

**Module Driver  
for  
COMBI-Modul 167  
Version 1.0**

**Software Manual**

**Edition April 2001**

In this manual are descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, PHYTEC Meßtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Meßtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Meßtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.

Additionally, PHYTEC Meßtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Meßtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2001 PHYTEC Meßtechnik GmbH, D-55129 Mainz.

Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may occur without the express written consent from PHYTEC Meßtechnik GmbH.

	EUROPE	NORTH AMERICA
Address:	PHYTEC Technologie Holding AG Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 255 Ericksen Avenue NE Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (800) 0749832 <a href="mailto:order@phytec.de">order@phytec.de</a>	1 (800) 278-9913 <a href="mailto:info@phytec.com">info@phytec.com</a>
Technical Support:	+49 (6131) 9221-31 <a href="mailto:support@phytec.de">support@phytec.de</a>	1 (800) 278-9913 <a href="mailto:support@phytec.com">support@phytec.com</a>
Fax:	+49 (6131) 9221-33	1 (206) 780-9135
Web Site:	<a href="http://www.phytec.de">http://www.phytec.de</a>	<a href="http://www.phytec.com">http://www.phytec.com</a>

1<sup>st</sup> Edition April 2001

---

<b>1</b>	<b>Driver Library for the COMBI-Modul 167 .....</b>	<b>1</b>
<b>2</b>	<b>Use of the Driver Library .....</b>	<b>3</b>
2.1	Driver Library Function Contents.....	3
2.2	Constants for Port and Channel Numbers .....	3
2.3	Selection of Primary or Alternative Function.....	5
2.4	Direct Access to the C167 Components .....	5
2.5	Support of Memory Models.....	6
<b>3</b>	<b>Inputs and Outputs on the COMBI-Modul 167 .....</b>	<b>7</b>
<b>4</b>	<b>Driver Functions for the COMBI-Modul 167 .....</b>	<b>9</b>
4.1	Initializing Function.....	11
4.2	Functions to Access Digital Inputs .....	12
4.3	Functions to Access Digital Outputs .....	15
4.4	Function for Analog Inputs and Outputs .....	18
4.5	Counter Functions.....	23
4.6	Functions for PWM Outputs:.....	27
4.7	Functions for Access to the Display Units .....	31
4.8	Functions for Releasing Interrupts .....	35
<b>5</b>	<b>Timer Functions.....</b>	<b>37</b>
<b>6</b>	<b>Using Interrupts.....</b>	<b>41</b>
<b>7</b>	<b>Error Codes.....</b>	<b>43</b>
<b>8</b>	<b>PCM_ENABLE_WARNING Switch.....</b>	<b>44</b>
<b>9</b>	<b>Software Structure .....</b>	<b>45</b>
<b>10</b>	<b>Hints on Using the Demo Program .....</b>	<b>47</b>
<b>11</b>	<b>Appendix A .....</b>	<b>51</b>
	<b>Index .....</b>	<b>53</b>

## Index of Figures

Figure 1: External Connections.....	47
-------------------------------------	----



## **1 Driver Library for the COMBI-Modul 167**

The COMBI-Modul 167 offers a large amount of digital and analog input/output channels and counters to the user. With these components processing of different types of industry-standard signals can be done easily. Most of the module's functionality is achieved by using the integrated features of the Infineon's C167 microcontroller. Programming the COMBI-Modul 167 assumes exact knowledge of the controller's internal structure and the peripheral input/output units as well.

The driver library places functions to the customers disposal, that allow comfortable and easy access to all the units on-board of the COMBI-Modul 167. They make rapid realization of complex projects possible, without needing detailed knowledge of programming the on-chip components of the C167 controller. The symbolic terms printed on the modules connector row's are used for functions that access the various input and output channels. Detailed knowledge of internal dependencies between output connectors and port pin on the microcontroller is not necessary. The driver functions also pay attention to partly done negations of signal levels caused by the peripheral input/output units of the COMBI-Modul 167. Furthermore proper initialization of the C167's components for channels with alternative functions is guaranteed.

The driver library for the COMBI-Modul 167 is completed by one timer function, that offers a system time with a resolution of 1 ms. These timer functions respectively the initializing function is included in a separate library which can be linked to the application software.



## 2 Use of the Driver Library

### 2.1 Driver Library Function Contents

The driver library PCMDRV67.LIB places functions of different categories at user's disposal as listed below:

- Read/write a single digital input/output
- Read/write a group of digital inputs/outputs
- Read/write an analog input/output
- Read/write a counter channel
- Write to a PWM output
- Enable interrupts for digital inputs
- Set the resolution of an analog output channel
- Set various counter modes
- Set the status displays and reads the state of the on-board switches

The library PCMTMR67.LIB extends the driver functions for a system timer with a resolution of 1 ms. This offers a defined time base for measurement and control tasks. A more detailed description of these functions can be found in *sections 4 and 5*.

### 2.2 Constants for Port and Channel Numbers

The files PCMDRV67.INC and PCMDRV67.H contain symbolic constants for port and channel numbers to enable access to various inputs/outputs on the COMBI-Modul 167. These constants refer to terms to be found on the connector rows. The access to all inputs/outputs can be done using this symbolic constants without special knowledge of the internal links between port pins on the C167 controller and the dedicated input/output. For inputs/outputs with alternative functions there are additional constants defined.

Altogether the following symbols can be used for programming purposes (*refer to section 3*):

Primary functions:

IN0	... IN23	digital inputs
OUT0	... OUT17	digital transistor and relay output
AIN0	... AIN3	analog inputs
AOUT0	... AOUT1	analog outputs

The symbolic constants for the primary functions correspond directly to the labels on the connector rows.

Alternative functions:

CIN20	... CIN23	Counter channels as alternative function for digital inputs IN20..IN23
PWMOUT16	... PWMOUT17	PWM-channels as alternative function for transistor outputs OUT16 and OUT17

The symbolic constants for the alternative functions refer to the name of the dedicated input/output ports that is used for the alternative function. The terms are according to the labels on the connector rows. This means, the term CIN20 is used for a counter that counts pulse signals coming over the input IN20. The examples listed below show the use of these constants for programming purposes:

```
Port8 = PCMGetInPort (IN8);           // read request to port8
PCMSetOutPort (OUT0, 1);              // output a high level
PCMSetOutPort (OUT0, 0);              // to port0
PCMSetCounter (CIN20, 0);             // process
Cnt = PCMGetCounter (CIN20);          // counter for port20
```

All further constants definitions (ON/OFF, RUN/STOP, HIGHRES/LOWRES, ...) are described at the dedicated functions these constants can be used.

## 2.3 Selection of Primary or Alternative Function

A selection of primary or alternative function is done only for outputs OUT16 and OUT17. The driver function sets implicit the mode necessary for this operation. While calling the functions **PCMSetPWMChannel** and **PCMSetPWMRegister** the corresponding PWM channel will be completely initialized. Afterwards calling the function **PCMSetOutPort** deactivates the PWM channel and brings a static level to this output. It is not necessary to explicit disable the PWM channel.

The counter functions for inputs IN20 to IN23 are automatically activated by the initializing function. So every rising edge of signals connected to inputs IN20 to IN23 leads to incrementing of the dedicated counter. The function **PCMGetInPort** queries the current state of the input, the function **PCMGetCounter** calculates the current counter level. Both input functions work in parallel, switching between work modes is not caused. The definition of the operation mode as a counter is done at once at time of calling the function **PCMInitialize**. That is why, another use of the timer/counter channels is possible.

## 2.4 Direct Access to the C167 Components

The C167 controller's on-chip components, used by the COMBI-Modul 167, allow partly utilization of certain operating modes. Use of these modes may exceed the extent the driver library's functionality. It is principle possible to use these certain modes with support of own routines by the software programmer. It has to be noted, that the function **PCMInitialize** initializes all required resources used by the driver functions. A summary of all resources affected by this can be found in attachment A. The reprogramming of on-chip components by own software routines should be done always after calling the function **PCMInitialize**.

## **2.5 Support of Memory Models**

The library with the driver functions for the COMBI-Modul 167 and the system timer are independent from memory model used (small, medium, large). The hand over of parameters respectively return values is done always using the processor registers. Exclusive numeric data types are used and no pointers. Because of the explicit prototype declaration as "FAR" the assembler and C-compiler use the CALLS instruction for a function call, independent from the adjusted memory model. The CALLS instruction allows a function call to any segment within the controller's address space.

### 3 Inputs and Outputs on the COMBI-Modul 167

The summary below shows the connection between symbolic constants and inputs/outputs on the COMBI-Modul 167. The term for the primary function and the alternate function if available are indicated each. Furthermore the usable functions for access to the respective group of inputs/outputs are listed. A more detailed description of these functions and their parameters is provided in *sections 4 and 5*.

Primary Function	Alternate Function	Port Pin C167	Description	Available Functions
IN0 IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN9 IN10 IN11 IN12 IN13 IN14 IN15		P2.0 P2.1 P2.2 P2.3 P2.4 P2.5 P2.6 P2.7 P2.8 P2.9 P2.10 P2.11 P2.12 P2.13 P2.14 P2.15	Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable Input 24V, INT-capable	PCMGetInGroup1, PCMGetInPort, PCMSetInputInterrupt
IN16 IN17 IN18 IN19 IN20 IN21 IN22 IN23	CIN20 CIN21 CIN22 CIN23	P5.4 P5.5 P5.6 P5.7 P3.5 P3.6 P5.12 P5.13	Input 24V Input 24V Input 24V Input 24V Input 24V, Timer/Count.4 Input 24V, Timer/Count.3 Input 24V, Timer/Count.6 Input 24V, Timer/Count.5	PCMGetInGroup2, PCMGetInPort  PCMGetCounter, PCMSetCounter, PCMSetCounterMode
OUT0 OUT1 OUT2 OUT3 OUT4 OUT5 OUT6 OUT7		P8.0 P8.1 P8.2 P8.3 P8.4 P8.5 P8.6 P8.7	Relay output Relay output Relay output Relay output Relay output Relay output Relay output Relay output	PCMSetOutGroup1, PCMSetOutPort

Primary Function	Alternate Function	Port Pin C167	Description	Available Functions
OUT8 OUT9 OUT10 OUT11 OUT12 OUT13 OUT14 OUT15		P3.0 P3.1 P3.2 P3.3 P7.4 P7.5 P7.6 P7.7	Transistor output 24V Transistor output 24V Transistor output 24V Transistor output 24V Transistor output 24V Transistor output 24V Transistor output 24V Transistor output 24V	PCMSetOutGroup2, PCMSetOutPort
OUT16 OUT17	PWMOUT16 PWMOUT17	P7.0 P7.1	Transistor output 24V Transistor output 24V	PCMSetOutPort, PCMSetPWMChannel, PCMSetPWMRegister
AIN0 AIN1 AIN2 AIN3		P5.0 P5.1 P5.2 P5.3	Input 0..10V Input 0..10V Input 0..10V Input 0..10V	PCMGetADCChannel
AOUT0 AOUT1		P7.2 P7.3	Output 0..10V Output 0..10V	PCMSetDACChannel, PCMSetDACResolution

Table 1: Summary of Inputs and Outputs on the COMBI-Modul 167

## **4 Driver Functions for the COMBI-Modul 167**

The functions offered by the driver library for the COMBI-Modul 167 (PCMDRV67.LIB) are structured in categories listed below:

### **Initializing functions**

PCMInitialize

### **Functions for query digital inputs:**

PCMGetInGroup1

PCMGetInGroup2

PCMGetInPort

### **Functions for setting digital outputs:**

PCMSetOutGroup1

PCMSetOutGroup2

PCMSetOutPort

### **Functions for query analog inputs:**

PCMGetADCCchannel

### **Functions for setting analog outputs:**

PCMSetDACResolution

PCMSetDACChannel

### **Functions for setting digital outputs:**

PCMSetPWMChannel

PCMSetPWMRegister

**Functions to set and query counter states:**

PCMGetCounter  
PCMSetCounter  
PCMSetCounterMode

**Functions to access control components:**

PCMSetRunLED  
PCMSetSysErrLED  
PCMSetCANErrLED  
PCMGetSwitch  
PCMGetHexNumber  
PCMGetDIPSwitch

**Functions to enable and disable interrupts:**

PCMSetInputInterrupt

## 4.1 Initializing Function

Function: **PCMInitialize**

Syntax: WORD PCMInitialize (void)

Input: ---

Output: WORD (R4) = number of driver version

Usage: Initialize the COMBI-Modul 167 (defines the port pin's data direction, resets the outputs, initialize ADC, PWM and counters, turns off the status LEDs, suspends the interrupts, sets up the interrupt levels).

Comment: This function initializes all resources required from the driver functions (an overview of all affected resources can be found in Attachment A). Any reprogramming of on-chip components by user software routines always should be done after calling this function first.

After calling this initializing function the COMBI-Modul 167 is in the following pre-operational state:

- Digital outputs are inactive (relays sloped down, transistors cut off)
- PWM-outputs inactive
- Interrupt disabled for inputs
- 10-bit resolution for analog outputs predefined
- counter released, mode of operation is upward counter, counts at rising signal edge
- Display-LEDs inactive

## 4.2 Functions to Access Digital Inputs

Function: **PCMGetInGroup1**

Syntax: WORD PCMGetInGroup1 (void)

Input: ---

Output: WORD (R4) = value on inputs IN0..IN15  
(bit0 = IN0, ... , bit15 = IN15)

Usage: Query the input ports IN0 to IN15.

*Refer also to:*

PCMGetInGroup2, PCMGetInPort, PCMSetInputInterrupt

Example:

```
main
{
WORD    Input;

// ...

Input = PCMGetInGroup1 ();

// ...

}
```

Function:     **PCMGetInGroup2**

Syntax:       BYTE PCMGetInGroup2 (void)

Input:         ---

Output:        BYTE (RL4) = value on inputs IN16..IN23  
(bit0 = IN16, ... , bit7 = IN23)

Usage:         Query the input ports IN16 to IN23.

Comment:       The ports IN20..IN23 are inputs for the counters  
CIN20..CIN23 at the same time. Using the counters  
the bit positions 4..7 of the return value indicate the  
current state on the counter inputs.

*Refer also to:*

PCMGetInGroup1,            PCMGetInPort,            PCMGetCounter,  
PCMSetCounter

Example:

```
main
{

WORD     Input;

// ...

Input = PCMGetInGroup2 ();

// ...

}
```

Function:     **PCMGetInPort**

Syntax:       bit PCMGetInPort (BYTE Port)

Input:         Port (R8) = number of the port to be read (IN0..IN23)

Output:        bit (R4.0) = current value of the input port

Usage:         Query the several input ports IN0 to IN23.

Comment:       Invalid port numbers will be ignored without any error message. The constants IN0..IN23 are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMGetInGroup1, PCMGetInGroup2

Example:

```
main
{
bit   Port12;

// ...

Port12 = PCMGetInPort (IN12);

// ...

}
```

### 4.3 Functions to Access Digital Outputs

Function:     **PCMSetOutGroup1**

Syntax:       void PCMSetOutGroup1 (BYTE RelaisValue)

Input:         RelaisValue (R8) = output value to relay group  
(bit0 = OUT0, ... , bit7 = OUT7)

Output:       ---

Usage:         Set the relay outputs OUT0 to OUT7.

*Refer also to:*

PCMSetOutGroup2, PCMSetOutPort

Example:

```
main
{

//switch on relay for OUT0 and OUT1
PCMSetOutGroup1 (0x03);

// ...

}
```

Function:     **PCMSetOutGroup2**

Syntax:       void PCMSetOutGroup2 (BYTE TransistorValue)

Input:         TransistorValue (R8) = output value for transistor  
                  group (bit0 = OUT 8, ... , bit7 = OUT15)

Output:        ---

Usage:         set transistor outputs OUT8 to OUT15.

*Refer also to:*

PCMSetOutGroup1, PCMSetOutPort

Example:

```
main
{

// switch on transistors for
// outputs OUT8 and OUT10
PCMSetOutGroup2 (0x05);

// ...

}
```

**Function:** **PCMSetOutPort**

**Syntax:** void PCMSetOutPort (BYTE Port, bit PortValue)

**Input:** Port (R8) = number of the port (OUT0..OUT17)  
PortValue (R15.0) = output bit value

**Output:** ---

**Usage:** set a single output port OUT0 to OUT17.

**Comment:** This function is the only one that allows to direct set the PWM outputs OUT16 and OUT17. Invalid port numbers will be ignored without any error message. The constants OUT0..OUT17 are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetOutGroup1, PCMSetOutGroup2, PCMSetPWMChannel

**Example:**

```
main
{

//switch on relay output OUT4
PCMSetOutPort (OUT4, 1);

// ...

// turn off transistor output OUT16
PCMSetOutPort (OUT16, 0);

// ...

}
```

#### 4.4 Function for Analog Inputs and Outputs

**Function:** **PCMGetADCChannel**

**Syntax:** int PCMGetADCChannel (BYTE Channel)

**Input:** Channel (R8) = number of the input channel (AIN0..AIN3)

**Output:** int (R4) = conversion result of the ADC (10-bit)

**Usage:** Reading an analog input AIN0 to AIN3.

**Comment:** This function returns the directly conversion value of the ADC (0x000..0x1FF). The corresponding voltage [V] can be calculated by multiplication with the constant RES\_HIGH. Using an invalid value for the channel number this function returns a negative value (-1). The constants AIN0..AIN3 as well as RES\_HIGH are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetDACChannel, PCMSetDACResolution

**Example:**

```
main
{
int ADCin0;
float Uin0;
// ...
//Read ADC Conversion value
// ADCin0 = [0x000, ..., 0x3FF]
ADCin0 = PCMGetADCChannel (AIN0);
// Calculate the input voltage
// of the AD-Converter in Volt
Uin0 = (float)ADCin0*RES_HIGH;
// ...
}
```

Function:	<b>PCMSetDACChannel</b>
Syntax:	BYTE PCMSetDACChannel (BYTE Channel, int ADCOut)
Input:	Channel (R8) = number of output channel (AOUT0, AOUT1) ADCOut (R9) = Conversion value for DAC (8 or 10-bit)
Output:	BYTE (R4) = Error code
Usage:	Writing the analog Outputs AOUT0 or AOUT1.
Comment:	<p>This function expects the conversion value for the DAC as the second parameter. This value can be calculated by dividing the voltage [V] that wants to be output and the constant RES_HIGH or RES_LOW (refer to the example). Which one of the both constants should be used, depends on the current selected resolution. Using the function <b>PCMSet-DACResolution</b> it can be selected between 8- and 10-bit. After initialization the DAC uses a resolution of 10-bit. If the value of the parameter ADCOut exceeds the limit of 0xFF at 8-bit resp. 0x3FF at 10-bit resolution, the value will be limited internally to the allowed maximum value, so that a voltage of 10V. will be at the output. This limitation prevents so called „“ caused by an unintentionally range exceed.</p> <p>If the user defines the symbol CM_EN-ABLE_WARNING (refer to section 8), the internal limitation will be indicated within the calling program by an error code &gt; 0x100 (Warning).</p> <p>The constants AOUT0, AOUT1 as well as RES_LOW and RES_HIGH are defined in the files PCMDRV67.INC resp. PCMDRV67.H.</p>

Refer also to:

PCMGetADCChannel, PCMSetDACResolution

Example:

```
main
{

float Uout0;
int  DACout0;

PCMInitialize ();

// ...

// set the output voltage to 2.75 V
Uout = 2.75;

// convert the output voltage (in volt) into
// the output value for the DA-Converter
// (DACout0 = [0x000, ..., 0x3FF] if 10-bit resolution)
DACout0 = (int) (Uout0/RES_HIGH);

PCMSetDACChannel (AOUT0, DACout0);

// ...

}
```

Function:	<b>PCMSetDACResolution</b>
Syntax:	BYTE PCMSetDACResolution (BYTE Channel, BYTE Resolution)
Input:	Channel (R8) = output channel number (AOUT0, AOUT1) Resolution (R9) = resolution to be set (LOWRES=8-bit, HIGHRES=10-bit)
Output:	BYTE (R4) = Error code
Usage:	Setting the resolution of the analog output AOUT0 or AOUT1 to 8-or 10-bit.
Comment:	The analog output voltage is generated by a PWM channel with a downline active low-pass. Reducing the resolution to 8-bit causes a reduction of WELLIGKEIT of the output voltage. After initialization the DAC works with a resolution of 10-bit. The constants AOUT0, AOUT1 as well as LOWRES and HIGHRES are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetDACChannel

Example:

```
main
{

int DACout0;

// a.o. set DAC-resolution to 10-bit
// (standard resolution)
PCMInitialize ();

DACOut = 0x80;

// DAC- resolution 10-bit:
// DACOut = [0x000 ... 0x3FF]
// DACOut = 0x80 ---> output voltage = 2 V
PCMSetDACChannel (AOUT0, DACOut);

// switch resolution from10-bit to 8-bit
PCMSetDACResolution (AOUT0, LOWRES);

// DAC- resolution 8-bit:
// DACOut = [0x00 ... 0xFF]
// DACOut = 0x80 ---> output voltage = 5V
PCMSetDACChannel (AOUT0, DACOut);

// ...

}
```

## 4.5 Counter Functions

Function: **PCMGetCounter**

Syntax: WORD PCMGetCounter (BYTE Channel)

Input: Channel (R8) = counter channel number  
(CIN20..CIN23)

Output: WORD (R4) = counter level

Usage: Read a counter channel CIN20..CIN23.

Comment: The inputs for counters are alternative functions of the input ports IN20..IN23. The counters will be released by the initialization function and incremented with each rising edge on the corresponding input port. The function **PCMGetCounter** makes a preset of the counter with a certain value possible. Using the function **PCMGetCounterMode** the count direction and edges can be changed. The constants CIN20..CIN23 are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetCounter, PCMSetCounterMode

Example: *Refer to Example for **PCMSetCounter***

**Function:** **PCMSetCounter**

**Syntax:** BYTE PCMSetCounter (BYTE Channel, WORD ChannelValue)

**Input:** Channel (R8) = counter channel number (CIN20..CIN23)  
ChannelValue (R9) = counter value to be set (pre-adjust value)

**Output:** BYTE (R4) = Error code

**Usage:** Setting a counter channel CIN20..CIN23.

**Comment:** The counter inputs are alternative functions for the digital inputs IN20..IN23. The counter inputs will be released by the initialization function and incremented with each rising edge on the corresponding input port. The function **PCMSetCounter** makes a preset of the counter with a certain value possible. Using the function **PCMSetCounterMode** the count direction and edges can be changed. The constants CIN20..CIN23 are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMGetCounter, PCMSetCounterMode

Example:

```
main
{

WORD Counter;

// Preset counter for input IN20
// to a counter value 0x100
PCMSetCounter (CIN20, 0x100);

// ...

// read current counter state
Counter = PCMGetCounter (CIN20);

// ...

}
```

**Function:** **PCMSetCounterMode**

**Syntax:** BYTE PCMSetCounterMode (BYTE Channel, BYTE Edge, bit Direction, bit Run)

**Input:** Channel (R8) = counter channel number (CIN20..CIN23)  
Edge (R9) = count edge (FALLING\_EDGE, RISING\_EDGE, ANY\_EDGE)  
Direction (R15.0) = count direction (UP, DOWN)  
Run (R15.1) = Release Bit (RUN, STOP)

**Output:** BYTE (R4) = Error code

**Usage:** Setting the counter mode .

**Comment:** The counter inputs are alternative functions for the digital inputs IN20..IN23. All constants that have to be hand-over as parameters are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetCounter, PCMGSetCounter

**Example:**

```
main
{

// Set Mode for counter on input IN20
// Backward counter, count on each edge
PCMSetCounterMode (CIN20, ANY_EDGE, DOWN, RUN);

// ...

}
```

## 4.6 Functions for PWM Outputs:

Function: **PCMSetPWMChannel**

Syntax: BYTE PCMSetPWMChannel  
(BYTE Channel, unsigned Frequency, int Pulse,  
bit Run)

Input: Channel (R8) = PWM channel number  
(PWMOUT16, PWMOUT17)  
Frequency (R9) = Frequency (5Hz..65535Hz)  
Pulse (R10) = H-Duration of the output voltage in  
relation to the cycle duration in per cent (0..100%)  
Run (R15.0) = Release Bit (RUN, STOP)

Output: BYTE (R4) = Error code

Usage: Output a PWM signal on output PWMOUT16 or  
PWMOUT17.

Comment: Generation of PWM signals is an alternative function  
of the outputs OUT16 and OUT17. Because of the  
resolution of the PWM registers and internal prescalers  
within the C167 controller the lower cut-off frequency  
of the PWM signal is 5 Hz. If the frequency declared  
as parameter is below 5 Hz, it will be ignored and this  
parameter will be processed as 5 Hz internally. If the  
user defines the symbol PCM\_ENABLE\_WARNING  
(refer to section 8) the internal increase is detected by  
an error code > 0x100 (Warning) in the calling  
program. The parameter Pulse defines the duration of  
the turn-on time for this output (H-width) in per cent  
in relation to the entire cycle duration. If the parameter  
is set to Pulse = 0 the output will be permanent on  
L-level, Pulse = 100 causes a permanent H-level. If  
Pulse = 25, the output signal is on H-level for one  
fourth of the cycle duration time.

The constants PWMOUT16, PWMOUT17, RUN and STOP are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetPWMRegister, PCMSetOutPort

Example:

```
main
{

PCMInitialize ();

// ...

// Output a PWM-signal on Output OUT16
// Parameter:
// Frequency = 1kHz (cycle duration = 1ms)
// H-Duration = 75% (corresponds to 0.75ms)
// resulting PWM-signal:
// 0.75ms H-level / 0.25ms L-level

PCMSetPWMChannel (PWMOUT16, 1000, 75, RUN);

// ...

// Turn-off PWM-Output
PCMSetPWMChannel (PWMOUT16, 0, 0 STOP);

// ...

}
```

**Function:** **PCMSetPWMRegister**

**Syntax:** void PCMSetPWMRegister (BYTE Channel, unsigned PP, unsigned PW, bit ClockSel)

**Input:** Channel (R8) = PWM channel number (PWMOUT16, PWMOUT17)  
PP (R9) = direct value for PPx  
PW (R10) = direct value for PWx  
ClockSel (R15.0) = InputClockSelection-Bit (0=CPUclock/1, 1=CPUclock/64)

**Output:** ---

**Usage:** Write a PWM signal to output PWMOUT16 or PWMOUT17 by direct setting of the PWM registers.

**Comment:** The PWM channel will run in standard mode (Mode 0, Edge Aligned PWM). The PWM channel will be operated in standard mode (Mode 0, Edge Aligned PWM). Using this function the internal negation of the output signal done in the driver unit will be compensated. So that the output will be at L-level for the interval of the PWM-timer  $0000 \leq PTx < PWx$ . The constants PWMOUT16 and PWMOUT17 are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMSetPWMChannel, PCMSetOutPort

Example:

```
main
{

PCMInitialize ();

// ...

// Write a PWM-signal to output OUT16 by
// direct programming of the PWM-Registers
// Parameter:
// Frequency = 1kHz (cycle duration = 1ms)
// H-Period = 75% (corresponds to 0.75ms)
// resulting PWM-signal:
// 0.75ms H-level / 0.25ms L-level

// Register values at 20MHz CPU-Clock
// PP0 = 20000
// PW0 = 5000
// CPUclock/1

PCMSetPWMRegister (PWMOUT16, 20000, 5000, 0);

// ...

// Turn-off the PWM-output
PCMSetOutPort (OUT16, 0);

// ...

}
```

## 4.7 Functions for Access to the Display Units

Functions: PCMSetRunLED, PCMSetSysErrLED,  
PCMSet-CANErrLED

Syntax: void PCMSetRunLED (bit State)  
void PCMSetSysErrLED (bit State)  
void PCMSetCANErrLED (bit State)

Input: bit (R15.0) = LED state (ON/OFF)

Output: ---

Usage: Turn-on resp. turn-off the Run LED, SystemError-LED  
or CANError-LED.

Comment: The constants ON (=1) and OFF (=0) are defined in  
the files PCMDRV67.INC resp. PCMDRV67.H.

Example:

```
main
{
// a.o. turn-off LED's
PCMInitialize ();

PCMSetRunLED (ON);

// ...

if (error)
{
PCMSetSysErrLED (ON);
PCMSetRunLED (OFF);
}
}
```

Function:     **PCMGetSwitch**

Syntax:       BYTE PCMGetSwitch (void)

Input:        ---

Output:       BYTE (RL4) = position of the Run/Stop switch  
              SWITCH\_STOP     (0) -> position STOP  
              SWITCH\_RUN    (1) -> position RUN  
              SWITCH\_MRES(2) -> position MRES

Usage:        Query the Run/Stop switch.

Comment:     The constants SWITCH\_STOP, SWITCH\_RUN and SWITCH\_MRES are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

Example:

```
main
{
// a.o. turn-off LED's
PCMInitialize ();

while (PCMGetSwitch() != SWITCH_RUN);
PCMSetRunLED (ON);

do
{

// cycle loop

}
while (PCMGetSwitch() == SWITCH_RUN);
PCMSetRunLED (OFF);
}
```

**Function:** **PCMGetHexNumber**

**Syntax:** BYTE PCMGetHexNumber (void)

**Input:** ---

**Output:** BYTE (RL4) = number adjusted on the Hex- decode switch

**Usage:** Query the number adjusted on the Hex-encoding switch.

**Comment:** The Hex decode switch is reserved for the CPU address. This is only relevant for systems using several CPU's.

**Example:**

```
main
{

BYTE CPUAddr;

// ...

// Get CPU-address
CPUAddr = PCMGetHexNumber ();

printf ("selected CPU address: %02BX\n",
        CPUAddr);

// ...

}
```

Function:     **PCMGetDIPSwitch**

Syntax:       BYTE PCMGetDIPSwitch (void)

Input:         ---

Output:        BYTE (RL4) = value adjusted on DIP-switches

Usage:         Query the value of the DIP-switch adjusted using the slides 1 and 2.

Comment:       The DIP-switch is reserved for selection of the CAN transmission baudrate. If the COMBI-Modul 167 is equipped with a second serial interface the DIP-switch is dropped, so that the function **PCMGetDIPSwitch** returns undefined values.

Example:

```
main
{

BYTE DIPSw;

// ...

// Query DIP-Schalter
DIPSw = PCMGetDIPSwitch ();

printf ("DIP-Switch: %02BX\n", DIPSw);

// ...
}
```

## 4.8 Functions for Releasing Interrupts

Function: **PCMSetInputInterrupt**

Syntax: BYTE PCMSetInputInterrupt  
(BYTE Port, BYTE Edge, bit Enable)

Input: Port (R8) = number of the ports to be released  
(IN0..IN15)  
Edge (R9) = Interrupt edge  
(FALLING\_EDGE, RISING\_EDGE, ANY\_EDGE)  
Enable (R15.0) = Release-Bit (ENABLE, DISABLE)

Output: BYTE (R4) = Error code

Usage: Suspend or release the interrupt for input port. The procedure how to use interrupt functions is described in *section 6*. All constants to hand-over as parameter are defined in the files PCMDRV67.INC resp. PCMDRV67.H.

*Refer also to:*

PCMGetInGroup1, PCMGetInGroup2, PCMGetInPort

Example: A more detailed description is given in *section 6*.



## 5 Timer Functions

The driver library with all the functions described so far is complemented by a second library. Using this second library the timer 8 of the C167 can be used as system timer. The exclusion of this timer function into a separate library relieves the use of resources, used by the system timer (Timer 8 respectively Interrupt 3Eh) for own applications. For this the linking process has only to go without the use of the PCMTMR67.LIB library. All other driver functions can be used without any restrictions.

The timer library PCMTMR67.LIB puts the following functions to users disposal:

Function:	<b>PCMInitTimer</b>
Syntax:	WORD PCMInitTimer (void)
Input:	---
Output:	WORD (R4) = driver's version number (identical with version number for PCMDRV67.LIB)
Usage:	Initialization of timer 8 as system timer with a resolution of 1 ms. The initialization has to be done additional to the call of function <b>PCMInitialize</b> .

Function:     **PCMGetTimerTicks**

Syntax:       DWORD PCMGetTimerTicks (void)

Input:         ---

Output:        DWORD (R4:R5) = Number of timer ticks

Usage:         Calculating the time in ms (number of timer ticks)  
                  since call of function PCMInitTimer.

Comment:       To standardize the timer ticks the constant  
                  CLOCKS\_PER\_SEC is defined in the files  
                  PCMTMR67.INC resp. PCMTMR67.H. In  
                  PCMTMR67.H the symbol **time\_t** is defined as data  
                  type for the time.

Function:     **PCMDisableTimer**

Syntax:       void PCMDisableTimer (void)

Input:         ---

Output:        ---

Usage:         Deactivate timer 8 and suspend the timer interrupt.

The following example is to make the use of the timer function clear:

```
void main (void)
{

time_t Time, Delay;

    PCMInitialize ();
    PCMInitTimer ();

// delay time = 5 seconds
    Delay = 5*CLOCKS_PER_SEC;

// calculate current system time
    Time = PCMGetTimerTicks ();

// delay of 5 seconds
    while ( (Time+Delay) > PCMGetTimerTicks() );

    // ...
}
```



## 6 Using Interrupts

The input ports IN0..IN15 on the COMBI-Modul 167 are leaded internal to port 2 of the C167 controller. That is why they can release edge triggered interrupts using the CAPCOM unit. After calling the initialize function all interrupts are suspend first. The function **PCMSetInputInterrupt** can be used to determine the release edge of the interrupt and enable them for any of the input ports IN0..IN15. Please note that this function only initializes the CAPCOM-registers for the respective channel according to the parameters given. An interrupt handler can not be installed by this function. The user has to declare a suitable interrupt function that can used as an handler by his own. This can be done as described below:

```
void function name (void) INTERRUPT INx_VECT
{
    // ...
}
```

The constant INx\_VECT has to be replaced corresponding to number of the input ports used by IN0\_VECT..IN15\_VECT. These constants are defined in the files PCMDRV67.INC resp. PCMDRV67.H, too.

The explicit prevision of the interrupt handler by the user is necessary, because the interrupt vector table is only situated in RAM at the time using the monitor mode. Otherwise it is located in the Flash and therefore not can be changed at program's run time. Therefore the interrupt vector must be known at the moment of compiling, so that a segment with an appropriate jump instruction can be generated statically.

The example shown below is to make the use of the interrupt input (interrupt at the falling edge on input) clear:

```
void Input2Handler (void) INTERRUPT IN2_VECT
{
    // ...
}

void main (void)
{
    PCMInitialize ();

    // Release interrupt for input 2
    PCMSetInputInterrupt (IN2, FALLING_EDGE,
                          ENABLE);

    // ...

    // Suspend interrupt for input 2
    PCMSetInputInterrupt (IN2, 0, DISABLE);
}
```

The values for the priority registers ILVL and GLVL in the CAPCOM-Unit are calculated by the driver function internally in dependence of the input port number, using the algorithm described below:

$$\begin{aligned} \text{ILVL} &= \text{BASEILVL} + \text{Port number}/4 && (\text{BASEILVL} = 4) \\ \text{GLVL} &= \text{Port number modulo } 4 \end{aligned}$$

The input IN0, ILVL = 4 and GLVL = 0, posses the lowest priority. The values ILVL = 4 and GLVL = 1 are used for input IN1. The highest priority posses input IN15 with ILVL = 7 and GLVL = 3. The CPU interrupt level (ILVL in PSW) will be set to the value ILVL = 3 only once by the initialization function **PCMInitialize**, if the CPU interrupt level was equal or higher than 4 at the time of the function call.. In opposite to that the function PCMSetInputInterrupt does not change the ILVL-value any more, so therefore the occur of an interrupt can be stopped by modification of the PSW value within the application program.

## **7 Error Codes**

The error codes returned by the driver functions are defined in the file PCMDRV67.INC resp. PCMDRV67.H. The meaning of these codes is listed below:

***PCM\_SUCCESSFUL (0x00):***

Function done successful

***PCM\_INVALID\_CHANNEL (0xFF):***

The channel number used for this function call is not valid

***PCM\_INVALID\_AD\_CHANNEL (0xFFFF):***

invalid channel number for ADC

***PCM\_INVALID\_EDGE (0xFE):***

The value for the count- and interrupt signal edge used for this function call is not valid

***PCM\_INVALID\_RESOLUTION (0xFD):***

The value for the PWM resolution used for this function call is not valid

***PCM\_INVALID\_PULSE\_VALUE (0xFC):***

The value for the pulse width used for this function call is not valid

If the user defines the symbol PCM\_ENABLE\_WARNING (*refer to section 8*) the internal limitation of parameters will be reported by a warning to the calling program:

***PCM\_SUPPRESS\_OVERFLOW (0x0100):***

Parameter limited internal

## 8 PCM\_ENABLE\_WARNING Switch

The symbol `PCM_ENABLE_WARNING` is not defined in the default state. That means, the standard declarations for the prototypes of the driver functions are valid, so that they only give errors back as return value. With that all errors can be recognized by invalid parameters because these errors give an error code lower than `0x100`. Such an error means the function could not be executed.

If the user defines the symbol `PCM_ENABLE_WARNING`, the return code of certain functions will be extended to a `WORD`. So warnings can be recognized by the calling function too. These warnings indicate that a certain parameter is limited internal to his permissible value, f.e. overflow of the output value on analog modules. Warnings come with an return code larger than `0x100`. So they assume the extension of the return value to register `RH4`.

If the symbol `PCM_ENABLE_WARNING` is not defined (default state), by oppression of warnings the return value coming from the driver can be analyzed easier.

```
int DACOut;
```

```
DACOut = (int)(9.99/RES_HIGH);
```

```
if ( PCMSetDACChannel(AOUT0, DACOut) )  
{  
    // error condition  
}
```

## 9 Software Structure

Easy access to the board's function components is enabled with the driver function for the COMBI-Modul 167. The following files are included in the standard driver contents:

PCMDRV67.LIB	Function library to access the COMBI-Modul 167 function components
PCMTMR67.LIB	Function library for system timer
PCMDRV67.H PCMDRV67.INC	Definition file for library with driver functions (constants, return codes, prototypes)
PCMTMR67.H PCMTMR67.INC	Definition file for library with system timer (constants, return codes, prototypes)
PCMDEMO.C	Demo program for PCMDRV67.LIB and PCMTMR67.LIB library functions
PCMDEMO.L66	Linker file for demo program
START167.A66	Startup for demo program

Source code for these libraries is available for purchase from PHYTEC. When purchasing the drivers in source code the following files will be added to the software:

PCMDRV67.A66	Source code for the function library to access the COMBI-Modul 167 function components
PCMTMR67.A66	Source code for the system timer function library
PCM67DEF.INC	Definition of internal constants for the driver libraries



## 10 Hints on Using the Demo Program

Some external connections are required in order to demonstrate the complete capabilities of the demo program PCMDemo.C:

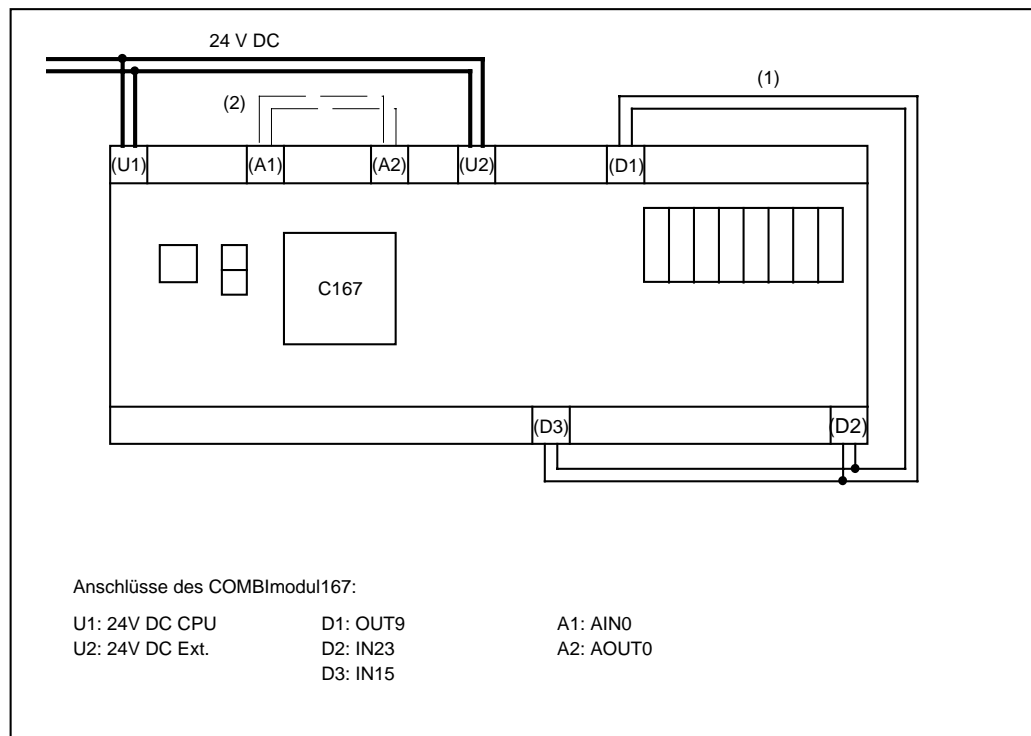


Figure 1: External Connections

- (1) Output OUT9 has to be connected to inputs IN15 and IN23
- (2) Output AOUT0 has to be connected to input AIN0

It is not necessary that the connections, mentioned above, exist at the same time. Either connection 1 or 2 can be cancelled.



- The voltage value, that is output to the analog output port AOUT0, is calculated based on the system time (in seconds). The residual value of the system time divided by 10 (system time modulo 10) is used for the voltage value in volt. Thus discrete voltage values at 1-volt steps in the range of 0 to 9 Volt (Uout0) come out.. To calculate the output value (DACout0) for the DA-Converter out of these voltage values, the voltage value must be divided by the constant RES\_HIGH (resp. RES\_LOW @ 8-bit resolution).
- The value, read from the analog input AIN0 (ADCin0) corresponds to the direct conversion value of the ADC. The voltage value in volt (Uin0) belonging to this is calculated by multiplication with the constant RES\_HIGH.



## 11 Appendix A

The driver functions for the COMBI-Modul 167 use on-chip components of the C167 as listed below:

- P2, P3, P4, P5
- DP3, DP7, DP8
- PP0..3, PW0..3, PT0..3
- PWMCON0, PWMCON1
- CC0IC...CC15IC
- CCM0..CCM3
- T3..T6
- T3CON..T6CON
- T3IC..T6IC
- ADCON
- The CPU interrupt level (Bitfeld ILVL im PSW) is limited to a value of ILVL = 3 (if ILVL was set to a value lower/equal 3 before calling the function **PCMInitialize**, this value will not be changed by the initialization function).

If using the library for the system timer additional resources will be taken, as shown below:

- T8
- T78CON (bit fields for T7 unchanged)
- T8REL
- T8CON
- Interrupt vector 3Eh



---

**Index**
**A**

Alternate Function.....7

**C**

Constants Definition.....4

Counter Direction.....27

Counter Flank.....27

Counter Inputs.....25

Counter Mode.....27

Counters .....24

**D**

DAC Resolution.....20

Demo Program .....49

DIP Switch .....37

**E**

Error Codes .....45

**H**

Hex-encoding switch.....36

**I**

ILVL.....44

Input Ports .....4

Interrupt Level.....44

Interrupts .....38, 43

INx\_VECT .....43

**L**

LEDs .....33

**M**

Memory Model.....6

**O**

On-chip Components.....5

Output Ports .....4

**P**

PCMDRV67.LIB.....3, 9

PCMGetCounter.....24

PCMGetDIPSwitch.....37

PCMGetInGroup1 .....12

PCMGetInGroup2 .....13

PCMGetInPort.....14

PCMInitialize .....5, 11

PCMSetCANErrLED .....33

PCMSetCounter.....25

PCMSetInputInterrupt.....43

PCMSetOutGroup1 .....15

PCMSetOutGroup2 .....16

PCMSetOutPort.....17

PCMSetPWMRegister.....31

PCMSetRunLED .....33

PCMSetSysErrLED.....33

PCMTMR67.LIB .....3, 39

Primary Function.....7

PWM channel.....22

PWM Signal .....29, 31

**R**

RES\_HIGH.....20

RES\_LOW.....20

Run/Stop Switch.....35

**S**

Status LEDs.....33

SWITCH\_MRES.....35

SWITCH\_RUN.....35

SWITCH\_STOP.....35

System Timer.....39

**T**

time\_t.....40

Timer Functions .....39



---

<b>Document:</b>	<b>Driver for COMBI-Modul 167</b>
<b>Document number:</b>	<b>L-344e_1, April 2002</b>

---

**How would you improve this manual?**

---

---

---

---

---

**Did you find any mistakes in this manual?** \_\_\_\_\_ page

---

---

---

---

**Submitted by:**

Customer number: \_\_\_\_\_

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

\_\_\_\_\_

**Return to:**

PHYTEC Technologie Holding AG  
Postfach 100403  
D-55135 Mainz, Germany  
Fax : +49 (6131) 9221-33

Published by

**PHYTEC**

---

© PHYTEC Meßtechnik GmbH 2002

Ordering No. L-344e\_1  
Printed in Germany