Universal Driver Software User Manual

DS-MPE-GPIO

PCIe Minicard Digital I/O Module

For Version 7.0.0 and later

Revision A.0 May 2015

Revision	Date	Comment
A.0	5/13/2015	Initial release

FOR TECHNICAL SUPPORT PLEASE CONTACT:

support@diamondsystems.com

Diamond Systems Corporation
555 Ellis Street
Mountain View, CA 94043 USA
Tel 1-650-810-2500
Fax 1-650-810-2525
www.diamondsystems.com

CONTENTS

1. INTRODUCTION

This user manual contains all essential information about DS-MPE-GPIO Universal Driver 7.0 demo applications, programming guidelines and usage instructions. This manual also includes the Universal Driver API descriptions with usage examples.

2. HARDWARE OVERVIEW

2.1 Description

DS-MPE-GPIO is a rugged, low cost 36-channel digital I/O PCIe MiniCard module which is ideal for digital I/O expansion in embedded and OEM applications. An FPGA provides 36 buffered digital I/O lines that can be configured to operate in a simple I/O mode in the form of 8-bit and 4-bit ports, or in counter/timer and PWM modes. Two ports are fixed digital I/O ports with programmable direction in 8-bit groups. One port can be operated as either a 4-bit DIO or 4 counter/timers with 1 input and 1 output per counter. The other port can be operated as either 8 DIO or up to 8 pulse width modulators (PWMs). The DS-MPE-GPIO product comes with the 36-channel PCIe MiniCard digital I/O module, the CK-DAQ02 cable kit, and a hardware kit with jumpers and screws.

2.2 Specifications

- 36 buffered digital I/O lines
- Configurable for up to 4 24-bit PWMs
- Configurable for 4 programmable counter/timers
- 3245 transceivers for high current output
- +3.3VDC input power
- Latching connectors for increased ruggedness
- Support for Windows 7, Windows Embedded 7, and Linux 3.2.0
- Universal Driver support for all functions
- PCIe MiniCard full size form factor (50.95mm x 30mm)
- Operating temperature of -40°C to +85°C

3. GENERAL PROGRAMMING GUIDELINES

3.1 Initialization and exit function calls

All the Universal Driver 7.0 demo applications begin with the following functions and should be called in a sequence to initialize the Universal Driver and the board. These functions should be called prior to the calling of any other DS-MPE-GPIO board specific functions.

1. dscInit ( ), this function initializes Universal Driver
2. MPEGPIOInitBoard(), this function initializes the DS-MPE-GPIO board
3. DSCGetBoardInfo(), this function collects board information from the universal driver and returns boardinfo structure to be used in board specific functions

At the termination of the demo applications the user should call dscFree() function to close the file handles which are opened by the dscInit() function.

These calls are important in initializing and freeing resources used by the driver. Following is an example of the framework for an application using the driver:

```
#include "DSCUD_demo_def.h"
#include "mpegpio.h"
ERRPARAMS errorParams; //structure for returning error code and error string
DSCCBP dsccbp; //structure containing PCI board settings
BoardInfo *bi=NULL; //Structure containing board base address
MPEGPIOINIT Init; //structure containing board FPGA revision ID, major ID
etc.

int main()
{
        if ( (dscInit ( DSC_VERSION ) != DE_NONE) )
        {
                dscGetLastError ( &errorParams );
                printf ( "dscInit error: %s %s\n", dscGetErrorString (
                errorParams.ErrCode ),     errorParams.errstring );
                return 0;
        }
        dsccbp.boardtype = DSC_MPEGPIO;
        dsccbp.pci_slot = 0;
        if (MPEGPIOInitBoard ( &dsccbp ,&Init ) !=DE_NONE)
        {
                dscGetLastError (&errorParams);
                printf ("MPEGPIOInitBoard error: %s %s\n",
                dscGetErrorString(errorParams.ErrCode),
                errorParams.errstring);
                return 0;
        }
        bi = DSCGetBoardInfo ( dsccbp.boardnum );
        /* Application code goes here */
        dscFree ( );
        return 0;
}
```

In the above example, DSC_VERSION, DSC_MPEGPIO, and DE_NONE are macros defined in the included header file, *dscud.h file.*

3.2 Error handling

All the Universal Driver functions provide a basic error handling mechanism that stores the last reported error in the driver. If the application is not behaving properly, check for the last error by calling the function dscGetLastError(). This function takes an ERRPARAMS structure pointer as its argument.

Nearly all the available functions in the Universal Driver API return a BYTE value upon completion. This value represents an error code that will inform the user whether the function call was successful or not. User should always check if the result returns a DE_NONE value (signifying that no errors were reported), as the code below illustrates:

```
BYTE result;

ERRPARAMS errparams;
if ((result = dscInit(DSC_VERSION)) != DE_NONE)
{
      dscGetLastError (&errparams);
      printf (stderr, "dscInit failed: %s (%s)\n", dscGetErrorString(result),
      errparams.errstring);
      return result;
}
```

In this code snippet, the BYTE result of executing a particular driver function (dscInit() in this case) is stored and checked against the expected return value (DE_NONE). Anytime a function does not complete successfully, an error code other than DE_NONE will be generated, and the current API function will terminate. The function dscGetErrorString() provides a description of the error that occurred.

4. UNIVERSAL DRIVER API DESCRIPTION

4.1 MPEGPIODIOConfig

Function Definition
BYTE MPEGPIODIOConfig(BoardInfo* bi, int Port, int Config);

Function Description
This function sets the digital I/O port direction for the selected port.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0-5 for port A, B, C, D, E, F
Config	For ports 0, 1, 3, and 4: 0 = input, 1 = output For ports 2 and 5: this is a 4 bit value where each bit sets the direction for the corresponding bit in the port

Return Value
Error code or 0.

Usage Example
To set port 0 in output mode,

```
Port = 0
Config = 1
MPEGPIODIOConfig (bi,port,config);
```

4.2 MPEGPIODIOConfigAll

Function Definition
BYTE MPEGPIODIOConfigAll(BoardInfo* bi, int* Config);

Function Description
This function sets the digital I/O port directions for all ports at once.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Config	Pointer to 6 configuration values for ports A, B, C, D, E, F

Return Value
Error code or 0.

Usage Example
To set all ports in output mode,

```
for (port = 0;port < 6;port ++)
{
      Config[port] = 1;
}
MPEGPIODIOConfigAll(bi,& Config);
```

4.3 MPEGPIODIOOutputByte

Function Definition
BYTE MPEGPIODIOOutputByte(BoardInfo* bi, int Port, int Data);

Function Description
This function outputs the specified data to the specified port's output register.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0-5 for port A, B, C, D, E, F
Data	8 bit value to write to the port

Return Value
Error code or 0.

Usage Example
To set Port 0, all pins high, except 4th and 7th pin,

```
Port=0;
Data=0x77;
MPEGPIODIOOutputByte(bi, Port, Data);
```

4.4 MPEGPIODIOInputByte

Function Definition
BYTE MPEGPIODIOInputByte(BoardInfo* bi, int port, int* data);

Function Description
This function reads the data from the specified port and returns it in the location specified by the pointer to data.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0-5 for port A, B, C, D, E, F
Data	pointer to receive the data read from the port

Return Value
Error code or 0.

Usage Example
To read port 0 input values and display on the screen,

```
Port=0;
MPEGPIODIOInputByte (bi, Port, &Data);
printf ("The PORT 0: 0x%x",Data);
```

4.5 MPEGPIODIOOutputBit

Function Definition
BYTE MPEGPIODIOOutputBit(BoardInfo* bi, int Port, int Bit, int Value);

Function Description
This function outputs a single bit to an output port. The other bits remain at their current values.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0-5 for port A, B, C, D, E, F
Bit	0-7 indicates the bit position in the port
Value	Bit value, 0 or 1

Return Value
Error code or 0.

Usage Example
To set Port 0 6th bit to 1,

```
Port=0;
Bit=6;
Value=1;
MPEGPIODIOOutputBit (bi, Port, Bit,Value);
```

4.6 MPEGPIODIOInputBit

Function Definition
BYTE MPEGPIODIOInputBit(BoardInfo* bi, int port, int bit, int* value);

Function Description
This function reads in the specified bit from the specified port and returns it in the location specified by the pointer to data.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0-5 for port A, B, C, D, E, F
Bit	0-7 indicates the bit position in the port
Value	pointer to location to receive the bit data; return data is always 0 or 1

Return Value
Error code or 0

Usage Example
To read PORT0 7th bit and display on the screen

```
Value=0;
Port=0
Bit=7;
MPEGPIODIOInputBit (bi, port, bit, &value);
printf ("The PORT0 7th bit value %d : ",value);
```

4.7 MPEGPIOCounterSetRate

Function Definition
BYTE MPEGPIOCounterSetRate(BoardInfo* bi, MPEGPIOCTR *ctr);

Function Description
This function programs a counter for timer mode with down counting. The counter is started immediately. This is a streamlined function intended to provide a convenient way to program a counter/timer for the most common needs.

A DIO pin on port E or F may be used for output. If a DIO pin is selected for counters 0-3 on port E, then port E must also be specifically configured for output by the user. If a DIO pin is selected for counters 4-7 on port F, it will be automatically configured for output by the FPGA.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
MPEGPIOCTR	Structure with the following member variables: Num - Counter number, 0-7 Rate - Counter output frequency, Hz OutEn - enable output onto corresponding I/O pin; 0 = disable output OutPol - 1 = output pulses high, 0 = output pulses low; only used if OutEn ctrOutWidth - 0-3 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks , only used if CtrOutEn = 1 and CtrClock = 2 or 3

Return Value
Error code or 0.

Usage Example
To configure counter 0 with 100Hz, output enabled, polarity high and output pulse width of 1000 clocks,

```
MPEGPIOCTR Counter;
Counter. Num = 0;
Counter.Rate = 100;
Counter.OutEn= 1;
Counter.Outpol = 1;
Counter.ctrOutWidth = 3;
MPEGPIOCounterSetRate (bi, &counter);
```

4.8 MPEGPIOCounterConfig

Function Definition
BYTE MPEGPIOCounterConfig(BoardInfo* bi, MPEGPIOCTR *Ctr);

Function Description
This function programs a counter for up or down counting and starts the counter running.

A DIO pin on port D may be selected as the input source. If a DIO pin is selected as the clock, port D must also be specifically configured for input by the user.

A DIO pin on port E or F may be used for output. If a DIO pin is selected for counters 0-3 on port E, then port E must also be specifically configured for output by the user. If a DIO pin is selected for counters 4-7 on port F, it will be automatically configured for output by the FPGA.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
MPEGPIOCTR	Structure with the following member variables: Num - Counter number, 0-7 Data - Initial load data, 32-bit unsigned binary Clock - Clock source, 0-3 (see FPGA specification for usage) CountDir - 0 = down counting, 1 = up counting Reload - 0 = one-shot counting, 1 = auto-reload (only valid in count down mode) ctrOutWidth - 0-3 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks , only used if CtrOutEn = 1 and CtrClock = 2 or 3

Return Value
Error code or 0

Usage Example
To configure counter 0 with 100Hz, counter direction down, and auto reload,

```
MPEGPIOCTR *Ctr
Ctr.Num = 0;
Ctr.clock = 2; //50MHz
Ctr.data = 50000000/100;
Ctr.Reload = 1;
MPEGPIOCounterConfig (bi, &Ctr);
```

4.9 MPEGPIOCounterRead

Function Definition
BYTE MPEGPIOCounterRead(BoardInfo* bi, MPEGPIOCTR* Ctr);

Function Description
This function latches a counter and reads the value.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Ctr	Structure with the following member variables: Num – Counter number 0-7 Data – return data, 32-bit unsigned binary

Return Value
Error code or 0

Usage Example
To read current value of counter 0 when it is running and display on the screen,

```
unsigned long CtrData;
ctrNum = 0;
MPEGPIOCounterRead (bi, ctrNum, & CtrData);
printf ("The counter value %d ",CtrData);
```

4.10 MPEGPIOCounterFunction

Function Definition
BYTE MPEGPIOCounterFunction(BoardInfo* bi, MPEGPIOCTR* Ctr);

Function Description
This function can be used to program any desired function into a counter, except reading which is done by MPEGPIOCounterRead.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
CtrNum	Counter number, 0-7
CtrData	return data, 32-bit unsigned binary
CtrCmd	Counter command, 0-15 (see FPGA specification for available commands)
CtrCmdData	Auxiliary data for counter command, 0-3 (see FPGA specification for usage)

Return Value
Error code or 0

Usage Example
To reset counter 0 using counter function,

```
MPEGPIOCTR counter;
counter.CtrNo = 0;
counter.CtrCmd =MPEGPIO_COUNTER_CMD_RESET_ONE;
MPEGPIOCounterFunction(bi,&counter);
```

4.11 MPEGPIOCounterReset

Function Definition
BYTE MPEGPIOCounterReset(BoardInfo* bi, int CtrNum);

Function Description
This function resets a counter. When a counter is reset, it stops running, all its registers are cleared to 0, and any DIO line used for input or output is released back to normal DIO operation. The DIO directions for ports D and E, if used by the counter, are NOT automatically reset back to its prior setting, this must be done by the user. The DIO direction for port F (counters 4-7 output) will be automatically reset to the prior setting by the FPGA.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
CtrNum	Counter number, 0-7

Return Value
Error code or 0.

Usage Example
To reset counter 4,

```
int CtrNum = 4;
MPEGPIOCounterReset(bi,CtrNum);
```

4.12 MPEGPIOPWMConfig

Function Definition
BYTE MPEGPIOPWMConfig(BoardInfo* bi, MPEGPIOPWM* MPEGPIOpwm);

Function Description
This function configures a PWM for operation.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
MPEGPIOpwm	Structure with the following member variables: Num - PWM number, 0-3 Rate - output frequency in Hz Duty - initial duty cycle, 0-100 Polarity - 0 = pulse high, 1 = pulse low OutputEnable - 0 = disable output, 1 = enable output on DIO pin Run - 0 = don't start PWM, 1 = start PWM

Return Value
Error code or 0.

Usage Example

To configure PWM 0 with 100 Hz, 50 % duty cycle and output enabled,

```
MPEGPIOPWM MPEGPIOpwm;
MPEGPIOpwm.Num=0;
MPEGPIOpwm.Rate=100;
MPEGPIOpwm.Duty=50;
MPEGPIOpwm.Outputenable=1;
MPEGPIOpwm (bi, & MPEGPIOpwm);
```

4.13 MPEGPIOPWMStart

Function Definition
BYTE MPEGPIOPWMStart(BoardInfo* bi, int Num);

Function Description
This function starts a PWM running.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3

Return Value
Error code or 0

Usage Example
To start PWM 0,

```
Num=0;
MPEGPIOPWMStart (bi,Num);
```

4.14 MPEGPIOPWMStop

Function Definition
BYTE MPEGPIOPWMStop(BoardInfo* bi, int Num);

Function Description
This function stops a PWM.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3 for single PWM or 0xFF / 255 to stop all PWMs

Return Value
Error code or 0

Usage Example
To stop PWM 0,

```
Num=0;
MPEGPIOPWMStop (bi, Num);
```

4.15 MPEGPIOPWMCommand

Function Definition
BYTE MPEGPIOPWMCommand(BoardInfo* bi, MPEGPIOPWM* MPEGPIOpwm);

Function Description
This function is used to modify a PWM configuration.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
MPEGPIOpwm	Structure with the following member variables: Num - PWM number, 0-3 Command - Counter command, 0-15 CmdData - 0 or 1 for auxiliary PWM command data (used for certain commands) Divisor - 24-bit value for use with period and duty cycle commands

Return Value
Error code or 0

Usage Example
To modify PWM configuration at run time, this function is used. To set PWM 0 to 50Hz,

```
MPEGPIOPWM MPEGPIOpwm;
MPEGPIOpwm.Num=0;
MPEGPIOpwm. Command= MPEGPIO_PWM_LOAD_C0_COUNTER;
MPEGPIOpwm. Divisor=50000000/50;
MPEGPIOPWMCommand(bi, &MPEGPIOpwm);
```

4.16 MPEGPIOPWMReset

Function Definition
BYTE MPEGPIOPWMReset(BoardInfo* bi, int Num);

Function Description
This function resets an individual PWM. After the PWM is reset, it stops and its settings are no longer valid. The config function must be used to configure it for further use. Resetting an individual PWM releases its DIO pin back to normal operation.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	0-3 = individual PWM; 0xFF = all PWMs

Return Value
Error code or 0.

Usage Example
To reset PWM channel 0,

```
int Num = 0;
MPEGPIOPWMReset(bi,  Num);
```

4.17 MPEGPOUserInterruptConfig

Function Definition
BYTE MPEGPOUserInterruptConfig(BoardInfo* bi, MPEGPIOUSERINT* MPEGPIOuserint);

Function Description
This function installs a pointer to a user function that runs when interrupts occur. It configures the interrupt handler to run the user function either before or after the standard interrupt function or in standalone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
MPEGPIOuserint	Structure with the following member variables: IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard Function Source - Identifies interrupt source: 1 = digital I/O, 2 = counter 0 output, 3 = counter 1, 4 = counter 2 output, 5 = counter 3 output Enable - 0 = disable interrupts, 1 = enable interrupts

Return Value
Error code or 0.

Usage Example
To install a function to be called whenever interrupt occurs,

```
Void intfun ()
{
printf ("My fun called ");
}
Void main ()
{
      MPEGPIOUSERINT MPEGPIOuserint;
      MPEGPIOuserint. IntFunc   = intfun;
      MPEGPIOuserint.Mode=0;
      MPEGPIOuserint.Source=0;
      MPEGPIOuserint.Enable=1;
      MPEGPIOUserInterruptSet (bi,& MPEGPIOuserint);
}
```

4.18 MPEGPIOUserInterruptRun

Function Definition
BYTE MPEGPIOUserInterruptRun(BoardInfo* bi, int Source);

Function Description
This function is used to start user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	Identifies interrupt source: 1 = digital I/O, 2 = counter 0 output, 3 = counter 1 output, 4 = counter 2 output, 5 = counter 3 output

Return Value
Error code or 0.

Usage Example
To start user interrupts with counter 0 as interrupt source at 100Hz interrupt rate,

```
Source =2;
MPEGPIOUserInterruptRun(bi, 2);
Num=0;
Rate=100;
MPEGPIOCounterSetRate (bi, Num, Rate);
```

4.19 MPEGPIOUserInterruptCancel

Function Definition
BYTE MPEGPIOUserInterruptCancel(BoardInfo* bi, int Source);

Function Description
This function is used to cancel user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	Identifies interrupt source: 1 = digital I/O, 2 = counter 0 output, 3 = counter1 output, 4 = counter 2 output, 5 = counter 3 output

Return Value
Error code or 0.

Usage Example
To cancel running user interrupt and if the user interrupt configured with counter 0 as input source,

```
Source =2;
MPEGPIOUserInterruptCancel(bi, Source);
```

4.20 MPEGPIOInitBoard

Function Definition
BYTE MPEGPIOInitBoard(DSCCB* dsccb, MPEGPIOINIT Init);

Function Description
This function Initialize the board.

Function Parameters

Name	Description
Dsccb	The handle of the board to operate on
Init	MPEGPIOINIT structure variable

Return Value
Error or 0.

Usage Example
```
MPEGPIOINIT init;
dsccbp.boardtype = DSC_MPEGPIO;
MPEGPIOInitBoard (&dsccbp, unit);
```

4.21 MPEGPIOFreeBoard

Function Definition
BYTE MPEGPIOFreeBoard(DSCB board);

Function Description
This function stops any active interrupt processes and frees the interrupt resources assigned to a board.

Function Parameters

Name	Description
Board	The handle of the board to operate on

Return Value
Error or 0

Usage Example
```
MPEGPIOFreeBoard(board);
```

4.22 MPEGPIOLED

Function Definition
BYTE MPEGPIOLED (BoardInfo* bi, int State);

Function Description
This function turns the LED on or off.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
State	1 = on, 0 = off

Return Value
Error or 0.

Usage Example
To LED turn on,

```
MPEGPIOLED (bi, 1);
```

4.23 MPEGPIOConfig

Function Definition
BYTE MPEGPIOConfig(BoardInfo* bi, int Value);

Function Description
This function outputs the specified 2-bit data to the on-board PIC to set the DIO pull-up/down configuration.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Value	Value to store, 0-3 CFG1 = Branch A (DIO port D-F) pull, 1 = UP, 0 = Down CFG2 = Branch B (DIO port A-C) pull, 1 = UP, 0 = Down

Return Value
Error or 0.

Usage Example
To configure all DIO pins in pull up,

```
MPEGPIOConfig(bi, 0x03);
```

5. UNIVERSAL DRIVER APPLICATION

The Universal Driver supports following applications for DS-MPE-GPIO board:

- DIO
- Counter
- PWM
- User Interrupt

5.1 DIO

Description

Digital I/O is fairly straightforward. This function supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte.

Usage instructions

The DIO application provides various operation on DIO channel i.e. input byte, output byte, input bit and output. This section describes about input byte and output byte DIO operation. The DIO port must be configured in either input or output mode based on the DIO operation need to be performed. E.g. configure DIO port in output mode for output byte.

Output Byte:

- Select the configure option from the main menu
- Enter the port number
- Configure the port in output mode:
- Exit from the current menu
- Select output byte option from the main menu
- Enter the port number
- Enter the desired value in Hex to be sent on DIO port

To set PORT 0 all the pins are 5V except pin 3 and pin 7, run the DIO application and provide input as follows:

- Select the configure option from the main menu : 1
- Enter the port number : 0
- Configure the port in output mode: 1
- Exit from the current menu : -1
- Select the output byte option from the main menu : 2
- Enter the port number: 0
- Enter the desired value in Hex to be sent on DIO port : 0x77

Measure the voltage on PORT0 all the pins using a multi meter. It should shows 5V on all the pins except pin 3 and pin 7, the application generated expected voltage.

Input Byte:

- Select the configure option from main menu
- Enter the port number
- Configure the port in the input mode:
- Exit from the current menu
- Select the input byte option from main menu
- Enter the port number

- Reads and displays current voltage levels on the screen

Provide 5V to PORT0 pin0 from VCC. It should read and display 0x01. To see the output, run the DIO application and provide input as follows:

- Select the configure option from the main menu : 1
- Enter the port number : 0
- Configure the port in input mode: 0
- Exit from the current menu : -1
- Select the output byte option from the main menu : 3
- Enter the port number: 0

The application displaying 0x01 on the screen indicates that it is running as expected.

5.2 Counter

Description

Generally the counter is used as rate generator. The counter can also be configured in count up or count down direction.

Usage Instruction

This application gets input from the user as follows:

- Enter the counter number (0-7)
- Enter output frequency(1-50MHZ)
- Enter Output Enable (0=disable, 1=Enable)
- Enter Output Polarity (0=negative, 1= positive; default = 1)
- Enter output pulse width (0-3 ,0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks, default = 3)
- Waits for key press
- If any key is pressed , application Stops rate generator

To generate a 100Hz rate generator using counter 0, run the counter application and provide input as follows:

- Enter the counter number (0-7):2
- Enter output frequency(1-50MHZ):100
- Enter Output Enable (0=disable, 1=Enable) :1
- Enter Output Polarity : 1
- Enter output pulse width :3

Place an oscilloscope probe on the counter 0 output pin (PORTE0 is the output pin for Counter 0) and set the voltage division to 1V range and second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated the expected rate.

- Press Enter key to repeat or ‘q’ key to quit.

5.3 PWM

Description

This application generates a pulse width modulation (PWM) signal. The PWM signal is a method for generating an analog signal using a digital source. The PWM signal consists of two main components that defines its behavior: a duty cycle and a frequency. The duty cycle describes the amount of time the signal is in a high (on) state as a percentage of the total time of it takes to complete one cycle. The frequency determines how fast the PWM completes a cycle (i.e. 1000Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states. By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices. This application gets duty cycle and frequency value from the user and generates a PWM signal.

Usage Instruction

This application gets input from the user as follows:

- Enter PWM Number (0-3)
- Enter output frequency (1-50MHZ)
- Enter polarity value (0-1)
- Enter duty cycle (0-100)
- Output is generated
- Waits for key press
- If any key is pressed, application stops the PWM output

To generate a 100Hz PWM waveform with 50% duty cycle on PWM channel 0, run the PWM application and provide input as follows:

- Enter PWM Number (0-3):0
- Enter output frequency (1-50MHZ):100
- Enter polarity value (0-1):0
- Enter duty cycle (0-100):50

Place an oscilloscope probe on PWM channel 0 output pin (PORTC0 is the output pin for PWM 0) and set the voltage division to 1V range and second division to 1ms. It should show a PWM wave form with 50% duty cycle and 100Hz frequency, the application generated expected rate.

- Press Enter key to repeat or ‘q’ key to quit

5.4 User Interrupt

Description

The user interrupt feature of the Universal Driver enables a user to run their own code when a hardware interrupt is generated by an I/O board. This is useful for applications that require special operations to be performed in conjunction with the interrupt, or applications where the code is run at regular fixed intervals. The Universal Driver includes example programs for each board with the user interrupt capability to illustrate how to use this feature. This application gets interrupt frequency as user input and calls the user function periodically at rate of interrupt frequency.

Usage Instruction

This application gets input from the user as follows:

- Enter clock source (0-3=Counter0-3 output 4= DIO)
- Enter interrupt rate (1-50MHZ)
- It calls user function based interrupt rate
- Press any key to cancel interrupt.

This application installs a function where a count value is incremented by one whenever the function gets called. To confirm the user function is getting called per interrupt rate, run the UserInt application and provide input as follows:

- Enter clock source (1=DIO ,2-5=Counter0-4 output):2
- Enter interrupt rate (1-50MHZ) :100

It calls the user function based on the interrupt rate and prints the count value every second. Since it configured for 100Hz it should display count value as follows:

UserInt count value =0

UserInt count value =100

UserInt count value =200

UserInt count value =300

Press any key to cancel the interrupt.

6. COMMON TASK REFERENCE

6.1 Feature Overview

I/O connectors

DS-MPE-GPIO provides 2 I/O connectors for the attachment of all user I/O signals. These connectors are shown below. The connectors are JST number BM20B-GHDS-G-TF. Diamond's I/O cable number 6980501 (2 per board) is used to connect the user's signals to these connectors. This cable comes in a kit of 2 as part number CK-DAQ02. This cable has a common 2mm pitch IDC connector at the opposite end which may be plugged into a custom board or may be cut off and the wiring then used as needed. Unused signals do not need to be terminated. However if the cable wiring is cut, care should be taken to avoid shorting any unused wires to any other voltages in the system in order to prevent possible damage to the board or incorrect I/O readings.

Digital I/O signals

There are 36 digital I/O signals, divided into six groups as DIOA0- DIOA7, DIOB0- DIOB5, DIOC0- DIOC3, DIOD0- DIOD7, DIOE0- DIOE5 and DIOF0- DIOF3. Ports A-C signals (DIOA0- DIOA7, DIOB0- DIOB5, DIOC0- DIOC3) are on I/O connector J1 and ports D-F signals (DIOD0- DIOD7, DIOE0- DIOE5 and DIOF0- DIOF3) are on connector J2.

Digital I/O signals use 3.3V signaling only. Each signal's direction of ports C and F are independently programmable, whereas Ports A, B, D, E are byte wise programmable. On system startup or reset, all signals are automatically set to input mode.

All digital I/O signals have programmable pull-up/down resistors and are divided into two groups for this purpose. Group A includes I/O signals ports A to C and Group B includes I/O signals ports D to F. All signals within each group have the same pull direction. The default configuration for DS-MPE-GPIO is for all DIO signals to be pulled low.

Counter/timers

The board provides 8 32-bit counter/timers. Counter mode means the circuit will count external events that are connected via one of the digital I/O lines. Timer mode means the circuit will generate output pulses at a user-specified rate. The pulse width is programmable for both polarity (high or low pulse) and width (1, 10, 100, or 1000 clock pulses). The clock for timer mode is provided internally from an on-board 50MHz oscillator. This oscillator is divided by 50 to provide a 1MHz clock for very low pulse rates. The available range of output rates is 50MHz / 20ns (50MHz / 1) to .0002328Hz / 4295 sec (1MHz / 2^32).

The counter/timers use the digital I/O lines for their I/O signals. When a counter/timer is programmed to use external clock or output, the associated digital I/O line is taken over for the counter/timer and a DIO pin on port E or F may be used for output. If a DIO pin is selected for counters 0-3 on port E, then port E must also be specifically configured for output by the user. If a DIO pin is selected for counters 4-7 on port F, it will be automatically configured for output by the FPGA. The I/O pin may be assigned as either an input (clock) to the counter/timer or an output.

The I/O pin assignment is as follows:

+Connector J2 Pin	Input pin	Counter/Timer
2	D0	0
3	D1	1
4	D2	2
5	D3	3
6	D4	4
7	D5	5
8	D6	6
9	D7	7

Connector J2 Pin	Output pin	Counter/Timer
10	E0	0
11	E1	1
12	E2	2
13	E3	3
16	F0	4
17	F1	5
18	F2	6
19	F3	7

Pulse Width Modulators (PWMs)

The board offers 4 24-bit PWMs. Each PWM may be programmed for output frequency, duty cycle, and output polarity. Duty cycle is defined as the percentage of time the output signal will have the indicated polarity during each period. For example, a 1KHz output frequency (1ms period) with 20% duty cycle and positive output polarity will exhibit a repetitive waveform that is high for 0.2ms at the start of the period and low for 0.8ms during the remainder of the period. Each PWM contains 2 counters. Counter C0 controls the output frequency, and counter C1 controls the duty cycle.

The PWMs use the digital I/O lines for their output signals. When a PWM is running and its output is enabled, the associated digital I/O line is taken over to be used as the output for the PWM, and its direction is forced to output. The I/O pin assignment is as follows:

Connector J1 Pin	Output DIO pin	PWM
16	C0	0
17	C1	1
18	C2	2
19	C3	3

6.2 Digital I/O Software Task Reference

This section describes the various data acquisition tasks that may be performed with the DS-MPE-GPIO and gives step by step instructions on how to achieve them using the Universal Driver functions. Tasks include:

- Program entry / exit sequence
- Digital I/O
- Counter/timer operation
- PWM operation
- User interrupts

Program Entry/Exit sequence

1. All driver usage begins with the function MPEGPIOInitBoard(). This function must be called prior to any other function involving the DS-MPE-GPIO.
2. At the termination of the program the programmer may use MPEGPIOInitBoard(), but this is not required. This function is normally used in a development environment where the program is being repeatedly modified and rerun.

Digital I/O operations

Digital I/O operation is relatively simple. First configure the DIO port direction with one of the below functions:

BYTE MPEGPIODIOConfig() configures a port for input or output.

BYTE MPEGPIODIOConfigAll() configures for all the ports at once.

Then execute whichever I/O function is desired. Byte read/write enables 4, 5 or 8 bits of digital I/O to be updated at once. Bit operation enables a single bit to be updated.

BYTE MPEGPIODIOOutputByte() outputs 8 bits of data

BYTE MPEGPIODIOInputByte(BoardInfo* bi, int Port, byte* Data);

BYTE MPEGPIODIOOutputBit(BoardInfo* bi, int Bit, int Value);

BYTE MPEGPIODIOInputBit(BoardInfo* bi, int Bit, int* Value);

To configure the digital I/O pull-up/down resistors, use MPEGPIODIOConfig(). The programmed value is stored in a small flash device on the board, so that the board will retain the latest configuration the next time it is powered up.

Counter/timer operations

The counter/timers are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure the counters for common counting and timing operations. For non-standard or specialized operations, the individual commands can be used to configure the counter/timers exactly as desired.

Simplified Programming

To program a counter/timer as a rate generator with a specific frequency, use MPEGPIOCounterSetRate().The counter is programmed for down counting, and an external clock is selected. The counter output may optionally be enabled onto a digital I/O pin, with programmable polarity and pulse width.

To program a counter/timer for counting operation, use the following functions:

1. Use MPEGPIOCounterConfig() to configure the counter for either up or down counting and start the counter running. A Digital I/O pin may be selected for either the input or the output. This function is typically used to count external events.
2. Use MPEGPIOCounterRead() to read the current contents of the counter. This function can be used repeatedly to monitor the operation. This is normally used with event counting.
3. When the counting function is no longer needed, use MPEGPIOCounterReset() to reset the counter and return any assigned Digital I/O pin to normal digital I/O operation.

Detailed Programming

To program a counter/timer using individual commands, use MPEGPIOCounterFunction(). This function must be used multiple times to execute each command needed to configure the counter. All commands take effect immediately upon execution. The typical command sequences for the most common operations are provided below. See the full list of counter/timer commands in the appendix.

For a rate generator:

Command	Function
15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired.
1	Load counter with desired divisor to select the desired output pulse rate. The output rate is the selected clock frequency divided by the divisor.
2	Select the count direction. For a rate generator the direction should be down.
6	Select clock source. Normally an internal clock (50MHz or 1MHz) will be selected.
7	Enable auto-reload. This means that the counter will operate continuously.

If the rate generator output is desired, use the following two commands:

8	Enable counter output on the associated digital I/O pin. The desired output polarity is also selected with this command. If counters 0-3 are configured for output, the output will only appear on port E if it is also configured for output direction with the DIRE register bit. Enabling output for counters 0-3 does not automatically force port E to output mode. Only the counter for which output is enabled will have its output appear on port E. The other port E bits will operate in normal DIO output mode. If this command is executed for counters 4-7, the corresponding port F pin will be automatically configured for output and override the DIRFn register bit. Reading the DIRFn register bit will return a 1. When the counter output is subsequently disabled, the port F pin will return to its previously programmed direction as indicated by its associated DIRFn register bit.
9	Select the desired output pulse width.

Finally, enable the counter/timer with the following command:

4	Start the counter/timer running. (This function is also used to stop the counter/timer.) When the rate generator is no longer needed, either of the following commands can be used:
4	Stop the counter/timer running. The existing settings are maintained so the counter can be restarted later if desired. If it was assigned for the output pulse, the digital I/O line is still tied to the counter/timer and cannot be used for normal digital I/O operations.

15 Reset the counter/timer. This stops the counter/timer and releases the digital I/O line back to normal digital I/O operation.

Alternatively, the function MPEGPIOCounterReset() can be used to reset the counter/timer.

For an event counter:

15 Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired. This will reset the counter data register to 0.

2 Select the count direction. For an event counter the direction should be up.

6 Select clock source. Normally the associated digital I/O pin will be selected to enable counting external pulses.

7 Enable or disable auto-reload. If auto-reload is enabled, the counter will operate continuously, meaning that when it reaches 2^32-1 it will roll over to zero. In most cases auto-reload will be disabled for event counting.

4 Start the counter/timer running. (This function is also used to stop the counter/timer.)

While the counter is operating, its current count can be read by using the MPEGPIOCounterRead() function. When the counting function is no longer needed, the function MPEGPIOCounterReset() can be used to reset the counter/timer.

PWM operations

The PWMs are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure them for common operations. For non-standard or specialized operations, the individual commands can be used to configure the PWMs exactly as desired.

To configure and start a PWM:

1. MPEGPIOPWMConfig() configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. MPEGPIOPWMStart() can be used to start the PWM running if the config function did not start it.

To stop a PWM:

MPEGPIOPWMStop() stops a PWM from running. The output is driven to the inactive state. For a PWM with positive output polarity, the output will go low.

To restart a PWM that has been stopped, use MPEGPIOPWMStart().

To reset a PWM and return its assigned digital I/O output pin to normal operation, use MPEGPIOPWMReset().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use MPEGPIOPWMCommand(). This function must be executed multiple times, once for each command, to carry out the desired configuration. The available commands are listed in the appendix. All commands take effect immediately upon execution.

User interrupts

Universal Driver enables the installation of user-defined code to be run when an interrupt occurs. The interrupt can be triggered by a variety of sources. The interrupt can run as the only procedure when the interrupt occurs (standalone or alone mode). The available modes depend on the source of the interrupt:

Source	Source number	Modes supported
Digital input interrupts	1	0 Alone
Counter/timer interrupts	2, 3, 4, 5	0 Alone

User interrupts are very easy to use. Just 3 steps are required: Configure, run, and stop.

Configure:
MPEGPIOUserInterruptConfig() selects the source for the user interrupts and also installs a pointer to the user's code to run when the interrupt occurs

Run (alone mode):

1. If a counter/timer is being used to drive interrupts, then configure it with MPEGPIOCounterSetRate().
2. If a digital input is being used to drive interrupts, it is configured with MPEGPIOUserInterruptConfig().
3. Use MPEGPIOUserInterruptRun() to start the interrupt operation.

Stop (alone mode):
Use MPEGPIOUserInterruptCancel() to stop user interrupts.

LED control

The DS-MPE-MPEGPIO contains an LED that is user-programmable. This can be used as a visual indication that the board is responding to commands. Turn the LED on and off, use MPEGPIOLED().

6.3 Performing Digital IO Operations

Description

The driver supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte. Digital I/O is fairly straightforward - to perform digital input, provide a pointer to the storage variable and indicate the port number and bit number if relevant. To perform digital output, provide the output value, the output port and bit number, if relevant.

The five Universal Driver functions described here are MPEGPIODIOConfig (), MPEGPIODIOOutputByte(), MPEGPIODIOInputByte(), MPEGPIODIOOutputBit() and MPEGPIODIOInputBit().

Step-By-Step Instructions:

If a digital input is being performed, create and initialize a pointer to hold the returned value of type byte*.

Some boards have digital I/O ports with fixed directions, while others have ports with programmable directions. Make sure that the port is set to either input or output direction. Use function MPEGPIODIOConfig () to set the direction if necessary.

Call the selected digital I/O function. If bit operations are being performed, then pass in an integer value telling the driver for which particular bit (0-7) of the DIO port which is wish to be operated.

Example of usage for digital I/O operations:

```
BoardInfo *bi;
int port, bit;
int digital_value;      // the value ranges from 0 to 255

/* 1. Configure Port 0 in output mode */

port = 0;
dir = 1;    //    0 = Input, 1 = Output
mode = 0;   //    0 = Normal, 1 = Latched

if ((result = MPEGPIODIOConfig(bi, port, dir, mode)) != DE_NONE)
      return result;

/* 2. input bit - read bit 3 of port 0 (port 0 is in input mode) */
port = 0;
bit = 3;
if ((result = MPEGPIODIOInputBit(bi, port, bit, &digital_value)) != DE_NONE)
      return result;

/* 3. input byte - read all 8 bits of port 0 (port 0 is in input mode) */
port = 0;
if ((result = MPEGPIODIOInputByte(bi, port, &digital_value)) != DE_NONE)
      return result;

/* 4. output bit - 3 examples (assumes port 1 is in output mode) */
port = 1;
bit = 7;

/* 4a. MPEGPIODIOOutputBit(), set bit 7 of port 1 to 1, leave other bits
alone */
if ((result = MPEGPIODIOOutputBit(bi, port, bit, 1)) != DE_NONE)
```

```
      return result;

/* 5. output byte - set port 1 to "01010101" (port 1 is in output mode) */
port = 1;
if ((result = MPEGPIODIOOutputByte(dscb, port, 0x55)) != DE_NONE)
      return result;
```

6.4 Performing Counter Operation

There are two functions available for counter operation; MPEGPIOCounterSetRate() and MPEGPIOCounterConfig(). The MPEGPIOCounterSetRate function configures the counter in down mode and enables the counter output on corresponding DIO pins (optional) whereas MPEGPIOCounterFunction() provides options to configure counter in up or down mode and selects internal or external clock source.

Performing Counter Operation with MPEGPIOCounterSetRate()

Description:

The counter operation makes the counter to run in specified rate. The counter application uses following Universal driver function MPEGPIOCounterSetRate() and MPEGPIOCounterReset().

Step-By-Step instructions:

Create MPEGPIOCTR structure variable to hold the counter settings and initialize counter structure variables, then call MPEGPIOCounterSetRate () to configure the counter and finally call MPEGPIOCounterReset() to stop the running counter.

Example of Usage for Counter Operations:

```
MPEGPIOCTR counter; //Structure to hold counter settings
counter.Num = 0; //Select counter Number
counter .Rate =1000; //Select counter rate
counter.OutEn = 1; // Enable counter output
counter.OutPol = 1; // Select output pulse polarity as high
counter.ctrOutWidth = 3; // Select output pulse width as 1000 clocks

//The following function configures the counter with desired rate


if (MPEGPIOCounterSetRate(bi,&counter) !=DE_NONE)

{
      dscGetLastError (&errorParams);
      printf("MPEGPIOCounterSetRate error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
printf ("Press any key to stop counting \n");
while ( !kbhit())
{
      dscSleep(1000);
      MPEGPIOCounterRead(bi,&counter);
      printf("Counter Data %ld \r",counter.Data);
      fflush(stdout);
}

if(MPEGPIOCounterReset(bi,counter.Num) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("MPEGPIOCounterReset error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

Performing Counter Operation with MPEGPIOCounterConfig()

Description:

The following example shows how to configure the counter in up mode with external clock source. The counter application uses following Universal Driver function MPEGPIOCounterConfig(), MPEGPIOCounterRead() and MPEGPIOCounterReset() to accomplish this task.

Step-By-Step Instructions:

Create MPEGPIOCTR structure variable to hold the counter settings and initialize counter structure variables, then call MPEGPIOCounterConfig(), to configure the counter then call MPEGPIOCounterRead() to read counter data and MPEGPIOCounterReset() to reset the corresponding DIO pin for normal operation.

Example of usage for counter operations:

```
MPEGPIOCTR counter; //Structure to hold counter settings
counter.Num = 0;    //Select counter Number
counter.CountDir = 1; // Select Up direction mode
counter.Clock = 0; // Select external clock source.
counter.Reload = 1; // Enable auto reload
if(MPEGPIODIOConfig(bi,3,0)  !=DE_NONE)  ;  //  DIO  pin  on  port  D  must  be
//selected as the input source for external clock
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIODIOConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
// The following function configures the counter with above settings
if (MPEGPIOCounterConfig(bi,&counter) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOCounterConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
// The following function reads and displays counter data every second.
while ( !kbhit())
{
      dscSleep (1000);
      MPEGPIOCounterRead (bi,&counter);
      printf ("Counter Data %ld \r",counter.Data);
      fflush (stdout);
}

// the following function stops running counter and releases the
//corresponding DIO for normal operation.
if (MPEGPIOCounterReset(bi,counter.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOCounterReset error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

6.5 Performing PWM operation

Description:

The PWM operation generates a PWM signal with desired frequency and duty cycle value. The following functions are used for the PWM operation MPEGPIOPWMConfig(), MPEGPIOPWMStop() and MPEGPIOPWMReset().

Step-By-Step instructions:

Create a MPEGPIOPWM structure variable to hold the PWM settings and initialize PWM structure variables, then call MPEGPIOPWMConfig() to configure the PWM and finally call MPEGPIOPWMStop() application to stop the running PWM signal.

Example of usage for PWM operations:

```
MPEGPIOPWM pwm; //structure to hold PWM settings

pwm.Num = 0; //select PWM channel
pwm.Rate = 100; // Select Output Frequency
pwm.Duty = 50; // Select Duty cycle value
pwm.Polarity = 0; // Select Polarity value
pwm.OutputEnable = 1; //Enable PWM output
pwm.Run = 1;
//The following function configures PWM circuit
if (MPEGPIOPWMConfig(bi,&pwm) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOPWMConfig error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

printf ("Press any key to stop PWM \n");
while ( !kbhit()) //detects any key pressed
{
}
//The following function stops the selected PWM.
if (MPEGPIOPWMStop(bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf ("MPEGPIOPWMStop error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}

// The following function releases corresponding DIO pin for normal operation
if ( MPEGPIOPWMReset(bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError (&errorParams);
      printf (" MPEGPIOPWMReset error: %s %s\n",
      dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
      return 0;
}
```

7. I/O CONNECTORS

7.1 DS-MPE-GPIO Digital GPIO Connector

The digital I/O signals are provided on two miniature 20-pin latching connectors with 16 I/O lines per connector.

DS-MPE-GPIO-Digital GPIO Connector Pinout

+3.3V (fused)	**1**	**2**	DIO A0
DIO A1	**3**	**4**	DIO A2
DIO A3	**5**	**6**	DIO A3
DIO A5	**7**	**8**	DIO A6
DIO A7	**9**	**10**	DIO B0
DIO B1	**11**	**12**	DIO B2
DIO B3	**13**	**14**	DIO B4
DIO B5	**15**	**16**	DIO C0
DIO C1	**17**	**18**	DIO C2
DIO C3	**19**	**20**	Ground

+3.3V (fused)	**1**	**2**	DIO D0
DIO D1	**3**	**4**	DIO D2
DIO D3	**5**	**6**	DIO D3
DIO D5	**7**	**8**	DIO D6
DIO D7	**9**	**10**	DIO E0
DIO E1	**11**	**12**	DIO E2
DIO E3	**13**	**14**	DIO E4
DIO E5	**15**	**16**	DIO F0
DIO F1	**17**	**18**	DIO F2
DIO F3	**19**	**20**	Ground

Description: 20-pin (2x10) 1.25mm pitch vertical SMT latching connector

Part Number: JST BM20B-GHDS-G-TF

8. APPENDIX: REFERENCE INFORMATION

Counter/timer commands

The counter/timers are programmed with a series of commands. Each command is a 4 bit value. A command may have a 2 bit argument, referred to as CCD1-0 in the descriptions below. A series of commands is required to configure a counter/timer for operation. See the counter/timer usage instructions in the manual for more information.

0 Clear the selected counter. If count direction is up the counter register is cleared to 0. If count direction is down the counter register is set to the reload value. All other counter settings are preserved. If the counter is running it continues running.

1 Load the selected counter with data in registers 0-3. This is used for down counting operations only.

2 Select count direction, up or down.

3 Enable / disable external gate. This command is not implemented in this design.

4 Enable / disable counting.

5 Latch selected counter. A counter must be latched before its contents can be read. Latching can occur while the counter is counting. The latched data can be read with the MPEGPIOCounterRead() function.

6 Select counter clock source according to the table below:

Options	Function
0	External input pin, active low; see table below
1	Reserved
2	Internal clock 50MHz
3	Internal clock 1MHz

The counter external input pins are defined in the table below. If any counter is configured for external clock, the input will only work if port D is also configured for input direction with the direction register. Selecting external clock does not automatically configure port D for input, because doing so would adversely affect the other port D bits that are not used for counter clocks. Only the counter for which external clock was selected will have its clock routed to the corresponding port D pin. The other port D pins will operate in normal DIO input mode.

Connector J2 Pin	Input pin	Counter/Timer
2	D0	0
3	D1	1
4	D2	2
5	D3	3
6	D4	4
7	D5	5
8	D6	6
9	D7	7

A counter must be enabled for the external input function to override the normal DIO operation. When one or more counters are reset with command 15, any I/O pins tied to the counter or counters are released to normal DIO operation.

7 Enable / Disable Auto-Reload. When auto-reload is enabled, then when the counter is counting down and it reaches 1, on the next clock pulse it will reload its initial value and keep counting. Otherwise on the next clock pulse it will count down to 0 and stop.

8 Enable counter output and select the output pulse polarity. The initial logic level of the output pin will be the inactive state (the opposite state of the selected polarity). The output pulse always comes at the end of each clock period. The counter outputs are enabled on DIO pins according to the following table. If counters 0-3 are configured for output, the output will only appear on port E if it is also configured for output direction with the direction register bit. Enabling output for counters 0-3 does not automatically force port E to output mode. Only the counter for which output is enabled will have its output appear on port E. The other port E bits will operate in normal DIO output mode.

If this command is executed for counters 4-7, the corresponding port F pin will be automatically configured for output and override the direction register bit.

Connector J2 Pin	Output pin	Counter/Timer
10	E0	0
11	E1	1
12	E2	2
13	E3	3
16	F0	4
17	F1	5
18	F2	6
19	F3	7

9 Select counter output pulse width. Only has effect when counter output is enabled with command 8 and clock source selected with command 6 is internal 50MHz or 1MHz. The counter output pulse width is defined according to the table below. If the selected pulse width is equal to or greater than the clock period, the output will stay at its active state indefinitely. If external clock is selected, the output pulse is always 1 clock wide, meaning that it will transition to active on the terminal count clock pulse and then transition to inactive on the next clock pulse.

Options	Function
0	1 clock (default if command is not executed; mandatory if external clock is selected)
1	10 clocks
2	100 clocks
3	1000 clocks

15 Reset one or all counters. When any counter is reset, all its registers are cleared to zero, and any DIO lines assigned to that counter for input or output are released to normal DIO operation. A command of 0xFF will reset all counters.

Each counter's output operates as follows: when disabled or during normal counting operation, the output is 0. When count direction is up, the output is always 0. When count direction is down, then when the counter reaches 1, the output will go high, and the counter will reload to its initial value on the next clock pulse. Thus a counter value of n will result in a divide by n output pulse rate. If a counter latch command is requested during this process, the command will be delayed until the reload is completed.

PWM commands

The PWMs are programmed with a series of commands. Each command is a 4 bit value. A command may have a 1 bit argument, referred to as PWMCD in the descriptions below. A series of commands is required to configure a PWM for operation.

0 Stop all / selected PWM as indicated by PWMCD.
0 = stop all PWMs (opposite polarity for “all” compared to other commands)
1 = stop single PWM
Command 0x00 = stop all PWMs.

When a PWM is stopped, its output returns to its inactive state, and the registers are reloaded with their initial values. If the PWM is subsequently restarted, it will start at the beginning of its waveform, i.e. the start of the active output pulse.

1 Load counter C0 or C1 selected by PWMCD:
0 = load C0 / period counter
1 = load C1 = duty cycle counter

2 Set output polarity. The pulse occurs at the start of the period.
0 = pulse high
1 = pulse low

3 Enable/disable pulse output as indicated by PWMCD
0 = disable pulse output; output = opposite of polarity setting from command 2
1 = enable pulse output

4 Reset all PWMs / selected PWM
0 = reset PWM selected with PWM2-0
1 = reset all PWMs

When a PWM is reset, it stops running, and any DIO line assigned to that PWM for output is released to normal DIO operation. The direction of the DIO line will revert to its value prior to the PWM operation.

5 Enable/disable PWM outputs on DIO port F
0 = disable output
1 = enable output on DIO pin; this forces the DIO pin to output mode

6 Select clock source for PWM
0 = 50MHz
1 = 1MHz

7 Start all PWMs / selected PWM

0 = start PWM selected with PWM2-0
1 = start all PWMs
Command 0x7F = start all PWMs.

If a PWM output is not enabled, its output is forced to the inactive state, which is defined as the opposite of the value selected with command 2. The PWM may continue to run even though its output is disabled. PWM outputs may be made available on I/O pins according to the table below using command 5. When a PWM output is enabled, the corresponding DIO pin is forced to output mode. To make the pulse appear on the output pin, command 3 must additionally be executed, otherwise the output will be held in inactive mode (the opposite of the selected polarity for the PWM output).

Connector J1 Pin	Output DIO pin	PWM
16	C0	0
17	C1	1
18	C2	2
19	C3	3