Device Serialization for ProASIC^PLUS® Devices

Introduction

This application note describes the creation of a design which incorporates a device-unique value that can be used as a serialization ID or encryption key, and inserted into the design programming file during production programming. For each device, single or multiple keys of any bit length can be implemented, and these keys are accessible to the logic inside the device. These keys can be used for encryption algorithms like AES and TDES, and for design revision and serial number identification. The ability to securely create unique device serialization can safeguard against counterfeit devices and uses. ProASIC^PLUS device serialization, combined with advanced security features such as FlashLock™ or Permanent FlashLock, enables very security-conscious designs.

This example includes a 32-bit device serialization application using the ProASIC^PLUS evaluation board. Download the Libero® Integrated Design Environment (IDE) project design files from http://www.actel.com/documents/DeviceSerialization.zip.

Required Software and Hardware for this Demo Project

- ActivePerl (http://www.activestate.com)
- ProASIC^PLUS Starter Kit (Actel Part Number APA-EVAL-KIT)

ProASIC^PLUS Family

Unlike SRAM-based FPGAs that require external configuration devices, Actel Flash-based ProASIC^PLUS devices are nonvolatile, single-chip FPGAs that are live at power-up, and have the highest security standard in the market. Design information is stored in nonvolatile memory cells, which are effectively small capacitors. Any physical deconstruction of the device will disrupt the programmed data. With this advantage, consumer electronics, automotive applications, and industrial products that require internal serial number identification, encryption, and/or decryption can safely store this information within the device. This will enhance product communication, enable subscription based services, ease upgrades, and facilitate maintenance over the life span of the product.

ProASIC^PLUS Evaluation Board Overview

The ProASIC^PLUS evaluation board has on-board voltage regulation, enabling I/O voltages (V_{DDP}) to be set to either 2.5 V or 3.3 V. The system clock can be generated using the on-board oscillator and ProASIC^PLUS PLLs. Eight LEDs and four switches provide simple inputs and outputs to the system. Prototyping headers connect to all the ProASIC^PLUS device I/Os, enabling components to be easily added to the evaluation board. Finally, the board is equipped with programming headers to support ISP programming using FlashPro, FlashPro Lite, or Silicon Sculptor II. Actel supplies these evaluation boards with an APA075-PQ208 or APA300-PQ208 soldered on, or as a socketed board without a ProASIC^PLUS chip. Either an APA075 or an APA300 device can be used for the example design.

Implementation Description

Overview

In order to store the unique ID or key information in a ProASIC^PLUS device, a key/serialization HDL logic core must be instantiated in the design. In the serialization core, the logic will typically be buffer macros, "BFR", or multiplexers, "MUX2H". These macros are used to store the actual data of the key information.

The normal design flow is followed to generate a programming file (STAPL file). This STAPL file will have the default key value stored in the design. In order to change the default value stored in the STAPL file, a GCF file which contains the physical location of each stored bit on the device is exported. With the original STAPL file and exported GCF files, a Perl script is used to create a new template STAPL file that contains a variable field for each key value (Figure 1).

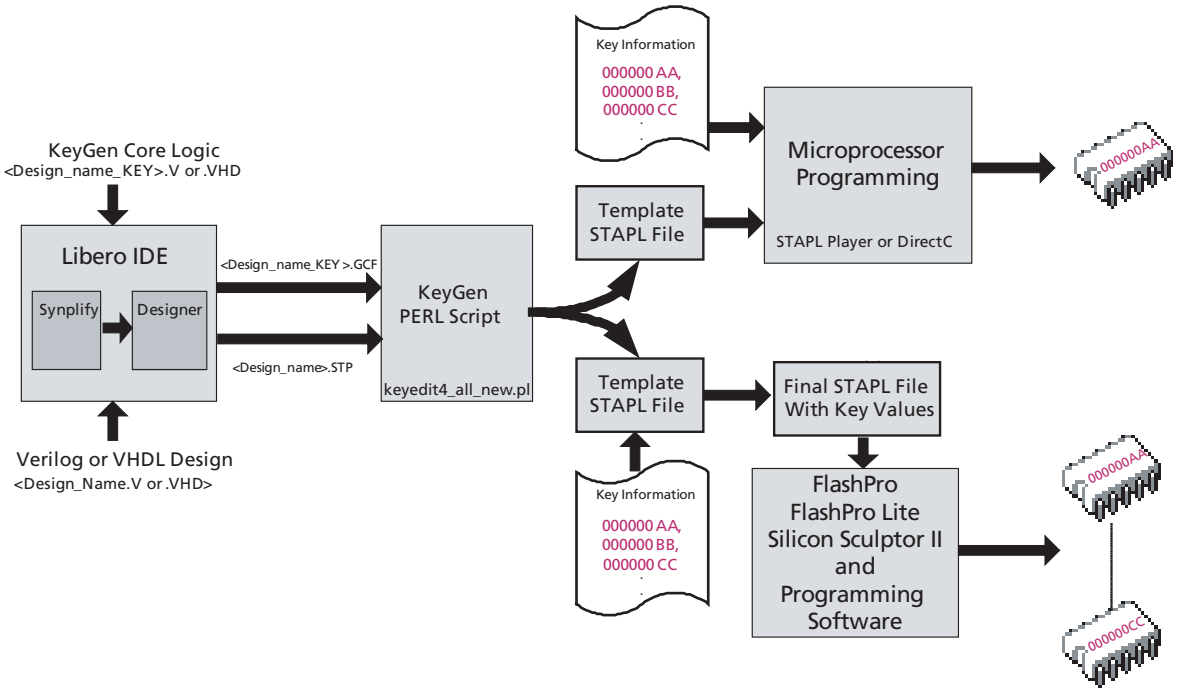

Figure 1 • **Overview of the Device Serialization Design Flow**

Step 1 – Design Creation

Create your design using your regular design flow and connect the output of the KeyCore ("eeprom.vhd – Keycore" section on page 4) to the design. In this reference design, the output of the KeyCore is connected to registers for an LED display. Specific naming conventions must be followed in the design entry so that the Perl script can identify the KeyCore and process it accordingly. The "UniqueKey.vhd", "eeprom.vhd – Keycore", and "GCF Constraint File" sections show the corresponding design files; Figure 2 on page 5 shows the RTL view of this sample design.

For the KeyCore, the instance name must begin with the keyword "DECRYPT", and the name of the BFR instance within the KeyCore must begin with "b", followed by numbers.

Normally, the compile stage in Designer removes all redundant buffers and inputs connected to ground. To instruct the place-and-route tool to not remove KeyCore from the design, create a ProASIC Constraint File (GCF) entry with the dont_touch constraint. The wildcard character (*) can be used to preserve all sub-blocks under the named block. If this constraint is used, any instances (including buffers and inverters connected to global nets, promoted global nets, and spine nets) stay intact. For this reference design, I/O placement constraints are also included to assign the pinout to the designated location of the ProASIC^PLUS Evaluation Board.

Syntax: `dont_touch hier_instance_name [, hier_instance_name … ];`

Example: `dont_touch DECRYPT_ONE/*;`

For this demo design, Aclr is assigned to pin 55 using the GCF constraint, which is connected to the mechanical switch, SW1, on the evaluation board. When SW1 is closed (logic '1'), the output of the registers is cleared and thus all the LEDs are turned off. When SW1 is opened (logic '0'), the lower 8 bits of the KeyCore are displayed on the LEDs of the evaluation board, with b0 at DS8, b1 at DS7, and so on, with b8 at DS0. Though this reference design uses a 32-bit KeyCore, the defined KeyCore can be any length, depending upon application requirements.

Syntax: `set_io "pin_number" "netName/portName";`

Example: `set_io "55" "Aclr";`

UniqueKey.vhd

```
UniqueKey.vhd
LIBRARY IEEE;
USE ieee.std_logic_1164.all;

ENTITY  UniqueKey IS
    PORT (
           Clk         :IN STD_LOGIC;
           Aclr        :IN STD_LOGIC;
           Output      :OUT std_logic_vector(31 DOWNTO 0));

END UniqueKey;

ARCHITECTURE Behavior OF UniqueKey IS

    COMPONENT eeprom
        PORT (data : OUT std_logic_vector(31 DOWNTO 0));
    END COMPONENT;

SIGNAL reg_data : std_logic_vector(31 DOWNTO 0);
SIGNAL output_data : std_logic_vector(31 DOWNTO 0);

BEGIN

    DECRYPT_ONE: eeprom PORT MAP(data => reg_data);

    PROCESS(Clk, Aclr)
    BEGIN
        IF Aclr = '1' THEN
            Output_data <= (OTHERS => '0');
        ELSIF Clk'EVENT AND Clk = '1' THEN
            output_data <= reg_data;
        END IF;
    END PROCESS;
   Output <= output_data;
END Behavior;
```

eeprom.vhd – Keycore

```
-- eeprom.vhd
LIBRARY IEEE;
USE ieee.std_logic_1164.all;

ENTITY eeprom IS
      PORT  (
         data : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END eeprom;

ARCHITECTURE behavior OF eeprom IS
    COMPONENT BFR
        PORT(A : in std_logic;
         Y : out std_logic);
     END COMPONENT;

BEGIN
    b0  : BFR PORT MAP (A   => '0',Y  => data(0));
    b1  : BFR PORT MAP (A   => '0',Y  => data(1));
    b2  : BFR PORT MAP (A   => '0',Y  => data(2));
    b3  : BFR PORT MAP (A   => '0',Y  => data(3));
    …
    …
    …
    b30 : BFR PORT MAP (A   => '0',Y  => data(30));
    b31 : BFR PORT MAP (A   => '0',Y  => data(31));
END behavior;
```

GCF Constraint File

```
dont_touch DECRYPT_ONE/*;
//
// I/O constraints

//Output to LED DS8 to DS1
set_io "96" "Output(0)";
set_io "95" "Output(1)";
set_io "94" "Output(2)";
set_io "93" "Output(3)";
set_io "92" "Output(4)";
set_io "91" "Output(5)";
set_io "90" "Output(6)";
```

```
set_io "87" "Output(7)";

//Other I/O constraints. Refer to the design file
…
…

// SW1 for the ACLR
set_io "55" "Aclr";
set_io "24" "Clk";
```

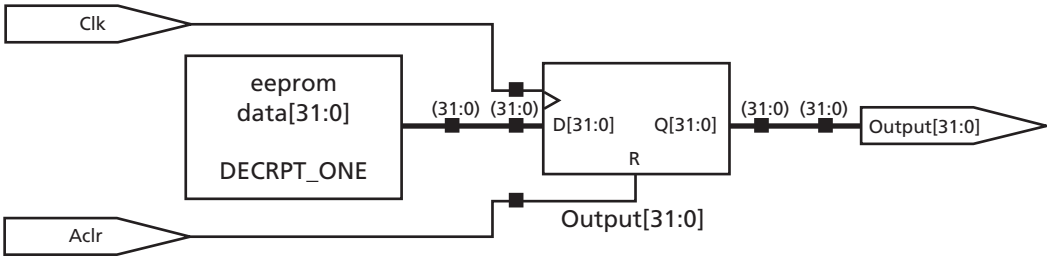

Figure 2 • **RTL View of UniqueKey.vhd**

Step 2 – Synthesis and Simulation

There are no changes to either synthesis or simulation. The default value of the KeyCore may be changed to perform the simulation.

Step 3 – Designer Place-and-Route

Import the ProASIC constraint file (GCF file) that was created in step 1 into Designer. As mentioned in the previous section, this constraint file must be included as a source input file to preserve the KeyCore logic as well as I/O pin assignment.

Complete the Compile, Layout, and Timing Analysis steps to generate the programming file in STAPL format, as BIT format is not supported for device serialization applications. Next, export the constraint file that includes the KeyCore buffer location.

In this example, the location of the buffers from the KeyCore is assigned by the place-and-route tools. You can manually place the buffers if needed, either by using ChipPlanner or by adding a set_location GCF constraint.

To export the constraint file that contains the KeyCore buffer location, go to **File** > **Export** > **Constraint Files** (Figure 3 on page 6). Open up the exported constraint file using any text editor and replace all occurrences of set_initial_location with set_location.

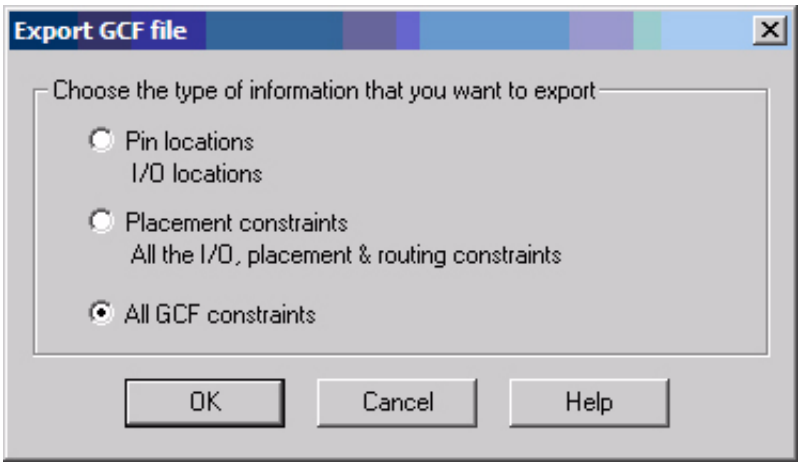

Figure 3 • **Export GCF File**

Step 4 – Run the script to generate the template STAPL file

The original STAPL and GCF constraint files exported from Designer are the input files for the KeyGen Perl script (keyedit4_all_new.pl). The KeyGen Perl script uses the input files to generate the template STAPL file, which contains a key variable field to store user defined key values.

In order to run the Perl script, a Perl script interpreter is required. This can be downloaded from various open source websites. The Perl script interpreter used in this demo design is ActivePerl (standard distribution) and it can be downloaded from http://www.activestate.com/Products/ActivePerl/.

Syntax: `keyedit4_all_new.pl <gcf> <original STAPL> <new STAPL>`

Example: `keyedit4_all_new.pl UniqueKey.gcf UniqueKey.stp Final.stp`

Opening the newly generated template STAPL file using a text editor will show additional instructions and variable fields that are added for the purpose of this application. The key variable field in this STAPL file is named DECRYPT_ONE, with a default value set to zero.

BOOLEAN DECRYPT_ONE[32]= $00000000;

For this example, change the default value as follows:

BOOLEAN DECRYPT_ONE[32]= $000000AA;

At the end of the STAPL file, the CRC value must be changed to 0 in order to disable the CRC check. Note that the default value for the key is always set to zero and the 32-bit key value is represented in hex. By modifying the key variable in the STAPL file, a unique identifier can be set for each device. Scripts and/or software applications can be used to dynamically modify the key value, enabling custom key generation. In microprocessor applications, the STAPL file should be loaded into memory so that the key value can be changed dynamically according to the application needs.

Step 5 – Program the design onto the device

The STAPL file containing the specified key value is programmed into the ProASIC^PLUS device using standard methodologies. This example uses FlashPro Programming software. Refer to the *FlashPro User's Guide* for more information.

After successfully programming the device, the lower eight bits of the key value will be displayed on the LEDs (DS1 to DS8). Ensure the JP4 is installed for the clock source of the design. Any text editor can be used to modify the key value variable in the STAPL file in this example. Reprogramming the device will illustrate these changes.

Conclusion

The Actel ProASIC^PLUS family has been built on a secure flash fabric, and it offers many advanced security features including device serialization, FlashLock, and Permanent Lock. When used in combination, these offer a very robust, secure, programmable solution. This application note has detailed how to implement device serialization programming techniques enabling each device to have unique key values stored within it. This can be done without modifying the existing layout of the design. The key values can be of any length and are accessible to the internal logic. Any application that requires unique key information can take advantage of this feature.

Appendix

File Name	Description
UniqueKey.vhd	This VHDL file contains a simple example that displays the lower 8 bits of a 32-bit key on the LEDs of the ProASIC^PLUS evaluation board.
eeprom.vhd	This VHDL file contains the logic that is used to store the key information. This KeyCore is implemented using a buffer macro, BFR, with its input tied to GND, which gives the initial value of the KeyCore to zero.
UniqueKey_IO.gcf	Pin assignment and don't_touch ProASIC constraint file
UniqueKey.gcf	ProASIC constraint file exported from Designer that details the key core buffer location
UniqueKey.stp	Original STAPL file
Keyedit4_all_new.pl	Perl script used to create a template STAPL file that can be used for key entry Syntax (Case Sensitive): keyedit4_all_new.pl <gcf> <original STAPL> <new STAPL>
UniqueKey_final.stp	Final template STAPL file which contains the algorithm and variable field for key entry

Related Documents

Datasheets

ProASIC^PLUS Flash Family FPGAs

http://www.actel.com/documents/ProASICPlus_DS.pdf

User's Guides

ProASIC^PLUS Starter Kit User's Guide and Tutorial

http://www.actel.com/documents/PAstartkit_UG.pdf

Designer User's Guide

http://www.actel.com/documents/designer_UG.pdf

FlashPro User's Guide

http://www.actel.com/documents/flashpro_UG.pdf

www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan
www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488

51900108-0/8.05