



# PCM-VDX Digital I/O DOS Utilities Package

## 1 Introduction

**1.0** The PCM-VDX DOS Utilities Package consists of an application programming interface (API) and example application programs.

**1.1** The applications were built with Borland C++ Version 3.1.

**1.2** The applications are for use with WinSystems, Inc. PCM-VDX-1-256 and PCM-VDX-2-512 Single Board Computers (SBC) which provide 16-lines of TTL-compatible digital I/O.

**1.3** This software is provided on an 'as-is' basis and no warranty as to usability or fitness of purpose is inferred or claimed.

**1.4** WinSystems, Inc. does not provide support for the modification of these programs. Customer application specific queries can be sent to: [support@winsystems.com](mailto:support@winsystems.com).

## 2 Installation and Build

**2.1** The API functions and the sample application programs are provided in source code form in a compressed zipped folder.

**2.2** Each application can be built by creating a Project within the Borland C++ IDE. The API file (*pcmvdx.c*) and the specific application file should be added to the project.

**2.3** An executable file is built that can be run from a DOS prompt. Each program will specify if any command line arguments are required.

### 3 Application Usage

**3.1** The PCM-VDX Digital I/O consists of sixteen dedicated programmable I/O pins consisting of two individual 8-bit ports. Each port can be configured as GPIO or Pulse Width Modulation (PWM) outputs.

All GPIO pins are independent and can be configured as inputs or outputs. When configured as outputs, pins have 8 mA drive capability and are unterminated; when configured as inputs, pins are pulled-high with a 75k ohm resistance. Each input pin also supports interrupt triggers.

All PWM pins are independent and can be configured to output a continuous frequency or a fixed number of pulses. The frequency is selected by programming high and low pulse count values. An interrupt can be used to indicate when a pulse count has completed.

The features are configured and controlled utilizing PCI configuration and I/O access instructions.

**3.2** The file *pcmvdx.c* implements the application interface and presents to applications a set of standard C functions that may be called directly from the application. An application must include *pcmvdx.h*.

**3.3** All application programs assume that any Multi-Function Port being used has been configured for GPIO using the BIOS Setup Utility.

#### **3.4 Application Programming Interface**

The Application Programming Interface file (*pcmvdx.c*) provides the following functions.

##### **3.4.1 void init\_gpio(void)**

This function initializes all I/O pins as inputs, disables all interrupt sensing, and sets the internal port and direction image values. It requires no arguments.

##### **3.4.2 int init\_pwm(unsigned int addr, unsigned int irqnum, unsigned long pwmclk)**

This function initializes the PWM I/O base address, interrupt, and clock source and disables all interrupt sensing. It requires arguments of I/O address, interrupt number, and PWM internal clock frequency. The address provides a 16-bit I/O base address for the PWM registers. It must be mapped on a 512-byte boundary. The interrupt number provides the IRQ path used to generate the CPU interrupt. The clock argument selects either a 10 MHz (0) or 50 MHz (1) internal clock source. A zero is returned if the function is successful. If an invalid interrupt is selected, a one is returned.

##### **3.4.3 void set\_port\_dir(int port, int direction)**

This function configures each bit of the specified port as either input or output. It requires arguments of I/O port and direction. The port provides the GPIO port (0 or 1) and the direction configures each bit as either an input (0) or output (1).

**3.4.4 int read\_bit(int port, int bit\_number)**

This function returns the state of the specified I/O pin. It requires arguments of I/O port and bit number. The port provides the GPIO port (0 or 1) and the bit number (0-7) provides the bit location to read. The value is returned.

**3.4.5 void write\_bit(int port, int bit\_number, int val)**

This function writes a 0 or 1 to the specified I/O pin. It requires arguments of I/O port, bit number, and value. The port provides the GPIO port (0 or 1) and the bit number (0-7) provides the bit location to write. The value either clears the bit (0) or sets the bit (1).

**3.4.6 void set\_bit(int port, int bit\_number)**

This function sets the specified bit in the specified port. It requires arguments of I/O port and bit number. The port provides the GPIO port (0 or 1) and the bit number (0-7) provides the bit to set.

**3.4.7 void clear\_bit(int port, int bit\_number)**

This function clears the specified bit in the specified port. It requires arguments of I/O port and bit number. The port provides the GPIO port (0 or 1) and the bit number (0-7) provides the bit to set.

**3.4.8 int enab\_int(int port, int bit\_number, int irqnum, int polarity)**

This function enables an interrupt for the specified port and bit at the specified polarity. This function does not setup the interrupt controller, nor does it supply an interrupt handler. It requires arguments of I/O port, bit number, interrupt number, and polarity. The port provides the GPIO port (0 or 1) and the bit number (0-7) provides the bit interrupt to enable. The interrupt number provides the IRQ used to generate the CPU interrupt. The polarity selects whether the interrupt is active high (1) or active low (0). A zero is returned if the function is successful. If an invalid interrupt is selected, a one is returned.

**3.4.9 void disab\_int(int port)**

This function disables interrupts for the specified port. It requires an argument of I/O port. The port provides the GPIO port (0 or 1).

**3.4.10 void clr\_int(int port, int bit\_number)**

This function clears the specified bit interrupt status once it has been recognized. The interrupt remains enabled. It requires arguments of I/O port and bit number. The port provides the GPIO port (0 or 1) and the bit number provides the bit (0-7) to clear.

**3.4.11 int get\_int(void)**

This function returns the first bit with an interrupt pending. It requires no arguments. It returns a value from 1 (Port 0 – Bit 0) to 16 (Port 1 - Bit 7). If no interrupt is pending, a zero is returned. This function does NOT clear the interrupt.

**3.4.12 void enab\_pwm(int chan, unsigned long low, unsigned long high)**

This function enables the specified PWM channel in continuous mode and programs both the pulse low and high registers. It requires arguments of PWM channel, pulse low count, and pulse

high count. The channel provides the PWM bit to enable (0-15). The pulse low count and pulse high count generate the frequency of the waveform ( $0-2^{16}$ ).

#### **3.4.13 void enab\_pwm\_rc(int chan, unsigned long low, unsigned long high, unsigned long repeat)**

This function enables the specified PWM port in non-continuous mode. The repeat count value and both the pulse low and high registers are loaded with the selected values. It requires arguments of PWM channel, pulse low count, pulse high count, and repeat count. The channel provides the PWM bit to enable (0-15). The pulse low count and pulse high count generate the frequency of the waveform ( $0-2^{16}$ ). The repeat count determines the number of pulses to generate. If enabled an interrupt is generated at completion of the repeat count.

#### **3.4.14 void disab\_pwm(int chan)**

This function disables the specified PWM port and resets both the pulse low and high counts. It requires a single argument of PWM channel. The channel provides the PWM channel (0-15) to disable.

#### **3.4.15 void enab\_pwm\_int(int chan)**

This function enables interrupts for the specified PWM channel. It is only valid in non-continuous mode. This function does not setup the interrupt controller, nor does it supply an interrupt handler. It requires a single argument of PWM channel. The channel provides the PWM channel (0-15) interrupt to enable.

#### **3.4.16 void disab\_pwm\_int(int chan)**

This function disables interrupts for the specified PWM channel. It requires a single argument of PWM channel. The channel provides the PWM channel (0-15) interrupt to disable.

#### **3.4.17 void clr\_pwm\_int(int chan)**

This function clears the specified PWM channel interrupt status once it has been recognized. The interrupt remains enabled. It requires a single argument of PWM channel.

#### **3.4.18 int get\_pwm\_int(void)**

This function returns the first PWM channel with an interrupt pending. It requires no arguments. It returns a value from 1 (PWM0) to 16 (PWM15). If no interrupt is pending, a zero is returned. This function does NOT clear the interrupt.

## 4 Sample Applications

The application programs provide demonstrations of all of the functions available in the *pcmvdx.c* API. All programs assume that both Port 0 and Port 1 have been appropriately configured for either GPIO or PWM mode using the BIOS Setup Utility. Some of the programs require an oscilloscope to verify signals on the selected PWM channel.

### 4.1 GPIO\_Prt

This program requires all bits of Port 0 to be connected to the corresponding bits of Port 1. It configures all bits of Port 0 as output and those of Port 1 as input. It then verifies that each bit of Port 0 can be set and cleared. The ports directions are then reversed and each bit of Port 1 is tested and verified.

### 4.2 GPIO\_Int

This program requires a single bit of Port 0 to be connected to the corresponding bit of Port 1. The global variable *BitNo* should reflect which bit was selected. It configures the Port 0 bit as output and the Port 1 bit as input. It also enables interrupts on the Port 1 bit. The desired IRQ is passed as a command line argument and a new interrupt handler is installed. Each time a key is hit, a single pulse is generated and receipt of the proper interrupt is verified.

### 4.3 PWM\_Prt

This program requires two PWM channels to be connected to an oscilloscope. The selected channels should be reflected in the variables *ch1* and *ch2*. The I/O base address is hard coded to 1200h. If a different address is desired, the *IO\_BASE* definition can be changed. The two channels are programmed with various frequencies and duty cycles while in continuous mode. The waveform properties are displayed on the console and should be verified. Each time a key is hit a new waveform is displayed.

### 4.4 PWM\_Int

This program requires a single PWM channel to be connected to an oscilloscope. The selected channel should be reflected in the variable *ChanNo*. The I/O base address is hard coded to 1200h. If a different address is desired, the *IO\_BASE* definition can be changed. The channel is programmed to output a 10 pulse waveform with a frequency of 2.5 MHz and a 50% duty cycle each time a key is hit. The desired IRQ is passed as a command line argument and a new interrupt handler is installed. Each time a key is hit, the receipt of the proper interrupt is verified.