Version 0.01a For EDK 6.2i 24/06/2004

1 Goals

This document is intended for those who have a working design written in Verilog. The main principles can be extended to other languages.

- Building an OPB interface.
- Using IBM's OPB toolkit to simulate the design via OPB transactions.
- Adding the design to an EDK project.

2 Constructing the OPB interface[1]

2.1 What is the OPB?

The On-chip Peripheral Bus (OPB) is as bus specification designed by IBM used by Xilinx in MicroBlaze systems to communicate with peripherals. Devices on the bus are can be either masters or slaves. This document assumes that the design will be attached as a slave device.

2.2 The OPB protocol

The protocol described in this document is older than that used by Xilinx in the EDK, however, it allows for the design to be tested with the OPB toolkit. Slight modifications can then be made to bring the interface up to the new standard. This is addressed in section 6. An updated version of the OPB toolkit may be available with the new standard.

A typical declaration of a slave device is:

```
module opbinterface(OPB_clk, OPB_select, OPB_RNW, OPB_hwXfer,
       OPB_fwXfer, OPB_seqAddr, OPB_ABus, OPB_DBus,
       Sln_xferAck, Sln_hwAck, Sln_fwAck, Sln_errAck,
       Sln_toutSup, Sln_retry, Sln_DBusEn, Sln_DBus );
```

The ports beginning with OPB are inputs from the bus while the ports beginning with Sln are outputs to the bus.

A typical transaction is shown in Figure 1.

OPB_clk The clock at which the bus is run. Using the Xilinx Multimedia Boards, this is at 27MHz.

OPB_select Driven high when OPB_ABus and other bus signals are valid.

OPB_RNW "Read-not-write" Low when the MicroBlaze is requesting a write. High when requesting a read.

OPB_hwXfer, OPB_fwXfer Used to indicate the valid bits on a write. OPB_hwXfer is high when a halfword or a fullword is being sent. OPB_fwXfer is high only when a fullword is being sent. These signals are replace by the byte enable signal in the OPB standard used by Xilinx.

OPB_seqAddr Indicates that the bus is requesting multiple reads or writes from sequential addresses. This is ignored in this document's treatment of the OPB protocol. If you wish to use its functionality see [1].

OPB_ABus The address for which a given read or write is requested.

[1] It is notable that a Xilinx core exists which can interface a design with the OPB. It is accessible through the *Edit → Import Peripheral Wizard* command in EDK. However currently only VHDL is supported. Since, currently, mixed language designs cannot be used as IPs in the EDK the simplest alternative is to design an interface from scratch.

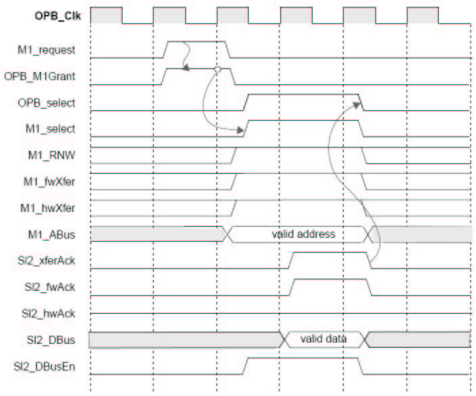

Figure 1: A typical OPB transaction [1]

OPB_DBus The data bus used for a write transaction.

Sln_xferAck The response from the slave device once it has finished its part of the transaction (taking data from or putting data on the bus).

Sln_hwAck Acknowledging a halfword transfer. (Obsolete in the OPB standard used by Xilinx)

Sln_fwAck Acknowledging a fullword transfer. (Obsolete in the OPB standard used by Xilinx).

Sln_errAck Set when the slave encounters an error processing a request. It is set in addition to Sln_xferAck.

Sln_toutSup Set by the slave to that its operation will occur for an extended period of time. "Timeout suppression"

Sln_retry Set by the slave to request a retry of the transaction.

Sln_DBusEn Set by the slave to indicate the data on Sln_DBus is valid. (Obsolete in the OPB standard used by Xilinx).

Sln_DBus The port used by the slave to put data on the bus during a read operation.

As implemented by Xilinx, the bus consists of a 32bit Address signal and a 32bit Data signal. Detailed descriptions of the bus signals and the transaction can be found in [1].

2.3 Coding the Interface

When writing the code for the interface include the parameters:

```
parameter C_BASEADDR = 32'hffffff00;
parameter C_HIGHADDR = 32'hffffffff;
```

to indicate the range of addresses that your device will respond to on the bus. These parameters will be overridden by the EDK when the device is inserted into the system. The value set in the HDL code will be useful during simulation.

3 Testing the OPB interface - Using IBM's CoreConnect Toolkit

3.1 What is the CoreConnect Toolkit?

The CoreConnect toolkit consists of several sub-toolkits for verifying a design that uses any of various on-chip bus standards created by IBM. Of interest in this document is the OPB toolkit.

The OPB toolkit is located in *CoreConnect/tk/OPB_TOOLKIT2X*.

The toolkit consists of several models to mimic the behavior of devices on the bus. The types of devices/models are Arbiter, Master and Slave. A device can be tested by connecting it to these models. The instantiation of the models and controlling their behavior is done using scripts written in the toolkit specific *OPB Bus Functional Language* that can be compiled with a compiler accompanying the toolkit.

3.2 Inserting the device into the toolkit

The various models used in the toolkit are found in *CoreConnect/tk/OPB_TOOLKIT2X/verilog*. Two sets of models exist in the directory: those with the "1x" suffix and those without. The "1x" suffix is the 32bit OPB model and is the focus in this document. opb_complex1x.v is the top-level module within which the device under test can be inserted.

Before the device under test can be inserted, new signals must be created in the toolkit's HDL to accommodate the new device. Several lines must be changed to do this. Below is the output from *diff* indicating the changes that were made in *opb_complex1x.v* to create the new signals and instantiate the device under test.

```
107a108,114
>  wire   sl3_xferack;
>  wire   sl3_hwack;
>  wire sl3_fwack;
>  wire sl3_errack;
>  wire sl3_toutsup;
>  wire sl3_retry;
>  wire[ 0 : 31 ] sl_dbus_3;
270,273c278,281
<  assign  opb_xferack  = ((sl0_xferack  | sl1_xferack ) | sl2_xferack );
<  assign  opb_hwack  = ((sl0_hwack  | sl1_hwack ) | sl2_hwack );
<  assign  opb_fwack  = ((sl0_fwack  | sl1_fwack ) | sl2_fwack );
<  assign  opb_errack  = ((sl0_errack  | sl1_errack ) | sl2_errack );
---
>  assign  opb_xferack  = ((sl0_xferack  | sl1_xferack ) | sl2_xferack | sl3_xferack );
>  assign  opb_hwack  = ((sl0_hwack  | sl1_hwack ) | sl2_hwack | sl3_hwack);
>  assign  opb_fwack  = ((sl0_fwack  | sl1_fwack ) | sl2_fwack | sl3_fwack);
>  assign  opb_errack  = ((sl0_errack  | sl1_errack ) | sl2_errack |sl3_errack);
280,281c288,290
<  assign  opb_retry  = ((sl0_retry  | sl1_retry ) | sl2_retry );
<  assign  opb_toutsup  = ((sl0_toutsup  | sl1_toutsup ) | sl2_toutsup );
---
>  assign  opb_retry  = ((sl0_retry  | sl1_retry ) | sl2_retry | sl3_retry);
>  assign  opb_toutsup  = ((sl0_toutsup  | sl1_toutsup ) | sl2_toutsup | sl3_tousup);
>
286c295
<  always @ (m_dbus_en or m_dbus[0] or m_dbus[1] or m_dbus[2] or
   sl_dbus[0] or sl_dbus[1] or sl_dbus[2] or sl_dbus_en or opb_reset)
---
>  always @ (m_dbus_en or m_dbus[0] or m_dbus[1] or m_dbus[2] or
   sl_dbus[0] or sl_dbus[1] or sl_dbus[2] or sl_dbus[3] or sl_dbus_en or opb_reset)
379c388,392
<  end
---
>    end
>  always @ (sl_dbus_3)
>    begin
>     sl_dbus [ 3 ] <= sl_dbus_3 [ 0 : 31 ];
>    end
393a407,410
```

```
>    opbinterface dut (.OPB_clk(opb_clk), .OPB_select(opb_select),
     .OPB_RNW(opb_rnw), .OPB_hwXfer(opb_hwxfer),
>       .OPB_fwXfer(opb_fwxfer), .OPB_seqAddr(opb_seqaddr),
     .OPB_ABus(opb_abus), .OPB_DBus(opb_dbus),
>       .Sln_xferAck(sl3_xferack), .Sln_hwAck(sl3_hwack),
     .Sln_fwAck(sl3_fwack), .Sln_errAck(sl3_errack),
>       .Sln_toutSup(sl3_toutsup), .Sln_retry(sl3_retry),
     .Sln_DBusEn(sl_dbus_en [ 3 ]), .Sln_DBus(sl_dbus_3));
```

3.3 Creating a test

Creating a test consists of instantiating the bus models and controlling the models' behavior. This is accomplished by compiling files written in the bus functional language (bfl) with the bus functional compiler (BFC).

The toolkit comes with several sample bfl files in *CoreConnect/tk/OPB_TOOLKIT2X/SampleTest*. Open *sample.bfl* in your favorite text editor. *sample.bfl* instantiates a bus arbiter, a master device and a slave device and simulates communication between the master and the slave. Add communications to your device by adding read and write commands below the *set_device* instruction that instantiates the master. The commands below each *set_device* instruction are independent of the set of commands below other *set_device* instructions. Each command set is followed sequentially. A description of the bus functional language is given in [2] which can be found in *CoreConnect/tk/OPB_TOOLKIT2X*.

Note that you do not need to instantiate your own device in the bfl file.

Copy *sample.bfl* into it parent directory and type *BFC sample.bfl* to compile the file. Upon running BFC for the first time, several questions will be asked. Set Verilog as the target (you can also explore the other options). The default values for most options are fine. The location of the *plb_dcl*, *opb_dcl* and *dcr_dcl* files is *CoreConnect/tk/TOOLKIT_DCL*.

After compiling, *sample.v* will be created. Open *sample.v* and add the line

```
`include "opb_dcl.inc"
```

just below the *module* line so that it will compile properly with the correct definitions.

3.4 Running the simulation

Compile sample.v along with the *1x.v files and your design with a simulator of your choice (you may have to move the .inc files into the appropriate directories). Load *opb_complex* and *sample* as the two top-level modules for the simulation.

4 Adding the design to a MicroBlaze system

4.1 Creating the base system

Open the Xilinx EDK and select *File* → *New Project* → *Base System Builder* and generate a MicroBlaze system[2].

Locate directory of your new project. A subdirectory titled *pcores* will exist. Your design will be placed within this directory. Build the directory tree shown in figure 2 where *yourdesignname* is replaced by the name with which

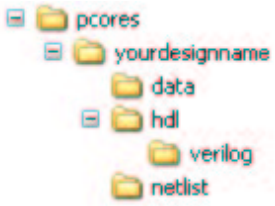

Figure 2: The pcore directory tree

your design will appear in the EDK.

[2]see the ECE532 labs if you unfamiliar with this process

Within the *verilog* directory place all your HDL files including any wrappers for black-box netlist files that may be included in your design.

Within the *netlist* directory, place any netlist files that you are including in your design. These may include *edn* files generated by CoreGen.

The *data* directory contains three text files that must be created: *yourdesignname_v2_1_0.bbd*, *yourdesignname_v2_1_0.mpd*, and *yourdesignname_v2_1_0.pao*. The string "_v2_1_0" in the file names are indicative of the syntax version used in the files and thus they are not free to be altered. The string "yourdesignname" in the file name must correspond to the name given to the parent directory.

.pao "Peripheral Analyze Order" Contains references to your HDL files in an order to resolve all dependancies.

.bbd "Black-Box Definition" A look up file containing references to netlist files.

.mpd "Microprocessor Peripheral Description" Contains a description of the core to be important including port connections and parameters.

Details regarding the syntax of these three files is given in [3]. Sample files are given below.

4.2 Peripheral Analyze Order

```
##############################################################################
##
## A Header section
##
##
##
##
## Some Comments
##
##############################################################################

lib yourdesignname sin_evaluator
lib yourdesignname add_19_19_20_bit
lib yourdesignname add_37_26_37_bit
lib yourdesignname c_k_shift
lib yourdesignname ClkGen
lib yourdesignname datamemory
lib yourdesignname genmult_7_21_22
lib yourdesignname genmult_16_18_20
lib yourdesignname genmult_20_7_28
lib yourdesignname mult_7_21_21
lib yourdesignname mult_16_18_19_bit
lib yourdesignname mult_20_7_27_bit
lib yourdesignname sub_19_19_20_bit
lib yourdesignname theta_shift
lib yourdesignname datafifo
lib yourdesignname Accumulator
lib yourdesignname DataSelector
lib yourdesignname InnerProduct
lib yourdesignname FunctionEvaluator
lib yourdesignname Pipeline
lib yourdesignname Controller
lib yourdesignname OpbInterface
```

The *pao* file contains references to all your HDL files including netlist wrappers. The order of the references reflects the order required to resolve dependencies with the top-level design listed last. Note that the references do not include the extension such as ".v".

4.3 Black-Box Definition

```
##############################################################################
```

```
##
## Comments
##
##
##
##
###############################################################################

Files
###############################################################################
add_19_19_20_bit.edn, add_37_26_37_bit.edn, c_k_shift.edn, datamemory.edn,
genmult_16_18_20.edn, genmult_20_7_28.edn, genmult_7_21_22.edn, sin_evaluator.edn,
sub_19_19_20_bit.edn, theta_shift.edn
```

The *bbd* file includes references to all the netlist files used by your design that should be in the *netlist* directory. The two required parts of the file are the line with the string "Files" followed by a comma delimited list of the netlist files. Note that, unlike in the *pao* file, the file names listed include their extensions.

4.4 Microprocessor Peripheral Description

```
##################################################################
##
## Comments
##
##
##
##################################################################

BEGIN OpbInterface

## Peripheral Options
OPTION IMP_NETLIST = TRUE
OPTION HDL = VERILOG
OPTION CORE_STATE = ACTIVE
OPTION ARCH_SUPPORT = virtex2:virtex2p
OPTION IPTYPE = PERIPHERAL
OPTION STYLE = MIX

## Bus Interfaces
BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL or Parameters for Verilog
PARAMETER C_BASEADDR = 0xFFFFFFFF, MIN_SIZE = 0x100, BUS = SOPB
PARAMETER C_HIGHADDR = 0x00000000, BUS = SOPB
PARAMETER OPB_WIDTH = 0x00000020

## Ports
PORT OPB_clk = "", DIR = I, SIGIS = CLK, BUS = SOPB, DEFAULT =
PORT OPB_RNW = OPB_RNW, DIR = I, BUS = SOPB, DEFAULT = OPB_RNW
PORT OPB_select = OPB_select, DIR = I, BUS = SOPB, DEFAULT = OPB_select
PORT OPB_hwXfer = "", DIR = I, DEFAULT = net_vcc
PORT OPB_fwXfer = "", DIR = I, DEFAULT = net_vcc
PORT OPB_seqAddr = OPB_seqAddr, DIR = I, BUS = SOPB, DEFAULT = OPB_seqAddr
PORT OPB_ABus = OPB_ABus, DIR = I, VEC = [0:(OPB_WIDTH-1)], BUS = SOPB, DEFAULT = OPB_ABus
PORT OPB_DBus = OPB_DBus, DIR = I, VEC = [0:(OPB_WIDTH-1)], BUS = SOPB, DEFAULT = OPB_DBus
PORT Sln_errAck = Sl_errAck, DIR = O, BUS = SOPB, DEFAULT = Sl_errAck
PORT Sln_retry = Sl_retry, DIR = O, BUS = SOPB, DEFAULT = Sl_retry
PORT Sln_toutSup = Sl_toutSup, DIR = O, BUS = SOPB, DEFAULT = Sl_toutSup
PORT Sln_xferAck = Sl_xferAck, DIR = O, BUS = SOPB, DEFAULT = Sl_xferAck
```

```
PORT Sln_DBus = Sl_DBus, DIR = O, VEC = [0:(OPB_WIDTH-1)], BUS = SOPB, DEFAULT = Sl_DBus
PORT Sln_hwAck = "", DIR = O, DEFAULT =
PORT Sln_fwAck = "", DIR = O, DEFAULT =
END
```

The meaning of each part of the file can be found in [3], however, below is a brief description of some notable lines.

The *BEGIN* line is used to indicate the file that includes the top-level module.

OPTION STYLE = MIX indicates that the design includes both HDL files and netlist files. If the design does not include any netlist files, the option should be changed to *OPTION STYLE = HDL*.

The lines beginning with *PARAMETER* are used to give the EDK access to parameters within the top-level module. The values given to the parameters within the *mpd* file are the default values. The parameters listed in the file can be altered using the EDK's GUI.

The *Ports* section of the file lists the ports of the top-level module and to what the ports should be connected when the imported into the EDK. Listed ports can be left unconnected to be connected to a system net from within the EDK as is the case with *Sln_hwAck* and *Sln_fwAck* in the above example.

4.5 Adding the core to the base system

If the EDK was open while you created the file in the *data* directory, the EDK will have to be restarted since it refreshes its list of possible peripherals upon startup.

With the EDK project that was created earlier open, select the menu item *Project → Add/Edit Cores...*.

A dialog box will appear. In the right will be a listbox of available peripherals. Scroll through this list to find the name given to your design in place of *yourdesignname* in the directory tree. Select your design and click the *Add* button so that it appears in the table on the left of the dialog box.

Change the *Base Address* and *High Address* corresponding to your design. This will be the address range available to your design in the memory addressable space. Changing values here alter the values of the *C_BASEADDR* and *C_HIGHADDR* parameters in the top-level module of the design.

Click on *Bus Connections* to switch tabs. Locate your design in the table on the left. Click within the box corresponding to the intersection between your design and *mb_opb* so that an "s" appears in the box. This connects your device to the MicroBlaze's OPB as a slave device.

Click on *Ports* to switch tabs. Several of your designs ports were connected automatically due to what was done in the *Bus Connections* tab. Additional ports must be connected. Locate your device and associated unconnected ports in the listbox on the right of the dialog box. Select the signals you which to connect and add them to the table on the left using the *<<Add* button. It is required that you add *OPB_clk*. If you created the OPB interface based on the older OPB specification used by the OPB toolkit the two signals *OPB_hwXfer* and *OPB_fwXfer* will need to be added. After adding the signals, they will be listed in the table on the left of the dialog box. Change the net name to which each is connected. *OPB_clk* should be connected to *sys_clk_s*. Connecting both *OPB_hwXfer* and *OPB_fwXfer* to *net_vcc* sends the signal to your device that all data sent to it will be as full-word transfers (32 bits). If this is inappropriate (that is, you desire data of different sizes will be sent), you may wish to alter your OPB interface to use the Byte-Enable scheme implemented by Xilinx (see section 6).

Click on *Parameters* to switch tabs. Selecting your design from the drop-down list in the upper right of the dialog box lists available parameters that were made available in the *.mpd* file. The values of these parameters can be changed by selecting them and adding them to the table in the left of the dialog.

Click *OK* to finish adding the device to the system.

4.6 Adding Software to the system

Adding software to the system is cover well in the Lab guides. It is notable, however, to ensure that optimizations are disabled for code communicating to your device.

5 System Level Simulation

The EDK can instantiate a model that includes the MicroBlaze which will run your software in simulation.

Before generating the system level simulation, ensure that the software on the generated system will work in simulation. The software to be used in simulation should be in "Executable" mode rather than using the XmdStub such that its execution begins immediately. It should be set to initialize the BRAMs as well.

The simulation does not include a model of the UART or the JTAG interface, so a program that interacts with either will not work. Thus your software should not include calls to printf, for example.

Set the simulation options via *Project Options* which can be found in the *Options* menu. In the HDL and Simulation tab, select *Verilog* as the *HDL* and *ModelSim* as the *Simulator Compile Script*. Set up the correct paths to the Xilinx and EDK libraries. The Xilinx libraries usually come installed with ISE and are located in *Xilinx/verilog/mti_se*. The EDK library will need to be generated if they have not been already. Generating the libraries is addressed in Xilinx's answer database, record number: 18386.

Generate the simulation model wrappers by running the menu item *Tools → Sim Model Generation*. Running the menu item *Hardware Simulation* will open up ModelSim in the correct environment to begin the simulation.

Once ModelSim is open, type *do system.do* to execute the generated script that compiles all the necessary HDL files.

Type *vsim -t ps work.system work.system_conf work.glbl* to start the simulation. *work.system* is the top-level module, *work.system_conf* is the module used to initialize the BRAMs with your software and *work.glbl* contains required global signals. You may need to add options such as "-L XILINXCORELIB_VER" if you use CoreGen components.

You will need to drive the clock and the reset signal of the system yourself. This can be done with the following commands which can be placed in a script for convenience.

```
force -freeze sim:/system/sys_rst 0 0
force -freeze sim:/system/sys_clk 1 0, 0 {50 ns} -r 100000
run 100 ns
force -freeze sim:/system/sys_rst 1 0
```

The simulation can then be run as long as desired.

6 Migrating to the Xilinx OPB standard

As mentioned earlier, Xilinx uses a slightly different OPB standard than the standard the IBM's toolkit available to connect. Xilinx calls devices that use the older OPB standar "legacy devices" in their documenation. The above method of connecting a legacy device is acceptable if it can be assured that the software interacting with the device will only send data in full word transfers. This may not always be the ideal. In this case, the OPB interface must be adjusted to support the OPB standard implemented by Xilinx.

The Xilinx standard does away with the following output slave signals: *Sln_hwAck, Sln_fwAck, Sln_DBusEn*

The Xilinx standard replaces the *OPB_hwXfer, OPB_fwXfer* signals with a single 4-bit wide *OPB_BE* byte-enable signal. The *OPB_BE* signal extends the dynamic bus sizing available in the OPB standard. Each bit of *OPB_BE* corresponds to a "byte lane" in the data bus signals. A bit value of 1 corresponds to the byte lane having valid data. Bit 0 of *OPB_BE* corresponds to valid data in the bits 0:7 of *OPB_DBus*, and so on. Thus a full-word transfer would be indicated by *OPB_BE* being equal to the binary value 1111. A visual presentation of valid *OPB_BE* values can be found in [4] within the section *Specifications for OPB Usage in Xilinx-developed OPB Devices*.

These changes can usually be made with relative ease. A new port entry must be added to connect the *OPB_BE* signal in the .mpd file.

References

[1] IBM. *On-Chip Peripheral Bus Architecture Specifications Version 2.1.*

[2] IBM. *OPB Bus Functional Model Toolkit User's Manual.*

[3] Xilinx. *Embedded System Tools Guide.*

[4] Xilinx. *MicroBlaze Hardware Reference Guide.*