ADPKGOCP

Reference OCP-IP Development Blockset

Reference 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

ADPKGOCP Reference Guide

Version 1.0

Table of Contents

1. Introduction

The ADPKGOCP reference IP library is a collection of VHDL modules and packages aimed at easing system level design of FPGAs on Alpha Data ADM-XRC Series boards. The core functionality of these blocks is based upon a small number of OCP-IP based profiles, which are provided as abstract signal types to simplify interfacing of multiple blocks. The libraries and blocks are provided as standard VHDL, and system level connection can be specified easily in that language, requiring no proprietary vendor specific tools, and complete compatibility with industry standard Synthesis and Simulation tools.

1.1. OCP-IP Profiles

OCP-IP (Open Core Protocol - Intellectual Property) provides a common standard for system on-chip core interfaces. It is an open flexible standard which allowing it to encompass a variety of different profiles sharing a similar data transfer methodology. Even if the signals of 2 OCP-IP modules to not strictly match it should be relatively easy to connect them with minor modification (some tools are proposed to automate this, but they are not considered within this library).

The ADPKGOCP library contains 2 different profiles for register type access to modules and for burst access to modules. There are also sub-profiles of different data widths. The profiles are defined within the ocpad package as VHDL records, and there is a type for the Master-to-Slave transfer and for the Slave-to-Master responses.

Register Types (32 or 64 bit wide data)

Master to Slave(ocpad_register32_m2sT, ocpad_register64_m2sT)

```
Cmd : OCP Command Type (3 bits, “000” = IDLE, “001” = WRITE, “010” =
READ)
Addr : 32 bit wide address bus
Data : 32 or 64 bit wide write data
DataByteEn : 4 or 8 bit wide date byte enables
Reset_n : Active Low Synchronous Reset
```

Slave to Master (ocpad_register32_s2mT, ocpad_register64_s2mT)

```
CmdAccept : Command (and Write Data) accept
Data : 32 or 64 bit wide read data
Resp : OCP Response Type (2 bits, “00” – None, “01” – Valid, “10” –
Failed, “11” – Error)
```

With the register type access, each command only reads/writes one word (burst length assumed to be 1). The Write Data should be valid with the OCP Cmd. The Read Data is valid with the Resp. The Master should hold the Cmd, Addr, Data lines until the CmdAccept is received from the Slave.

Burst Types 32,64 or 128 bits

Master to Slave(ocpad_burst32_m2sT, ocpad_burst64_m2sT, ocpad_burst128_m2sT)

```
Cmd : OCP Command Type (3 bits, “000” = IDLE, “001” = WRITE, “010” =
READ)
Addr : 32 bit wide address bus
BurstLength : 8 bit wide burst length
BurstSingleReq : 1=BurstLength is exact, known at start; 0=Use
DataLast
Data : 32, 64 or 128  bit wide write data
DataByteEn : 4,8 or 16 bit wide date byte enables
DataLast : Indicates last beat of burst
DataValid : Indicates Data is Valid
RespAccept : Indicate that the Master has Accepted response data
Reset_n : Active Low Synchronous Reset
```

Slave to Master (ocpad_burst32_s2mT, ocpad_burst64_s2mT, ocpad_burst128_s2mT)

```
CmdAccept : Command accept
DataAccept : Data accept
Data : 32, 64 or 128 bit wide read data
Resp : OCP Response Type (2 bits, “00” – None, “01” – Valid, “10” –
Failed, “11” – Error)
RespLast : Indicate last data word of burst
Error : error
```

1.2. VHDL IP Modules and Packages

A number of groups of modules are provided in this package to aid system level development. The main groupings are:

- Basic Connection Interfaces and Tools (and the adpkgocp directory)
- Local Bus Bridge IP
- Memory Interface Bridge IP
- Profile2Profile Bridge IP
- Data Flow IP
- Examples

This package is designed to sit on top of the Alpha Data SDK IP modules provided with the Alpha Data card. This includes many of the basic local bus interfacing state machine modules, some FIFOs and the memory interfacing packages provided in the SDK. The SDK version used with the examples in this blockset is 4.9.0. Minor changes to the examples may be required when moving to a newer version of the SDK.

2. Basic Connection and Interfacing IP

This chapter describes the basic connection IP blocks provided in the ADPKGOCP directory. These blocks allow multiple masters to drive a single slave, or multiple slaves to sit on a single master bus. Also included are some simple target slave blocks for easy connection to registers or simple slave buses. There are also some monitor and Chipscope blocks for monitoring the OCP buses. These blocks are simple examples with very basic arbitration schemes. There are also not registered so may not provide a solution capable of running at the highest clock frequency.

2.1. Multiport Blocks

```
entity MULTIPORT_OCP_IF2_b64 is
  generic (
    round_robin_arb : boolean := false;
    use_timeout  : boolean := false;
    timeout_bits : integer := 16);
  port(
    -- OCP Interface (Block Data Flow Profile)
    ocp_m2s_memif : out ocpad_burst64_m2sT;
    ocp_s2m_memif : in  ocpad_burst64_s2mT;
    ocp_m2s0 : in  ocpad_burst64_m2sT;
    ocp_s2m0 : out ocpad_burst64_s2mT;
    ocp_m2s1 : in  ocpad_burst64_m2sT;
    ocp_s2m1 : out ocpad_burst64_s2mT;
    status   : out std_logic_vector(3 downto 0);
    Clk : in std_logic);
end MULTIPORT_OCP_IF2_b64;
```

The purpose of these blocks is to allow multiple masters to access a single burst accessible slave resource. The most common application for this will be to provide simple multi-port access to a bank of memory. One port will most likely connected up to the local bus bridge to provide host access. The second port will be connected to the users application e.g. a data capture module receiving data from an ADC and writing it to memory. Blocks are provided for all three bust types and are provided with 2, 3 or 4 master ports. Generics can be set to select round-robin arbitration: this passes control to the next requesting master when a transaction completes, preventing a single master from hogging the port. Generics can also be used to specify a timeout on a slave request, freeing the port in the case of a failure to respond.

2.2. Demux Blocks

```
entity ocp_demux_b128 is
  generic (
    number_of_slaves  : integer := 2;
    demux_address_lsb : integer := 16);
  port(
    clk : in std_logic;
    master_m2s : in ocpad_burst128_m2sT;
    master_s2m : out  ocpad_burst128_s2mT;
    slaves_m2s : out ocpad_burst128_m2sT_array(0 to 
number_of_slaves-1);
    slaves_s2m : in  ocpad_burst128_s2mT_array(0 to 
number_of_slaves-1));
end ocp_demux_b128;
```

The de-multiplexing blocks are provided so that a single master can drive multiple slaves. The “demux_address_lsb” is used to determine the address space available for each slave and for selecting which slave the OCP-IP transaction is passed onto. The master address is divided so that Addr(31:demux_address_lsb) is used to select a slave in the range (0,number_of_slaves-1). Only Addr(demux_address_lsb-1:0) is passed to the slave, with the upper bits set to zero.

2.3. Dummy Targets

```
entity ocp_dummy_target64 is
  port (
    -- OCP Interface
    ocp_clk : in std_logic;
    ocp_m2s : in ocpad_burst64_m2sT;
    ocp_s2m : out ocpad_burst64_s2mT);
end ocp_dummy_target64;
```

For rapid prototyping, simulation and development of modules where accurate simulation of the target slave device is not required, some dummy targets are provided. These can be replaced later in the design cycle by the real target (e.g. a memory) once the mater module operation has been verified.

2.4. Register Interface and Slave Interface Targets

These modules provide a simple way of generating an array of registers or providing a simple bus interface for connecting to a block RAM or similar logic.

```
entity ocp_register_interface64 is
  generic (
    number_of_registers : natural := 1);
  port (
    -- OCP Interface
    ocp_clk : in std_logic;
    ocp_m2s : in ocpad_register64_m2sT;
    ocp_s2m : out ocpad_register64_s2mT;
    -- simple bus interface
    registers_out : out register64_array(0 to number_of_registers-1);
    registers_back : in register64_array(0 to number_of_registers-1);
    register_written : out std_logic_vector(number_of_registers-1
downto 0)
    );
end ocp_register_interface64;
```

The register interface module attaches a number of registers to the OCP bus. These start at address 0 and successive registers occupy successive addresses. To make the registers readable, the registers_out value must be fed backto the registers_back array. The registers back array should also be used to generate status and other read only registers. The register_written signal pulses high for one clock cycle shortly after the registers out value has changed following a write to the register.

```
entity ocp_slave_interface64 is
  generic (
    read_latency : integer := 1);
  port (
    -- OCP Interface
    ocp_clk : in std_logic;
    ocp_rst : in std_logic;
    ocp_m2s : in ocpad_register64_m2sT;
    ocp_s2m : out ocpad_register64_s2mT;
    -- simple bus interface
    simple_bus_addr : out std_logic_vector(21 downto 0);
    simple_bus_write : out std_logic;
    simple_bus_read : out std_logic;
    simple_bus_wdata : out std_logic_vector(31 downto 0);
    simple_bus_rdata : in std_Logic_vector(31 downto 0));
end ocp_slave_interface64;
```

The slave interface maps the OCP profile to a simple 32 bit bus interface. This is suitable for connecting up to simple memory blocks such as block RAM. For write operation, a single pulse on write will occur at the same time as the addr and wdata signals are valid. For read operations, the read signal will validate the address. The module returns to the OCP master, the contents of rdata after the specified “read_latency” clock cycles. This is fixed, and any devices on the bus must set rdata on the correct clock cycle.

2.5. Chipscope and other bus monitors

One of these modules can be connected to an OPC-IP profile interface to monitor the transactions for on-chip debugging purposes.

If connecting a chipscope cable is difficult in the system, it is also possible to connect an ocp_monitor block to capture the OCP burst transactions. This block captures OCP profile activity starting with a read or write command for 512 clock cycles into swinging buffers, so only the most recent activity is recorded. The data buffers can be connected to a register profile and read back to the host for debugging.

3. Local Bus Bridge

The Local Bus on all Alpha Data PMC cards is the route to the PCI and the host. All communication between the FPGA application and the host goes through this bus. The OCP-IP bridge contains significant functionality beyond that of the simple local bus interfaces provided in the SDK.

A selection of bridge modules is provided to support the different burst profile widths. The most appropriate burst profile width depends largely on the memory architecture of the FPGA board, however all widths will work with all cards. There is also a choice of the register profile interface widths. A bridge with no burst interface is also provided for low data rate designs.

The package localbus_if_types is provided to abstract away the differences between PLX and FPGA based PCI bridges in the design. This works in conjunction with the files, localbus_io,localbus_io_plx (or localbus_io_iobufs, localbus_io_plx_iobufs if you disable the IOB insertion in XST).

The local bus bridge module provides an array of up to 7 register profile master ports. It can also include 1 or 2 DMA engines, each of which can support up to 8 burst profile master interfaces. Each DMA engine uses 2 DMA channels in the card bridge chip, so 2 DMA engines in this bridge is only possible with cards which return 4 DMA channels available from INFO (XRC4FX, XRC5T1, XRC5T2) .

The number of OCP interfaces and DMA engines is configured by the module generics.

The module also has a system status input for outputting system information (such as DCM lock status).

A interrupt port (32 bits wide) is provided which will trigger on any rising edge of the 32 signals. Registers are provided to mask these interrupts, clear the latch and show the status. The irq output signal should be connected through an inverter to the fint_l pin to interrupt the host.

3.1. Example Bridge Entity

```
entity localbus2ocp_bridge_r64b128 is
  generic (
    number_of_ocp_nonburst : integer := 7;
    number_of_ddma_engines : integer := 2;
    number_of_ocp_burst_per_engine : integer := 4;
    la_top           : natural := 23;
    family           : family_t := family_virtex4);
  port (
    lclk             : in    std_logic;  -- Local Bus Clock
    ocp_clk_nb       : in    std_logic;  -- System Clock for non-
burst
    ocp_clk_b        : in    std_logic;  -- System Clock for burst
    rst              : out   std_logic;  -- Global Reset (LRESET)
    -- OCP Interfaces
    ocp_nonburst_bus_outputs_array         : out
ocpad_register64_m2sT_array(0 to number_of_ocp_nonburst-1);
    ocp_nonburst_bus_inputs_array          : in
ocpad_register64_s2mT_array(0 to number_of_ocp_nonburst-1);
    ocp_burst_bus_outputs_array         : out
ocpad_burst128_m2sT_array(0 to
number_of_ocp_burst_per_engine*number_of_ddma_engines-1);
    ocp_burst_bus_inputs_array          : in
ocpad_burst128_s2mT_array(0 to
number_of_ocp_burst_per_engine*number_of_ddma_engines-1);
    -- Local Bus Interface
    localbus_inputs  : in  localbus_inputs_type;
    localbus_outputs : out localbus_outputs_type;
    system_status    : in  std_logic_vector(31 downto 0);
    system_irq       : in  std_logic_vector(31 downto 0);
    irq              : out std_logic);
end localbus2ocp_bridge_r64b128;
```

3.2. Register OCP-IP Profile Interfaces

Up to 7 OCP Slaves can be driven using the Non-burst interfaces:

ocp_nonburst_bus_outputs_array

ocp_nonburst_bus_inputs_array

These are synchronous to the ocp_clk_nb

Each of these has a 512kB window in the Local Bus 4MB slave address space. These windows start at address 0x080000. (Addresses 0x000000- 0x07FFFF are used for the bridge registers) A control register is provided in the bridge register space to allow the upper address bits to be set, allowing the slave to have an addressable space larger than 512kB.

(addresses 0x000014 to 0x00002C)

3.3. Burst OCP-IP Interfaces

The burst accessible interfaces are not mapped directly to the local bus 4MB slave address space. Slave reads across the local bus initiated by the board bridge FPGA/ASIC can be relatively inefficient when accessing SDRAM type memories, due to the large latency, and the need to pre-fetch and flush data. This “target” bridge therefore has DMA engines which read the data directly from the memory and buffer it in FIFOs. These FIFOs are then read using demand-mode DMA over the Local Bus, thus removing the need for pre-fetch. The software approach for reading or writing data using these target DMA engines, is therefore more complex, since registers in the target need to be set first, to start the SDRAM to FIFO (or FIFO to SDRAM) transfer. Then the Demand Mode DMA process needs to be set up by the driver to transfer data from the FIFOs to host memory.

3.4. Information ROM

The bridge contains a host accessible information ROM. This is implemented using a block RAM, and can be initialised after bitgen using data2bram tools. An associated .bmm file needs to be used throughout the ISE design flow. This allows status, version and other bitstream specific information to be encoded in the .bit file.

3.5. Bridge Memory Map

Address	Function
0x000000	Demand Mode DMA channel 0 Xfer Size (Engine #0 - Write)
0x000004	Demand Mode DMA channel 1 Xfer Size (Engine #0 - Read)
0x000008	Demand Mode DMA channel 2 Xfer Size (Engine #1 - Write)
0x00000C	Demand Mode DMA channel 3 Xfer Size (Engine #1 - Read)
0x000010	Not Used
0x000014	Register Space #1 Page
0x000018	Register Space #2 Page
0x00001C	Register Space #3 Page
0x000020	Register Space #4 Page
0x000024	Register Space #5 Page
0x000028	Register Space #6 Page
0x00002C	Register Space #7 Page
0x000030	DMA Engine #0 Write Target/Burst Length
0x000034	DMA Engine #0 Write Row Size
0x000038	DMA Engine #0 Write Number of Rows
0x00003C	DMA Engine #0 Write Column Increment
0x000040	DMA Engine #0 Write Start Address
0x000044	DMA Engine #0 Read Target/Burst Length/FGSG
0x000048	DMA Engine #0 Read Row Size
0x00004C	DMA Engine #0 Read Number of Rows
0x000050	DMA Engine #0 Read Column Increment

0x000054	DMA Engine #0 Read Start Address
0x000058	DMA Engine #1 Write Target/Burst Length
0x00005C	DMA Engine #1 Write Row Size
0x000060	DMA Engine #1 Write Number of Rows
0x000064	DMA Engine #1 Write Column Increment
0x000068	DMA Engine #1 Write Start Address
0x00006C	DMA Engine #1 Read Target/Burst Length/FGSG
0x000070	DMA Engine #1 Read Row Size
0x000074	DMA Engine #1 Read Number of Rows
0x000078	DMA Engine #1 Read Column Increment
0x00007C	DMA Engine #1 Read Start Address
0x000080	IRQ Enable Register
0x000084	IRQ Clear Register
0x000088	System Status Register
0x00008C	IRQ Status Register
0x000090	Demand Mode DMA channel 0 Xfer Left (Engine #0 - Write)
0x000094	Demand Mode DMA channel 1 Xfer Left (Engine #0 - Read)
0x000098	Demand Mode DMA channel 2 Xfer Left (Engine #1 - Write)
0x00009C	Demand Mode DMA channel 3 Xfer Left (Engine #1 - Read)
0x0000A0	DMA Engine #0 Write Status LSW
0x0000A4	DMA Engine #0 Write Status MSW
0x0000A8	DMA Engine #0 Read Status LSW
0x0000AC	DMA Engine #0 Read Status MSW
0x0000B0	DMA Engine #1 Write Status LSW
0x0000B4	DMA Engine #1 Write Status MSW
0x0000B8	DMA Engine #1 Read Status LSW
0x0000BC	DMA Engine #1 Read Status MSW
0x07F800-07FFFC	Information ROM
0x080000-0FFFFC	Register Space #1
0x100000-17FFFC	Register Space #2
0x180000-1FFFFC	Register Space #3
0x200000-27FFFC	Register Space #4
0x280000-2FFFFC	Register Space #5
0x300000-37FFFC	Register Space #6
0x380000-3FFFFC	Register Space #7

3.6. DMA Engine Operation

Each DMA engine is connected to a Demand Mode DMA FIFO. For data in the direction of FPGA to HOST, the Demand Mode DMA channel (1 or 3) should be set up to read a specified amount of data and store it somewhere in the host memory. The XFER Size register (0x000004) should be written with the number of bytes to transfer. Note that the least significant bits of this word will be ignored if the size does not match the Burst Profile width.

The DMA Engines read registers should then be written to. The burst length (bits 7:0) specifies the number of beats for each read burst. The target bits (10:8) select one of up to 8 target devices. The Row Size specifies the address increment between bursts. This allows it to skip addresses. The Number of Rows specifies the number of bursts to perform before modifying the address by the column skip value (allowing a second dimension of data skipping). The start address should be written last as a write to this register also starts the DMA engine. The DMA engine will read data from the slave device (memory bank) until the DMA FIFO connected to the demand mode channel is nearly full. In this case the DMA engine will pause until there is space in the FIFO for the next burst.

The upper 16 bits of the Target/Burst/FGSG register is used to specify Fine Grained Scatter Gather reads. This operation allows data words within a burst to be discarded before it is transferred over the DMA, allowing fine grained data skips. (N.B. the data is still read from memory). Bits 19:16 specify the data skip pattern length (1 to 12). Bits 31:20 indicate a pattern of data to read (‘1’) or ignore (‘0’). E.G. setting the pattern length to 0x2, and the pattern to 0x001 would result in only every second word being read from the OCP slave. Note that the OCP DMA engine registers should be set up to transfer the full amount of data, but the Software Demand mode DMA and the demand mode DMA channel registers (0x000004, 0x00000C) should be set to reflect the reduced amount of data being sent over PCI. The granularity is set by the OCP bus width.

For data in the direction HOST to FPGA, the demand mode DMA channel should be set up to write data in on one of the Write Demand Mode DMA channels (0 or 2) . The DMA engine registers specify at what addresses on the slave device (memory bank) any data in the FIFO is to be written to. The DMA engine will generate write bursts to the target slave device when enough data is in the FIFO. Note that FGSG operation is only supported for reading.

The top bit (31) of the Write Target/Burst registers (0x30,0x58) is used as a timeout disable for the DMA engine (for both read and write). The OCP interface will timeout, and send dummy data to complete the DMA if the OCP slave fails to respond in a timely manner (e.g. if the slave logic locks up due to some bug) the DMA transfer will always complete. If blocking operation is required (e.g. the OCP slave is a FIFO) then this bit should be set.

4. Memory Interfaces

4.1. Memory Interface Bridge

Rather than incorporating memory interfaces into this library, a bridge module is provided so that the memory modules provided in the SDK memory application can easily be attached. This provides a easy way to dual port the memory and provide a data capture buffer which a data sink module can sink data from an external source (such as an ADC input), and store it in the memory bank until it can be transferred over to the host using DMA.

The SDK memory interfaces have a simple single beat transaction based interface using the following signals:

SR	Synchronous Reset

CE	Command Enable
W	Write
TAG	Tag
A	Address (configurable)
D	Write Data
BE	Byte Enable
Q	Read Data
QTAG	Read Tag
VALID	Read Data Valid
READY	Controller Online

The OCP2MEMIF bridges act as slaves to ***B32,B64*** or ***B128*** masters, and generate the appropriate SDK MEMIF signals for the memory port module. The bridge propagates the data width, so that data width must match. This can be achieved by changing the profile width using an OCP Profile to Profile asynchronous bridge, or by joining multiple memory port modules in parallel to increase the data width.

4.2. Memory Simulation Modules

```
entity ocpsimmem128 is
  generic (
    initialise_dwidth : integer := 0;
    initialise_file   : boolean := false;
    input_file        : string  := "";
    addr_width        : integer := 16);
  port (
    clk        : in  std_logic;
    -- OCP Port
    ocp_m2s    : in  ocpad_burst128_m2sT;
    ocp_s2m    : out ocpad_burst128_s2mT;
    -- Save Data
    save       : in  boolean;
    save_start : in  integer;
    save_end   : in  integer;
    output_file : in string  := "");

end ocpsimmem128;
```

A number of ocpsimmem (OCP Simulation Memory) modules are provided for testbench development and core verification. These are not synthesizable. The memory contents default to all zero, but can be initialised to a incrementing counter values with data widths ranging from 8 bits up to the memory width (max 128 bits). A binary file can also be specified as the initial value of the memory data. The memory depth is specified using the addr_width parameter. The memory contents can be saved during the simulation by strobing the Boolean save signal. This will save the memory contents from address save_state to address save_end to the specified output file.

4.3. Block RAM Interfaces

```
entity ocp_bram_interface32 is
  generic (
    read_latency : integer := 1);
  port (
    -- OCP Interface
    ocp_clk : in std_logic;
    ocp_rst : in std_logic;
    ocp_m2s : in ocpad_register32_m2sT;
    ocp_s2m : out ocpad_register32_s2mT;
    -- simple bus interface
    bram_addr : out std_logic_vector(15 downto 0);
    bram_write : out std_logic;
    bram_read : out std_logic;
    bram_be   : out std_logic_vector(3 downto 0);
    bram_wdata : out std_logic_vector(31 downto 0);
    bram_rdata : in std_Logic_vector(31 downto 0)
    );

end ocp_bram_interface32;
```

Attaching a register profile to a block RAM is supported with the ocp_bram_interface modules. These have configurable latency to allow the use of the output register in the BRAM if timing is critical.

5. Profile to Profile Bridges

There may be many situations where an IP module developed for one board using a particular bus width needs to be re-used on a different board which works best with a different bus width. The profile to profile bridges are provided to give an easy way of connecting two profiles of different types together. This will be less efficient than modifying the target IP module to work with the correct profile, however it may provide sufficient performance in many cases without having to modify any code.

This collection of modules covers the cases where changing the burst width is required. These modules do not handle the cases where the burst size of the smaller bus width profile does not always provide complete word widths at the larger bus width. That is if the bus width doubles, the burst length of the smaller bus must be a multiple of 2. And if it quadruples, the length must be a multiple of 4. Also when reducing the widths the maximum burst length must be less than 256 of the smallest word size.

Bridges are also provided to allow a register master to drive a burst capable slave. The burst lengths are always 1 in this case.

6. Data Flow

One of the most common applications is data capture from a live source. A range of data sink modules are provided to receive data from some external source and store it in memory. The data width need not match the memory width, and a shift register is used to pack multiple samples into a single memory write word if required. The data flow interface is much simpler than the OCP-IP based profile interfaces and so can also be used to interface to a very wide range of IP. The data flow has data, data_valid and ready for data signals, and can easily be connected to FIFOs or data flow protocols such as the Xilinx Local Link. Data source modules are also provided to read data from memory banks and act as a source of data for either an internal computational data flow module or an external output such as a DAC.

6.1. Data Sources

Each data source has a Master Mode burst interface for connection to a memory bank. This interface will read out data depending on how the state machine in the data source module is programmed.

```
entity data_source_b128r32 is
  generic (
    data_width : integer := 16;
    data_shift : integer := 16;
    shift_count_width : integer := 4;
    family     : family_t := family_virtex4;
    fifo_order : integer := 9;
    use_rate_control : boolean := true;
    use_rfd : boolean := true);
  port (
    -- Interface to OCP Burst Profile
    burst_clk : in  std_logic;
    burst_m2s : out ocpad_burst128_m2sT;
    burst_s2m : in  ocpad_burst128_s2mT;
    -- Interface to OCP Register Profile
    reg_clk  : in  std_logic;
    reg_m2s  : in  ocpad_register32_m2sT;
    reg_s2m  : out ocpad_register32_s2mT;
    -- Interface to Data Flow
    data_clk : in  std_logic;
    data : out std_logic_vector(data_width-1 downto 0);
    data_valid : out std_logic;
    rfd : in std_logic;
    -- Interrupt out
    irq : out std_logic);
end data_source_b128r32;
```

These modules are programmed through a number of registers:

Address R32/R64	Function
0x00/0x00	Start Address of Data
0x04/0x08	Word Limit – Number of (OCP Burst Width) Words to Output
0x08/0x10	Interrupt Limit – Interrupt generated after this number of (Data Width) words have been output.
0x0C/0x18	CONTROL/STATUS Register: Bit 0: Write 1 to Start, Read indicates if Running Bit1: Enable Continuous Operation, Once Word Limit is reached the address pointer will reset to the start address
0x10/0x20	Rate : If set, Data will be output and Valid asserted every “x” clock cycles.

The *data width* is configurable, this number of bits are output from the output shift register for every Data Valid. The *data shift* may be set to a different value from data width, to handle 12 bit data stored as 16 bit values in memory. In this module, the shift values must be a divisor of the OCP Burst Width. The rate control hardware instantiation can be disabled as can the ready for data (RFD) operation to save hardware or improve maximum clock speed. The burst, reg and data clocks can all run asynchronously.

6.2. Data Sinks

The data sink modules can be similarly programmed. These accept a data stream and write it to memory via an OCP Burst Interface.

```
entity data_sink_b128r32 is
  generic (
    data_width : integer := 16;
    data_shift : integer := 16;
    shift_count_width : integer := 4;
    family     : family_t := family_virtex4;
    fifo_order : integer := 9);
  port (
    -- Interface to OCP Burst Profile
    burst_clk : in  std_logic;
    burst_m2s : out ocpad_burst128_m2sT;
    burst_s2m : in  ocpad_burst128_s2mT;
    -- Interface to OCP Register Profile
    reg_clk  : in  std_logic;
    reg_m2s  : in  ocpad_register32_m2sT;
    reg_s2m  : out ocpad_register32_s2mT;
    -- Interface to Data Flow
    data_clk : in  std_logic;
    data : in std_logic_vector(data_width-1 downto 0);
    data_valid : in std_logic;
    rfd : out std_logic;
    flush_packet : in std_logic;
    -- Interrupt out
    irq : out std_logic);
end data_sink_b128r32;
```

The following registers are defined for data sinks

Address R32/R64	Function
0x00/0x00	Start Address of Data
0x04/0x08	Word Limit – Number of (OCP Burst Width) Words to Record
0x08/0x10	Interrupt Limit – Interrupt generated after this number of (Data Width) words have been received.
0x0C/0x18	CONTROL/STATUS Register: Bit 0: Write 1 to Start, Read indicates if Running Bit1: Enable Continuous Operation, Once Word Limit is reached the address pointer will reset to the start address Bit 2: Clear FIFO, Word Count, and stop data capture.
0x10/0x20	Current Address (Read Only)
0x14/0x28	Current Memory Write Word Count (Read Only)
0x18/0x30	Current Interrupt Data Word Count (Read Only)

The data width and shift width can be specified in the same way as in the Data Source. RFD indicates that the data capture is running and the FIFO is able to accept data. Flush Packet can be used to complete the data capture, if less data is received than is specified in Word

Limit. This also flushes the input shift register, to support data packets, which are not a multiple of the burst width word size.

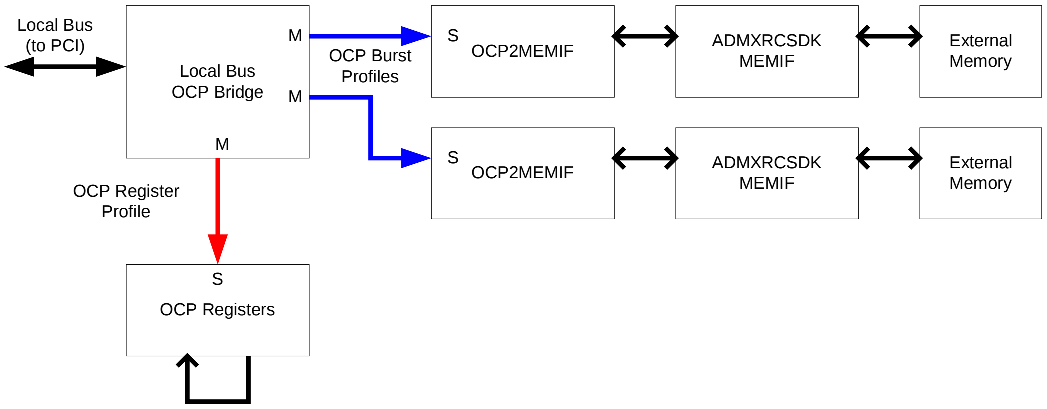
7. Examples

A couple of examples are provided to show how to connect up the blocks. These examples do not exhaustively cover all supported boards. It should however be possible to modify one of these examples to work with the desired board with minimal difficulty. The source code for host applications for testing these designs is also provided.

7.1. Memory Interface Example

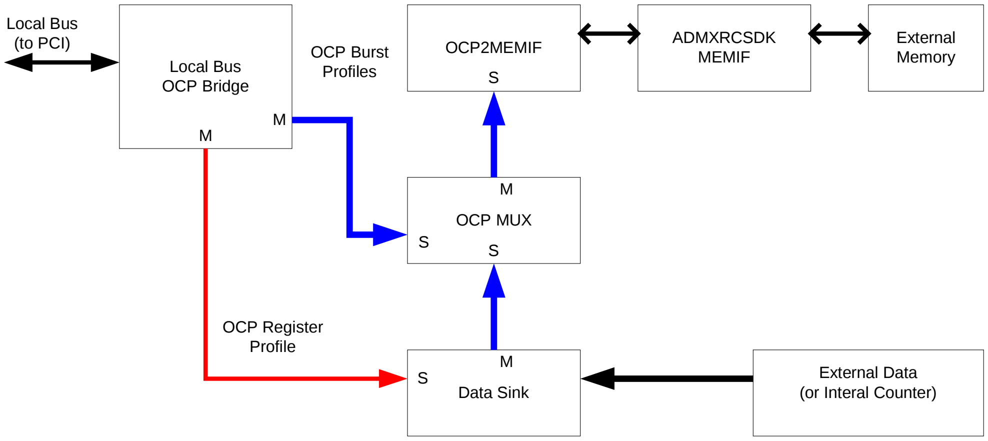
This example shows how to connect the memory interfaces provided in the SDK up to the OCP-IP local bus bridge. It also contains a host test program with functions demonstrating the data transfer from host to the memory and back using the demand mode DMA engines.

The diagram shows the top level structure, width the OCP Profile arrows indicating the Master/Slave control direction (not the data flow which is bi-directional.)



7.2. Data Capture Example

This example extends the basic memory example, by adding an OCP dual port MUX to the memory interface. The second port of this is connected up to a data sink module, which captures data from either an external source, or in internal counter.


Local Bus
(to PCI)
Local Bus
OCP Bridge
M
M
OCP Burst
Profiles
OCP2MEMIF
S
ADMXRCSDK
MEMIF
External
Memory
M
OCP MUX
S
S
OCP Register
Profile
M
S
Data Sink
External Data
(or Interal Counter)