

**Xilinx Spartan-6 LX16 Evaluation Board
PicoBlaze SHA-1
Authentication Design**

SPARTAN 

January, 2013

Table of Contents

Introduction.....	2
Overview	3
Reference Design Requirements	5
Software	5
Hardware	5
Hardware Design Block Diagrams	6
Supplied Files.....	8
Setting Up the S6LX16 Evaluation Board.....	10
PC Setup.....	12
Installing the UART Driver and Virtual COM Port	12
Installing a Serial Console on a Windows 7 Host.....	13
Running the Demo Files	14
SHA-1 Programming Test Demo.....	14
SHA-1 Authentication Example Demo	16
Implementing the Hardware Designs.....	17
SHA-1 Programming Test	17
Introduction.....	17
SHA-1 Authentication Example	23
Introduction.....	23
Troubleshooting Design Changes	30
Introduction.....	30
Reference Material.....	36
Revision History	37

Introduction

This document describes a basic PicoBlaze™ 8-bit embedded microcontroller design implemented and tested on the Xilinx Spartan-6 LX16 evaluation board. This example design utilizes the Maxim DS28E01PMOD (1-Wire EEPROM with SHA-1 Engine Plugin Module).

Two Xilinx ISE projects are provided as hardware and software examples for demonstrating SHA-1 challenge-and-response authentication. Both applications communicate with the Maxim PMOD using the 1-Wire protocol to implement the Secure Hash Algorithm (SHA-1).

SHA1_PROG_TEST – implements the PicoBlaze microcontroller and software application to perform the tasks necessary to program the Maxim DS28E01-100 EEPROM with the SHA-1 security key and test that the programming is performed correctly.

SHA1_AUTH_EXAMPLE – implements the PicoBlaze microcontroller and software application to perform the tasks necessary to authenticate the secret programmed in the Maxim DS28E01-100 EEPROM matches the SHA-1 secret stored in the FPGA design.

Overview

The Xilinx Virtex family of FPGA devices, as well as the larger Spartan-6 devices, support AES bitstream encryption as a means to protect FPGA IP. For the smaller Spartan-6 and Spartan-3 devices that do not support bitstream encryption, the challenge-and-response mechanism of the Secure Hash Algorithm (SHA-1) authentication protocol is a viable and cost-effective means of FPGA design copy protection for cost-sensitive applications. SHA-1 can also be used to secure external peripherals, which is not possible with AES bitstream encryption. Using the SHA-1 authentication scheme the FPGA can determine if it has been copied and if so, can disable its logic rendering the product useless or limited in function. This Identification Friend or Foe (IFF) method is applicable to all Xilinx FPGA devices. SHA-1 authentication of the FPGA is achieved by using a secure memory chip such as the Maxim DS28E01 1-Wire EEPROM with SHA-1 Engine. [Figure 1](#) describes a block diagram for this approach. The 1-Wire interface of the DS28E01 SHA-1 EEPROM makes it particularly well-suited for this application because it requires only a single FPGA pin for communication. The SHA-1 IFF method is effective for this level of security because:

- It is irreversible. It is computationally infeasible to determine the input corresponding to the SHA-1 result.
- It is collision-resistant. It is very impractical to find another input message that produces the same SHA-1 result.
- It has a high avalanche effect. Any change in input causes a significant change to the SHA-1 result.

At its core, the SHA-1 algorithm works by comparing a secret key programmed in the EEPROM with a secret key in the FPGA. However, since simply comparing the hash of a secret key would produce predictable and repeatable communications between the FPGA and the EEPROM, and thus the ability to snoop and clone the FPGA, extra security is further introduced into the hash scheme by adding a series of randomly generated challenge bytes and the unique serial number identifier of the EEPROM itself. Since the random challenge bytes and unique serial number defeat the repeatability of the communication stream, security is significantly improved. The SHA-1 authentication IFF method works by following these steps:

1. When power is applied, the FPGA fetches and loads its complete bitstream from its configuration memory.
2. The microcontroller in the FPGA becomes active and performs the authentication, which consists of the following steps:
 - a. Generate a series of random number bytes and send it as a challenge to the secure EEPROM.
 - b. Instruct the EEPROM device to compute a HASH based on its (pre-programmed) secret key, the random challenge bytes, and its unique ID.
 - c. Compute a HASH based on the same random challenge bytes, unique ID of the EEPROM device, and the FPGA's own secret key.
 - d. Compare the HASH computed by the EEPROM to its own computed HASH.

If the HASH from the EEPROM and FPGA match, the microcontroller identifies the circuit as a 'Friend', because it is able to validate the secret key in the EEPROM device. The FPGA transitions to normal operation by enabling the remainder of the FPGA design we'll call the 'secret sauce'. If the HASH do not match, however, the circuit must be a 'Foe'. In this case the FPGA keeps the 'secret sauce' disabled.

To prevent a contract manufacturer (CM) from overbuilding and thus cloning the product, the EEPROM should be pre-programmed with the secret key and only the required quantity of EEPROM devices needed for the scheduled build should be sent to the CM. This insures the CM only builds the number of units they are authorized to build. To maximize the security and integrity of the product manufacturing, the CM should not be allowed to have access to the secret key or to program the EEPROM.

This design employs the Xilinx PicoBlaze microcontroller to implement the SHA-1 authentication algorithm and provide communication with the Maxim DS28E01 EEPROM using the 1-Wire communication protocol. The PicoBlaze also uses a UART to provide a simple user interface over a standard RS232 serial port.

There are two Xilinx ISE projects to this design: One for programming the EEPROM with the SHA-1 secret key (SHA1_PROG_TEST) and another for performing the Identification Friend or Foe (IFF) test to authenticate the FPGA with the EEPROM (SHA1_AUTH_EXAMPLE). Both projects perform SHA-1 authentication on the EEPROM. The SHA1_PROG_TEST performs the authentication to verify that the programming of the EEPROM with the secret key was done correctly. The SHA1_AUTH_EXAMPLE project does not program the secret key and instead meant to describe a system that can be easily used to implement the IFF method to disable user logic until the EEPROM has been authenticated.

Reference Design Requirements

Software

The software requirements for this reference design are:

- Windows XP, Windows 7 Professional
www.xilinx.com/ise/ossupport/index.htm
The PicoBlaze software compiler supplied with this design is a Windows binary, and thus the Windows requirement.
- Xilinx ISE Design Suite 14.3, Logic Edition or WebPack

Hardware

The hardware setup for this reference design is:

- Computer with 500 MB RAM and 500 MB virtual memory (recommended)
www.xilinx.com/ise/products/memory.htm
- Xilinx Spartan-6 LX16 evaluation board
- USB-A to USB-mini B cable
- Maxim DS28E01PMOD (1-Wire EEPROM with SHA-1 Engine Plugin Module)
- Xilinx Platform Cable USB

Hardware Design Block Diagrams

The following figures show high-level block diagrams of the hardware designs. The designs require:

- PicoBlaze processor
- BRAM (1024 x 18-bit)
- USB UART port
- LEDs
- Maxim DS28E01 SHA-1 EEPROM PMOD
- Xilinx Platform Cable USB

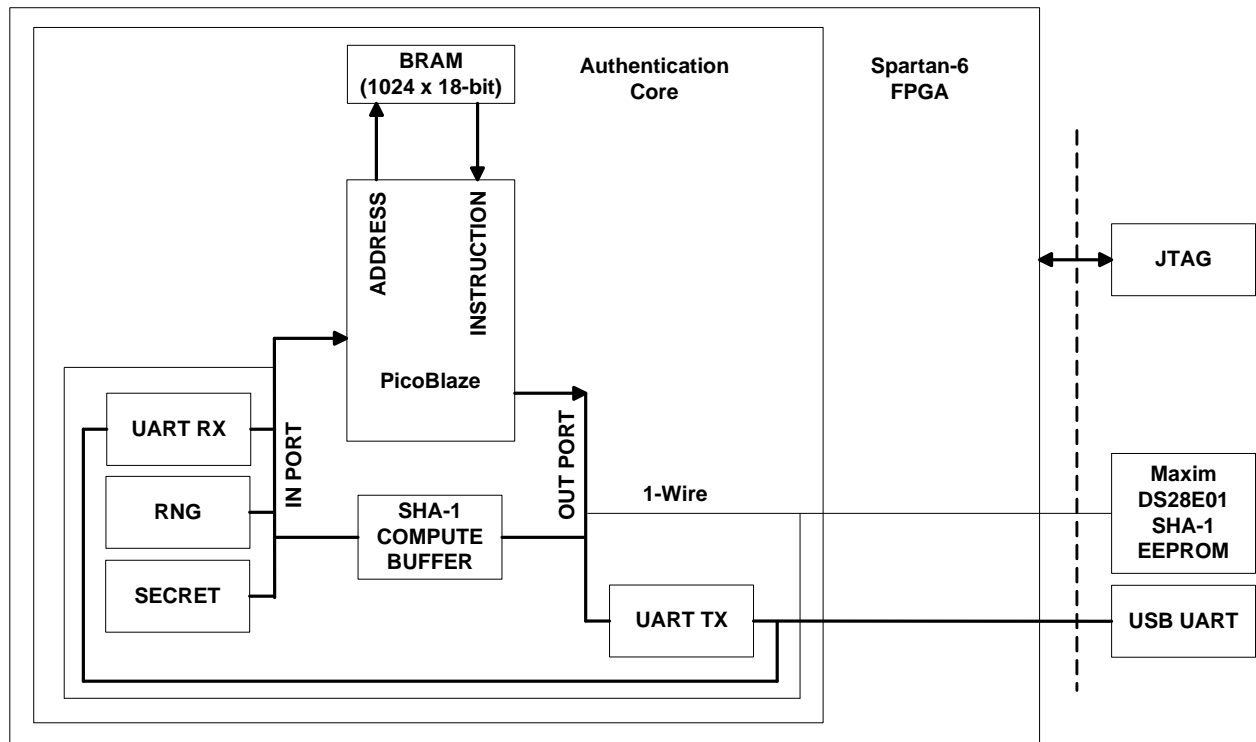


Figure 1 - SHA-1 Authentication Example Design Block Diagram

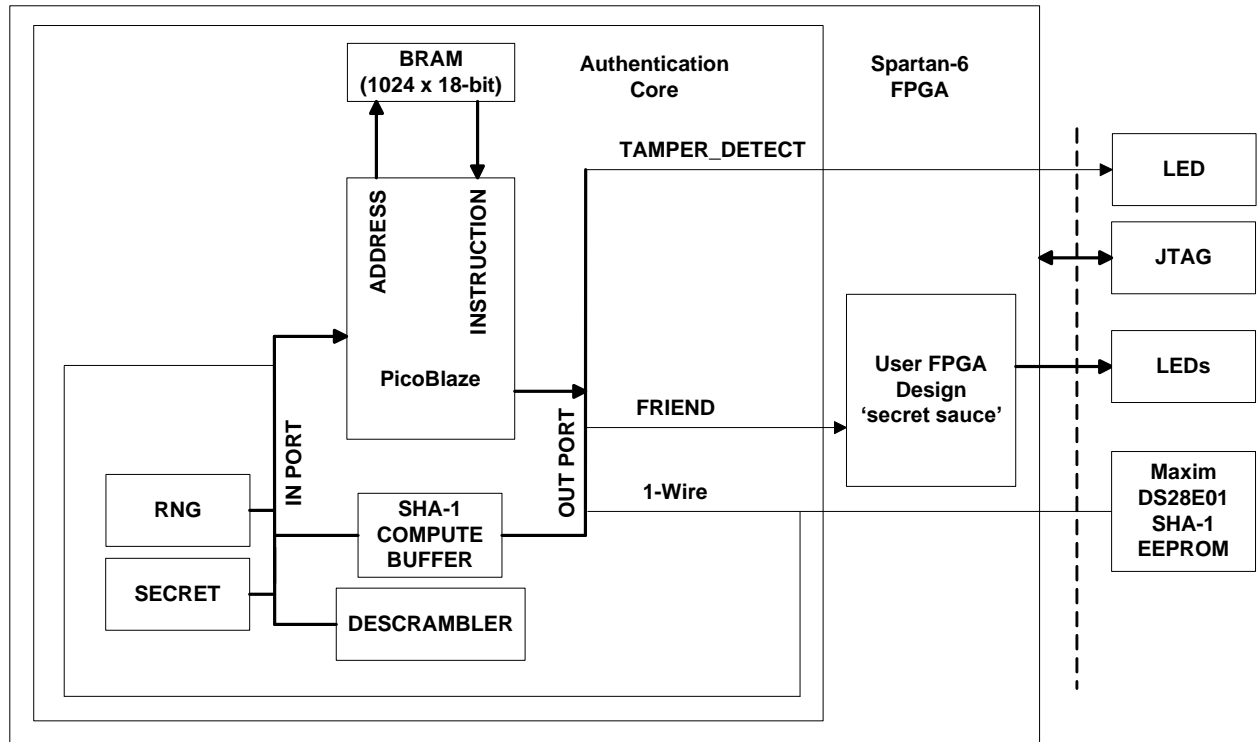


Figure 2 - SHA-1 Authentication Example Design Block Diagram

Supplied Files

The following directory structure is included with this reference design:

SHA1_PROG_TEST\: Contains files for the EEPROM programming example project:

ise\SHA1_PROG_TEST.xise: ISE project file.

KCPSM6_Release2_31March11: PicoBlaze ancillary files and documentation.

KCPSM6_Release2_31March11\ KCPSM6_User_Guide_30Sept10.pdf:

PicoBlaze user's guide.

KCPSM6_Release2_31March11\kcpsm6.exe: Windows executable to compile the PicoBlaze application.

KCPSM6_Release2_31March11\kcpsm6.vhd: PicoBlaze microcontroller source.

source\psm\assemble_pb.bat: Batch script to run the compilation of the PicoBlaze source code.

source\psm\sha1prog.psm: The PicoBlaze source code for the project.

source\psm\sha1prog.log: The PicoBlaze assembler log file.

source\psm\ROM_FORM.VHD: BRAM instantiation template used by the PicoBlaze assembler.

source\hdl\KCPSM3_UART\VHDL\uart_rx.vhd: PicoBlaze UART receive wrapper.

source\hdl\KCPSM3_UART\VHDL\uart_tx.vhd: PicoBlaze UART transmit wrapper.

source\hdl\KCPSM3_UART\VHDL\kcuart_rx.vhd: PicoBlaze UART receive macro.

source\hdl\KCPSM3_UART\VHDL\kcuart_tx.vhd: PicoBlaze UART transmit macro.

source\hdl\KCPSM3_UART\VHDL\bbfifo_16x8.vhd: PicoBlaze UART FIFO macro.

source\hdl\RND.vhd: VHDL source file for a serial random number generator

source\hdl\SHA1PROG.VHD: Compiled PicoBlaze application formatted to initialize the Spartan-6 Block RAM.

source\hdl\SHA1_PROG_TEST.vhd: Top level VHDL source file for SHA-1 programming test FPGA design.

source\ucf\SHA1_PROG_TEST.ucf: User Constraints File to define the timing and pin location constraints for this project.

ready_for_download\sha1_prog_test.bit: Project bitstream

ready_for_download\demo.bat: Batch script to download the bitstream.

ready_for_download\download.cmd: Xilinx iMPACT command file.

SHA1_AUTH_EXAMPLE: Contains files for the authentication example project:

ise\SHA1_AUTH_EXAMPLE.xise: ISE project file.

KCPSM6_Release2_31March11: PicoBlaze ancillary files and documentation.

KCPSM6_Release2_31March11\ KCPSM6_User_Guide_30Sept10.pdf:

PicoBlaze user's guide.

KCPSM6_Release2_31March11\kcpsm6.exe: Windows executable to compile the PicoBlaze application.

KCPSM6_Release2_31March11\kcpsm6.vhd: PicoBlaze microcontroller source.

scrambler\scrambler.exe: Command line application to scramble the compiled PicoBlaze application (sha1auth.psm) BRAM contents for added design security.

scrambler\scramble_pb.bat: Batch script to compile the PicoBlaze source code and scramble the BRAM contents.

source\psm\sha1auth.psm: The PicoBlaze source code for the project.

source\psm\sha1auth.log: The PicoBlaze assembler log file.

source\psm\ROM_FORM.VHD: BRAM instantiation template used by the PicoBlaze assembler.

source\hdl\AUTH_TEST.vhd: VHDL source file for SHA-1 authentication core FPGA design

source\hdl\AUTH_TEST_TB.vhd: VHDL source file for SHA-1 authentication core simulation testbench

source\hdl\include.vhd: Scrambled PicoBlaze code authentication keys.

source\hdl\sha1auth_scr.vhd: Compiled and scrambled PicoBlaze application formatted to initialize the Spartan-6 Block RAM.

source\hdl\check.vhd: PicoBlaze code descrambler.

source\hdl\ledflash.vhd: Simple VHDL file that maps the upper bits of a large counter to blink the LEDs on the FPGA board.

source\hdl\RND.vhd: VHDL source file for a serial random number generator

source\hdl\SHA1_AUTH_EXAMPLE.vhd: Top level VHDL source file.

source\ucf\SHA1_AUTH_EXAMPLE.ucf: User Constraints File to define the timing and pin location constraints for this project.

ready_for_download\sha1_auth_example.bit: Project bitstream

ready_for_download\demo.bat: Batch script to download the bitstream.

ready_for_download\download.cmd: Xilinx iMPACT command file.

S6LX16_PicoBlaze_SHA1_ise14_3.pdf: This document.

Setting Up the S6LX16 Evaluation Board

The board jumpers should be set as follows (refer to the table and diagram below):

Jumper	Function	Pin Setting	Description
JP1	On-board Flash Power	Pins 1-2	Apply power to the Flash
JP2	JTAG Chain Connection	Pins 1-2	FPGA only in JTAG chain
JP3	VS5 Setting	Pins 1-2	Set VS5 for 2.5V operation
JP4	FPGA PROG	None	Allow FPGA configuration
JP5	FPGA HSWAPEN	None	Do not enable pre-configuration pull-ups
JP6	Suspend Mode	Pins 1-2	PSoC controls Suspend
JP7	VS3 Power Mode	Pins 1-2	Disable power savings
JP8	MXP 5V Source	Pins 1-2	MXP 5V source tied to USB
JP15	USB charge current	None	Charge current set point at 500mA
JP16	Default power source	None	A/C used for charging if plugged in
JP17	Battery charger enable	None	Battery charger enabled
JP18	PSoC Expansion 5V Source	Pins 1-2	Set to 5V USB
JP19	PSoC Expansion Vadj Source	Pins 1-2	Set to 1.8V

Perform the following steps to set up the board for running the applications.

1. Connect the USB-JTAG cable with pod and ribbon connector between the JTAG connector on the board and a USB port on the PC
2. Plug the USB cable into the PC and port P1 on the S6LX16 board. LED D18 will light. LEDs D16 and D17 provide the status of the battery charger, so one or both of these may light.
3. Plug the Maxim DS28E01PMOD, component side up, into the upper row of pins on the J8 PMOD connector.
4. Connect the 12V power supply to site J2 (optional, not required)

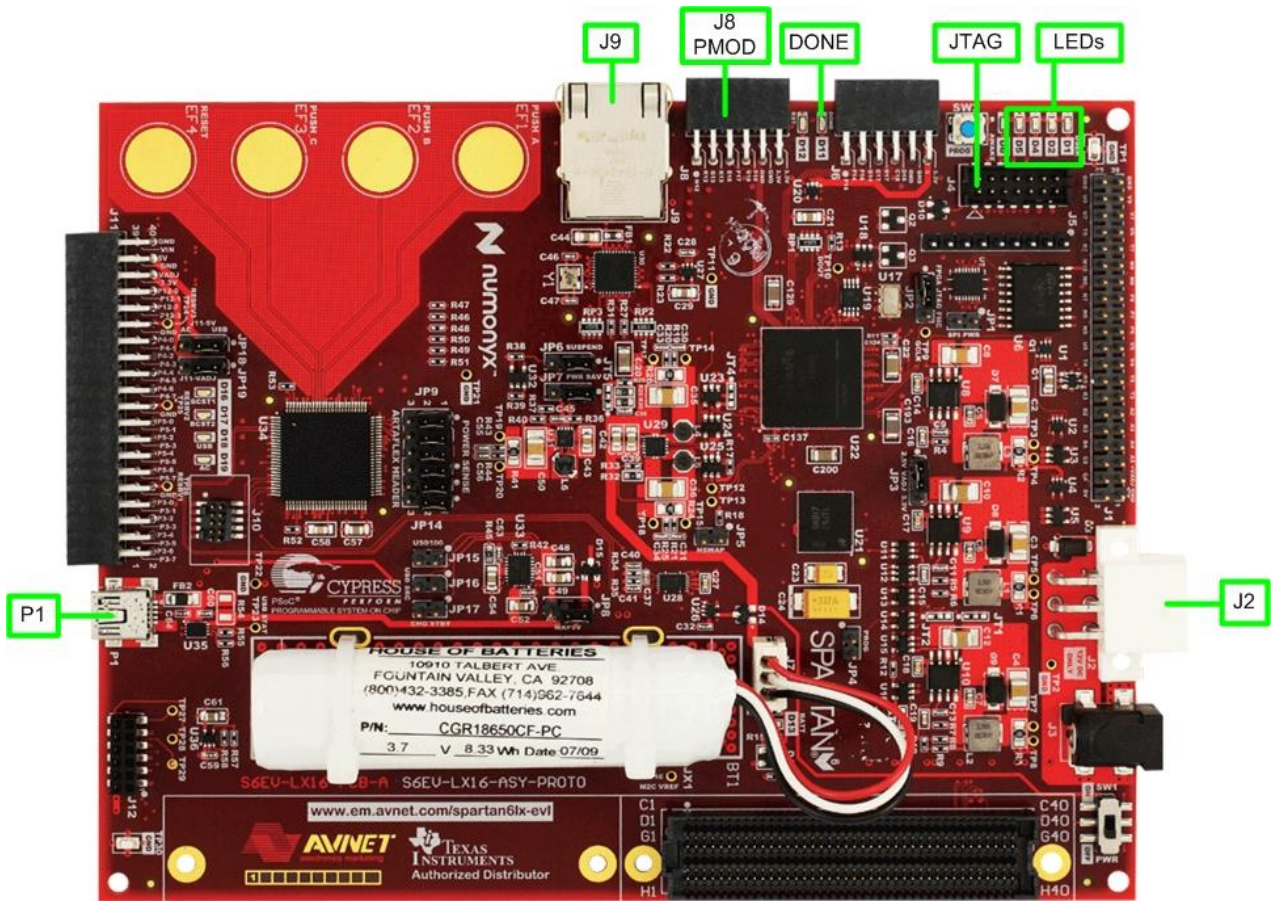



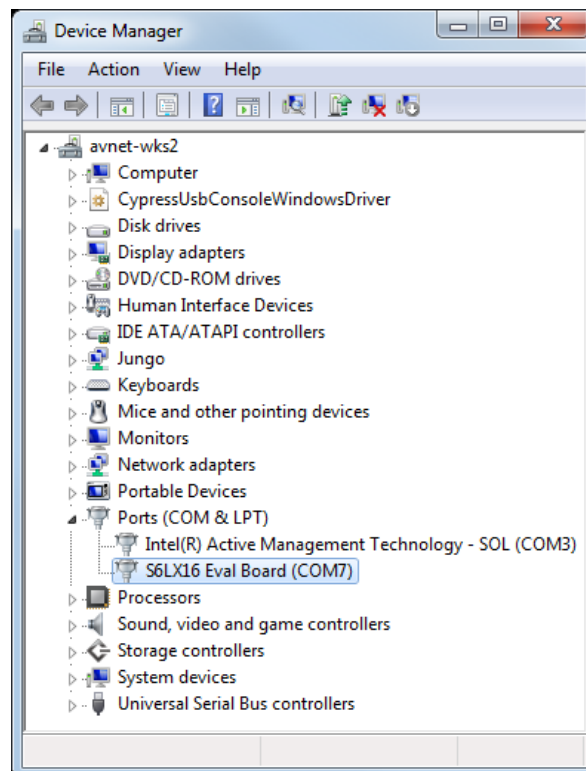
Figure 3 – Spartan-6 LX16 Evaluation Board

PC Setup

Installing the UART Driver and Virtual COM Port

If the LX16 Evaluation board has not been connected to the host PC before, it will be necessary to install the software driver for the virtual COM port.

1. Slide the power switch, SW1, from **CHARGE** to **ON**. This will power the board, as indicated by the D12 amber power-good LED illuminating.
2. The PC recognizes the board as new hardware and launches the *Found New Hardware Wizard*.
3. Follow the instructions in the *Avnet Programming Utility (AvProg) 4.05 User Guide* to complete the installation of the USB driver for the S6LX16 board.
4. Windows will automatically assign a COM port to the board. The Windows Device Manager shows which COM port is assigned to the evaluation board's USB-to-UART bridge. Follow these instructions to determine the assigned COM port:
 - a. Open the Device Manager by clicking on the **Windows Button**  and navigate to **Control Panel → Device Manager**.
 - b. In the Device Manager, scroll down to **Ports (COM & LPT)** and expand the list. You will see the **S6LX16 Eval Board** and its assigned COM port. In the example below it is COM7. Make note of this COM port number for use with the serial terminal you will use elsewhere in this design tutorial.



Installing a Serial Console on a Windows 7 Host

Starting with Windows 7, Microsoft no longer includes the HyperTerminal terminal emulator software. However, this example design requires use of terminal emulation software for a serial console connection to the S6LX16 Eval Board. A suitable free and open-source replacement for HyperTerminal is TeraTerm. Download and install instructions for TeraTerm can be found at <http://en.sourceforge.jp/projects/ttssh2>.

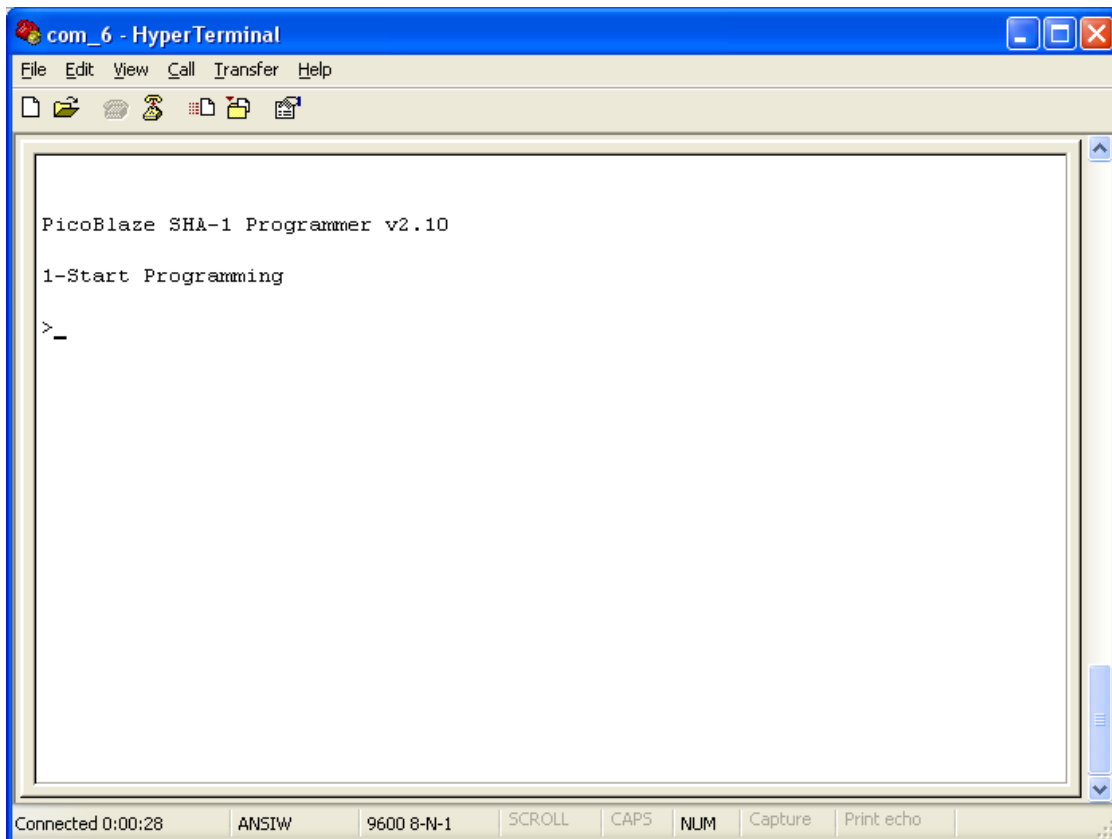
Running the Demo Files

You can load the FPGA and run the test applications without building the design by using the demo scripts and the pre-built bitstream files. You must have the Xilinx tools installed on your host, and have the hardware set up and connected as per the previous steps.

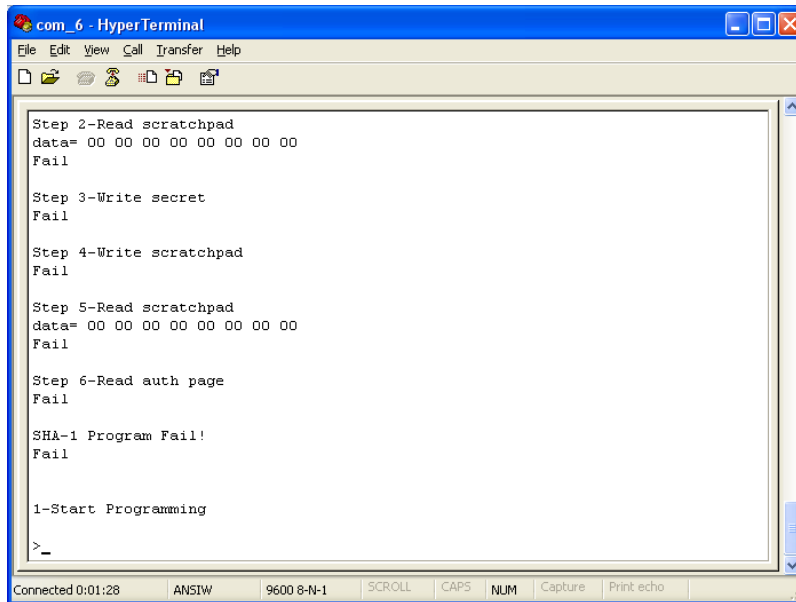
SHA-1 Programming Test Demo

1. Set up the board as described in [Setting Up The S6LX16 Evaluation Board](#)
2. Slide the power switch, SW1, from **CHARGE** to **ON**. This will power the board, as indicated by the D12 amber power-good LED illuminating.
3. Start a Serial terminal session and set the serial port parameters to **9600** baud rate, **no** parity, **8** bits, **1** stop bit and no flow control.
4. Open a command window in the
`<installation>\SHA1_PROG_TEST\ready_for_download` folder and enter:

demo.bat
5. The SHA-1 Programmer PicoBlaze application will run as soon as the FPGA is loaded with the bitstream. The program prompts you to press '1' to perform the steps to program the SHA-1 EEPROM.



6. Depending on the status of the Maxim DS28E01PMOD you will see different output in your Serial terminal session.
 - a. If the Maxim DS28E01PMOD isn't plugged into the LX16 board, or if it is plugged into an incorrect spot on the board you will see this:



```

com_6 - HyperTerminal
File Edit View Call Transfer Help

Step 2-Read scratchpad
data= 00 00 00 00 00 00 00 00
Fail

Step 3-Write secret
Fail

Step 4-Write scratchpad
Fail

Step 5-Read scratchpad
data= 00 00 00 00 00 00 00 00
Fail

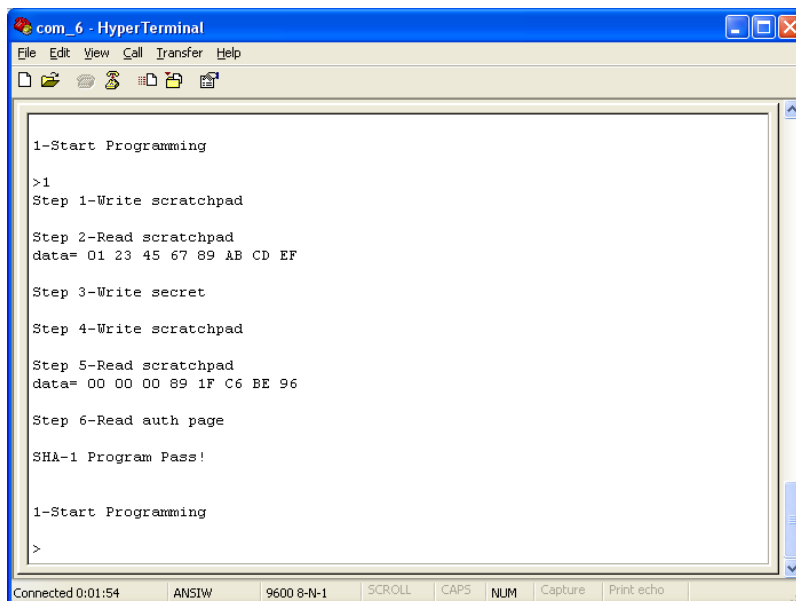
Step 6-Read auth page
Fail

SHA-1 Program Fail!
Fail

1-Start Programming
>_
Connected 0:01:28  ANSIW  9600 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo

```

- b. If the Maxim PMOD is plugged into the correct spot you will see this:



```

com_6 - HyperTerminal
File Edit View Call Transfer Help

1-Start Programming
>1
Step 1-Write scratchpad

Step 2-Read scratchpad
data= 01 23 45 67 89 AB CD EF

Step 3-Write secret

Step 4-Write scratchpad

Step 5-Read scratchpad
data= 00 00 00 89 1F C6 BE 96

Step 6-Read auth page

SHA-1 Program Pass!

1-Start Programming
>
Connected 0:01:54  ANSIW  9600 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo

```

7. You can rerun the demo as many times as you like by typing '1' at the command prompt. Notice that the data bytes for step 5 change each time the program is run. These bytes are the challenge bytes used by the SHA-1 algorithm and are produced by the serial random number generator in the FPGA design.

SHA-1 Authentication Example Demo

1. Set up the board as described in [Setting Up The S6LX16 Evaluation Board](#)
2. Slide the power switch, SW1, from **CHARGE** to **ON**. This will power the board, as indicated by the D12 amber power-good LED illuminating.
3. Start a Serial terminal session and set the serial port parameters to **9600** baud rate, **no** parity, **8** bits, **1** stop bit and no flow control.
4. Open a command window in the
`<installation>\SHA1_AUTH_EXAMPLE\ready_for_download` folder and enter:

demo.bat

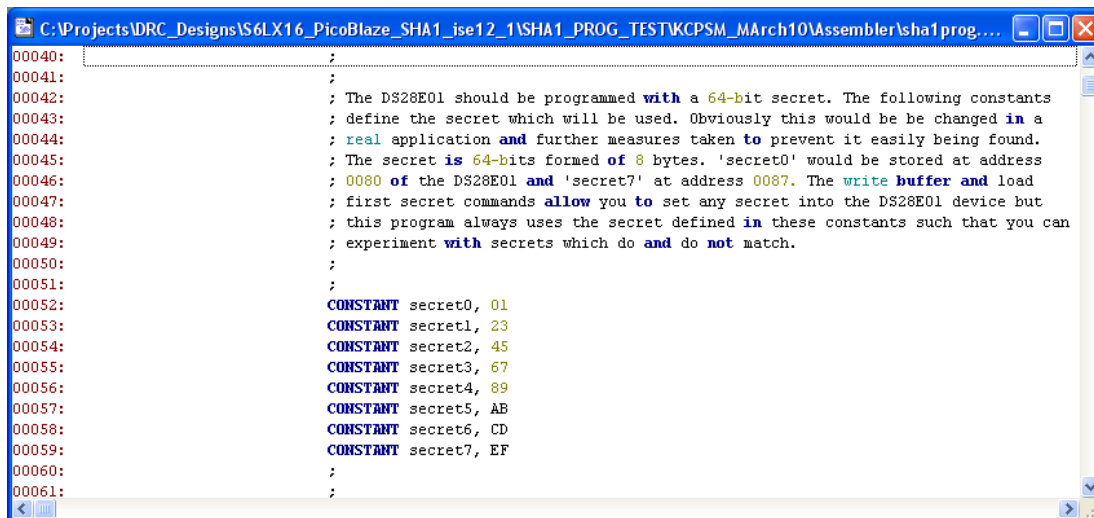
5. The SHA-1 Authentication Example PicoBlaze application will run as soon as the FPGA is loaded with the bitstream. When the application starts you should see the D5 LED quickly blink, indicating that the PicoBlaze code descrambling keys have been authenticated. This verifies that the BRAM contents have not been tampered with. If the D5 LED stays lit this indicates that either the PicoBlaze BRAM has been modified or the design was implemented without updating the descrambling keys from the last time the PicoBlaze code was compiled. A second or so after the D5 LED is turned off the D1, D2 and D4 LEDs should start to blink in a binary counting pattern. If they do not start to blink this indicates that the SHA-1 authentication has failed. The SHA-1 authentication will fail if:
 - a. The SHA-1 EEPROM hasn't been programmed.
 - b. The secret key programmed in the EEPROM doesn't match the key in the FPGA.
 - c. The Maxim DS28E01PMOD isn't plugged into the LX16 board, or if it is plugged into the incorrect PMOD connector on the board.
6. You can rerun the demo as many times as you like by executing the demo.bat batch script.

Implementing the Hardware Designs

SHA-1 Programming Test

Introduction

This design must be run first before the authentication test design. The secret key is hard-coded in the PicoBlaze application code and should be changed if this design is to be used in a real-world application. You can tell from the PicoBlaze code file snippet below ([sha1prog.psm](#)) that the 8 bytes of the secret key are set to “01 23 45 67 89 AB CD EF”. This is obviously not very secure and is likely to be the first key that any thief would attempt. It is strongly recommended the user change this key to something more meaningful and thus more secure.



```

00040: ;
00041: ;
00042: ; The DS28E01 should be programmed with a 64-bit secret. The following constants
00043: ; define the secret which will be used. Obviously this would be changed in a
00044: ; real application and further measures taken to prevent it easily being found.
00045: ; The secret is 64-bits formed of 8 bytes. 'secret0' would be stored at address
00046: ; 0080 of the DS28E01 and 'secret7' at address 0087. The write buffer and load
00047: ; first secret commands allow you to set any secret into the DS28E01 device but
00048: ; this program always uses the secret defined in these constants such that you can
00049: ; experiment with secrets which do and do not match.
00050: ;
00051: ;
00052: CONSTANT secret0, 01
00053: CONSTANT secret1, 23
00054: CONSTANT secret2, 45
00055: CONSTANT secret3, 67
00056: CONSTANT secret4, 89
00057: CONSTANT secret5, AB
00058: CONSTANT secret6, CD
00059: CONSTANT secret7, EF
00060: ;
00061: ;

```

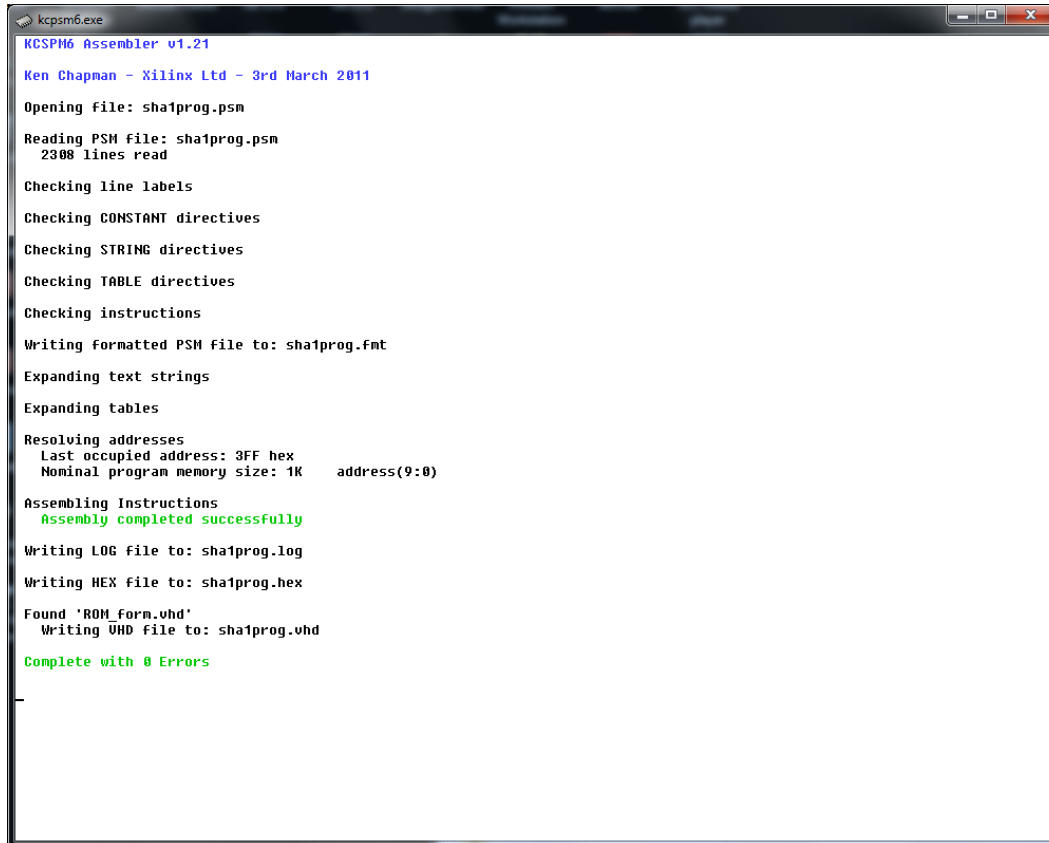
1. If you want to modify the authentication secret you will need to recompile the PicoBlaze application before implementing the FPGA design. A batch script is provided run the PicoBlaze assembler.
2. The PicoBlaze assembler will produce the necessary files for implementing this design. The new files will overwrite the old files, so be sure to backup your original files (that you know work) before recompiling. Open a command window in the <installation>\SHA1_PROG_TEST\source\psm folder and enter:

assemble_pb.bat

The batch file will run the following command to assemble the PicoBlaze source file for the project and produce the files needed for implementation:

```
..\..\KCPSM6_Release2_31March11\kcpsm6 sha1prog.psm
```

- When the PicoBlaze assembler runs, and if there aren't any errors in the PicoBlaze code, you should see a screen like the one below. If there are errors you will be interactively alerted with the line number to where the problem is in the code file and prompted to fix the error before re-running the assembler.



```
kcpsm6.exe
KCSPM6 Assembler v1.21
Ken Chapman - Xilinx Ltd - 3rd March 2011

Opening file: sha1prog.psm
Reading PSM file: sha1prog.psm
2308 lines read

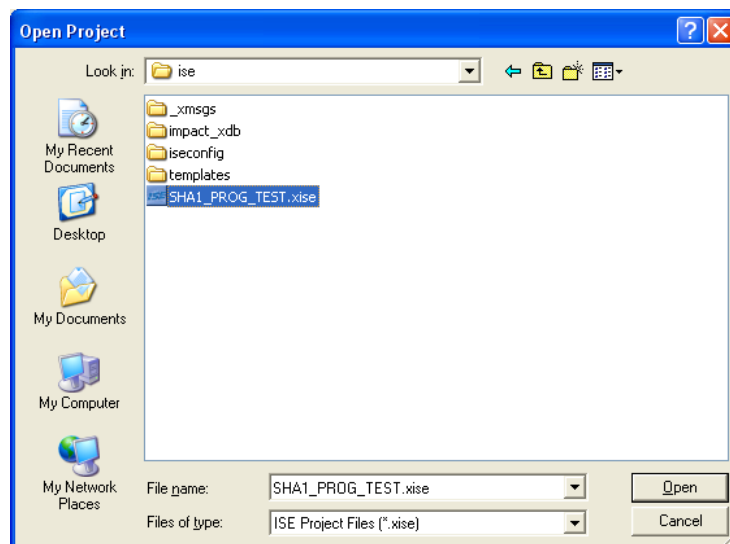
Checking line labels
Checking CONSTANT directives
Checking STRING directives
Checking TABLE directives
Checking instructions
Writing formatted PSM file to: sha1prog.fmt
Expanding text strings
Expanding tables
Resolving addresses
Last occupied address: 3FF hex
Nominal program memory size: 1K address(9:0)

Assembling Instructions
Assembly completed successfully

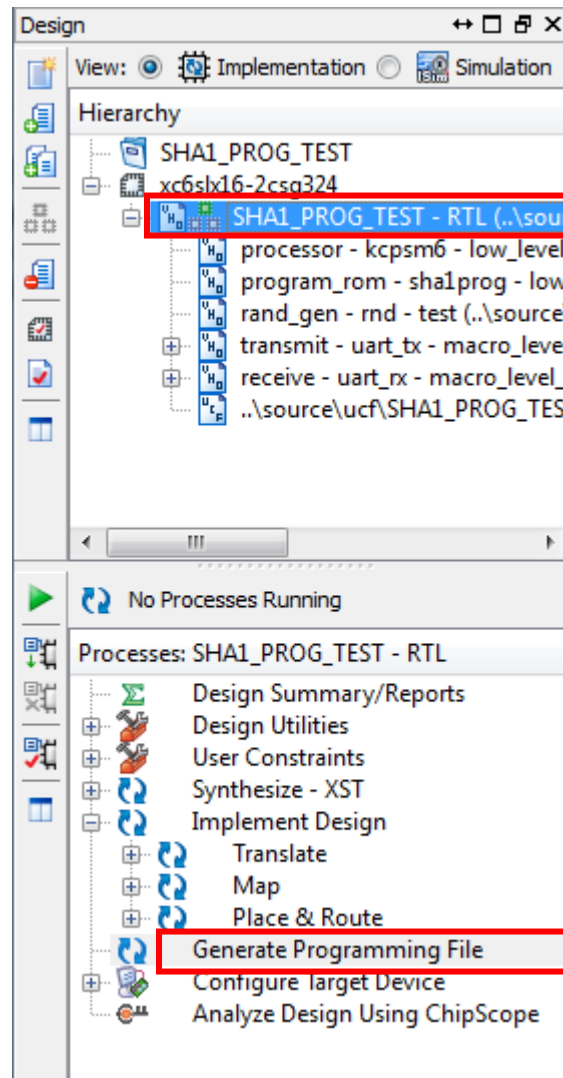
Writing LOG file to: sha1prog.log
Writing HEX file to: sha1prog.hex
Found 'ROM_form.vhd'
Writing VHD file to: sha1prog.vhd

Complete with 0 Errors
```

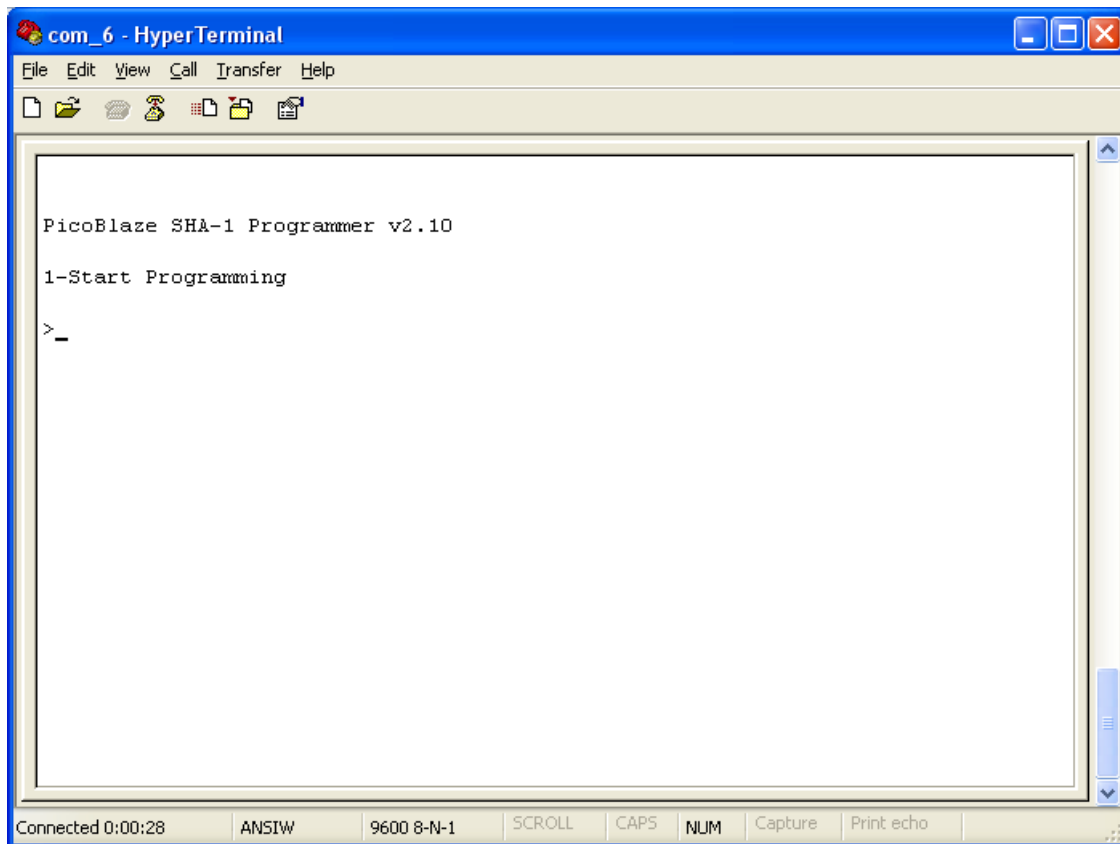
- Open the <installation>\SHA1_PROG_TEST\ise\SHA1_PROG_TEST.xise file in Xilinx ISE Project Navigator (ISE).



5. Select top-level file in the design hierarchy (**SHA1_PROG_TEST - RTL**) in the **Hierarchy** pane and then double-click on **Generate Programming File** in the **Processes** pane of the ISE GUI to build the design. This will take a few minutes to run the implementation tools to create the FPGA bitstream. You can complete the next couple of steps while you are waiting.



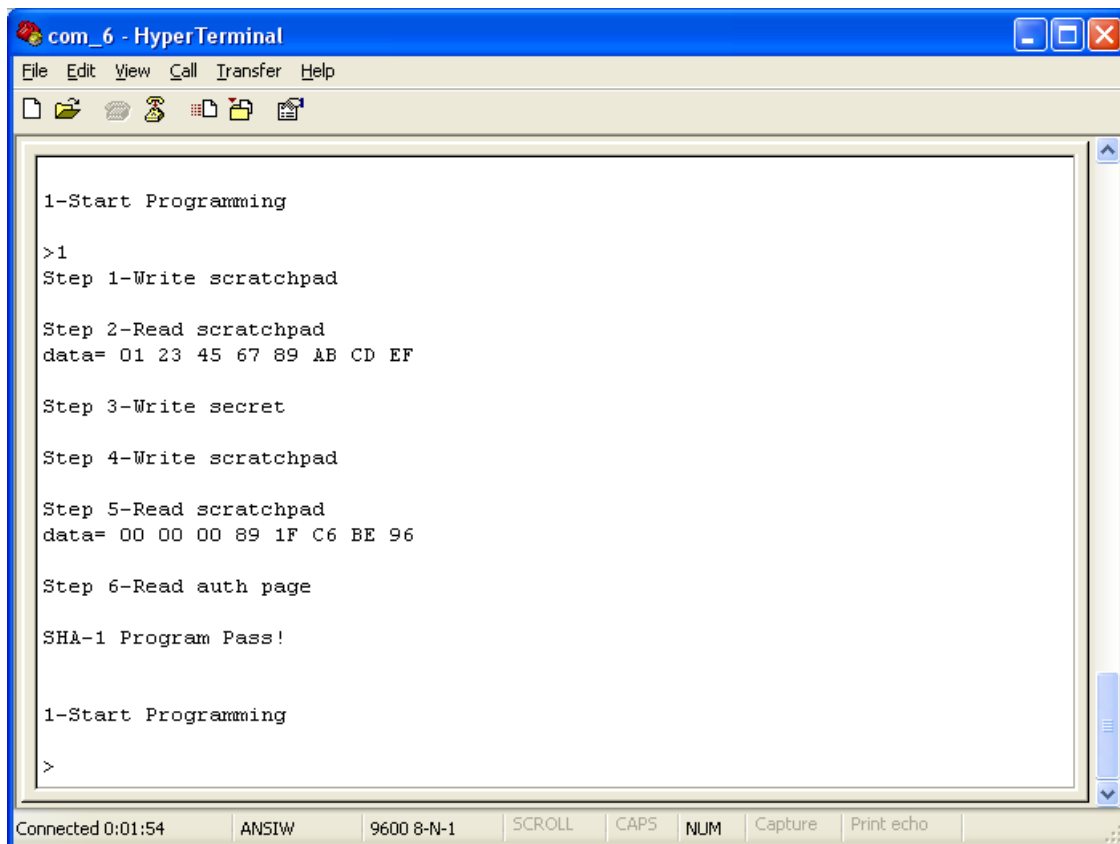
6. Plug the Maxim DS28E01PMOD, component side up, into the upper row of pins on the J8 PMOD connector.
7. Slide the board power switch (**SW1, PWR**) to the **ON** position.
8. Start a serial terminal session and set the serial port parameters to **9600** baud rate, **no** parity, **8** bits, **1** stop bit and no flow control.
9. Select **Configure Target Device** from the ISE GUI to download the FPGA design to the board. The PicoBlaze application will run on the board and you should see the following in your serial terminal.



```
com_6 - HyperTerminal
File Edit View Call Transfer Help
PicoBlaze SHA-1 Programmer v2.10
1-Start Programming
>_
Connected 0:00:28  ANSIW  9600 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
```

10. Type '1' at the prompt to start the process of programming the DS28E01 EEPROM with the SHA-1 secret key.

11. The PicoBlaze application will automatically perform the steps to:
- Write the secret key to the scratchpad memory area of the DS28E01 EEPROM
 - Read the EEPROM scratchpad memory back into the FPGA
 - Issue the command to the EEPROM to transfer the secret key from the scratchpad memory to its special write-only area of memory. This is part of the security features of the DS28E01 EEPROM. The secret key can be written to the device, but it can never be read back.
 - Write the EEPROM scratchpad memory with a series of 3 random number challenge bytes.
 - Read the challenge bytes back from the EEPROM. The random challenge bytes are written to the EEPROM straight from the random number generator in the previous step and the PicoBlaze application code doesn't know what they are, but needs to know them to create the SHA-1 HASH.
 - Read the unique ID and the computed HASH from the EEPROM, compute the HASH in the FPGA, and compare them. If they match then the programming has been successful. If they don't then something has gone wrong that needs to be corrected before these steps can be run again.



```
com_6 - HyperTerminal
File Edit View Call Transfer Help

1-Start Programming

>1
Step 1-Write scratchpad

Step 2-Read scratchpad
data= 01 23 45 67 89 AB CD EF

Step 3-Write secret

Step 4-Write scratchpad

Step 5-Read scratchpad
data= 00 00 00 89 1F C6 BE 96

Step 6-Read auth page

SHA-1 Program Pass!

1-Start Programming

>
```

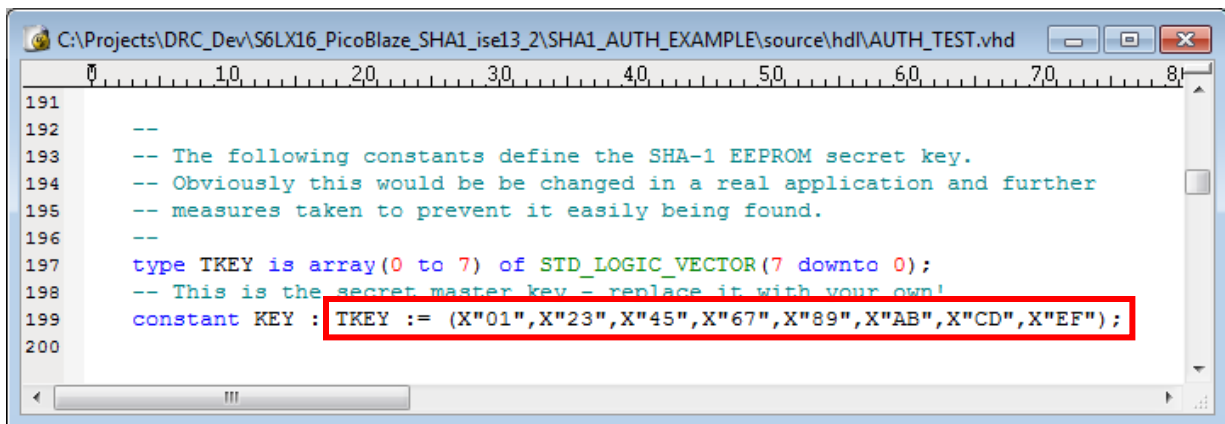
Connected 0:01:54 ANSIW 9600 8-N-1 SCROLL CAPS NUM Capture Print echo

12. Feel free to press '1' again to reprogram the EEPROM. Notice that for each time you press '1' to perform the programming steps the random number challenge bytes reported in **Step 5-Read scratchpad** are different. This is by design because each time running the application new bytes are read from the random number generator. The random number generator is a ring oscillator using a combinatorial logic loop and a flip-flop on the output that goes metastable very often and shifts out a random bit '1' or '0'. It is this metastability that creates the random byte as these bits are shifted into a register in the PicoBlaze application code. Since this ring oscillator scheme also varies with process, voltage and temperature, it produces a truly random number all of the time.
13. This concludes the programming of the EEPROM. You may now proceed to the SHA-1 Authentication Example project.

SHA-1 Authentication Example

Introduction

The SHA-1 Programming Test design must be run first before this design. The secret key is hard-coded in the VHDL source code and should be changed if this design is to be used in a real-world application. You can tell from the VHDL code file snippet below (auth_test.vhd) that the 8 bytes of the secret key are set to "01 23 45 67 89 AB CD EF". This is obviously not very secure and is likely to be the first key that any thief would attempt. It is strongly recommended the user change this key to something more meaningful and thus more secure. Of course, this secret key must also match the secret key that was programmed in the EEPROM in the SHA-1 Programming Test design.

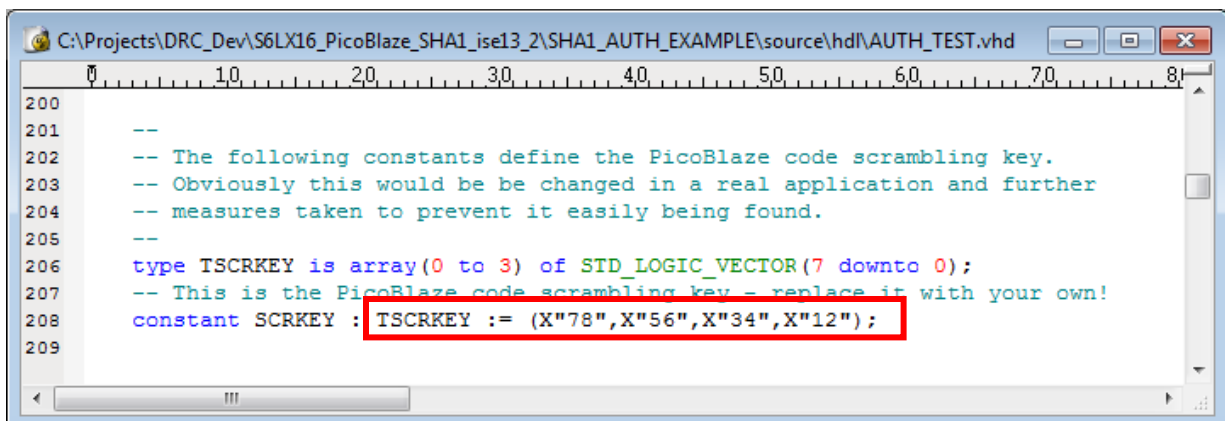


```

191
192 --
193 -- The following constants define the SHA-1 EEPROM secret key.
194 -- Obviously this would be changed in a real application and further
195 -- measures taken to prevent it easily being found.
196 --
197 type TKEY is array(0 to 7) of STD_LOGIC_VECTOR(7 downto 0);
198 -- This is the secret master key - replace it with your own!
199 constant KEY : TKEY := (X"01",X"23",X"45",X"67",X"89",X"AB",X"CD",X"EF");
200

```

For extra design security, the PicoBlaze application BRAM contents are also scrambled with a secret key. This is done to detect if a thief has attempted to steal the FPGA design by modifying the BRAM contents. If tampering is detected then the "TAMPER_DETECT" LED (D5) on the board will remain lit and the FPGA design will be disabled. The secret scrambler/descrambler key is stored in the AUTH_TEST.VHD VHDL source code file:

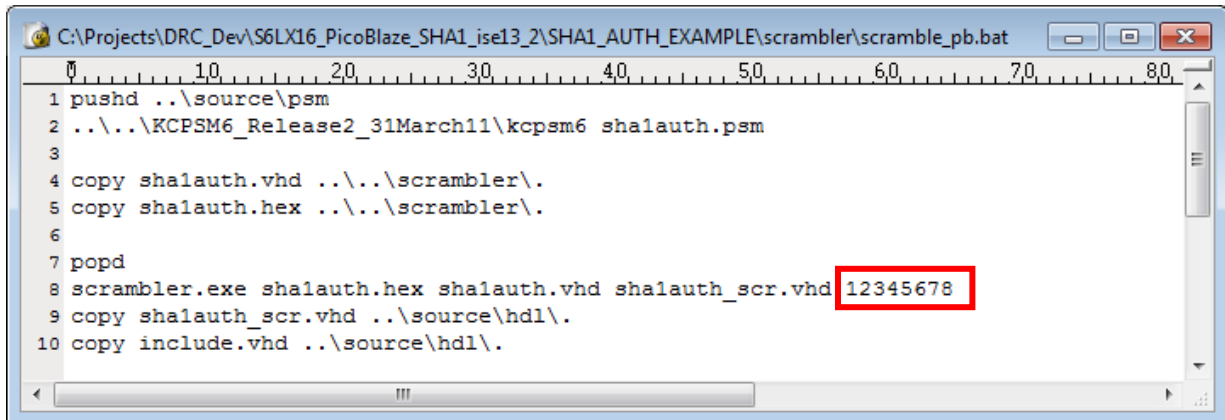


```

200
201 --
202 -- The following constants define the PicoBlaze code scrambling key.
203 -- Obviously this would be changed in a real application and further
204 -- measures taken to prevent it easily being found.
205 --
206 type TSCRKEY is array(0 to 3) of STD_LOGIC_VECTOR(7 downto 0);
207 -- This is the PicoBlaze code scrambling key - replace it with your own!
208 constant SCRKEY : TSCRKEY := (X"78",X"56",X"34",X"12");
209

```


The key used when you run the scrambler application at the command line must match the scrambler key stored in the VHDL source file. Below is the contents of the **scramble_pb.bat** script:



```
C:\Projects\DRC_Dev\S6LX16_PicoBlaze_SHA1_ise13_2\SHA1_AUTH_EXAMPLE\scrambler\scramble_pb.bat
1 pushd ..\source\psm
2 .....\KCPSM6_Release2_31March11\kcpsm6 shalauth.psm
3
4 copy shalauth.vhd .....\scrambler\
5 copy shalauth.hex .....\scrambler\
6
7 popd
8 scrambler.exe shalauth.hex shalauth.vhd shalauth_scr.vhd 12345678
9 copy shalauth_scr.vhd ..\source\hdl\
10 copy include.vhd ..\source\hdl\
```

For the same reasons described above this is obviously not a very secure key and should be changed.

This design implements the Identify Friend or Foe (IFF) SHA-1 authentication method as described in [Figure 2](#). As described previously, the IFF method holds the user logic, or ‘secret sauce’, disabled until the EEPROM is authenticated with the FPGA. In this design the user logic is a large binary counter that has the upper bits mapped to the LEDs connected to the FPGA. The user logic could easily be any design more sophisticated and worthy of protecting from cloning or unauthorized use. The LEDs in this case are an illustrative example of how SHA-1 IFF authentication works.

If you want to modify the authentication secret you will need to recompile the PicoBlaze application before implementing the FPGA design. A batch script is provided to run the PicoBlaze assembler. Likewise, if you want to change the scrambler key you need to re-run the scrambler application on the design with the new key. Don't forget to update the AUTH_TEST.VHD VHDL source with the new authentication secret and scrambler keys.

1. The PicoBlaze assembler and scrambler will produce the necessary files for implementing this design. The new files will overwrite the old files, so be sure to backup your original files (that you know work) before recompiling. Open a command window in the <installation>\SHA1_AUTH_EXAMPLE\scrambler folder and enter:

scramble_pb.bat

The batch file will run the following commands to assemble the PicoBlaze source file for the project, scramble the BRAM contents, and produce the files needed for implementation:

```
pushd ..\source\psm
..\..\KCPSM6_Release2_31March11\kcpsm6 shalauth.psm

copy shalauth.vhd ..\..\scrambler\.
copy shalauth.hex ..\..\scrambler\.

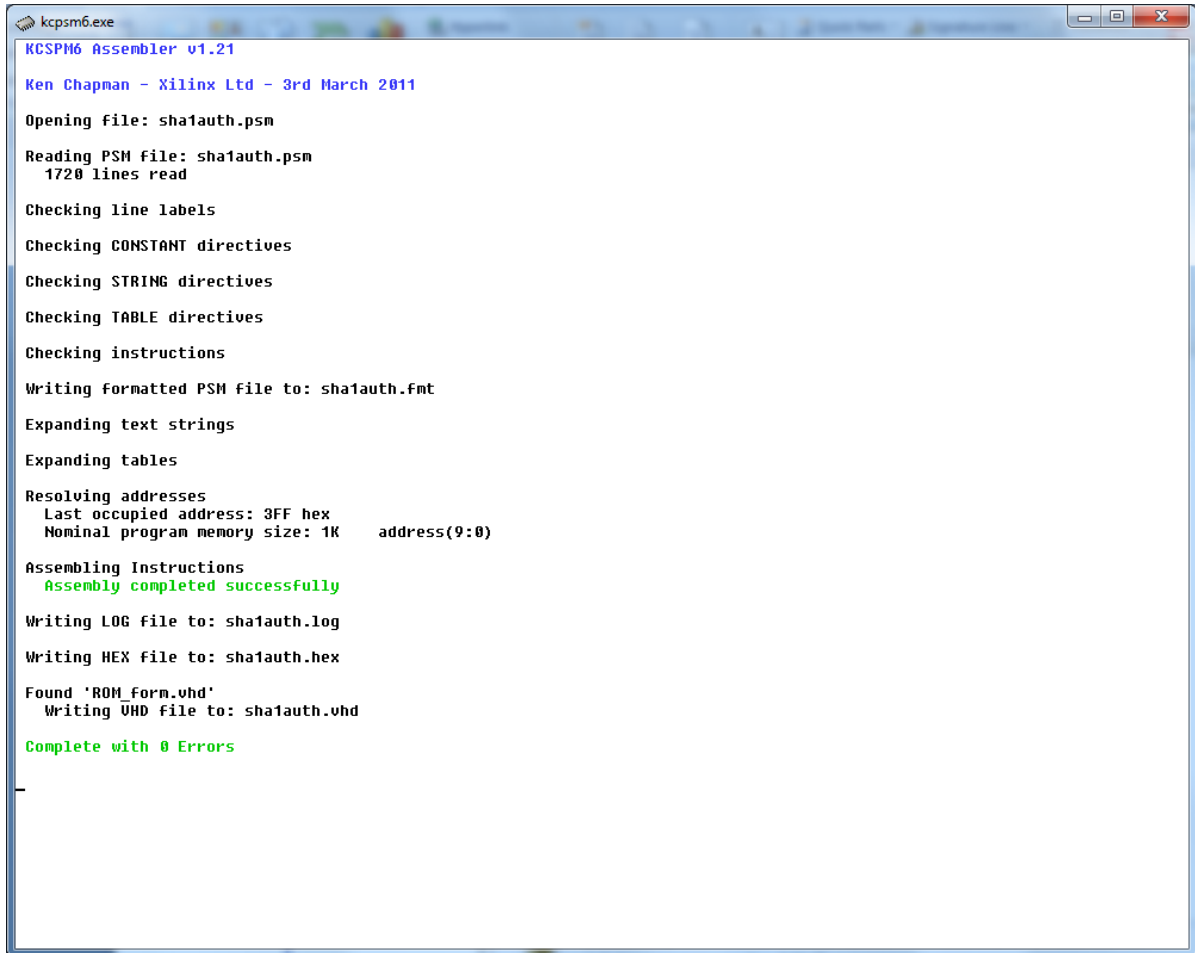
popd
scrambler.exe shalauth.hex shalauth.vhd shalauth_scr.vhd 12345678
copy shalauth_scr.vhd ..\source\hdl\.
copy include.vhd ..\source\hdl\.
```

2. The secret key for the scrambler is required to be a 32 bit hexadecimal value. The scrambler command line is formatted as

scrambler.exe <input>.hex <input>.vhd <output>.vhd <secret key>

Remember that whatever value you choose for the scrambler at the command line you must also specify in the AUTH_TEST.VHD VHDL source file.

- When the PicoBlaze assembler runs, and if there aren't any errors in the PicoBlaze code, you should see a screen like the one below. If there are errors you will be interactively alerted with the line number to where the problem is in the code file and prompted to fix the error before **re-running the assembler**.



```
kcpsm6.exe
KCSPM6 Assembler v1.21
Ken Chapman - Xilinx Ltd - 3rd March 2011

Opening file: sha1auth.psm
Reading PSM file: sha1auth.psm
1720 lines read

Checking line labels
Checking CONSTANT directives
Checking STRING directives
Checking TABLE directives
Checking instructions
Writing formatted PSM file to: sha1auth.fmt
Expanding text strings
Expanding tables
Resolving addresses
Last occupied address: 3FF hex
Nominal program memory size: 1K address(9:0)

Assembling Instructions
Assembly completed successfully

Writing LOG file to: sha1auth.log
Writing HEX file to: sha1auth.hex
Found 'ROM_form.vhd'
Writing VHD file to: sha1auth.vhd

Complete with 0 Errors
```

4. The scrambler will produce a file named `include.vhd` which contains the code signature keys required for the descrambler portion of the design to operate correctly in the FPGA. New keys will be generated every time the scrambler is run, even if the PicoBlaze source code has not changed. The contents of this file must be copied to the `AUTH_TEST.VHD` VHDL source code file (new keys replace the old keys) before the design is implemented.

Contents of `include.vhd`:

```

1  constant INIT_VAL:UNSIGNED (143 downto 0) := "X"2CED2CED2CED2CED2CED2CED2CED2CED2CED2"; -- Initial State
2  constant LAST_VAL:UNSIGNED (143 downto 0) := "X"B4B9B47156030B26A90CD0F580E8BB2A0715"; -- Final State

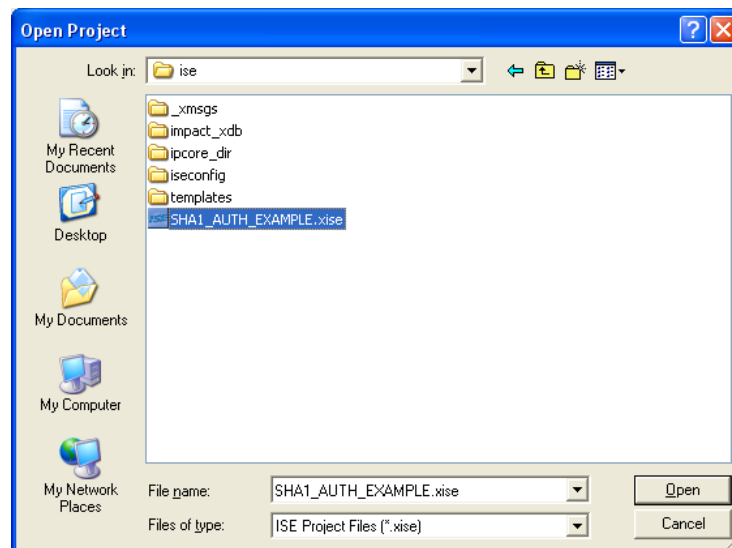
```

VHDL code snippet from AUTH_TEST.VHD

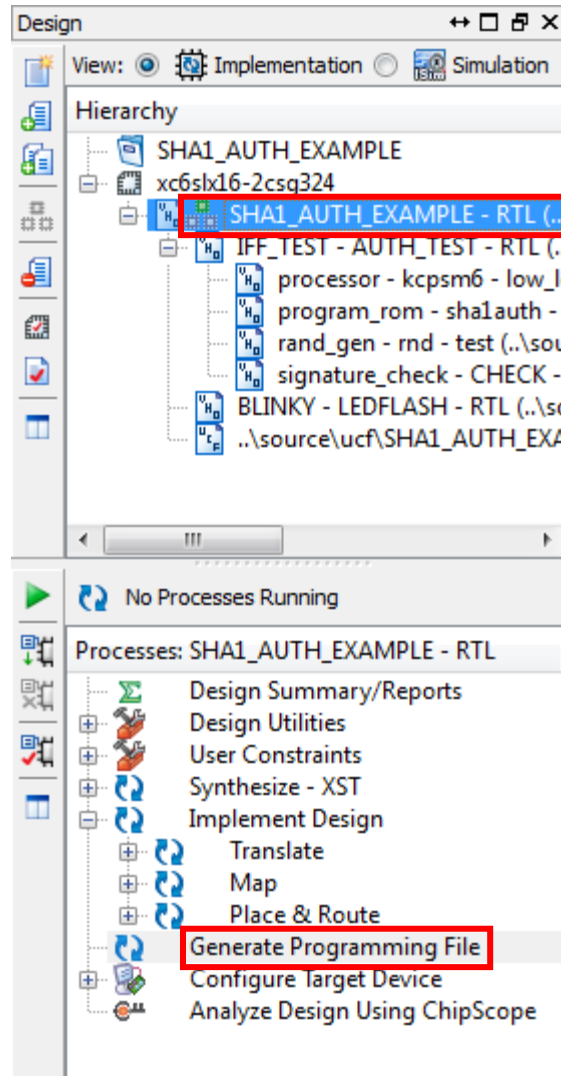
```
C:\Projects\DRC_Dev\S6LX16_PicoBlaze_SHA1_isel3_2\SHA1_AUTH_EXAMPLE\source\hdl\AUTH_TEST.vhd
```

```
210 --  
211 -- Code Signature Keys  
212 -- These are produced by the scrambler and are pasted here  
213 --  
214 constant INIT_VAL:UNSIGNED(143 downto 0) := "X"2CED2CED2CED2CED2CED2CED2CED2CED2CED"; -- Initial State  
215 constant LAST_VAL:UNSIGNED(143 downto 0) := "x"B4B9B47156030B26A90CDOF580E8BB2A0715"; -- Final State
```

- Open the `<installation>\SHA1_AUTH_EXAMPLE\ise\SHA1_AUTH_EXAMPLE.xise` file in Xilinx ISE Project Navigator (ISE).



6. Select top-level file in the design hierarchy (**SHA1_AUTH_EXAMPLE - RTL**) in the **Hierarchy** pane and then double-click on **Generate Programming File** in the **Processes** pane of the ISE GUI to build the design. You can complete the next couple of steps while you are waiting.



7. Plug the Maxim DS28E01PMOD, component side up, into the upper row of pins on the J8 PMOD connector.
8. Slide the board power switch (**SW1, PWR**) to the **ON** position.
9. Select **Configure Target Device** from the ISE GUI to download the FPGA design to the board. The PicoBlaze application will immediately start to run on the board.
10. The PicoBlaze application will automatically perform the steps to:
 - a. Write the EEPROM scratchpad memory with a series of 3 random number challenge bytes.
 - b. Read the challenge bytes back from the EEPROM. The random challenge bytes are written to the EEPROM straight from the random number generator in the previous step and the PicoBlaze application code doesn't know what they are, but needs to know them to create the SHA-1 HASH.
 - c. Read the unique ID and the computed HASH from the EEPROM, compute the HASH in the FPGA, and compare them. If they match then the authentication has been successful. If they don't then something has gone wrong that needs to be corrected before these steps can be run again.
11. Notice that when the application completes the D5 LED will quickly blink, indicating that the PicoBlaze code signature keys have been authenticated. This verifies that the BRAM contents have not been tampered with. If the D5 LED stays lit this indicates that either the PicoBlaze BRAM has been modified or the design was implemented without updating the code signature keys from the last time the PicoBlaze code was compiled. A second or so after the D5 LED is turned off the D1, D2 and D4 LEDs should start to blink in a binary counting pattern. This is a result of the EEPROM being identified as a 'Friend'. When the PicoBlaze application code determines the EEPROM HASH matches the FPGA HASH, the FRIEND signal asserted to the user logic, thus enabling the user logic to start operation. The user logic is a large binary counter with the upper bits mapped to the LEDs on the LX16 board. If they do not start to blink this indicates that the SHA-1 authentication has failed and the EEPROM has been identified to be a 'Foe'. The SHA-1 authentication will fail if:
 - a. The SHA-1 EEPROM hasn't been programmed.
 - b. The secret key programmed in the EEPROM doesn't match the key in the FPGA.
 - c. The Maxim DS28E01PMOD isn't plugged into the LX16 board, or if it is plugged into the incorrect PMOD connector on the board.
12. This concludes the IFF authentication of the EEPROM and FPGA.

Feel free to experiment with these FPGA designs. Perhaps try out different secret authentication and scrambler keys, or add your own user logic instead of the binary counter. Keep in mind, though, that the PicoBlaze microcontroller is limited to 1024 instructions (the depth of the FPGA block RAM), so any changes to the PicoBlaze code will have to be made with this limited resource in mind.

Troubleshooting Design Changes

Introduction

Because the Authentication Test design does not have a UART to aid in debugging any design changes, you can use the instantiated (but commented out) Xilinx ChipScope Integrated Logic Analyzer (ILA) and Integrated Controller (ICON) cores to debug any design changes you may make. The PicoBlaze address, port_id, in_port, out_port, read_strobe and write_strobe ports are already connected to the ChipScope ILA core and there is a ChipScope project included in the design that has the waveform and trigger already setup as well. This is not meant to be a full tutorial on how to use ChipScope, but rather a set of instructions, albeit brief, to describe how to use the pre-configured ChipScope ILA and ICON cores in this design.

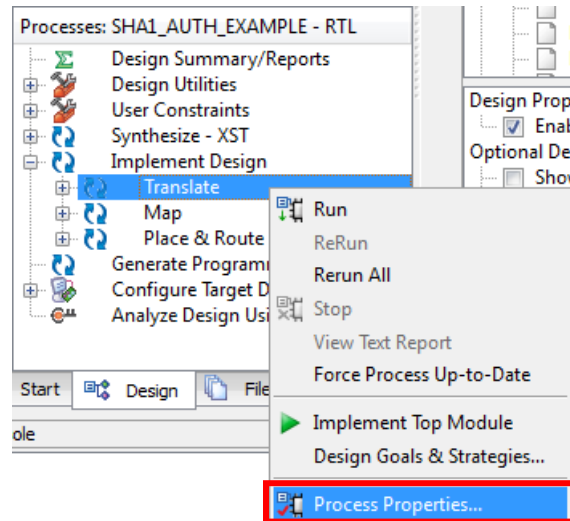
1. The VHDL code segments in the AUTH_TEST.VHD file are shown below that you will need to uncomment to use ChipScope to debug your design changes:

```
00167:  --
00168:  -- declaration of ChipScope ILA and ICON
00169:  -- VERY useful for debugging PicoBlaze as it is running in hardware
00170:  -- IMPORTANT!:
00171:  -- Remember to remove these (using comments) from the design when you are satisfied
00172:  -- with your design changes and your debugging is complete. It is a significant
00173:  -- security risk, and waste of FPGA resources, to leave them in the design
00174:  --
00175:  -- component icon
00176:  --     PORT (
00177:  --         CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
00178:  --     end component;
00179:  --
00180:  -- component ila
00181:  --     PORT (
00182:  --         CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
00183:  --         CLK : IN STD_LOGIC;
00184:  --         TRIG0 : IN STD_LOGIC_VECTOR(35 DOWNTO 0));
00185:  --     end component;
00186:  --
```

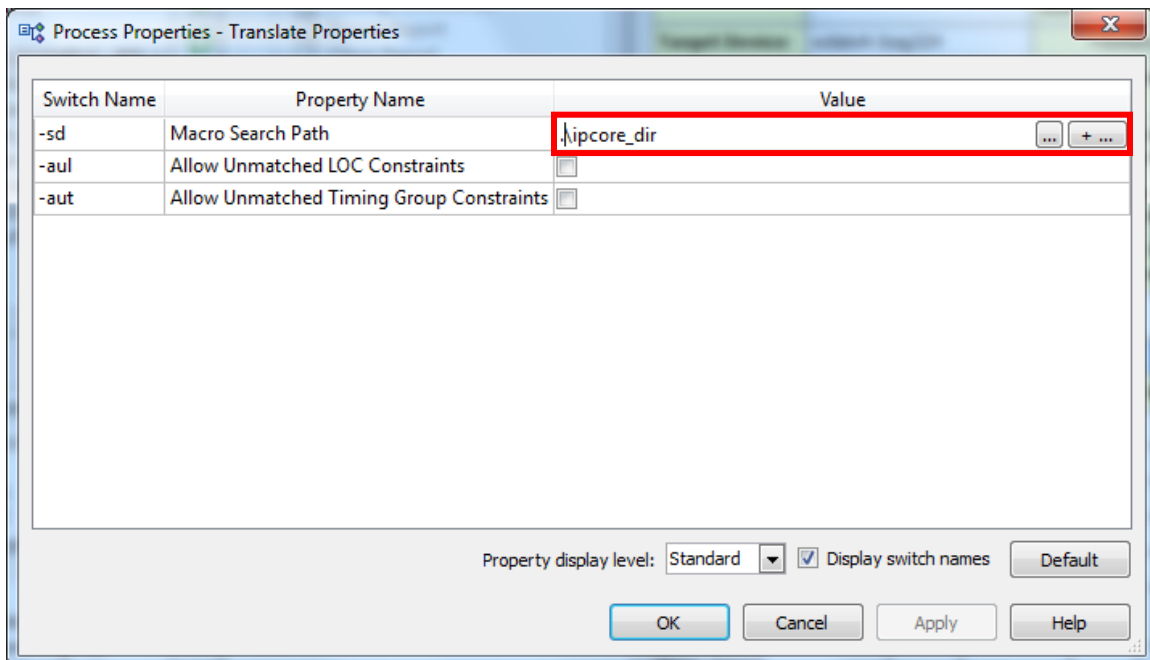
```
00269:  --
00270:  -- ChipScope signals
00271:  --
00272:  -- signal trig : std_logic_vector(35 downto 0);
00273:  -- signal control : std_logic_vector(35 downto 0);
00274:  --
```

```
00607:  -----
00608:  -- ChipScope ICON and ILA core instances.
00609:  -- Use these to debug running PicoBlaze code
00610:  -- The ILA is currently configured to trigger on address[9:0] to easily track program
00611:  -- IMPORTANT!:
00612:  -- Remember to remove these (using comments) from the design when you are satisfied
00613:  -- with your design changes and your debugging is complete. It is a significant
00614:  -- security risk, and waste of FPGA resources, to leave them in the design
00615:  -----
00616:  -- icon_1 : icon
00617:  -- port map (
00618:  --     CONTROL0 => control);
00619:  --
00620:  -- TRIG <= write_strobe & read_strobe & in_port & out_port & port_id & address(9 downto 0);
00621:  --
00622:  -- ila_1 : ila
00623:  -- port map (
00624:  --     CONTROL => control,
00625:  --     CLK => s6_clk,
00626:  --     TRIG0 => trig);
00627:  --
```

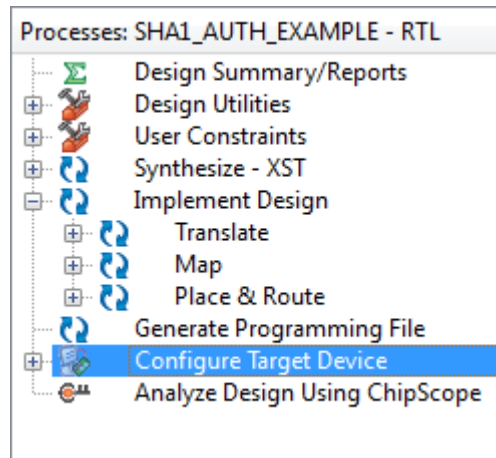
2. Run the scramble_pb.bat script as described earlier if you made any PicoBlaze code changes.
3. Update the descrambler keys in AUTH_TEST.VHD as described earlier if you ran the scramble_pb.bat script
4. Verify the path to the ChipScope core netlists is correct in Translate settings in ISE.
 - a. Right-click on **Translate** and then click on **Process Properties...**



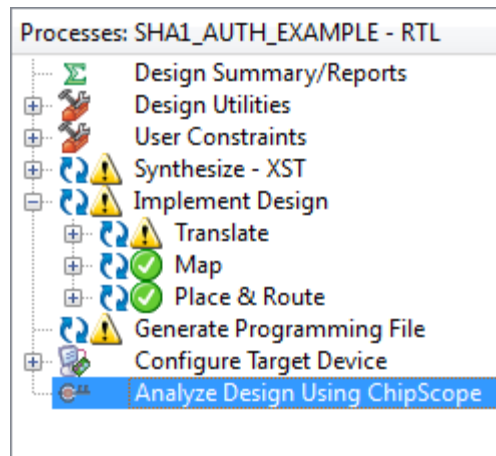
- b. Change the Macro Search Path to **./ipcore_dir**. This will resolve to an absolute path once we close this window, so remember that if you ever move this project to a new folder.



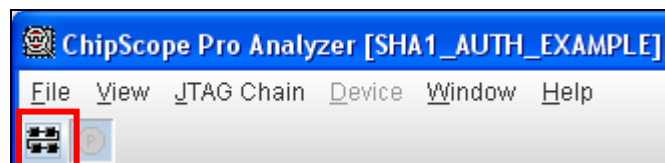
- With the S6LX16 board turned on and the Xilinx Platform Cable USB connected, implement the design and program the FPGA with the bitstream. Select **Configure Target Device** in the ISE GUI.



- Select **Analyze Design Using ChipScope** in the ISE GUI.



- Click on the icon on the ChipScope toolbar to open the JTAG cable. ChipScope will recognize the ILA and ICON cores are part of the FPGA design and will load the Trigger and Waveform windows with the busses and signals.

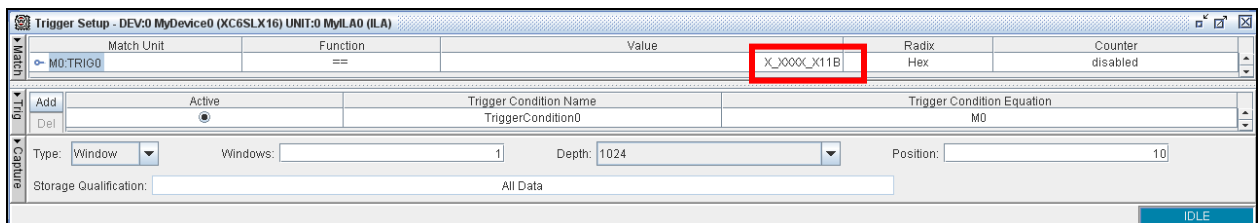


8. Open the <installation>\SHA1_AUTH_EXAMPLE\source\psm\sha1auth.log PicoBlaze assembly log file in a text editor. This file correlates the BRAM addresses to the PicoBlaze instructions in the source code. Since the ChipScope ILA is already configured to use the PicoBlaze code address as a trigger, look for the start address of the section of code you are attempting to debug. For example, examine the section of code where the count for the matching number of authentication bytes should equal 0x14.

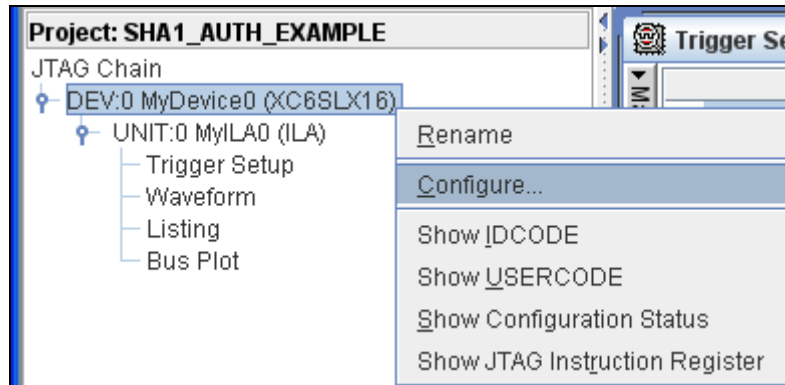
```

111 11C01      ADD sC, 01                ;decrement match counter
112 00030      display_mac_byte: LOAD s0, s3          ;display byte
113 19A01      SUB sA, 01                ;counts bytes per variable
114 32117      JUMP Z, 117[next_mac_var]
115 11B01      ADD sB, 01
116 2210C      JUMP 10C[mac_match_byte]
117 1DB0B      next_mac_var: COMPARE sB, 0B[var_A3]    ;test for last byte
118 3211B      JUMP Z, 11B[report_mac]
119 19B07      SUB sB, 07                ;point to next variable
11A 2210B      JUMP 10B[mac_match_var]
11B          ;
11B          ;MAC has passed if all 20 bytes matched
11B          ;
11B 1DC14      report_mac: COMPARE sC, 14            ;20 bytes should have matched
11C 3611F      JUMP NZ, 11F[read_mac_CRC]           ;AUTHENTICATION FAILED, so jump ahead
11D 01001      LOAD s0, 01[friend]                ;AUTHENTICATION PASSED, so set the friend bit
11E 2D007      OUTPUT s0, 07[auth_result_out_port] ;before falling into read_mac_CRC
11F          ;
11F          ;Next two bytes received are the 16-bit CRC
11F          ;Read 16-bit CRC into [s5,s4] and send value to UART
11F          ;
11F 2022C      read_mac_CRC: CALL 22C[read_send_test_CRC16] ;read, display and test CRC value
120          ;
120          ;Read one byte that should be value AA hex.
120          ; Would actually read AA hex continuously until master reset
120          ;
120 20279      CALL 279[read_byte_slow]              ;read data into s3
121 00030      LOAD s0, s3                        ;display byte
122          ;
122 20047      CALL 047[reset_before_next_cmd]
123 2004A      CALL 04A[read_ROM_command]           ;read ROM command and display result
  
```

9. Set this as the address to trigger on in ChipScope



10. This step is a bit complicated, but necessarily so because of the way this design operates in the FPGA.
- Hold your finger on the reset switch (**EF4**) on the S6LX16 board
 - Configure the FPGA in ChipScope. Click on OK in the following window to verify the configuration setup and program the FPGA.

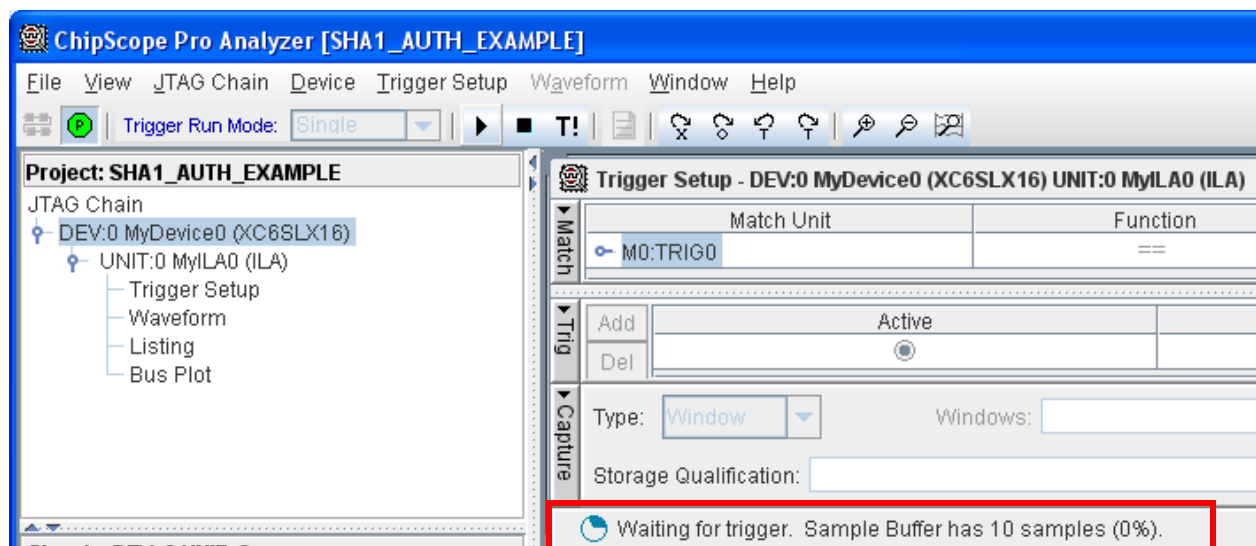


- Arm the ChipScope trigger



- Release the reset button

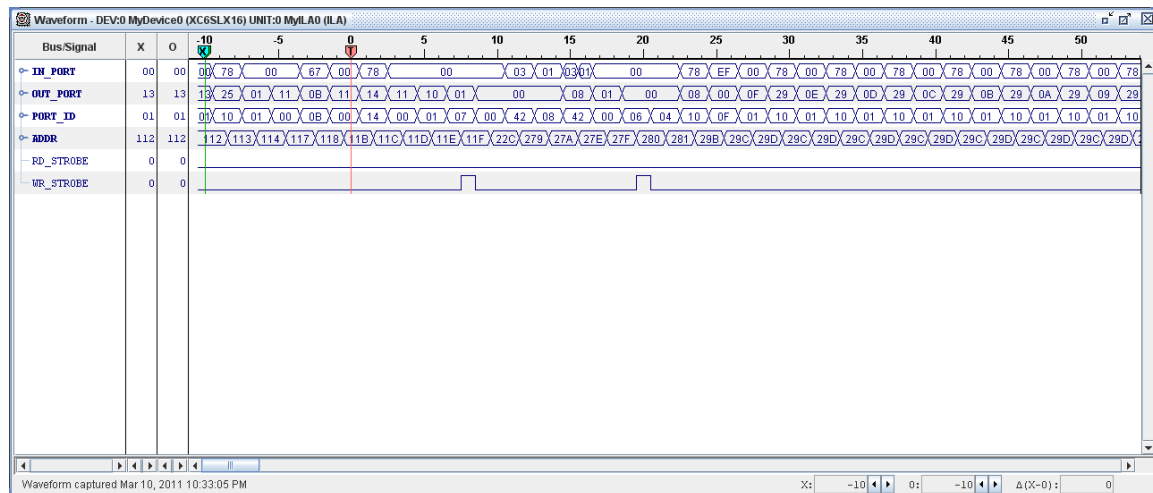
11. The design will now run and the trigger is armed and waiting to be tripped. This should only take a couple of seconds.



12. Zoom in on the waveform window to the trigger point.



13. Examine the address in the waveform and follow how it tracks in the PicoBlaze log file. This way you can step through your code and verify it is working correctly and debug where things are broken.



IMPORTANT! Remember to remove these (using comments) ChipScope ILA and ICON cores from the design when you are satisfied with your design changes and your debugging is complete. It is a significant security risk, and waste of FPGA resources, to leave them in the design.

ChipScope is a powerful and very useful tool with more features and capabilities than are described here. You are encouraged to read the [ChipScope User's Guide](#) for more information about how to use this tool in your designs.

This concludes this reference design tutorial.

Reference Material

Avnet Spartan-6 LX16 Evaluation Board

www.em.avnet.com/spartan6lx16-evl

Maxim DS28E01 PMOD Product Page

www.maxim-ic.com/ds28e01pmod

Maxim DS28E01 Product Page

www.maxim-ic.com/fpga

Maxim DS28E01 Technical Documents

www.maxim-ic.com/datasheet/index.mvp/id/4766/t/do

Xilinx® FPGA IFF Copy Protection with 1-Wire SHA-1 Secure Memories

www.maxim-ic.com/app-notes/index.mvp/id/3826

Protect Your FPGA Against Piracy: Cost-Effective Authentication Scheme Protects IP in SRAM-Based FPGA Designs

www.maxim-ic.com/app-notes/index.mvp/id/4594

Maxim DS28E01 Data Sheet

datasheets.maxim-ic.com/en/ds/DS28E01-100.pdf

Xilinx FPGA Design Security Solutions

www.xilinx.com/products/design_resources/security/index.htm

Xilinx FPGA IFF Copy Protection Using Maxim DS2432 Secure EEPROMs

www.xilinx.com/support/documentation/application_notes/xapp780.pdf

Xilinx PicoBlaze User Resources

www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm

Xilinx UG129 PicoBlaze User Guide

www.xilinx.com/support/documentation/ip_documentation/ug129.pdf

Xilinx ChipScope User Guide

www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/chipscope_pro_sw_cores_ug029.pdf

Revision History

Version	Date	Author	Details
1.0	08/22/2010	TC	ISE 12.1
1.1	09/10/2010	TC	Added more to overview. Draft review edits.
2.0	03/09/2011	TC	ISE12.4. Added scrambler and ChipScope.
2.1	04/19/2011	TC	Review edits.
3.0	01/09/2013	TC	ISE 14.3 updates