

ChipScope ILA Tools Tutorial

*(for ChipScope ILA
Software v4.2i)*

UG044 / PN 0401957 (v4.2.2) July 24, 2003





"Xilinx" and the Xilinx logo shown above are registered trademarks of Xilinx, Inc. Any rights not expressly granted herein are reserved. CoolRunner, RocketChips, Rocket IP, Spartan, StateBENCH, StateCAD, Virtex, XACT, XC2064, XC3090, XC4005, and XC5210 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

ACE Controller, ACE Flash, A.K.A. Speed, Alliance Series, AllianceCORE, Bencher, ChipScope, Configurable Logic Cell, CORE Generator, CoreLINX, Dual Block, EZTag, Fast CLK, Fast CONNECT, Fast FLASH, FastMap, Fast Zero Power, Foundation, Gigabit Speeds...and Beyond!, HardWire, HDL Bencher, IRL, J Drive, JBits, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, NanoBlaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, Real-PCI, RocketIO, SelectIO, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, SMARTswitch, System ACE, Testbench In A Minute, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex-II Pro, Virtex-II EasyPath, Wave Table, WebFITTER, WebPACK, WebPOWERED, XABEL, XACT-Floorplanner, XACT-Performance, XACTstep Advanced, XACTstep Foundry, XAM, XAPP, X-BLOX +, XC designated products, XChecker, XDM, XEPLD, Xilinx Foundation Series, Xilinx XDTV, Xinfo, XSI, XtremeDSP and ZERO+ are trademarks of Xilinx, Inc.

The Programmable Logic Company is a service mark of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx provides any design, code, or information shown or described herein "as is." By providing the design, code, or information as one possible implementation of a feature, application, or standard, Xilinx makes no representation that such implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of any such implementation, including but not limited to any warranties or representations that the implementation is free from claims of infringement, as well as any implied warranties of merchantability or fitness for a particular purpose. Xilinx, Inc. devices and products are protected under U.S. Patents. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

The contents of this manual are owned and copyrighted by Xilinx. Copyright 1994-2003 Xilinx, Inc. All Rights Reserved. Except as stated herein, none of the material may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of any material contained in this manual may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

ChipScope ILA Tools Tutorial

UG044 / PN 0401957 (v4.2.2) July 24, 2003

The following table shows the revision history for this document.

	Version	Revision
07/12/01	1.0	Initial Release.
08/10/01	1.1	Added Virtex™-II information; Updated screen shots; Replaced text in “Modifying the UCF File” section; Added “Running Implementation after Using the Core Generator” section.
08/20/01	1.2	Miscellaneous edits for clarity.
08/30/01	3.0	Added software version to title page. Updated document version number from 1.2 to 3.0 to coincide with software version. Miscellaneous text edits for clarity.
10/19/01	4.0	Revised document to be compatible with ChipScope software version 4.1i.
03/19/02	4.2.1	Revised document to be compatible with ChipScope software version 4.2i.
07/24/03	4.2.2	Administrative change only: Document number changed from UG004 to UG044.

Contents

ChipScope ILA Tools Tutorial

Overview	1
ChipScope ILA Tools Description.....	1
Using the ChipScope Core Generator.....	1
Generating an ICON Core.....	2
Generating an ILA Core	3
Using Cores in a Verilog Design Flow	5
Instantiating the ICON Module	5
Instantiating the ILA Module	6
Using Cores in a VHDL Design Flow	7
Instantiating the ICON Component	8
Instantiating the ILA Component	9
Synthesizing the Design	10
Modifying the UCF File	10
Using the ChipScope Core Inserter.....	10
Specifying ILA Options	13
Implementing the Design.....	17
Running Implementation after Using the Core Generator	17
Running Implementation after Using the Core Inserter.....	18
Setting Up the Board-Under-Test	20
Using the ChipScope Analyzer.....	20
Opening a Cable Connection	20
Setting Up the Boundary Scan Chain	20
Configuring the Target Device.....	21
Setting Up the Trigger	21
Running and Viewing the Waveform	22
Using the ILA Tool in the 4.2i FPGA Editor	23

ChipScope ILA Tools Tutorial

Overview

The ChipScope™ ILA Tools Tutorial includes instructions for synthesizing a design and setting up a board-under-test. It also includes steps to create and add one or more Integrated Logic Analyzer (ILA) cores and a single Integrated Controller (ICON) core to a design, using both the core generator flow and the core inserter flow in VHDL and in Verilog. A brief introduction to the ChipScope Analyzer software is included, as well as instructions on how to modify ILA input connections using the Xilinx FPGA Editor.

ChipScope ILA Tools Description

Table 1-1 lists the ChipScope ILA Tools and their description.

Table 1-1: ChipScope ILA Tools Description

Tool Name	Description
ChipScope Core Generator	Provides netlists and instantiation templates for the Integrated Controller (ICON) core and Integrated Logic Analyzer (ILA) cores.
ChipScope Core Inserter	Automatically inserts the ICON core and one or more ILA cores into the user's synthesized design.
ChipScope Analyzer	Provides capability for trigger setup and trace display for the ILA core(s) in the user's instrumented design. Also provides JTAG Boundary Scan configuration and communication capability.

Using the ChipScope Core Generator

The ChipScope Core Generator allows you to generate EDIF files and example code to instantiate ChipScope ICON and ILA cores into your design. Alternately, the ICON and ILA cores can be inserted into your design netlist using the ChipScope Core Inserter. Refer to [Using the ChipScope Core Inserter, page 10](#) for details. To start the ChipScope Core Generator, go to the Windows main menu bar and select **Start** → **Programs** → **ChipScope** → **ChipScope Core Generator**.

Generating an ICON Core

After starting the ChipScope Core Generator, choose to generate an Integrated Controller (ICON) core, then click **Next**. The ICON core example in this tutorial uses the following parameters:

- Device Settings: Virtex™ / Virtex-E / Spartan™-II/Spartan-IIE
- Number Of Control Ports: 1
- Enable External Trigger Input: No
- Enable External Trigger Output: No
- Disable JTAG Clock BUFG Insertion: No
- Include Boundary Scan Ports: No

Figure 1 shows the options chosen in the *ChipScope Core Generator* dialog box. **Browse** to select the file destination, then once the options have been chosen, click **Next**.

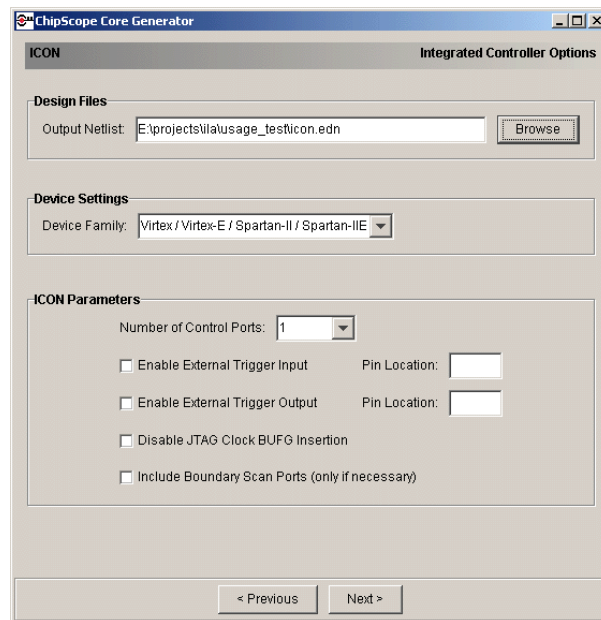


Figure 1: Choosing ICON Core Generation Options

Figure 2 displays the choices for HDL sample files, which include instantiation templates and correct attributes. Both the language and synthesis tool are chosen, or you can choose not to generate any HDL files at all by deselecting **Generate Example Files**.

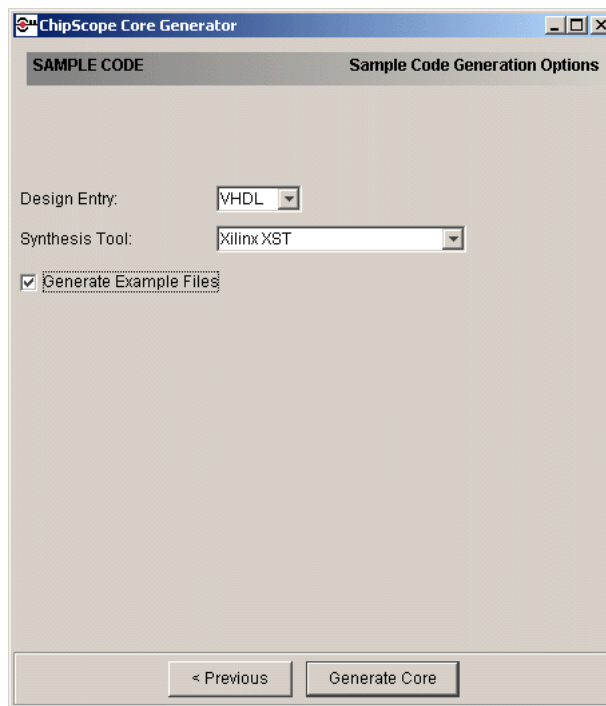


Figure 2: Choosing HDL Template Options

After choosing the appropriate options, click **Generate Core**. A message box opens, informing you when the core generation is complete. To move on to the next section, select **Start Over**.

Generating an ILA Core

This time, select the **ILA** radio button and click **Next**. The ILA core in this tutorial uses the following parameters:

- ILA Component Name Type: Long Name
- Device Settings: Virtex / Virtex-E / Spartan-II / Spartan-IIE
- Clock Settings: Rising Edge Of Clock
- Trigger Same As Data: No
- Trigger Width: 8
- Data Depth: 256
- Data Width: 16
- Extended Matching: Yes
- Number Of Match Units: 2

Figure 3 shows the options chosen in the *ChipScope Core Generator* dialog box. **Browse** to select the file destination, then once the options have been chosen, click **Next**.

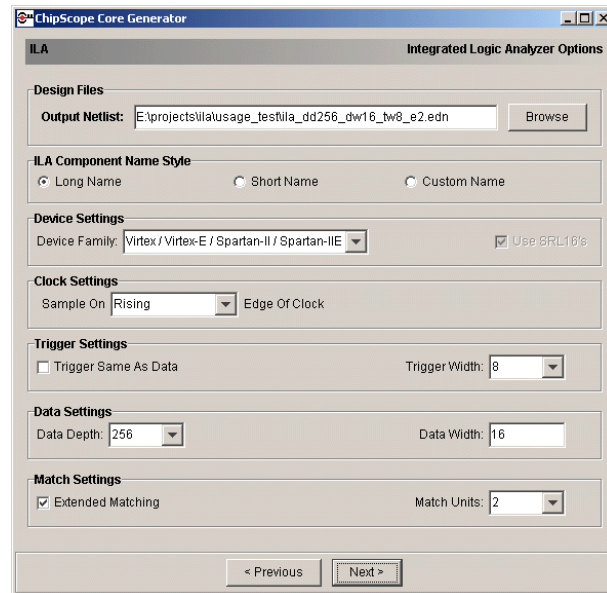


Figure 3: Choosing ILA Core Generation Options

After choosing the appropriate options, click **Generate Core**. A message box opens, informing you when the core generation is complete.

Using Cores in a Verilog Design Flow

The following Verilog code illustrates how to add ICON and ILA core modules to an existing design. It consists of a 32-bit counter whose upper four bits attach to output pads.

Note: The design must be synthesized using a standard Verilog synthesis tool that can produce EDIF or XST netlists for use with the Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) implementation tools.

```

module top ( clk, cnt );
  input clk;
  output [3:0] cnt;
  reg [31:0] counter;

  always @(posedge clk)
    counter = counter + 1;

  assign cnt = counter[31:28];

endmodule

```

Instantiating the ICON Module

The ICON module provides communication between the JTAG interface and each ILA unit. The ICON module declaration, the module instance, and the signals that the module uses, must be instantiated into the design. The **control_width** constant defines the width of the control bus while the **control_bus** signal provides a communication path between the ICON module and a single ILA module.

This tutorial uses an ICON module with a single control port and no Boundary Scan ports.

Note: This example shows instantiations using XST. Please follow the template instructions for instantiation in designs processed by other synthesis tools, as compiler directives and port directions may be different for different tools.

To instantiate the ICON module, use the following code (the ICON module-related Verilog statements are shown in bold):

```

module top ( clk, cnt );
  input clk;
  output [3:0] cnt;
  reg [31:0] counter;
  wire [41:0] control_bus;

  always @(posedge clk)
    counter = counter + 1;

  assign cnt = counter[31:28];

  icon i_icon
  (
    .CONTROL0(control_bus)
  );

endmodule

module icon
  (
    CONTROL0
  );
  output [41:0] CONTROL0;
endmodule

```

Instantiating the ILA Module

The ILA module captures data samples from a design when the trigger signal satisfies certain trigger conditions. The ILA module declaration, the module instance, and the signals that the module uses, must be instantiated into the design. The **data_width** and **trig_width** constants define the width of the data and trigger buses, respectively. These signals connect the ILA module to the part of the original design that is under test.

This tutorial uses an ILA module with a data depth of 256, data width of 16, and a trigger width of 8.

Note: This example shows instantiations using XST. Please follow the template instructions for instantiation in designs processed by other synthesis tools, as compiler directives and port directions may be different for different tools.

To instantiate the ILA module, use the following code (the ILA module-related Verilog statements are shown in **bold**):

```

module top ( clk, cnt );
    input clk;
    output [3:0] cnt;
    reg [31:0] counter;
    wire [41:0] control_bus;
    wire [15:0] data;
    wire [7:0] trig;

    always @(posedge clk)
        counter = counter + 1;

    assign cnt = counter[31:28];

    icon i_icon
    (
        .CONTROL0(control_bus)
    );

    assign data = counter[15:0];
    assign trig = counter[7:0];

    ila_dd256_dw16_tw8_e2 i_ila_dd256_dw16_tw8_e2
    (
        .CONTROL(control_bus),
        .CLK(clk),
        .DATA(data),
        .TRIG(trig)
    );

endmodule

module icon
(
    CONTROL0
);
    output [41:0] CONTROL0;
endmodule

module ila_dd256_dw16_tw8_e2
(
    CONTROL,
    CLK,
    DATA,
    TRIG
);

```

```
input [41:0] CONTROL;  
input CLK;  
input [15:0] DATA;  
input [7:0] TRIG;  
endmodule
```

Using Cores in a VHDL Design Flow

The following VHDL code illustrates how to add ICON and ILA core components to an existing design. It consists of a 32-bit counter whose upper four bits are attached to pads.

Note: The design must be synthesized using a standard VHDL synthesis tool that can produce EDIF or XST netlists for use with Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) implementation tools.

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_unsigned.all;  
  
entity top is  
  port  
  (  
    clk : in std_logic := '0';  
    cnt : out std_logic_vector(31 downto 0)  
  );  
end top;  
  
architecture behave of top is  
  signal counter : std_logic_vector(31 downto 0)  
    := (others => '0');  
begin  
  process(clk)  
  begin  
    if ( clk'event and clk = '1' ) then  
      counter <= counter + 1;  
    end if;  
  end process;  
  cnt <= counter(31 downto 28);  
end behave;
```

Instantiating the ICON Component

The ICON component provides communication between the JTAG interface and each ILA unit. The ICON component declaration, the component instance, and the signals that the component uses, must be instantiated into the design. The **control_width** constant defines the width of the control bus while the **control_bus** signal provides a communication path between the ICON component and a single ILA component.

This tutorial uses an ICON component with a single control port and no Boundary Scan ports.

Note: This example shows instantiations using XST. Please follow the template instructions for instantiation in designs processed by other synthesis tools, as compiler directives and port directions may vary for different tools.

To instantiate the ICON component, use the following code (the ICON module-related Verilog statements are shown in **bold**):

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity top is
  port
  (
    clk : in std_logic := '0';
    cnt : out std_logic_vector(31 downto 0)
  );
end top;

architecture behave of top is
  signal counter : std_logic_vector(31 downto 0)
    := (others => '0');
  signal control_bus : std_logic_vector(41 downto 0);
  component icon
  port
  (
    CONTROL0      : out std_logic_vector(41 downto 0)
  );
  end component;

begin
  process (clk)
  begin
    if ( clk'event and clk = '1' ) then
      counter <= counter + 1;
    end if;
  end process;

  cnt <= counter(31 downto 28);
  i_icon : icon
  port map
  (
    CONTROL0    => control_bus
  );
end behave;

```

Instantiating the ILA Component

The ILA component captures data samples from a design when the trigger signal satisfies certain trigger conditions. The ILA component declaration, the component instance, and the signals that the component uses, must be instantiated into the design. The **data_width** and **trig_width** constants define the width of the data and trigger buses, respectively. These signals connect the ILA component to the part of the original design that is under test.

This tutorial uses an ILA component with a data depth of 256, data width of 16, and a trigger width of 8. Also, the data and trigger signals are separate from one another.

Note: This example shows instantiations using XST. Please follow the template instructions for instantiation in designs processed by other synthesis tools, as compiler directives and port directions may be different for different tools.

To instantiate the ILA component, use the following code (the ILA module-related Verilog statements are shown in **bold**):

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity top is
  port
  (
    clk : in std_logic := '0';
    cnt : out std_logic_vector(31 downto 0)
  );
end top;

architecture behave of top is
  signal counter : std_logic_vector(31 downto 0)
    := (others => '0');
  signal control_bus : std_logic_vector(41 downto 0);
  component icon
    port
    (
      CONTROL0 : out std_logic_vector(41 downto 0)
    );
  end component;
  signal data : std_logic_vector(15 downto 0);
  signal trig : std_logic_vector(7 downto 0);
  component ila_dd256_dw16_tw8_e2
    port
    (
      CONTROL : in std_logic_vector(41 downto 0);
      CLK : in std_logic;
      DATA : in std_logic_vector(15 downto 0);
      TRIG : in std_logic_vector(7 downto 0)
    );
  end component;

begin
  process (clk)
  begin
    if ( clk'event and clk = '1' ) then
      counter <= counter + 1;
    end if;
  end process;

  cnt <= counter(31 downto 28);
  i_icon : icon
  port map

```

```

    (
        CONTROL0    => control_bus
    );
    data <= counter(15 downto 0);
    trig <= counter(7 downto 0);
    i_ila_dd256_dw16_tw8_e2 : ila_dd256_dw16_tw8_e2
        port map
        (
            CONTROL => control_bus,
            CLK     => clk,
            DATA   => data,
            TRIG    => trig
        );
end behave;

```

Synthesizing the Design

Once you have instantiated the ICON and ILA components into your design, use an HDL synthesis tool to compile the design to an EDIF or XST netlist file. The behavior of the ICON and ILA cores is not provided to the synthesis tool in the form of HDL files. Instead, the ICON and ILA cores act as “black boxes” whose behavior is provided during the implementation process in the form of EDIF netlists.

Modifying the UCF File

All of the constraints needed by the ICON and ILA cores during implementation are now included in NCF files that are automatically generated along with the EDIF netlists. You no longer need to modify the design UCF file to include these constraints. If you have included any ICON and/or ILA core constraints in the design UCF file, you will need to remove them to avoid any errors during the implementation process.

Note: If you move the ICON and/or ILA core EDIF netlist files from the directory in which they were generated, make sure you also move the corresponding NCF files to the same location. If you fail to do this, you might encounter erratic or erroneous behavior during implementation and/or run-time.

Using the ChipScope Core Inserter

Instead of using the ChipScope Core Generator to generate cores that must then be manually instantiated and added to the design, you can use the ChipScope Core Inserter (after synthesis) to embed the cores directly into the EDIF or XST netlist at once, allowing implementation to proceed immediately. To start the ChipScope Core Inserter, go to the Windows main menu bar and select **Start** → **Programs** → **ChipScope** → **ChipScope Core Inserter**.

The Core Inserter requires an EDIF or XST netlist, so the following design must be synthesized before core insertion can proceed. An EDIF netlist can have a *.edf or *.edn file extension and a XST netlist has the *.ngc file extension. In this tutorial, we will assume the following design has been synthesized to the XST netlist called **base.ngc** and translated into the netlist called **base.ngd** using the Xilinx ISE Foundation 4.2i Project Navigator tool:

```

module top ( clk, cnt );
    input clk;
    output [3:0] cnt;
    reg [31:0] counter;

    always @(posedge clk)
        counter = counter + 1;

```

```

assign cnt = counter[31:28];

endmodule

```

The first view of the Core Inserter is where you specify the input and output files (Figure 4).

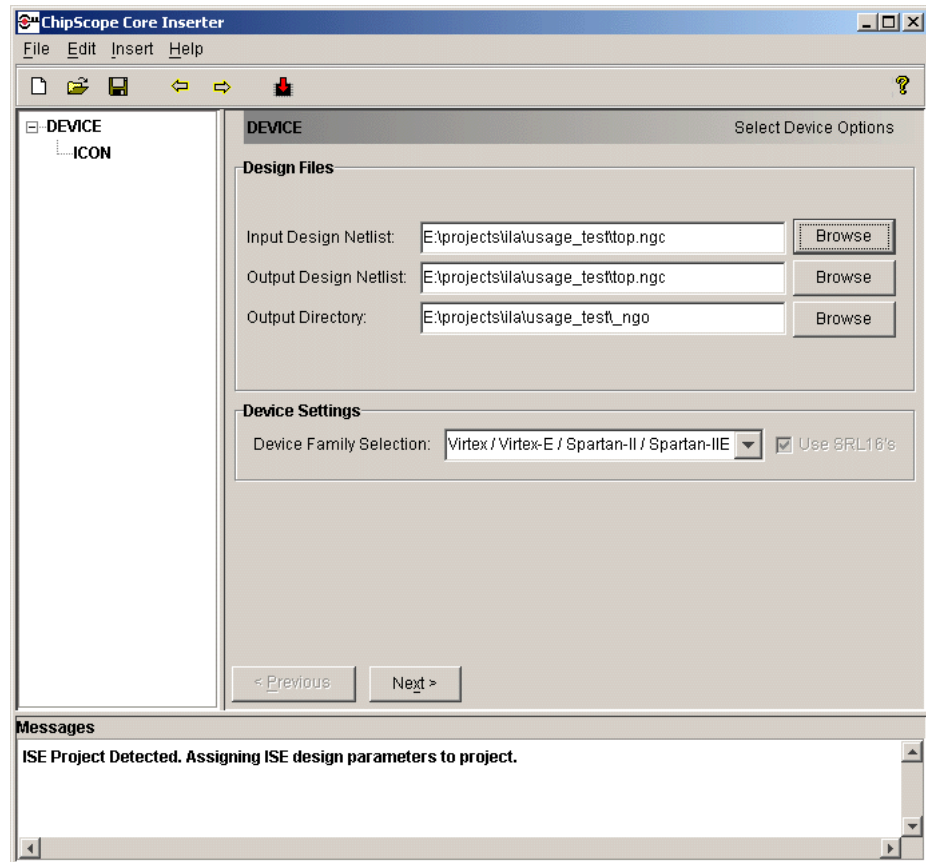


Figure 4: Specifying Core Insertion Files

First specify the Input Design Netlist (EDIF or XST) file. This is the synthesized design. Once this netlist is specified, the Core Inserter fills in all the other fields automatically. The default Device Family Selection and Use SRL16's selections are fine as is for this tutorial. Click **Next** to continue.

Figure 5 shows the available ICON options. The default is fine for this tutorial. Click **Next** to continue.

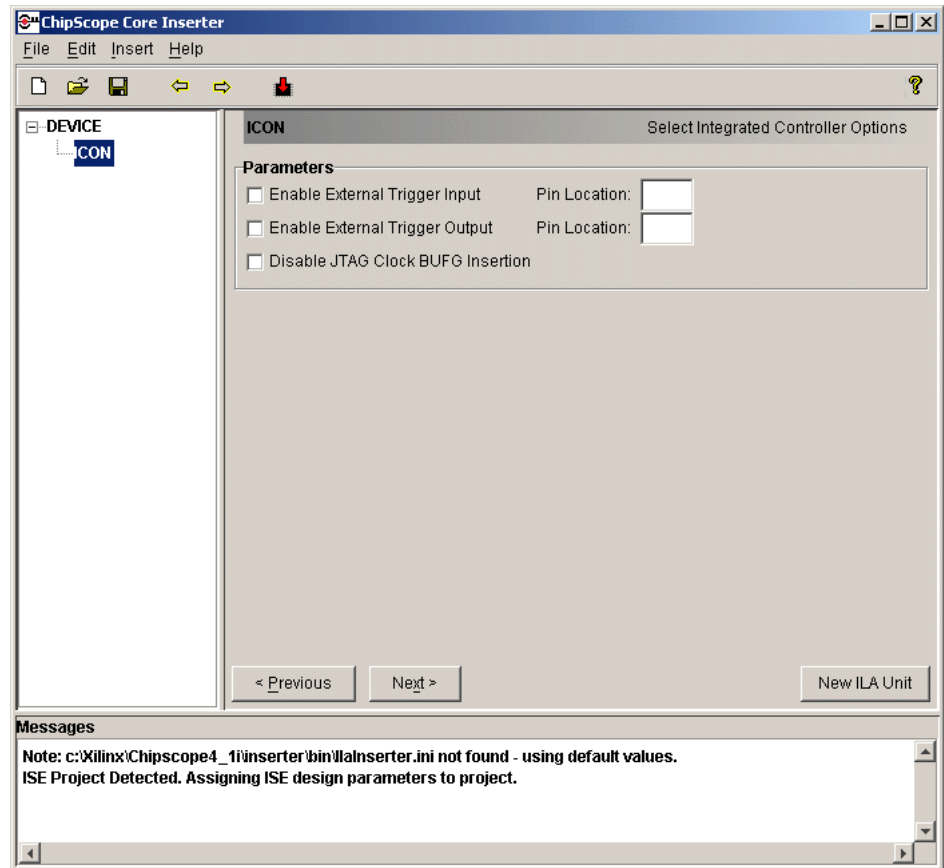


Figure 5: Specifying ICON Options

Specifying ILA Options

Figure 6 shows the available ILA options. The top half of the screen contains the parameters; the bottom half contains the *Net Connections* panel where the connectivity is specified. Each port type (Clock, Trigger, Data) has a plus (+) sign next to it.

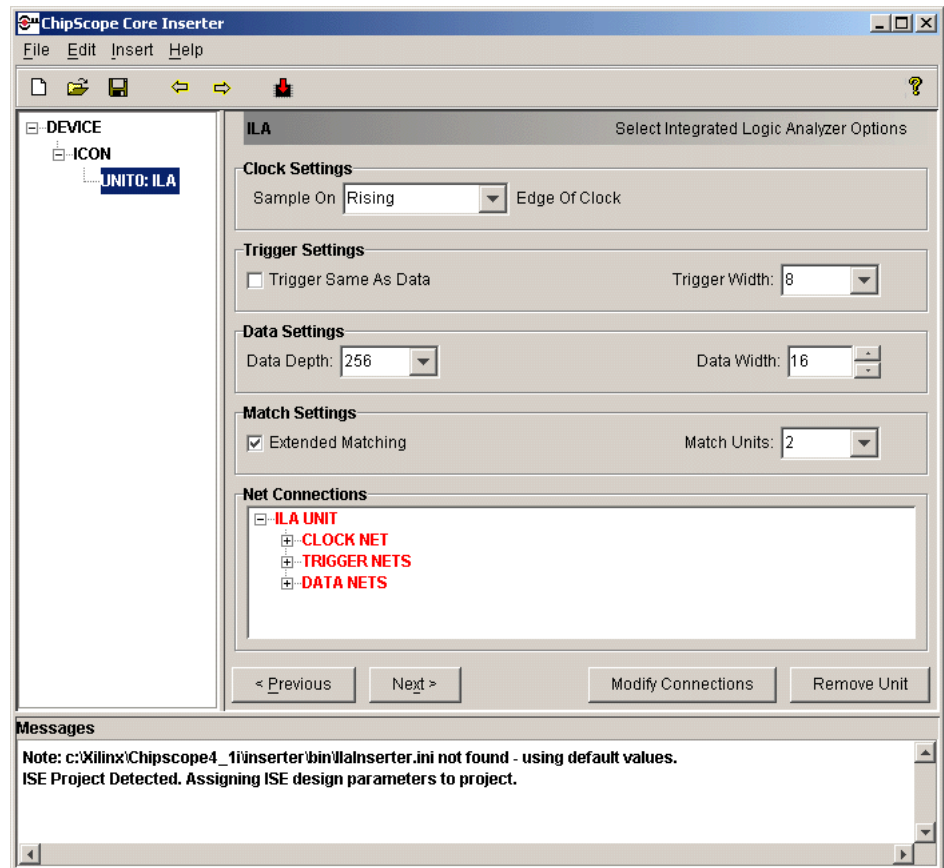


Figure 6: Specifying ILA Options

Click the plus (+) next to the clock to see that no net is currently selected (Figure 7). When the **Clock Net** selection is colored red, it means that no net is currently connected.

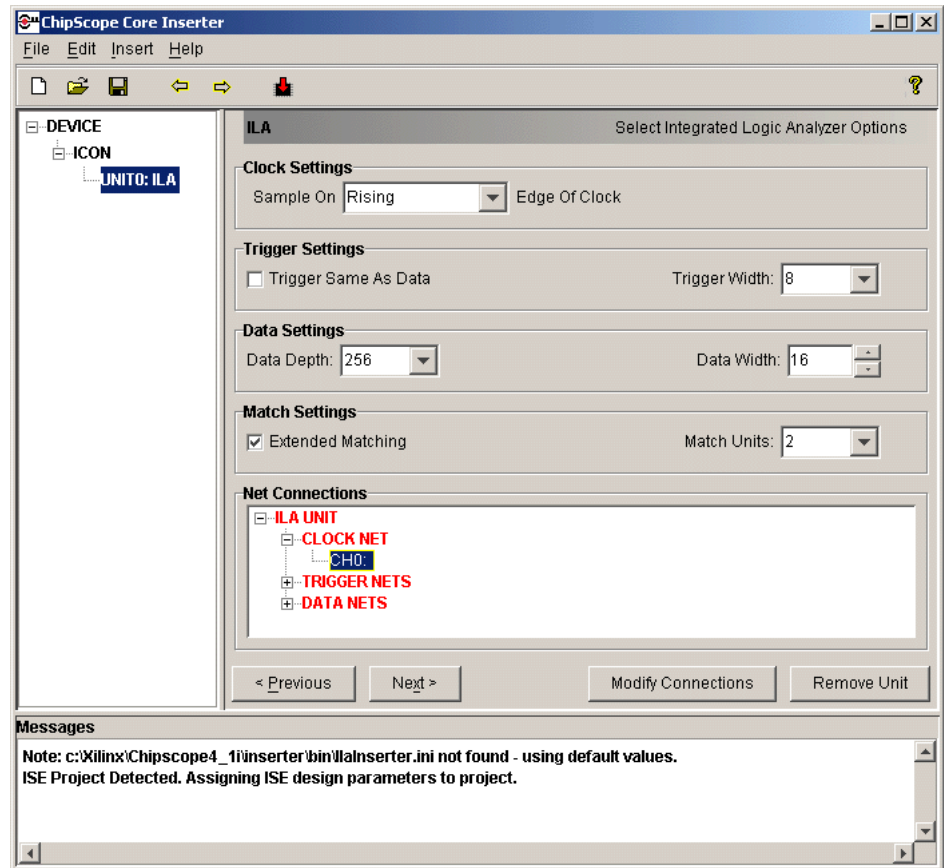


Figure 7: Choosing a Clock Net

Double-click the **CH0**: selection under **Clock Net**, or highlight **CH0**: and click **Modify Connections**. The *Select Net* dialog box appears (Figure 8).

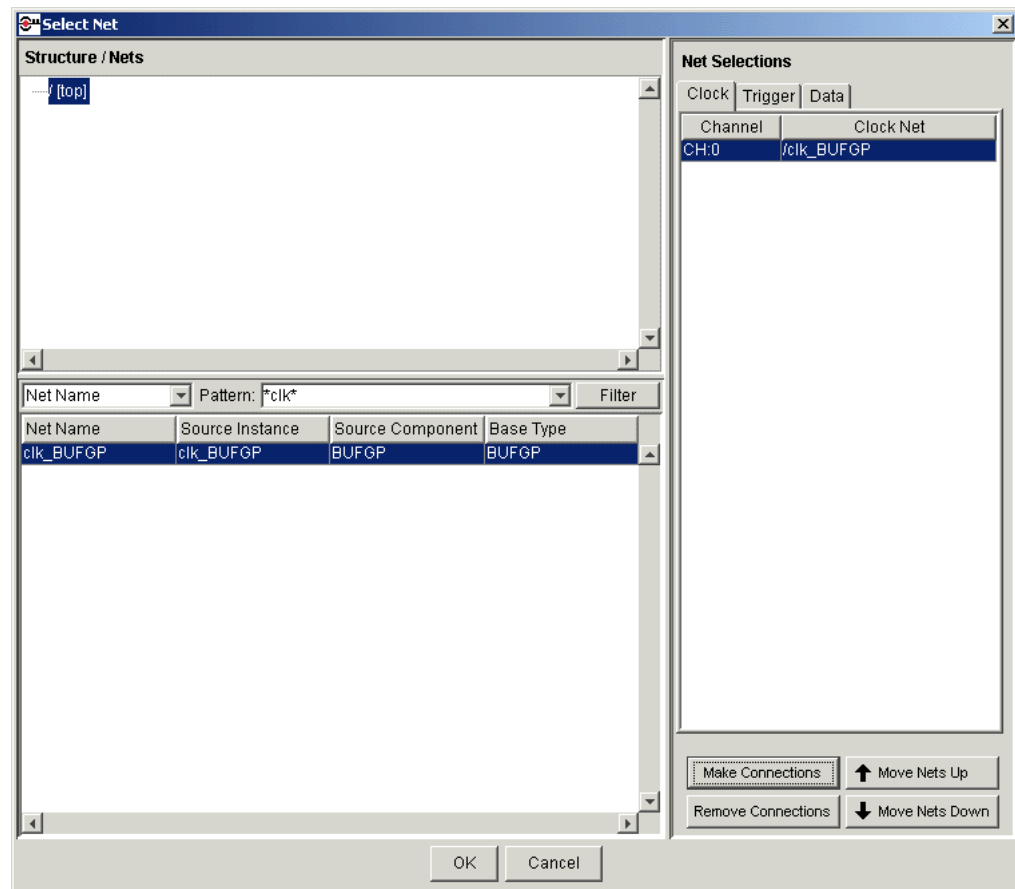


Figure 8: Choosing a Net in the Select Net Dialog Box

The hierarchy of the design (none in this case) can be traversed on the upper left panel of the *Select Net* dialog box. The nets available at the selected level of hierarchy are listed in the table at the lower left. The ILA input channels that can be connected to nets are located in the upper right *Net Selections* panel.

Net names can be easily located in a given level of hierarchy by searching for patterns in the net names, instance names of net driver source components, types of net driver source components, or the type of base components (primitives or black boxes) that are driving the nets. Wildcards (*,?) can be used in search filters for keywords, net names, parts of net names, etc.:

- “?” is used to represent one character
- “*” is used to represent multiple characters

Select the “Net Name” category in the pull-down list, type the keyword “*clk*”, including wildcards, into the *Pattern* text box and click **Filter** to find the net names that include the keyword.

For the clock net, select the net named `clk_BUFGP`, which is the clock net after the global buffer (BUFG or BUFGP). To assign the `clk_BUFGP` net to the clock input of the ILA core, select the **Clock** tab in the *Net Selections* tabbed panel and select the **CH:0** row in the table. Now click on the **Make Connections** button on the lower right of the *Select Net* dialog

box. Click on the **OK** button at the bottom of the *Select Net* dialog box to view the clock net connection you just made. See [Figure 9](#).

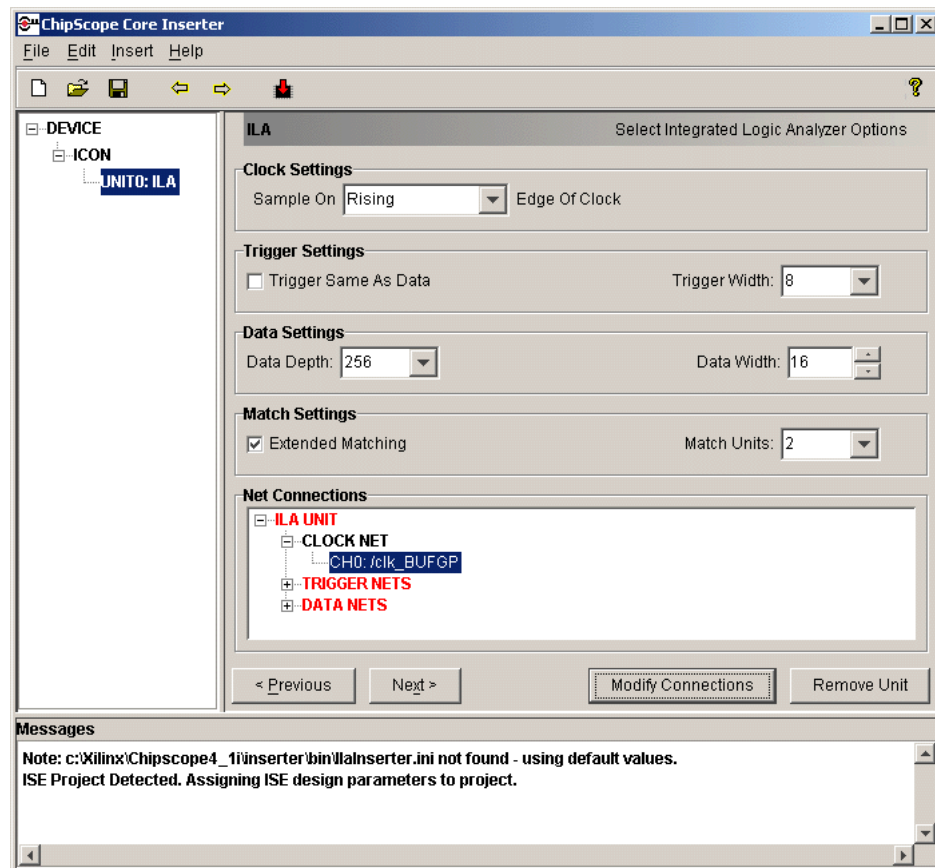


Figure 9: Core Inserter Clock Net Chosen

Repeat this procedure to select `counter[7:0]` for the trigger nets, and `counter[15:0]` for the data nets. The connecting channels for data and trigger may be different for different vendors (due to bus expansion, etc.) so the names will be `counter_15...`

Note: These bus-wide assignments can be done all at one time by selecting multiple source nets and ILA core channel destinations before clicking on the **Make Connections** button in the *Select Net* dialog box.

When finished, click **Next**. At this time, a dialog box appears with the question “Proceed with Core Insertion?” For the purposes of this tutorial, only one core is needed, so click **Yes**. The core insertion progress then is displayed in the *Messages* panel at the bottom of the window. A “Core Generation Complete” message appears in the *Messages* panel when the process is complete.

Note: To add another ILA core to the design, click **No**, and select **Edit** → **New ILA Unit**. A new unit (UNIT1:ILA) will appear in the tree on the left, and the options and nets can be chosen as they were for Unit 0. Up to 15 ILA cores can be parameterized this way.

At this point, the ICON, ILA, and top-level design `*.ngo` files have been created in the directory specified in the ChipScope Core Inserter “Device” window, [Figure 4, page 11](#). Make sure that you use these `*.ngo` files during the design implementation stage that is described in the following section. Failure to do so could result in errors during the implementation process, or later, while running the ChipScope Analyzer.

Implementing the Design

Running Implementation after Using the Core Generator

After using the Core Generator to generate the EDIF netlist, then use the Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) tools to implement your design. Use the following script to produce device BIT files that are compatible with JTAG configuration.

For JTAG configuration:

```
ngdbuild -sd <netlist path> -p <part type> top.ngc
map top
par -ol 5 -w top top
bitgen -w -g UserID:<user ID> -g StartupClk:JtagClk top top_jtag
```

The **ngdbuild** **"-sd"** option specifies the location of the ICON and ILA core EDIF netlist (***.edf** or ***.edn**) files or XST netlist (***.ngc**) files and NCF constraint files used by the top design. Note that you should modify the *<netlist path>*, *<part type>*, and *<user ID>* parts of the script.

Running Implementation after Using the Core Inserter

After inserting ChipScope cores into the design, the design must be implemented using the Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) tools.

Note: The ChipScope Core Inserter 4.2i will only work with Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) implementation tools. It will not work with any Xilinx Alliance Series™ 4.1i (or earlier), Xilinx Foundation Series™ 4.1i (or earlier), or Xilinx WebPACK™ implementation tools.

Using Core Inserter 4.2i with Command Line Implementation

Since the Core Inserter generates a file of a different format than synthesis (an NGO file instead of an EDIF or XST netlist) the implementation flow is slightly different. If you are using the command line, use the NGO file produced by the Core Inserter instead of the EDIF or XST netlist.

For JTAG configuration:

```
ngdbuild -sd <netlist path> -p <part type> base_ila.ngo
map base_ila
par -ol 5 -w base_ila base_ila
bitgen -w -g UserID:<user ID> -g StartupClk:JTAGClk base_ila base_ila
```

Using Core Inserter 4.2i with Xilinx ISE Foundation 4.2i

If you are using Core Inserter 4.2i with Xilinx ISE Foundation 4.2i implementation tools, some flow manipulation is necessary to ensure the correct files are implemented. To insert ChipScope cores via the ChipScope Core Inserter into a design processed by Xilinx ISE Foundation 4.2i, follow these steps:

1. **Synthesize** and **Translate** the design.

After creating a project and adding the HDL or EDIF/XST source files, **Synthesize** (if required for the selected flow) and **Translate** the design. This can be done by simply double-clicking **Translate** under **Implement Design**.

2. Run the ChipScope Core Inserter.

Launch the Core Inserter from **Start** → **Programs** → **ChipScope 4.2i** → **ChipScope Core Inserter**.

Note: ChipScope Core Inserter version 4.1i (or earlier) cannot be used with Xilinx ISE Foundation 4.2i.

3. **Browse** for the top level design to fill in the Input Design Netlist. Verify the following parameters are set under the **Device** tab:

- For Exemplar, Synopsys, and Synplicity designs:

Input Design Netlist = <ProjectDir>\<design>.[edf|edn]

Output Design Netlist = <ProjectDir>_ngo\<design>.ngo

Output Directory = <ProjectDir>_ngo

- For XST designs:

Input Design Netlist = <ProjectDir>\<design>.ngc

Output Design Netlist = <ProjectDir>\<design>.ngc

Output Directory = <ProjectDir>_ngo

Browse for the input <design>.edf, <design>.edn, or <design>.ngc file. The Core Inserter will detect the presence of the ISE project when it finds the **_ngo** directory, and the two output fields will be filled in automatically. This can only be done after the **Translate** step has been run at least once.

4. Complete ICON and ILA core insertion in the ChipScope Core Inserter.

Define ChipScope Core parameters and connect all signals, then insert the cores. The Core Inserter places new NGO files for the top level, as well as for the ICON and ILA cores, in the **_ngo** directory. Save the project before exiting the Core Inserter.

5. Rerun the **Translate** step.

When you return to Xilinx ISE Foundation 4.2i, you should see that the green check mark by the **Translate** step has disappeared. Double-click **Translate**, or right-click **Translate** and select **Run**. The **RUN** command will start with the new NGO files.

Note: DO NOT select **Rerun All**, as this will rerun Synthesis and the first portion of **Translate**, which will undo the Core Insertion that was just performed.

6. Continue with **Implementation**.

Make sure to specify the ChipScope Core Inserter bitstream generation options (JTAG clock when required, enable Readback) when creating the bitstream.

You must follow all of these steps after making any modifications to the HDL source, or after deleting any of the implementation data. Simply running through the standard ISE flow will not insert the ChipScope cores into the design.

Setting Up the Board-Under-Test

Before running the ChipScope software, set up the system or board-under-test. This tutorial assumes that a MultiLINX™ download cable will be used to communicate between the host PC and the board-under-test. Follow these steps to ensure that ChipScope software functions properly:

1. Power down the system.
2. Connect either the USB or the RS-232 connector of the MultiLINX download cable to the host computer using the appropriate cable.
3. If using the RS-232 connection, make sure that a serial communications port is available (that is, not used by another resource) for the MultiLINX cable.
4. Connect the MultiLINX cable to the V_{CC} (2.5-5.0V) and GND header pins on the board-under-test.
5. Connect the MultiLINX cable to the JTAG header pins (TDI, TMS, TCK, and TDO) that lead to the chain containing the target device to be tested.
6. If necessary, disable any device from driving the TDI, TMS, and TCK pins that are now connected to the MultiLINX cable.
7. Turn on power to the board to check the MultiLINX LED.

You are now ready to start the ChipScope Core Analyzer program.

Using the ChipScope Analyzer

To start the ChipScope Analyzer, go to the Windows main menu bar and select **Start** → **Programs** → **ChipScope** → **ChipScope Analyzer**. When the ChipScope Analyzer starts, the *Select Project* dialog box opens.

After you select "Create a new Project" and click **OK**, another dialog box opens. **Browse** for an appropriate project location, create a project file named **tutorial.cpj**, then click **Open**. The ChipScope window is now called ChipScope Analyzer [tutorial].

Opening a Cable Connection

After you start the ChipScope Analyzer and create the **tutorial.cpj** project, connect to the MultiLINX cable or Parallel Cable III cable by clicking the **Open cable/Search JTAG chain** button in the toolbar. After a few moments, the status bar at the bottom of the main ChipScope window should report the connection status (that is, "Successfully opened Parallel Cable (port = LPT1)").

Setting Up the Boundary Scan Chain

When the cable connection opens, the Boundary Scan chain is automatically detected. A window appears indicating which Xilinx and non-Xilinx devices were detected, and in what order. If all the devices in the chain are detected as Xilinx, nothing more needs to be done. If non-Xilinx devices are in the chain, you must specify the IR (Instruction Register) length.

Configuring the Target Device

You can configure the target device after setting up the Boundary Scan chain. This tutorial assumes that the target device is programmed using JTAG configuration mode. To configure the device, select **Configure** → **JTAG Configuration**, or click the **Configure with last used settings** button. The *Configuration* dialog box opens.

Click **Select New File**, and the *Open Configuration File* dialog box opens. **Browse** for the BIT file that was created earlier in the tutorial for JTAG configuration. After selecting the file, click **Open**. Once the file opens, you can view the configuration progress in the status bar at the bottom of the main ChipScope window. When the configuration process finishes, the DONE status appears at the lower right corner of the window.

Setting Up the Trigger

The BIT file that was used to configure the target device contains a single ILA unit connected to the **CONTROL0** port of the ICON unit (hence the name ILA "Unit 0"). The ChipScope Analyzer automatically queries the device and populates the toolbar with the correct settings. The ILA parameters appears at the bottom of the window in the status bar. Remember that the design contains a counter whose lower 8 bits connect to the trigger word and whose lower 16 bits connect to the data word. To set up the tutorial, follow these steps:

1. Set the M0 match type to ">" using the pull-down menu.
2. Set the M0 match value to "00000001" by clicking the lowest order bit once.
3. Set the M1 match type to "<" using the pull-down menu.
4. Set the M1 match value to "00000011" by clicking the two lowest order bits once.
5. Set the M0 match length type to "Count" by clicking the **Count** button.
6. Set the M0 match length value to "1".
7. Set the M1 match length type to "Count" by clicking the **Count** button.
8. Set the M1 match length value to "1".
9. Set the Trigger Condition type to "boolean" using the pull-down menu.
10. Set the Trigger Condition equation to "M0 AND M1" by clicking the **Trigger Condition** equation buttons.
11. Set the Capture Type to "One Shot" by clicking the **One Shot** button.
12. Set the trigger Position to "100" by entering the value in the Position text box.
13. Disable the external trigger output by leaving the Ext. Out box unchecked.

The trigger is now set up to trigger only when the 8-bit trigger word is in the non-inclusive range of 1 to 3 (i.e., 2). When this condition is satisfied at least once, the entire sample buffer is stored with the trigger value occurring at clock cycle 100.

Running and Viewing the Waveform

After setting up the trigger, select **Run/Stop** → **Run** to arm it. The trigger status opens in the Sample Buffer status window. After the sample buffer status is **FULL**, the waveform appears in the main ChipScope Analyzer window. Notice that the trigger value of 2 (binary 00000010) occurs at clock cycle position 0 (**Figure 10**).

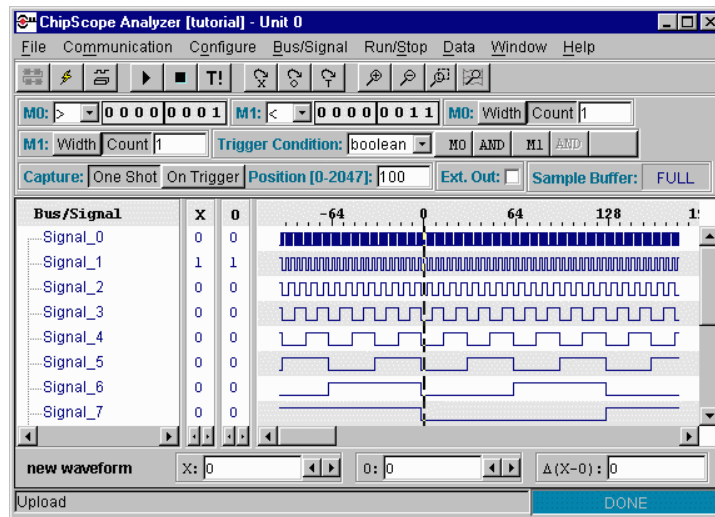


Figure 10: Sample Buffer Waveform Window

This completes the ChipScope ILA Tools tutorial. Now you can experiment with different trigger setups to become more familiar with the ChipScope Analyzer.

Using the ILA Tool in the 4.2i FPGA Editor

The Integrated Logic Analyzer (ILA) ILA command is accessed from the **Tools** menu of the Xilinx ISE Alliance 4.2i (or later) or Xilinx ISE Foundation 4.2i (or later) FPGA Editor. This section describes how to use the ILA command to view and change the nets connected to the capture units of the ILA core in your design. Use the following procedure to attach a different net to a capture unit in your design.

Note: For more information on the fields in the *ILA* dialog box, refer to the FPGA Editor online help.

1. Select the net in your design that you want to attach to the capture unit. Notice, the net's name appears in the **History** toolbar at the bottom of the FPGA Editor main window.
2. Select **Tools** → **ILA** to display the *ILA* dialog box (Figure 11).

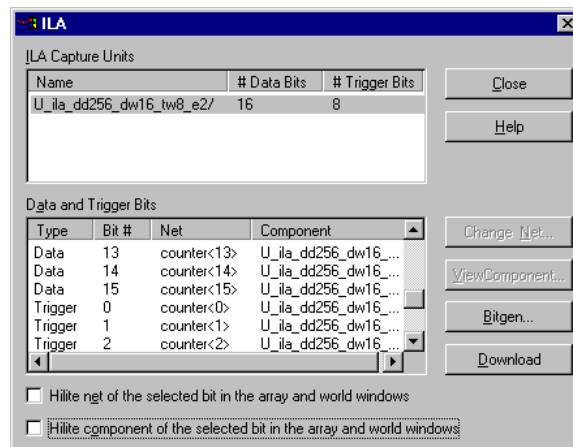


Figure 11: ILA Dialog Box

3. Select the applicable ILA capture unit from the list in the ILA Capture Unit pane. After you select the capture unit, the nets attached to its data and trigger bits are displayed in the #Data Bits and #Trigger Bits columns in the pane.
4. Select a **Data Bit** or **Trigger Bit** from the Data and Trigger Bits pane.
5. Select one or both of the check boxes at the bottom of the dialog box to highlight the net or component of the selected bit in the Array and World windows of the FPGA Editor tool.
6. Find a data or trigger bit that is close to your selected net by observing the selection in the Array or World window.
7. Click **Change Net** to display the *ILA Net* dialog box (Figure 12).

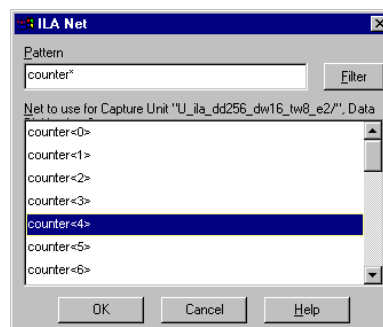


Figure 12: ILA Net Dialog Box

8. Enter the name of the net you want to connect to the capture unit in the **Pattern** field. You can use the **Filter** button or select from the list of available nets.
9. Click **OK** to unroute the net attached to the capture unit and attach your selected net.
10. To create a BIT file of your modified design, use one of the following scripts:
 - For JTAG mode configuration:

```
bitgen -w -g UserID:<user ID> -g StartupClk:JtagClk top top_jtag
```
11. Configure the target device with the modified BIT file using the ChipScope software tool.