The On-Chip Peripheral Bus

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

Physical Implementation

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

Naming Conventions

For OPB slave devices,

prefix  meaning
OPB_  Signals from OPB bus logic to slave
Sln_  Signals from slave to OPB

OPB slave signals (Xilinx)

Slave

Sln_xferAck
Sln_ToutSup
Sln_retry
Sln_DBus[0:31]
Sln_errAck

OPB

OPB_Clk
OPB_Rst
OPB_select
OPB_RW
OPB_SeqAddr
OPB_BE[0:3]
OPB_ABus[0:31]
OPB_DBus[0:31]

OPB Signals

OPB_Clk  Bus clock: master synchronization
OPB_Rst  Global asynchronous reset
OPB_ABus[0:3]  Address
OPB_BE[0:3]  Byte enable
OPB_DBus[0:31]  Data to slave
OPB_RW  1=read from slave, 0=write to slave
OPB_select  Transfer in progress
OPB_SeqAddr  Next sequential address pending (unused)

Bytes, Bits, and Words

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>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>15</td>
<td>16</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th></th>
<th></th>
<th></th>
<th>15</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Halfword</th>
<th></th>
<th></th>
<th></th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
In VHDL

entity opb_peripheral is
  generic (
    C_BASEADR : std_logic_vector(0 to 31) := "X""FFFFFFF"";
    C_HIGHADR : std_logic_vector(0 to 31) := "X""0000000"";
    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_RNW : in std_logic;
    OPB_CLK : in std_logic;
    OPB_Aselect : in std_logic;
    OPB_RSelect : in std_logic;
    OPB_RWAddr : in std_logic;
    Sln_DBus : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_xferAck : out std_logic;
    Sln_xferAck : out std_logic;
  end entity opb_peripheral;

Typical OPB Read Cycle Timing

OPB signals arrive late; DBus and xferAck needed early.

Typical OPB Write Cycle Timing

Back-to-back Read Cycles

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.

Abort Read Cycle

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

Xilinx Rules

Designing an OPB Peripheral

The On-Chip Peripheral Bus – p. 11/30
The On-Chip Peripheral Bus – p. 14/30
The On-Chip Peripheral Bus – p. 16/30
The On-Chip Peripheral Bus – p. 17/30
The On-Chip Peripheral Bus – p. 19/30
The On-Chip Peripheral Bus – p. 20/30
The On-Chip Peripheral Bus – p. 21/30
The On-Chip Peripheral Bus – p. 22/30
The On-Chip Peripheral Bus – p. 23/30
The On-Chip Peripheral Bus – p. 24/30
The On-Chip Peripheral Bus – p. 25/30
The On-Chip Peripheral Bus – p. 26/30
The On-Chip Peripheral Bus – p. 27/30
The On-Chip Peripheral Bus – p. 28/30
The On-Chip Peripheral Bus – p. 29/30
The On-Chip Peripheral Bus – p. 30/30
The On-Chip Peripheral Bus – p. 10/30
The On-Chip Peripheral Bus – p. 19/30
The On-Chip Peripheral Bus – p. 20/30
### Input Registers

```vhdl
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;
```

### Output Registers

```vhdl
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;
```

### Chip Select

```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';
```

### FSM: Declarations

```vhdl
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: Sequential

```vhdl
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: Combinational

```vhdl
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;
    when Selected =>
      next_state <= Read;
    when Read =>
      next_state <= Xfer;
    when Xfer =>
      next_state <= Write;
    when others =>
      next_state <= Idle;
    end case;
  end if;
end process fsm_comb;
```
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:
  when Xfer =>
    next_state <= Idle;
  when others =>
    next_state <= Idle;
  end case;
end if;
end process fsm_comb;