ADPKGOCP

Software Library

User Guide

Version 1.0

Alpha Data
4 West Silvermills Lane
Edinburgh EH3 5BD
UK

Phone: +44 (0) 131 558 2600
Fax: +44 (0) 131 558 2700
Email: support@alpha-data.com

2570 North First Street, Suite 440
San Jose, CA 95131

Phone: (408) 467 5076
Fax: (408) 436 5524
Email: support@alpha-data.com

Table of Contents

1. Introduction

This document describes a number of software API library files provided to make the design of control software for FPGA applications built using the AD-PKG-OCP VHDL packages easier. These libraries capture the register definitions specified in the LB2OCP bridge module and other package modules, and provide functions to control and modify these more easily.

The libraries are implemented in C, although with a structure not too dissimilar from C++ objects. Each library file is essentially a class with a complex object like data structure, creator and destructor functions, and a group of object specific functions (methods) which use the object data structure to interact with the FPGA application.

The libraries work in conjunction with the ADM-XRC-SDK API (ADMXRC2). While they sit on top of this API for some functionality, it is still necessary to use the ADMXRC2 function calls to configure the FPGA, and set clocks, and perform other appropriate set up operations at the FPGA board level.

The following libraries are currently defined:

- lb2ocp.c – Local Bus/OCP Bridge Interface Control
- ocpsink.c – Control functions for Data Sink Components

2. LB2OCP

This library is contained in the files lb2ocp.h and lb2ocp.c. This provides functions for controlling the bridge, accessing its registers and setting up high speed DMA transfers between the host and any slaves (e.g. SDRAM memory) on burst OCP profile interfaces.

2.1. Object Structure

The information stored about the bridge is stored in the LB2OCP structure, this will typically be referenced using the PLB2OCP pointer.

```
typedef struct _LB2OCP {
  ADMXRC2_HANDLE card;
  volatile DWORD *fpgaSpace;

  unsigned int num_dma_engines;
  unsigned int burst_size[4];
  int row_skip[4];
  int col_skip[4];
  unsigned int n_rows[4];

  DWORD dmamode;
  DWORD ier;

} LB2OCP;

typedef LB2OCP* PLB2OCP;
```

The structure contains a handle to the FPGA board it is running on, as well as a pointer to the base address of the FPGAs memory map in user space. The structure also contains information on how the DMA channels and interrupts are configured.

2.2. Object Initialisation

```
PLB2OCP LB2OCP_Create(ADMXRC2_HANDLE card, int use64bit_dma, int
use64bit_registers);
void LB2OCP_Destroy(PLB2OCP bridge);
void LB2OCP_SetNumberDMAEngines(PLB2OCP bridge, unsigned int
num_dma_engines);
```

The LB2OCP_Create function should be used to create an object for controlling the LB2PLB bridge functionality in the FPGA, and connect it to the card. Whether the FPGA design has been built to run with a 32 or 64 bit DMA and register access should also be specified here.

Since this function allocates memory for the LB2OCP structure, LB2OCP_Destroy should be used to delete it.

The number of DMA engines parameter defaults to 0. This must be changed to enable the DMA engine access functions. Note that setting a value of 1 will enable 1 target DMA engine (corresponding to 1 channel in each direction – i.e. 2 DMA channels in the PCI to Local Bus bridge). A value of 2 should only be used with boards such as the ADM-XRC-5T1 which have 4 DMA channels in the PCI to Local Bus bridge, and if they have 2 DMA engines instantiated in the Local Bus to OCP bridge.

2.3. DMA Control Methods

```
int LB2OCP_TranferDataToFPGA(PLB2OCP bridge,
                        unsigned int engine,
```

```
                        unsigned int target,
                        unsigned int size,
                        unsigned int ocp_start_addr,
                        unsigned int host_offset,
                        ADMXRC2_DMADESC bufhandle);

int LB2OCP_TranferDataToHost(PLB2OCP bridge,
                        unsigned int engine,
                        unsigned int target,
                        unsigned int size,
                        unsigned int ocp_start_addr,
                        unsigned int host_offset,
                        ADMXRC2_DMADESC bufhandle);
```

The 2 main methods for performing DMA transfers are TransferDataToFPGA and TransferDataToHost. These allow specification of the DMA engine (0 or 1), the DMA target (the FPGA OCP module being accessed – design dependent), the size in bytes of the transfer, the start address on the OCP target bus, the host offset (start address) in the host DMA buffer, and the host DMA buffer itself. The ADMXCR2 API functions should be used to set up the DMA buffer.

The DMA transfer can be further configured to adjust the OCP burst size and the Row/Column 2D scatter gather options within the bridge DMA engines, using the following 4 functions.

```
int LB2OCP_SetBurstSize(PLB2OCP bridge, unsigned int channel,
unsigned int size);
int LB2OCP_SetRowSkip(PLB2OCP bridge, unsigned int channel, unsigned
int size);
int LB2OCP_SetColumnSkip(PLB2OCP bridge, unsigned int channel,
unsigned int size);
int LB2OCP_SetNRows(PLB2OCP bridge, unsigned int channel, unsigned
int size);
```

2.4. Register Space Access

Function calls are also provided for accessing the 7 user space register areas. This is a simple way of accessing the control registers of any module attached to a non-bursting register profile. A more object oriented approach to this is demonstrated by the ocpsink functions documented in section 3, but the basic bridge library provides these functions to allow access without having to develop a module specific control class.

```
DWORD LB2OCP_ReadRegister(PLB2OCP bridge, unsigned int space,
unsigned int offset);
DWORD LB2OCP_ReadRegisterPaged(PLB2OCP bridge, unsigned int space,
unsigned int offset);
void LB2OCP_WriteRegister(PLB2OCP bridge, unsigned int space,
unsigned int offset, DWORD data);
void LB2OCP_WriteRegisterPaged(PLB2OCP bridge, unsigned int space,
unsigned int offset, DWORD data);
```

The space should be set as a value from 1 to 7 to match the 512k window within which the target register is being addressed. The offset is the register offset (in 32 bit DWORDs) of the register within the window. The paged versions of the function allow the full 32 bit OCP addresses to be used, supporting OCP modules with addressable regions larger than 512kB, by writing the MSBs of the offset to the space page register.

2.5. Interrupts

```
void LB2OCP_EnableInterrupts(PLB2OCP bridge, unsigned int mask);
void LB2OCP_DisableInterrupts(PLB2OCP bridge, unsigned int mask);
void LB2OCP_ClearInterrupts(PLB2OCP bridge, unsigned int mask);
int LB2OCP_TestInterrupts(PLB2OCP bridge, unsigned int mask);
```

These functions control the Interrupt registers in the LB2OCP bridge. Waiting for the interrupt should be achieved by attaching and event to the ADMXRC2 card using the ADMXRC2 API functions. Test Interrupts returns 1 if any bits in the mask are set.

2.6. Other Functions

```
DWORD LB2OCP_ReadInfoRom(PLB2OCP bridge, unsigned int offset);
int LB2OCP_Status(PLB2OCP bridge);
```

The read info ROM function returns a word from the user configurable BRAM in the FPGA. The contents of this ROM should be set using Data2BRAM and bitgen, during the bitstream generation.

The Status function returns the status word supplied to the bridge. This usually contains top level status information such as clock locking status and memory controller status and can be used to check basic operation and diagnose some FPGA operational problems. The contents of this register is firmware dependent.

3. OCPSINK

This class is an example of a module access object for a specific OCP module attached to multiple host accessible profile interfaces within the FPGA. This interface is designed to work with the dataflow/data_sink modules.

3.1. Object Structure

```
typedef struct _OCPSINK {

  volatile DWORD *fpgaSpace;
  unsigned int burst_engine;
  unsigned int burst_target;
  unsigned int reg_bank;
  unsigned int irq_mask;
  unsigned int use64bit;

} OCPSINK;

typedef OCPSINK* POCPSINK;
```

The object contains a pointer to the start of its address space. It also contains information (burst_engine,burst_target) useful for the LB2OCP class to transfer data from the capture memory, by specifying the capture buffer location in terms of LB2OCP burst bus location. The register bank and irq mask also specify parameters used by the related LB2OCP object. A use64bit flag is used to modify the address map depending on whether a data_sink_b??r32 or data_sink_b??r64 has been instantiated.

3.2. Object Initialisation

```
POCPSINK OCPSINK_Create(PLB2OCP bridge,
                        unsigned int burst_engine,
                        unsigned int burst_target,
                        unsigned int reg_bank,
                        unsigned int reg_offset,
                        unsigned int irq_mask,
                        unsigned int use64bit);

void OCPSINK_Destroy(POCPSINK sink);
```

Creator and Destructor functions are provided, to allocate memory for the object, and connect it to an associated LB2OCP object. The register bank and offset are used to specify the start address of the OCP module registers. The offset allows an OCP DEMUX to be used in the FPGA if more than one object needs to reside in a particular 512kB space.

3.3. Object Control

```
int OCPSINK_StartCapture(POCPSINK sink,
                         unsigned int start_addr,
                         unsigned int word_limit,
                         unsigned int irq_limit,
                         unsigned int continuous);

void OCPSINK_HaltContinuous(POCPSINK sink);

void OCPSINK_Clear(POCPSINK sink);
int OCPSINK_IsRunning(POCPSINK sink);
```

Functions are provided to start the data capture operation, halt the continuous mode (stop at end of word limit), clear (stop and clear now). A function is also provided to test if the capture is still running. To wait on interrupts, the LB2OCP and ADMXRC2 library functions should be used to set the appropriate IRQ mask bits and attach a Windows event to wait on.

3.4. Monitor Functions

Some monitoring/debug functions can also be provided to check the status of the data capture, and possibly check why it has locked up – e.g. insufficient data received.

```
unsigned int OCPSINK_GetCurrentAddress(POCPSINK sink);
unsigned int OCPSINK_GetCurrentDataCount(POCPSINK sink);
unsigned int OCPSINK_GetCurrentIntCount(POCPSINK sink);
```