DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
831-457-8891 **Fax** 831-457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

IP Reflective Memory

16 Channel Optically Isolated Drivers

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision A
Corresponding Hardware: Revision F
10-2009-0206
FLASH revision C1

IP-REM-MEM

Dynamic Engineering
150 DuBois St., Suite C
Santa Cruz, CA 95060
831-457-8891
FAX: 831-457-4793

Table of Contents

Introduction

The IpRefMem driver is a Windows device driver for the IP-Test 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 IpRefMem driver package has two parts. The driver is installed into the Windows® OS, and the User Application "UserApp" executable.

The driver is delivered as installed or executable items to be used directly or indirectly by the user. The UserApp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

UserApp is a stand-alone code set with a simple, and powerful menu plus a series of "tests" that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system. The test suite is designed to accommodate up to 5 boards. The number of boards can be expanded. See Main.c to increase the number of handles.

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, and while we can guarantee operation we can't foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

IpRefMem has a Spartan2 Xilinx FPGA to implement the IP Interface, FIFO's protocol control and status for the IO. The main feature of the design is the memory array. The Hardware automatically clears the RAM and establishes the network. The main feature of the driver is to communicate with the RAM array. Writing to the RAM will automatically update the rest of the networked memory. Reading will retrieve the current value stored into memory. The driver also provides the ability to change the

hardware operation to use features other than the defaults.

When the IpReflMem board is recognized by the IP Carrier Driver, the carrier driver will start the IpRefMem driver which will create a device object for the board. If more than one is found additional copies of the driver are loaded. The carrier driver will load the info storage register on the IpRefMem with the carrier switch setting and the slot number of the IpRefMem device. From within the IpRefMem driver the user can access the switch and slot information to determine the specific device being accessed when more than one are installed.

The reference software application has a loop to check for devices. The number of devices found, the locations, and PLL information are printed out at the top of the menu.

IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move data in and out of the device.

Note

This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. For more detailed information on the hardware implementation, refer to the IpRefMem user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include IpRefMem.sys, IpRefMemPublic.h, IpPublic.h, WdfCoInstaller01009.dll, IpDevices.inf and IpDevices.cat.

IpRefMemPublic.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.

Note: Other IP module drivers are included in the package since they were all signed together and must be present to validate the digital signature. These other IP module driver files must be present when the IpRefMem driver is installed, to verify the digital signature in IpDevices.cat, otherwise they can be ignored.

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

Windows 7 Installation

Copy IpDevices.inf, IpDevices.cat, WdfCoInstaller01009.dll, IpRefMem.sys and the other IP module drivers to a removable memory device or other accessible location as preferred.

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 IpRefMem 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 IpRefMemPublic.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 IpRefMem 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 Win32 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
```

The IOCTLs defined for the IpRefMem driver are described below:

IOCTL_IP_REF_MEM_GET_INFO

Function: Returns the driver and firmware revisions, module instance number and location and other information.
Input: None
Output: DRIVER_IP_DEVICE_INFO structure
Notes: This call does not access the hardware, only stored driver parameters. NewIpCntl indicates that the module's carrier has expanded slot control capabilities. See the definition of DRIVER_IP_DEVICE_INFO below.

```
 //  Driver version and instance/slot information
typedef struct _DRIVER_IP_DEVICE_INFO {
   UCHAR    DriverRev;         // Driver revision
   UCHAR    FirmwareRev;       // Firmware major revision
   UCHAR    FirmwareRevMin;    // Firmware minor revision
   UCHAR    InstanceNum;       // Zero-based device number
   UCHAR    CarrierSwitch;     // 0..0xFF
   UCHAR    CarrierSlotNum;    // 0..7 -> IP slots A, B, C, D, E, F, G or H
   UCHAR    CarDriverRev;      // Carrier driver revision
   UCHAR    CarFirmwareRev;    // Carrier firmware major revision
   UCHAR    CarFirmwareRevMin;// Carrier firmware minor revision
   UCHAR    CarCPLDRev;        //**Used for PCIe carriers only**0xFF for others
   UCHAR    CarCPLDRevMin;     //**Used for PCIe carriers only**0xFF for others
   BOOLEAN  Ip32MCapable;      // IP capable of both 8MHz and 32MHz operation
   BOOLEAN  NewIpCntl;         // New IP slot control interface
   WCHAR    LocationString[IP_LOC_STRING_SIZE];
} DRIVER_IP_DEVICE_INFO, *PDRIVER_IP_DEVICE_INFO;
```

IOCTL_IP_REF_MEM_SET_IP_CONTROL

Function: Sets various control parameters for the IP slot the module is installed in.
Input: IP_SLOT_CONTROL structure
Output: None
Notes: Controls the IP clock speed, interrupt enables and data manipulation options for the IP slot that the board occupies. See the definition of IP_SLOT_CONTROL below. For more information refer to the IP carrier hardware manual.

```
typedef struct _IP_SLOT_CONTROL {
   BOOLEAN  Clock32Sel;
   BOOLEAN  ClockDis;
   BOOLEAN  ByteSwap;
   BOOLEAN  WordSwap;
   BOOLEAN  WrIncDis;
   BOOLEAN  RdIncDis;
   UCHAR    WrWordSel;
   UCHAR    RdWordSel;
   BOOLEAN  BsErrTmOutSel;
   BOOLEAN  ActCountEn;
} IP_SLOT_CONTROL, *PIP_SLOT_CONTROL;
```

IOCTL_IP_REF_MEM_GET_IP_STATE

Function: Returns control/status information for the IP slot the module is installed in.
Input: None
Output: IP_SLOT_STATE structure
Notes: Returns the slot control parameters set in the previous call as well as status information for the IP slot that the board occupies. See the definition of IP_SLOT_STATE below.

```
typedef struct _IP_SLOT_STATE {
   BOOLEAN  Clock32Sel;
   BOOLEAN  ClockDis;
   BOOLEAN  ByteSwap;
   BOOLEAN  WordSwap;
   BOOLEAN  WrIncDis;
   BOOLEAN  RdIncDis;
   UCHAR    WrWordSel;
   UCHAR    RdWordSel;
   BOOLEAN  BsErrTmOutSel;
   BOOLEAN  ActCountEn;
 // Slot Status
   BOOLEAN  IpInt0En;
   BOOLEAN  IpInt1En;
   BOOLEAN  IpBusErrIntEn;
   BOOLEAN  IpInt0Actv;
   BOOLEAN  IpInt1Actv;
   BOOLEAN  IpBusError;
   BOOLEAN  IpForceInt;
   BOOLEAN  WrBusError;
   BOOLEAN  RdBusError;
} IP_SLOT_STATE, *PIP_SLOT_STATE;
```

IOCTL_IP_REF_MEM_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.
Input: IP_REF_MEM_PLL_DATA structure
Output: None
Notes: After the PLL has been configured, the register array data is analyzed to determine the programmed frequencies, and the IO clock A-D initial divisor fields in the base control register are automatically updated.

IOCTL_IP_REF_MEM_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers
Input: None
Output: IP_REF_MEM_BASE_PLL_DATA structure
Notes: The register data is output in the IP_REF_MEM_PLL_DATA structure In an array of 40 bytes

IOCTL_IP_REF_MEM_GET_REVISION

Function: Returns IP module revision
Input: None
Output: IP_REM_MEM_REVISION
Notes: See the definition of IP_REM_MEM_REVISION below.

```
typedef struct _IP_REF_MEM_REVISION {
   UCHAR    minorFlashRev;
   UCHAR    majorFlashRev;
} IP_REF_MEM_REVISION, *PIP_REF_MEM_REVISION;
```

IOCTL_IP_REF_MEM_GET_IP_ID

Function: Returns IP module identification information
Input: None
Output: IP_IDENTITY
Notes: See the definition of IP_IDENTITY below.

```
typedef struct _IP_IDENTITY {
   UCHAR    IpManuf;
   UCHAR    IpModel;
   UCHAR    IpRevision;
   UCHAR    IpCustomer;
   USHORT   IpVersion;
} IP_IDENTITY, *PIP_IDENTITY;
```

IOCTL_IP_REF_MEM_SET_CONTROL

Function: Sets the master interrupt enable.
Input: IP_REF_MEM_BASE_CONFIG
Output: None
Notes: Sets bits in the IP_REF_MEM_BASE_CONFIG structure. See the definition of IP_REF_MEM_BASE_CONFIG below. Detailed definitions can be found in the 'IPREFMEM_BASE' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_REF_MEM_BASE_CONFIG
{
   BOOLEAN           RamClkSel;
   BOOLEAN           ClkOutSel;
   BOOLEAN           NodeMatchIntEn;
   BOOLEAN           NetRamDisable;
   BOOLEAN           NetIpDisable;
   BOOLEAN           NetPtDisable;
   BOOLEAN           NetIdDisable;
} IP_REF_MEM_BASE_CONFIG, *PIP_REF_MEM_BASE_CONFIG;
```

IOCTL_IP_REF_MEM_GET_CONTROL

Function: Clears the master interrupt enable.
Input: None
Output: IP_REF_MEM_BASE_CONFIG
Notes: Gets bits in the IP_REF_MEM_BASE_CONFIG structure. See the definition of IP_REF_MEM_BASE_CONFIG above. Detailed definitions can be found in the 'IPREFMEM_BASE' section under Register Definitions in the Hardware manual.

IOCTL_IP_REF_MEM_GET_NODE_SWITCH

Function: Reads the DIPSWITCH located on the IP. Both Net Address and Option Control.
Input: None
Output: IP_REF_MEM_NODE_SW
Notes: Detailed definition can be found in the 'IPREFMEM_NODESWITCH' section under Register Definitions in the Hardware manual.

IOCTL_IP_REF_MEM_GET_MESSAGE_COUNT

Function: Read roll over count of message received.
Input: None
Output: USHORT
Notes: 16 bit counter advances when valid messages are received. Rolls over at end count. Can be used for message traffic indication and network operation. Detailed definition can be found in the 'IPREFMEM_MESSAGECOUNT' section under Register Definitions in the Hardware manual.

IOCTL_IP_REF_MEM_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when an interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. In order to un-register the event, set the event handle to NULL while making this call.

IOCTL_IP_REF_MEM_ENABLE_INTERRUPT

Function: Sets the master interrupt enable.
Input: None
Output: None
Notes: Sets the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver ISR. This allows the driver to set the master interrupt enable without knowing the state of the other base configuration bits.

IOCTL_IP_REF_MEM_DISABLE_INTERRUPT

Function: Clears the master interrupt enable.
Input: None
Output: None
Notes: Clears the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_REF_MEM_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the IP bus. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_REF_MEM_SET_VECTOR

Function: Writes an 8 bit value to the interrupt vector register.
Input: UCHAR
Output: None
Notes: Required when used in non auto-vectored systems.

IOCTL_IP_REF_MEM_GET_VECTOR

Function: Returns a stored vector value.
Input: None
Output: UCHAR
Notes:

IOCTL_IP_REF_MEM_SET_MEM_DATA

Function: Write to Memory array.
Input: IP_MEMORY_WRITE
Output: None
Notes: Write the data and address. See the definition of IP_MEMORY_WRITE below.

```
typedef struct _IP_MEMORY_WRITE
{
   ULONG   MemoryOffset;
   USHORT  MemoryData;
} IP_MEMORY_WRITE, *PIP_MEMORY_WRITE;
```

IOCTL_IP_REF_MEM_GET_MEM_DATA

Function: Reads from Memory array.
Input: None
Output: USHORT
Notes: Detailed definition can be found in the 'IPREFMEM_ADDRESSMATCH' section under Register Definitions in the Hardware manual.

IOCTL_IP_REF_MEM_SET_NET_ADD_MATCH

Function: Set the address to match for incoming messages.
Input: USHORT
Output: None
Notes: Detailed definition can be found in the 'IPREFMEM_ADDRESSMATCH' section under Register Definitions in the Hardware manual.

IOCTL_IP_REF_MEM_GET_NET_ADD_MATCH

Function: Read from Address Match register.
Input: None
Output: USHORT
Notes:

IOCTL_IP_REF_MEM_CLEAR_STATUS

Function: Writes to status register to clear sticky bits.
Input: USHORT
Output: None
Notes: Write a '1' to the bit to be cleared

IOCTL_IP_REF_MEM_GET_STATUS

Function: Reads from status register.
Input: None
Output: USHORT
Notes:

IOCTL_IP_REF_MEM_SET_LED_CONFIG

Function: Writes to LED Control Register
Input: IP_REF_MEM_LED_CONFIG
Output: None
Notes: Sets bits in the IP_REF_MEM_LED_CONFIG structure. See the definition of IP_REF_MEM_LED_CONFIG above. Detailed definitions can be found in the 'IPREFMEM_LED_CNTL' section under

```
typedef struct _IP_REF_MEM_LED_CONFIG {
   BOOLEAN  MasterStatus;
   BOOLEAN  MasterEnable;
   BOOLEAN  ErrorDetected;
   BOOLEAN  LocalStatus;
   BOOLEAN  IpMessageSent;
   BOOLEAN  NetMessageSent;
   BOOLEAN  SpareLed0;
   BOOLEAN  SpareLed1;
} IP_REF_MEM_LED_CONFIG, *PIP_REF_MEM_LED_CONFIG;
```

IOCTL_IP_REF_MEM_GET_LED_CONFIG

Function: Reads from LED Control Register.
Input: None
Output: IP_REF_MEM_LED_CONFIG
Notes: Gets bits in the IP_REF_MEM_LED_CONFIG structure. See the definition of IP_REF_MEM_LED_CONFIG above. Detailed definitions can be found in the 'IPREFMEM_LED_CNTL' section under.

IOCTL_IP_REF_MEM_CLR_BAD_MESS_CNT

Function: Writes to LED Control Register
Input: USHORT
Output: None
Notes: Bad Message Counter is incremented whenever a bad message is detected – the error LED will flash and the counter will advance. 16 bit counter reset to “0000”. Counts 0->FFFF->0.

IOCTL_IP_REF_MEM_GET_BAD_MESS_CNT

Function: Reads from LED Control Register.
Input: None
Output: USHORT
Notes: Bad Message Counter is incremented whenever a bad message is detected – the error LED will flash and the counter will advance. 16 bit counter reset to “0000”. Counts 0->FFFF->0. Reading from this port will return the current count of bad messages.

Warranty and Repair

http://www.dyneng.com/warranty.html

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be “cockpit error” rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer’s making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer’s invoicing policy.

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 C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax
support@dyneng.com