Laparoscopic Impulse Engine™
Impulse Engine 2000™

Software Development Kit

Release 4.1
CDOCS-IE
Revised: November 1999
Copyright Immersion Corporation
1999
Immersion Corporation

This manual and the software described herein are copyrighted with all rights reserved. This manual and software may not be copied, in whole or in part, without the written consent of Immersion Corporation, except in the normal use of the software or to make a backup copy of the software. All copies must retain this original copyright notice. This does not allow copies to be made for others, whether or not sold. Also, translating, without permission from Immersion Corporation, into another language or format is considered a form of copying. An exception to this allows individual persons to translate the manuals for their own use.

Impulse Engine 2000 and Impulse Stick are registered trademarks of Immersion Corporation. Immersion Corp is a registered trademark of Immersion Corporation. Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

This document is published for international use.

Version 1.0 March, 1995
Version 2.0 October, 1995
Version 3.0 September, 1997
Version 4.0 Beta November, 1998
Version 4.0 December, 1998
Version 4.1 November, 1999

© Immersion Corporation
2158 Paragon Drive
San Jose, CA  95131
Tel: +1 (408) 467-1900
Fax: +1 (408) 467-1901
Web: http://www.immersion.com
Table of Contents

1 OVERVIEW .........................................................................................................................................................6
1.1 CAPABILITIES OF THE IMMERSION IMPULSE ENGINE I/O CARD .................................................................7
1.2 ISA CARD ADDRESSING (NOT APPLICABLE TO PCI INTERFACES) ..............................................................8
  1.2.1 Base Address ........................................................................................................................................8
  1.2.2 Addressing Two Impulse Engines or Two Impulse Sticks ........................................................................8
2 INSTALLATION INSTRUCTIONS .....................................................................................................................9
  2.1 SYSTEM REQUIREMENTS ..............................................................................................................................9
  2.2 COMPONENTS AND ACCESSORIES ................................................................................................................9
  2.3 HARDWARE INSTALLATION ...........................................................................................................................9
  2.4 SOFTWARE INSTALLATION FOR WINDOWS DRIVERS ................................................................................11
    2.4.1 NT 4 ...................................................................................................................................................11
    2.4.1.1 Installation (NT4) .........................................................................................................................11
    2.4.1.2 Removal (NT4) ...........................................................................................................................11
    2.4.2 Windows 98 ..........................................................................................................................................12
      2.4.2.1 Installing ISA cards (Windows 98) ...............................................................................................12
      2.4.2.2 Installing PCI cards (Windows 98) ...............................................................................................12
      2.4.2.3 Removal (Windows 98) ...............................................................................................................13
    2.4.3 Windows 95 ..........................................................................................................................................13
    2.4.4 Limitations ........................................................................................................................................13
3 DOS SOFTWARE DEVELOPMENT KIT ..............................................................................................................14
  3.1 IMPULSE ENGINE DEMO SOURCE FILES .................................................................................................14
    3.1.1 Compiling and Linking .......................................................................................................................17
    3.1.2 Demo Programs and Borland C/C++ Compiler ................................................................................17
    3.1.3 Demo Configuration ...........................................................................................................................17
  3.2 IMMERSION IMPULSE ENGINE COMMON SOFTWARE LIBRARY ..............................................................18
    3.2.1 COMPILER.H ..................................................................................................................................18
    3.2.2 IEIO.C and IEIO.H .........................................................................................................................19
    3.2.3 READCFG.C and READCFG.H .....................................................................................................21
    3.2.4 TIMEFUNC.C and TIMEFUNC.H .................................................................................................22
4 WINDOWS 95, 98, NT DEVELOPMENT ENVIRONMENT ...............................................................................23
  4.1 INTRODUCTION .......................................................................................................................................23
  4.2 DEFINITIONS .............................................................................................................................................23
  4.3 FUNCTIONS .............................................................................................................................................25
    4.3.1 Device Initialization / End Functions ..................................................................................................25
    4.3.2 Device Info Functions .......................................................................................................................27
    4.3.3 Analog Input Functions .....................................................................................................................28
    4.3.4 Analog Output Functions ..................................................................................................................28
    4.3.5 Encoder Input Functions ....................................................................................................................33
    4.3.6 Digital Input / Output ........................................................................................................................38
  4.4 USAGE .......................................................................................................................................................40
    4.4.1 Implementation .................................................................................................................................40
4.4.2 Compilation ................................................................................................................................. 40
4.5 EXAMPLES ........................................................................................................................................ 41

5 REGISTRY SETTINGS FOR WINDOWS SYSTEMS ............................................................................ 43

5.1 NT4 – ISA INTERFACE .................................................................................................................. 43
5.2 NT4 – PCI INTERFACE .................................................................................................................. 45
5.3 WINDOWS 98 – ISA INTERFACE .................................................................................................. 45
5.4 WINDOWS 98 – PCI INTERFACE .................................................................................................. 46
5.5 WINDOWS 95 – ISA INTERFACE .................................................................................................. 46
1 Overview

Congratulations on your purchase of the Laparoscopic Impulse Engine, Impulse Engine 2000, or Impulse Stick by Immersion Corporation. The Laparoscopic Impulse Engine is a high fidelity simulation device for laparoscopic surgical procedures. A picture of the Laparoscopic Impulse Engine is shown in Figure 1. The Impulse Engine 2000 is a research quality force feedback joystick. It allows applications to track the movements of the user and convey high fidelity tactile sensations to the user through force feedback. The Impulse Engine 2000 is shown in Figure 2. The Impulse Stick is a robust arcade quality force feedback joystick suitable for industrial applications and high abuse environments, and is shown in Figure 3.

Figure 1: Laparoscopic Impulse Engine  Figure 2: Impulse Engine 2000  Figure 3: Impulse Stick
1.1 Capabilities of the Immersion Impulse Engine I/O Card

The configuration of your Impulse Engine ISA/PCI bus interface card depends on which Impulse Engine model you have. The number of input and output channels for an Impulse Engine 2000, Impulse Stick, and Laparoscopic Impulse Engine are:

<table>
<thead>
<tr>
<th>Model</th>
<th>Digital In</th>
<th>Digital Out</th>
<th>Encoder In</th>
<th>Analog Out</th>
<th>Analog In</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impulse Engine 2000</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Impulse Stick</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Laparoscopic Impulse Engine</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

These I/O capabilities can all be accessed in software using the functions described in the Library Reference later in this chapter.

The D/A output and encoder readings must be enabled using their respective enable functions before they will work. The D/A outputs will float upon powerup until they are enabled, after which they will output 2.5 volts.

If velocity data is needed with the Impulse Engine 2000 / Impulse Stick, the Laparoscopic Impulse Engine ISA or PCI can be purchased as an option. Please contact Immersion Corporation for more information.

IETEST. EXE is a DOS program provided to test the individual functions of your Impulse Engine and I/O card. There are two versions, one for the ISA interface card, and a different one for the PCI interface card; please check that you have the correct DOS development software for your interface.

1. The Digital In function allows you to read digital inputs such as switches and pedals. If you purchased the flightgrip option, you can test the handle buttons with this feature.
2. The Digital Out function is used to set digital outputs to control an external device. One bit of digital input is accessible inside the housing on the Impulse Engine.
3. The DAC function controls the torque command to the motors. It is a linear scale from 0 to 4095, with 0 being full on in one direction, 4095 full in the other direction, and 2048 being off. The two motors can be tested by selecting channel 0 for the X axis, and channel 1 for the Y axis. Use the +,8,9 keys on your numeric keypad to vary the force in the positive direction, and the -,2,3, keys to vary the force in the negative direction. The 5 key zeros the force at 2048.
4. The Encoder function can be used to read the values directly from the encoders. Upon start up, all of the encoder channels present in your device (see table above) should read 0, and the other channels should read -1. These channels should count up and down as you move the device throughout its range of motion.
1.2 ISA Card Addressing (not applicable to PCI interfaces)

1.2.1 Base Address

The I/O card’s default base address is 0x2C0 and can be changed using the five DIP switches. Each switch is numbered on the I/O card.

- A switch setting of ‘On’ corresponds to a ‘0’
- A switch setting of ‘Off’ corresponds to a ‘1’

If Switch #5 is On, the base address is in the two hundred hex range (i.e., 0x200 - 0x2FF). If Switch #5 is Off, the base address is in the three hundred hex range (i.e., 0x300 - x3FF).

Switches #4 - #1 correspond to the next hex digit. The last digit is always 0.

Examples:

<table>
<thead>
<tr>
<th>#4</th>
<th>#3</th>
<th>#2</th>
<th>#1</th>
<th>#5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
<td>1100 - C-&gt;0x2C0</td>
</tr>
<tr>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>0000 - 0-&gt;0x300</td>
</tr>
<tr>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>1111 - O-&gt;0x7F0</td>
</tr>
</tbody>
</table>

These addresses are defined in the configuration files *ie2000.dat*, *ie3000.dat*, and *ietest.dat*.

PCI cards do not have base address switches. Instead the base address is assigned by Windows during system startup. The addresses used can be viewed in the Device Manager’s Resource property page for the device entry, which will have a title “PCI Impulse Engine Device”.

1.2.2 Addressing Two Impulse Engines or Two Impulse Sticks

If you have purchased a two Impulse Engine / Impulse Stick system, the first card’s default address will be factory set to 0x2C0, and the second card’s default address will be 0x300. The configuration file *ie2000.dat* defines these addresses for use in the *ie2000* demo program and the file *ietest.dat* defines the addresses for the *ietest* test program. Both of these programs start up using the first card as the default, but can be toggled between them by using the Toggle menu function.

If you wish to use these boards in different computers with the same software, you may want to change the second board’s address back to the default of 0x2C0, so that both boards are the same.

Windows automatically assigns two or more PCI cards to different base addresses, which do not overlap the standard port ranges for ISA Impulse Engine cards.
2 Installation Instructions

This section describes how to install the hardware and software components of your Impulse Engine or Impulse Stick. If you need further assistance, please call Immersion Corporation at (408) 467-1900.

2.1 System requirements

- Pentium 120 or faster processor
- 16 Mb of RAM
- 32 Mb recommended for Windows applications
- 5 Mb Disk Space
- CD-ROM
- ISA or PCI slot
- Microsoft Visual C++ 5.0 or later (for DOS or Windows programming)
- Borland C++ 4.2 or later (for DOS programming)

2.2 Components and Accessories

The various parts of the Impulse Engine 2000 and Impulse Stick system are:

**System Components:**
The Impulse Engine 2000 or Impulse Stick Hardware (shown in Figure 2 or Figure 3)
The I/O Card (ISA or PCI)

**Accessories included:**
Interface Cable: HDB15 for ISA systems, HDB1526 for PCI systems
PS12-U Power Supply
IEC-xx Power Cable for your country

The various parts of the Laparoscopic Impulse Engine system are:

**System Components:**
The Laparoscopic Impulse Engine Hardware and all motor, encoder, and tool cables
The Interface Box (Amplifiers and power circuitry)
The I/O Card (ISA or PCI)

**Accessories included:**
HDB-26 Cable (connects interface box to I/O card)
PS12-U Power Supply
IEC-xx Power Cable for your country

2.3 Hardware Installation

1. Insert the I/O card into the appropriate slot of an IBM compatible personal computer. Be sure to visually inspect the card for any damage before inserting it into your computer.

2. Connect the interface cable to the I/O card and to the Impulse Engine or Impulse Stick.

For DOS systems:
3. Insert the included software disk, select the correct \DOS\XXX directory for your interface card. Run the demo software ietest.exe, before powering up the joystick. Select #5, Encoder, and move around the device to verify that the encoder positions are being updated correctly. You may also select #1, Digital In, to read the state of the buttons.

4. Ensure that the power switch of the Impulse Engine or Impulse Stick is in the “0” (off) position before plugging in the power supply. Power up the joystick while holding the handle. The red LED on the top panel (IE 2000) or front panel (Impulse Stick, Lap IE power module) will light, indicating that the system is powered. The green LED will indicate when the motors are enabled. Select #4, Analog Out, to test the motors. You will need to enter the motor number to test (0 or 1), then use the numeric keypad to increment or decrement the output values (-2048 to 2048). If you have two interface boards, #6 will toggle between the two.

For Windows system:

3. Install the correct device driver for your operating system and interface card as described in Section 2.4

4. Run the test program, ImpTest.exe, before powering up the joystick (located in \Windows\TestApps\bin).

5. If you have more than one Impulse Engine interface board (ISA or PCI) installed in the computer, you can select the board you want to test with the Choose Device pull down.

6. The program automatically polls all of the encoders and digital inputs. If you move the device or hold down one of the buttons, you should see the values change. You can use the Set Center button to zero the values of the encoders at the center of the workspace. If you have a Laparoscopic Impulse Engine or purchased the ‘velocity’ ISA or PCI card option, you should see the velocity values also change.

7. Ensure that the power switch on the Impulse Engine or Impulse Stick is in the “0” (off) position before plugging in the power supply. Make sure that the two DAC values are 0. Power up the joystick while holding the handle. The red LED on the top panel (IE 2000) or front panel (Impulse Stick, Lap IE power module) will light, indicating that the system is powered. The green LED will indicate when the motors are enabled.

8. Check the ‘Enable Spring’ box, and verify that there is a centering spring on the device. The spring force should be proportional to the displacement of the handle. Alternatively, you may enter values from -2048 to 2048 in the DAC fields (only first two fields apply to the IE2000), then select ‘Output’ to send them directly to the device.

Your system is now checked out and ready to run. Enjoy your system.
2.4 Software Installation for Windows Drivers

The following procedures apply to Impulse Engine drivers version 4.0 or later. Please ensure that your computer is powered off before installing or removing any interface boards.

2.4.1 NT 4

Under NT4, Imp.sys and Impulse.dll are used. These are included in the SDK’s “Windows\Drivers\NT4” directory. Since NT4 has no “Add Hardware” wizard, the installation procedure is pretty manual, and only needs to be done once, regardless of how many devices you have in the computer.

2.4.1.1 Installation (NT4)

1. Insert an ISA and/or PCI card into your system.
2. Copy imp.sys to <winnt>\system32\drivers.
3. Copy impulse.dll to <winnt>\system32.
4. In the Explorer view, double click on the icon for the appropriate reg file: Use ImpPci.reg if you only have PCI cards. Use Implsa-IE2000.reg if you have an Impulse Engine 2000 or Impulse Stick with an ISA card, or multiple systems with ISA and PCI cards. The file Implsa-LapIE.reg is used for the Laparoscopic Impulse Engine using an ISA card, or a combination of ISA and PCI cards.
5. Restart the computer.
6. To confirm the success of the installation, open the Control Panel’s "Devices" applet. The name "Imp" should appear in the list, and the word “Started” should appear in the device status column. See section 2.3 to test the system.

2.4.1.2 Removal (NT4)

1. Run the regedit application. Click on "HKEY_LOCAL_MACHINE".
2. Use the Edit menu's Find tool to find the driver info. Enter "Imp" in the "Find what" edit box, and check "Match whole string only". Click OK.
3. The program will find the driver's service key, named "Imp". Delete the key.
4. The key you found is most likely under CurrentControlSet. Press the F3 key to find other similar keys in ControlSet001, ControlSet002, etc. Delete those also.
5. On each subsequent restart, the driver will not be loaded.
2.4.2 Windows 98

Under Windows 98, Impulse.sys and Impulse.dll are used. These are included in the “Windows\Drivers\Win98” directory. They must be installed once for each card you have, using the following procedures:

2.4.2.1 Installing ISA cards (Windows 98)

1. Insert an ISA card into your system.

2. Start the computer, and log on. Open Control Panel’s “Add New Hardware” wizard.

3. Click on the "Next" button twice to get the dialog "Is the device you want to install listed below?"

4. Click "No, the device isn't in the list", then "Next".

5. The system asks "Do you want Windows to search for your new hardware?" Click on "No", then on "Next".

6. Click on "Next" to start the search, then on "Next" to finish it.

7. The system prompts "Select the type of hardware you want to install". Choose "Other devices".

8. Click on the "Have Disk" button. Browse to your install directory. Select Impulse.inf. Click OK.

9. Choose the entry "ISA Impulse Engine Device", then click on "Next".

10. Check that the I/O range assigned by the system matches the address switches on your card. Click on OK.

11. The "Copying Files" popup may report that "The file impulse.dll (on Unknown) cannot be found". Click on OK to bypass this spurious error. The system will copy the file to the correct place.

12. The system then reports that "Windows has finished ...". Click on the "Finish" button.

13. The system pops up the "System Settings Change" dialog that asks "Do you want to shut down your computer now?" Click on "Yes" to shut the system down, and check your address switch settings.

14. After this, the driver will be loaded during each system startup.

15. To confirm the success of the installation, open the Control Panel's "System" applet. Click on the "Device Manager" tab. Under the "Other devices" node, there should appear an icon labelled "ISA Impulse Engine Device". Right click on this icon to get its properties. The properties info should indicate the device is working normally. See section 2.3 to test the system.

2.4.2.2 Installing PCI cards (Windows 98)

1. Insert a PCI card into your system.

2. Start the computer. The system will detect the newly installed card, and pop up the "New Hardware Found" display to report "PCI card found ...". If the system's users have never created user accounts, you will not be able to access your network to get the drivers. In this event, they must be installed from a floppy disk or the local disk. Some systems pop up the PCI card found dialog before the logon dialog; this restriction applies to those systems also.
3. Then the "Add New Hardware" wizard dialog appears. Click on the "Next" button.

3. Choose "Display a list of all the drivers ...", and click on the "Next" button.

4. Choose "Other devices", then click on "Next".

5. Click on the "Have Disk" button. Browse to your install directory. Select Impulse.inf. Click OK.

6. Choose the entry "PCI Impulse Engine Device", then click on "Next".

7. The "Copying Files" popup may report that "The file impulse.dll (on Unknown) cannot be found." Click OK to bypass this spurious error. The system will copy the file to the correct place.

8. The system then reports that "Windows has finished ...". Click on the "Finish" button.

9. After this, the driver will be loaded during each system startup.

10. To confirm the success of the installation, open the Control Panel's "System" applet. Click on the "Device Manager" tab. Under the "Other devices" node, there should appear an icon labelled "PCI Impulse Engine Device". Right click on this icon to get its properties. The properties info should indicate the device is working normally. See section 2.3 to test the system.

### 2.4.2.3 Removal (Windows 98)

1. Open the Control Panel's "System" applet.

2. Click on the "Device Manager" tab.

3. Click on the icon of the "ISA Impulse Engine Device" or the "PCI Impulse Engine Device" you want to remove.

4. Press the delete key.

5. If you are deleting all Impulse devices, delete all the inf files in the \<windows>\Inf\Other folder that have “IMPULSE” in their names.

5. On each subsequent restart, the driver will not be loaded for the deleted device.

### 2.4.3 Windows 95

Copy `vimpulse.vxd` and `Impulse.dll` to the Windows\System directory. No other installation procedure is needed.

### 2.4.4 Limitations

1. `Impulse.sys` and `Imp.sys` do not currently work on NT 5.

2. `Vimpulse.vxd` only supports ISA cards on Windows 95.
3 DOS Software Development Kit

This chapter describes the Software Development Kit for interfacing with the Impulse Engine 2000 or Impulse Stick in a DOS application. The section Impulse Engine Demo Source Files describes the high level demo program C source files of the Immersion Impulse Engine SDK. The section Impulse Engine Common Software Library describes the lower level C source files of the Immersion Impulse Engine SDK. Please use the example programs in the correct directory for your interface boards - DOS\ISA or DOS\PCI. Not all functions or files may exist in both directories.

3.1 Impulse Engine Demo Source Files

There are several demo files in the Dos/ISA folder. These demos can be recreated for use with the PCI interface by using those ieio functions instead of the ones in the ISA folder.

IE2000.EXE is an Impulse Engine / Impulse Stick demo program. The graphics device driver file EGAVGA.BGI must be in the same directory as the demo programs to run them. IE2000.EXE provides four samples of 2D force feedback sensations.

1. The Spring demo consists of 2 simulated springs that can be compressed from the left side. As they are compressed to the right, the user will feel a force pushing back to the left proportional to the displacement. A spring constant, K, for each spring defines the magnitude of this proportional force.
2. The Damping demo simulates a viscous field that resembles a thick liquid. The user is free to move through the field, but feels a resistive force proportional to the speed at which they are moving.
3. The Textures demo highlights some of the crisp sensations that can be experienced with the Impulse Engine. Each grid line can be felt as you move over it, similar to feeling the grooves on a washboard.
4. The Balls demo contains 3 balls of different diameters and different stiffnesses. As you try to enter the ball you will feel a force field attempting to keep you out of the center. This demonstrates varying levels of hardness that you can create for objects.

IE3000.EXE also provides two samples of 3D force feedback sensations for the Laparoscopic Impulse Engine.

1. The Laparo Spring demo consists of 4 simulated springs that can be compressed from above. As they are compressed down, the user will feel a force pushing back up proportional to the displacement. A spring constant, K, for each spring defines the magnitude of this proportional force. The user can also use the gripper to grab the springs and pull them up. Letting go with the gripper causes the spring to snap back.
2. The Laparo Holes demo contains 5 holes of different diameters in a tabletop surface. After entering one of the holes, you will feel a force field attempting to keep you inside.

The Immersion Impulse Engine DOS SDK consists of the following source code files. The first set of files contains the high level functions that perform all graphics and Impulse Engine device-specific routines. Use these as examples for writing your own haptic demos.

IE3DMain.cpp main() routine for IE3000/Lap
IE2DMain.cpp main() routine for IE2000, Impulse Stick
IE3DDemo.cpp High-level haptic 3D demo functions for IE3000/Lap
IE3DDemo.h Definitions for IE3DDemo.cpp
IE2DDemo.cpp  High-level haptic 2D demo functions for IE2000, Impulse Stick and IE3000/Lap

IE2DDemo.h  Definitions for IE2DDemo.cpp

IE3DFunc.cpp  IE3000/Lap-specific functions (kinematics, Jacobian)

IE2DFunc.cpp  IE2000, Impulse Stick-specific functions (kinematics, Jacobian)

IENDFunc.h  Definitions for IE3DFunc.cpp, IE2DFunc.cpp

IEFunc.cpp  Mid-level graphics and generic i/o card access functions for all Impulse Engine devices

IEFunc.h  Definitions for IEFunc.cpp

IEGlobs.cpp  Global variables used in IE demos

IEExtrns.h  Extern variables used in IE demos

IEDefs.h  '#define's in IE demos

IE3000.dat  User configuration file for IE3000/Lap haptic demo gains and parameters. Set the variable Device equal to 0 for the IE3000 or equal to 1 for the IE Laparo.

IE2000.dat  User configuration file for IE2000, Impulse Stick haptic demo gains and parameters. If the base address of your IOC card is not the default 0x2C0, you must change this file to run ie2000.exe or demo2000.exe properly.

The second set of files are part of Immersion’s general software library and contain all the low level i/o card interface, file i/o, timing, and compiler-specific functions:

ieio.c  Low-level functions for interfacing with Immersion's Impulse Engine I/O Card 2.0. Includes all function calls necessary to perform digital I/O, D/A output, encoder readings, and A/D conversions. The D/A output and encoder readings must be enabled using their respective enable functions before they will work. The D/A outputs will float upon powerup until they are enabled, after which they will be output zero (0.0) volts.

ieio.h  Definitions for ieio.c.

readcfg.c  Functions allowing run-time user modification of haptic demo gains and parameters, which are saved in the configuration file ie2000.dat and ie3000.dat.

readcfg.h  Definitions for readcfg.c

timefunc.c  Functions for setting up timed interrupt service routines for haptic servo routines.

timefunc.h  Definitions for timefunc.c

compiler.h  Compiler-specific definitions

ietest.cpp  Test functions of the various I/O capabilities
ietest.dat  User configuration file for the ietest diagnostic tool. If the base address of your IOCard(s) is not the default 0x2C0, you must change this file to run ietest.exe properly.
3.1.1 Compiling and Linking:

To build your own DOS based Immersion Impulse Engine application:

• Write source code for use with the Immersion Impulse Engine SDK using the low-level functions in ieio.cpp.

• Be sure to #include the file ieio.h in your source code. If you are coding in C++, you will need to either C- 
  compile ieio.c or indicate to the compiler that ieio.c is not a C++ file (e.g. in Borland: extern “C”
  (#include “ieio.h”).

• Compile ieio.c, readcfg.c, timefunc.c, and your own source code and link the resulting object files 
together to produce an executable file.

• If you are using the Borland Graphics Library, be sure to #include the file graphics.h and link in the BGI 
  library as well.

To build the Immersion Impulse Engine ie2000 application,

• Compile IE2DMain.cpp, IE2DDemo.cpp, IE2DFunc.cpp, IEFunc.cpp, IEGlobs.cpp, 
ieio.c, readcfg.c, and timefunc.c as a DOS application. Link the resulting object files and the Borland 
  BGI library together to produce a DOS executable file.

To build the Immersion Impulse Engine ie3000 application,

• Compile IE3DMain.cpp, IE3DDemo.cpp, IE2DDemo.cpp, IE3DFunc.cpp, IEFunc.cpp, 
  IEGlobs.cpp, ieio.c, readcfg.c, and timefunc.c as a DOS application. Link the resulting object 
  files and the Borland BGI library together to produce a DOS executable file.

3.1.2 Demo Programs and Borland C/C++ Compiler

The Immersion Impulse Engine DOS demo programs are written mainly in C, with the exception of some C++ style 
comments (/\... vs /*... */ ) and some calls to the C++ <iostream.h> operator << (cout <<.. vs. 
printf(...)). The programs were compiled in the Borland C++ for DOS and Windows environment. While 
almost all the code is portable to other C or C++ compilers, the calls to Borland’s graphics library are specific to 
Borland C++ and will not compile properly with other compilers. Furthermore, to run the demo programs, the graphics 
device driver file EGAVGABGI (included with the SDK) must reside in the same directory as the demo program.

3.1.3 Demo Configuration

At runtime, IE2000.EXE and DEMO2000.EXE read all the configuration and gain parameters in IE2000.DAT. 
IE3000.EXE reads all the configuration and gain parameters in IE3000.DAT. The user can experiment by 
changing the parameters in these data files, but doing so can affect the performance of the demos. If the base 
address of your IOCard is not the default 0x2C0, you must change the Card1Addr and/or Card2Addr values this file 
to the demo programs properly. There is also a screen scaling factor that may need to be adjusted to compensate for 
the different resolution and range of travel in the IE2000 and Impulse Stick.
3.2 Immersion Impulse Engine Common Software Library

The following pages contain descriptions of various functions that can be used to develop your Impulse Engine applications. Similar functions are grouped into a C file and header file for modularity and ease of use.

`IEIO.H` and `IEIO.C` are the source code files to interface with Immersion's Impulse Engine I/O Card. They include all function calls necessary to perform digital I/O, D/A output, encoder readings, and A/D conversions. The D/A output and encoder readings must be enabled using their respective enable functions before they will work. The D/A outputs will float upon powerup until they are enabled, after which they will output zero (0.0) volts.

`COMPILER.H` is a header file that ensures compatibility of the Impulse Engine SDK with compilers from Borland, Watcom, and Microsoft. If you use a different compiler, please contact Immersion for assistance.

These files and others in the Immersion Common Software Library are described in more detail below. Please feel free to contact Immersion Corporation for additional application information.

3.2.1 COMPILER.H

Overview

This file is used to handle the different hardware access functions that are used by different compilers. Its purpose is to make as much of the remaining source code as possible compiler independent.

Notes/Limitations

This file currently supports the Borland, Microsoft, Watcom and Symantec compilers.

Definitions

- `INPB` – The function to read a byte from an I/O port
- `INPW` – The function to read a word from an I/O port
- `OUTPB` – The function to write a byte to an I/O port
- `OUTPW` – The function to write a word to an I/O port
- `DISABLE()` – The function to disable interrupts
- `ENABLE()` – The function to enable interrupts
- `SETVECT` – The function to set an interrupt vector
- `GETVECT` – The function to read an interrupt vector
- `ISR_FUNC` – The return type for an interrupt response function
- `ISR_PARAMS` – The parameters for an interrupt response function

Types

- `PISR_FUNC` – A pointer to an interrupt response function (the type for the function in calls to GETVECT and SETVECT)
3.2.2 IEIO.C and IEIO.H

Overview

These are the functions for low level access to the ISA or PCI cards that are used to control the impulse engines. Be sure to link in the files from the correct directory (DOS/ISA or DOS/PCI) for your interface card.

Notes/Limitations

The stIEIOCard structure must be allocated in the application and only a pointer to it sent to the functions.

Definitions

DAC_CHANNELS – The number of possible digital to analog outputs
ENC_CHANNELS – The number of possible encoder channels
ADC_CHANNELS – The number of possible analog to digital inputs
CMASK_X – The masks used to identify channels (X is the channel number)
CMASK_ALL_DAC – The mask to identify all the DAC channels
CMASK_ALL_ENC – The mask to identify all the ENC channels
DEFAULT_DAC_ZERO – The value that is the default 0 volt output for the DACs

Types

stIEIOCard – This is the type used to access a card. Its members should not be accessed directly.

Functions

void IEIO_Init(stIEIOCard *card, unsigned int baseAddress);
This is called to initialize the card for use at the given base address. The parameters are a pointer to the card structure to use and the base address for the card. If the baseAddress is set to 0, the card will be used at the default base address.

void IEIO_End(stIEIOCard *card);
This is called to terminate the usage of the card. It will disable all the DACs and encoder channels that are left on and terminate the usage of the card.

int IEIO_DigIn(stIEIOCard *card);
This will read the 8 bit digital input for the given card and return the value.

int IEIO_DigOut(stIEIOCard *card, int x);
This will write the 8 bit digital output for the given card. It will modify all 8 bits of the port. The return value is the value that was written.

void IEIO_EnableDAC(stIEIOCard *card, unsigned int dacMask, int value);
This will enable the DAC channels that are specified and write the given value to them. The channels for this function are supplied as bitmask values (CMASK_XX or CMASK_ALL_DAC) so that multiple channels may be handled at the same time. This command will only affect the given channels and leave the other channels unmodified.
void IEIO_DisableDAC(stIEIOCard *card, unsigned int dacMask);
This will disable the DAC channels that are specified. The channels for this function are supplied as bitmask values (CMASK_XX or CMASK_ALL_DAC) so that multiple channels may be handled at the same time. This command will only affect the given channels and leave the other channels unmodified.

int IEIO_DACOut(stIEIOCard *card, int dacNum, int value);
This will write the supplied value to the given channel on the given card. The functions assumes the channel has already been enabled. Also, if IEIO_DACOffset has already been called, the value is offset by this amount before it is written. The actual value written to the port is returned.

void IEIO_DACOffset(stIEIOCard *card, int dacNum, int value);
This will set up an offset of writes to the specified DAC channel. This offset will not be applied until the next time the channel is written.

void IEIO_EnableEnc(stIEIOCard *card, unsigned int encMask);
This will enable the encoder channels that are specified. The channels for this function are supplied as bitmask values (CMASK_XX or CMASK_ALL_ENC) so that multiple channels may be handled at the same time. This command will only affect the given channels and leave the other channels unmodified.

void IEIO_DisableEnc(stIEIOCard *card, unsigned int encMask);
This will disable the encoder channels that are specified. The channels for this function are supplied as bitmask values (CMASK_XX or CMASK_ALL_ENC) so that multiple channels may be handled at the same time. This command will only affect the given channels and leave the other channels unmodified.

long IEIO_EncInRaw(stIEIOCard *card, int encNum);
This will read the raw encoder value from the specified channel on the specified card from the encoder chip and return it to the application.

long IEIO_EncIn(stIEIOCard *card, int encNum);
This will do a processed read from the specified encoder on the specified card. The read will handle the encoder wrap around and account for the encoder zero position. The processed encoder value is returned.

void IEIO_EncZero(stIEIOCard *card, int encNum, long value);
This will set the zero position for the specified encoder channel on the specified card. The value parameter should be a returned value from the IEIO_EncInRaw function.

void IEIO_StartADC(stIEIOCard *card, int adcNum);
This will start the analog to digital conversion for the specified channel on the given card.

int IEIO_ADCIn(stIEIOCard *card, int adcNum);
This will read the analog value from the given channel on the card. This call will block until the conversion is complete or times out.

int IEIO_ADCBusy(stIEIOCard *card, int adcNum);
This function is used to determine if the converter has finished the conversion for the specified channel.
3.2.3 READCFG.C and READCFG.H

Overview

These functions provide a method to read a configuration file for runtime data. This will allow the file to be modified and the performance of the program to be changed without recompiling the source code.

Definitions

RCE_APPLY_DEFAULTS – This is specified as a flag in the call the ReadCfgExecute if the default values are to be applied to all the entries before they are read from the file.

Functions

int ReadCfgAddInt(char *name, int *var, int deflt);
This will add an integer value to list. The name is the string that is used to reference the value in the configuration file. The variable to hold the data is referenced by var. The default value for the variable is the last parameter.

int ReadCfgAddLong(char *name, long *var, long deflt);
This will add a long value to list. The name is the string that is used to reference the value in the configuration file. The variable to hold the data is referenced by var. The default value for the variable is the last parameter.

int ReadCfgAddFloat(char *name, float *var, float deflt);
This will add a float value to list. The name is the string that is used to reference the value in the configuration file. The variable to hold the data is referenced by var. The default value for the variable is the last parameter.

int ReadCfgExecute(char *filename,unsigned int flags);
This will execute the configuration read from the specified file. It returns the number of items that were matched in the file or a negative value if the file could not be opened. The flags will determine if the defaults should be applied to the variables before the file is read.

void ReadCfgPrint(char *filename);
This will print the current state of the configuration table variables. The output will go to the specified file unless the filename is NULL in which case the output will go to the screen.

void ReadCfgClear(void);
This will clear the configuration table entries. After a call to this function, the ReadCfgAddXXX functions can be used to create a new table and ReadCfgExecute can be called again.
3.2.4 TIMEFUNC.C and TIMEFUNC.H

Overview

These functions enable the application to run a single function at the given frequency.

Notes/Limitations

Only one function in an application can be setup this way.
If the frequency is too large, the program will crash.
Dos and graphics calls should not be made in the function
When this is being used, standard time functions on the PC will not work.

Functions

`int TF_Start(void (*func)(void), float freq);`
This will start the function executing at the given frequency. The function should take no parameters and return nothing.

`int TF_End(void);`
This will end the periodic execution of the
4 Windows 95, 98, NT Development Environment

4.1 INTRODUCTION

This section describes the interface to the Impulse Engine Dynamic-Link Library Impulse.dll. Using this library, Win32 programs are able to read the position, digital and analog sensors and control the digital output and motor current on the impulse device. Velocity encoder values can also be read if the card has this feature. The DLL opens, uses and closes the appropriate driver file for the operating system on which the program is running. The functions exported by the DLL are described next followed by directions on using the DLL.

4.2 DEFINITIONS

These definitions are contained in the header file, impulse.h.

**IMPULSE_DEFAULT_ADDRESS**
The default base address value for the ISA impulse device controller card.

**IMPULSE_DEFAULT_VEL_DIVIDE**
The default value of the divisor variable used in the velocity-computing scheme.

**IE_DRIVER**
Code to select the driver when using the Imp_GetInfo function.

**IEDEVICE_1**
Code to select the first ISA card in API functions. If there is no ISA card installed, The DLL will substitute the first unassigned PCI card.

**IEDEVICE_2**
Code to select the second ISA card.

**IEDEVICE_N**
Code to select the n'th card in the DLL's device list, i.e. use IEDEVICE_N+0 to select card 0.

**IE_MAXIMUM_DEVICES**
Maximum number of ISA or PCI cards.

These describe the maximum number of channels supported by the DLL.

**IE_MAX_ADC_NUM**
Maximum number of Analog Input (ADC) channels.

**IE_MAX_ADC_MASK**
Mask to select all ADC channels.

**IE_MAX_DAC_NUM**
Maximum number of Analog Output (DAC) channels.
IE_MAX_DAC_MASK  
Mask to select all DAC channels.

IE_MAX_DIG_IN_NUM  
Maximum number of digital input channels.

IE_MAX_DIG_IN_MASK  
Mask to select all digital input channels.

IE_MAX_DIG_OUT_NUM  
Maximum number of digital output channels.

IE_MAX_DIG_OUT_MASK  
Mask to select all digital output channels.

IE_MAX_ENC_NUM  
Maximum number of encoder channels.

IE_MAX_ENC_MASK  
Mask to select all encoder channels.

These describe error conditions reported by the DLL.

IE_SUCCESS  
API functions return this value when they succeed.

IE_NO_DEVICES  
Returned when there are no devices attached.

IE_INVALID_DEVICE  
Returned when the selected device is not attached.
4.3 FUNCTIONS

The primary interface exported by Impulse.dll is a version of that previously developed for the NT 4.0 Impulse driver. The interface has been enhanced to support use of PCI Impulse Engine cards and velocity encoders. The DLL now can operate in NT 4.0 and Windows 98 to control both PCI and ISA cards, and in Windows 95 to control ISA cards. Impulse.dll also supports the previous Windows 95 interface, which is described in Impulse.doc.

During its initialization, the DLL queries the driver to build a list of installed cards. Cards are then identified in function calls by a selector, for example IE_DEVICE_1. IEDEVICE_1 and IEDEVICE_2 refer to the first and second ISA cards, which have base addresses in the ranges 0x2XX and 0x3XX, respectively. If these cards are not present, the first and second PCI cards installed will be substituted, for the convenience of older apps. New apps can select cards by their position in the device list maintained by the DLL. To do so, they should form a selector by adding the card index to IEDEVICE_N. Thus IEDEVICE_N+2 selects the third installed card. If any ISA cards are installed, they will be stored first in the DLL’s device list.

This interface has functions to manipulate both single and multiple channels. Imp_AdcInput is a function that accesses a single ADC by its zero-based index, and Imp_AdcMultiInput is the corresponding multiple input function that accesses one or more ADC’s with a select mask. Programs that use the multiple channel functions should allocate at least as many values to receive or supply data as the addressed card actually has. The programs can do this by allocating a large number of variables in an array, say IE_MAX_VALUES, but a better way is to call the Imp_GetInfo function to find out how many DAC channels the card has, for example.

Multi-channel functions only read or write the 0-based array elements selected by the mask passed in the calls. For example, if the mask value 0x5 is passed, Imp_EncMultiInput will only overwrite array elements 0 and 2 with data from the device, and Imp_DacMultiOffset will only read array elements 0 and 2 to pass data to the device.

Almost all functions in the new interface return status/error codes rather than data values, to improve the reliability of operations. The most common error code returned will be IE_INVALID_DEVICE, which is returned when the device selector passed to a function does not match the index of any device in the device list. The usual way this can happen is that the registry information to describe the interface cards does not match the currently installed set of cards. See section 5 for more information on the registry setup for the interface cards.

The following functions are available through the use of the Impulse Engine DLL:

4.3.1 Device Initialization / End Functions

UINT Imp_Initialize(BOOL OpenDevices);

Description
Initialize impulse.dll on application startup. This must be the first API call used in any program.

Parameters
OpenDevices
Flag to control whether the call opens connections to attached devices.

Returns
Standard error codes listed above.
UINT Imp_End(void);

Description
Shutdown Impulse.dll on application termination

Parameters
None

Returns
Standard error codes listed above.

UINT Imp_InitDevice(UINT Card);

Description
This is the initialization function for an impulse device. It is used to setup the device to a default state and to disable the power output. It will also reset the encoder counts for the axes.

Parameters
Card
Device selector.

Returns
Standard error codes listed above.

UINT Imp_EndDevice(UINT Card);

Description
This is the termination function that is used to shutdown the card. It must be called at the end of the application to return the device to the disabled state.

Parameters
Card
Device selector.

Returns
Standard error codes listed above.

UINT Imp_OpenDevice(UINT Card);

Description
Unused NT4 legacy function.

Parameters
UINT Imp_CloseDevice(UINT Card);

Description
Unused NT4 legacy function.

Parameters
Card
Device selector.

Returns
Standard error codes listed above.

UINT Imp_GetInfo(UINT Sel, PDEVICE_INFO Info);

Description
Get information about the device driver, or about a specific device.

Parameters
Sel
This is a selector for the type of information requested.
Info
DEVICE_INFO output data structure defined in Impulse.h.

Returns
Standard error codes listed above.

UINT Imp_GetStatus (UINT Card, PDEVICE_STATUS Status);

Description
Get status of device hardware.

Parameters
Card
Device selector.
4.3.3 Analog Input Functions

BOOL Imp_AdcBusy(UINT Card, UINT AdcNum);

Description
Busy test one ADC.

Parameters
Card
Device selector.
AdcNum
ADC channel number.

Returns
Standard error codes listed above.

UINT Imp_AdcMultiBusy(UINT Card, UINT AdcMask, PBOOL Results);

Description
Busy test one or more ADC’s.

Parameters
Card
Device selector.
AdcMask
This is a bitmask that specifies the ADC channels to disable.
Results
Array to receive the busy state of selected ADCs.

Returns
Standard error codes listed above.

UINT Imp_AdcInput(UINT Card, UINT AdcNum, PINT Result);
Input from one ADC synchronously. It is important that you use Imp_AdcBusy to check the busy flag before reading the value. Otherwise you may read invalid data. See ImpTest sample code.

**Parameters**

*Card*
Device selector.

*AdcNum*
ADC channel number.

*Result*
This is the pointer to the int where the ADC value will be stored

**Returns**
Standard error codes listed above.

---------------------------------------------

UINT Imp_AdcMultiInput(UINT Card, UINT AdcMask, PINT Results);

**Description**
Input from one or more ADC's synchronously. It is important that you use Imp_AdcMultiBusy to check the busy flag before reading the value. Otherwise you may read invalid data. See ImpTest sample code.

**Parameters**

*Card*
Device selector.

*AdcMask*
This is a bitmask that specifies the ADC channels to disable.

*Results*
Array to receive the input values from selected ADCs.

**Returns**
Standard error codes listed above.

---------------------------------------------

UINT Imp_AdcStart(UINT Card, UINT AdcNum);

**Description**
Start one ADC conversion.

**Parameters**

*Card*
Device selector.

*AdcNum*
ADC channel number.

**Returns**
Standard error codes listed above.
UINT  Imp_AdcMultiStart(UINT Card, UINT AdcMask);

Description
Start one or more ADC conversions.

Parameters
Card
Device selector.
AdcMask
ADC channel select mask.

Returns
Standard error codes listed above.

-----------------------------------------------------------------------------------------------------------------------------

4.3.4 Analog Output Functions

-----------------------------------------------------------------------------------------------------------------------------

UINT  Imp_DacDisable(UINT Card, UINT DacNum);

Description
Disable one DAC.

Parameters
Card
Device selector.
DacNum
DAC channel number.

Returns
Standard error codes listed above.

-----------------------------------------------------------------------------------------------------------------------------

UINT  Imp_DacMultiDisable (UINT Card, UINT DacMask);

Description
Disable one or more DAC’s. After a call to this function, further calls to DAC output functions will ignore the disabled channels until the channels are reenabled.

Parameters
Card
Device selector.
DacMask
DAC channel select mask.

Returns
UINT  Imp_DacEnable(UINT Card, UINT DacNum);

Description
Enable one DAC.

Parameters
Card        Device selector.
Dacnum      DAC channel number.

Returns
Standard error codes listed above.

UINT  Imp_DacMultiEnable (UINT Card, UINT DacMask);

Description
Enable one or more DAC's. After this is called, calls to the DAC output functions will actually control the current and force of the device motors. Before this is called, such function calls are ignored.

Parameters
Card        Device selector.
DacMask     DAC channel select mask.

Returns
Standard error codes listed above.

UINT  Imp_DacOffset(UINT Card, UINT DacNum, INT Offset);

Description
Set the offset of one DAC.

Parameters
Card        Device selector.
DacNum      DAC channel number
Offset      Pointer to array of output fields

Returns
UINT Imp_DacMultiOffset(UINT Card, UINT DacMask, PINT Offsets);

Description
Set the offsets of one or more DAC's.

Parameters
Card
Device selector.

DacMask
DAC channel select mask.

Offsets
This is the pointer to the integer array that contains the offset values to write to the DACs. The array must be at least as long as the number of channels specified by the bitmask. The first value of the array (Offsets [0]) will set the offset of motor 0 and so on.

Returns
Standard error codes listed above.

UINT Imp_DacOutput(UINT Card, UINT DacNum, INT Value);

Description
Output to one DAC.

Parameters
Card
Device selector.

DacNum
DAC channel number

Value
Pointer to array of output fields

Returns
Standard error codes listed above.

UINT Imp_DacMultiOutput(UINT Card, UINT DacMask, PINT Values);

Description
Output to one or more DAC's. This will write values to the motor control DACs specified in the bitmask to control the force output.

Parameters
Card
Device selector.

DacMask
DAC channel select mask.

Values
This is the pointer to the integer array that contains the values to write to the DACs. The array must be at least as long as the number of channels specified by the bitmask. The first value of the array (Values[0]) will control motor 0 and so on. The values specified in the array are added to modifiable offsets. If using the default values of the offsets, the valid range for the array values is from -2048 to 2047 with 0 representing no force. The offsets can be specified by calling one of the DAC offset functions, but the valid range of the total output is from 0 to 4095.

Returns
Standard error codes listed above.

---------------------------------------------------------------

UINT Imp_AmpEnable(UINT Card);

Description
Enable DAC amps on a PCI card. For the convenience of legacy apps, the Impulse software turns on the amps whenever any DAC is enabled.

Parameters
Card
Device selector.

Returns
Standard error codes listed above.

---------------------------------------------------------------

UINT Imp_AmpDisable(UINT Card);

Description
Disable DAC amps on a PCI card. For the convenience of legacy apps, the Impulse software turns off the amps whenever no DAC is enabled.

Parameters
Card
Device selector.

Returns
Standard error codes listed above.

---------------------------------------------------------------

4.3.5 Encoder Input Functions

---------------------------------------------------------------

UINT Imp_EncDisable(UINT Card, UINT EncNum);

Description
Disable one encoder. If no other encoders are enabled, turn PCI amps off.

Parameters

Card
Device selector.

EncNum
Encoder channel number.

Returns
Standard error codes listed above.

UINT  Imp_EncMultiDisable (UINT Card, UINT EncMask);

Description
Disable one or more encoders. After a call to this function, further calls to encoder input functions will ignore the channels that are disabled.

Parameters

Card
Device selector.

EncMask
Encoder channel select mask.

Returns
Standard error codes listed above.

UINT  Imp_EncEnable(UINT Card, UINT EncNum);

Description
Enable one encoder. If PCI amps were not on, turn them on.

Parameters

Card
Device selector.

EncNum
Encoder channel number.

Returns
Standard error codes listed above.

UINT  Imp_EncMultiEnable(UINT Card, UINT EncMask);

Description
Enable one or more encoders. After this is called, calls to the encoder input functions will actually read the enable encoder channels in the device. Before this is called, encoder input function calls are ignored.
Parameters

Card
Device selector.

EncMask
Encoder channel select mask.

Returns

Standard error codes listed above.

-----------------------------------------------

UINT  Imp_EncInput(UINT Card, UINT EncNum, PINT Result);

Description

Input from one encoder.

Parameters

Card
Device selector.

EncNum
Encoder channel number.

Result
Pointer to output field.

Returns

Standard error codes listed above.

-----------------------------------------------

UINT  Imp_EncMultiInput(UINT Card, UINT EncMask, PINT Results);

Description

Input from one or more encoders. The zero point for the axes will depend on the position of the device when
the first call to this function is made so some form of a calibration function will probably be necessary.

Parameters

Card
Device selector.

EncMask
Encoder channel select mask.

Results
This is the pointer to the int array where the encoder values will be stored. The array must be at
least as long as the number of channels specified by the mask. The first value of the array
(Results[0]) will read encoder 0 and so on.

Returns

Standard error codes listed above.

-----------------------------------------------

UINT  Imp_EncRawInput(UINT Card, UINT EncNum, PINT Result);
**Description**  
Raw input from one encoder.

**Parameters**
- **Card**  
  Device selector.
- **EncNum**  
  Encoder channel number.
- **Result**  
  Pointer to input field.

**Returns**
Standard error codes listed above.

---

```c
UINT Imp_EncMultiRawInput(UINT Card, UINT EncMask, PINT Results);
```

**Description**
This is used to read the raw encoder values from the card. Use this function as part of a calibration routine to set the zero point of encoders.

**Parameters**
- **Card**  
  Device selector.
- **EncMask**  
  Encoder channel select mask.
- **Results**  
  This is the pointer to the int array where the encoder values will be stored. The array must be at least as long as the number of channels specified by the bitmask. The first value of the array (encs[0]) will read encoder 0 and so on.

**Returns**
Standard error codes listed above.

---

```c
UINT Imp_EncZero(UINT Card, UINT EncNum, INT Value);
```

**Description**
Set the zero point of one encoder.

**Parameters**
- **Card**  
  Device selector.
- **EncMask**  
  Encoder channel number.
- **Value**  
  Output value.
UINT  Imp_EncMultiZero(UINT Card, UINT EncMask, PINT Zeros);

Description
Set the zero points of one or more encoders. Use this as part of a calibration routine

Parameters
Card
Device selector.
EncMask
Encoder channel select mask.
Zeros
This is the pointer to the short array where the encoder zero values will be stored. The array must be at least as long as the number of channels specified by the bitmask. The first value of the array (Zeros [0]) will set the offset for encoder 0 and so on.

Returns
Standard error codes listed above.

UINT  Imp_EncVel( UINT Card, UINT  EncNum, PINT Value );

Description
Input from one velocity encoder. PCI cards and some ISA cards have this feature. The value returned is calculated by the following formula: Velocity = (Frequency/Divisor)/Magnitude. Frequency is the board clock frequency. ISA cards use a 10 Mhz clock, while PCI cards use a 20 Mhz clock. Divisor is the value set by calling Imp_SetVelDivide, and Magnitude is the input value in counts per cycle.

Parameters
Card
Device selector.
EncNum
Encoder channel number.
Value
Pointer to input field

Returns
Standard error codes listed above.

UINT  Imp_EncMultiVel( UINT Card, UINT  EncMask, PINT Values );

Description
Input from one or more velocity encoders.
38

Parameters

Card
Device selector.
Encmask
Encoder channel mask
Values
Pointer to array of output fields

Returns
Standard error codes listed above.

____________________________________________________________________________

UINT Imp_SetVelDivide(UINT Card, UINT ClkDiv);

Description
Set velocity encoder prescaler divisor. It must be called after the encoders have been enabled.

Parameters

Card
Device selector.
ClkDiv
This is the value used in the velocity-computing scheme of the card. It has a range of 0 - 255. A high value will enable the scheme to capture slower velocities but it will decrease its resolution. A low value will increase the resolution but decrease the ability to capture slow movements. Good results have been observed at the default value of 10.

____________________________________________________________________________

4.3.6 Digital Input / Output

____________________________________________________________________________

UINT Imp_DigInput (UINT Card, PUINT Result);

Description
Read digital input values.

Parameters

Card
Device selector.
Result
Input field

Returns
Standard error codes listed above.

____________________________________________________________________________

UINT Imp_DigOutput(UINT Card, UINT Value);
Description
Write digital output values.

Parameters
Card
Device selector.
Value
The value to write to the output.

Returns
Standard error codes listed above.
4.4 USAGE

4.4.1 Implementation

1) Create a variable of type UINT in your application through a declaration such as the following:

```c
UINT card;
```

and use this variable as the parameter for the DLL functions.

2) Make the call to `Imp_Initialize`, `Imp_InitDevice` and calls to the enabling functions to access the device. Close the device using the `Imp_EndDevice` function, and the API by calling the `Imp_End` function.

3) Use the macros found in `Impulse.h` as parameters to the DLL functions. Use the macros the following way:

```c
int Encoders[IE_MAX_ENC_NUM];
...
/* to read encoder 0 */
Imp_EncInput(card, 0, Encoders);
...
/* to read all encoders */
Imp_EncMultiInput(card, IE_MAX_ENC_MASK, Encoders);
```

4) Follow the instructions in section 2.4 to install the drivers on your system

4.4.2 Compilation

To access the functions in the DLL from your application follow these steps:

1) Include in your application the header files:

```c
#include "impulse.h"
```

2) Insert the file `impulse.lib` in your project, or call the Win32 functions `LoadLibrary` and `GetProcAddress` to access the functions named in `Impulse.h`.

Note: - `impulse.lib` is specific to Microsoft Visual C++
4.5 EXAMPLES

Two example programs are included in directory: \windows\DemoApps\bin:

BallDemo.exe is a Windows application that simulates three solid bubbles on the screen.

ImpTest.exe is a Windows application that allows real time display of encoder, digital, and analog (LapIE only) input data and the ability to output analog commands to control the motors. It also includes a simple spring function that applies a centering force to the device as it is moved around the workspace.

Each example program includes a Microsoft Visual Studio workspace which can be used to ensure that all of the files and libraries are attached correctly. The common header and library files are contained in the Include and Lib directories. Please see the source code for examples on how to use the DLL.

The following is the basic structure for these programs.

1) In the initialization for the program, do something similar to what follows:

```c
#include impulse.h

int EncIn[IE_MAX_ENC_NUM];
int EncZero[IE_MAX_ENC_NUM];
int Force[IE_MAX_ENC_NUM];
int j;
UINT device, result;

device = IE_DEVICE_1;

/* Initialize the API */
result = Imp_Initialize(FALSE);
if (result != IE_SUCCESS)
{
    printf( "Imp_Initialize() returned %X\n", result);
    return;
}

/* Initialize the stick */
result = Imp_InitDevice(device);
if (result != IE_SUCCESS)
{
    printf( "Imp_InitDevice() returned %X\n", result);
    return;
}

/* Enable */
Imp_EncMultiEnable(device, IE_MAX_ENC_MASK);
Imp_DacMultiEnable(device, IE_MAX_ENC_MASK);

/* Get the center position */
printf ("Center the stick and press a key\n");
getch();
```
// Read in twice to clear buffer
Imp_EncMultiRawInput(device, IE_MAX_ENC_MASK, EncZero);
Imp_EncMultiRawInput(device, IE_MAX_ENC_MASK, EncZero);
Imp_EncMultiZero(device, IE_MAX_ENC_MASK, EncZero);

2) Perform some application specific processing involving reading the encoders to determine device position, processing the data, and setting the DAC output values to adjust motor forces. See the ImpTest program for a simple centering spring example, and BallDemo program for a more complex example.

3) At the end of the program:

    /* Disable */
    Imp_DacDisable(device,IE_MAX_DAC_MASK);
    Imp_EncDisable(device,IE_MAX_ENC_MASK);

    /* End */
    printf ("Ending\n\n");
    Imp_EndDevice(device);

    // Shutdown API
    Imp_End();
5 Registry Settings for Windows Systems

5.1 NT4 – ISA Interface

These are the contents of the default registry settings created by Implsaa-xxx.Reg on an NT4 system:

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Imp]
"ErrorControl"=dword:00000001
"Start"=dword:00000002
"Type"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Imp\Parameters]

(IE2000 / Impulse Stick systems)

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Imp\Parameters\Device1]
"Encoders"=dword:00000002
"Adcs"=dword:00000000
"Dacs"=dword:00000002
"Digital Inputs"=dword:00000003
"Digital Outputs"=dword:00000001
"Model Name"="IE2000"
"Model Number"=dword:00007d0
"Port Base"=dword:000002c0

(Laparoscopic Impulse Engine systems)

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Imp\Parameters\Device1]
"Encoders"=dword:00000004
"Adcs"=dword:00000002
"Dacs"=dword:00000003
"Digital Inputs"=dword:00000003
"Digital Outputs"=dword:00000001
"Model Name"="Lap IE"
"Model Number"=dword:000007d0
"Port Base"=dword:000002c0

A "Start" value of 3 requires the driver to be started by the Control Panel Device Manager applet’s Start command before the test apps can be executed. Open the applet window, select (highlight) Imp in the device list, and click Start button. The applet will indicate success or failure. To start the driver automatically whenever your system is booted, change the "Start" value to 0.

The values for "Encoders" through "Digital Outputs" should match those shown in the Overview section. "Model Name" and "Model Number" should be set to the model you are using.

The "Port Base" value should match the jumper settings on your card.

If you have two cards installed, add a Device2 section like the Device1 section above, with values appropriate to the second card.
When you are done editing the file, close it, and double-click on its name in the explorer view to get the contents installed in the registry. Reboot your system if you changed the “Start” value.
5.2 NT4 – PCI Interface

These are the contents of the default registry settings created by ImpPci.Reg on an NT4 system:

REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Imp]
"ErrorControl"=dword:00000001
"Start"=dword:00000002
"Type"=dword:00000001

ImpPci.reg should be used during installation when you have only PCI cards.

5.3 Windows 98 – ISA Interface

These are the contents of the default registry settings created by installation of an ISA card on a Windows 98 system:

The “Hardware Key”:

[HKEY_LOCAL_MACHINE\Enum\Root\UNKNOWN\0001]
"Class"="UNKNOWN"
"Driver"="UNKNOWN\0002"
"Mfg"="Immersion Corp"
"HardwareID"="IeIsa_Dev0100"
"ClassGUID"="{4d36e97e-e325-11ce-bfc1-08002be10318}"
"DeviceDesc"="ISA Impulse Engine Device"
"ConfigFlags"=hex:04,00,00,00
"ForcedConfig"=hex:00,04,00,00,00,00,00,20,00,00,00,02,00,00,00,01,00,02,00,\n c0,02,c1,02,00,00,00,00,f0,ff,02,00,c0,02,c0,02,3f,01,00,00,00,00,00,00
"Capabilities"=hex:14,00,00,00

[HKEY_LOCAL_MACHINE\Enum\Root\UNKNOWN\0001\LogConfig]
"0"=hex:0a,04,00,00,00,00,00,00,2c,00,00,02,00,00,00,02,00,0c,00,00,00,00,\n 00,00,00,00,00,f0,ff,02,00,c0,02,c1,02,00,00,04,0f,f0,ff,02,00,00,03,01,03,\n 00,00,04,0f,00,00,00,00

The “Software Key”:

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Unknown\0002]

"DevLoader"="*ntkern"
"NTMPDriver"="Impulse.sys"
"InfPath"="IMMERS~2.INF"
"InfSection"="IeIsaInstall"
"ProviderName"="Immersion"
"DriverDate"="9-14-1998"
"DriverDesc"="ISA Impulse Engine Device"
"MatchingDeviceId"="IeIsa_Dev0100"
5.4 Windows 98 – PCI Interface

These are the contents of the default registry settings created by installation of a PCI card on a Windows 98 system:

The “Hardware Key”:

[HKEY_LOCAL_MACHINE\Enum\PCI\VEN_10B5&DEV_9050&SUBSYS_112210B5&REV_02]

[HKEY_LOCAL_MACHINE\Enum\PCI\VEN_10B5&DEV_9050&SUBSYS_112210B5&REV_02\BUS_00&D EV_10&FUNC_00]

"Capabilities"=hex:14,00,00,00
"HardwareID"="PCI\VEN_10B5&DEV_9050&SUBSYS_112210B5&REV_02,PCI\VEN_10B5&DEV_9 050&SUBSYS_112210B5,PCI\VEN_10B5&DEV_9050&REV_02&CC_FF00,PCI\VEN_10B5&DEV_9 050&CC_FF0000,PCI\VEN_10B5&DEV_9050&CC_FF00"
"CompatibleIDs"="PCI\VEN_10B5&DEV_9050&REV_02,PCI\VEN_10B5&DEV_9050,PCI\VEN _10B5&CC_FF0000,PCI\VEN_10B5&CC_FF00,PCI\VEN_10B5,PCI\CC_FF0000,PCI\CC_FF0 0"
"DeviceDesc"="PCI Impulse Engine Device"
"HWRevision"="002"
"Class"="UNKNOWN"
"Driver"="UNKNOWN\0001"
"Mfg"="Immersion Corp"
"ClassGUID"="{4d36e97e-e325-11ce-bfc1-08002be10318}"
"ConfigFlags"=hex:00,00,00,00

[HKEY_LOCAL_MACHINE\Enum\PCI\VEN_10B5&DEV_9050&SUBSYS_112210B5&REV_02\BUS_00&D EV_10&FUNC_00\LogConfig]

"0000"=hex:00,04,00,00,00,30,00,00,00,00,00,01,00,00,00,01,00,14,00,00,fc,be,ff,ff,be,ff,01,00,00,00,80,ff,ff,ff,ff,00,00,00,00,00,ff,ff,ff,ff,01,00,00,00,10,00,00,00,24,00,00,00,02,00,00,00,01,00,00,00,80,fc,ff,fc,00,00,00,00,80,ff,00,00,00,ff,ff,ff,ff,00,00,00,14,00,00,00,24,00,00,00,02,00,00,01,00,00,00,00,ff,ff,ff,ff,00,00,00,00,00,ff,ff,ff,ff,00,00,00,18,00,00,00,00,00,00,00,00

The “Software Key”:

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Unknown\0001]

"InfPath"="IMMERS-2.INF"
"InfSection"="IePciInstall"
"ProviderName"="Immersion"
"DriverDesc"="PCI Impulse Engine Device"
"MatchingDeviceId"="PCI\VEN_10B5&DEV_9050"
"DevLoader"="*ntkern"
"NTMPDriver"="Impulse.sys"
"DriverDate"="9-14-1998"

5.5 Windows 95 – ISA Interface

No registry keys are needed to use the DLL and driver on Windows 95