Universal Driver Software User Guide
FP-GPIO96

FeaturePak 96-bit digital I/O module

For Version 7.0.0 and later

1.0 Table of Contents

1.0 Introduction

This user manual contains all essential information about FP-GPIO96 Demo applications programming guidelines and usage instructions. This manual also includes the Universal driver API descriptions with usage example.

2.0 Hardware Overview

2.1 Description

FP-GPIO96 is a general purpose I/O FeaturePakTM module using a high-capacity (700K gate equivalent) PCI Express FPGA for maximum density and flexibility. The base hardware configuration features 96 digital I/O lines grouped into 12 8-bit ports. All ports have I/O buffers to protect the FPGA and offers 3.3V logic drive levels. The ports are organized into a combination of byte-wide, nibble-wide, and bit-wide direction control for maximum flexibility and application compatibility.

The built-in FPGA personality provides multiple configuration options. All 96 I/O lines may be used in common I/O mode. Six of these ports can be reconfigured to enable an array of additional features, including 8 32-bit up/down counter/timers with programmable input source and gate, 4 24-bit PWM circuits with 0-100% duty cycle capability, and interrupt/latched mode operation.

2.2 Specifications

- 96 total I/O lines brought out to the FeaturePak connector
    - 48 I/O lines on the primary I/O connector with 3 8-bit buffers and 2 4-bit buffers for increased direction configuration flexibility
    - 48 I/O lines on the secondary I/O connector with 6 8-bit buffers
- Configurable digital I/O pull-up/down resistors, each I/O group independently configurable
- Byte-wide, nibble-wide and bit-wide port direction control
- 8 32-bit counter/timers for timing and general purpose use
- 4 24-bit pulse-width modulator circuits
- One PCI Express x1 lane host interface
- +3.3VDC input voltage
- FeaturePak form-factor compliant
- Zero height expansion module
- -40°C to +85°C operating temperature
- Universal Driver software support

3.0 General programming guidelines

3.1 Initialization and exit function calls

All demo applications begins with following functions and should be called in sequence to initialize the universal driver and board .These functions should be called prior to calling any other FP-GPIO96 board specific functions.

- dscInit ( ), this function initializes the Universal Driver
- GPIO96InitBoard (), this function initializes the FP-GPIO96 board.
- DSCGetBoardInfo() ,this function collects board information from driver 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 opened in dscInit() function.

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

```
#include "DSCUD_demo_def.h"
#include "fpgpio96.h"
ERRPARAMS errorParams; //structure for returning error code and error string
DSCCBP dsccbp; // structure containing board settings
BoardInfo *bi=NULL; //Structure containing board base address
GPIO96INIT init; // Structure for returning board's FPGA ID, Board 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_FPGPIO96;

        if ( (GPIO96InitBoard (&dsccbp, init )!= DE_NONE) )
        {
                dscGetLastError ( &errorParams );
                printf (  "dscInitBoard 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_ FPGPIO96, and DE_NONE are macros defined in the included header file, *dscud.h file.*

3.2 Error handling

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

Nearly all of 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 as to whether or not the function call was successful. 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( "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.

4.0 Universal Driver API Description

4.1 GPIO96DIOConfig

Function Definition

BYTE GPIO96DIOConfig(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	0-11 for port A, B, C, D, E, F, G, H, J, K, L, M
Config	For ports 0, 1, 3, and 6-11: 0 = input, 1 = output. For port 2: This is a 2-bit value (0-3) where the higher bit is for CH bits 4-7 And, the lower bit is for CL bits 0-3 For ports 4 and 5: this is an 8 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
GPIO96DIOConfig(bi,port,config);
```

4.2 GPIO96DIOConfigAll

Function Definition

BYTE GPIO96DIOConfigAll(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 12 configuration values for ports A, B, C, D, E, F, G, H, J, K, L, M

Return Value

Error code or 0.

Usage Example

To set all ports in output mode

```
for(port = 0;port < 12;port ++)
{
        Config[port] = 1;
}
GPIO96DIOConfigAll(bi,& Config);
```

4.3 GPIO96DIOOutputByte

Function Definition

BYTE GPIO96DIOOutputByte(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	0-11 for port A, B, C, D, E, F, G, H, J, K, L, M
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;
GPIO96DIOOutputByte(bi, port, Data);
```

4.4 GPIO96DIOInputByte

Function Definition

BYTE GPIO96DIOInputByte(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	0-11 for port A, B, C, D, E, F, G, H, J, K, L, M
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;
GPIO96DIOInputByte(bi, port, &Data);
printf("The PORT 0: 0x%x",Data);
```

4.5 GPIO96DIOOutputBit

Function Definition

BYTE GPIO96DIOOutputBit(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	0-11 for port A, B, C, D, E, F, G, H, J, K, L, M
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;
GPIO96DIOOutputBit(bi, port, bit,value);
```

4.6 GPIO96DIOInputBit

Function Definition

BYTE GPIO96DIOInputBit(BoardInfo* bi, int port, int bit, int* value);

Function description

This function reads 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	0-11 for port A, B, C, D, E, F, G, H, J, K, L, M
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;
GPIO96DIOInputBit (bi, port, bit, &value);
printf ("The PORT0 7th bit value %d : ",value);
```

4.7 GPIO96CounterSetRate

Function Definition

BYTE GPIO96CounterSetRate(BoardInfo* bi, int Num, float Rate);

Function description

This function programs a counter for timer mode with down counting and continuous operation (reload enabled).

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
Num	Counter number, 0-1
Rate	Desired output rate, Hz

Return Value

Error code or 0.

Usage Example

To configure counter 0 with 100 Hz

```
        Num=0;
        Rate=100;
    GPIO96CounterSetRate (bi, Num, Rate);
```

4.8 GPIO96CounterConfig

Function Definition

BYTE GPIO96CounterConfig(BoardInfo* bi, GPIO96CTR *Ctr);

Function description

This function performs a general counter configuration operation with all options and starts the counter running.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Ctr	Structure with 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)

Return Value

Error code or 0.

Usage Example

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

```
GPIO96CTR *Ctr
Ctr.Num = 0;
Ctr.clock = 2; //50MHz
Ctr.data = 50000000/100;
Ctr.Reload = 1;
GPIO96CounterConfig(bi,&Ctr);
```

4.9 GPIO96CounterOutputEnable

Function Definition

BYTE GPIO96CounterOutputEnable(BoardInfo* bi, int Group, int Enable);

Function description

This function enables the counter I/O signals on DIO ports in groups of 4 counters (0-3 or 4-7).

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
Group	0 = counters 0-3, 1 = counters 4-7
Enable	0 = disable I/O for selected group, 1 = enable I/O for selected group

Return Value

Error code or 0

Usage Example

To enable counter 0-3 output on DIO pin

```
Group = 0;
Enable = 1;
GPIO96CounterOutputEnable(bi, Group,Enable);
```

4.10 GPIO96CounterRead

Function Definition

BYTE GPIO96CounterRead(BoardInfo* bi, int CtrNum, unsigned long * CtrData);

Function description

This function latches a counter and reads the value.

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

Return Value

Error code or 0

Usage Example

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

```
unsigned long CtrData;
ctrNum = 0;
GPIO96CounterRead(bi, ctrNum, & CtrData);
printf("The counter value %d ",CtrData);
```

4.11 GPIO96CounterFunction

Function Definition

BYTE GPIO96CounterFunction(BoardInfo* bi, GPIO96CTR* Ctr);

Function description

This function can be used to program any desired function into a counter, except reading which is done by GPIO96CounterRead.

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 the counter function ()

```
GPIO96CTR counter;
counter.CtrNo = 0;
counter.CtrCmd =FPGPIO96_COUNTER_CMD_RESET_ONE;
GPIO96CounterFunction(bi,&counter);
```

4.12 GPIO96PWMConfig

Function Definition

BYTE GPIO96PWMConfig(BoardInfo* bi, GPIO96PWM* GPIO96pwm);

Function description

This function configures a PWM for operation.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
GPIO96pwm	Structure with 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

```
GPIO96PWM GPIO96pwm;
GPIO96pwm.Num=0;
GPIO96pwm.Rate=100;
GPIO96pwm.Duty=50;
GPIO96pwm.Outputenable=1;
GPIO96pwm (bi,& GPIO96pwm);
```

4.13 GPIO96PWMStart

Function Definition

BYTE GPIO96PWMStart(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;
         GPIO96PWMStart (bi,Num);
```

4.14 GPIO96PWMStop

Function Definition

BYTE GPIO96PWMStop(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

Return Value

Error code or 0.

Usage Example

To stop PWM 0

```
Num=0;
GPIO96PWMStop ( bi, Num);
```

4.15 GPIO96PWMCommand

Function Definition

BYTE GPIO96PWMCommand(BoardInfo* bi, GPIO96PWM* GPIO96pwm);

Function description

This function is used to modify a PWM configuration.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
GPIO96pwm	Structure with following member variables Num - PWM number, 0-3 Command - Counter command, 0-15 (see FPGA specification for available commands) 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 can be used. To set PWM 0 to 50Hz.

```
GPIO96PWM GPIO96pwm;

        GPIO96pwm.Num=0;
        GPIO96pwm. Command= FPGPIO96_PWM_LOAD_C0_COUNTER;
        GPIO96pwm. Divisor=50000000/50;
        GPIO96PWMCommand(bi, &GPIO96pwm);
```

4.16 GPIO96UserInterruptSet

Function Definition

BYTE GPIO96UserInterruptSet(BoardInfo* bi, GPIO96USERINT* GPIO96userint);

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
GPIO96userint	Structure with following member variables IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard Function, the only valid mode is 0 Source - 0-4; 0-3 = counter/timer 0-3 output, 1 = digital input D1; (used only if mode = 0) 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()
{
GPIO96USERINT GPIO96userint;
GPIO96userint. IntFunc   = intfun;
GPIO96userint.Mode=0;
GPIO96userint.Source=0;
GPIO96userint.Enable=1;
GPIO96UserInterruptSet (bi,& GPIO96userint);
}
```

4.17 GPIO96UserInterruptStop

Function Definition

BYTE GPIO96UserInterruptStop(BoardInfo* bi, GPIO96USERINT* GPIO96userint);

Function description

This function stops user interrupts. If the interrupt mode is Before or After, the main interrupt operation may continue. If mode is Alone, the interrupt operation is stopped.

Function Parameters

Name	**Description**
BoardInfo	The handle of the board to operate on
GPIO96userint	Structure with following member variables IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard Function, the only valid mode is 0 Source - 0-4; 0-3 = counter/timer 0-3 output, 1 = digital input D1; (used only if mode = 0) Enable - 0 = disable interrupts, 1 = enable interrupts

Return Value

Error code or 0.

Usage Example

To stop user interrupt

```
GPIO96USERINT* GPIO96userint;
GPIO96userint.Enable = 0;
GPIO96UserInterruptStop(bi,& GPIO96userint);
```

4.18 GPIO96InitBoard

Function Definition

BYTE GPIO96InitBoard(DSCCB* dsccb, GPIO96INIT Init);

Function description

This function Initialize the board.

Function Parameters

Name	Description
dsccb	The handle of the board to operate on
Init	GPIO96INIT structure variable

Return Value

Error or 0.

Usage Example

```
GPIO96INIT init;
dsccbp.boardtype = DSC_FPGPIO96;
GPIO96InitBoard (&dsccbp, init )
```

4.19 GPIO96FreeBoard

Function Definition

BYTE GPIO96FreeBoard(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

```
GPIO96FreeBoard(board);
```

4.20 GPIO96LED

Function Definition

BYTE GPIO96LED(BoardInfo* bi, int State);

Function description

This function turns the blue 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,

```
GPIO96LED(bi,1);
```

5.0 Universal Driver Demo Application Description and Usage Instructions

The Universal driver supports following applications for FP-GPIO96 board,

- DIO
- Counter
- PWM
- User Interrupt

5.1 DIO Application

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 DIO operation need to be performed. E.g. configure DIO port in output mode for output byte.

Output Byte:

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

E.g.: To set PORT 0 all the pins are 5V except pin 3 and pin 7. Run DIO application and provide input as follows

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

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

Input Byte:

- Select configure option from main menu
- Enter port number
- Configure the port in input mode:
- Exit from current menu
- Select input byte option from main menu
- Enter port number
- Reads and displays current voltage levels on the screen

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

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

If the application shows 0x01 on the screen, the application works as expected.

5.2 Counter Application

Description

Generally the Counter is used as rate generator, Also the counter can be configured in count up or count down direction.

Usage instructions

This application gets input from user as follows

- Enter counter number (0-1)
- Enter output frequency(1-50MHZ)
- Waits for key press
- If any key is pressed , application Stops rate generator

E.g.: To generate 100 Hz rate generator using counter 0. Run Counter application and provide input as follows

- Enter counter number (0-1):0
- Enter output frequency(1-50MHZ):100

Place Oscilloscope probe on counter 0 output pin (PORTC0 is the output pin for Counter 0) and set voltage division to 1V range and second division to 1ms. If it shows a periodic pulse with 100 Hz frequency, the application generated expected rate.

- Press any key and automatically application stops counter output.

5.3 PWM Application

Description

This application generates PWM signal. Pulse Width Modulation (PWM) Signal is a method for generating an analog signal using a digital source. A PWM signal consists of two main components that define 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. 1000 Hz 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 user and generates PWM signal.

Usage instructions

This application gets input from 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 PWM output.

E.g.: To generate 100 Hz PWM waveform with duty cycle 50 % on PWM channel 0. Run 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 Oscilloscope probe on PWM channel 0 output pin (PORTF0 is the output pin for PWM 0) and set voltage division to 1V range and second division to 1ms. If it shows a PWM wave form 50 % duty cycle and 100 Hz frequency, the application generated expected rate.

- Press any key and automatically application stops PWM output

5.4 User Interrupt Function

Description

The User Interrupt feature of the Universal Driver enables you to run your 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 you want to run your code at regular fixed intervals. The Universal Driver includes example programs for each board with user interrupt capability to illustrate how to use this feature. This application gets interrupt frequency as user input and calls user function periodically at rate of interrupt frequency.

Usage instructions

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.

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

- Enter clock source (0-3=Counter0-3 output 4= DIO):0
- Enter interrupt rate (1-50MHZ) :100

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

```
UserInt count value =0
UserInt count value =100
UserInt count value =200
UserInt count value =300
```

Finally Press any key to cancel the interrupt.

6.0 Common Task Reference

6.1 Data Acquisition Feature Overview

I/O Connectors

The FP-GPIO96 uses the FeaturePak standard, which provides a card form-factor and a connector suitable for high-density and compact applications.

The FP-GPIO96 uses and MXM 230-pin connector for all I/O, both the main computer and for user I/O connections. The board contains gold-plated fingers conforming to the MXM physical standard for interfacing to the MXM connector.

The user I/O consists of 100 signals organized as 2 50-pin groups. On a general purpose baseboard, these signals directly drive 2 50-pin I/O connectors. The 50-pin connectors may then be connected to application-specific cabling, or an I/O adapter board may be plugged onto them to provide a custom I/O connector configuration for the system enclosure. A custom or application-specific main board may elect to use these signals in any way desired without conflicting with the standard.

Digital I/O signals

There are 96 digital I/O signals, grouped into 12 8-bit ports as DIOA0-DIOA7, DIOB0-DIOB7, DIOC0-DIOC7, DIOD0-DIOD7, DIOE0-DIOE7, DIOF0-DIOF7, DIOG0-DIOG7, DIOH0-DIOH7, DIOJ0-DIOJ7, DIOK0-DIOK7 DIOL0-DIOL7 and DIOM0-DIOM7. Signals DIOA-DIOF are on the primary I/O connector pins 1-48, and signals DIOG-DIOM are on the secondary I/O connector pins 1-48. Digital I/O signals use 3.3V signaling only. Ports A, B, D and G-M are 8-bit ports with direction programmable byte by byte. Port C is divided into two 4-bit nibbles with independent direction control. Ports E-F are 8-bits ports with direction programmable bit by bit. 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 DIOA-DIOF, and Group B includes I/O signals DIOG-DIOM. All signals within each group have the same pull direction. The as-shipped configuration for FP-GPIO96 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 an 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 an external clock or output, the associated digital I/O line is taken over for the counter/timer and its direction is set as needed to support the selected function. 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:

Primary I/O connector	DIO Bit	Counter/Time r
17	DIOC0	0
18	DIOC1	1
19	DIOC2	2
20	DIOC3	3
21	DIOC4	4
22	DIOC5	5
23	DIOC6	6
24	DIOC7	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 the 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:

Primary I/O connector	DIO Bit	PWM
41	DIOF0	0
42	DIOF1	1
43	DIOF2	2
44	DIOF3	3

6.2 Data Acquisition Software Task Reference

This section describes the various data acquisition tasks that may be performed with the FP-GPIO96 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

<u>Program Entry/Exit sequence</u>

1. All driver usage begins with the function GPIO96InitBoard() This function must be called prior to any other function involving the GPIO96.
2. At the termination of the program the programmer may use GPIO96FreeBoard(), but this is not required. This function is normally used in a development environment where the program is being repeatedly modified and rerun.

<u>Digital I/O operations</u>

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

BYTE GPIO96DIOConfig() configures a single bit for input or output.
BYTE GPIO96DIOConfigAll() configures all 21 bits at once.

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

BYTE GPIO96DIOOutputByte() outputs 8 bits of data
BYTE GPIO96DIOInputByte(BoardInfo* bi, int Port, byte* Data);
BYTE GPIO96DIOOutputBit(BoardInfo* bi, int Bit, int Value);
BYTE GPIO96DIOInputBit(BoardInfo* bi, int Bit, int* Value);

To configure the digital I/O pull-up/down resistors, use GPIO96DIOConfig(). 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.

<u>Counter/timer operations</u>

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 GPIO96CounterSetRate(). 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 GPIO96CounterConfig() 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 (but not both). This function is typically used to count external events.
2. Use GPIO96CounterRead() 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.

Detailed Programming

To program a counter/timer using individual commands, use GPIO96CounterFunction(). 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.

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.

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.
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 GPIO96CounterRead() function.

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. GPIO96PWMConfig() configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. GPIO96PWMStart() can be used to start the PWM running if the config function did not start it.

To stop a PWM:

GPIO96PWMStop() 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 GPIO96PWMStart().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use GPIO96PWMCommand(). 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) or it can run before or after the driver's built-in interrupt function (Before and After modes). The available modes depend on the source of the interrupt:

Source	Source no.	Modes supported
A/D interrupts	0	Not supported on FP-GPIO96
D/A interrupts	1	Not supported on FP-GPIO96
Counter/timer interrupts	2, 3	0 Alone
Digital input interrupts	4	0 Alone

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

Configure:

GPIO96UserInterruptSet() selects the source for the user interrupts and also installs a pointer to the user's code to run when the interrupt occurs. If user interrupts are being run in Before or After mode, this function must be called before the function that initiates the standard interrupt function.

Run (alone mode):

1. If a counter/timer is being used to drive interrupts, then configure it with GPIO96CounterSetRate().
2. If a digital input is being used to drive interrupts, it is configured with GPIO96UserInterruptSet ().
3. Use GPIO96UserInterruptSet () to start the interrupt operation.

Stop (alone mode):

Use GPIO96UserInterruptStop() to stop user interrupts.

LED control

The FP-GPIO96 contains a blue 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 GPIO96LED().

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, you provide a pointer to the storage variable and indicate the port number and bit number if relevant. To perform digital output, you provide the output value and the output port and bit number, if relevant.

The five Universal Driver functions described here are GPIO96DIOConfig, GPIO96DIOOutputByte, GPIO96DIOInputByte, GPIO96DIOOutputBit and GPIO96DIOInputBit.

Step-By-Step Instructions:

If you are performing digital input, 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 the port is set to the required direction, input or output. Use function GPIO96DIOConfig() to set the direction if necessary.

Call the selected digital I/O function. Pass it a int port value, and either a pointer to or a constant byte digital value. If you are performing bit operations, then you will also need to pass in a int value telling the driver which particular bit (0-7) of the DIO port you wish to operate on.

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 = GPIO96DIOConfig(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 = GPIO96DIOInputBit(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 = GPIO96DIOInputByte(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. GPIO96DIOOutputBit(), set bit 7 of port 1 to 1, leave other bits alone */
if ((result = GPIO96DIOOutputBit(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 = GPIO96DIOOutputByte(dscb, port, 0x55)) != DE_NONE)
    return result;
```

6.4 Performing PWM operation

Description:

The PWM operation generates PWM signal with desired frequency and duty cycle value. The following functions are used for PWM operation GPIO96PWMConfig() and GPIO96PWMStop().

Step-By-Step Instructions:

Create GPIO96PWM structure variable to hold PWM settings and initialize PWM structure variables , then call GPIO96PWMConfig() to configure the PWM and finally call GPIO96PWMStop() Application to stop the running PWM signal .

Example of Usage for PWM Operations:

```
        GPIO96PWM pwm ; //structure to hold OWM 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(GPIO96PWMConfig(bi,&pwm) !=DE_NONE)
        {
                dscGetLastError(&errorParams);
                printf("GPIO96PWMConfig   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(GPIO96PWMStop(bi,pwm.Num) !=DE_NONE)
        {
        dscGetLastError(&errorParams);
        printf("GPIO96PWMStop      error:      %s      %s\n",      dscGetErrorString(errorParams.ErrCode),
errorParams.errstring );
        return 0;
        }
```

6.5 Performing Counter Operation

Description:

The counter operation makes the counter to run in specified rate. The counter application uses following Universal driver function GPIO96CounterSetRate() and GPIO96CounterFunction().

Step-By-Step Instructions:

Create GPIO96CTR structure variable to hold Counter settings and initialize counter structure variables, then call GPIO96CounterSetRate () to configure the counter and finally call GPIO96CounterFunction() to stop the running counter.

Example of Usage for Counter Operations:

```
GPIO96CTR counter; //Structure to hold counter settings
Int CtrNo =0 //Select counter Number
Float rate =1000 //Select counter rate
//The following function configures the counter with desired rate
        if(GPIO96CounterSetRate(bi,counter.CtrNo,Rate) !=DE_NONE)
        {
                dscGetLastError(&errorParams);
                printf("GPIO96CounterConfig error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                errorParams.errstring );
                return 0;
        }
        printf("Press any key to stop counting \n");
        while( !kbhit())
        {
        }
        counter.CtrCmd = GPIO96_COUNTER_CMD_RESET_ONE; //Reset the counter
        if(GPIO96CounterFunction(bi,&counter) !=DE_NONE)
        {
                dscGetLastError(&errorParams);
                printf("GPIO96CounterFunction error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                errorParams.errstring);
                return 0;
        }
```

6.6 Performing User Interrupt Operations

Description:

The User Interrupt application configure a counter to generate desired interrupt rate and at the same interrupt rate, a registered user interrupt function is called .To do User Interrupt application following Universal driver function are used.

Step-By-Step Instructions

Create GPIO96USERINT structure variable to hold Interrupt settings and initialize Interrupt structure variables, then call GPIO96UserInterruptSet () to configure the counter and finally call GPIO96UserInterruptStop () to stop the running interrupt with Enable variable is equal to zero.

Example of Usage for User Interrupt Operations:

```
    int count =0;
    void userfun (void *param)
    {
      count ++;
    }
    Void main()
    {

        GPIO96USERINT inter; // structure to hold Interrupt settings
         inter.IntFunc=userfun;// Set user function as function pointer
         inter.Mode=0; // only mode 0is supported by GPIO96
         inter.Enable=1; // Enable interrupt
         inter.Source=0;   // Select interrupt source 0-Counter o input 1=DIO input
    //The following function configures user Interrupt.
    if(GPIO96UserInterruptSet(bi,&inter) !=DE_NONE)
                {
                        dscGetLastError ( &errorParams );
                        printf ( " GPIO96UserInterruptSet error: %s %s\n", dscGetErrorString (
errorParams.ErrCode ), errorParams.errstring );
                        return 0;
    }
  Float rate =1000//Select Interrupt source frequency rate
                //Start counter with desired rate if counter is chosen
                if(GPIO96CounterSetRate(bi,0,rate) !=DE_NONE)
                {
                        dscGetLastError ( &errorParams );
                        printf ( " GPIO96CounterSetRate error: %s %s\n", dscGetErrorString (
errorParams.ErrCode ), errorParams.errstring );
                        return 0;
                }

                printf("Press any key to terminate the application \n");
                while( !kbhit())
                {
                        printf("Count value %d \n ",count);
                        dscSleep(1000);
                }
```

```
                    inter.Enable=0; // Disable interrupt
                    if(GPIO96UserInterruptSet(bi,&inter) !=DE_NONE)
                    {
                    dscGetLastError ( &errorParams );
                    printf ( " GPIO96UserInterruptSet error: %s %s\n", dscGetErrorString ( errorParams.ErrCode ),
errorParams.errstring );
                    return 0;
                     }

}
```

Appendix: Reference Information

Counter/timer commands

The counter/timers are programmed with a series of commands. Each command is a 4 bit value. The commands may have an associated option. A series of commands are 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 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 the count direction, up or down

3 Enable/disable external gate.

4 Enable/disable counting

5 Latch the selected counter. The 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 GPIO96CounterRead() function.

6 Select the counter clock source according to the table below:

Option	Clock source
0	External input pin, active low
1	Reserved
2	Internal clock 50MHz
3	Internal clock 1MHz

If an external DIO pin is selected as the counter input, the DIO pin's direction is automatically set for the input mode. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail.

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 1111, 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.

Primary I/O connector	DIO Bit	Counter/Timer
17	DIOC0	0
18	DIOC1	1
19	DIOC2	2
20	DIOC3	3
21	DIOC4	4
22	DIOC5	5
23	DIOC6	6
24	DIOC7	7

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 the selected pulse width, the output will go high. When the counter reaches 1, it 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. A command may have an associated parameter, referred to as PWMCD in the descriptions below. A series of commands is required to configure a PWM for operation. See the PWM usage instructions in the manual for more information.

0 Stop all PWMs / selected PWM
  0 = stop all PWMs (opposite polarity for "all" compared to other PWM commands)
  1 = stop single PWM

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 (period counter) or C1 (duty cycle counter)
  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
  0 = disable pulse output; output = opposite of polarity setting from command 0010
  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 a PWM
  0 = 50MHz
  1 = 5MHz

7 Start all PWMs / selected PWM
  0 = start PWM selected with PWMCD
  1 = 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).

Primary I/O connector	**DIO Bit**	**PWM**
41	DIOF0	0
42	DIOF1	1
43	DIOF2	2
44	DIOF3	3