to be found in the COMMON/AVPCL.H. file. The DOS\AVPCL.H file has a reference to this.

DOS\LIBTEST.C and DOS\PCLTEST.C refer to the corresponding C demonstration programs in the COMMON subdirectory. DOS\MAKE.BAT serves to compile the demonstration programs. DOS\LIBTEST.EXE and DOS\PCLTEST.EXE are programs compiled with Borland C++ 4.52.

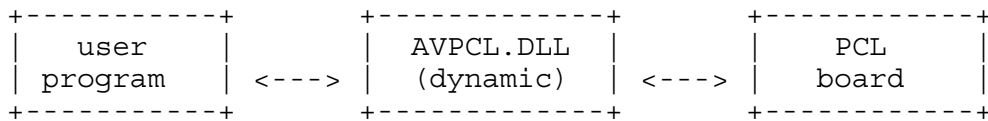
Operating Instructions PCL-5

They run with DOS, Windows 3.x and Windows 95, but not with Windows NT.

The DOS\AVPCL.PAS file refers to COMMON\AVPCL.PAS. It is a unit for Borland Pascal. It makes use of DOS\AVPCLPAS.OBJ. DOS\PCLTEST.PAS is a Pascal demonstration program.

Windows 16 bit

The Windows 16-bit designs access the PCL with DLL WIN16\AVPCL.DLL:



As an import library, WIN16\AVPCL.LIB permits to access the DLL. The corresponding module definition file is WIN16\AVPCL.DEF.

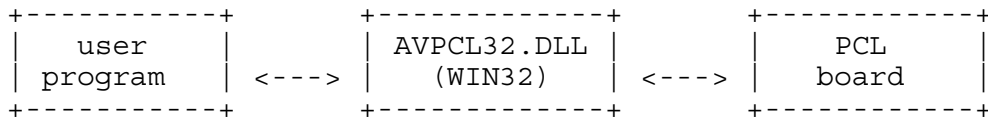
The C function prototypes are to be found in the COMMON/AVPCL.H file. The WIN16\AVPCL.H has a reference to this.

WIN16\LIBTEST.C and WIN16\PCLTEST.C refer to the corresponding C demonstration programs in the COMMON subdirectory. WIN16\MAKE.BAT serves to compile the demonstration programs. WIN16\LIBTEST.EXE and WIN16\PCLTEST.EXE are the programs compiled with Borland C++ 4.52. They run with Windows 3.x and Windows 95, but not with Windows NT. While the program is running, the AVPCL.DLL file must be in the actual library or in the PATH.

The WIN16\AVPCL.PAS file refers to COMMON\AVPCL.PAS. It is a unit for Borland Pascal for Windows. WIN16\PCLTEST.PAS is a Pascal demonstration program.

Windows 32 bit

With Windows 95 the PCL is accessed to via WIN32\AVPCL32.DLL:



As an import library, WIN32\AVPCL32.LIB permits to access the DLL. WIN32\AVPCL32.DEF is the corresponding module definition file.

The C function prototypes are to be found in the COMMON/AVPCL.H file. The WIN32\AVPCL.H file has a reference to this.

WIN32\LIBTEST.C and WIN32\PCLTEST.C refer to the corresponding C demonstration programs in the COMMON subdirectory. WIN32\MAKE.BAT serves to compile the demonstration programs. WIN32\LIBTEST.EXE and WIN32\PCLTEST.EXE are the programs com-

Operating Instructions PCL-5

0x320, you have to add the following entries:

```
IoPortAddress :REG_DWORD :0x240
IoPortAddress1 :REG_DWORD :0x300
IoPortAddress2 :REG_DWORD :0x320
```

AVPCLNT.SYS will only read out the registry entry if the driver has been restarted. If you start again SETUP.EXE at a later time in order to change the base address, you have to restart Windows NT again or, more simple, input the following commands in a console window:

```
net stop avpclnt
net start avpclnt
```

The WINNT\SETUP.C file contains the source text of the installation program, WINNT\SETUP.RC the corresponding resources.

As an import library, WINNT\AVPCL32.LIB permits access to the DLL. The corresponding module definition file is WINNT\AVPCL32.DEF.

The C function prototypes are located in the COMMON\AVPCL.H file. The WINNT\AVPCL.H file has a reference to this.

WINNT\LIBTEST.C and WINNT\PCLTEST.C refer to the corresponding C demonstration programs in the COMMON subdirectory. WINNT\MAKE.BAT serves to compile the demonstration programs. WINNT\LIBTEST.EXE and WINNT\PCLTEST.EXE are the programs compiled with Borland C++ 4.52. They run with Windows NT and Windows 95, depending on the type/version of AVPCL32.DLL used, which are to be found in the WINNT or WIN32 subdirectory. While the program is running, the corresponding AVPCL32.DLL must be in the actual directory or in the path.

The WINNT\AVPCL.PAS file refers to COMMON\AVPCL.PAS. It is a unit for Borland Delphi.

Microsoft Visual C++

The import libraries (*.LIB) are in Object Module Format (OMF). Newer versions of Microsoft Visual C++ can't handle this file format any more. They need files in Common Object File Format (COFF) instead. The file COFF\AVPCL32.LIB is the 32-bit import library for Windows 95 and Windows NT in COFF file format.

Alpermann+Velte
Otto-Hahn-Str. 42
D 42369 Wuppertal
Fon: +49-202-244111-0
Fax: +49-202-244111-5
e-Mail: info@alpermann-velte.com

File AVPCL.H

```

/*****\
 *
 * Macros and function prototypes of Windows DLL AVPCL.DLL
 *
 * Copyright (c) 1995,98 by Alpermann+Velte
 *
 \*****/

/* <avpcl.h> Alpermann+Velte DLL for AV-PCL */
/* @(#)avpcl.h 2.21 8/27/99 (c) 1995-99 Alpermann+Velte */

#ifndef _AV_AVPCL_H
#define _AV_AVPCL_H

#ifdef __cplusplus
extern "C" {
#endif

/*****
 *
 * Features
 *
 *-----
 *
 * This DLL allows access to the following products of Alpermann+Velte:
 *
 * PCL-5 : Single VITC/LTC reader
 * PCL-5L : Single LTC reader
 * PCL-5V : Single VITC reader
 * PCL-6 : Single VITC/LTC reader with recorder remote interface
 *
 * Additionally, it still supports this older products:
 *
 * PCL-1 : Single VITC/LTC reader
 * PCL-2 : Double VITC/LTC reader
 *
 * PCL-3 : Single VITC/LTC reader with recorder remote interface
 * PCL-10 : Single VITC/LTC reader
 * PCL-12 : Single VITC/LTC reader with input selector
 *
 * The DLL supports all features of the PCL cards, excluding hardware
 * interrupts. */

/*****
 *
 * Basics
 *
 *-----
 *
 * AVPCL exports many #defines, typedefs and functions. To avoid naming
 * conflicts, all identifiers start with a prefix: 'PCL_XXX' for #defines
 * and typedefs and 'pclxxx' for functions. Identifiers starting with
 * an underbar (_) are for internal use only. */

/*****
 *
 * Program options
 *
 *-----
 *
 * This options enable special features of AVPCL. To use any of these,
 * a special PCL firmware EPROM is needed. */

// #define PCL_OPT_GETCMD /* get command bytes from device */
// #define PCL_OPT_MTD /* read mtd times */
#define PCL_OPT_REALTIME /* pcl as real time source */

/*****
 *
 * Error codes
 *
 *-----
 *
 * Most functions return a int error code. A value greater or equal to
 * zero means 'no error', negative values mean 'error'. A list of
 * possible return codes is shown below: */

#define PCL_OK 0 /* function successfully completed */
#define PCL_NO_NEWD 1 /* no new data received */

#define PCL_ERR_INV_PORT -1 /* invalid port id */
#define PCL_ERR_NOT_FOUND -2 /* pcl hardware not found */
#define PCL_ERR_NO_HANDLE -3 /* no free handle found */
#define PCL_ERR_INV_HDL -4 /* invalid handle */
#define PCL_ERR_NOTOPEN -5 /* handle not opened by pclOpen() */
#define PCL_ERR_NULL -6 /* null pointer assignment */
#define PCL_ERR_TIMEOUT -7 /* timeout while waiting for pcl ready */
#define PCL_ERR_INV_REG -8 /* invalid register number */
#define PCL_ERR_INV_BANK -9 /* invalid transmit bank number */
#define PCL_ERR_RANGE -10 /* parameter out of range */
#define PCL_ERR_INIT -11 /* can't initialize library */
#define PCL_ERR_IOCTL -12 /* error in device driver while ioctl */
#define PCL_ERR_VER -13 /* version of dll and sys different */

/*****
 *
 * Handles
 *
 *-----
 *
 * To access the PCL card, you need to open a communication port with
 * pclOpen() described later. It is possible to handle several PCL cards
 * simultaneous, you need to open a communication port for each PCL card.
 * You find the number of ports available in PCL_HANDLES: */

#define PCL_HANDLES 10 /* number of handles available */

/*****
 *
 * Access modes
 *
 *-----
 *
 * There are three different modes to access a timecode reader register.
 * This modes are named PCL_REQ_XXX: */

/* PCL_REQ_XXX timecode request mode */

```

```

#define PCL_REQ_ASYNC 0x80 /* asynchron */
#define PCL_REQ_SYNC 0x40 /* synchron */
#define PCL_REQ_PERM 0x20 /* permanent */

/*****
 *
 * BCD/Hex conversion
 *
 *-----
 *
 * Several parameters of the PCL card are coded BCD. To convert binary
 * numbers to BCD and vice versa, you might find the following macros
 * useful: */

#define BCD2HEX(bcd) (((bcd) & 0x0F) + (((bcd) & 0xF0) >> 4) * 10)
/* convert unsigned char from bcd to hex */

#define HEX2BCD(hex) (((hex) / 10) << 4) + ((hex) % 10)
/* convert unsigned char from hex to bcd */

/* Some Macros from windows.h
 *----- */

#define MAKEWORD(a, b) ((WORD)(((BYTE)(a) | ((WORD)((BYTE)(b)) << 8)
#define MAKELONG(a, b) ((LONG)(((WORD)(a) | ((DWORD)((WORD)(b)) <<
16)
#define LOWORD(l) ((WORD)(l))
#define HIWORD(l) ((WORD)(((DWORD)(l) >> 16) & 0xFFFF))
#define LOBYTE(w) ((BYTE)(w))
#define HIBYTE(w) ((BYTE)(((WORD)(w) >> 8) & 0xFF))

/*****
 *
 * Typedefs
 *
 *----- */

/* Some Typedefs from windows.h */

#if !defined(WINAPI) || defined(_NTDDK_)

#define _BYTE_DEFINED
#define _BYTE_DEFINED typedef unsigned char BYTE; /* size is 1 */
#define !_BYTE_DEFINED

#define _WORD_DEFINED
#define _WORD_DEFINED typedef unsigned short WORD; /* size is 2 */
#define !_WORD_DEFINED

#define _DWORD_DEFINED
#define _DWORD_DEFINED typedef unsigned long DWORD; /* size is 4 */
#define !_DWORD_DEFINED

typedef DWORD PCL_TC; /* timecode, bcd */

/* Timecode is coded BCD in a double word.
0xhhmmssff with hh = hours, mm = minutes, ss = seconds, ff = frames
e.g. 0x12345621 means 12:34:56:21 */

typedef DWORD PCL_USER; /* user-bits, bcd */

/* User-Bits are coded BCD similar to timecode described above. */

typedef DWORD PCL_OPRT; /* output port bits (pcl3) */

/* Output port bits are coded in a double word. 01 corresponds to bit 0,
024 to bit 23. Bit 24 to 31 are ignored and should be set to zero.
e.g. 0x00810001 has 01, 017 and 023 set and all other outputs reset. */

typedef WORD PCL_IPRT; /* input port bits (pcl3) */

/* Input port bits are coded in a single word. I1 corresponds to bit 0,
I16 to bit 15. e.g. 0x8101 has I1, I9 and I16 set and all other inputs
reset. */

typedef DWORD PCL_SHOU; /* shift out data (pcl3) */

/* Shift out data is coded in a double word. It corresponds to data $0 to $3
of register $E of PCL card */

typedef DWORD PCL_SHIN; /* shift in data (pcl3) */

/* Shift in data is coded in a double word. It corresponds to data $4 to $7
of register $E of PCL card */

typedef BYTE PCL_TDAT[7]; /* transmit data */

/* Transmit data corresponds to command $6C of PCL card */

typedef BYTE PCL_TSTR[8]; /* transmit string */

/* Transmit string corresponds to commands $60 up to $67 of PCL card */

typedef BYTE PCL_CDAT[8]; /* command data */

/* Command data corresponds to data $0 to $7 of register $F PCL card */

typedef BYTE PCL_SIND[12]; /* serial input data */

/* Serial input data corresponds to data $0 to $B of register $D of PCL */

```



```

/* If you need additional informations from any timecode register of
the pcl card, you can use pclGetRegister().

pclGetRegister(hdl, PCL_REG_MIXED, &time, NULL, NULL, NULL, NULL);
is equivalent to pclGetTc(hdl, &time). */

/*-----*/
#define IOCTL_PCL_GET_REG_SYNC_PCL_CTL_CODE(48)

int _PCL_FDEF pclGetRegSync( /* @48, read register synchronous */
int hdl, /* in: handle from pclOpen() */
int reg, /* in: registers number like PCL_REG_xxx */
PCL_TC_PCL_OUT *time, /* out: timecode from register, bcd */
int _PCL_OUT *bits, /* out: misc. bits from byte $3 of register,
meaning depends of 'newd',
like PCL_BITS_xxx_xxx */
PCL_TC_PCL_OUT *user, /* out: user-bits from register, bcd */
int _PCL_OUT *frame, /* out: framerate as detected by pcl,
like PCL_FRAME_xxx */
int _PCL_OUT *newd /* out: identifier of Readed timecode,
like PCL_NEWD_xxx */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */

/* pclGetRegSync is the most efficient way to get any of the timecode
registers of the pcl card, because it doesn't wait for the pcl to
respond. pclGetRegSync first checks if there is a new timecode in the
specified register. If no, it returns PCL_NO_NEWD. If there is
a new timecode, it reads the value, sends a new request to
the pcl and returns PCL_OK. pclGetRegSync never waits for the pcl card!

If there is a call to pclGetTc() similar to this...

while (running) {
PCL_TC time;
pclGetTc(hdl, &time);
doAnything(time)
}

...it might be replaced with something like that:

pclGetRegSync(hdl, PCL_REG_MIXED, NULL, NULL, NULL, NULL)
while (running) {
PCL_TC time;
if (
pclGetRegSync(hdl, PCL_REG_MIXED, &time, NULL, NULL, NULL, NULL)
== PCL_OK
) {
doAnything(time);
} else {
doAnythingElse();
}
}

Please note, that pclGetRegSync() only returns PCL_OK if there is a new
timecode readed in the specified register. This is opposite to
pclGetTc() which always returns a timecode. In the example above,
with pclGetTc() doAnything() is called in every loop, but with
pclGetRegSync() it is only called if there is a new timecode readed
by the pcl card. The first call to pclGetRegSync() outside the loop
guarantees that doAnything() isn't called with a timecode that was
readed before the program starts. */

/*-----*/
#define IOCTL_PCL_GET_EVENT_PCL_CTL_CODE(51)

int _PCL_FDEF pclGetEvent( /* @51, read event register */
int hdl, /* in: handle from pclOpen() */
PCL_TC_PCL_OUT *dist1, /* out: event distance 1 bcd */
PCL_TC_PCL_OUT *dist2, /* out: event distance 2 bcd */
PCL_TC_PCL_OUT *dist3, /* out: event distance 3 bcd */
int _PCL_OUT *evtc, /* out: status event time, like PCL_EVTC_xxx */
int _PCL_OUT *evub, /* out: status event user, like PCL_EVUB_xxx */
int _PCL_OUT *eflg, /* out: event found flag, like PCL_EFLG_xxx */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */

/* PCL_EVTC_xxx status event time */
#define PCL_EVTC_FRU 0x01 /* units frames of distance 3 <> 0 */
#define PCL_EVTC_FRT 0x02 /* tens frames of distance 3 <> 0 */
#define PCL_EVTC_SEU 0x04 /* units seconds of distance 3 <> 0 */
#define PCL_EVTC_SET 0x08 /* tens seconds of distance 3 <> 0 */
#define PCL_EVTC_MIN 0x10 /* minutes of distance 3 <> 0 */
#define PCL_EVTC_HOU 0x20 /* hours of distance 3 <> 0 */
#define PCL_EVTC_SIG12 0x40 /* sign mod.12 hours */
#define PCL_EVTC_SIG 0x80 /* sign absolute */

/* PCL_EVUB_xxx status event user */
#define PCL_EVUB_UB1 0x01 /* 1st event user <> mixed reg. user */
#define PCL_EVUB_UB2 0x02 /* 2nd event user <> mixed reg. user */
#define PCL_EVUB_UB3 0x04 /* 3rd event user <> mixed reg. user */
#define PCL_EVUB_UB4 0x08 /* 4th event user <> mixed reg. user */
#define PCL_EVUB_UB5 0x10 /* 5th event user <> mixed reg. user */
#define PCL_EVUB_UB6 0x20 /* 6th event user <> mixed reg. user */
#define PCL_EVUB_UB7 0x40 /* 7th event user <> mixed reg. user */
#define PCL_EVUB_UB8 0x80 /* 8th event user <> mixed reg. user */

/* PCL_EFLG_xxx event found flag */
#define PCL_EFLG_UMASK 0x0F /* event user found */
#define PCL_EFLG_TMASK 0xF0 /* event time found */

/* Please refer to description of register $8 in chapter C1 for details. */
/*-----*/
#define IOCTL_PCL_GET_TRANSFER_PCL_CTL_CODE(52)

int _PCL_FDEF pclGetTransfer( /* @52, read transfer register $A */
int hdl, /* in: handle from pclOpen() */
BYTE_PCL_OUT *tran /* out: recorder status, like PCL_TRAN */
);

/* Note: pclGetTransfer is not supported by PCL firmware version less or
equal V1.3 */

/* Please refer to description of register $A in chapter C1 for details. */

/*-----*/
#define IOCTL_PCL_GET_IO_PCL_CTL_CODE(53)

int _PCL_FDEF pclGetIo( /* @53, read i/o ports (pcl3) */
int hdl, /* in: handle from pclOpen() */
PCL_OPRT_PCL_OUT *oprt, /* out: output ports 1-24, like PCL_OPRT_xxx */
int _PCL_OUT *ana, /* out: analog output */
PCL_IPRT_PCL_OUT *iprt /* out: input ports 1-16, like PCL_IPRT_xxx */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */

/* PCL_OPRT_xxx output port bits */
#define PCL_OPRT_1 0x00000011 /* output port 1 */
#define PCL_OPRT_2 0x00000021 /* output port 2 */
#define PCL_OPRT_3 0x00000041 /* output port 3 */
#define PCL_OPRT_4 0x00000081 /* output port 4 */
#define PCL_OPRT_5 0x00000101 /* output port 5 */
#define PCL_OPRT_6 0x00000201 /* output port 6 */
#define PCL_OPRT_7 0x00000401 /* output port 7 */
#define PCL_OPRT_8 0x00000801 /* output port 8 */
#define PCL_OPRT_9 0x0001001 /* output port 9 */
#define PCL_OPRT_10 0x0002001 /* output port 10 */
#define PCL_OPRT_11 0x0004001 /* output port 11 */
#define PCL_OPRT_12 0x0008001 /* output port 12 */
#define PCL_OPRT_13 0x0010001 /* output port 13 */
#define PCL_OPRT_14 0x0020001 /* output port 14 */
#define PCL_OPRT_15 0x0040001 /* output port 15 */
#define PCL_OPRT_16 0x0080001 /* output port 16 */
#define PCL_OPRT_17 0x0100001 /* output port 17 */
#define PCL_OPRT_18 0x0200001 /* output port 18 */
#define PCL_OPRT_19 0x0400001 /* output port 19 */
#define PCL_OPRT_20 0x0800001 /* output port 20 */
#define PCL_OPRT_21 0x1000001 /* output port 21 */
#define PCL_OPRT_22 0x2000001 /* output port 22 */
#define PCL_OPRT_23 0x4000001 /* output port 23 */
#define PCL_OPRT_24 0x8000001 /* output port 24 */

/* PCL_IPRT_xxx input port bits */
#define PCL_IPRT_1 0x0001 /* input port 1 */
#define PCL_IPRT_2 0x0002 /* input port 2 */
#define PCL_IPRT_3 0x0004 /* input port 3 */
#define PCL_IPRT_4 0x0008 /* input port 4 */
#define PCL_IPRT_5 0x0010 /* input port 5 */
#define PCL_IPRT_6 0x0020 /* input port 6 */
#define PCL_IPRT_7 0x0040 /* input port 7 */
#define PCL_IPRT_8 0x0080 /* input port 8 */
#define PCL_IPRT_9 0x0100 /* input port 9 */
#define PCL_IPRT_10 0x0200 /* input port 10 */
#define PCL_IPRT_11 0x0400 /* input port 11 */
#define PCL_IPRT_12 0x0800 /* input port 12 */
#define PCL_IPRT_13 0x1000 /* input port 13 */
#define PCL_IPRT_14 0x2000 /* input port 14 */
#define PCL_IPRT_15 0x4000 /* input port 15 */

/* Please refer to description of register $C for details. */
/*-----*/
#define IOCTL_PCL_GET_VERSION_PCL_CTL_CODE(59)

int _PCL_FDEF pclGetVersion( /* @59, read pcl firmware version number */
int hdl, /* in: handle from pclOpen() */
int _PCL_OUT *ident, /* out: software identification, != 0
means special design */
int _PCL_OUT *firmware, /* out: pcl firmware revision number:

bit 76543210
-----
00 PCL3
10 reserved
10 PCL10
11 PCL12
xx main version, 0..3
xxxx sub version, 0..F */

int _PCL_OUT *dll /* out: dll revision number:

111111
bit 543210987654321
-----
xxxxxxxx main version
xxxxxxxx sub version */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */
/*-----*/
#define IOCTL_PCL_GET_VER_SPECIAL_PCL_CTL_CODE(66)

int _PCL_FDEF pclGetVerSpecial( /* @66, read special pcl firmware version */
int hdl, /* in: handle from pclOpen() */
int _PCL_OUT *ident1, /* out: identifier from byte $a */
int _PCL_OUT *ident2, /* out: identifier from byte $b */
int _PCL_OUT *ident3 /* out: identifier from byte $c */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */
/*-----*/
#define IOCTL_PCL_SET_MIXED_PCL_CTL_CODE(67)

int _PCL_FDEF pclSetMixed( /* @67, $22, set mixed register */
int hdl, /* in: handle from pclOpen() */
PCL_TC time, /* in: timecode */
PCL_USER user, /* in: userbits */
int status, /* in: status like PCL_BITS_xxx_xxx */
int newd /* in: new data flag like PCL_NEWD_xxx */
);

/*-----*/
#define IOCTL_PCL_GET_SIO_PCL_CTL_CODE(54)

int _PCL_FDEF pclGetSio( /* @54, read sio registers */
int hdl, /* in: handle from pclOpen() */
int _PCL_OUT *ssta, /* out: sio status reg, like PCL_SSTA_xxx */

```



```

/* PCL_VLINE_xxxx vitc line mode */
#define PCL_VLINE_2LINES 0x00 /* 2-lines mode */
#define PCL_VLINE_BLOCK 0x01 /* block mode */

/* Note: pclViteSetup is not supported by PCL firmware version less or
equal V1.D */

/*-----*/
#define IOCTL_PCL_MIXED_ENABLE _PCL_CTL_CODE(11)

int _PCL_FDEF pclMixedEnable( /* @11, $20, mixed register: reader on/off */
    int hdl, /* in: handle from pclOpen() */
    int rmask /* in: reader mask, like PCL_RMASK_xxx */
);

/*-----*/
#define IOCTL_PCL_PRIORITY _PCL_CTL_CODE(12)

int _PCL_FDEF pclPriority( /* @12, $21, priority: mixed reg. tc priority */
    int hdl, /* in: handle from pclOpen() */
    int prior /* in: priority, like PCL_PRIOR_xxx */
);

/* PCL_PRIOR_xxxx timecode priority */
#define PCL_PRIOR_VLC 0x01 /* vitc-ltc-ctl */
#define PCL_PRIOR_LVC 0x02 /* ltc-vitc-ctl */

/*-----*/
#define IOCTL_PCL_TC_FRAMES _PCL_CTL_CODE(13)

int _PCL_FDEF pclTcFrames( /* @13, $30, tc frame rate */
    int hdl, /* in: handle from pclOpen() */
    int tcfr /* in: frame rate, like PCL_TCFR_xxx */
);

/* PCL_TCFR_xxxx tc frame rate */
#define PCL_TCFR_AUTO 0x00 /* automatic */
#define PCL_TCFR_24 0x01 /* 24 frames */
#define PCL_TCFR_25 0x02 /* 25 frames */
#define PCL_TCFR_30 0x03 /* 30 frames non drop mode */
#define PCL_TCFR_30DF 0x04 /* 30 frames drop mode */

/*-----*/
#define IOCTL_PCL_CTL_RATE _PCL_CTL_CODE(14)

int _PCL_FDEF pclCtlRate( /* @14, $31, ctl rate for timecode drop-outs */
    int hdl, /* in: handle from pclOpen() */
    int ctlrate /* in: ctl rate */
);

/*-----*/
#define IOCTL_PCL_CTL_COUNT _PCL_CTL_CODE(15)

int _PCL_FDEF pclCtlCount( /* @15, $32, ctl rate for independent counter */
    int hdl, /* in: handle from pclOpen() */
    int fps /* in: frames per second, bcd */
);

/*-----*/
#define IOCTL_PCL_CTL_MODE _PCL_CTL_CODE(16)

int _PCL_FDEF pclCtlMode( /* @16, $40, ctl mode */
    int hdl, /* in: handle from pclOpen() */
    int cmode /* in: ctl mode, like PCL_CMODE_xxx */
);

/* PCL_CMODE_xxxx ctl mode */
#define PCL_CMODE_UNSIGNED 0x00 /* ctl without sign */
#define PCL_CMODE_SIGNED 0x01 /* ctl with sign */

/*-----*/
#define IOCTL_PCL_CTL_RESET _PCL_CTL_CODE(17)

int _PCL_FDEF pclCtlReset( /* @17, $41, ctl reset */
    int hdl /* in: handle from pclOpen() */
);

/*-----*/
#define IOCTL_PCL_CTL_PRESET _PCL_CTL_CODE(18)

int _PCL_FDEF pclCtlPreset( /* @18, $42, ctl preset */
    int hdl, /* in: handle from pclOpen() */
    PCL_TC time /* in: ctl preset value, bcd */
);

/*-----*/
#define IOCTL_PCL_CTL_DIR _PCL_CTL_CODE(19)

int _PCL_FDEF pclCtlDir( /* @19, $44, ctl direction */
    int hdl, /* in: handle from pclOpen() */
    int cdir /* in: ctl direction, like PCL_CDIRE_xxx */
);

/* PCL_CDIRE_xxxx ctl direction */
#define PCL_CDIRE_REV 0x00 /* h = reverse, l = forward */
#define PCL_CDIRE_FWD 0x01 /* l = reverse, h = forward */

/*-----*/
#define IOCTL_PCL_EVENT_TIME _PCL_CTL_CODE(20)

int _PCL_FDEF pclEventTime( /* @20, $50, set event time */
    int hdl, /* in: handle from pclOpen() */
    PCL_TC time /* in: event time, bcd */
);

/*-----*/
#define IOCTL_PCL_EVENT_USER _PCL_CTL_CODE(21)

int _PCL_FDEF pclEventUser( /* @21, $51, set event user */
    int hdl, /* in: handle from pclOpen() */
    PCL_USER user /* in: event time, bcd */
);

/*-----*/
#define IOCTL_PCL_TRAN_BANK _PCL_CTL_CODE(22)

int _PCL_FDEF pclTranBank( /* @22, $60-$67, set transmit bank 1-8 */
    int hdl, /* in: handle from pclOpen() */
    int bank, /* in: transmit bank, 1-8 */
    const BYTE _PCL_IN *tstr /* in: transmit data, like PCL_TSTR */
);

/*-----*/
#define IOCTL_PCL_SIO_CMD _PCL_CTL_CODE(23)

int _PCL_FDEF pclSioCmd( /* @23, $68, set sio command reg */
    int hdl, /* in: handle from pclOpen() */
    int scmd /* in: sio command reg, like PCL_SCMD_xxx */
);

/* PCL_SCMD_xxxx sio command reg */
/* bit 0: receiver disable/enable */
#define PCL_SCMD_RDIS 0x00 /* receiver disable */
#define PCL_SCMD_RENA 0x01 /* receiver enable */
/* bit 7-5: parity mode */
#define PCL_SCMD_NONE 0x00 /* no parity trans/rec */
#define PCL_SCMD_ODD 0x01 /* odd parity trans/rec */
#define PCL_SCMD_EVEN 0x02 /* even parity trans/rec */
#define PCL_SCMD_MARK 0x03 /* mark parity trans/rec */
#define PCL_SCMD_SPACE 0x04 /* space parity trans/rec */

/*-----*/
#define IOCTL_PCL_SIO_CTL _PCL_CTL_CODE(24)

int _PCL_FDEF pclSioCtl( /* @24, $69, set sio control reg */
    int hdl, /* in: handle from pclOpen() */
    int sctl /* in: sio control reg, like PCL_SCTL_xxx */
);

/* PCL_SCTL_xxxx sio control reg */
/* bit 3-0: baudrate */
#define PCL_SCTL_38400 0x00 /* 38400 bps */
#define PCL_SCTL_50 0x01 /* 50 bps */
#define PCL_SCTL_75 0x02 /* 75 bps */
#define PCL_SCTL_110 0x03 /* 109.92 bps */
#define PCL_SCTL_135 0x04 /* 134.58 bps */
#define PCL_SCTL_150 0x05 /* 150 bps */
#define PCL_SCTL_300 0x06 /* 300 bps */
#define PCL_SCTL_600 0x07 /* 600 bps */
#define PCL_SCTL_1200 0x08 /* 1200 bps */
#define PCL_SCTL_1800 0x09 /* 1800 bps */
#define PCL_SCTL_2400 0x0A /* 2400 bps */
#define PCL_SCTL_3600 0x0B /* 3600 bps */
#define PCL_SCTL_4800 0x0C /* 4800 bps */
#define PCL_SCTL_7200 0x0D /* 7200 bps */
#define PCL_SCTL_9600 0x0E /* 9600 bps */
#define PCL_SCTL_19200 0x0F /* 19200 bps */
/* bit 4: rs232/422 select */
#define PCL_SCTL_RS232 0x00 /* rs-232 */
#define PCL_SCTL_RS422 0x01 /* rs-422 */
/* bit 6-5: word length */
#define PCL_SCTL_B8 0x00 /* 8 bits/word */
#define PCL_SCTL_B7 0x01 /* 7 bits/word */
#define PCL_SCTL_B6 0x02 /* 6 bits/word */
#define PCL_SCTL_B5 0x03 /* 5 bits/word */
/* bit 7: stop bits */
#define PCL_SCTL_S1 0x00 /* 1 stop bit */
#define PCL_SCTL_S2 0x01 /* 2 stop bits */

/*-----*/
#define IOCTL_PCL_REC_MODE _PCL_CTL_CODE(25)

int _PCL_FDEF pclRecMode( /* @25, $6a, set receive mode */
    int hdl, /* in: handle from pclOpen() */
    int rmode /* in: receive mode, like PCL_RMODE_xxx */
);

/* PCL_RMODE_xxxx receive mode */
#define PCL_RMODE_NORMAL 0x00 /* normal mode, byte by byte */
#define PCL_RMODE_SONY 0x01 /* sony protocol, controlling device */
#define PCL_RMODE_PANA 0x02 /* panasonic prot., controlling device */

/*-----*/
#define IOCTL_PCL_TRAN_STR _PCL_CTL_CODE(26)

int _PCL_FDEF pclTranStr( /* @26, $6b, transmit string */
    int hdl, /* in: handle from pclOpen() */
    int num /* in: number of bytes to transmit */
);

/*-----*/
#define IOCTL_PCL_TRAN_DATA _PCL_CTL_CODE(27)

int _PCL_FDEF pclTranData( /* @27, $6c, transmit data */
    int hdl, /* in: handle from pclOpen() */
    int num, /* in: number of bytes to transmit */
    const BYTE _PCL_IN *tdata /* in: transmit data, like PCL_TDAT */
);

/*-----*/
#define IOCTL_PCL_MODE_OUT _PCL_CTL_CODE(28)

int _PCL_FDEF pclModeOut( /* @28, $70, mode output (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int omode, /* in: output mode, like PCL_OMODE_xxx */
    WORD freq /* in: port pb7 frequency */
);

/* PCL_OMODE_xxxx mode output */
#define PCL_OMODE_NORMAL 0x00 /* normal */
#define PCL_OMODE_FREQ 0x01 /* port pb7 frequency */

```

Operating Instructions PCL-5

Page 28

```
#define IOCTL_PCL_OUT_PORT_PCL_CTL_CODE(29)
int_PCL_FDEF pclOutPort( /* @29, $71, output ports (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    PCL_OPRT oprt /* in: ports 1-24, like PCL_OPRT_xxx */
);
/*-----*/

#define IOCTL_PCL_PORT_HIGH_PCL_CTL_CODE(30)
int_PCL_FDEF pclPortHigh( /* @30, $72, set port to h (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int oprt /* in: number -1 of port ($00-$17) */
);
/*-----*/

#define IOCTL_PCL_PORT_LOW_PCL_CTL_CODE(31)
int_PCL_FDEF pclPortLow( /* @31, $73, set port to l (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int oprt /* in: number -1 of port ($00-$17) */
);
/*-----*/

#define IOCTL_PCL_POS_PULSE_PCL_CTL_CODE(32)
int_PCL_FDEF pclPosPulse( /* @32, $74, pos. pulses (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    PCL_OPRT oprt, /* in: ports 1-24, like PCL_OPRT_xxx */
    int pmode, /* in: pulse width mode, like PCL_PMODE_xxx */
    int pwidth /* in: pulse width */
);
/*
    PCL_PMODE_xxx pulse width mode */
#define PCL_PMODE_INDIV 0x00 /* individual pulse width */
#define PCL_PMODE_SAME 0x01 /* same pulse width for all ports conc. */
/*-----*/

#define IOCTL_PCL_NEG_PULSE_PCL_CTL_CODE(33)
int_PCL_FDEF pclNegPulse( /* @33, $75, neg. pulses (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    PCL_OPRT oprt, /* in: ports 1-24, like PCL_OPRT_xxx */
    int pmode, /* in: pulse width mode, like PCL_PMODE_xxx */
    int pwidth /* in: pulse width */
);
/*-----*/

#define IOCTL_PCL_POS_PULSE_OUT_PCL_CTL_CODE(34)
int_PCL_FDEF pclPosPulseOut( /* @34, $76, pos. pulse out (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int oprt, /* in: number -1 of port ($00-$17) */
    int pwidth /* in: pulse width */
);
/*-----*/

#define IOCTL_PCL_NEG_PULSE_OUT_PCL_CTL_CODE(35)
int_PCL_FDEF pclNegPulseOut( /* @35, $77, neg. pulse out (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int oprt, /* in: number -1 of port ($00-$17) */
    int pwidth /* in: pulse width */
);
/*-----*/

#define IOCTL_PCL_PULSE_WIDTH_PCL_CTL_CODE(36)
int_PCL_FDEF pclPulseWidth( /* @36, $78, pulse width: preset pulse width
    (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    PCL_OPRT oprt, /* in: ports 1-24, like PCL_OPRT_xxx */
    int pwidth /* in: pulse width */
);
/*-----*/

#define IOCTL_PCL_ANALOG_OUT_PCL_CTL_CODE(37)
int_PCL_FDEF pclAnalogOut( /* @37, $79, analog out (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int value /* in: (-10 + value * 20/256) volt */
);
/*-----*/

#define IOCTL_PCL_SHIFT_DATA_PCL_CTL_CODE(38)
int_PCL_FDEF pclShiftData( /* @38, $80, shift out data (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    PCL_SHOU shou /* in: shift out data */
);
/*-----*/

#define IOCTL_PCL_SHIFT_WIDTH_PCL_CTL_CODE(39)
int_PCL_FDEF pclShiftWidth( /* @39, $81, shift out width (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int swid /* in: shift out width, like PCL_SWID_xxx */
);
/*
    PCL_SWID_xxx shift out width */
#define PCL_SWID_8 0x00 /* 8 bit */
#define PCL_SWID_16 0x01 /* 16 bit */
#define PCL_SWID_24 0x02 /* 24 bit */
#define PCL_SWID_32 0x03 /* 32 bit */
/*-----*/

#define IOCTL_PCL_SHIFT_MODE_PCL_CTL_CODE(40)
int_PCL_FDEF pclShiftMode( /* @40, $82, shift out mode (pcl3) */
    int hdl, /* in: handle from pclOpen() */
    int smode /* in: shift out mode, like PCL_SMODE_xxx */
);
/*
    PCL_SMODE_xxx shift out mode */
#define PCL_SMODE_DIS 0x80 /* disable */
#define PCL_SMODE_ONE 0x00 /* one shot */
#define PCL_SMODE_CONT 0x01 /* continuous */
/*-----*/

#define IOCTL_PCL_REC_CTL_PCL_CTL_CODE(41)
int_PCL_FDEF pclRecCtl( /* @41, $90, recorder control */
    int hdl, /* in: handle from pclOpen() */
    int rctl /* in: command, like PCL_RCTL_xxx */
);
/*
    PCL_RCTL_xxx recorder control command */
#define PCL_RCTL_STOP 0x01 /* stop */
#define PCL_RCTL_PLAY 0x02 /* play */
#define PCL_RCTL_REC 0x03 /* rec */
#define PCL_RCTL_STILL 0x04 /* still/pause */
#define PCL_RCTL_FFWD 0x05 /* fast forward */
#define PCL_RCTL_ADV 0x06 /* frame advance */
#define PCL_RCTL_PREV 0x07 /* fast reverse */
#define PCL_RCTL_BACK 0x08 /* frame reverse (back) */
#define PCL_RCTL_TRIMF 0x09 /* fine trim forward (1.4+) */
#define PCL_RCTL_TRIMR 0x0A /* fine trim reverse (1.4+) */
#define PCL_RCTL_EJECT 0x0B /* eject */
#define PCL_RCTL_EDON 0x0C /* edit on */
#define PCL_RCTL_EDOFF 0x0D /* edit off */
/*-----*/

#define IOCTL_PCL_REC_CTL_DATA_PCL_CTL_CODE(42)
int_PCL_FDEF pclRecCtlData( /* @42, $90, recorder control, with data */
    int hdl, /* in: handle from pclOpen() */
    int rctl, /* in: command, like PCL_RCTL_xxx */
    int data /* in: data, depends on command */
);
/*
    PCL_RCTD_xxx recorder control command, variable */
#define PCL_RCTD_VFWD 0x12 /* var forward */
#define PCL_RCTD_VREV 0x13 /* var reverse */
#define PCL_RCTD_JFWD 0x14 /* jog forward */
#define PCL_RCTD_JREV 0x15 /* jog reverse */
#define PCL_RCTD_SFWD 0x1F /* shtl forward */
#define PCL_RCTD_SREV 0x20 /* shtl reverse */
#define PCL_RCTD_EDPR 0x26 /* edit preset */
/*
    PCL_EDPR_xxx edit preset data */
#define PCL_EDPR_AUDIO1 0x01 /* audio 1 */
#define PCL_EDPR_AUDIO2 0x02 /* audio 2 */
#define PCL_EDPR_AUDIO3 0x04 /* audio 3 */
#define PCL_EDPR_TIMECODE 0x04 /* timecode */
#define PCL_EDPR_AUDIO4 0x08 /* audio 4 */
#define PCL_EDPR_VIDEO 0x10 /* video */
#define PCL_EDPR_ASSEMBLE 0x20 /* assemble */
#define PCL_EDPR_INSERT 0x40 /* insert */
/*-----*/

#define IOCTL_PCL_REC_CTL_TC_PCL_CTL_CODE(43)
int_PCL_FDEF pclRecCtlTc( /* @43, $90, recorder control, with timecode */
    int hdl, /* in: handle from pclOpen() */
    int rctl, /* in: command, like PCL_RCTL_xxx */
    PCL_TC time /* in: timecode, depends on command */
);
/*
    PCL_RCTC_xxx recorder control command, with tc */
#define PCL_RCTC_CUEUP 0x16 /* cue up with data (goto) */
#define PCL_RCTC_GOTO 0x16 /* goto (cue up with data) */
/*-----*/

#define IOCTL_PCL_REC_STAT_PCL_CTL_CODE(44)
int_PCL_FDEF pclRecStat( /* @44, $91, recorder status request */
    int hdl, /* in: handle from pclOpen() */
    BYTE_PCL_OUT *stat, /* out: recorder status, like PCL_STAT */
    int_PCL_OUT *ident /* out: recorder identification flag */
);
/* Note: Set several 'out' parameters to NULL if you don't need them */
/*****
 * Internal functions
 *
 * The following functions are internal to AVPCL. Not data is transferred
 * to or from PCL card.
 *****/
/*-----*/

#define IOCTL_PCL_TC_2_BCD_PCL_CTL_CODE(50)
PCL_TC_PCL_FDEF pclTc2Bcd( /* @50, convert binary timecode to packed bcd */
    int hours, /* in: hours of binary timecode */
    int minutes, /* in: minutes of binary timecode */
    int seconds, /* in: seconds of binary timecode */
    int frames /* in: frames of binary timecode */
);
/*-----*/

#define IOCTL_PCL_BCD_2_TC_PCL_CTL_CODE(49)
int_PCL_FDEF pclBcd2Tc( /* @49, convert packed bcd timecode to binary */
    PCL_TC time, /* in: packed bcd timecode */
    int_PCL_OUT *hours, /* out: hours of binary timecode */
    int_PCL_OUT *minutes, /* out: minutes of binary timecode */
    int_PCL_OUT *seconds, /* out: seconds of binary timecode */
    int_PCL_OUT *frames /* out: frames of binary timecode */
);
/* Note: Set several 'out' parameters to NULL if you don't need them */
/*-----*/
```

```

#define IOCTL_PCL_TC_2_FRAMES _PCL_CTL_CODE(60)

long _PCL_FDEF pclTc2Frames( /* @60, convert packed bcd timecode to long */
    PCL_TC time, /* in: packed bcd timecode */
    int frate /* in: frame rate (25 or 30) */
);
/*-----*/
DWORD _PCL_FDEF pclIoctlError( /* @61, get last ioctl error */
    void /* no parameters needed */
); /* result: last ioctl error if any pclXXX
    function returned PCL_ERR_IOCTL */

/* Note: This function is only useful with the windows nt device driver
    avpclnt.sys. With avpcl.dll or avpcl32.dll (win95 version) this
    function always returns 0 (no error). */
/*-----*/

#define IOCTL_PCL_GET_BASE_ADDRESS _PCL_CTL_CODE(62)

unsigned _PCL_FDEF pclGetBaseAddress( /* @62, get pcl base address */
    int index /* in: index to IoPortAddress in registry */
);

/* pclGetBaseAddress() reads the base address of the pcl card from the
    Windows NT device driver. It is only useful under Windows NT. Under
    all other platforms, pclGetBaseAddress() returns 0.

    index = 0 returns the DWORD registry entry 'HKEY_LOCAL_MACHINE\SYSTEM\
    CurrentControlSet\Services\AvPclNT\Parameters\IoPortAddress',
    index = 1 returns the DWORD registry entry 'HKEY_LOCAL_MACHINE\SYSTEM\
    CurrentControlSet\Services\AvPclNT\Parameters\IoPortAddress1',
    index = 2 returns the DWORD registry entry 'HKEY_LOCAL_MACHINE\SYSTEM\
    CurrentControlSet\Services\AvPclNT\Parameters\IoPortAddress2'
    and so on.

    If the registry entry doesn't exist, pclGetBaseAddress() returns 0. */
/*-----*/
/*-----*/
*
* Program options
*
*-----*/
/* This options use special features of PCL card. To use any of these,
    a special PCL firmware EPROM is needed. */
/*-----*/

#ifdef PCL_OPT_GETCMD /* get command bytes from device */

#define IOCTL_PCL_OPT_GETCMD _PCL_CTL_CODE(100)

int _PCL_FDEF pclOptGetCmd( /* @100, get command bytes from device */
    int hdl, /* in: handle from pclOpen() */
    int _PCL_OUT *rcntl, /* out: command, like PCL_RCTx_XXX */
    BYTE _PCL_OUT *full /* out: sony command bytes, like PCL_FULL */
);

/* PCL_RCTF_XXX recorder control status */
#define PCL_RCTF_STATS 0xEF /* status sense */
#define PCL_RCTF_NONE 0xFE /* no new cmd */
#define PCL_RCTF_UNKNOWN 0xFF /* unknown cmd */

#endif // PCL_OPT_GETCMD
/*-----*/
#ifdef PCL_OPT_MTD /* read mtd times */

#define IOCTL_PCL_MTD_GET_STR _PCL_CTL_CODE(101)

int _PCL_FDEF pclMtdGetStr( /* @101, get mtd time as string */
    int hdl, /* in: handle from pclOpen() */
    int mtd, /* in: mtd register like PCL_MTDREG_XXX */
    char _PCL_OUT *str, /* out: ascii string, max. 10 chars */
    BYTE _PCL_OUT *state /* out: state of time like PCL_MTDSTAT_XXX */
);

/* Note: Set several 'out' parameters to NULL if you don't need them */

#define PCL_MTD_STR 11 /* size of *str parameter */

/* PCL_MTDREG_XXX mtd time */
#define PCL_MTDREG_A 0x0A /* mtd time a (display mode a) */
#define PCL_MTDREG_B 0x0B /* mtd time b (display mode b) */
#define PCL_MTDREG_C 0x0C /* mtd time c (display mode c) */
#define PCL_MTDREG_D 0x0D /* mtd time d (display mode d) */

#define PCL_MTDREG_E 0x0E /* mtd time e (display mode e) */
#define PCL_MTDREG_F 0x0F /* mtd time f (display mode f, maz tc) */
#define PCL_MTDREG_H 0x02 /* mtd time h (display mode 2, time) */
#define PCL_MTDREG_I 0x03 /* mtd time i (display mode 3, date) */
#define PCL_MTDREG_T 0x06 /* mtd time (display mode 6) */
#define PCL_MTDREG_6 0x06 /* mtd time (display mode 6) */

/* PCL_MTDSTAT_XXX mtd status */
#define PCL_MTDSTAT_BLINK 0x01 /* mtd time is blinking */
#define PCL_MTDSTAT_OFF 0x02 /* mtd time is off */
#define PCL_MTDSTAT_ERRLTC 0x04 /* no mtd time available */

#endif // PCL_OPT_MTD
/*-----*/
#ifdef PCL_OPT_REALTIME

typedef struct {
    struct { /* time from pcl5 */
        WORD valid; /* validation flags like PCL_RTVAL_XXX */
        WORD micro; /* microseconds */
        WORD milli; /* milliseconds */
        WORD sec; /* seconds */
        WORD min; /* minutes */
        WORD hour; /* hours (0-23) */
        WORD day; /* day of month (1-31) */
        WORD month; /* month (1-12) */
        WORD year; /* year (1990-2089) */
        WORD tzzone; /* time zone like PCL_RTZONE_XXX */
        WORD _0, _1; /* reserved */
    } ltc;
    struct { /* time from system clock (gmt) */
        WORD milli; /* milliseconds */
        WORD sec; /* seconds */
        WORD min; /* minutes */
        WORD hour; /* hours (0-23) */
        WORD day; /* day of month (1-31) */
        WORD month; /* month (0-11) */
        WORD year; /* year (minus 1900) */
        WORD _0, _1, _2; /* reserved */
    } system;
} PCL_REALTIME;

/* PCL_RTVAL_XXX realtime validation */
#define PCL_RTVAL_INVALID 0x00 /* nothing valid */
#define PCL_RTVAL_TIME 0x01 /* time is valid (sync bit in user) */
#define PCL_RTVAL_DATE 0x02 /* date is valid (plausible and no wrap) */
#define PCL_RTVAL_VALID 0x03 /* everything is valid */

/* PCL_RTZONE_XXX time zone */
#define PCL_RTZONE_UTC 0x00 /* utc (from tz bits in user) */
#define PCL_RTZONE_MEZ 0x01 /* mez (from tz bits in user) */
#define PCL_RTZONE_MESZ 0x02 /* mesz (from tz bits in user) */
/*-----*/

#define IOCTL_PCL_GET_REALTIME _PCL_CTL_CODE(102)

int _PCL_FDEF pclGetRealtime( /* @102, $15, get realtime */
    int hdl, /* in: handle from pclOpen() */
    PCL_REALTIME _PCL_OUT *tm /* out: realtime from pcl */
);

/*
    return values:

    PCL_NO_NEWD: PCL didn't read a ltc within the last 40ms.
    PCL_ERR_TIMEOUT: PCL didn't respond to command.
*/
/*-----*/

int _PCL_FDEF pclGetRealTest(
    int hdl,
    DWORD _PCL_OUT *cnt,
    DWORD _PCL_OUT *cnt1,
    WORD _PCL_OUT *cntR
);

#endif // PCL_OPT_REALTIME
/*-----*/
#ifdef __cplusplus
}
#endif
#endif //***** _AV_AVPC_L_H */

```