DYNAMIC ENGINEERING
150 DuBois St., Suite B/C Santa Cruz, CA 95060
831-457-8891
https://www.dyneng.com sales@dyneng.com
Est. 1988

IP-1553
BC, MT, RT & RTM

Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision 1p0
Corresponding Hardware: Revision 01
FLASH revision 0101

Ip1553

Dynamic Engineering
150 DuBois St., Suite B/C
Santa Cruz, CA 95060
831-457-8891

Table of Contents

INTRODUCTION 5

Driver Installation 6

Windows 10 Installation 6

Driver Startup 7

IO Controls 7

IOCTL_IP_HQT_GET_INFO **Error! Bookmark not defined.**
IOCTL_IP_HQT_SET_IP_CONTROL **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_IP_STATE **Error! Bookmark not defined.**
IOCTL_IP_HQT_WRITE_SYNC_WORD **Error! Bookmark not defined.**
IOCTL_IP_HQT_READ_SYNC_WORD **Error! Bookmark not defined.**
IOCTL_IP_HQT_LOAD_TX_TIME **Error! Bookmark not defined.**
IOCTL_IP_HQT_START_TX **Error! Bookmark not defined.**
IOCTL_IP_HQT_STOP_TX **Error! Bookmark not defined.**
IOCTL_IP_HQT_INIT_SAMPLE_COUNT **Error! Bookmark not defined.**
IOCTL_IP_HQT_START_RX **Error! Bookmark not defined.**
IOCTL_IP_HQT_STOP_RX **Error! Bookmark not defined.**
IOCTL_IP_HQT_READ_TIME **Error! Bookmark not defined.**
IOCTL_IP_HQT_RESET_FIFO **Error! Bookmark not defined.**
IOCTL_IP_HQT_READ_FIFO **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_STATUS **Error! Bookmark not defined.**
IOCTL_IP_HQT_REGISTER_EVENT **Error! Bookmark not defined.**
IOCTL_IP_HQT_ENABLE_INTERRUPT **Error! Bookmark not defined.**
IOCTL_IP_HQT_DISABLE_INTERRUPT **Error! Bookmark not defined.**
IOCTL_IP_HQT_FORCE_INTERRUPT **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_INT_ENABLES **Error! Bookmark not defined.**
IOCTL_IP_HQT_SET_INT_ENABLES **Error! Bookmark not defined.**
IOCTL_IP_HQT_READ_TIME_REGS **Error! Bookmark not defined.**
IOCTL_IP_HQT_READ_TIME_REGS **Error! Bookmark not defined.**
IOCTL_IP_HQT_WRITE_TIME_REGS **Error! Bookmark not defined.**
IOCTL_IP_HQT_SET_VECTOR **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_VECTOR **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_ISR_STATUS **Error! Bookmark not defined.**
IOCTL_IP_HQT_GET_BASE_CONTROL **Error! Bookmark not defined.**
IOCTL_IP_HQT_SET_BASE_CONTROL **Error! Bookmark not defined.**
IOCTL_IP_HQT_SET_STATUS **Error! Bookmark not defined.**

WARRANTY AND REPAIR 12

Service Policy 12
Support 12

Introduction

The Ip1553 driver is a Windows device driver for IP-1553 Industry-pack (IP) module from Dynamic Engineering. This driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The Ip1553 software package has two parts. The driver for Windows® 10/11 OS, and the User Application "UserAp" executable.

The driver is delivered electronically. The files supplied are installed into the client system to allow access to the hardware. The UserAp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

Sample Application

The application "userApp" (de_1553_user_app_console.c) demonstrates proper usage of library functions/operations for libipack, libip1553 and Holt API code. As previously mentioned, the Dynamic Engineering IP-1553 module is employed for demonstration purposes.

NOTE: To run the test with a loopback the user must start two instances of the application and choose the appropriate channels. From one application the user should start the channel as a listener first, and then start sending data over the other channel second. For example, RT, RT-MT, or MT should be started first, and then BC modes may be started second.

The test software can be ported to your application to provide a running start. It is recommended to port the Register tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The hardware manual defines the pinout, the bitmaps and detailed configurations for each feature of the design. The driver handles all aspects of interacting with the hardware. For added explanations about what some of the driver functions do, please refer to the hardware manual.

We strive to make a useable product. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us.

When the Ip1553 board is recognized by the IP Carrier Driver, the carrier driver will start the Ip1553 driver which will create a device object for the board. This in turn will start the Ip1553 channel driver (1 or 2 channels is added depending on the device). If more than one is found additional copies of the driver are loaded. The carrier driver will load

the info storage register on the Ip1553 with the carrier switch setting and the slot number of the Ip1553 device. From within the Ip1553 driver the user can access the switch and slot information to determine the specific device being accessed when more than one is installed.

Driver Installation

There are several files provided in each IP driver package. These files include .sys, .cat, .inf.

Please note: Your carrier driver may need to be updated to use the IP module. The list of IP modules is compiled along with the driver and due to signing requirements.

Public.h and IpPublic.h are C header files that define the Application Program Interface (API) to the driver. These files are required at compile time by any application that wishes to interface with the driver, but are not needed for driver installation. IpPublic.h is supplied with the carrier driver. Public.h. is supplied with UserAp.

Warning: The appropriate IP carrier driver must be installed before any IP modules can be detected by the system.

Windows 10/11 Installation

Copy the supplied system files to a folder of your choice.

With the IP hardware installed, power-on the host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an item for each IP module installed on the IP carrier. The label for a module installed in the first slot of the first PCIe3IP carrier would read ***PcieCar0 IP Slot A****.
- Right-click on the first device and select ***Update Driver Software***.
- Insert the removable memory device prepared above if necessary.
- Select ***Browse my computer for driver software***.
- Select ***Browse*** and navigate to the memory device or other location prepared above.
- Select ***Next***. The IpBis6Gpio device driver should now be installed.
- Select ***Close*** to close the update window.
- Right-click on the remaining IP slot icons and repeat the above procedure as necessary.

***** If the [***Carrier] IP Slot [x]*** devices are not displayed, click on the ***Scan for hardware changes*** icon on the Device Manager tool-bar.

Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

A handle can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system.

The interface to the device is identified using a globally unique identifier (GUID), which is defined in Public.h.

The *main.c* file provided with the user test software can be used as an example to show how to obtain a handle to an IpHQT device.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single module. IOCTLs are called using the function DeviceIoControl() (see below), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with CreateFile()
  DWORD         dwIoControlCode,  // Control code defined in API header file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned,  // Pointer to return length parameter
  LPOVERLAPPED  lpOverlapped,     // Optional pointer to overlapped structure
);                                //   used for asynchronous I/O
```

IP-1553 API

In this code base, Holt provides a standard API (located in the Holt directory of the UserApp) to access the holt components on the IP device. This library code has been ported to Windows to allow users an easier means of getting started with the use of the device. It is recommended that users look at the sample code to see how the Holt API functions.

To support the Holt API, a set of basic library functions has been added to provide the correct device access. This basic functionality can be found in the HoltApi.c/h files. These functions do not need to be called directly but instead can be invoked through the Holt API.

The API created to interface with the Holt API is listed below:

```
/*****************************************************************************
*  io_write_16
*
*  Write 1553 Chip or register on module
*
*  Parameters:
*  handle   -   Handle to channel.
*  base     -   area on the device being written (Module, Holt Chip Regs or RAM)
*  offset   -   register or memory offset
*  vals     -   16 bit value written
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
extern int io_write_16(HANDLE handl, MULTI_BASE_SEL base, UINT32 offset, UINT16 val);

/*****************************************************************************
*  io_read_16
*
*  Read 1553 Chip or register on module
*
*  Parameters:
*  handle   -   Handle to channel.
*  base     -   area on the device being read
*  offset   -   register or memory offset
*  vals     -   16 bit value read
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
extern int io_read_16(HANDLE handl, MULTI_BASE_SEL base, UINT32 offset, PUINT16 val);

/*****************************************************************************
*  io_read_16_multi
*
*  Read 1553 Chip or register on module
*
```

```
*  Parameters:
*  handle   -   Handle to channel.
*  base     -   area on the device being read
*  offset   -   register or memory offset
*  val_arr  -  pointer to array of values read (need to pass allocated memory)
*  count    -   number of successive reads
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
extern int io_read_16_multi(HANDLE handl, MULTI_BASE_SEL base, UINT32 offset, UINT16*
val_arr, UINT32 count);

/*****************************************************************************
*  io_write_16_multi
*
*  Read 1553 Chip or register on module
*
*  Parameters:
*  handle   -   Handle to channel.
*  base     -   area on the device being written
*  offset   -   register or memory offset
*  val_arr  -  pointer to array of values written
*  count    -   number of successive writes
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int io_write_16_multi(HANDLE handl, MULTI_BASE_SEL base, UINT32 offset, UINT16*
val_arr, UINT32 count);

/*****************************************************************************
*  io_rd_mod_wr
*
*  Read/modify/write 1553 Chip, only valid for register space
*
*  Parameters:
*  handle   -   Handle to channel
*  offset   -   register offset
*  mask     -   Mask specifying bits of interest
*  val      -   16 bit value to write.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int io_rd_mod_wr(HANDLE handl, UINT32 offset, UINT16 val, UINT16 mask);

/*****************************************************************************
*  lib_1553_reset
*
*  Reset 1553 chip/device
*
*  Parameters:
```

```
*  handl     - Handle to any channel as the channel reset depends on dev_num.
*  dev_num   - device/channel number (0 or 1)
*  enbl_ints - Enable interrupts for this device.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int ip_1553_reset(HANDLE handl, UINT32 DevNum, int enbl_ints);

/*****************************************************************************
*  io_carrier_write_32
*
*  Write to the carrier device through the IP device
*
*  Parameters:
*  handle   -   Handle to device.
*  offset   -   Carrier Offset
*  vals     -   32 bit value written
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int io_carrier_write_32(HANDLE handl, UINT32 offset, UINT32 val);

/*****************************************************************************
*  io_carrier_read_32
*
*  Read carrier device through the IP device
*
*  Parameters:
*  handle   -   Handle to channel.
*  offset   -   register or memory offset
*  vals     -   32 bit pointer to fill value read
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
extern int io_carrier_read_32(HANDLE handl, UINT32 offset, UINT32* val);

/*****************************************************************************
*  enable_carr_ints
*
*  Turns on int in the carrier, there are two channel ints 0 and 1, chan
*  determimes which to turn on.
*
*  Parameters:
*  handle   -   Handle to channel.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int enable_carr_ints(HANDLE handl);
```

```
/*****************************************************************************
*  enable_chan_ints
*
*  Turns on ints for each channel
*
*  Parameters:
*  handle   -   Handle to channel.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int enable_chan_ints(HANDLE handl);

/*****************************************************************************
*  disp_de_regs
*
*  Display the register values on the Xilinx
*
*  Parameters:
*  handle   -   Handle to channel.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern void disp_de_regs(HANDLE handl);

/*****************************************************************************
*  disp_de_regs
*
*  Display the register values for the carrier
*
*  Parameters:
*  handle   -   Handle to channel.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
extern void disp_carrier_regs(HANDLE handl);

/*****************************************************************************
*  force_int
*
*  Force interrupt for the IP device, must use the handle for the multi-device not the
channels
*
*  Parameters:
*  handle   -   Handle to device.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/

extern int force_int(HANDLE handl);
```

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
https://www.dyneng.com/warranty.html

Service Policy

The driver has gone through extensive testing, and while not infallible, problems experienced will likely be “cockpit error” rather than an error with the driver. We will work with you to determine the cause of the issue. If the effort is more than a quick conversation, we will offer a support contract. We can write updates to the driver to add features, create middleware etc.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite B/C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com