The On-Chip Peripheral Bus

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu

Columbia University
Spring 2006
Developed by IBM

Part of their CoreConnect architecture designed for integrating on-chip “cores”

Something like “PCI on a chip”

Spec. allows for 32- or 64-bit addresses and data

Xilinx Microblaze variant uses 32-bit only
Intended System Architecture

Source: IBM
Masters and Slaves

Most bus protocols draw a distinction between

**Masters**: Can initiate a transaction, specify an address, etc. E.g., the Microblaze

**Slaves**: Respond to requests from masters, can generate return data. E.g., a video controller

Most peripherals are slaves.

Masters speak a more complex protocol

Bus arbiter decides which master gains control
For OPB slave devices,

prefix       meaning
OPB_          Signals from OPB bus logic to slave
Sln_          Signals from slave to OPB
OPB slave signals (Xilinx)

<table>
<thead>
<tr>
<th>Slave</th>
<th>OPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sln_xferAck</td>
<td>OPB_Clk</td>
</tr>
<tr>
<td>Sln_toutSup</td>
<td>OPB_Rst</td>
</tr>
<tr>
<td>Sln_retry</td>
<td>OPB_select</td>
</tr>
<tr>
<td>Sln_DBus[0:31]</td>
<td>OPB_RNW</td>
</tr>
<tr>
<td>Sln_errAck</td>
<td>OPB_SeqAddr</td>
</tr>
</tbody>
</table>

OPB_BE[0:3]    
OPB_ABus[0:31]  
OPB_DBus[0:31]
<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPB_Clk</td>
<td>Bus clock: master synchronization</td>
</tr>
<tr>
<td>OPB_Rst</td>
<td>Global asynchronous reset</td>
</tr>
<tr>
<td>OPB_ABus[0:31]</td>
<td>Address</td>
</tr>
<tr>
<td>OPB_BE[0:3]</td>
<td>Byte enable</td>
</tr>
<tr>
<td>OPB_DBus[0:31]</td>
<td>Data to slave</td>
</tr>
<tr>
<td>OPB_RNW</td>
<td>1=read from slave, 0=write to slave</td>
</tr>
<tr>
<td>OPB_select</td>
<td>Transfer in progress</td>
</tr>
<tr>
<td>OPB_seqAddr</td>
<td>Next sequential address pending (unused)</td>
</tr>
<tr>
<td>SIn_DBus[0:31]</td>
<td>Data from slave. Must be 0 when inactive</td>
</tr>
<tr>
<td>SIn_xferAck</td>
<td>Transfer acknowledge. OPB_select → 0</td>
</tr>
<tr>
<td>SIn_retry</td>
<td>Request master to retry operation (=0)</td>
</tr>
<tr>
<td>SIn_toutSup</td>
<td>Suppress slave time-out (=0)</td>
</tr>
<tr>
<td>SIn_errAck</td>
<td>Signal a transfer error occurred (=0)</td>
</tr>
</tbody>
</table>
The OPB and the Microblaze are big-endian:

0 is the most significant bit, 31 is the least

Bytes and halfwords are left-justified:

<table>
<thead>
<tr>
<th>Byte</th>
<th>msb</th>
<th>lsb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>23</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halfword</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>0</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
entity opb_peripheral is
  generic (  
    C_BASEADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_OPB_AWIDTH : integer := 32;
    C_OPB_DWIDTH : integer := 32);
  port (  
    OPB_ABus   : in std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE     : in std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk    : in std_logic;
    OPB_DBus   : in std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW    : in std_logic;
    OPB_Rst    : in std_logic;
    OPB_select : in std_logic;
    OPB_seqAddr: in std_logic;
    Sln_DBus   : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck : out std_logic;
    Sln_retry  : out std_logic;
    Sln_toutSup: out std_logic;
    Sln_xferAck: out std_logic);
end entity opb_peripheral;
Typical OPB Read Cycle Timing

OPB_CLK

OPB_select

OPB_ABus

OPB_BE

OPB_RNW

Sln_DBus 0 valid 0

Sln_xferAck

OPB signals arrive late; DBus and xferAck needed early.
Typical OPB Write Cycle Timing

- OPB_CLK
- OPB_select
- OPB_ABus
- OPB_BE
- OPB_RNW
- OPB_DBus
- Sln_xferAck
Xilinx Rules

OPB data and address busses are 32 bits
Byte-wide peripherals use data byte 0 and word-aligned addresses (0, 4, ...)
Peripherals output 0 on everything when inactive
Xilinx does not support complete IBM OPB spec: Dynamic bus sizing is not used
Let’s design a peripheral that contains one of the BRAM blocks. Reading and writing this peripheral will turn into reading and writing the BRAM.
Read Cycle

**OPB_CLK**

**OPB_select**

**OPB_ABus**

**OPB_RNW**

**CS**

**State**

**Idle**  **Selected**  **Read**  **Xfer**  **Idle**

**RST**

**RAM DO**

0  0

**Sln_DBus**

0  0

**Sln_xferAck**
Back-to-back Read Cycles

- OPB_CLK
- OPB_select
- OPB_ABus
- OPB_RNW
- CS
- State: Idle, Selected, Read, Xfer, Idle
- RST
- RAM DO
- SIn_DBus
- SIn_xferAck

The diagram illustrates the timing and state transitions for back-to-back read cycles in an on-chip peripheral bus (OPB) system.
Aborted Read Cycle

The diagram illustrates the timing sequence for an aborted read cycle on the On-Chip Peripheral Bus (OPB). The signals include:

- **OPB_CLK**: Clock signal.
- **OPB_select**: Select signal.
- **OPB_ABus**: Address bus.
- **OPB_RNW**: Read/Write control.
- **CS**: Chip select.
- **State**: State transitions through Idle, Selected, Read, and Idle.
- **RST**: Reset signal.
- **RAM DO**: Data outputs from RAM.
- **Sln_DBus**: Bus lines.
- **Sln_xferAck**: Transfer acknowledgment.
The On-Chip Peripheral Bus – p. 20/30
signal WE, RST : std_logic;
signal RAM_DI, RAM_DO
  : std_logic_vector(0 to RAM_DWIDTH-1);
signal ABus
  : std_logic_vector(0 to RAM_AWIDTH-1);

RAMBlock : RAMB4_S8
port map (
  DO  => RAM_DO,
  ADDR => ABus,
  CLK  => OPB_Clk,
  DI   => RAM_DI,
  EN   => '1',
  RST  => RST,
  WE   => WE);
register_opb_inputs: process (OPB_Clk, OPB_Rst) begin
  if OPB_Rst = '1' then
    RAM_DI <= (others => '0');
    ABus <= (others => '0');
    RNW <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    RAM_DI <= OPB_DBus(0 to RAM_DWIDTH-1);
    ABus <=
      OPB_ABus(C_OPB_AWIDTH-3-(RAM_AWIDTH-1) to C_OPB_AWIDTH-3);
    RNW <= OPB_RNW;
  end if;
end process register_opb_inputs;
register_opb_outputs: process (OPB_Clk, OPB_Rst) begin
  if OPB_Rst = '1' then
    Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if output_enable = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= RAM_DO;
    else
      Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
    end if;
  end if;
end process register_opb_outputs;
```vhdl
chip_select <=
  '1' when OPB_select = '1' and
  OPB_ABus(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) =
  C_BASEADDR(0 to C_OPB_AWIDTH-3-RAM_AWIDTH)
else '0';
```
constant STATE_BITS : integer := 3;
constant Idle
  : std_logic_vector(0 to STATE_BITS-1) := "000";
constant Selected
  : std_logic_vector(0 to STATE_BITS-1) := "001";
constant Read
  : std_logic_vector(0 to STATE_BITS-1) := "011";
constant Xfer
  : std_logic_vector(0 to STATE_BITS-1) := "111";

signal present_state, next_state
  : std_logic_vector(0 to STATE_BITS-1);
fsm_seq : process(OPB_Clk, OPB_Rst)
begin
    if OPB_Rst = '1' then
        current_state <= Idle;
    elsif OPB_Clk'event and OPB_Clk = '1' then
        current_state <= next_state;
    end if;
end process fsm_seq;
fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select, RNW)
begin
  RST <= '1';
  WE <= '0';
  output_enable <= '0';
  if OPB_RST = '1' then
    next_state <= Idle;
  else
    case present_state is
    when Idle =>
      if chip_select = '1' then
        next_state <= Selected;
      else
        next_state <= Idle;
      end if;
    end case;
  end if;
end begin;
when Selected =>
  if OPB_Select = '1' then
    if RNW = '1' then
      RST <= '0';
      next_state <= Read;
    else
      WE <= '1';
      next_state <= Xfer;
    end if;
  else
    next_state <= Idle;
  end if;

when Read =>
  if OPB_Select = '1' then
    output_enable <= '1';
    next_state <= Xfer;
  else
    next_state <= Idle;
  end if;
-- State encoding is critical here:
-- xfer must only be true here
when Xfer =>
    next_state <= Idle;

when others =>
    next_state <= Idle;
end case;
end if;
end process fsm_comb;
Xilinx Processor IP Reference Guide

IBM On-Chip Peripheral Bus Architecture Specification