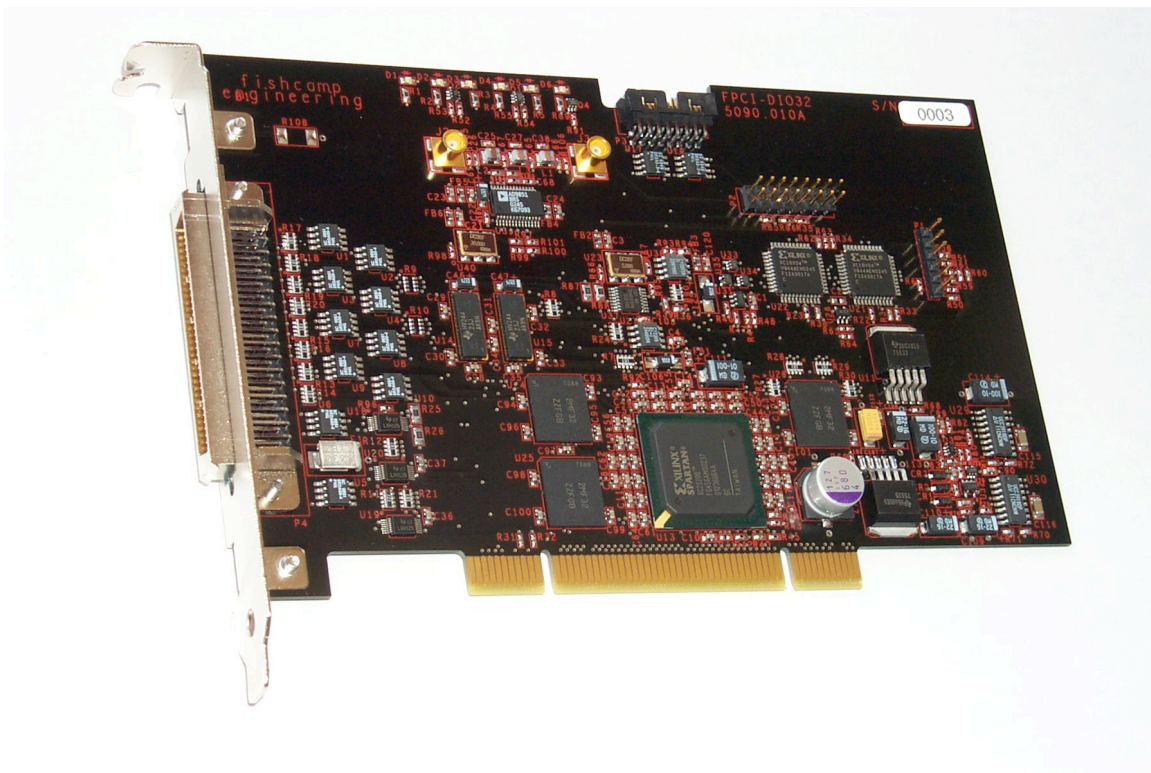




**fishcamp engineering**

## **FPCI-DIO32 - Digital Input/Output Card for PCI bus**

---





### **Limited Warranty**

The FPCI-DIO32 interface hardware is warranted to be free from defects in materials and workmanship for a period of one year from date of shipment from fishcamp engineering. Defects caused by misuse, abuse, or shipment are not covered.

Defective equipment that is subject to this limited warranty will be repaired or replaced at the option of fishcamp engineering if we are notified during the warranty period. The customer must obtain a Return Material Authorization (RMA) number before returning any equipment. Shipping costs from fishcamp engineering will be paid by fishcamp engineering. Equipment should be packaged in the original shipping container if possible, and the RMA number must be clearly marked on the outside of the package.

The information provided in this manual is believed to be correct, however fishcamp engineering assumes no responsibility for errors contained within. The software programs are provided "as is" without warranty of any kind, either expressed or implied.

No other warranty is expressed or implied. Fishcamp engineering shall not be liable or responsible for any kind of damages, including direct, indirect, special, incidental, or consequential damages, arising or resulting from its products, the use of its products, or the modification to its products. The warranty set forth above is exclusive and in lieu of all others, oral or written, express or implied.

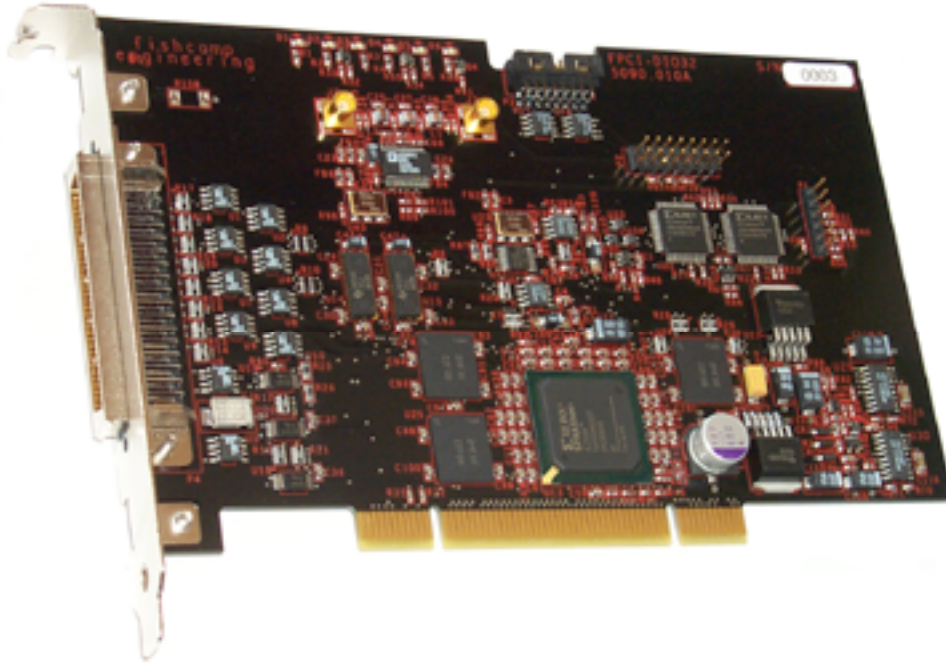
The information covered in this manual is subject to change without notice.

# Contents

<b>1</b>	<b>CHAPTER 1 - INTRODUCTION .....</b>	<b>6</b>
<b>2</b>	<b>CHAPTER 2 - INSTALLATION.....</b>	<b>7</b>
2.1	HARDWARE INSTALLATION.....	7
2.2	SOFTWARE INSTALLATION .....	7
2.3	DRIVER INSTALLATION.....	7
2.4	FCFW FRAMEWORK INSTALLATION.....	8
2.5	FPCIDIO32EXPLORER INSTALLATION.....	8
2.6	INSTALLATION VERIFICATION .....	8
<b>3</b>	<b>FPCIDIO32EXPLORER .....</b>	<b>9</b>
3.1	FPCI-DIO CARD TESTS.....	10
3.2	MASTER CLOCK FREQUENCY .....	11
3.3	CARD LEVEL REGISTER ACCESS .....	11
3.4	MODE0 REGISTERS .....	12
3.5	MODE1 REGISTER .....	13
3.6	PATTERN RECOGNITION REGISTERS .....	13
3.7	MODE1 WAVEFORM MANIPULATION.....	14
3.8	WAVEFORM EDITOR .....	15
3.8.1	Waveform Definition Edit Buttons.....	16
3.8.2	Clear Waveform .....	16
3.8.3	Add Wave Segment .....	16
3.8.4	Add Pattern Data Button .....	17
3.8.5	Delete Wave Segment, Delete Pattern Data.....	18
3.9	UPLOAD TO FPCI-DIO32 .....	18
3.10	CARD WAVEFORM DUMP WINDOW .....	19
<b>4</b>	<b>FPCI-DIO32 FUNCTION DESCRIPTION .....</b>	<b>20</b>
4.1	GENERAL.....	20
4.2	MODE 0 OPERATION .....	20
4.3	MODE1 OPERATION .....	21
<b>5</b>	<b>REGISTER LEVEL PROGRAMMING .....</b>	<b>23</b>
5.1	0x0200 0000 C_BK2_DATA REGISTER .....	23
5.2	0x0200 0004 C_DDS_FREQ REGISTER .....	23
5.3	0x0200 0008 C_DDS_CNTRL REGISTER.....	24
5.4	0x0200 000C C_MASTER_CNTRL REGISTER .....	24
5.4.1	c_master_cntrl Bit31 – Fuse map loaded status.....	25
5.4.2	c_master_cntrl Bits 30:9.....	25
5.4.3	c_master_cntrl Bit 8 - Invert I/O Mclk for input latch .....	25
5.4.4	c_master_cntrl Bits 7:6 - Operation Mode .....	25
5.4.5	c_master_cntrl Bit 5 - PRGMRS_LED.....	25
5.4.6	c_master_cntrl bit 4 - BUF_IO_CLK_OUT_EN .....	26
5.4.7	c_master_cntrl bit 3 - IO_CLK_EN4 .....	26
5.4.8	c_master_cntrl bit 2 - IO_CLK_EN3 .....	26
5.4.9	c_master_cntrl bit 1:0 - IO_CLK_EN2, IO_CLK_EN1 .....	26
5.5	0x0200 0010 C_MODE0_DATA REGISTER .....	27
5.6	0x0200 0014 C_MODE0_CNTRL REGISTER.....	27
5.7	0x0200 0018 C_MODE1_CNTRL REGISTER.....	29
5.8	0x0200 001C C_MATCH_PATTERN REGISTER.....	31
5.9	0x0200 0020 C_PATTERN_MASK REGISTER.....	31
<b>6</b>	<b>COOKBOOK.....</b>	<b>32</b>

6.1	FIRST EXAMPLE .....	32
6.2	SECOND EXAMPLE .....	34
<b>7</b>	<b>MODE 1 OPERATION.....</b>	<b>37</b>
<b>8</b>	<b>HARDWARE.....</b>	<b>39</b>
8.1	CUSTOMER I/O CONNECTOR.....	39
8.2	MATING CONNECTOR .....	39
8.3	CUSTOMER I/O SIGNAL DEFINITION .....	41
<b>1</b>		

## Chapter 1 - Introduction



The FPCI-DIO32 card is a digital input/output card for PCI bus computer platforms. The card is suitable for high-speed, dynamic I/O applications where data transfer operations are performed via DMA operations to and from on board local memory. A unique feature of the card is its ability to control, on-the-fly, the signal lines' data direction and tri-state drive thus allowing applications to simulate complex uP bus structures. The card is useful in circuit card assembly and semiconductor device characterization applications in the R&D lab as well as manufacturing test environments.

- 32 digital input/output signals with 4 handshake signals.
- Independent data direction control on each nibble (4 bit) lane. On-the-fly changes.
- Capability to emulate bi-directional and tri-state signal busses.
- Allows simultaneous pattern generation and data capture operations on signal lines.
- Pattern detection logic
- DDS clock synthesizer with milli-hertz frequency resolution.
- Expansion connector allows locking multiple cards together for wider data words.
- 68 Pin AMPLIMITE .050 Series connector for customer I/O.
- 4.25" X 7" card fully compliant with PCI Revision 2.2.
- Universal card operates in 3.3V/5V PCI signaling environments.
- On-board 16 MBytes of pattern memory.
- LV-TTL logic levels (32mA drive) on all user I/O signal lines (5V TTL Tolerant).
- Includes driver software and complete manual with example application software.
- One year warranty.

## 2 Chapter 2 - Installation

### 2.1 Hardware Installation

The card installation procedure you follow depends upon which model of computer you have. For detailed installation instructions, please refer to the manual that came with your computer. It should describe the installation of PCI bus expansion cards.

The card may be installed into any 32 bit PCI slot. The FPCI-DIO32 card is a ‘universal’ card in that it supports both 5V and 3.3V signaling environments over the PCI bus. The edge connector of the FPCI-DIO32 card is keyed to prevent any installation errors. It does not support the 64 Bit option in the PCI specification.

The FPCI-DIO32 card may be plugged into any available PCI slot in the computer. The accompanying software does not assume any particular slot. There are no switches nor jumpers to set on the card.

**WARNING** When handling the FPCI-DIO32 card, hold the card by its edges to avoid touching any of the integrated circuits or the connector that plugs into the slot on the main logic board of your computer. Make sure the power to the computer is off before installation.

### 2.2 Software Installation

The FPCI-DIO32 card comes with three software modules. The first one is the driver for MacOS X and is a required installation.

The second module is the ‘fcFw Framework’. This module greatly simplifies the writing of MacOS X applications written in ObjectiveC. This is an optional installation but is recommended.

The last module is a MacOS X application called FpciDio32Explorer that allows for testing and interactive operation of the FPCI-DIO32 card without the need to write any custom software. This is an optional installation but is recommended.

### 2.3 Driver Installation

The FPCI-DIO32 card will require the installation of its driver software on the computer before any application software can access the card. On the installation disk, that comes with the card, there should be an installer application named ‘fpciDio32.pkg’. Alternatively, you can get the latest driver from the fishcamp engineering website at <http://www.fishcamp.com>. The card driver is only compatible with MacOS X version 10.2 and later. Updates will be posted to the company’s support website.

Double click the driver installer 'fpciDio32.pkg'. This will install the FPCI-DIO32 driver. The file 'fpciDio32x.kext' will be installed in the /System/Library/Extensions directory on the host computer. You will need to reboot the computer before using the card.

## **2.4 FcFw Framework Installation**

The fcFw Framework is a MacOS X Objective C framework that greatly simplifies the programming of applications software for the FPCI-DIO32 card. The FpciDio32Explorer application was written using this Framework.

On the installation disk, that comes with the card, there should be an installer application named 'fcFwFramework.pkg'. Alternatively, you can get the latest version of the Framework from the fishcamp engineering website at <http://www.fishcamp.com>. The Framework is only compatible with MacOS X version 10.2 and later. Updates will be posted to the company's support website.

Double click the driver installer 'fcFwFramework.pkg'. This will install the fcFw.framework. The directory 'fcFw.framework' will be installed in the /Library/Frameworks directory on the host computer.

## **2.5 FpciDio32Explorer Installation**

The FpciDio32Explorer module is a standalone MacOS X application that will allow testing and interactive operation of the FPCI-DIO32 card.

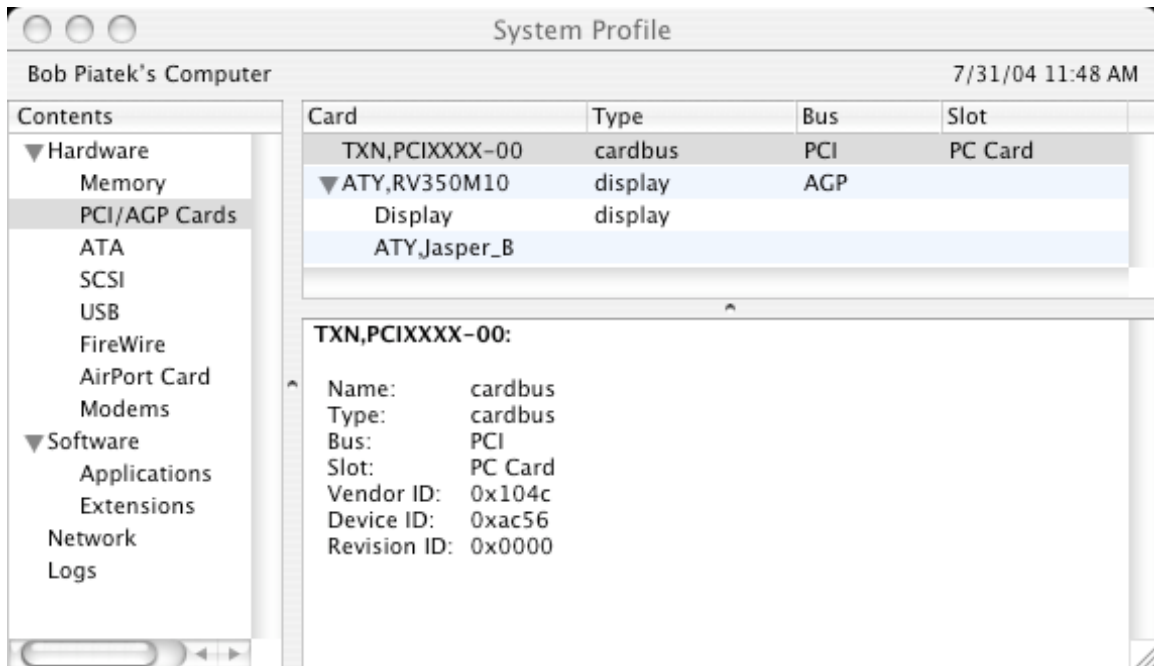
To install, simply copy the application to any convenient location on your hard disk.

## **2.6 Installation verification**

You can verify that the hardware and software installation was successful by a couple of methods. First of all, you can use Apple's 'System Profiler' application. It should be located in the /Applications/Utilities folder on your hard drive.

Double click the application and then click on the 'PCI/AGP' heading under 'Hardware' in the main window.





\*\*\*\*\*Describe what the user should see here \*\*\*\*\*

The second method that can be used to verify the installation, is to run the FpciDio32Explorer program. Run the application by double-clicking its icon in the finder.

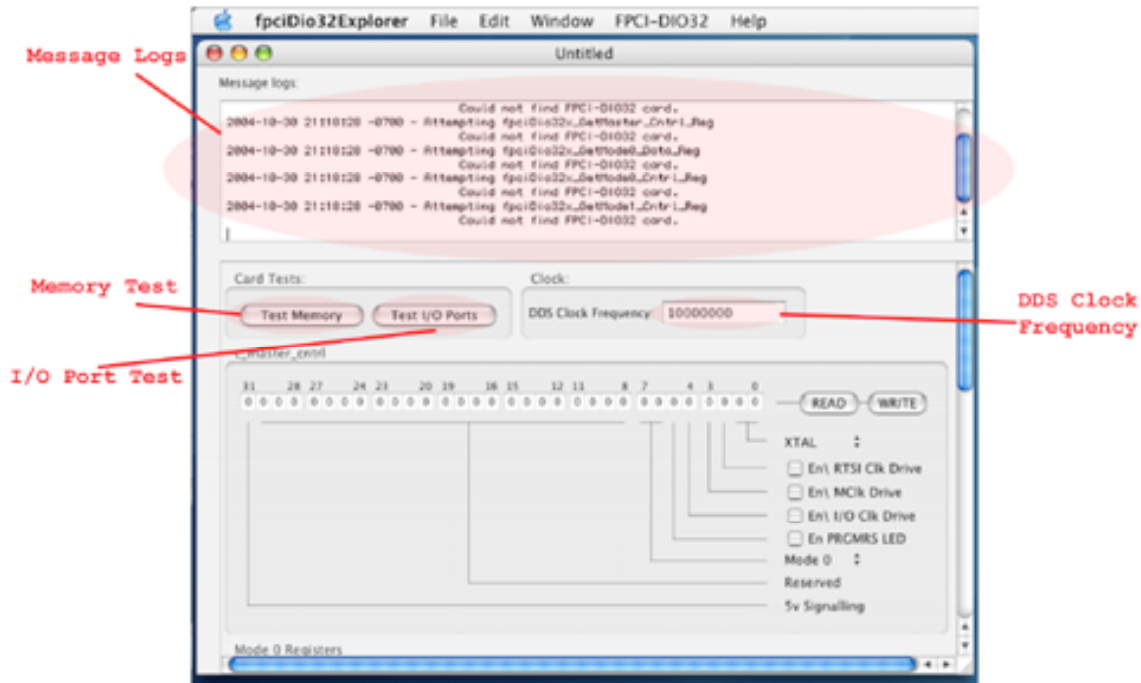


### 3 FpciDio32Explorer

The FpciDio32Explorer is an application that serves three purposes. First it allows the user to verify proper operation of the card's hardware and software driver. Secondly, it allows interactive operation of the card's features. This can be helpful to the user when trying to familiarize oneself with the operation of the FPCI-DIO32 card. Lastly, the application has a waveform editor that allows the manual creation of waveform definitions that can be used during mode1 operation of the card.

### 3.1 FPCI-DIO card tests

When the FpciDio32explorer application is first launched, the following window is displayed to the user:



The top area of the window is a scrolling text area that displays a running log of the operations performed on the FPCI-DIO32 card. The log entries are time-stamped and only the most recent 100 entries are displayed. After 100 entries are entered in the log, the oldest entry will be deleted whenever a new log entry is made.

When the application is first launched, it will search for an FPCI-DIO32 card installed in the computer. It will open the driver for the first instance of the card found. While this operation is executing, the program will log various information about how the operation went. If there was an error in trying to load the software driver for the card, or a card was not found, this will be evident from the log entries.

Below the log are a couple of UI push buttons that allow the user to run some tests on the FPCI-DIO32 card. The first button allows you to test the on card memory of the FPCI-DIO32. Pushing the button will initiate a test whereby a pattern is written into all available memory locations. Afterwards, the memory array is read back and compared with the original written pattern. Any errors are logged.

The second test button allows you to test the operation of the customer I/O data paths from the card. This test writes a series of data patterns to the I/O port on the card and

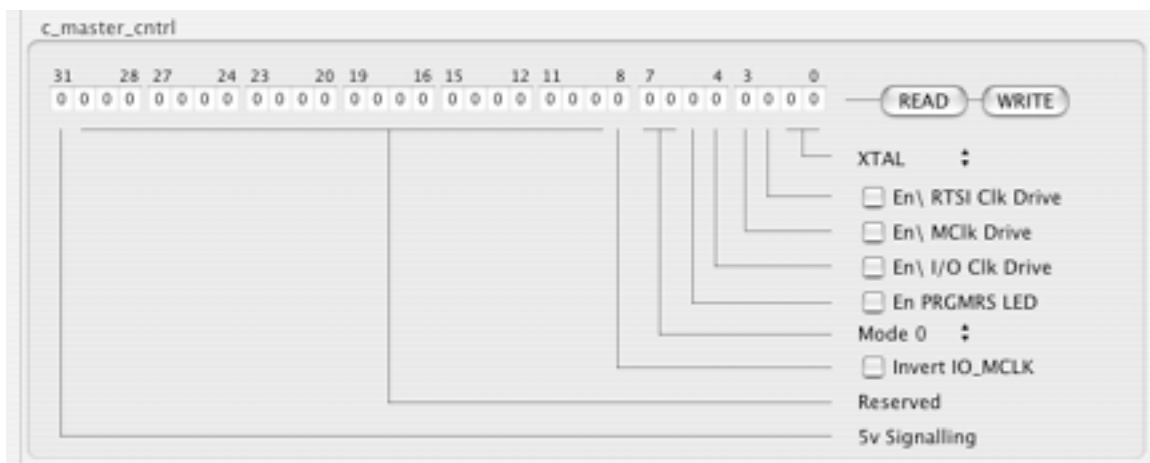
then reads the results back. Any errors are again logged in the message log area. It should be noted that during the course of this test, there should not be anything connected to the I/O connector on the FPCI-DIO32 card. Since the test writes some data patterns to the port, any user equipment that was connected to the card might behave in unpredictable ways. It could also interfere with successful completion of the port tests. A warning to this effect is displayed to the user before the tests are run. The results of the port test is logged in the message log.

### 3.2 Master Clock Frequency

To the right of the test buttons is a text field, which will allow you to enter a new value for the DDC clock generator on the card. The value is specified in Hz. Valid entries will be between 1Hz and 20 MHz. The clock generator on the card is set to the desired frequency as soon as the entry is made. You must hit the <enter> or <return> key to complete the entry of a new frequency.

### 3.3 Card Level Register Access

The main FpciDio32Explorer window can be scrolled down to reveal additional information about the card's registers that are accessible via software I/O calls. This UI to the registers allow interactive exploration of functions controlled by bit fields in these registers.

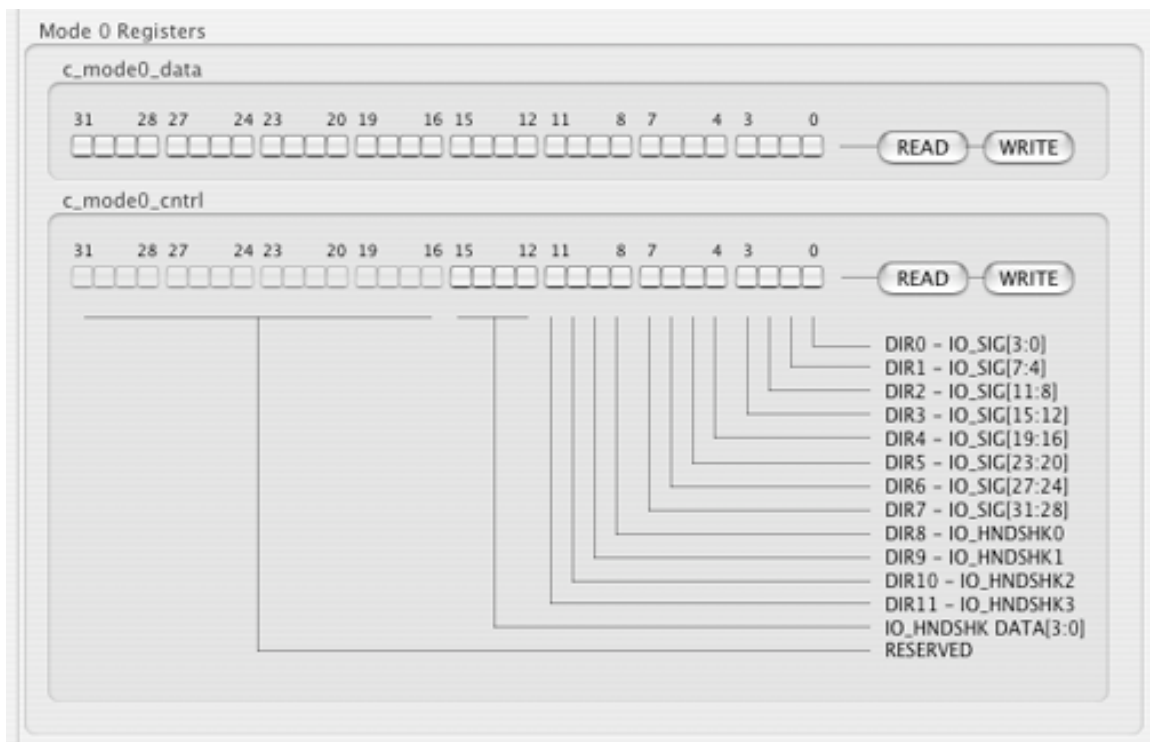


The first register displayed is the c\_master\_cntrl register on the card. For a detailed description of all of the bit fields in this register refer to section x.x. The user may alter the various bit fields of this register by means of the GUI controls on the right hand section of this window area. Manipulating the UI controls does not immediately affect the contents of the register on the card. The user must push the 'WRITE' button in order for changes to be made to the register. Pressing the 'READ' button will update the displayed bit fields for this register to reflect the current setting on the card.

Any 'READ' or 'WRITE' operations commanded to this register will be logged to the message log area of the main window.

### 3.4 Mode0 registers

The next register displayed is that of the registers associated with mode0 operation of the FPCI-DIO32 card.. Just like the previous register, there are UI elements that allow the user to manipulate the various bit fields of the mode0 registers. The following figure shows the mode0 register display of the program.

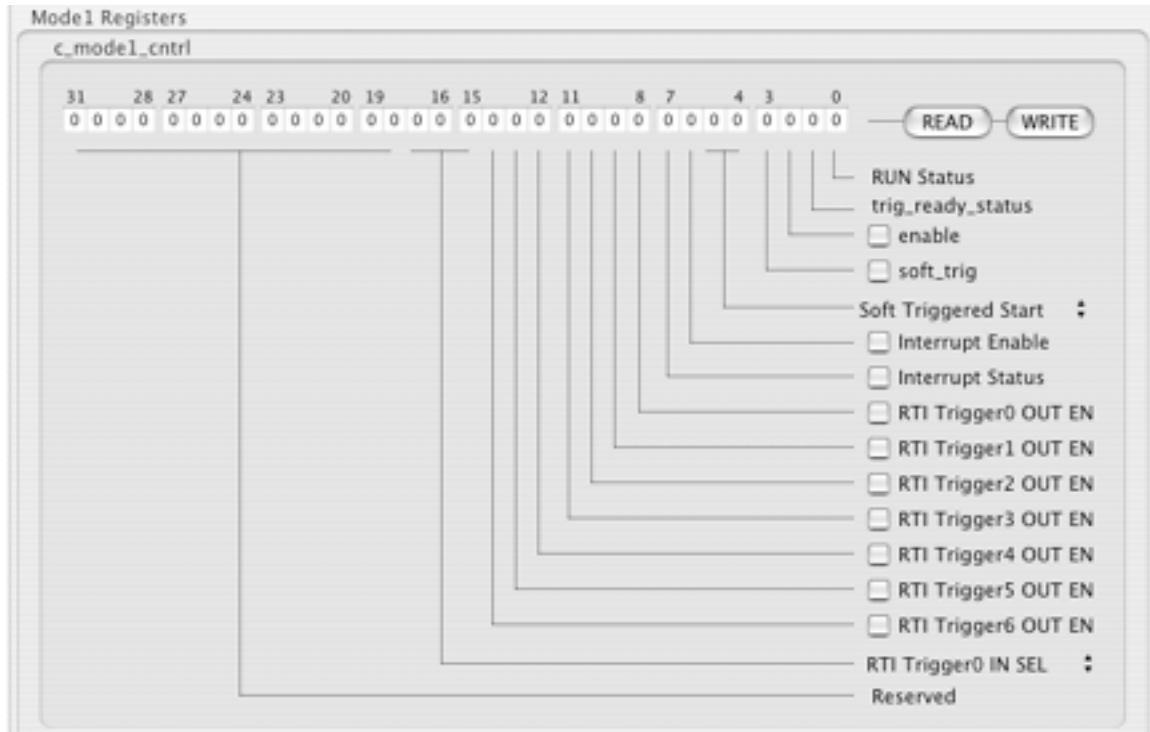


In this window section, there are actually two registers presented to the user. The first one is the display associated with the c\_mode0\_data register. Below that is presented the display of the c\_mode0\_cntrl register. Each register can be read or written to the FPCI-DIO32 independently of the other.

The user interface for these registers allows the user to manually input and output data to the customer I/O connector of the FPCI-DIO32 card. For instance, the user can set the direction bits in the c\_mode0\_cntrl register to the 'OUTPUT' direction and then set a data pattern using the c\_mode0\_data register. The selected data pattern should then be measurable at the pins of the I/O connector on the card. This can be useful, for instance, during initial troubleshooting of the cabling to the user's connected equipment.

### 3.5 Model register

The next register displayed in the window is the `c_model_cntrl` register. As with the other registers, you can read the current setting of this register from the card by pressing the 'READ' button. You can also set the various bits in the registers and write the new value to the card by pressing the "WRITE" button.



This register is used to control the model operation of the FPCI-DIO32 card. This mode is used when generating digital patterns and will need a proper waveform definition setup in the card's memory for proper operation. See the section later on that details the waveform edition of this program. Also, note that 'model' must be selected in the `c_master_cntrl` register for this register to be active.

### 3.6 Pattern recognition registers

The last registers displayed in the window are the two registers associated with the pattern recognition capability on the FPCI-DIO32 card. As with the other registers, you can read the current setting of this register from the card by pressing the 'READ' button. You can also set the various bits in the registers and write the new value to the card by pressing the "WRITE" button.



The pattern recognition logic is used by the FPCI-DIO32 card to look for user specified bit patterns on the customer I/O port. The detection of a pattern can be used to start the generation of a waveform sequence during model operation of the card.

The user has the ability to select a desired state for individual bits as well as define which bits are to be considered for the match. The `c_match_pattern` register will define the state of the bits that will constitute a valid match. The `c_mask_pattern` can be used to define which bits are to be included in the evaluation of the pattern. For each individual bit on the customer I/O port for the card, a valid match will occur when the bit's 'mask' is set to a '1' and the bit state is the same as the state as defined for the bit in the `c_match_pattern` register. If the bit's 'mask' value is '0' then that bit will not be included in the pattern evaluation. A valid match from the match pattern logic will occur when the above conditions are true for all of the bits in the I/O port.

### 3.7 Model waveform manipulation

The FPCI-DIO32 card allows complex waveforms to be generated as well as capturing external data much like a general purpose logic analyzer would. In order for the card to operate in this manner, you must first define the memory commands that define the waveform in the card's memory array. Once the waveform structure has been defined, the card will operate autonomously by reading and writing waveform signal states to and from the memory and the customer I/O connector. Transfer of the memory words is accomplished via a DMA controller on the card.

Waveform descriptions are best done programmatically, however the `FpciDio32Explorer` program provides a simple waveform editor for those applications not requiring complex waveforms or when the user is just trying to get familiar with model operation of the card. In this case, interactive editing and playing back of waveforms is possible via the keyboard and mouse of the host system. Captured waveform data may also be viewed.

A brief overview of the waveform definition structure is best at this time. The FPCI-DIO32 card allows very complex waveforms to be constructed in its program memory. A key concept to understand is that of a waveform segment. A waveform segment is a small portion of the final waveform being generated. Each waveform segment defines the data pattern words to be generated on a per clock basis. Waveform segments are limited in length to 252 data pattern words by the FPCI-DIO32 hardware. A waveform

segment need not use up the entire 252 pattern words associated with it. Any number of pattern words from 1 up to the maximum of 252 may be used. When the card is playing back a waveform, it reads each pattern word in the segment, in order, and outputs them to the customer I/O connector. One word is transferred on each clock cycle of the currently selected I/O port clock. In addition, each waveform segment may be repeated, or looped, from 1 to 65535 times before jumping to the next waveform segment.

The overall waveform being generated is constructed by chaining many waveform segments together in a list. Each waveform segment is played, in sequence until the end of the list is reached. The waveform segments are linked to each other by means of a pointer defined in the header of the waveform segment. The pointer points to the next waveform segment to execute after the current one is finished executing. The user is free to define the pointer such that it points to another waveform segment descriptor, or it can even point back to current waveform segment. The latter case is used when an infinite loop is desired. The next waveform segment can be any of the previously defined segments in the list. So for instance, the last segment can point back to the first segment in the list such that a waveform is repeatedly played over and over again once started.

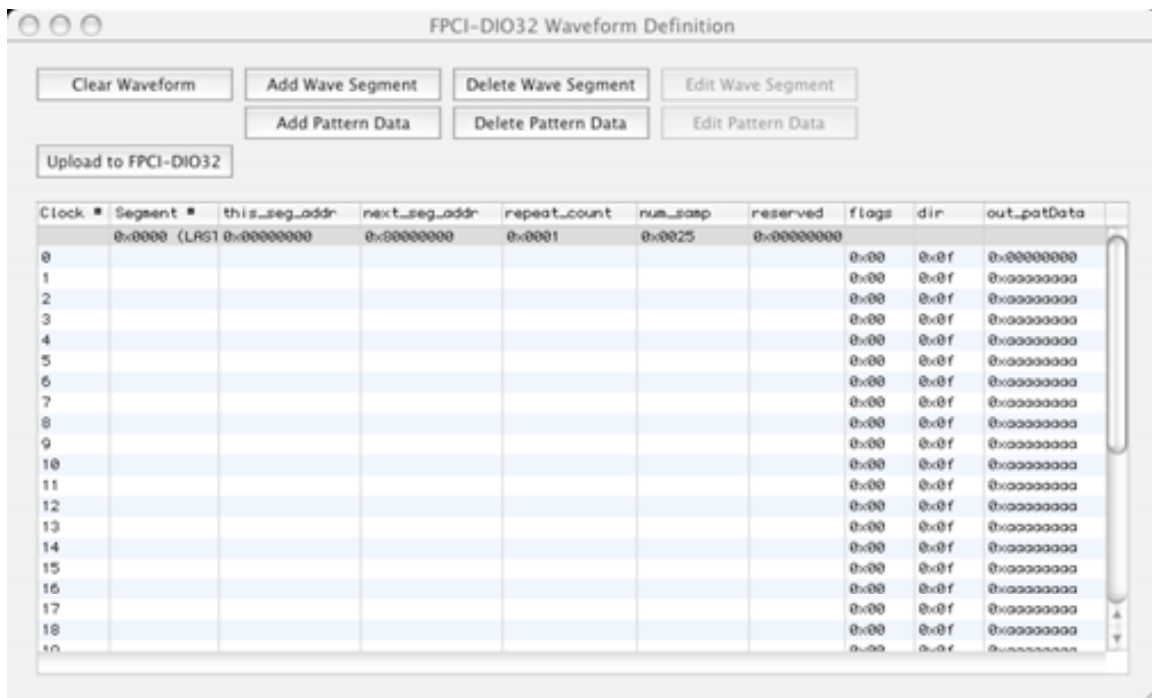
The FPCI-DIO32 allows pattern words to be defined with both the desired bit state for the 32 bits in the I/O port as well as the direction of the signal. The direction can be programmed with a resolution of 4 bits. That is, each nibble lane in the I/O port word may be defines as either 'inputs' or as 'outputs'. In addition, this direction can be programmed on a per clock cycle basis. This unique functionality on the card allows waveforms generation to emulate complex 3-state and bidirectional bus structures.

### **3.8 Waveform Editor**

The FpciDio32Explorer program allows display and manual editing of waveform segment definitions via the 'Waveform Definition' window. You can bring up this window by selecting the 'Waveform Definition' item under the 'FPCI-DIO32' menu in the title bar for the application.

The 'Waveform Definition' window will display the contents of waveform defined in memory local to the FpciDio32Explorer application. All waveform edit operations are performed on this local copy of the waveform description and do not alter the waveform description as stored in the memory on the FPCI-DIO32 card. After all edit operations are finished on the waveform, the user must upload the waveform to the FPCI-DIO32 card before starting waveform playback.

The waveform definition window displays the waveform data in a familiar spreadsheet format. The data presented has a one-for-one correspondence with the waveform segment descriptor record format that is interpreted by the FPCI-DIO32 card. For details on the various fields in this memory structure, refer to section x.x.



One thing to not is that the first column of the waveform display is a clock period count. This count starts at a count of '0' for the first pattern word in the waveform and increments for each sequential clock period in the waveform. However, only the pattern words defined in the waveform segments are numbered. The waveform display does not expand the waveform to account for any looping that is defined for the waveform segments. This was done in order to present a concise representation of the waveform clocks. It would also have been impossible to have a correct display for instances where the 'infinite' loop count were selected.

### 3.8.1 Waveform Definition Edit Buttons

The top area of the waveform definition window includes a number of user interface buttons that will allow you to edit the waveform definition in memory.

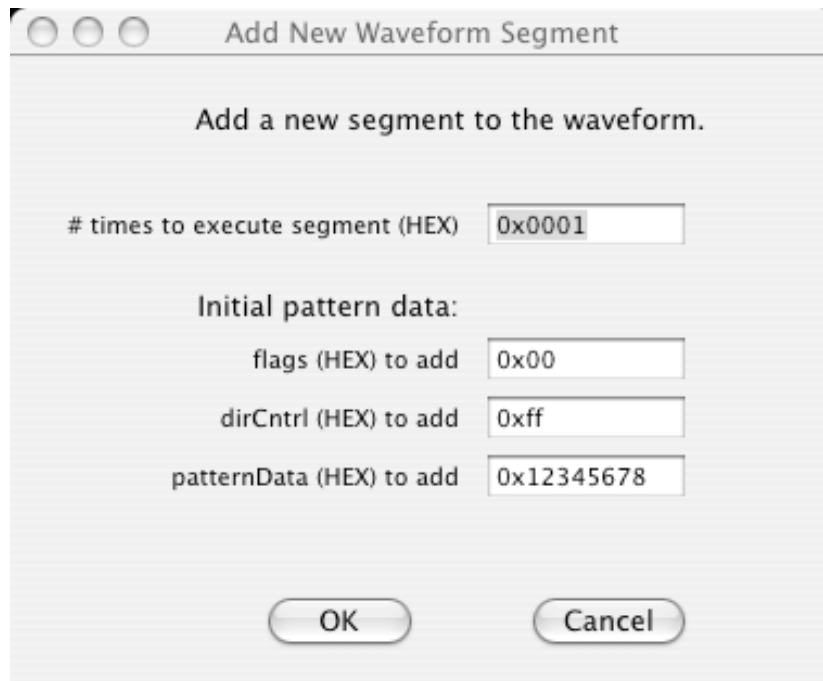
#### 3.8.2 Clear Waveform

Pressing this button will erase any currently defined waveform segments and their pattern words from memory. It is recommended that you use this button to initialize the memory array before adding any memory waveform segments.

#### 3.8.3 Add Wave Segment

This button will add a new waveform segment descriptor to the waveform. It will add it to the end of the currently defined list of waveform descriptors for the waveform. The following dialog box will be presented to the user:



A screenshot of a dialog box titled "Add New Waveform Segment". The dialog has a title bar with three window control buttons (minimize, maximize, close) on the left. The main content area contains the text "Add a new segment to the waveform." followed by four text input fields. The first field is labeled "# times to execute segment (HEX)" and contains "0x0001". The second field is labeled "Initial pattern data:" and contains "0x00". The third field is labeled "flags (HEX) to add" and contains "0xff". The fourth field is labeled "patternData (HEX) to add" and contains "0x12345678". At the bottom of the dialog are two buttons: "OK" and "Cancel".

Add New Waveform Segment

Add a new segment to the waveform.

# times to execute segment (HEX)

Initial pattern data:

flags (HEX) to add

dirCntrl (HEX) to add

patternData (HEX) to add

OK Cancel

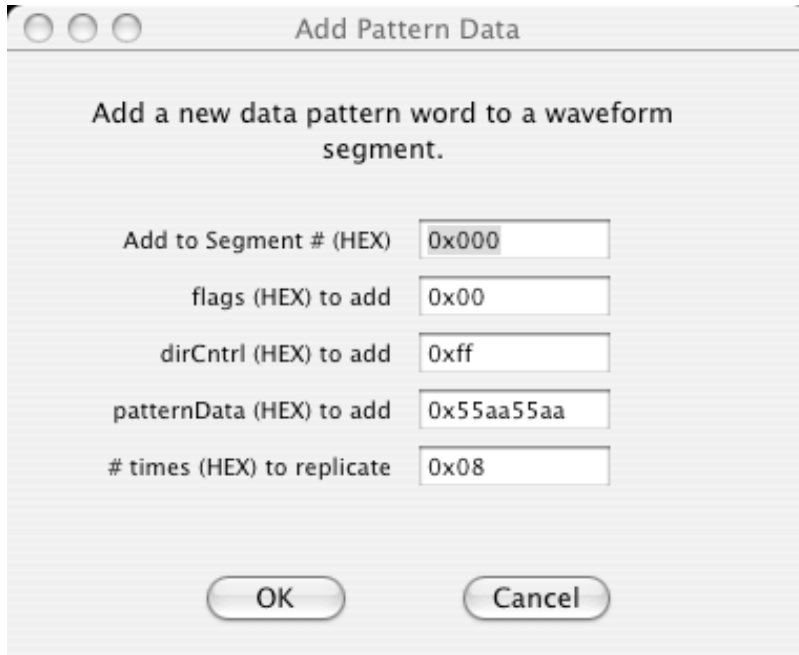
All fields in the dialog box are entered in Hex notation. Therefore only 0..9, A..F characters are allowed. You must <tab> to the next field or hit <return> in order for any user entries in the text fields to be recognized.

This dialog puts a GUI on the `wvSegRecord` structure. The program will allow the user to specify the `wv_repeat_count` field. It will also request the parameters for the first pattern word to be used by the waveform segment. All of the other fields will be calculated by the program. In particular, the program will fill in the segment address fields such that the new segment will be pointed to by the previous segment that was last in the waveform list. It will also make the new segment the last in the waveform description.

#### 3.8.4 Add Pattern Data Button

This button will allow the user to insert additional pattern words to a given waveform segment that you can specify in the dialog box. The pattern words defined will be added to the end of the selected waveform segment.

A common operation used when defining waveform patterns, is to replicate pattern words across several clock periods. Because of this, the dialog box will allow the user to specify a count that will be used to replicate the specified pattern. Remember that the FPCI-DIO32 card's hardware requires all waveform pattern words in a segment to reside in a single memory page. Memory pages are 256 words long. Given that each waveform segment header uses up the first 4 memory words, that will leave 252 spaces for pattern words to be defined.

A screenshot of a Windows-style dialog box titled "Add Pattern Data". The dialog has a title bar with three standard window control buttons (minimize, maximize, close). The main text inside the dialog reads "Add a new data pattern word to a waveform segment." Below this text are five input fields, each with a label to its left: "Add to Segment # (HEX)" with value "0x000", "flags (HEX) to add" with value "0x00", "dirCntrl (HEX) to add" with value "0xff", "patternData (HEX) to add" with value "0x55aa55aa", and "# times (HEX) to replicate" with value "0x08". At the bottom of the dialog are two buttons: "OK" and "Cancel".

Add Pattern Data

Add a new data pattern word to a waveform segment.

Add to Segment # (HEX) 0x000

flags (HEX) to add 0x00

dirCntrl (HEX) to add 0xff

patternData (HEX) to add 0x55aa55aa

# times (HEX) to replicate 0x08

OK Cancel

As with all dialog boxes in the FpciDio32Explorer program, text field edits are only recognized after an edit operation is completed. This can be done by either <tabbing> to another edit field or by hitting <return>.

Pressing the <OK> button will insert the pattern words into the waveform segment.

### 3.8.5 Delete Wave Segment, Delete Pattern Data

These buttons allow you to delete data from a waveform description. The 'Delete Wave Segment' button will delete an entire waveform segment descriptor. The segment that will be deleted is the one that has any part of it's description highlighted in the display. You can hilight the segment header or any of the pattern words defined for the segment in order to select it.

The 'Delete Pattern Data' button will only delete the currently selected pattern word. The program will issue a warning if no pattern word is selected.

## 3.9 Upload to FPCI-DIO32

After all edit operations are complete for your waveform, you need to upload the waveform description to the card before it can play the waveform back. That is because the DMA controller on the card only can read pattern words from the local memory on the card. Waveform descriptions cannot reside in the host computer's main memory.

Pressing this button will upload the entire waveform description to the FPCI-DIO32 card's memory. It also will cause a read of the card's memory back in order to fill out the

‘Card Wavform Dump’ window contents. This window is described in the next section of the manual.

After the upload operation it is a good idea to look at the ‘Card Waveform Dump’ window to verify the upload. A side effect of reading back the waveform description is that the contents of the input memory buffer that matches the new waveform description will also be displayed. This is useful, for instance, after playing back a waveform in order to examine what data was captured in the input buffer during the waveform playback.

To do this, you would perform the following actions.

- 1) define a waveform
- 2) Upload the waveform to the card
- 3) Command waveform playback.
- 4) Upload the waveform again
- 5) Review the contents of the ‘Card Waveform Dump’ window.

Step #4 is where the trick happens. The waveform description is already in memory on the card and after waveform playback, the input buffer contains all of the data that was captured during playback. When you re-upload the waveform description, the output waveform buffer is written with the same data since you did not make any changes to the waveform definition. However, the act of reading back the waveform data to fill in the contents of the ‘Card Waveform Dump’ window will refresh the column with the input buffer’s data to be displayed with the most recent data capture.

### **3.10 Card Waveform Dump Window**

This window is very similar to the ‘Waveform Definition’ window. While the ‘Waveform Definition’ window display the waveform data being stored in the host computer, the ‘Card Waveform Dump’ window displays the waveform data that exists in the local memory on the FPCI-DIO32 card. It also displays the input buffer associated with the last waveform playback operation.

This window is for display only. There are no operations that can be performed on the data. This window’s data is updated every time a new waveform description is uploaded to the FPCI-DIO32 card.

## **4 FPCI-DIO32 Function Description**

### **4.1 General**

The card has two distinct modes of operation. These are defined mode0 and mode1 respectively. Which mode of operation you use will depend upon the data transfer and data processing requirements of your application.

Mode0 is typically called ‘static transfer’ mode of operation. The data is transferred directly between the host CPU and the customer I/O port register on the FPCI-DIO32 card. As such, the maximum data transfer rate is limited by the speed of the host computer as well as the data transfer latency of the PCI interface bus. The time between successive word transfers can be rather unpredictable in this mode since it is dependant upon the workload of the host computer.

Mode0 operation is very useful for those applications where program execution decisions need to be made based upon the state of the I/O signals.

Mode1 is the ‘dynamic transfer’ mode of operation. The data words are transferred between the FPCI-DIO32 card’s local memory and the customer I/O port. A DMA engine is responsible for handling the individual word transfers. The host application software simply has to initialize the transfer to get it started.

Mode1 data transfers can be very high speed and have absolute predictability on the timing of the individual word transfers. The waveform data pattern has to be computed and uploaded to the FPCI-DIO32 card prior to starting waveform playback.

Mode1 transfers can be started via a software command from the host or by hardware events triggered by state changes on the customer I/O port signals.

### **4.2 Mode 0 Operation**

Mode 0 operation is intended for static input/output signal transmission. All data transfers from the host computer to the customer I/O ports on the FPCI-DIO32 card are accomplished via programmatic calls from the user’s application program. As such, data can only be transferred as fast as the application program can execute. No hardware DMA assist capabilities exist in this mode.

The FPCI-DIO32 card’s 32 bit I/O signals data direction must be defined before any data transfers are performed. However, the data direction is not independently controllable for each of the 32 bits in the port. Instead, the 32 bits are grouped into 8 sets of 4bit nibbles. Each of the nibble lanes can have their direction defined via the `c_mode0_ctrl` register. Once the direction of the signals has been programmed, the application program can output data to the ‘output’ signals by writing to the `c_mode0_data` register. Similarly, the

application program can read the state of the signal lines defined as ‘inputs’ by reading from the `c_mode0_data` register.

During writes to the `c_mode0_data` register, only those bits defined as ‘output’ signals will be relevant. The bits written to the register that are associated with ‘input’ signals will have no meaning. On reads from the `c_mode0_data` register, the true state of all of the I/O signals is returned. This is true whether the FPCI-DIO32 card is driving the signal pin or the bit is defined as an ‘input’ and the pin is being driven by some external equipment. Reads from the `c_mode0_data` register will always return the current state of the signals on the I/O port.

### **4.3 Model Operation**

For applications requiring waveform pattern generation capabilities or input data capture at high data rates or long record lengths, you will need to use model operation of the FPCI-DIO32 card.

The waveform will have to be constructed and uploaded to the card’s local memory before waveform playback is initiated. The waveform description format that you will use is described in the appendix of this document. Waveforms are described in terms of waveform segments. Individual waveform segments are linked together to form the desired waveform. Flexibility is provided by allowing waveform segments to repeat up to 65K times before moving on to the next segment in the list. This will allow very long waveforms to be described in the available pattern memory on the card. Additional flexibility is provided by allowing the sequence of waveform segments to be altered by simply changing the segment pointers in the list.

Waveform pattern words are output to the customer I/O port synchronous with the data transfer clock being used. This clock may originate from an on-board clock synthesizer, or provided by the external equipment interfaced to the card. The on-board clock synthesizer is a DDS based device that provides milli-hertz resolution on the desired clock frequency.

Pattern definition words define the state of the individual signal lines on the I/O port on a clock by clock basis during waveform playback. As well, the pattern word can define the data direction of the nibble lanes thus allowing you to change the data direction on the fly as waveform playback occurs. It does this by tri-stating the output buffers on the signal drivers to the I/O port.

Simultaneous with waveform output the card will capture the state of the I/O port signals in a separate memory buffer. Application software can later examine the contents of this input buffer to analyze the data that was collected.

A pattern word recognition function on the card allows waveform playback to be started on the detection of a specific signal state(s) on the customer I/O port. Otherwise, the

application software running on the host computer can command the waveform playback by means of a register write to the card.

## 5 Register Level Programming

All software interface operations between the host computer and the FPCI-DIO32 card are accomplished via accesses to either the waveform pattern memory or to a set of registers on the card. The registers allow commands to be written to the card's logic as well as the ability to read status information from the card.

The following manual section describes the registers and the bits fields defined in them. Application software written by the user can access these registers directly via the card's driver.

### 5.1 0x0200 0000 c\_bk2\_data Register

Field	Description	Read	Write	Reset Value
31:16	Reserved	Yes	No	Na
15:0	Upper 16 bits of memory Bank 0	Yes	Yes	Na

BK2 data holding register. In order to simplify the board level hardware design, data accesses to SDRAM Bank 2 is accomplished by means of a data holding register. Banks 2 and 0 are concatenated to make a single 48 bit wide memory array. Any accesses to this array are accomplished with a single memory cycle. Since the host computer's PCI interface is only 32 bits wide, accesses from the host are done in a two operation cycle. The low 32 bits of the memory word is accessed with a standard long word memory cycle. The host computer supplies the low 32 bits of the 48 bit memory word via the PCI bus interface. This register is the source and destination of the upper 16 bits of the 48 bit memory word. Only the lower 16 bits are valid.

### 5.2 0x0200 0004 c\_dds\_freq Register

Field	Description	Read	Write	Reset Value
31:16		Yes	Yes	Na
15:0		Yes	Yes	Na

These register lines are interfaced directly to the serial I/O pins of the DDS chip that generates the I/O Master clock. This clock is used to time the data I/O transfers from the card.

This register is used to set the frequency of the master clock.

You have to obey the serial protocol of the DDS chip when reading or writing this register. It is not recommended that the user access this register directly. Instead, there are high level driver calls that should be used.

### 5.3 0x0200 0008 c\_dds\_cntrl Register

Field	Description	Read	Write	Reset Value
31:16		Yes	Yes	Na
15:0		Yes	Yes	Na

This register is used in conjunction with the c\_dds\_freq register to program the DDS clock generator on the card. It allows certain functions to be configured on the chip.

As with the c\_dds\_freq register, it is not recommended that the user access this register directly. There are higher level driver calls that should be used instead.

### 5.4 0x0200 000C c\_master\_cntrl Register

Field	Description	Read	Write	Reset Value
31	Fuse map loaded status - '0' = 3.3v PCI signaling - '1' = 5v PCI signaling	Yes	No	Na
30:9	reserved	Yes	No	Na
8	Invert I/O Mclk for input latch	Yes	Yes	'0'
7:6	Operation Mode: 00 = mode 0 01 = reserved 10 = mode 1 11 = reserved	Yes	Yes	"00"
5	PRGMRS_LED	Yes	Yes	'0'
4	BUF_IO_CLK_OUT_EN Customer I/O IO_CLK signal enable '1' = disable output clock driver '0' = enable clock to customer I/O	Yes	Yes	'1'
3	IO_CLK_EN4 Mux Ea	Yes	Yes	'0'
2	IO_CLK_EN3 Mux Eb '0' = enable drive on RTI clock line '1' = disable drive on RTI clock line	Yes	Yes	'1'
1	IO_CLK_EN2 Mux S1	Yes	Yes	'0'
0	IO_CLK_EN1 Mux S0	Yes	Yes	'0'



This register is used to set various parameters on the card that are used in either mode0 or mode1 operation of the card. It is also the register used to define which of the two modes are to be used for subsequent I/O operations.

#### **5.4.1 c\_master\_cntrl Bit31 – Fuse map loaded status**

This bit is a read only bit that will give the status of the power-up configuration of the card. Since the card is a universal PCI bus card that can be operated in either 5v or 3.3v signaling environments, the card will have to correctly configure its logic for the environment that the host computer allows. This is done only at power-up time of the host computer. The card's logic will correctly configure itself and then write this bit to reflect the configuration that was performed.

#### **5.4.2 c\_master\_cntrl Bits 30:9**

These bits are not used.

#### **5.4.3 c\_master\_cntrl Bit 8 - Invert I/O Mclk for input latch**

During data transfers from external equipment to the FPCI-DIO32 card, the card will sample the input data with the currently selected I/O Mclk. This bit allows the user to specify whether the rising edge or the falling edge of the clock is used to clock the data. The use of this bit is application dependant. If input data transitions around the rising edge of the clock, then it would be beneficial to sample the data on the falling edge of the clock in order to correctly sample the data. Usually you need to have the data stable at the time the card samples the data. Not doing so can cause invalid data to be captured.

Setting this bit to a '1' will cause the FPCI-DIO32 card to sample the input data on the falling edge of the interface clock.

#### **5.4.4 c\_master\_cntrl Bits 7:6 - Operation Mode**

The FPCI-DIO32 card can operate in either of two modes. These are defined as mode0 and mode1. These two bits are written to define the desired operation mode. Notice that there are only two modes of operation for the card but two bits used to define the mode. This results in two 'reserved' codes for this bit field.

#### **5.4.5 c\_master\_cntrl Bit 5 - PRGMRS\_LED**

The card has a row of 6 LEDs on its top edge. They are labeled D1 through D6. One of these LEDs (D5) is controlled by writing to this bit in the register. Setting the bit to a '1' will turn on the LED.

This is useful during software debugging operations. You can write to this register at certain points in your program to give status on the programs execution.

#### 5.4.6 c\_master\_cntrl bit 4 - BUF\_IO\_CLK\_OUT\_EN

Pin 30 of the customer I/O connector to the card allows the connection of an interface clock signal. This is the clock that is used to sample both the output data and the input data being transferred by the card during mode1 operations. The card's design allows for this clock to be provided by the card itself or by the customer's equipment connected to the card.

If one of the on-card clock sources are being used for the I/O sample clock and you wish to drive this clock to the attached interface equipment, then this bit must be a '0'. A '1' written to this bit puts the clock driver into a high impedance state. There is a resistor termination network on this signal line that will result in a logic '1' on the signal if there are no other signals driving this pin.

#### 5.4.7 c\_master\_cntrl bit 3 - IO\_CLK\_EN4

This signal is used for factory test of the card only. It should always be set to its default value of '0' and never programmed by the user to be a '1'.

#### 5.4.8 c\_master\_cntrl bit 2 - IO\_CLK\_EN3

The FPCI-DIO32 card has an expansion connector labeled P3 at the top edge of the card. This is used to link several of the FPCI-DIO32 cards together for those applications that require wider digital word widths. In such application, you will need to define one card as the master card and the other cards as slaves. The master will need to provide the clock that will be used by all of the cards to time the I/O data transfers. Connector P3 carries the signals that allow this to be accomplished.

The IO\_CLK\_EN3 bit is an enable to allow the card to drive its clock onto the expansion bus. When this bit is a '0', the card will enable its clock driver. When programmed as a '1', the driver is disabled.

#### 5.4.9 c\_master\_cntrl bit 1:0 - IO\_CLK\_EN2, IO\_CLK\_EN1

These two register bits allow you to define which, of several available clocks is used as the I/O timing clock. The following table details the four options:

Code	Clock Source
00	33Mhz (fixed)
01	Buffered I/O clock
10	RTI clock
11	DDS clock (variable)

Code 00 – This is a fixed 33 Mhz clock source generated on the card. It is the output of a PLL clock generator running with a 5Mhz crystal reference frequency.

Code 01 – This is a buffered version of the clock on the customer I/O connector. This is useful when you wish to provide the transfer clock from external equipment interfaced to the card. When this is the case, make sure bit 4 in this register is programmed to a logic '1'. This will make sure that the FPCI-DIO32 card isn't also trying to drive the I/O clock at the same time as the external equipment.

Code 10 – This clock source selects the clock on the RTI expansion bus connector. Remember to make sure that bit 2 in this register is programmed to a '1' so that there is no conflict with more than one card trying to drive the RTI clock.

Code 11 – This selects the DDS based clock generator on the card. The DDS is a digital phase accumulator based generator. It runs with a master clock of 120 Mhz and has 32 bit resolution on its phase accumulator. As such, you can program clock frequencies within the range of 25 mHz to its maximum of 20 Mhz with millihertz resolution.

## 5.5 0x0200 0010 c\_mode0\_data Register

Field	Description	Read	Write	Reset Value
31:0	Data Pattern	Yes	Yes	00000000

During Mode 0 operation, the host application transfers data to the I/O port by programmatically reading and writing to this register. The data bits written to this register are driven to the signals on the I/O port as qualified by the setting of the 'Direction' control bits. The 'Direction' control bits are defined in the c\_mode0\_cntrl register. Only the nibble lanes that are defined as being output bits will permit the register contents to be driven.

This register is only used when the Mode 0 operation is selected in the c\_master\_cntrl register.

## 5.6 0x0200 0014 c\_mode0\_cntrl Register

Field	Description	Read	Write	Reset Value
31:16		Yes	No	Na
15:12	IO_HNDSHK[3:0] data value	Yes	Yes	"0000"

Field	Description	Read	Write	Reset Value
11	DIR11 – sets direction of IO_HNDSHK3 line ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
10	DIR10 – sets direction of IO_HNDSHK2 line ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
9	DIR9 – sets direction of IO_HNDSHK1 line ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
8	DIR8 – sets direction of IO_HNDSHK0 line ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
7	DIR7 – sets direction of IO_SIG [31:28] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
6	DIR6 – sets direction of IO_SIG [27:24] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
5	DIR5 – sets direction of IO_SIG [23:20] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
4	DIR4 – sets direction of IO_SIG [19:16] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
3	DIR3 – sets direction of IO_SIG [15:12] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
2	DIR2 – sets direction of IO_SIG [11:8] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
1	DIR1 – sets direction of IO_SIG [7:4] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’
0	DIR0 – sets direction of IO_SIG[3:0] lines. ‘1’ = output, ‘0’ = input	Yes	Yes	‘0’

This register allows you to specify the direction of the I/O signal when mode0 operation is being used. Each group of 4 bits on the card’s I/O port can be selected as being an output signal or an input signal. The bits in the nibble will be driven as outputs whenever the corresponding direction control bit is programmed to a ‘1’. The signals will be ‘inputs’ otherwise.

In addition, there are four hardware handshake lines on the card that are also brought out to the I/O connector. These bits can be programmed just as the regular data port signals. That is both the direction of the signal as well as the driven state may be specified. There are 4 direction control bits available so that each individual handshake signal may be independently programmed.

Bits 14:8 function the same for both mode0 and mode1 operation. During mode1 operation, Bit 15, the IO\_HNDSHK[3] signal is different however when operating in mode1. During mode1, this bit will contain a ‘data valid’ bit that will go true for the duration of the waveform playback. It will go false after the last pattern word is read from the waveform memory. This can be used as a dynamic handshake signal for certain

applications. Remember to set bit 11 in this register to a '1' so that the 'data valid' signal will be driven on the I/O connector.

## 5.7 0x0200 0018 c\_model\_cntrl Register

Field	Description	Read	Write	Reset Value
31:18	reserved	Yes	Yes	Na
17:15	RTI_trig_in_sel	Yes	Yes	Na
14:8	RTI trigger output enable: Bit [14] = trig6 enable Bit [13] = trig5 enable Bit [12] = trig4 enable Bit [11] = trig3 enable Bit [10] = trig2 enable Bit [9] = trig1 enable Bit [8] = trig0 enable	Yes	Yes	'0'
7	Interrupt status	Yes	Yes	'0'
6	Interrupt Enable	Yes	Yes	'0'
5:4	session_start_type 00 – software triggered start 01 – input pattern triggered start 10 – RTI trigger	Yes	Yes	'00'
3	soft_trig	Yes	Yes	'0'
2	enable	Yes	Yes	'0'
1	trig_ready_status	Yes	No	N/A
0	run_status	Yes	No	N/A

This register is used to define certain operating parameters used during mode1 operation of the card. Mode1 operation is the mode where the FPCI-DIO32 card is used as a waveform pattern generator. Waveform descriptions are stored in a local memory on the card and then played back under control of the host application or external equipment. Waveform descriptions allow looping, dynamic signal direction assignment as well as input data capture during playback. All of this happens under control of the hardware DMA engine on the FPCI-DIO32 card but its operation is directed via control bits in this register.

Several bits in this register are used to define the desired event that will be used to start the waveform playback. One of three different events that can be selected as defined by bits 4 and 5 in this register.

Software Triggered Start - Code '00' selects the 'software triggered start' event for waveform playback. This is where a command from an application program running on the host computer is intended to start the waveform playback.

Input Pattern Triggered Start – Code ‘01’ selects a waveform playback event when the customer I/O signals on the 32 bit port matches the conditions setup in the pattern recognition logic. Once the waveform generator is enabled, nothing will happen until the desired pattern is detected. Then waveform playback will start.

RTI Triggered Start – Code ‘10’ allows waveform playback to start on a signal from a master FPCI-DIO32 card when multiple cards are chained together over the RTI bus. This is how multiple cards are synchronized at the start of waveform playback. There are 7 RTI trigger lines that can be used. Bits 17:15 define which of the seven is actually used for a trigger. In addition, if this card is the master card, then you can select which RTI trigger signal is driven by the local trigger event. Bits 14:8 are reserved for this. A ‘1’ in the respective bit position will drive the local trigger signal onto that RTI trigger line. You may drive the trigger onto multiple lines if necessary.

Bit 7 ‘interrupt status’ – this bit will be set to a ‘1’ whenever the card has generated an interrupt to the host computer. It is a read only status bit.

Bit 6 ‘interrupt enable’ – the pattern generator logic on the card allows for interrupts to be sent to the host computer at definable points during the waveform playback. Bit 6 needs to be set to a ‘1’ in order for any of the waveform interrupts to be recognized. If an interrupt has occurred, then clearing bit 6 will also clear the interrupt. This is handled automatically by the cards driver software.

Bit 3 ‘soft\_trig’ – this is the bit that needs to be set by the host computer to trigger a waveform playback operation.

Bit 2 ‘enable’ – is the bit that should be set just before a waveform playback is commanded. All other operating bits, such as desired clock sources, pattern logic, etc. should be defined before setting bit 2. Once bit 2 is set, the host can poll bit 1 to determine when the card is ready to receive the trigger event. Setting bit 2 configures the DMA controller of the pattern generator. Among other things, it preloads the first 128 pattern words into the output FIFO on the card so that the data is ready when the start trigger arrives. It is possible to define a waveform that loops forever during playback. In this case, you can clear bit 2 to command a playback stop.

You must bring ‘enable’ low before a new waveform generation sequence. If you bring ‘enable’ low while a current waveform is running, then that waveform playback will be aborted.

Bit 1 ‘trig\_ready\_status’ – this is a read only status bit. Immediately after setting bit 2, this bit will be ‘0’. After the card’s internal logic is initialized and ready to accept a start trigger, this bit will go high. When operating with a software triggered start, you should wait for bit 1 to go high before issuing the start command. When relying on the hardware triggered start, then the trigger will be recognized at any time after bit 1 goes high.

Bit 0 ‘run status’ – this bit is a read only status bit. It will be high during the time that the card is playing back a waveform.

## 5.8 0x0200 001C c\_match\_pattern Register

Field	Description	Read	Write	Reset Value
31:0	Pattern being sought	Yes	Yes	‘0’

The FPCI-DIO32 card has a built in pattern recognition function. The search pattern is defined in this register and is used to monitor the 32 signal lines on the customer I/O connector. When the pattern match occurs, it can be used to trigger a waveform playback.

## 5.9 0x0200 0020 c\_pattern\_mask Register

Field	Description	Read	Write	Reset Value
31:0	Pattern Mask – a ‘1’ in a bit lane signifies that the bit is to be considered during pattern recognition.	Yes	Yes	‘0’

The pattern mask register is used to define ‘don’t care’ bits in the search pattern. You will need to set the corresponding bit in this register in order for that bit to be considered in the compare operation. A pattern hit will be recognized by the pattern detect logic if the particular bit’s mask is set to a one and the respective match pattern bit equals the corresponding bit on the I/O connector. If more than one bit in the word has its mask set, then all bits having their mask set must compare with the I/O port word for a hit to be recognized.

## 6 Cookbook

We will run through two software examples that will demonstrate the two operational modes of the FPCI-DIO32 card. The example assume that the card's driver and the fcFw.Framework library have been installed on the host computer. We will use Apple's Xcode development environment to compile the program. You will need the following two header files from the software distribution disk that came with the card:

- 1) fcDio32.h
- 2) fpciDio32xUserClient.h

These header files will need to be added to the software project as noted below.

### 6.1 First Example

The first example program demonstrates mode0 operation of the card. The following steps take you through the creation of the project that we will be using:

- 1) Open up the Xcode application
- 2) Create a new project by selecting the 'New Project...' menu item under the 'File' menu. This will bring up a dialog box requesting the type of project desired. Select 'Foundation Tool' under the 'Command Line Utility' heading. Name the project 'cook1' and then hit the 'finish' button.
- 3) Add the fcFw framework to the project. <control> click on the 'External Frameworks and Libraries' folder in the project window and select the 'Add->Existing Frameworks...' menu item. Navigate to the /Library/Frameworks/fcFw.Framework directory and then hit the 'Add' button.
- 4) Add the fcDio32.h header file to the project. <control> click on the 'Sources' folder in the project and select the 'Add->Existing Files...' menu item. Navigate to the location of the fcDio32.h header file and then hit the 'Add' button
- 5) Add the fpciDio32xUserClient.h header file to the project. <control> click on the 'Sources' folder in the project and select the 'Add->Existing Files...' menu item. Navigate to the location of the fpciDio32xUserClient.h header file and then hit the 'Add' button
- 6) Add the following lines to the cook1.m file:



```

01 #import <Foundation/Foundation.h>
02 // need to include the header for the fcFw framework
03 #import "fcDio32.h"
04
05
06
07 fcDio32*   gMyCard; // reference to our card object,
08               // handles interaction with the FPCI-DIO32 PCI card.
09
10
11
12
13 int main (int argc, const char * argv[]) {
14     UInt32      value, readValue;
15     kern_return_t  ret;
16
17
18
19     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
20
21     // create the software object that I will use
22     // to communicate with the card
23     gMyCard = [[fcDio32 alloc] init];
24
25
26     NSLog(@"Beginnning mode0 I/O port test\n");
27
28     // set mode 0, xtal clock source, no RTI clk,
29     // enable clk to customer I/O port
30     ret = [gMyCard fpciDio32x_SetMaster_Cntrl_Reg: 0x00000004];
31
32     // set I/O signals to 'output' direction
33     ret = [gMyCard fpciDio32x_SetMode0_Cntrl_Reg: 0x000000FF];
34
35     // write new port value
36     value = 0x12345678; // random value
37     ret = [gMyCard fpciDio32x_SetMode0_Data_Reg: value];
38
39     // read it back
40     ret = [gMyCard fpciDio32x_GetMode0_Data_Reg: &readValue];
41
42     NSLog(@"Port state read = %8x\n", readValue);
43
44
45     // all done with the card so release the card object
46     [gMyCard CloseCardConnection];
47
48     [pool release];
49     return 0;
50 }
51
52

```

The following example demonstrates mode0 operation of the FPCI-DIO32 card.

- 1) Open the driver to the card. Program line #23 calls a `fcFw.Framework` function that does this automatically.
- 2) Set the `c_master_cntrl` register to an appropriate value. Selecting mode0 operation and the desired master clock source is necessary. See line #30.
- 3) Set the direction of the I/O signals by writing to the `c_mode0_cntrl` register. Program line #33 does this by setting the low byte to `0xFF`. A '1' designates the nibble lane to be an output and there are 8 nibble lanes in the port word.
- 4) Output data to the I/O port by writing to the `c_mode0_data` register. Program line #37 writes an arbitrary number (`0x12345678`) to the I/O port register.
- 5) Reading from the `c_mode0_data` register will return the current state of the I/O signals. Line #40 reads back the value we just wrote to the I/O port register.
- 6) Close the driver when finished. This is done in line #46.

## 6.2 Second Example

The second example program demonstrates mode1 operation of the card. The following steps take you through adding the program statements to the project we created for example #1 above.

- 1) Start with the first example project above.
- 2) Add the following lines to the `cook1.m` file:

```
01  wvPatternWord  aPatternWord;
02  UInt32          newFrequency;
03  Boolean         ready;
04  Boolean         running;
05  UInt32          myMode1CntrlValue;
06
07
08
09  NSLog(@"Beginning mode0 I/O waveform generation #1.\n");
10
11
12  // reset the mode1 command register before we begin
13  [gMyCard fpciDio32x_SetMode1_Cntrl_Reg: 0x00000000];
14
15  // We will first need to generate the waveform pattern in the
16  // memory on the card. Start by clearing the memory array.
17  [gMyCard clearWaveformDefinition];
18
19  // waveforms are described by means of individual waveform segments.
20  // Very long waveforms are described by chaining multiple waveform
21  // segments together.
22  //
```

```

23 // Each waveform segment descriptor will have an array of pattern words
24 // to define how the waveform will look. There is one pattern word
45 // for each clock cycle of the waveform.
26 //
27 // In this example, we are generating a very short waveform and all
28 // of our data pattern words will fit into a single waveform segment descriptor.
29 //
30 // Create a new waveform segment record for the first (and only)
31 // segment of our waveform.
32 [gMyCard newWvSegRecord];
33
34 // setup a pattern word that will describe the signal states for a single
35 // clock cycle when the waveform is played back.
36 aPatternWord.flags = 0x00; // no interrupts
37 aPatternWord.dirCntrl = 0xff; // all 32 bits as outputs for this clock cycle
38 aPatternWord.patternData = 0x00000001; // LSB its high
39
40 // we will add 32 pattern words to the waveform segment. Each new pattern
41 // word will have the bit shifted one more position to the left.
42 for (i = 0; i < 33; i++)
43 {
44     // add the next pattern word to the waveform definition
45     [gMyCard addPatternWord2Segment: &aPatternWord atIndex: 0];
46
47     // shift the pattern one bit to the left
48     aPatternWord.patternData = aPatternWord.patternData << 1;
49     aPatternWord.patternData = aPatternWord.patternData & 0xffffffff;
50 }
51
52
53 // OK. At this point we have created the definition of our desired waveform
54 // in local memory. Before we can play back the waveform, we need to upload it
55 // to the card.
56 [gMyCard uploadWaveformData2Hardware];
57
58 // set the frequency of the DDS clock generator. this will set the rate
59 // at which pattern words are generated from the card
60 newFrequency = 1000000; // 1 MHz
61 ret = [gMyCard fpciDio32x_SetDDS_Frequency: newFrequency];
62
63
64
65 // setup the board's master control register to allow:
66 // 1) model operation
67 // 2) DDS Clock generator
68 // 3) Enable clock to be driven on the I/O port
69 //
70 ret = [gMyCard fpciDio32x_SetMaster_Cntrl_Reg: 0x00000087];
71
72
73 // now setup a sequence whereby we enable the card's model logic
74 ret = [gMyCard fpciDio32x_SetMode1_Cntrl_Reg: 0x00000000]; // reset mode1 logic
75 ret = [gMyCard fpciDio32x_SetMode1_Cntrl_Reg: 0x00000004]; // enable
76
77
78 NSLog(@"Waiting for mode1 logic to be ready.\n");
79
80 // after we enable the mode1 logic, we poll for the trig_ready_status bit
81 // to be high, signalling that we can issue a trigger command to begin waveform
82 // playback.

```

```

83  ready = false;
84  while (!ready)
85      {
86          ret = [gMyCard fpciDio32x_GetMode1_Cntrl_Reg: &myMode1CntrlValue];
87          myMode1CntrlValue = myMode1CntrlValue & 0x00000002;
88          if (myMode1CntrlValue == 2)
89              ready = true;
90      }
91
92  // the card is ready to begin waveform playback.
93  // issue a software 'trigger' command to start it playing.
94  ret = [gMyCard fpciDio32x_SetMode1_Cntrl_Reg: 0x0000000c]; // enable, and soft_trig
95
96
97  // we can poll the 'run_status' bit to determine when the waveform
98  // is finished playing. we first wait for it to go high
99  running = false;
100 while (!running)
101     {
102         ret = [gMyCard fpciDio32x_GetMode1_Cntrl_Reg: &myMode1CntrlValue];
103         myMode1CntrlValue = myMode1CntrlValue & 0x00000001;
104         if (myMode1CntrlValue == 1)
105             running = true;
106     }
107
108 NSLog(@"Waveform generator running.\n");
109
110 // now wait for it to go low
111 running = true;
112 while (running)
113     {
114         ret = [gMyCard fpciDio32x_GetMode1_Cntrl_Reg: &myMode1CntrlValue];
115         myMode1CntrlValue = myMode1CntrlValue & 0x00000001;
116         if (myMode1CntrlValue == 0)
117             running = false;
118     }
119
120 NSLog(@"Waveform generation finished.\n");

```

Things to not of this example:

- 1) Lines 15 to 50 define the waveform pattern in memory.
- 2) Line 56 uploads the waveform description to the FPCI-DIO32 card.
- 3) Lines 60 and 61 set the I/O clock to 1Mhz.
- 4) Lines 70 to 75 setup the mode1 logic on the card.
- 5) Lines 83 to 90 wait for the DMA engine to become ready.
- 6) Line 94 starts the waveform playback

## 7 Mode 1 Operation

Memory record format:

Waveforms made up of multiple segments. Each segment is described by a memory structure as follows.

Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
		wv_this_segment_address			
		wv_next_segment_address			
		wv_num_samples		wv_repeat_count	
		reserved			
Flags	DIR Cntrl	Pattern Data (first)			
...					
...					
...					
Flags	DIR Cntrl	Pattern Data (last)			

wv\_this\_segment\_address – the address in memory where this segment structure record resides. It is also the address of where the corresponding input data is stored in the input buffer during this waveform segment. The first memory structure needs to reside at address 0x0000 0000.

wv\_next\_segment\_address – the address in memory where the next waveform segment structure is stored. Any address with the MSB set denotes that this is the last segment structure for this sequence.

wv\_num\_samples - number of long words in this waveform segment's data structure (including the header words). The minimum number of samples allowed is xx.

wv\_repeat\_count - The number of times to repeat this waveform segment. A count of "0000" will repeat forever.

Flags - execution flags.

DIR Cntrl - direction control bits for the 8 nibble lanes. '1' = output.

Pattern Data - the pattern to be output during this sample clock.

Flags field definition:

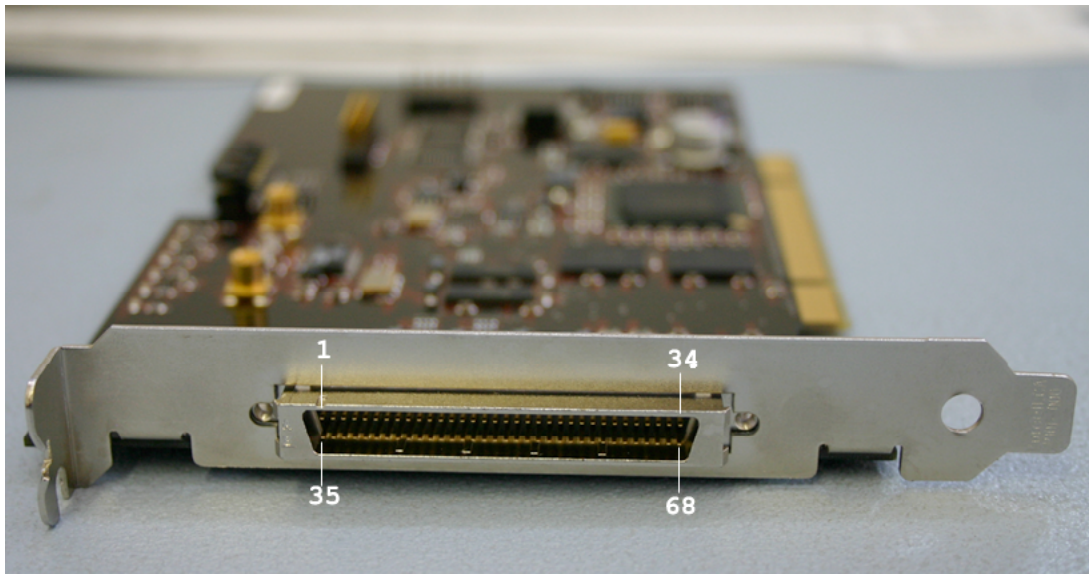
7	6	5	4	3	2	1	0
							INT

## 8 Hardware

### 8.1 Customer I/O connector

The connector mounted on the FPCI-DIO32 card is an Amplimite 050 series male plug from APM Incorporated. Part number = AMP 2-174225-5

The following picture details the orientation and the pin locations on the connector:



Orientation of the customer I/O connector.

### 8.2 Mating Connector

A suitable mating connector for the customer I/O connector is the AMP part number 749699-7.

Pin Number	Signal	Pin Number	Signal
1	IO_SIG31	35	GND
2	GND	36	IO_SIG30
3	IO_SIG28	37	IO_SIG29
4	IO_SIG27	38	GND
5	GND	39	IO_SIG26
6	IO_SIG24	40	IO_SIG25
7	IO_SIG23	41	GND
8	GND	42	IO_SIG22
9	IO_SIG20	43	IO_SIG21
10	IO_SIG19	44	GND
11	GND	45	IO_SIG18
12	IO_SIG16	46	IO_SIG17
13	IO_SIG15	47	GND
14	IO_SIG14	48	GND
15	GND	49	IO_SIG13
16	GND	50	IO_SIG12
17	GND	51	IO_SIG11
18	IO_SIG9	52	IO_SIG10
19	IO_SIG8	53	GND
20	IO_SIG7	54	GND
21	GND	55	IO_SIG6
22	IO_SIG4	56	IO_SIG5
23	IO_SIG3	57	GND
24	GND	58	IO_SIG2
25	IO_SIG0	59	IO_SIG1
26	IO_HNDSHK3	60	GND
27	IO_HNDSHK2	61	GND
28	No Connect	62	GND
29	No Connect	63	CNTRL_TERM
30	IO_CLK	64	GND
31	No Connect	65	DATA_TERM
32	IO_HNDSHK1	66	GND
33	IO_HNDSHK0	67	GND
34	+3.3V	68	GND

**Customer I/O connector pinout.**



### 8.3 Customer I/O signal definition

**IO\_SIG[31:0]** – customer I/O port data word. There are 32 signals in this word. The direction of the I/O signals in the word can be defined on the 4-bit nibble lanes. For instance, IO\_SIG[3:0] are in the first nibble lane. All bits in a nibble lane assume the same signal direction. The I/O signals are LVTTL compatible but at 5VTTL tolerant.

An SN74LVTH32244 device drives these signal pins. In addition, there is input diode protection as well as terminating resistors on the individual signals. The terminating resistors are 10K Ohm and are either pulled to GND or 3.3V depending upon the level of the DATA\_TERM pin.

**IO\_HNDSHK[3:0]** – handshake signals. These are general purpose handshake signals that are programmed via the mode0 control registers. In addition, IO\_HNDSHK[3] is used as a ‘data\_valid’ signal during mode1 operation. Each handshake signal’s direction is independently programmable.

An SN74LVTH125 device drives these signal lines. In addition, there is input diode protection as well as terminating resistors on the individual signals. The terminating resistors are 10K Ohm and are either pulled to GND or 3.3V depending upon the level of the CNTRL\_TERM pin.

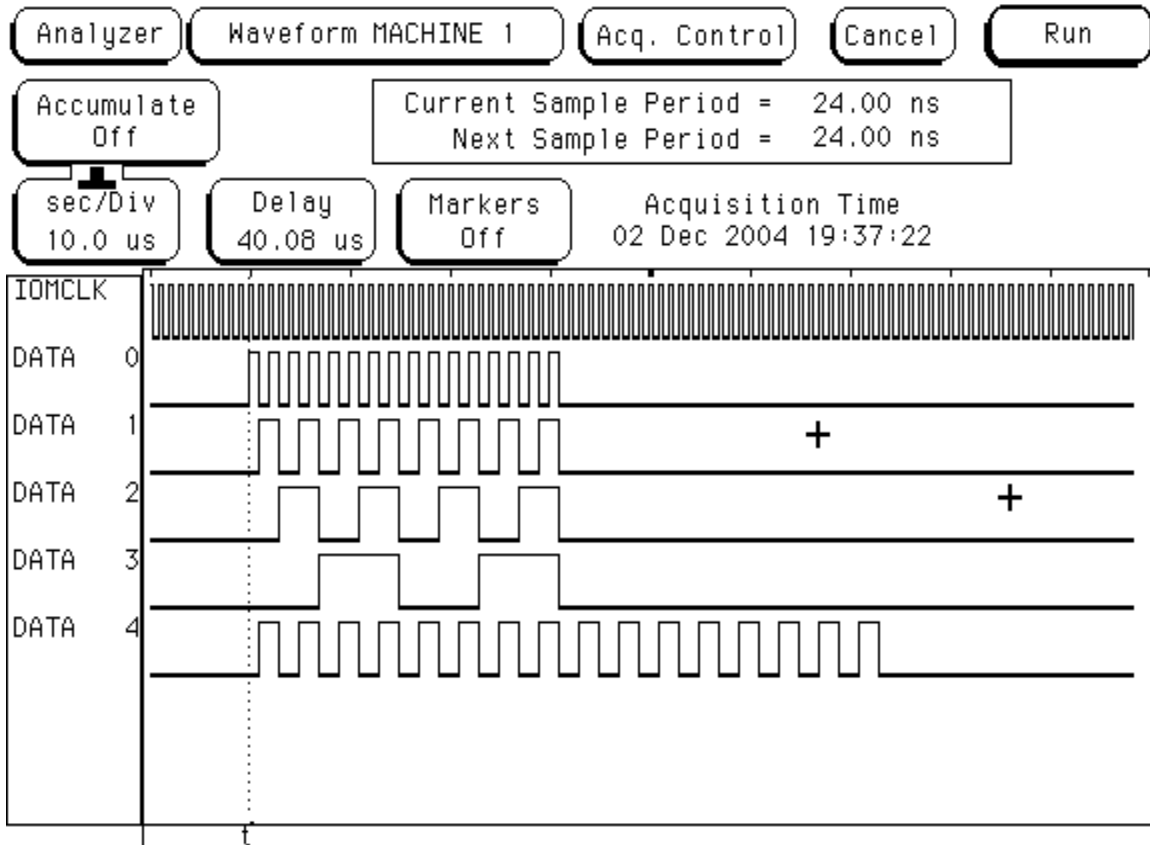
**IO\_CLK** – Customer I/O clock pin. This pin is used to either source or sink the data transfer clock that times the data word transfers over the IO\_SIG lines.

An SN74LVTH125 device drives the clock line. In addition, there is input diode protection as well as terminating resistors on the signal. The terminating resistors are 220 Ohm pull-up and a 330 Ohm pull-down.

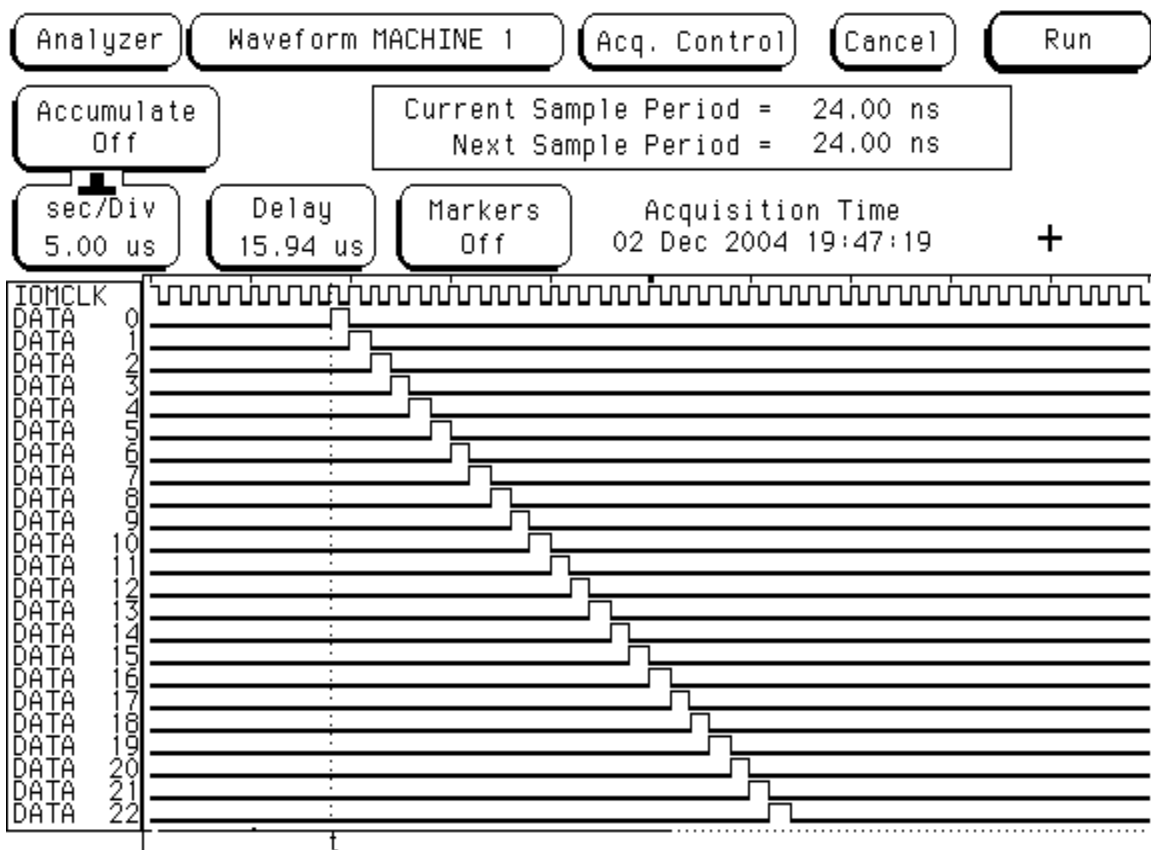
**DATA\_TERM** – This signal is used to set the level to which the IO\_SIG termination resistors are driven. This pin has an internal pull-down resistor of 2.2K Ohms to GND. Tie this pin to the 3.3V pin to terminate to a logic high level.

**CNTRL\_TERM** – This signal is used to set the level to which the IO\_HNDSHK termination resistors are driven. This pin has an internal pull-down resistor of 2.2K Ohms to GND. Tie this pin to the 3.3V pin to terminate to a logic high level.

**+3.3V** – this is a current limited voltage output pin to allow power to external circuitry. The signal is fused with a 0.5A PTC device. If the fuse trips, you will need to power down the card in order to reset it.



DDS = 1mhz  
2 wave segments  
wave segment #1 repeated 2x (counter 0 -> 0xf)  
data bit 4 = clock @ 1/4 rate of DDS clock.



single waveform segment with shifting bit pattern. DDS=1mhz.