LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) (1.04a)

DS284 June 22, 2011 **Product Specification**

Introduction

The LogiCORE™ IP ChipScope™ Pro Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. Two different kinds of inputs and two different kinds of outputs are available, both of which are customizable in size to interface with the FPGA design.

Features

- Provides virtual LEDs and other status indicators through asynchronous and synchronous input ports
- Has activity detectors on input ports to detect rising and falling transitions between samples
- Provides virtual buttons and other controls through asynchronous and synchronous output ports
- For synchronous outputs, provides ability to define a pulse train, which is a 16-cycle train of ones and zeros that run at design speed.

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family (1)	Kintex®-7(3), Virtex®-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA				
Supported User Interfaces	Not applicable.				
Provided with Core					
	Resources				**Frequency**
Configuration (4)	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max Freq**
Config1	60	87	0	0	399.808 MHz
Config2	131	291	0	0	399.808 MHz
Config3	227	543	0	0	399.808 MHz
Documentation	Product Specification User Guide				
Design Files	Netlist				
Example Design	Verilog /VHDL				
Test Bench	Not Provided				
Constraints File	Xilinx Constraints File				
Simulation Model	Not Provided				
Tested Design Tools (2)					
Design Entry Tools	CORE Generator tool, XPS				
Simulation	Not Provided				
Synthesis Tools	Not Provided.				
Support					
Provided by Xilinx, Inc.					

Notes:

1. For a complete listing of supported derivative devices, see the IDS Embedded Edition Derivative Device Support.
2. For a listing of the supported tool versions, see the ISE Design Suite 13: Release Note Guide.
3. For more information, see DS180 7 Series FPGAs.
4. Overview.For configuration details, see Table 6, page 11.

Functional Description

The VIO core is a customizable core that can both monitor and drive internal FPGA signals in real time. Unlike the ILA and IBA cores, no on-chip or off-chip RAM is required.

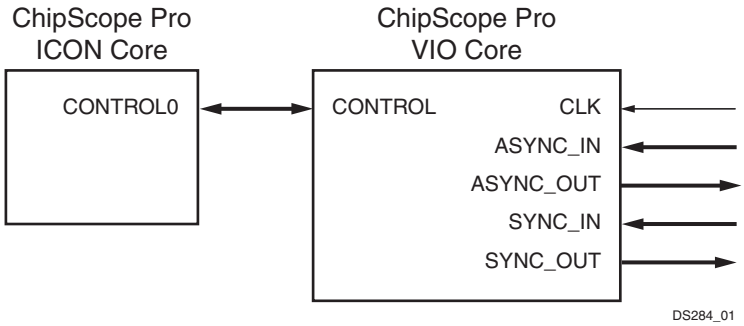

Figure 1: **VIO Core Connection to ICON Core**

Four types of signals are available in the VIO core:

- Asynchronous inputs:

  These are sampled using the JTAG clock signal that is driven from the JTAG cable. The input values are read back periodically and displayed in the Analyzer.
- Synchronous inputs:

  These are sampled using the design clock. The input values are read back periodically and displayed in the Analyzer.
- Asynchronous outputs:

  These are user-defined in the Analyzer and driven out of the core to the surrounding design. A logical 1 or 0 value can be defined for individual asynchronous outputs.
- Synchronous outputs:

  These are user-defined in the Analyzer, synchronized to the design clock and driven out of the core to the surrounding design. A logical 1 or 0 can be defined for individual synchronous outputs. Pulse trains of 16 clock cycles worth of ones and zeros can also be defined for synchronous outputs.

Activity Detectors

Every VIO core input has additional cells to capture the presence of transitions on the input. Since the design clock will most likely be much faster than the sample period of the Analyzer, it's possible for the signal being monitored to transition many times between successive samples. The activity detectors capture this behavior and the results are displayed along with the value in the Analyzer.

In the case of a synchronous input, activity cells capable of monitoring for asynchronous and synchronous events are used. This feature can detect glitches as well as synchronous transitions on the synchronous input signal.

Pulse Trains

Every VIO synchronous output has the ability to output a static 1, a static 0, or a pulse train of successive values. A pulse train is a 16-clock cycle sequence of 1s and 0s that drive out of the core on successive design clock cycles. The pulse train sequence is defined in the Analyzer and is executed only one time after it is loaded into the core.

VIO Interface Ports

The I/O signals of the VIO core shown in Table 1 consist of the control bus to ICON, as well as the four interface ports and the design clock. None of the ports are required, but at least one port must be enabled.

Table 1: **VIO Interface Ports**

Port Name	Direction	Description
ASYNC_IN[<*m*>-1:0]	IN	Asynchronous input port of width <*m*>. Optional (depends on enable_asynchronous_input_port). You must declare this port as a vector. For a one-bit port, use ASYNC_IN[0:0].
ASYNC_OUT[<*m*>-1:0]	OUT	Asynchronous output port of width <*m*>. Optional (depends on enable_asynchronous_output_port). This port must be declared as a vector. For a one-bit port, use ASYNC_OUT[0:0].
CLK	IN	Design clock used to register synchronous input or output ports. Optional (depends on enable_synchronous_input_port and/or enable_synchronous_output_port)
CONTROL[35:0]	INOUT(1)	Control bus to ICON core. Mandatory.
SYNC_IN[<*m*>-1:0]	IN	Synchronous input port of width <*m*>. Optional (depends on enable_synchronous_input_port). This port must be declared as a vector. For a one-bit port, use SYNC_IN[0:0].
SYNC_OUT[<*m*>-1:0]	OUT	Synchronous output port of width <*m*>. Optional (depends on enable_synchronous_output_port). This port must be declared as a vector. For a one-bit port, use SYNC_OUT[0:0].

Notes:

1. For projects created using Xilinx Platform Studio, the direction for CONTROL ports is IN.

Restrictions

A maximum of 15 VIO cores can be used in a single design.

CORE Generator™

The CORE Generator tool provides the ability to define and generate a customized VIO core for adding virtual inputs and outputs to your HDL designs. You can customize the virtual inputs and outputs to be synchronous to a particular clock in your design or to be completely asynchronous with respect to any clock domain in your design. You can also customize the number of input and output signals used by the VIO core.

Entering the Component Name

The Component Name field can consist of any combination of alpha-numeric characters in addition to the underscore symbol. However, the underscore symbol cannot be the first character in the component name.

Generating an Example Design

The VIO core generator normally generates standard Xilinx CORE Generator output files only, such as netlist and instantiation template files. To generate an example design that uses the VIO core, in addition to the normal generated files, select the Generate Example Design check box. This parameter is stored as example_design in the generated XCO parameter file.

Asynchronous Input Port

The VIO core includes an asynchronous input port when the Enable Asynchronous Input Port check box is selected. When enabled, you can specify a port width up to 256 bits by typing a value in the Width text field. The two parameters are stored as enable_asynchronous_input_port and asynchronous_input_port_width, respectively, in the generated XCO parameter file.

Asynchronous Output Port

The VIO core includes an asynchronous output port when the Enable Asynchronous Output Port check box is selected. When enabled, you can specify a port width up to 256 bits by typing a value in the Width text field. The two parameters are stored as enable_asynchronous_output_port and asynchronous_output_port_width, respectively, in the generated XCO parameter file.

Synchronous Input Port

The VIO core includes a synchronous input port when the Enable Synchronous Input Port check box is selected. When enabled, you can specify a port width up to 256 bits by typing a value in the Width text field. The two parameters are stored as enable_synchronous_input_port and synchronous_input_port_width, respectively, in the generated XCO parameter file.

Synchronous Output Port

The VIO core includes a synchronous output port when the Enable Synchronous Output Port check box is selected. When enabled, you can specify a port width up to 256 bits by typing a value in the Width text field. The two parameters are stored as enable_synchronous_output_port and synchronous_output_port_width, respectively, in the generated XCO parameter file.

Inverting the Clock Edge

The VIO core can use either a non-inverted or inverted CLK signal to acquire and generate data on the synchronous input and output signals, respectively. The Invert Clock Edge check box is used to invert the CLK signal that is coming into the VIO core. This parameter is stored as invert_clock_input in the generated XCO parameter file.

Note: The clock can only be inverted if synchronous inputs and/or outputs are used.

Generating the Core

After entering the VIO core parameters, click Generate to create the VIO core files. After the VIO core has been generated, a list of files that are generated will appear in a separate window called "Readme <corename>".

Using the VIO Core

To instantiate the example VIO core HDL files into your design, use the following guidelines to connect the VIO core port signals to various signals in your design:

- Connect the VIO core's CONTROL port signal to an unused control port of the ICON core instance in the design
- Connect all unused bits of the VIO core's asynchronous and synchronous input signals to a "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- For best results, make sure the synchronous input source signals are synchronous to the VIO clock signal (CLK); also make sure the synchronous output sink signals are synchronous to the VIO clock signal (CLK)

Example 1: VIO connection in VHDL and show how the VIO core is connected in vhdl and verilog respectively. Note how the control bus control0 is attached to the control port of the VIO. In the Verilog example an empty module declaration is created for the ICON and VIO module. This is used as a black box declaration so that the synthesis tool properly accounts for the generated netlists.

Example 1: VIO connection in VHDL

```
entity example_chipscope_vio is
port (
    clk_i : in std_logic
    );
end example_chipscope_icon;

architecture vio_arch of example_chipscope_vio is
  ------------------------------------------------------------------
  --
  --  Component declarations
  --
  ------------------------------------------------------------------
 component chipscope_icon
   port (
      CONTROL0 : inout std_logic_vector(35 downto 0));
 end component;

  component chipscope_vio
    port (
       CONTROL →: inout std_logic_vector(35 downto 0);
       CLK →: in std_logic;
       ASYNC_IN →: in  std_logic_vector(7 downto 0);
       ASYNC_OUT →: out std_logic_vector(7 downto 0);
       SYNC_IN →std_logic_vector(7 downto 0);
       SYNC_OUT →out std_logic_vector(7 downto 0)
    );
  end component;

------------------------------------------------------------------
  --  Local Signals
------------------------------------------------------------------
  signal control_0 →:std_logic_vector (35 downto  0);
  signal async_i →  :std_logic_vector (7 downto  0);
  signal async_o →:std_logic_vector (7 downto  0);
  signal sync_i →:std_logic_vector (7 downto  0);
  signal sync_o →:std_logic_vector (7 downto  0);
------------------------------------------------------------------
```

```
  --
  --  ICON Pro core instance
  --
 ------------------------------------------------------------------
 ICON_inst:  chipscope_icon
   port map (
     CONTROL0 => control0);

 ------------------------------------------------------------------
  --
  --  VIO Pro core instance
  --
 ------------------------------------------------------------------
  VIO_inst : chipscope_vio
    port map (
CONTROL →=> control_0,
       CLK →=> clk_i,
       ASYNC_IN →=> async_i,
       ASYNC_OUT →=> async_o,
       SYNC_IN →=> sync_i,
       SYNC_OUT →=> sync_o
    );
```

Example 2: VIO connection in Verilog

```
module example_chipscope_vio (
       input clk_i
       );
 //------------------------------------------------------------------
 //  Local Signals
 //------------------------------------------------------------------
  wire [35:0] control_0;
  wire [7:0] async_i;
  wire [7:0] async_o
  wire [7:0] sync_i;
  wire [7:0] sync_o;

 //------------------------------------------------------------------
 //
 //  ICON Pro core instance
 //
 //------------------------------------------------------------------
 chipscope_icon ICON_inst
    (
     .CONTROL0(control0));
 //------------------------------------------------------------------
 //
 //  VIO Pro core instance
 //
 //------------------------------------------------------------------
 chipscope_vio VIO_inst0
   (
     .CONTROL(control0),
     .CLK(clk_i),
     .ASYNC_IN(async_i),
     .ASYNC_OUT(async_o),
     .SYNC_IN(sync_i),
```

```
     .SYNC_OUT(sync_o));
endmodule

//-----------------------------------------------------------------
//
//  ICON Pro core module declaration
//
//-----------------------------------------------------------------
module chipscope_icon
  (
    inout [35:0] CONTROL0);
endmodule

//-----------------------------------------------------------------
//
//  VIO Pro core module declaration
//
//-----------------------------------------------------------------
module chipscope_vio
  (
  inout [35:0] CONTROL,
  input [7:0] ASYNC_IN,
  output [7:0] ASYNC_OUT,
  input [7:0] SYNC_IN,
  output [7:0] SYNC_OUT);
endmodule
```

Xilinx Platform Studio

Using the VIO Core in the Embedded Development Kit (EDK)

The VIO core can be inserted into an embedded processor design using the EDK. In this case, the VIO core depends on ICON and OPB_MDM component instances already being in the design, as shown in Figure 2.

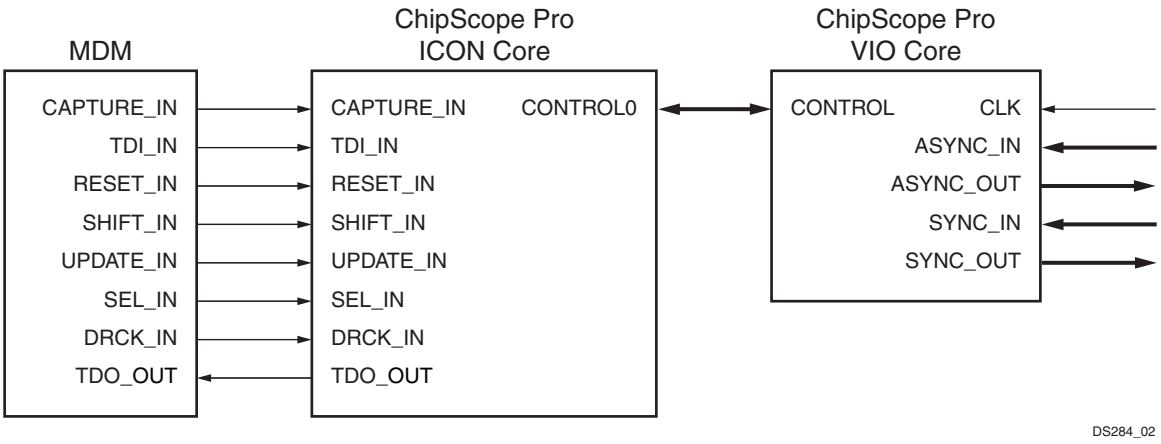

Figure 2: **VIO Core Connection to ICON Core**

In EDK, the VIO core is integrated into the tool using a Tcl script. When the EDK Hardware Platform Generator (Platgen) tool is run, the Tcl script is called and the script internally calls CORE Generator™ in command line mode. The Tcl script provides the CORE Generator an arguments file (.xco) to generate the VIO core netlist. The Tcl script also generates an HDL wrapper to match the VIO ports based on the core parameters found in Table 2.

Table 2: **EDK-specific Parameters**

Parameter Name	Allowable Values	Default Value	Description
c_async_input_enable	0, 1	0	0 = disable port, 1 = enable port
c_async_input_width	1-256	8	Asynchronous input port width, if used
c_async_output_enable	0, 1	0	0 = disable port, 1 = enable port
c_async_output_width	1-256	8	Asynchronous output port width, if used
c_family	virtex4, virtex5, virtex6, virtex6l, virtex7, kintex7, spartan3, spartan3a, spartan3adsp, spartan3e, spartan6, spartan6l, aspartan3, aspartan3a, aspartan3adsp, aspartan3e, aspartan6, avirtex4, qspartan6, qspartan6l, qrvirtex4, qspartan6, qspartan6l, qvirtex4, qvirtex5, qvirtex6	N/A	Device family to use.
c_rising_clock_edge	0, 1	1	Edge of input clock to use. 0 = falling edge, 1 = rising edge
c_sync_input_enable	0, 1	0	0 = disable port, 1 = enable port
c_sync_input_width	1-256	8	Synchronous input port width, if used
c_sync_output_enable	0, 1	0	0 = disable port, 1 = enable port
c_sync_output_width	1-256	8	Synchronous output port width, if used
c_use_srl16s	0, 1	1	0 = do no use SRL16s, 1 = use SRL16s

The Xilinx Synthesis Technology (XST) tool is used to synthesize the wrapper HDL generated for the VIO core. The NGC netlist outputs from the XST tool and the ChipScope Pro Core Generator are subsequently incorporated into the Xilinx ISE tool suite for actual device implementation.

Ports and Parameters

Ports

VIO Interface Ports

The I/O signals of the VIO core shown in Table 3 consist of the control bus to ICON, as well as the four interface ports and the design clock. None of the ports are required, but at least one port must be enabled.

Table 3: **VIO Interface Ports**

Port Name	Direction	Description
ASYNC_IN[<*m*>-1:0]	IN	Asynchronous input port of width <*m*>. Optional (depends on enable_asynchronous_input_port). You must declare this port as a vector. For a one-bit port, use ASYNC_IN[0:0].
ASYNC_OUT[<*m*>-1:0]	OUT	Asynchronous output port of width <*m*>. Optional (depends on enable_asynchronous_output_port). This port must be declared as a vector. For a one-bit port, use ASYNC_OUT[0:0].
CLK	IN	Design clock used to register synchronous input or output ports. Optional (depends on enable_synchronous_input_port and/or enable_synchronous_output_port)
CONTROL[35:0]	INOUT(1)	Control bus to ICON core. Mandatory.
SYNC_IN[<*m*>-1:0]	IN	Synchronous input port of width <*m*>. Optional (depends on enable_synchronous_input_port). This port must be declared as a vector. For a one-bit port, use SYNC_IN[0:0].
SYNC_OUT[<*m*>-1:0]	OUT	Synchronous output port of width <*m*>. Optional (depends on enable_synchronous_output_port). This port must be declared as a vector. For a one-bit port, use SYNC_OUT[0:0].

Notes:

1. For projects created using Xilinx Platform Sturdio, the direction for CONTROL ports is IN.

Parameters

CORE Generator Parameters

XCO Parameters

Table 4: **XCO Parameters**

Parameter Name	Allowable Values	Default Value	Description
component_name	String with A-z, 0-9, and _ (underscore)	vio	Name of instantiated component
enable_asynchronous_input_port	true, false	false	Enables the ASYNC_IN port
enable_asynchronous_output_port	true, false	false	Enables the ASYNC_OUT port
enable_synchronous_input_port	true, false	false	Enables the SYNC_IN port
enable_synchronous_output_port	true, false	false	Enables the SYNC_OUT port
asynchronous_input_port_width	1-256	8	Width of the ASYNC_IN port
asynchronous_output_port_width	1-256	8	Width of the ASYNC_OUT port
synchronous_input_port_width	1-256	8	Width of the SYNC_IN port
synchronous_output_port_width	1-256	8	Width of the SYNC_OUT port
invert_clock_input	true, false	false	Invert the design clock input
example_design	False = do not generate sample, True = generate example	false	Enable generration of an example design for the core.

XPS Parameters

Table 5: **EDK-specific Parameters**

Parameter Name	Allowable Values	Default Value	Description
c_async_input_enable	0, 1	0	0 = disable port, 1 = enable port
c_async_input_width	1-256	8	Asynchronous input port width, if used
c_async_output_enable	0, 1	0	0 = disable port, 1 = enable port
c_async_output_width	1-256	8	Asynchronous output port width, if used
c_family	virtex4, virtex5, virtex6, virtex6l, virtex7, kintex7, spartan3, spartan3a, spartan3adsp, spartan3e, spartan6, spartan6l, aspartan3, aspartan3a, aspartan3adsp, aspartan3e, aspartan6, avirtex4, qspartan6, qspartan6l, qrvirtex4, qspartan6, qspartan6l, qvirtex4, qvirtex5, qvirtex6	N/A	Device family to use.
c_rising_clock_edge	0, 1	1	Edge of input clock to use. 0 = falling edge, 1 = rising edge
c_sync_input_enable	0, 1	0	0 = disable port, 1 = enable port
c_sync_input_width	1-256	8	Synchronous input port width, if used
c_sync_output_enable	0, 1	0	0 = disable port, 1 = enable port
c_sync_output_width	1-256	8	Synchronous output port width, if used
c_use_srl16s	0, 1	1	0 = do no use SRL16s, 1 = use SRL16s

Performance And Resource Utilization

The performance and Utiliztion data is shown in Table 6.

Table 6: **Performance and Utilization for Specific Configuration Details**

Configuration Name	Device	VIO setup
Config1	Xc5vlx20t-2ff323	Asynchonous Input Port (7:0)
Config2	Xc5vlx20t-2ff323	Asynchonous Input Port (7:0), Synchronous Input Port (7:0), Asynchonous Output Port (7:0), Synchonous Output Port (7:0)
Config3	Xc5vlx20t-2ff323	Asynchonous Input Port (15:0), Synchronous Input Port (15:0), Asynchonous Output Port (15:0), Synchonous Output Port (15:0)

Verification

Xilinx has verified the VIO core in a proprietary test environment, using an internally developed bus functional model.

References

1. More information on the ChipScope Pro software and cores is available in the Software and Cores User Guide, located at http://www.xilinx.com/documentation.
2. Information about hardware debugging using ChipScope Pro in EDK is available in the Platform Studio online help, located at http://www.xilinx.com/documentation.

Information about hardware debugging using ChipScope Pro in System Generator for DSP is available in the Xilinx System Generator for DSP User Guide, located at http://www.xilinx.com/documentation.

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY.*

Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx ISE® Design Suite Embedded Edition software under the terms of the Xilinx End User License. The core is generated using the Xilinx ISE Design Suite software. For more information, visit the Chipscope VIO page.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

Revision History

Date	Version	Description of Revisions
03/24/08	1.0	Release 10.1 (Initial Xilinx release).
09/19/08	2.0	Release 10.1, Service Pack 3.
04/07/09	3.0	Release 11.1.
06/24/09	3.1	Release 11.2.
09/16/09	3.1.1	Corrections to document. Published with 11.3 software release.
6/22/11	3.2	Updated to v1.04a for the 13.2 release.

Notice of Disclaimer