XAPP1283 (v1.2.1) May 21, 2025

Internal Programming of BBRAM and eFUSEs

Author: Ed Peterson

Summary

AMD UltraScale™ and AMD UltraScale+™ FPGAs feature a built-in primitive named MASTER_JTAG that provides the same functionality as the external JTAG port. The internal FPGA logic can use MASTER_JTAG to load the AES decryption key into the battery-backed RAM (BBRAM). It can also use the MASTER_JTAG primitive to program various one-time programmable (OTP) eFUSEs from within the device, instead of using the external JTAG port. This application note abstracts the low-level commands needed to communicate with the MASTER_JTAG primitive by providing the necessary AMD supplied HDL and C library functions for a MicroBlaze™-based Soft Processor Core [Ref 1] design. These designs can be modified and enhanced for various use cases.

You can download the reference design files for this application note from the AMD website. For detailed information about the design files, see Reference Design.

Note: This application note does not apply to AMD Spartan™ UltraScale+™ FPGAs.

Introduction

AMD devices have featured both BBRAM and eFUSE non-volatile (NVM) memory since AMD Virtex™ 6 FPGAs. The primary use of a BBRAM in UltraScale and UltraScale+ devices is to store the 256-bit AES bitstream decryption key. This AES key can also be stored in the eFUSEs with the important difference that it can neither be modified nor erased. A 32-bit FUSE_USER register that can be programmed for custom use cases is also available.

UltraScale and UltraScale+ FPGAs have introduced an additional programmable 128-bit register, FUSE_USER_128. It is used as a maintenance log, a tamper activity log, or for a custom use case. A 384-bit eFUSE register that can be programmed with the hash of a RSA-2048 public key as a part of the device's authentication scheme is also available. The eFUSE-based control bits including read/write disables, decryption disable, external JTAG disable, and RSA enable can also be programmed. Refer to the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2] and *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3] for more information.

Note: For larger UltraScale and UltraScale+ FPGAs that use stacked silicon interconnect (SSI) technology there are some differences in procedure from monolithic devices which will be described in this application note. Additionally, see the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2] for a list of UltraScale and UltraScale+ FPGA devices that employ SSI technology.

Note: This application note also applies to eFUSEs located in the programmable logic (PL) area of AMD Zynq™ UltraScale+™ MPSoC/RFSoC devices. Only the PL FUSE_USER_128 and FUSE_USER eFUSEs are supported in the application note. For details on programming the eFUSEs within the processor system (PS) section of Zynq UltraScale+ devices, see (XAPP1319) [Ref 22].

Traditionally, BBRAM and eFUSEs are programmed through the external JTAG port using the AMD ISE iMPACT or AMD Vivado™ Hardware Manager. Starting with UltraScale and UltraScale+ FPGAs, BBRAM and eFUSEs can also be programmed using the internal MASTER_JTAG port.

The ability to program using an internal path has a number of advantages in security applications. For instance, it is possible to perform a field update of the BBRAM key remotely and securely by instantiating a secure key exchange function in the FPGA logic. The new key can travel safely over an unsecured network because it is encrypted (black), decrypted in the FPGA logic (red), and then stored in the BBRAM from within the device. Other security use cases are internal programming of eFUSEs such as tamper log information, AES decryptor disable, and JTAG disable as responses to a tamper event. Refer to the *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs Application Note* (XAPP1098) [Ref 7].

A MicroBlaze Soft Processor Core [Ref 1] is instantiated in the FPGA logic to interface with the internal MASTER_JTAG port. The AMD-supplied XILSKEY library provides an application programming interface (API) to simplify and abstract the necessary low-level JTAG commands. The hardware and software development required to use the XILSKEY library is outlined in Hardware and Software Development Overview. Refer to the *Standalone Library Documentation: BSP and Libraries Document Collection* [Ref 8] for more information about the XILSKEY library.

Specifically, one Vivado hardware project and three different SDK/Vitis applications are created from the design flow described in this application note. These SDK/Vitis applications perform the following tasks:

- Program the BBRAM AES key internally.
- Create a Hello World design with an encrypted bitstream to verify the BBRAM key.
- Program eFUSE bits internally. The internal eFUSE programming is verified by the SDK application as well as by the Vivado Hardware Manager.

This application note assumes that you are familiar with the Vivado tools flow methodology (including IP integrator), and are experienced with building simple SDK and/or Vitis projects. See *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3], *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 4], *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 5], *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 6], and *Vitis Unified Software Platform* [Ref 23] for more information.

Hardware and Software Requirements

The hardware and software requirements for the internal programming of BBRAM and eFUSEs memory are as follows.

For UltraScale FPGAs:

- KCU105 AMD Kintex™ UltraScale FPGA evaluation kit [Ref 9] based on the XCKU040-2FFVA1156E FPGA (if using a different evaluation kit or a custom board, the specific steps outlined in this application note will somewhat differ)
- AC to DC power adapter (12 VDC)
- USB type-A to micro-B USB cable for JTAG
- USB type-A to micro-B USB cable for UART
- AMD Vivado design suite version 2016.3 or later [Ref 10]
- Xilinx Software Development Kit (XSDK) version 2016.3 or later for non-SSIT UltraScale FPGA devices. Vitis version 2019.2 or later is required for SSIT UltraScale FPGA devices [Ref 11] [Ref 23]
- Serial communications terminal application such as Tera Term [Ref 12]

For UltraScale+ FPGAs:

- KCU116 Kintex UltraScale+ FPGA evaluation kit [Ref 13] based on the XCKU5P-2FFVB676E FPGA (if using a different evaluation kit or a custom board the specific steps outlined in this application note will somewhat differ)
- AC to DC power adapter (12 VDC)
- USB type-A to micro-B USB cable for JTAG
- USB type-A to micro-B USB cable for UART
- AMD Vivado design suite version 2017.3 or later for non-SSIT UltraScale+ FPGA devices. Version 2019.2 or later is required for SSIT UltraScale+ FPGA device and for Zynq UltraScale+ devices (PL only) [Ref 10].
- Xilinx Software Development Kit (XSDK) version 2017.3 or later for non-SSIT UltraScale+ FPGA devices. Vitis unified software platform version 2019.2 or later is required for SSIT UltraScale+ FPGA devices and for Zynq UltraScale+ devices (PL only) [Ref 11] [Ref 23].
- Serial communications terminal application such as Tera Term [Ref 12]

IMPORTANT: *The BBRAM key cannot be used if the EFUSE_KEY_ONLY control eFUSE is already programmed on the FPGA. Refer to the UltraScale Architecture Configuration User Guide* (UG570) *[Ref 2] for more information.*

Special Considerations When Using MASTER_JTAG to Internally Program eFUSEs

Certain special considerations must be taken to ensure reliability when MASTER_JTAG is used to internally program eFUSEs.

1. The configuration memory must not be accessed during internal programming. This includes activities such as a single-event upset (SEU) scan, readback capture, and partial reconfiguration (PR). For example, the AMD Soft Error Mitigation (SEM) IP core [Ref 14] should not be operational during eFUSE programming.
2. The board/system design must take into consideration the additional current required for programming eFUSEs. The programming current is available in the appropriate UltraScale device data sheets [Ref 17].

Hardware and Software Development Overview

This application note outlines the steps to create a MicroBlaze processor-based Vivado tools design that indirectly interfaces the MASTER_JTAG primitive using the general purpose I/O (GPIO). A Vivado hardware project is created and exported to SDK/Vitis IDE. The SDK/Vitis tool is launched with the MicroBlaze processor-based Vivado tools design as its hardware platform. Three different software applications based on the available software libraries and example code are then created. After these SDK/Vitis applications are compiled and built into executable and linkable format (ELF) files, they are brought back into the Vivado tools. Three separate FPGA bitstreams are then generated to perform and verify the functions described in this application note.

In addition to the MicroBlaze processor, two VHDL files are included in this project (see Reference Design for more information):

- `int_prog_ctl_us.vhd`
- `jtag_monitor.vhd`

INT_PROG_CTL_US Module

Most of the time, the JTAG signaling can come directly from the MicroBlaze processor's GPIO interface. However, to provide precise control over the duration of the actual eFUSE programming pulse, the INT_PROG_CTL_US module is required. This module instantiates the MASTER_JTAG primitive.

Note: For FPGAs that are SSIT devices, there will be multiple MASTER_JTAG instantiations - one for each SLR. Instructions with examples will be provided in the comments section of the `int_prog_ctl.vhd` file.

The INT_PROG_CTL_US module passes the MicroBlaze processor's GPIO signals directly to and from the MASTER_JTAG primitive, except when it enters the actual eFUSE programming phase. Upon entering this phase, the XILSKEY software polls for the INT_PROG_CTL_US module using a READY signal. After the software detects that the module is READY, it asserts the START_PROG input to command the module to take control of the JTAG signaling (TCK and TMS) to get a precise 5 µs programming pulse. When the eFUSE programming is complete, the

INT_PROG_CTL_US module asserts the END_PROG signal and allows the software to regain control of the JTAG signals. This interface is illustrated in Figure 1.

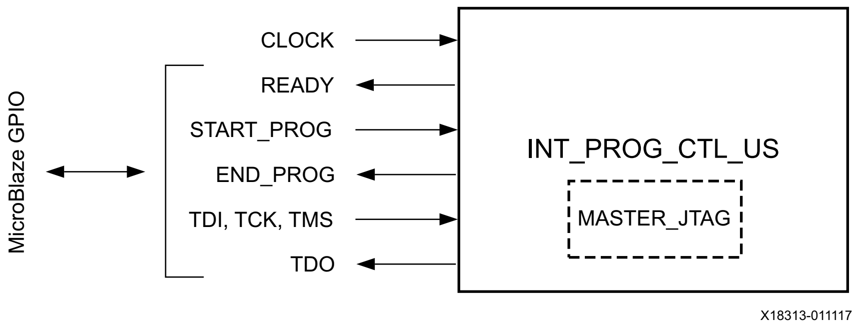


Figure 1: **MicroBlaze Interface to MASTER_JTAG using INT_PROG_CTL_US Module**

If you want to internally program just the BBRAM and not the eFUSE, then you can ignore this module. AMD recommends including this module because it consumes few resources and allows future internal eFUSE programming. However, if you decide to ignore it, then you must instantiate the MASTER_JTAG primitive and connect it directly to the MicroBlaze processor's GPIO.

If your design is also instantiating the AMD Security Monitor IP Core (see the *Security Monitor IP Core Product Brief* [Ref 16]), the INT_PROG_CTL_US module must be modified, because the Security Monitor instantiates the MASTER_JTAG primitive. The MASTER_JTAG primitive must be removed from the INT_PROG_CTL_US module and the corresponding JTAG signals brought up to the module's port list to connect to the Security Monitor's MASTER_JTAG interface.

JTAG_MONITOR Module

The optional JTAG_MONITOR module provides an EFUSE_PROG_PULSE output that can be routed up to an external header pin on the KCU105 board or KCU116 board to measure the eFUSE programming pulse width (this signal mirrors the amount of time the JTAG controller is in the programming phase).

The JTAG_MONITOR can also be used for debugging/monitoring signals and display the current JTAG state. When using the Vivado Integrated Logic Analyzer (ILA) IP core to monitor the JTAG states, the INT_PROG_CTL_US module must have its TEST_MODE generic set to TRUE (MASTER_JTAG is not instantiated) to allow the ILA to communicate using external JTAG. Otherwise, the ILA does not work, because external JTAG is disconnected when MASTER_JTAG is instantiated within a design. When TEST_MODE is set to TRUE it can only be used to debug the startup of the xilskey software eFUSE functionality. This is because in TEST_MODE the software will not be able to detect the MASTER_JTAG primitive. Therefore, it only has limited usefulness.

Note: In order to perform more comprehensive hardware debug (when MASTER_JTAG is instantiated), there are two options. The first option is to map MASTER_JTAG I/O to user I/O and make external JTAG connections using those user I/O. The second (and easier) option is to use the AMD LogiCORE™ IP Debug Bridge, which bridges user I/O JTAG to BSCAN peripherals such as ILA and MicroBlaze MDM. This bridge will allow for full debug without having to loop external signals (which is necessary for the first option). For either option, the TEST_MODE generic is set to FALSE.

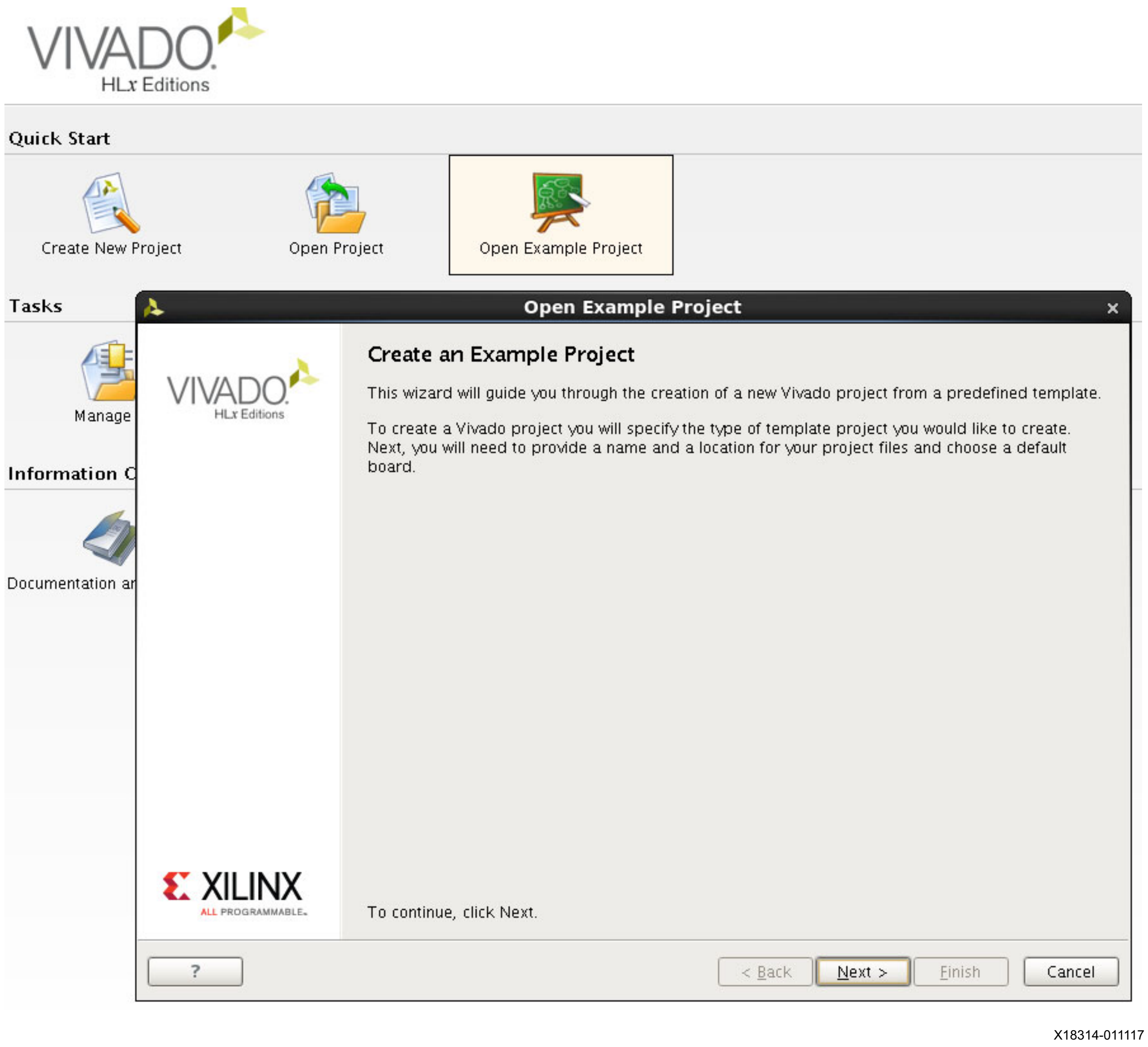
The JTAG_MONITOR module is useful for development and debugging. It is optional and can be removed in a field deployed system.

Building the Vivado Hardware Project

One common Vivado project is the hardware required for this application note. It is based upon a MicroBlaze soft processor using GPIO to indirectly interface with the MASTER_JTAG primitive through the INT_PROG_CTL_US Module. The following steps are used to create the Vivado hardware project.

Note: Some of the figures can vary slightly from the actual tool. This depends on which version of the Vivado tools are being used.

1. Open the Vivado tools (version 2016.3 or later for UltraScale FPGAs and version 2017.3 or later for UltraScale+ FPGAs) in graphical user interface (GUI) mode.
2. From the welcome screen under Quick Start, select **Open Example Project** (Figure 2).



Create an Example Project

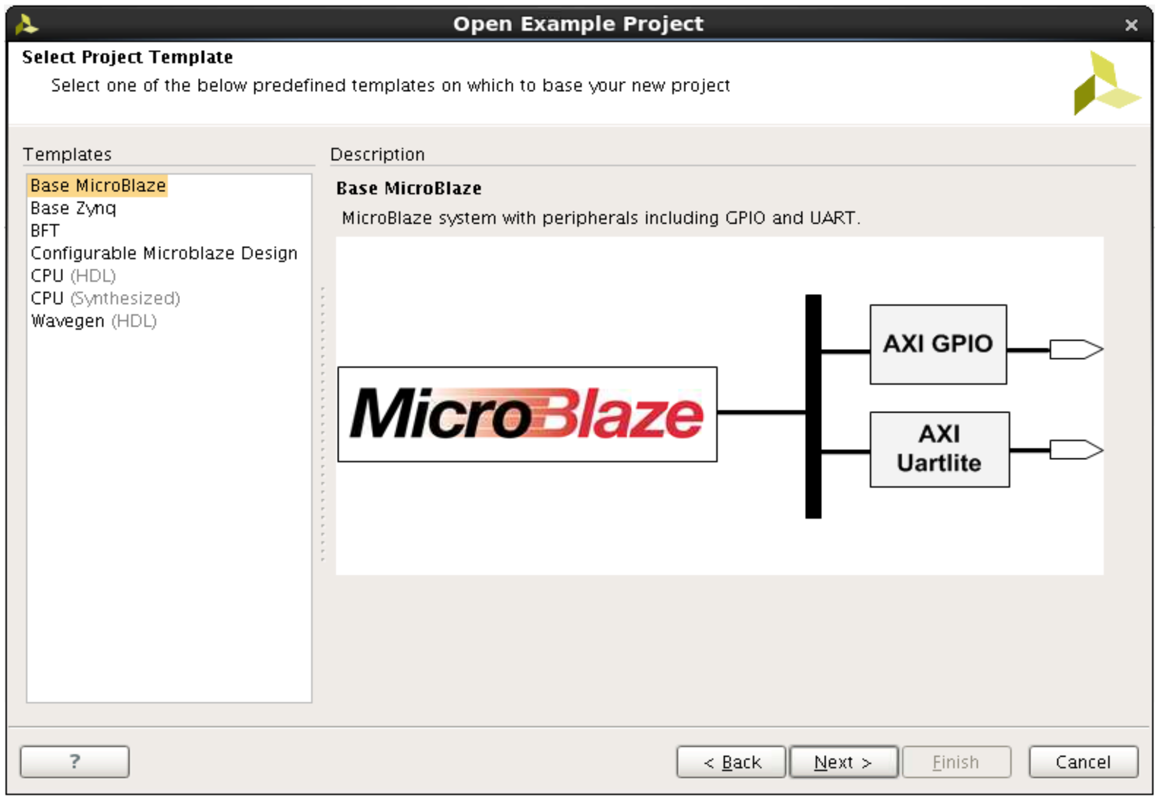
This wizard will guide you through the creation of a new Vivado project from a predefined template.

To create a Vivado project you will specify the type of template project you would like to create. Next, you will need to provide a name and a location for your project files and choose a default board.

To continue, click Next.

Figure 2: **Vivado Example Project**

3. Click **Next**, select **Base MicroBlaze**, and click **Next** (Figure 3).



Open Example Project

Select Project Template

Select one of the below predefined templates on which to base your new project

Templates

- Base MicroBlaze
- Base Zynq
- BFT
- Configurable Microblaze Design
- CPU (HDL)
- CPU (Synthesized)
- Wavegen (HDL)

Description

Base MicroBlaze

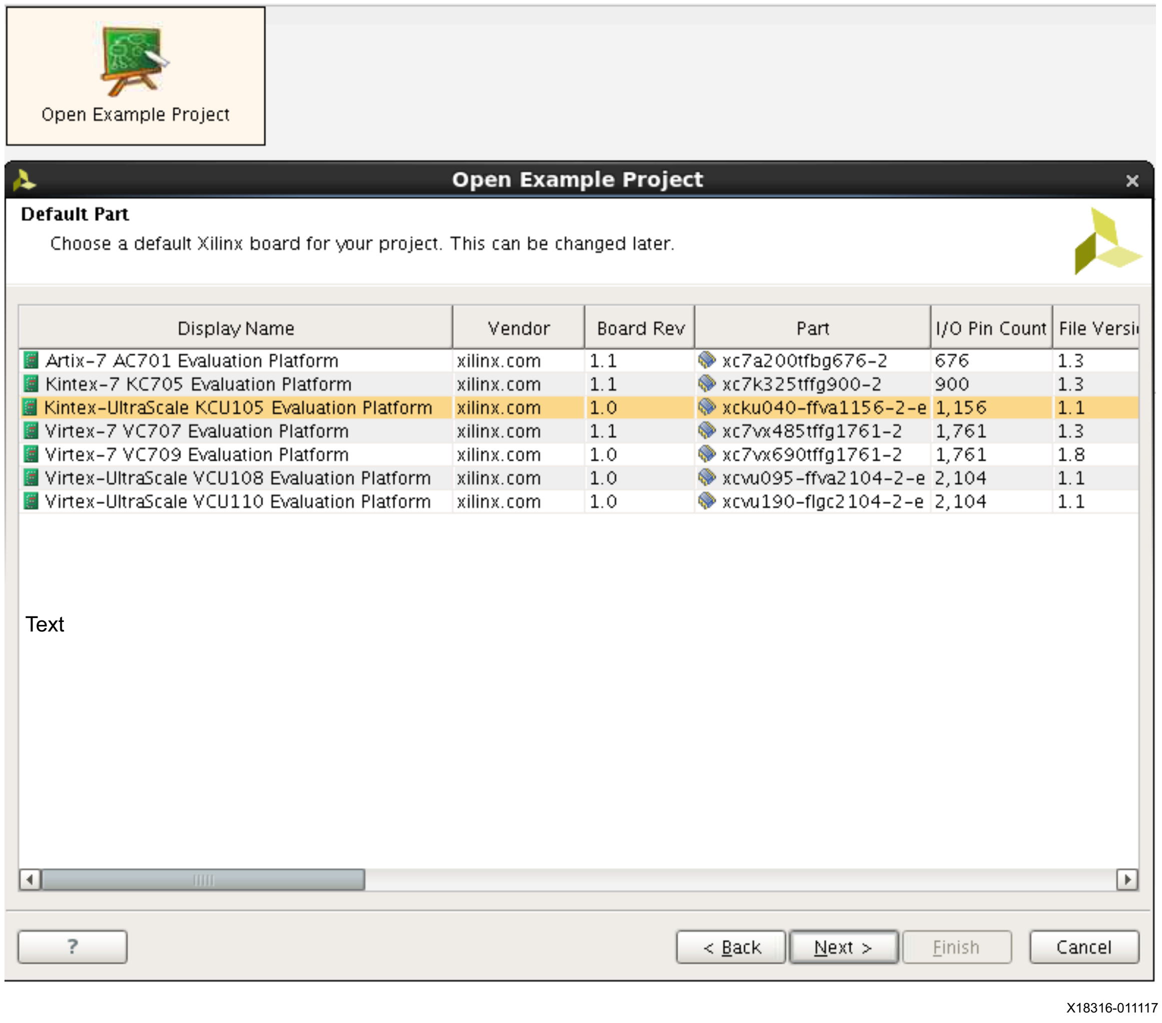
MicroBlaze system with peripherals including GPIO and UART.

Figure 3: **Project Template Base MicroBlaze**

4. Enter the project name: **internal_prog_proj**.
5. Enter the appropriate project location and (optionally) keep the create project subdirectory checkbox selected.

   Note: The maximum allowed path length on Windows is 260 bytes.

6. Click **Next**, select **Kintex-UltraScale KCU105 Evaluation Platform** or **Kintex UltraScale+ KCU116 Evaluation Platform**, and click **Next** (Figure 4).



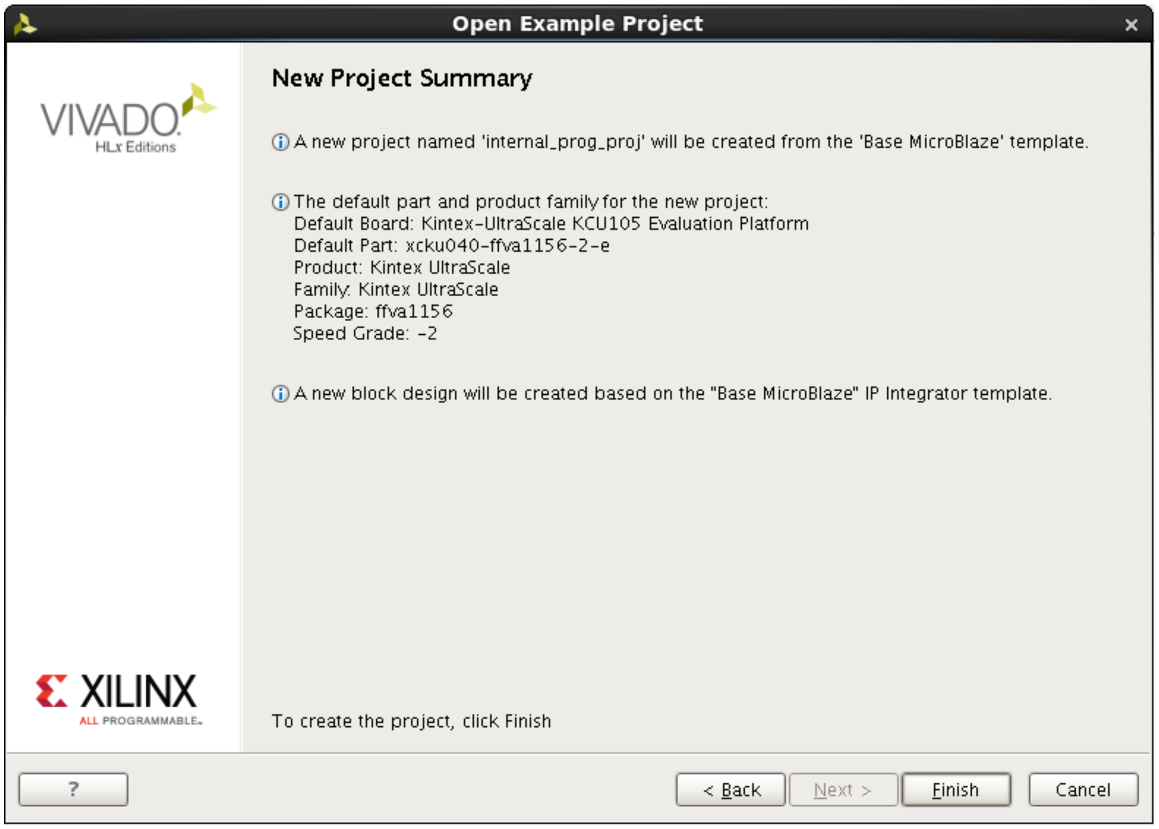
Default Part

Choose a default Xilinx board for your project. This can be changed later.

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Versi
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.3
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.3
Kintex-UltraScale KCU105 Evaluation Platform	xilinx.com	1.0	xcku040-ffva1156-2-e	1,156	1.1
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.3
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.8
Virtex-UltraScale VCU108 Evaluation Platform	xilinx.com	1.0	xcvu095-ffva2104-2-e	2,104	1.1
Virtex-UltraScale VCU110 Evaluation Platform	xilinx.com	1.0	xcvu190-flgc2104-2-e	2,104	1.1

Figure 4: **Project Evaluation Platform**

7. Click **Finish**. The Vivado tools create the base MicroBlaze processor project (Figure 5 and Figure 6).



Open Example Project

New Project Summary

A new project named 'internal_prog_proj' will be created from the 'Base MicroBlaze' template.

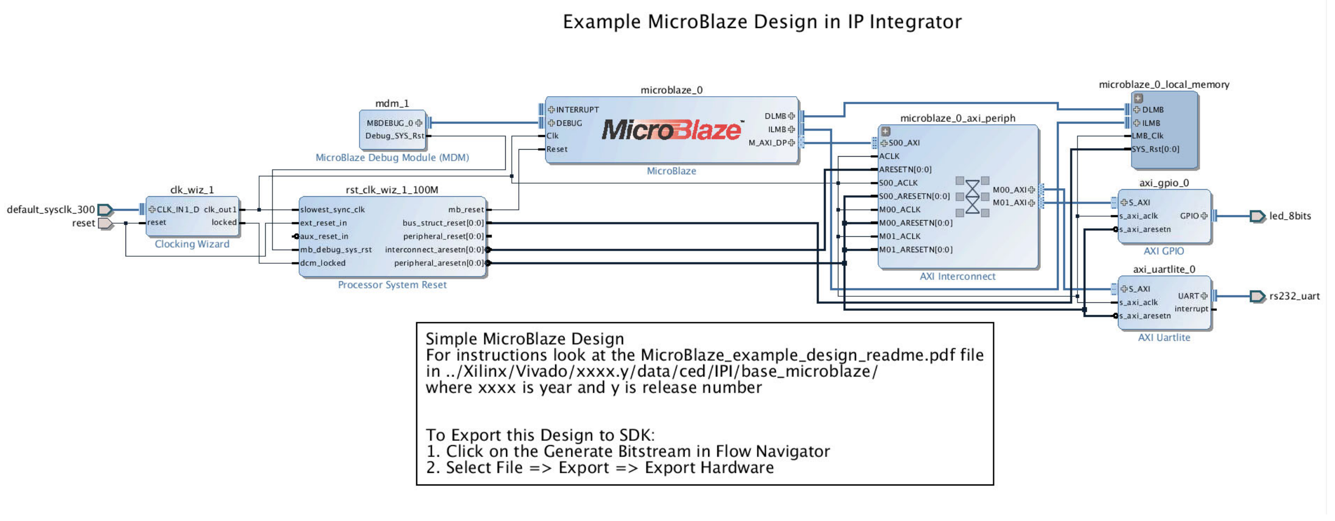
The default part and product family for the new project:
Default Board: Kintex-UltraScale KCU105 Evaluation Platform
Default Part: xcku040-ffva1156-2-e
Product: Kintex UltraScale
Family: Kintex UltraScale
Package: ffva1156
Speed Grade: -2

A new block design will be created based on the "Base MicroBlaze" IP Integrator template.

To create the project, click Finish

X18317-011117

Figure 5: **Project Summary**



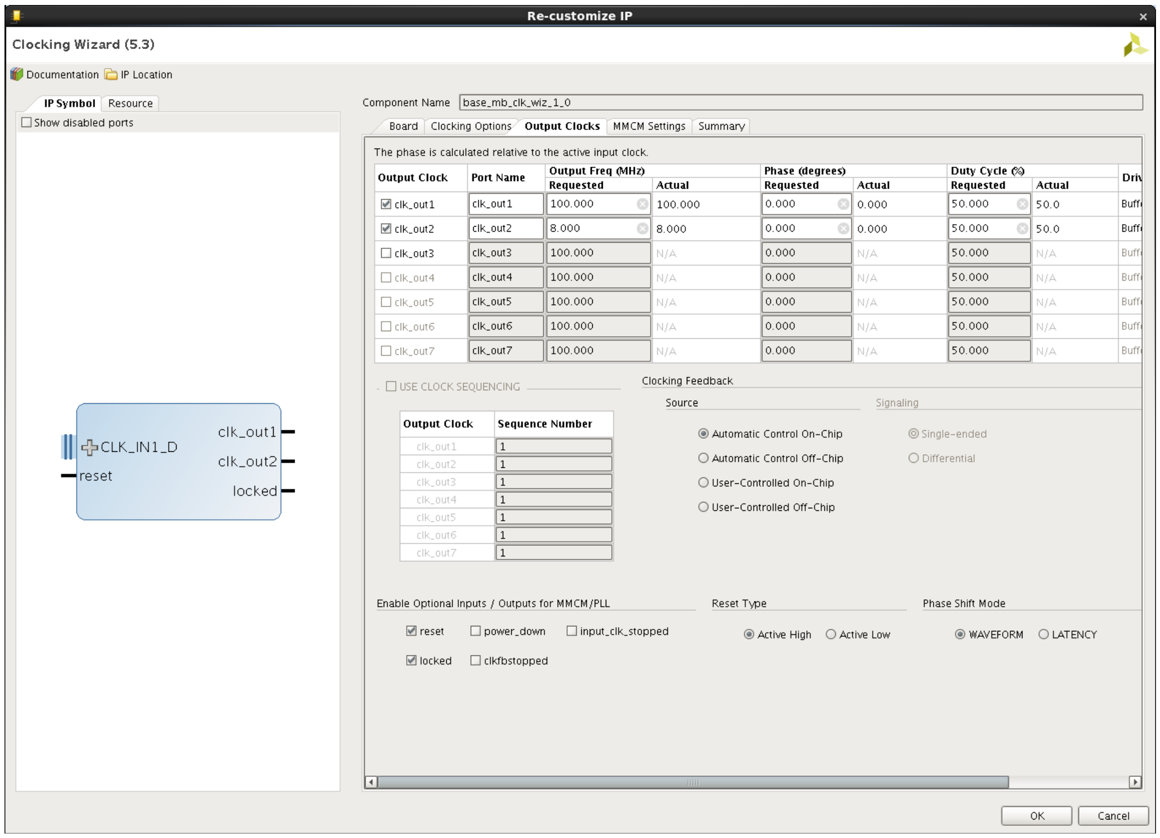
Simple MicroBlaze Design
For instructions look at the MicroBlaze_example_design_readme.pdf file
in ../Xilinx/Vivado/xxxx.y/data/ced/IPI/base_microblaze/
where xxxx is year and y is release number

To Export this Design to SDK:
1. Click on the Generate Bitstream in Flow Navigator
2. Select File => Export => Export Hardware

X18319-011117

Figure 6: **Example MicroBlaze Design in IP Integrator**

8. Double-click the **Clocking Wizard** IP block, select the **Output Clocks** tab, verify **clk_out1** is selected, and the requested and actual frequency is 100 MHz. Click **OK**. This is the main clock for the MicroBlaze core.
9. Check the box next to **clk_out2** under the Output Clock column and type in 8.000. Verify that the actual column reports back 8.000 MHz. This is the clock used by the INT_PROG_CTL_US Module to provide precise time during the actual programming of the eFUSEs. Click **OK** (Figure 7).



Component Name base_mb_clk_wiz_1_0

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Driv
		Requested	Actual	Requested	Actual	Requested	Actual	
✓ clk_out1	clk_out1	100.000	100.000	0.000	0.000	50.000	50.0	Buff
✓ clk_out2	clk_out2	8.000	8.000	0.000	0.000	50.000	50.0	Buff
☐ clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	Buff
☐ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	Buff
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	Buff
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	Buff
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	Buff

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip User-Controlled On-Chip User-Controlled Off-Chip

Signaling: [✓ Single-ended] Differential

Enable Optional Inputs / Outputs for MMCM/PLL: ✓ reset ☐ power_down ☐ input_clk_stopped ✓ locked ☐ clkfbstopped

Reset Type: [✓ Active High] Active Low

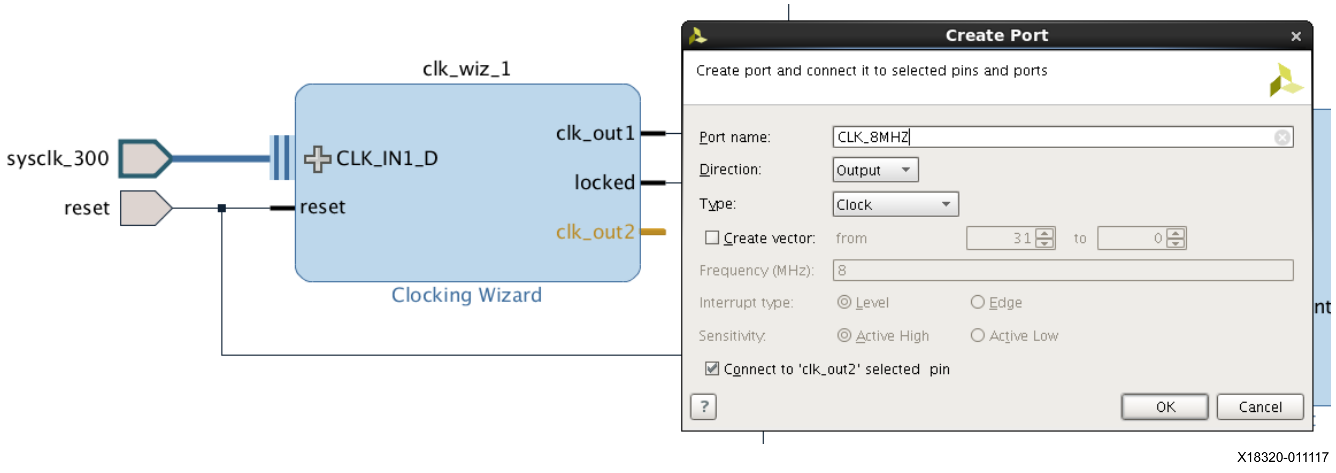
Phase Shift Mode: [✓ WAVEFORM] LATENCY

X18318-011117

Figure 7: **Clocking Wizard**

IMPORTANT: *For eFUSE programming, the 8.000 MHz clock feeds into the INT_PROG_CTL_US block. This is critical for generating the required programming pulses. The JTAG TAP machine must transition using an accurate clock during this phase to provide a 5 µs pulse for each eFUSE programmed. If the programming pulses are not accurate, eFUSE programming and reliability can be adversely affected.*

10. Right-click **clk_out2** port of the **Clocking Wizard** IP block, select **Create Port**, set **Port Name** to **CLK_8MHZ**, and click **OK** (Figure 8).



Create port and connect it to selected pins and ports

Port name: CLK_8MHZ

Direction: Output

Type: Clock

☐ Create vector: from 31 to 0

Frequency (MHz): 8

Interrupt type: ◎ Level ○ Edge

Sensitivity: ◎ Active High ○ Active Low

☑ Connect to 'clk_out2' selected pin

Figure 8: **Create Port**

11. From within the IP integrator block diagram window, search and add the following IP blocks by selecting the **Add IP** icon on the left (or top) side of the IP integrator window.
    a. **System Management Wizard** (monitors on-chip temperature and voltage prior to reading and writing eFUSEs).
    b. **AXI Timer** (tracks the eFUSE programming code section as a watchdog).
12. Click **Run Connection Automation** at the top of the IP integrator window, next to the Designer Assistance available.

13. Select the **All Automation** checkbox to select all the checkboxes, and click **OK** to automatically connect up the recently added IP blocks (Figure 9).

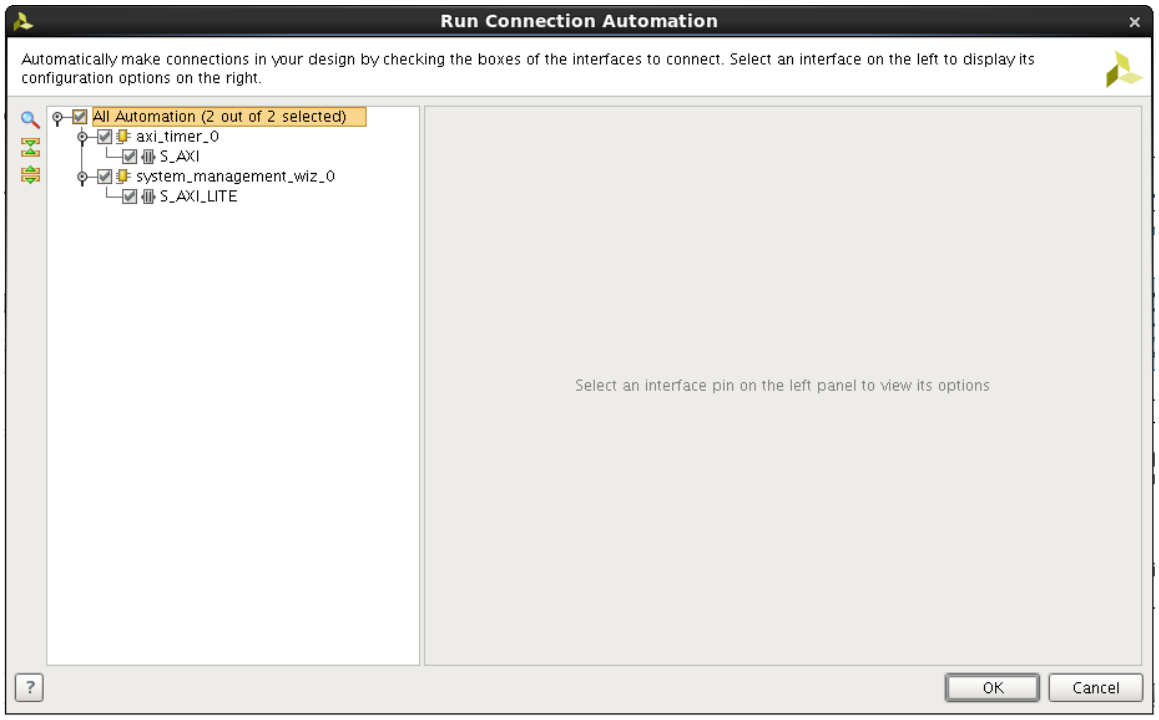
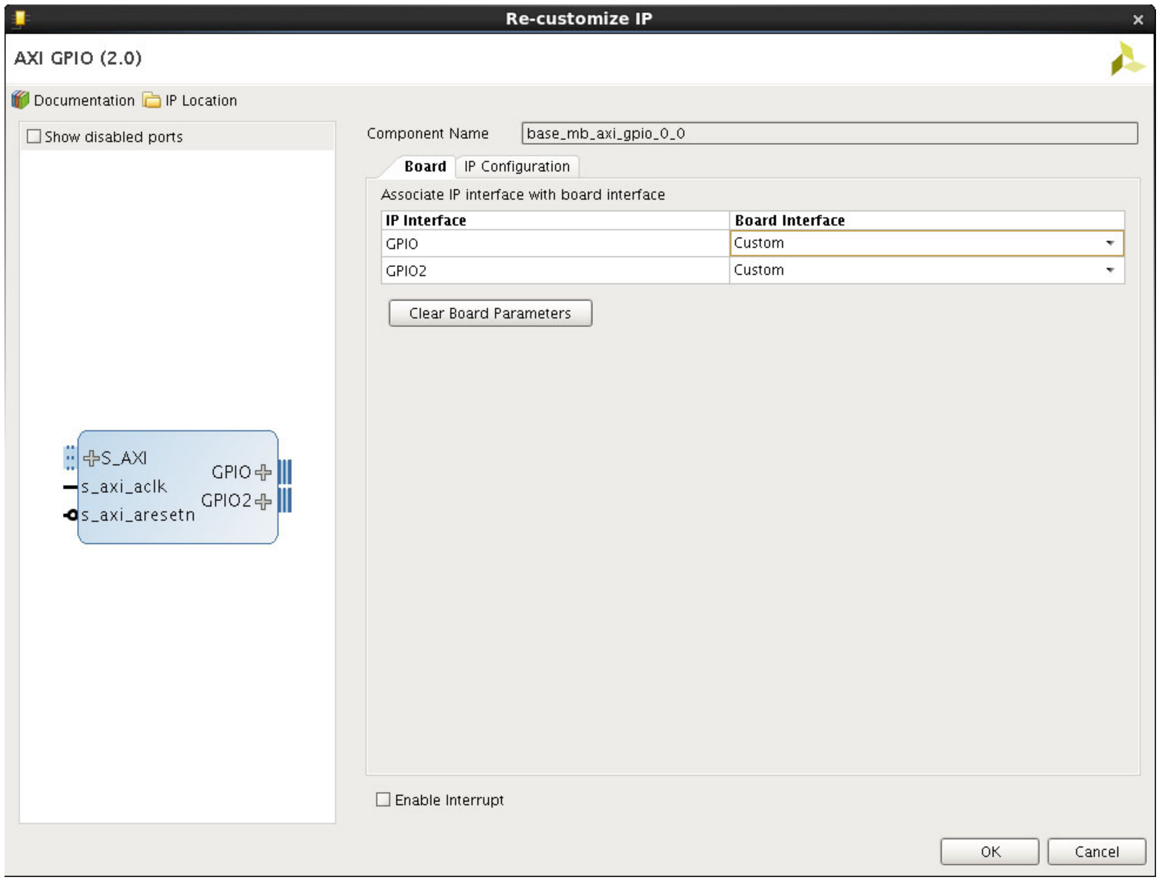

Run Connection Automation
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.
All Automation (2 out of 2 selected)
axi_timer_0
S_AXI
system_management_wiz_0
S_AXI_LITE
Select an interface pin on the left panel to view its options
OK
Cancel
X18321-011117

Figure 9: **Run Connection Automation**

14. Right-click on the **External Interface**: **led_8bits** led_8bits connected to the AXI GPIO IP block, and click **Delete**.

15. Double-click the **AXI GPIO** IP block to re-customize the IP.

16. Using the drop-down boxes, ensure that both the IP Interfaces GPIO and GPIO2 are set to **Custom Board Interface** (Figure 10).



Component Name base_mb_axi_gpio_0_0

Associate IP interface with board interface

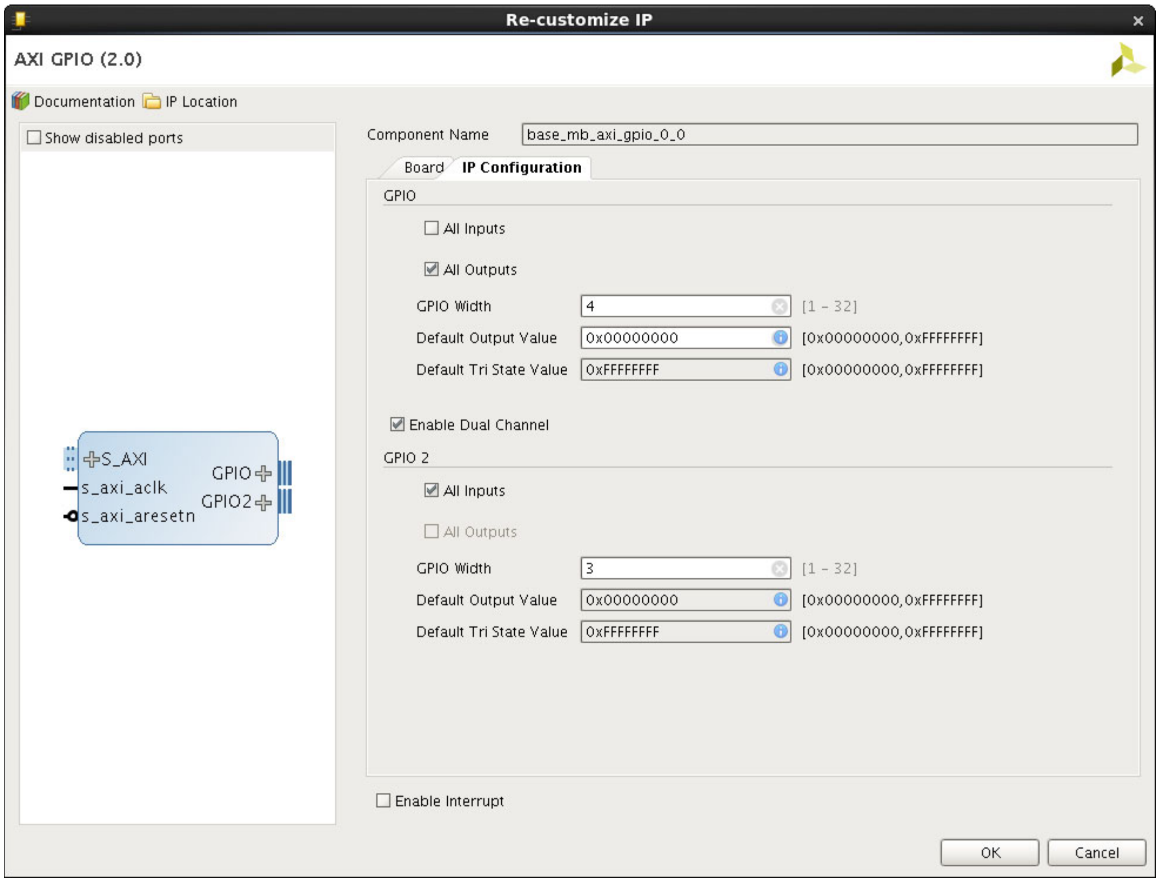
IP Interface	Board Interface
GPIO	Custom
GPIO2	Custom

☐ Enable Interrupt

X18322-011117

Figure 10: **Re-customize IP Interfaces**

17. Select the **IP Configuration** tab, check the **All Outputs** checkbox for GPIO, set **GPIO Width** to **4**, check the **Enable Dual Channel** checkbox, check the **All Inputs** checkbox for **GPIO 2**, set **GPIO Width** to **3**, and click **OK** (Figure 11).



Component Name: base_mb_axi_gpio_0_0

GPIO
- [ ] All Inputs
- [x] All Outputs

GPIO Width: 4 [1 - 32]
Default Output Value: 0x00000000 [0x00000000,0xFFFFFFFF]
Default Tri State Value: 0xFFFFFFFF [0x00000000,0xFFFFFFFF]

- [x] Enable Dual Channel

GPIO 2
- [x] All Inputs
- [ ] All Outputs

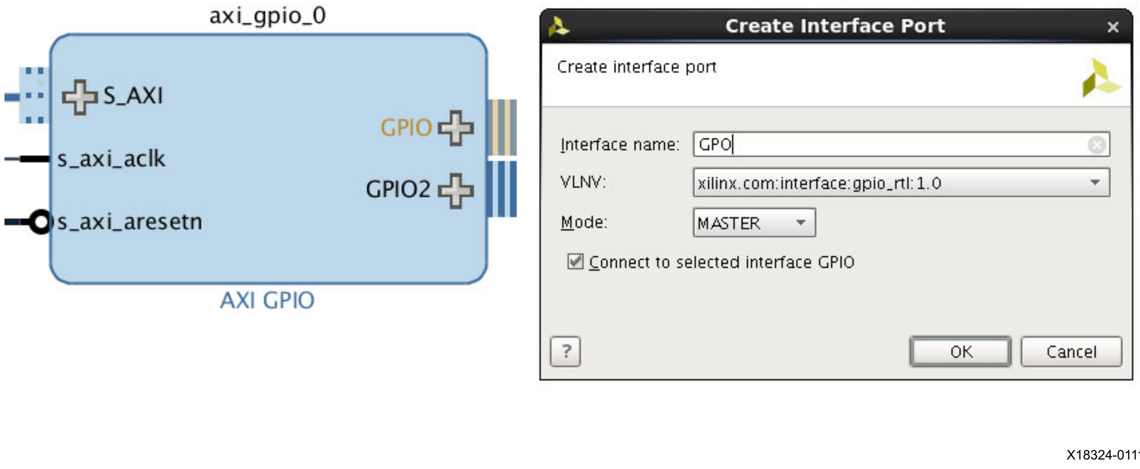
GPIO Width: 3 [1 - 32]
Default Output Value: 0x00000000 [0x00000000,0xFFFFFFFF]
Default Tri State Value: 0xFFFFFFFF [0x00000000,0xFFFFFFFF]

- [ ] Enable Interrupt

X18323-011117

Figure 11: **Re-customize IP GPIO Width**

18. Right-click the **GPIO** port of the **AXI GPIO** IP block, select **Create Interface Port**, set the **Interface Name** to **GPO**, and click **OK** (Figure 12).



Create interface port

Interface name: GPO

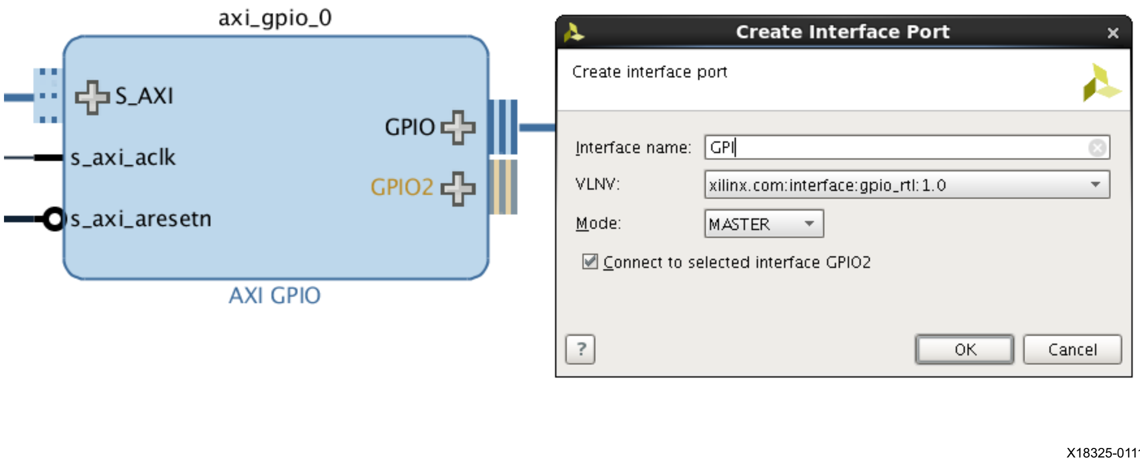
VLNV: xilinx.com:interface:gpio_rtl:1.0

Mode: MASTER

☑ Connect to selected interface GPIO

Figure 12: **Create Interface Port (GPIO)**

19. Right-click the **GPIO2** port of the **AXI GPIO** IP block, select **Create Interface Port**, set the **Interface Name** to **GPI**, and click **OK** (Figure 13).



Create interface port

Interface name: GPI

VLNV: xilinx.com:interface:gpio_rtl:1.0

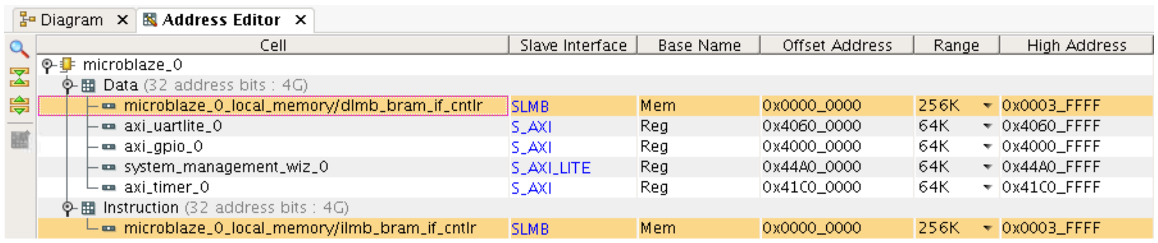
Mode: MASTER

☑ Connect to selected interface GPIO2

Figure 13: **Create Interface Port (GPIO2)**

20. Right-click the **Vp_Vn** port of the **System Management Wizard** IP block, and select **Make External**.

21. Select the **Address Editor** tab at the top of the IP integrator window, set the **Range** for **microblaze_0_local_memory/dlmd_bram_if_cntlr** to **256K** using the drop-down box, and set the **Range** for **microblaze_0_local_memory/ilmd_bram_if_cntlr** to **256K** using the drop-down box (Figure 14).



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	256K	0x0003_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
system_management_wiz_0	S_AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	256K	0x0003_FFFF

X18326-011117

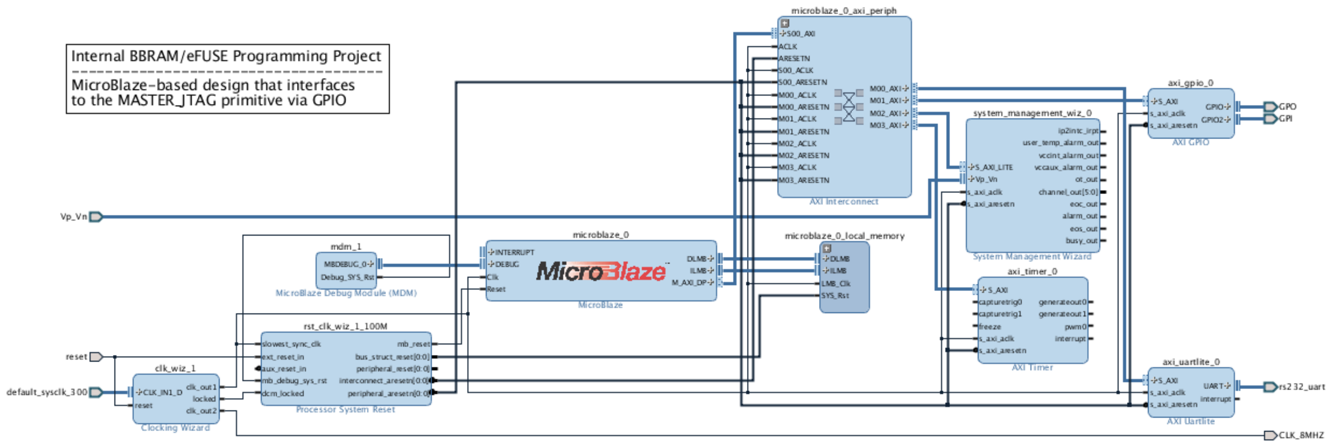
Figure 14: **Address Editor**

22. Select the **Diagram** tab at the top of the IP integrator window to return to the block diagram.

23. Click the **Validate Design** icon on the left (or top) side of the IP integrator window and correct any issues if the validation does not pass.

24. The large text labels on the block diagram can be moved and/or renamed.

25. Click the **Regenerate Layout** icon on the top left side of the IP integrator window to clean up the block diagram display. It looks similar to Figure 15. Complete the edits and save the block diagram.



X18327-011117

Figure 15: **MicroBlaze-based Internal BBRAM/eFUSE Programming Project Layout**

26. From the top-level Vivado tools menu, select **Tools** > **Project Settings** or **Settings** > **Target Language VHDL** > **OK**.

27. Select the **Sources** tab in the center window, select the **Hierarchy** tab, right-click **base_mb_i**, select **Create HDL Wrapper**, choose **Copy generated wrapper to allow user edits**, and click **OK** (Figure 16 and Figure 17).

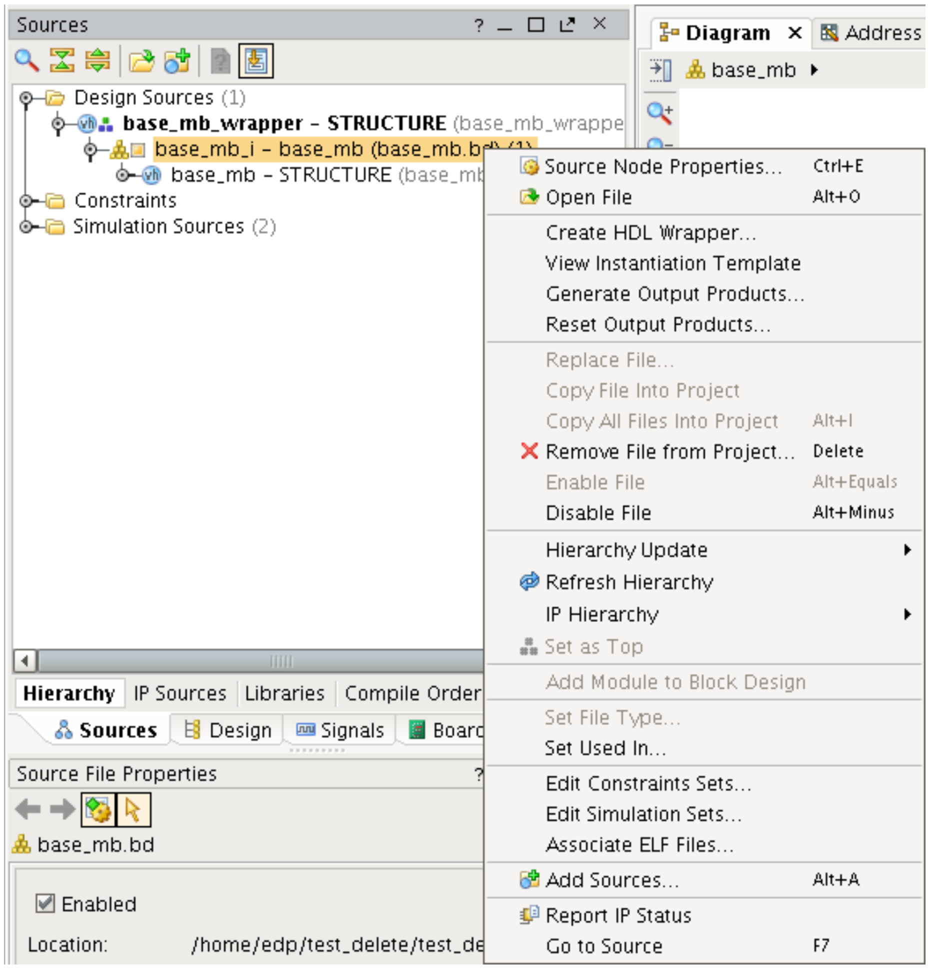
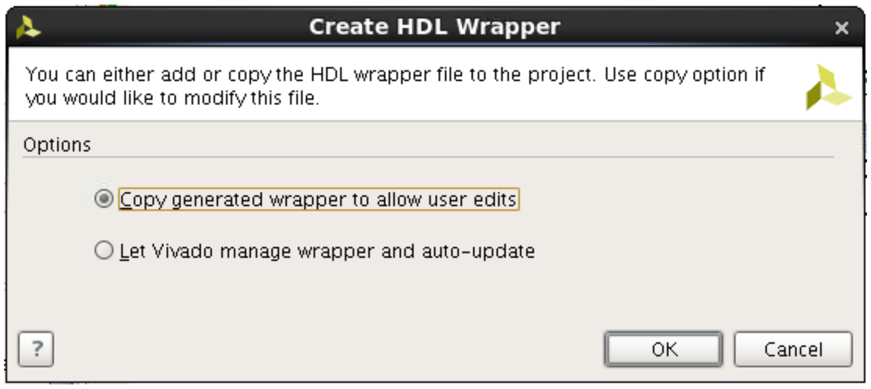


Figure 16: **Sources**



You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

[✓ Copy generated wrapper to allow user edits]

Let Vivado manage wrapper and auto-update

Figure 17: **Create HDL Wrapper**

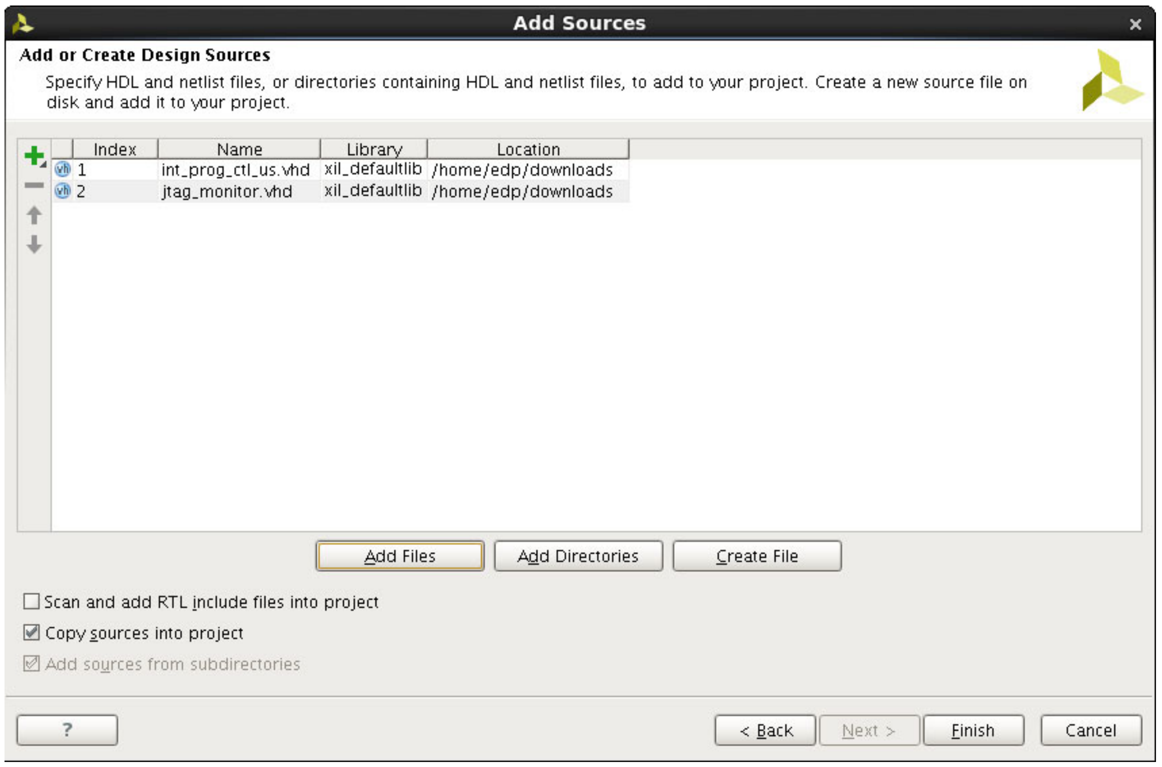
28. Double-click **base_mb_wrapper** in the center window under Design Sources to bring up the HDL file in the text editor and make the following changes:
    a. Remove the GPIO ports from the top-level port list and assign as internal signals.
    b. Instantiate and connect the INT_PROG_CTL_US component (this block instantiates the MASTER_JTAG primitive). See the INT_PROG_CTL_US Module for more information.
    c. Instantiate and connect the JTAG_MONITOR component (optional). See the JTAG_MONITOR Module for more information.

The following VHDL code sample demonstrates these changes (the VHDL wrapper file can also be downloaded from www.amd.com). See Reference Design for more information.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity base_mb_wrapper is
  port (
    Vp_Vn_v_n                : in  std_logic;
    Vp_Vn_v_p                : in  std_logic;
    default_sysclk_300_clk_n : in  std_logic;
    default_sysclk_300_clk_p : in  std_logic;
    reset                    : in  std_logic;
    rs232_uart_rxd           : in  std_logic;
    rs232_uart_txd           : out std_logic;
    EFUSE_PROG_PULSE         : out std_logic
  );
end base_mb_wrapper;

architecture STRUCTURE of base_mb_wrapper is

    signal clk_8mhz  : std_logic;
    signal gpi_tri_i : std_logic_vector(2 downto 0);
    signal gpo_tri_o : std_logic_vector(3 downto 0);
    signal jtag_mon  : std_logic_vector(3 downto 0);

    component base_mb is
        port (
            default_sysclk_300_clk_n : in  std_logic;
            default_sysclk_300_clk_p : in  std_logic;
            rs232_uart_rxd           : in  std_logic;
            rs232_uart_txd           : out std_logic;
            Vp_Vn_v_n                : in  std_logic;
            Vp_Vn_v_p                : in  std_logic;
            reset                    : in  std_logic;
            CLK_8MHZ                 : out std_logic;
            GPI_tri_i                : in  std_logic_vector(2 downto 0);
            GPO_tri_o                : out std_logic_vector(3 downto 0)
        );
    end component base_mb;

    component INT_PROG_CTL_US is
        generic (
            TEST_MODE : boolean := FALSE
        );
        port (
            RESET     : in  std_logic;
            CLK_8MHZ  : in std_logic; -- accuracy of the 5µs efuse prog pulse depends on this 8MHz
clock
            GPIO_IN   : in  std_logic_vector(3 downto 0); -- from software using gpio output
            GPIO_OUT  : out std_logic_vector(2 downto 0); -- to software using gpio input
            JTAG_OUT  : out std_logic_vector(3 downto 0)  -- for monitoring jtag signals
        );
    end component INT_PROG_CTL_US;

    component JTAG_MONITOR is
        port  (
            JTAG_MON          : in  std_logic_vector(3 downto 0); -- tdi=0, tms=1, tck=2, tdo=3
            EFUSE_PROG_PULSE : out std_logic
        );
    end component JTAG_MONITOR;



begin

    base_mb_i: base_mb
        port map (
            CLK_8MHZ                   => clk_8mhz,
            GPI_tri_i                  => GPI_tri_i,
            GPO_tri_o                  => GPO_tri_o,
            Vp_Vn_v_n                  => Vp_Vn_v_n,
            Vp_Vn_v_p                  => Vp_Vn_v_p,
            default_sysclk_300_clk_n => default_sysclk_300_clk_n,
            default_sysclk_300_clk_p => default_sysclk_300_clk_p,
            reset                      => reset,
            rs232_uart_rxd             => rs232_uart_rxd,
            rs232_uart_txd             => rs232_uart_txd
        );

    INT_PROG_CTL_US_i : INT_PROG_CTL_US
        generic map (
            TEST_MODE => FALSE
        )
        port map (
            RESET    => reset,
            CLK_8MHZ => clk_8mhz,
            GPIO_IN  => GPO_tri_o,
            GPIO_OUT => GPI_tri_i,
            JTAG_OUT => jtag_mon
        );

    JTAG_MONITOR_i : JTAG_MONITOR
        port map (
            JTAG_MON          => jtag_mon,
            EFUSE_PROG_PULSE => EFUSE_PROG_PULSE
        );

end STRUCTURE; -- base_mb_wrapper
```

29. Download the project VHDL files `int_prog_ctl_us.vhd` and `jtag_monitor.vhd` to your local computer. See Reference Design for more information.

30. Right-click **Design Sources** in the center Sources window, select **Add Sources**, click **Next**, select **Add or create design sources**, and click **Add Files**. Navigate to and add `int_prog_ctl_us.vhd` and `jtag_monitor.vhd` (files downloaded in step 29), check **Copy Sources into project**, and click **Finish** (Figure 18).


Add Sources

Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Index	Name	Library	Location
1	int_prog_ctl_us.vhd	xil_defaultlib	/home/edp/downloads
2	jtag_monitor.vhd	xil_defaultlib	/home/edp/downloads

☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

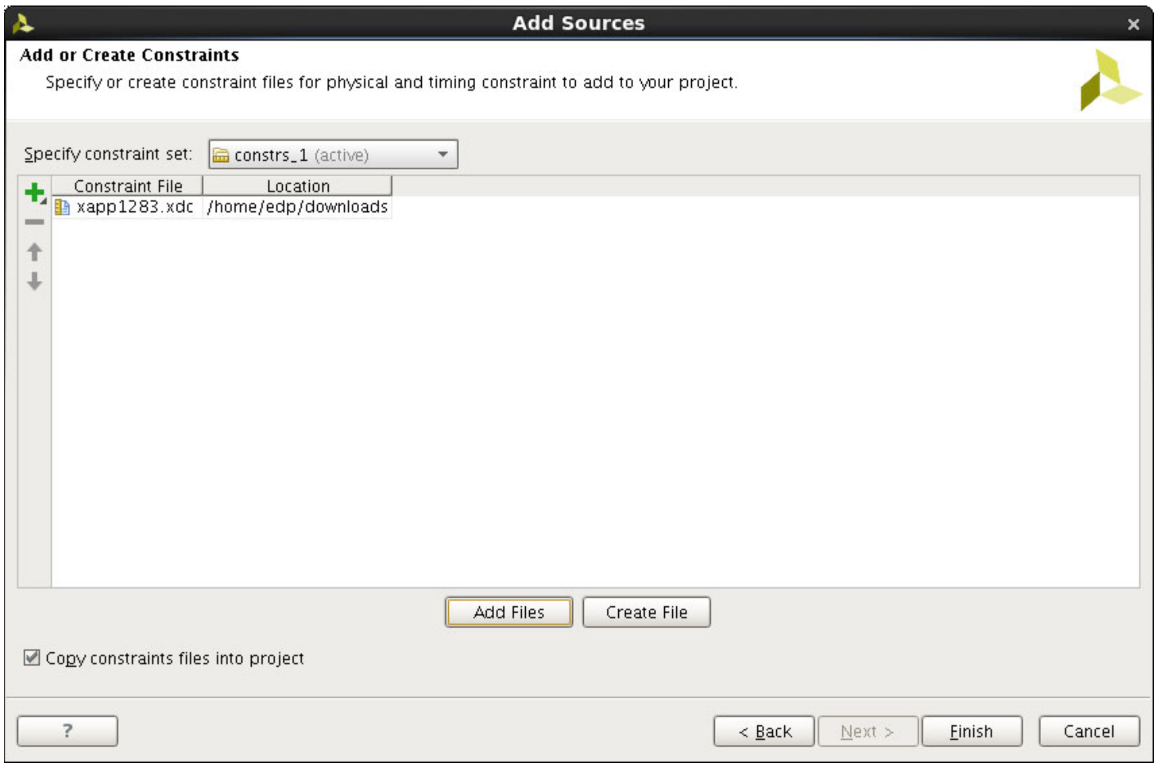
X18331-011117

Figure 18: **Add or Create Design Sources**

31. Download the project constraint file **`xapp1283.xdc`** (for UltraScale FPGAs) or **`xapp1283_usp.xdc`** (for UltraScale+ FPGAs) to your local computer (see Reference Design for more information). The constraint file sets up the external reset pin input and the EFUSE_PROG_PULSE monitor output pin.

    Note: If using a different evaluation kit or custom board, you will need to customize the constraint file appropriately (e.g., for specific physical pin locations). Also, if using SSIT FPGA devices there will be examples in the comment on how to include the MASTER_JTAG components for each SLR.

32. Right-click **Design Sources in the center Sources window,** select **Add Sources**, select **Add or create constraints**, click **Next**, and click **Add Files**. Navigate to and add the **`xapp1283.xdc`** or **`xapp1283_usp.xdc`** (file downloaded in step 31), check **Copy constraints files into project**, and click **Finish** (Figure 19).


Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
xapp1283.xdc	/home/edp/downloads

Figure 19: **Add or Create Constraints**

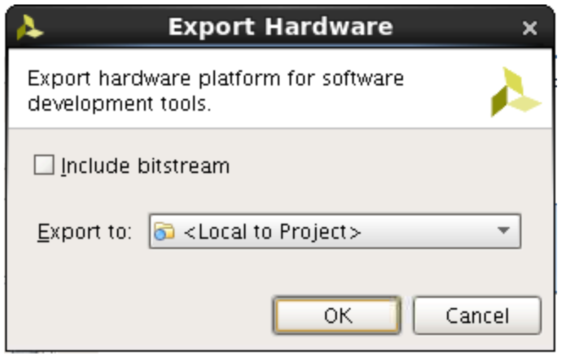
33. Select **Run Implementation** Run Implementation from the left flow navigator window under the Implementation section and click **Yes,** to automatically launch synthesis prior to implementation.

    Note: Any critical synthesis or implementation warnings associated with the board's dip switch can be safely ignored because it is not being used in this application note.

    Note: Interface port names can be slightly different than those in the provided reference design. Ensure the port names match prior to synthesis.

34. Click **Cancel** after the implementation is successfully completed.

35. Select **File** from the top-level Vivado tools menu, select **Export**, choose **Export Hardware**, and leave **Include bitstream** unchecked. Click **OK** to export **Local to Project** (Figure 20).



Export Hardware

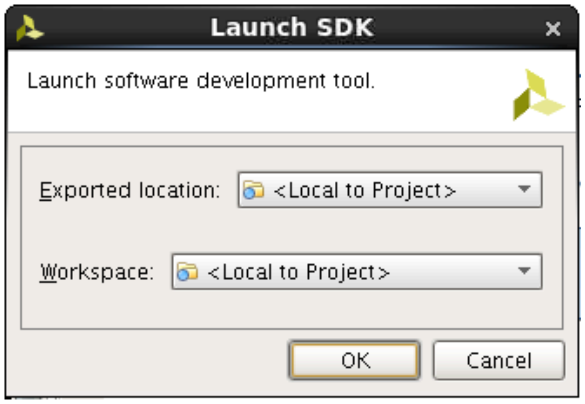
Export hardware platform for software development tools.

☐ Include bitstream

Export to: <Local to Project>

Figure 20: **Export Hardware**

36. Select **File** > **Launch SDK** from the top-level Vivado tools menu. Click **OK** to launch **Local to Project** (Figure 21). If using Vitis, select **Tools > Launch Vitis IDE**.



Launch SDK

Launch software development tool.

Exported location: <Local to Project>

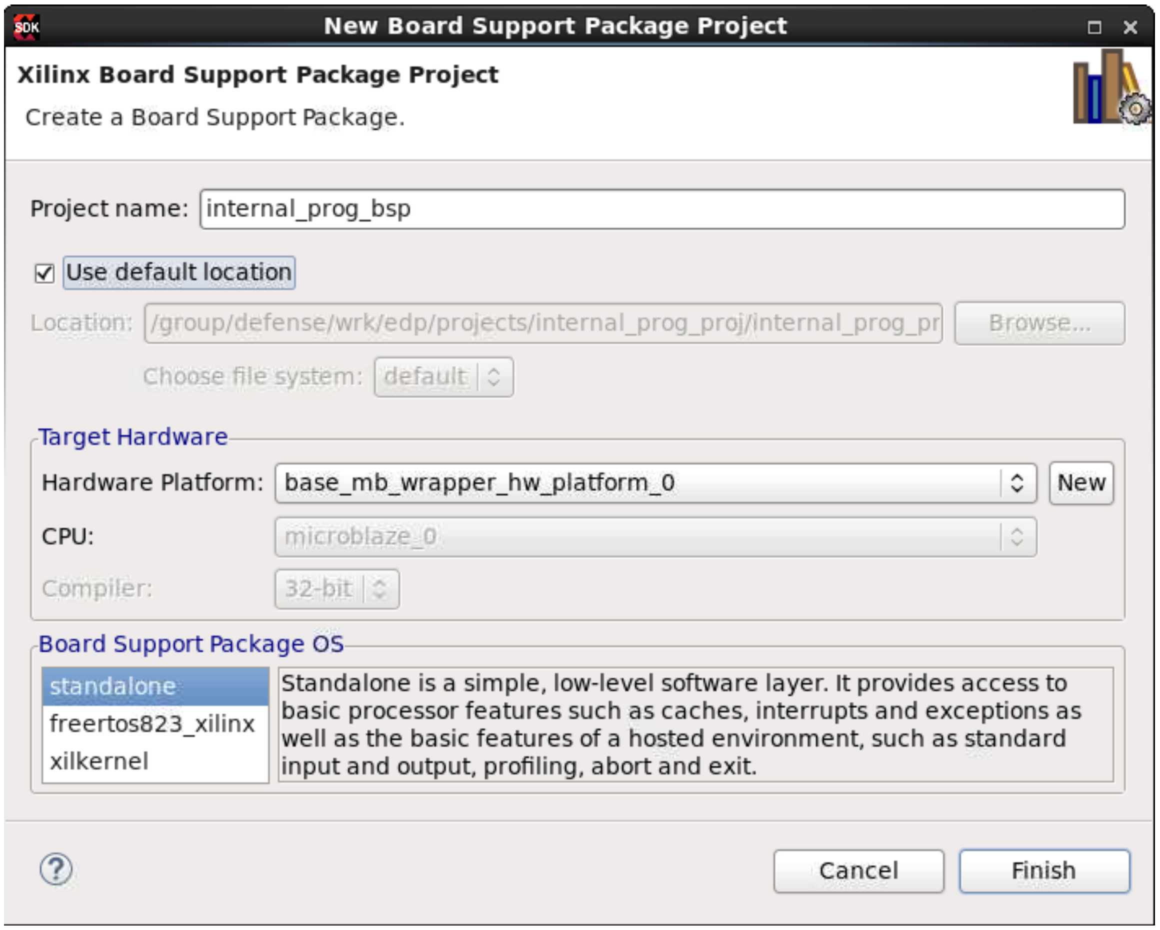
Workspace: <Local to Project>

Figure 21: **Launch SDK**

Building the SDK/Vitis Applications

The method to create three different SDK applications is explained in this section. In step 36 of the Vivado hardware section the SDK tool is launched with the MicroBlaze processor design as its hardware platform. The following steps are required for the creation of the SDK software applications. If using Vitis, the steps and filenames are similar to SDK. However there are some differences; see the Vitis references for details on building applications with that tool [Ref 23].

1. Select **File** > **New** > **Board Support Package** from the top-level SDK menu.
2. Set the project name to **internal_prog_bsp**, check the **Use default location** checkbox, select the **standalone Board Support Package OS**, and click **Finish** (Figure 22).


New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name: internal_prog_bsp

☑ Use default location

Location: /group/defense/wrk/edp/projects/internal_prog_proj/internal_prog_pr

Choose file system: default

Target Hardware

Hardware Platform: base_mb_wrapper_hw_platform_0

CPU: microblaze_0

Compiler: 32-bit

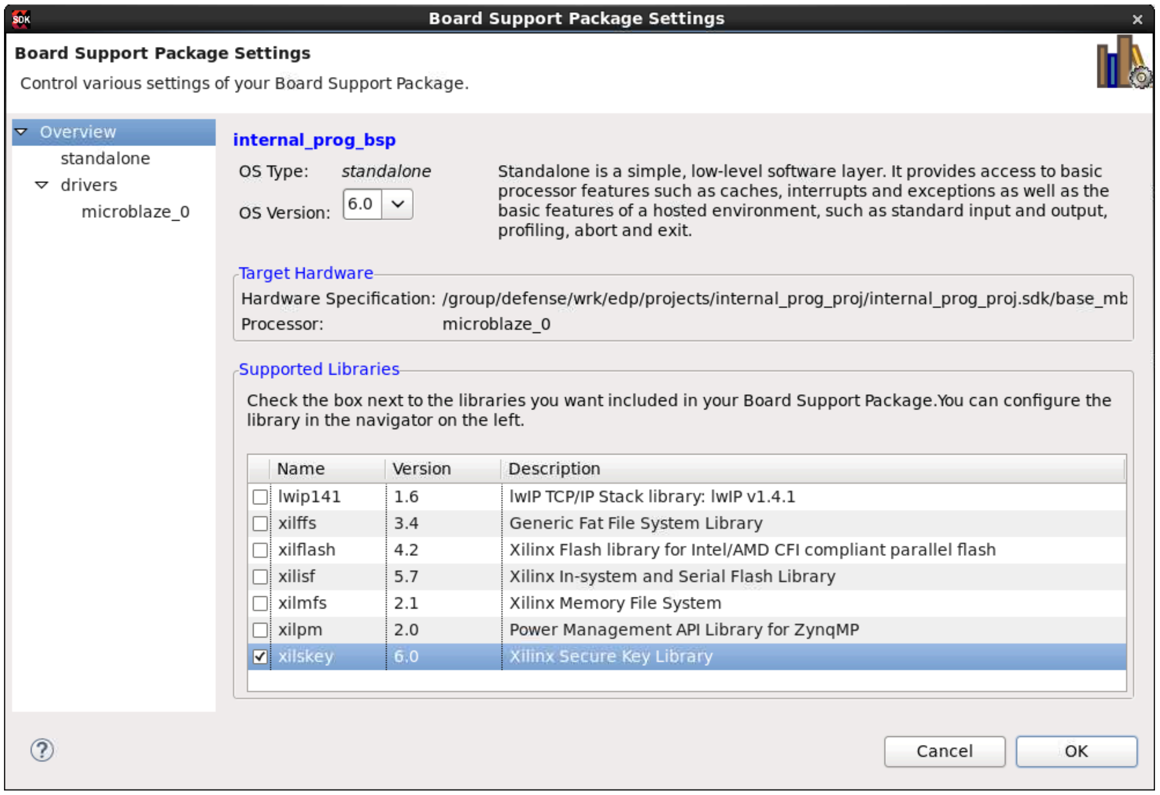
Board Support Package OS

standalone
freertos823_xilinx
xilkernel

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Figure 22: **Board Support Package Project**

3. Select **xilskey** from the Board Support Package Settings popup window (6.0 or later for UltraScale FPGAs, 6.3 or later for UltraScale+ FPGAs, 6.8 or later for SSIT FPGAs, or Zynq UltraScale+) **Xilinx Secure Key Library** and click **OK** (Figure 23).


Board Support Package Settings
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
drivers
microblaze_0
Cancel
OK

internal_prog_bsp

OS Type: *standalone*

OS Version: 6.0

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: /group/defense/wrk/edp/projects/internal_prog_proj/internal_prog_proj.sdk/base_mb

Processor: microblaze_0

Supported Libraries

Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

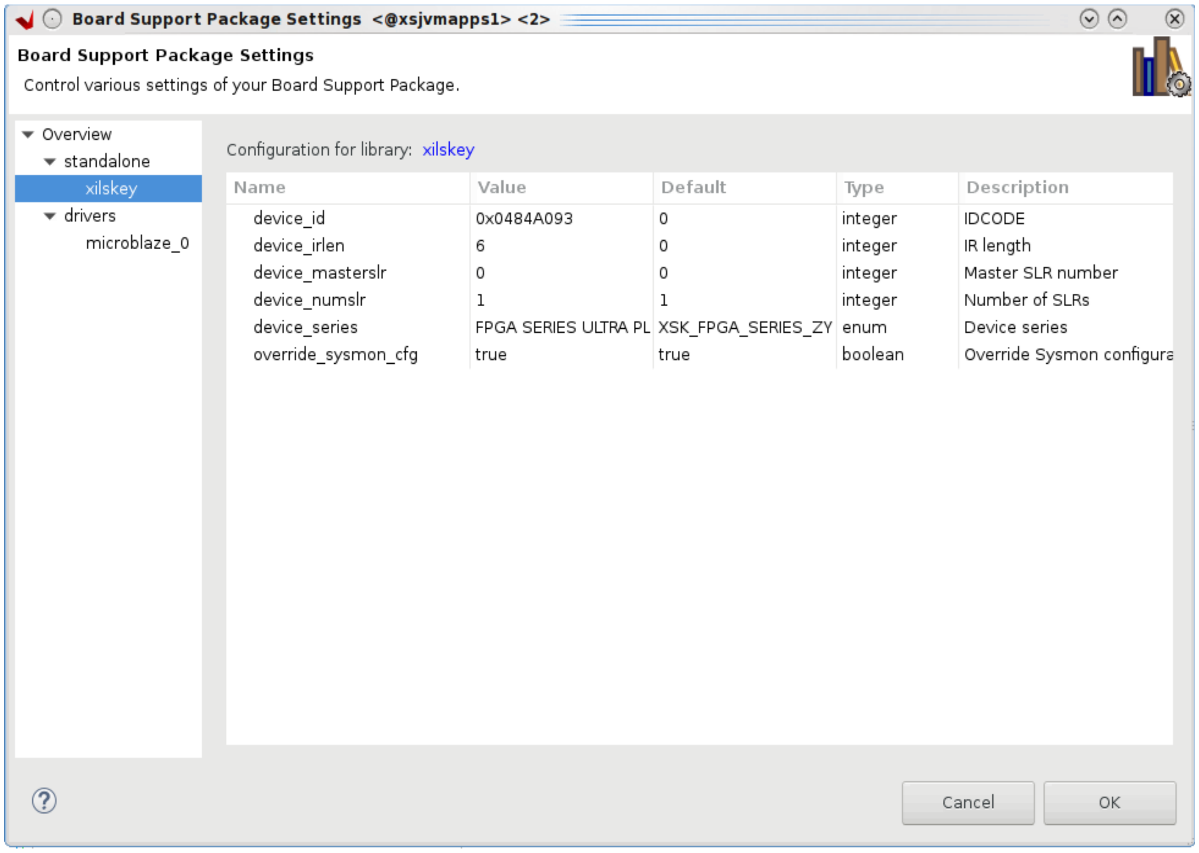
	Name	Version	Description
☐	lwip141	1.6	lwIP TCP/IP Stack library: lwIP v1.4.1
☐	xilffs	3.4	Generic Fat File System Library
☐	xilflash	4.2	Xilinx Flash library for Intel/AMD CFI compliant parallel flash
☐	xilisf	5.7	Xilinx In-system and Serial Flash Library
☐	xilmfs	2.1	Xilinx Memory File System
☐	xilpm	2.0	Power Management API Library for ZynqMP
✓	xilskey	6.0	Xilinx Secure Key Library

X18336-011117

Figure 23: **Board Support Package Settings**

4. Select the **system.mss** tab in the center window and select **Import Examples** under the **Libraries** section at the bottom.

   Note: If using Vitis version 2019.2 or later, it is necessary to modify the following xilskey library parameters: JTAG IDCODE, instruction register (IR) length (6), master SLR number and the number of SLRs. For FPGA devices, the JTAG IDCODE can be found in UG570 [Ref 2]. For Zynq UltraScale+ devices, the JTAG IDCODE in the PL is different than the published IDCODE for the PS. Contact your local AMD FAE or send an e-mail to *secure.solutions@amd.com* to get the appropriate PL JTAG IDCODE for your Zynq UltraScale+ device. The following figure (Figure 24) shows an example of these xilskey settings (your settings will differ depending on the device you are using).


Board Support Package Settings <@xsjvmapps1> <2>
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
xilskey
drivers
microblaze_0
Cancel
OK

Configuration for library: xilskey

Name	Value	Default	Type	Description
device_id	0x0484A093	0	integer	IDCODE
device_irlen	6	0	integer	IR length
device_masterslr	0	0	integer	Master SLR number
device_numslr	1	1	integer	Number of SLRs
device_series	FPGA SERIES ULTRA PL	XSK_FPGA_SERIES_ZY	enum	Device series
override_sysmon_cfg	true	true	boolean	Override Sysmon configura

Figure 24: **Examples for xilskey (1)**

5. Check the **xilskey_bbram_ultrascale_example** checkbox, check the **xilskey_efuse_example** checkbox, and click **OK** (Figure 25).

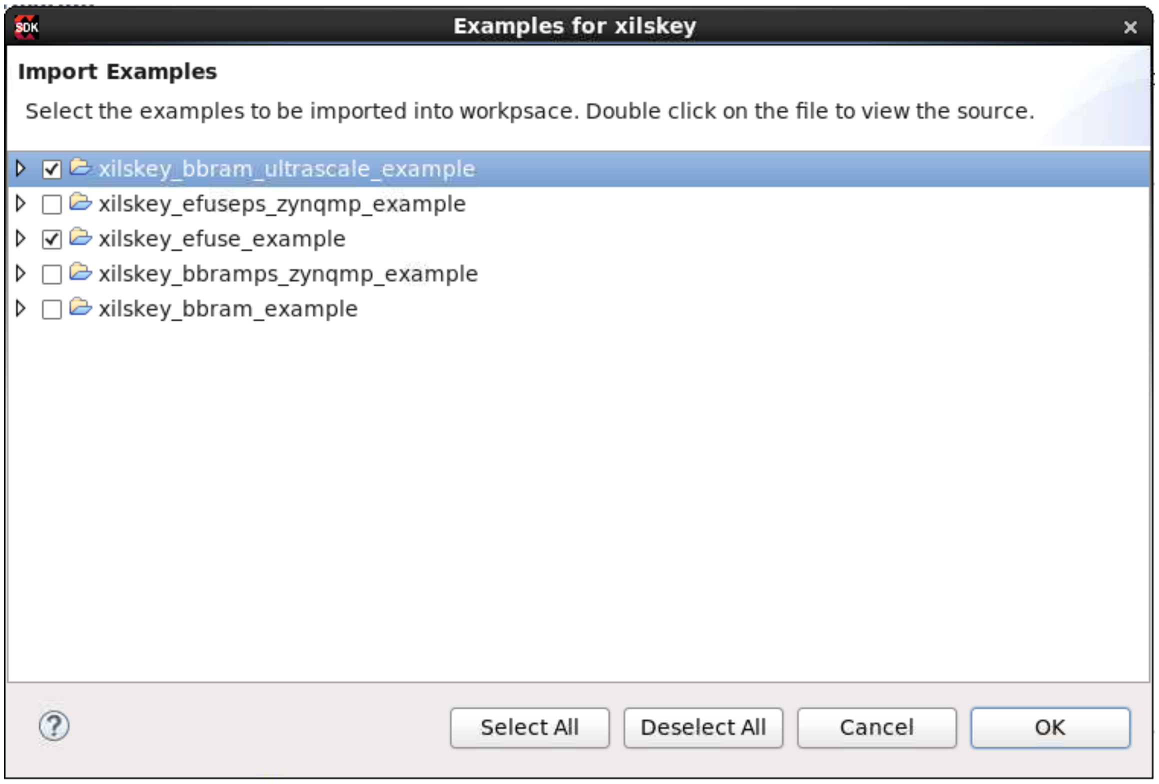
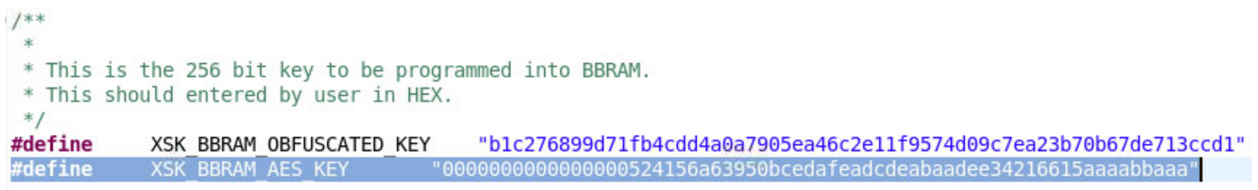

Examples for xilskey
Import Examples
Select the examples to be imported into workpsace. Double click on the file to view the source.
☑ xilskey_bbram_ultrascale_example
☐ xilskey_efuseps_zynqmp_example
☑ xilskey_efuse_example
☐ xilskey_bbramps_zynqmp_example
☐ xilskey_bbram_example
Select All
Deselect All
Cancel
OK
X18337-011117

Figure 25: **Examples for xilskey (2)**

6. Expand the **internal_prog_bsp_xilskey_bbram_ultrascale_example_1** project on the left side of the SDK project explorer window, expand **src**, and double-click **`xilskey_bbram_ultrascale_input.h`** to open up the header file in the SDK text editor.
7. Scroll down the header file to find the #define statement for **XSK_BBRAM_AES_KEY**, note and record the key value in quotations. Save any changes. The default value can be used in this example or you can enter your own value using exactly 64 hexadecimal characters. For this project, the obfuscated key (XSK_BBRAM_OBFUSCATED_KEY) is not used and can be ignored (Figure 26).

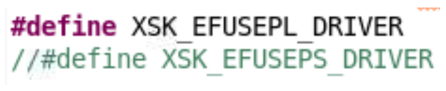


```
/**
 *
 * This is the 256 bit key to be programmed into BBRAM.
 * This should entered by user in HEX.
 */
#define     XSK_BBRAM_OBFUSCATED_KEY    "b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1"
#define     XSK_BBRAM_AES_KEY       "0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa"
```

Figure 26: **XSK_BBRAM_AES_KEY Value**

8. Expand the **internal_prog_bsp_xilskey_efuse_example_1** project on the left side of the SDK project explorer window, expand **src**, and double-click **`xilskey_input.h`** to open up the header file in the SDK text editor.

9. Scroll down the header file to find the #define statement for **XSK_EFUSEPS_DRIVER** and comment it out (this is only used for AMD Zynq™ SoC designs). Save the changes (Figure 27).



```
#define XSK_EFUSEPL_DRIVER
//#define XSK_EFUSEPS_DRIVER
```

Figure 27: **XSK_EFUSEPS_DRIVER**

10. Scroll through the **`xilskey_input.h`** file to see the different options (commands and values) for programming eFUSEs. For programming details refer to *Standalone Library Documentation: BSP and Libraries Document Collection* [Ref 8]. In this example a value is programmed into the 32-bit FUSE_USER register.

    Note: If using Vitis version 2019.2 or later, there will be multiple #define macro statements that will have the suffix "_CONFIG_ORDER_n" where *n* corresponds to the super logic region (SRL) configuration order when using SSIT devices. On some devices the Master SLR is SLR0, while on others the Master SLR is SLR1. It is important to note that all Master SLRs have a CONFIG_ORDER_INDEX value of 0. Consult the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2] to identify the Master SLR for the device being used. There is also a Tcl file included with this application note (`slr_report.tcl`) that can be sourced on an open Vivado implementation and provides a report on the device's SLRs.

    As an example, Vivado orders the BBRAM AES keys provided in the .nky key file as SLR0, SLR1, SLR2, etc. If UG570 states that the device's master is SLR0, then map the keys in the .nky file to the #define macros in that same order (1st key to _CONFIG_ORDER_0, 2nd key to _CONFIG_ORDER_1, 3rd key to _CONFIG_ORDER_2, etc.).

    However, if UG570 states that the device's master is SLR1, ensure that you provide the 2nd key in the `.nky` file at #define macro _CONFIG_ORDER_0, 1st key in the `.nky` file at #define macro _CONFIG_ORDER_1, the 3rd key at _CONFIG_ORDER_2, etc.

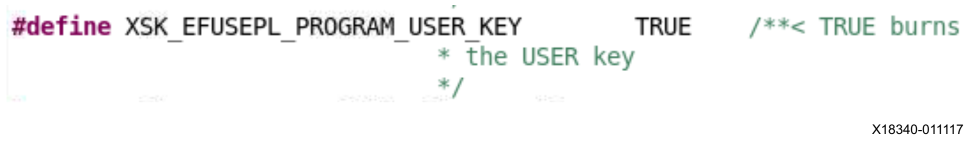
    This ordering scheme not only applies to BBRAM AES keys, it also applies to eFUSE AES keys, user fuses, and RSA PPK hashes. The target value to be programmed on the master is typically provided across the first #define macro (_CONFIG_ORDER_0), followed by the slave values (_CONFIG_ORDER_1, _CONFIG_ORDER_2, etc.).

    See the comments in the associated `int_prog_ctl_us.vhd` file and constraint (`.xdc`) files for SSIT usage examples.

IMPORTANT: *eFUSEs are OTP and the programming is irreversible. Use caution to choose eFUSEs that are programmed because they can affect the configuration and operation of the device. Also, the order in which eFUSEs are programmed should follow the guidelines shown in Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream* (XAPP1267) *[Ref 24]. See the section named eFUSE Programming General Recommendations which describes the proper order in which the eFUSEs should be programmed.*

IMPORTANT: *Do not enable XSK_EFUSEPL_ENABLE_RSA_AUTH (PL RSA authentication eFUSE bit) for the XCKU040-2FFVA1156E FPGA device located on the KCU105 evaluation platform. UltraScale devices have a minimum configuration width when RSA is enabled and the KCU105 evaluation board does not conform to* *this requirement. Enabling the PL RSA authentication eFUSE bit makes the device on this board permanently unconfigurable.*

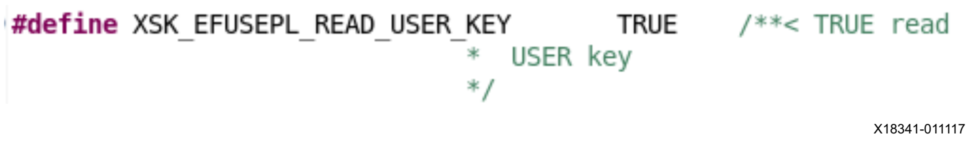
11. Find the #define statement for **XSK_EFUSEPL_PROGRAM_USER_KEY** and set its value to **TRUE** to program the 32-bit FUSE_USER register. Save the changes (Figure 28).



```
#define XSK_EFUSEPL_PROGRAM_USER_KEY        TRUE    /**< TRUE burns
                                    * the USER key
                                    */
```

Figure 28: **XSK_EFUSEPL_PROGRAM_USER_KEY Value**

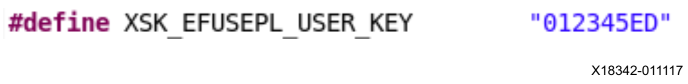
12. Find the #define statement for **XSK_EFUSEPL_READ_USER_KEY** and set its value to **TRUE** in order to read the 32-bit FUSE_USER register. Save the changes (Figure 29).



```
#define XSK_EFUSEPL_READ_USER_KEY       TRUE    /**< TRUE read
                                   *  USER key
                                   */
```

Figure 29: **XSK_EFUSEPL_READ_USER_KEY Value**

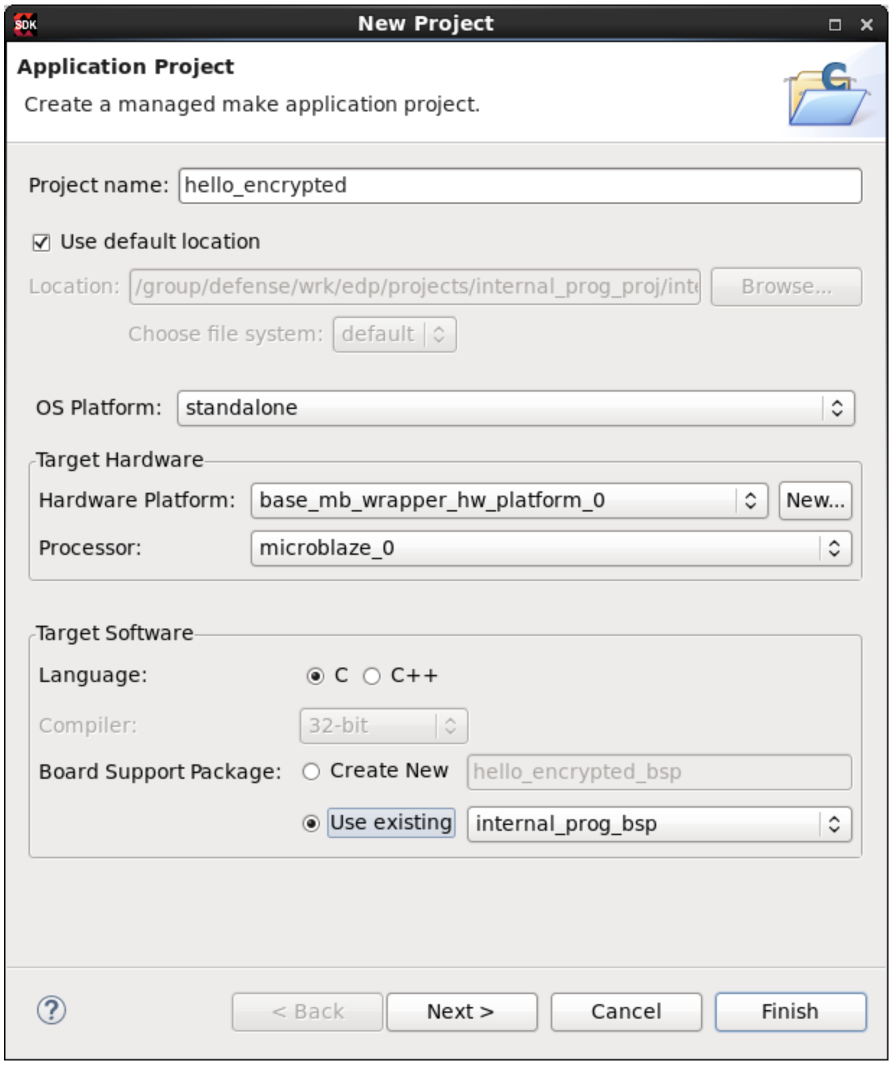
13. Find the #define statement for 32-bit FUSE_USER register value **XSK_EFUSEPL_USER_KEY** (this is not the AES decryption key) and set its value using exactly eight hexadecimal characters. All hexadecimal combinations are permitted with the exception of `0x00000000`. Note and record this value. Save the changes (see the example value in Figure 30).



```
#define XSK_EFUSEPL_USER_KEY        "012345ED"
```

Figure 30: **XSK_EFUSEPL_USER_KEY Value**

14. Select **File**, select **New**, select **Application Project**, and set the project name to **hello_encrypted**, from the top-level SDK menu. Under Board Support Package, check **Use existing**, click **Next**, select **Hello World** template, and click **Finish** (Figure 31 and Figure 32).



New Project

Application Project

Create a managed make application project.

Project name: hello_encrypted

[✓] Use default location

Location: /group/defense/wrk/edp/projects/internal_prog_proj/int

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: base_mb_wrapper_hw_platform_0

Processor: microblaze_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

Board Support Package: Create New hello_encrypted_bsp [✓ Use existing] internal_prog_bsp

X18343-011117

Figure 31: **Create an Application Project**

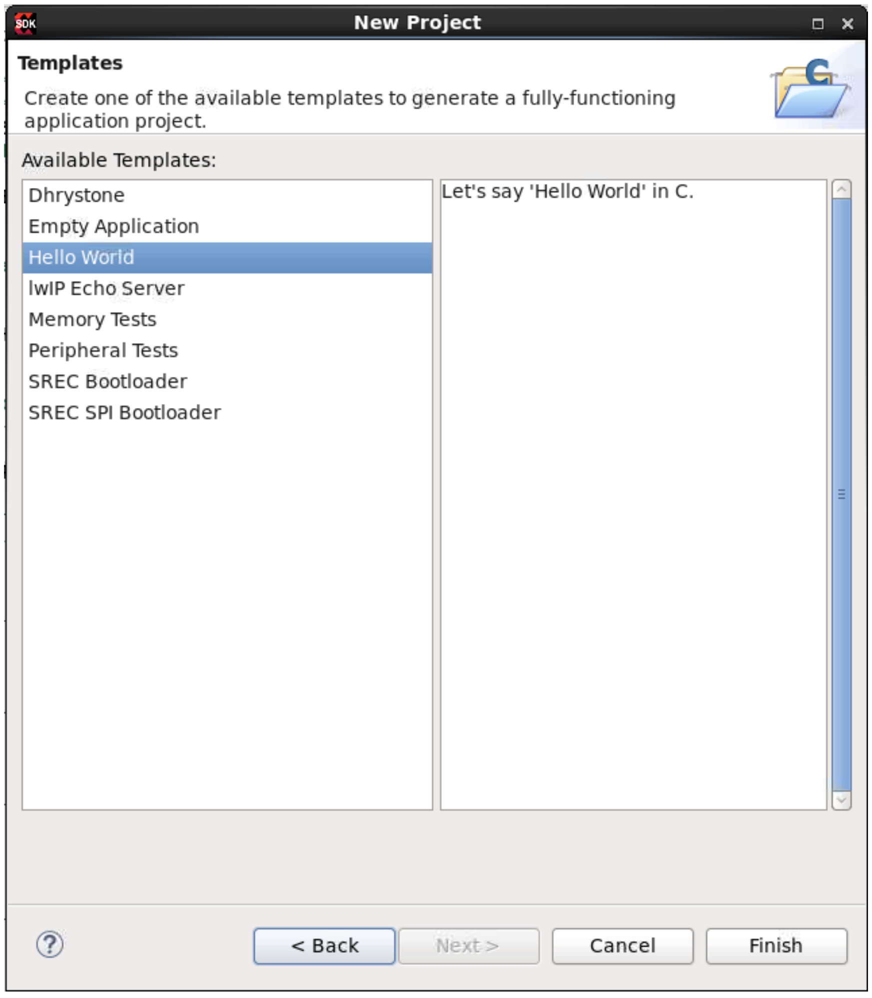


Figure 32: **Application Project Template**

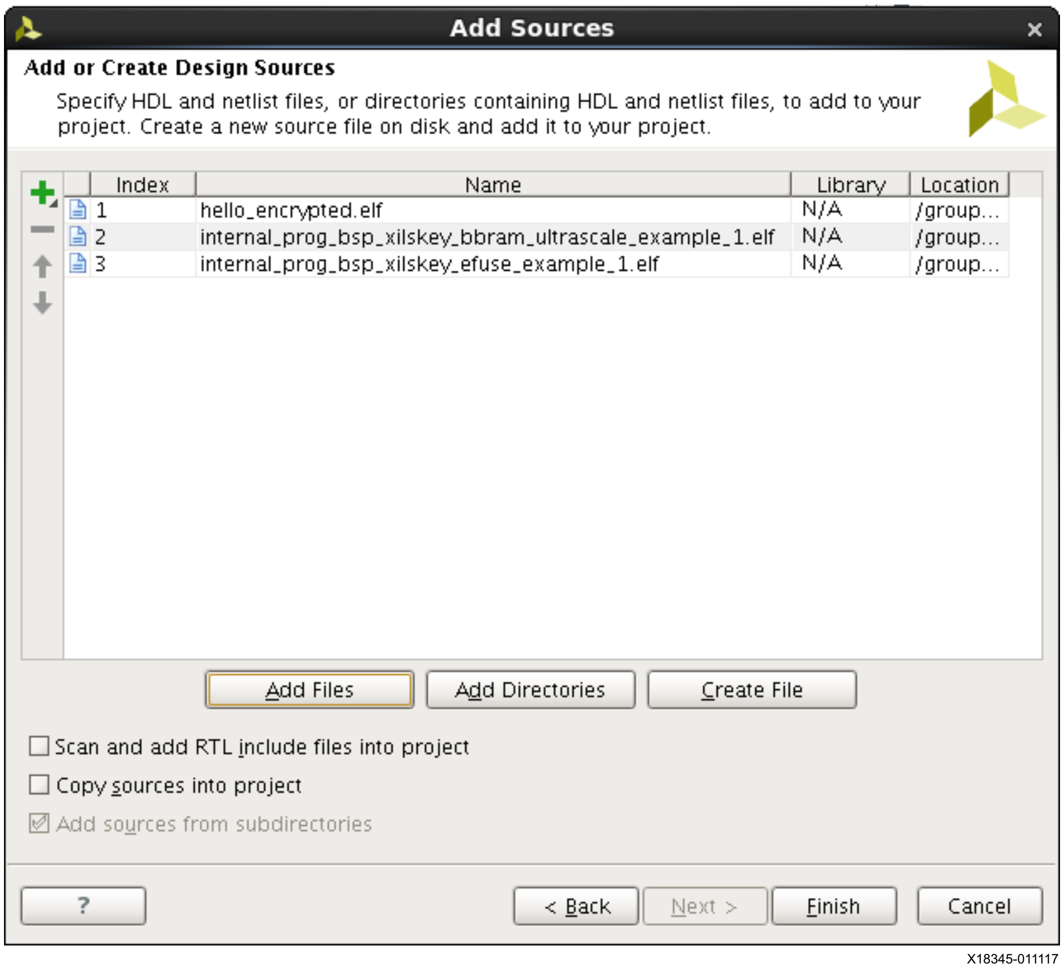
Note: Before building the projects, inspect the loader scripts (lscript.ld) for the BBRAM and eFUSE examples and ensure the heap size is at least 0xF00 and the stack size is at least 0xC00.

15. Select **Project** > **Clean** > **Clean all projects**, and click **OK** from the top-level SDK menu.
16. Select **Project** and click **Build All** from the top-level SDK menu. This generates the necessary binary ELF files for each of the three application projects.

Bitstream Generation

The three ELF files created in the SDK are imported back into the Vivado tools to merge the associated MicroBlaze processor instructions and data into each of the bitstreams. The bitstreams for programming the BBRAM AES key and the 32-bit user eFUSE register are unencrypted and the Hello application is encrypted (using the same key value from the C header file). The following steps outline the generation of these bitstreams:

1. Return to the Vivado tools GUI with the recently built internal_prog_proj project loaded.
2. Select the **Sources** tab in the center window, and then select the **Hierarchy** tab.
3. Right-click **Design Sources** > **Add Sources** > **Add or create design sources** > **Next** > **Add Files**.
4. Browse to `<project_path>/internal_prog_proj/internal_prog_proj.sdk`. From this location point to each of the following ELF files and add them to the project:
    a. hello_encrypted/<Debug or Release>/`hello_encrypted.elf`
    b. internal_prog_xilskey_bbram_ultrascale_example_1/<Debug or Release>/ `internal_prog_bsp_xilskey_bbram_ultrascale_example_1.elf`
    c. internal_prog_xilskey_efuse_example_1/<Debug or Release>/ `internal_prog_bsp_xilskey_efuse_example_1.elf`
5. Do not check **Copy sources into project**.
6. Click **Finish** (Figure 33).



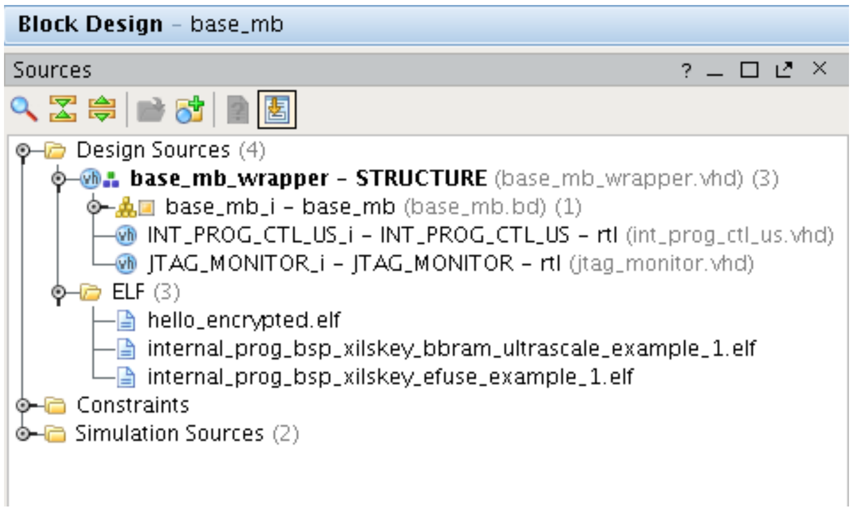
Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Index	Name	Library	Location
1	hello_encrypted.elf	N/A	/group...
2	internal_prog_bsp_xilskey_bbram_ultrascale_example_1.elf	N/A	/group...
3	internal_prog_bsp_xilskey_efuse_example_1.elf	N/A	/group...

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Figure 33: **Add Sources**

7. The three ELF files are now visible under Design Sources and ELF in the following figure (Figure 34).



Block Design – base_mb

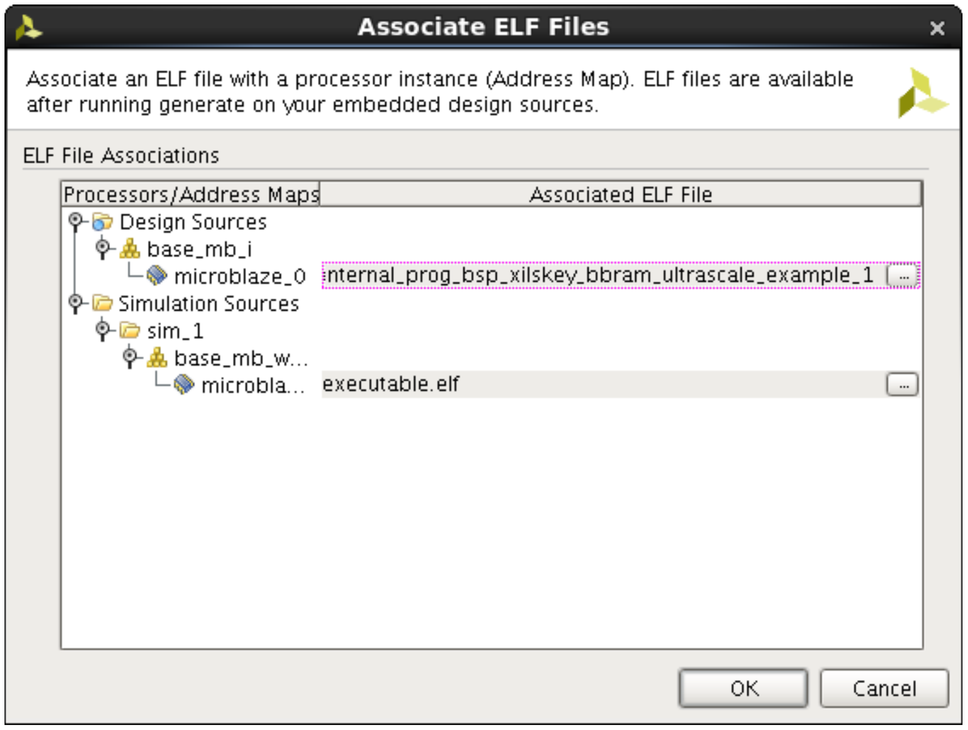
Sources

- Design Sources (4)
  - **base_mb_wrapper – STRUCTURE** (base_mb_wrapper.vhd) (3)
    - base_mb_i – base_mb (base_mb.bd) (1)
    - INT_PROG_CTL_US_i – INT_PROG_CTL_US – rtl (int_prog_ctl_us.vhd)
    - JTAG_MONITOR_i – JTAG_MONITOR – rtl (jtag_monitor.vhd)
  - ELF (3)
    - hello_encrypted.elf
    - internal_prog_bsp_xilskey_bbram_ultrascale_example_1.elf
    - internal_prog_bsp_xilskey_efuse_example_1.elf
- Constraints
- Simulation Sources (2)

X18346-011117

Figure 34: **Block Design**

8. Under Design Sources right-click on **internal_prog_bsp_xilskey_bbram_ultrascale_example_1.elf** > **Associate ELF Files**. Under Associated ELF File for Design Sources browse ..., select **internal_prog_bsp_xilskey_bbram_ultrascale_example_1.elf**, click **OK**, and click **OK** (Figure 35).



Associate ELF Files

Associate an ELF file with a processor instance (Address Map). ELF files are available after running generate on your embedded design sources.

ELF File Associations

Processors/Address Maps	Associated ELF File
Design Sources	
base_mb_i	
microblaze_0	nternal_prog_bsp_xilskey_bbram_ultrascale_example_1
Simulation Sources	
sim_1	
base_mb_w...	
microbla...	executable.elf

X18347-011117

Figure 35: **Associate ELF Files (BBRAM)**

9. Select **Generate Bitstream** Generate Bitstream from the flow navigator window on the left of the Vivado tools screen, under Program and Debug.
10. Click **Cancel** when the bitstream generation is completed.
11. Using a file browser, navigate to **project_path>/ internal_prog_proj/internal_prog_proj.runs/impl_1** (this path is used again when the bitstreams are programmed into the device).
12. Rename `base_mb_wrapper.bit` to `prog_bbram.bit`. Copy `prog_bbram.bit` to a temporary folder for later use.
13. Right-click on **internal_prog_bsp_xilskey_efuse_example_1.elf** under Design Sources and select **Associate ELF Files.** Under Associated ELF File for Design Sources, browse ..., select **internal_prog_bsp_xilskey_efuse_example_1.elf**, click **OK**, and click **OK** (Figure 36).

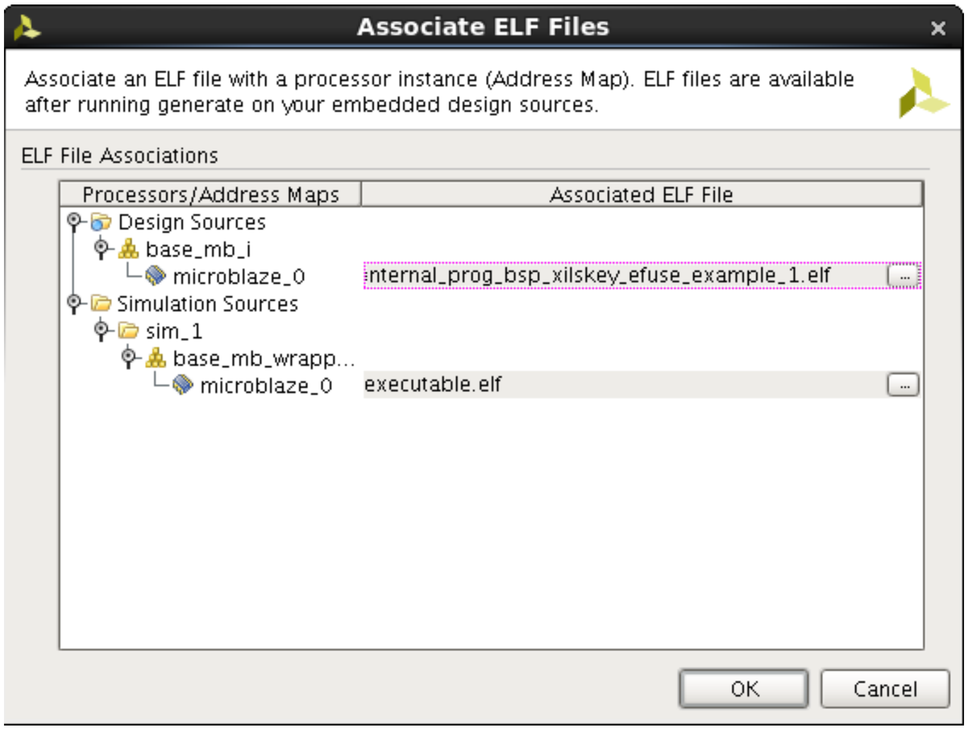
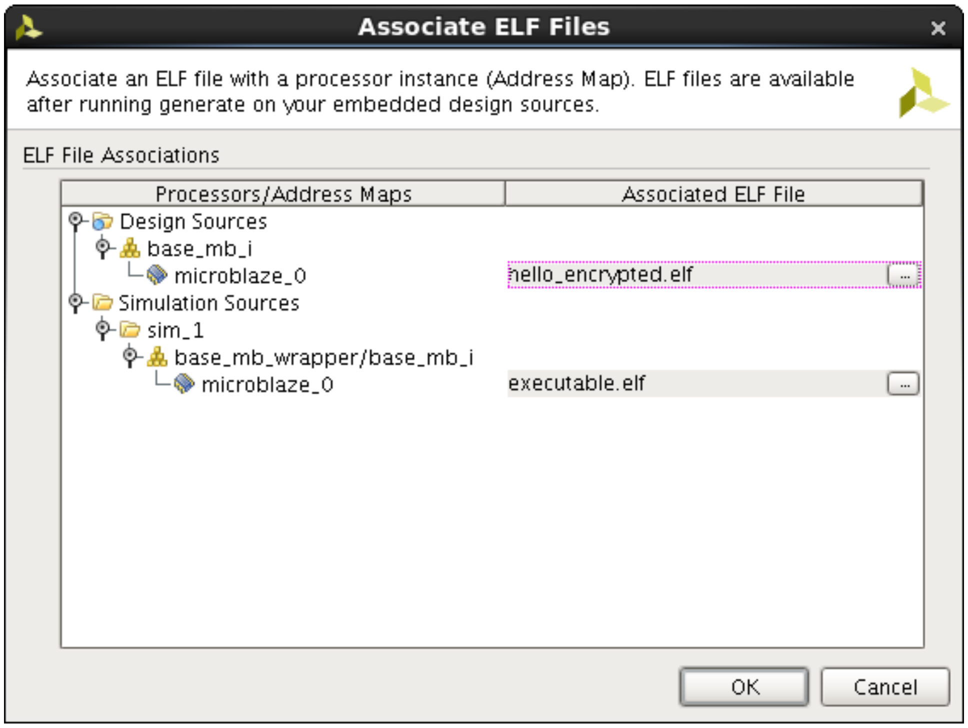


Figure 36: **Associate ELF Files (eFUSE)**

14. From the flow navigator window on the left of the Vivado tools screen, under Program and Debug, select **Generate Bitstream** Generate Bitstream.
15. When the bitstream generation is completed, click **Cancel**.
16. Using a file browser, navigate to **<project_path>/ internal_prog_proj/internal_prog_proj.runs/impl_1**.
17. Rename **base_mb_wrapper.bit** to **prog_efuse.bit**. Copy `prog_efuse.bit` to the same temporary folder as `prog_bbram.bit`.

18. Under Design Sources, right-click on **`hello_encrypted.elf`** > **Associate ELF Files**. Under Associated ELF File for Design Sources, browse ..., select **`hello_encrypted.elf`**, click **OK**, and click **OK** (Figure 37).

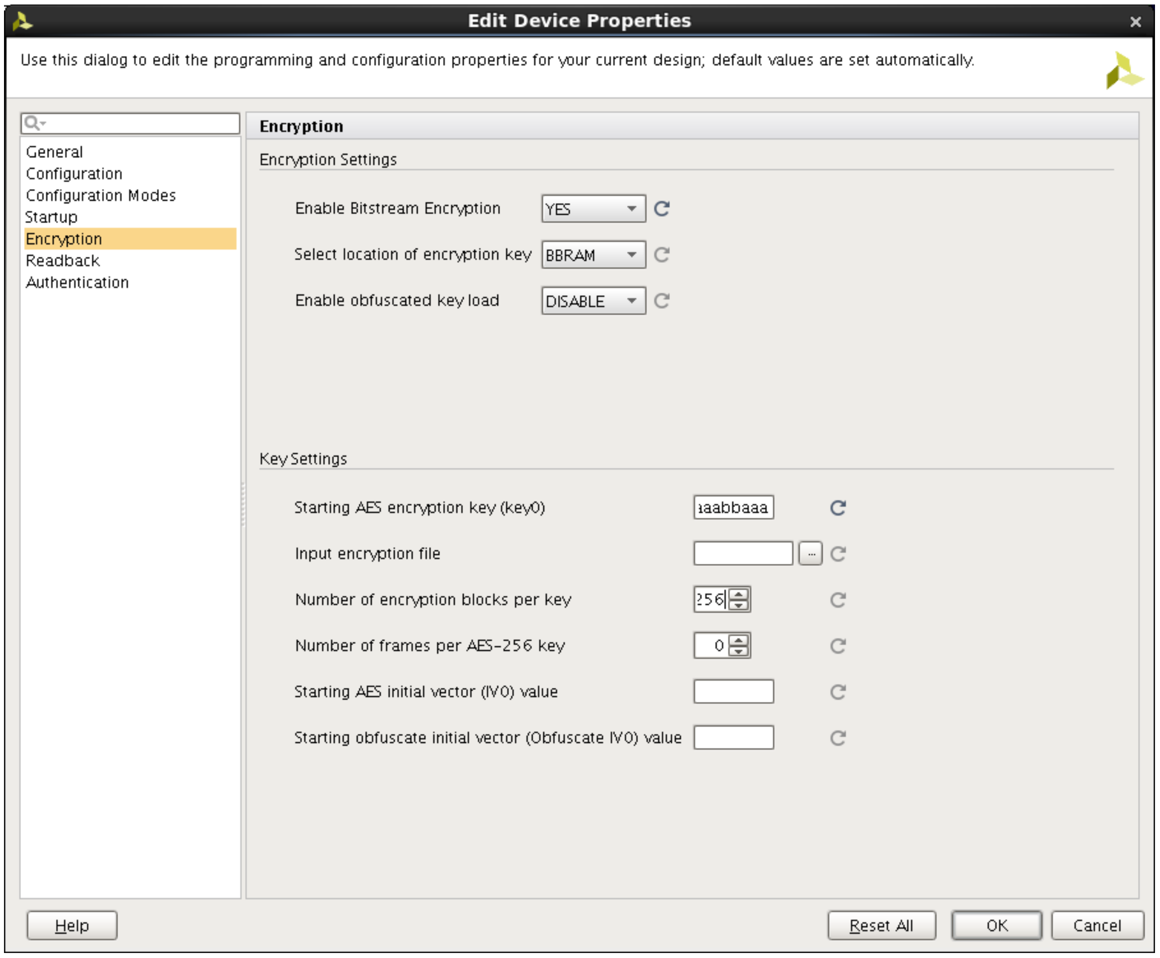


X18350-011117

Figure 37: **Associate ELF Files (hello)**

19. From the flow navigator window on the left of the Vivado tools screen, under Implementation, select **Open Implemented Design** Open Implemented Design.
20. Ignore any warning messages that appear and click **OK**.
21. From the flow navigator window on the left of the Vivado tools screen, under Program and Debug, select **Bitstream Settings** Bitstream Settings or right-click on **Generate Bitstream,** and select **Bitstream Settings**.
22. Select **Configure additional bitstream settings**, on the left, select **Encryption**, set **Enable Bitstream Encryption: Yes**, and **Select location of encryption key: BBRAM**.
23. Copy and paste the 256-bit AES key (the XSK_BBRAM_AES_KEY value from `xilskey_bbram_ultrascale_input.h`) from Building the SDK/Vitis Applications section into the **Starting AES encryption key (key0)** text box.

24. Set **Number of encryption blocks per key: 256**, click **OK**, and click **OK** (Figure 38). For more information on encryption block selection, refer to the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2]. For more information on key rolling, refer to the *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs Application Note* (XAPP1098) [Ref 7].


Edit Device Properties

Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.

General
Configuration
Configuration Modes
Startup
Encryption
Readback
Authentication

Encryption

Encryption Settings

Enable Bitstream Encryption	YES
Select location of encryption key	BBRAM
Enable obfuscated key load	DISABLE

Key Settings

Starting AES encryption key (key0)	ıaabbaaa
Input encryption file	
Number of encryption blocks per key	256
Number of frames per AES-256 key	0
Starting AES initial vector (IV0) value	
Starting obfuscate initial vector (Obfuscate IV0) value	

Figure 38: **Edit Device Properties**

25. From the flow navigator window on the left of the Vivado tools screen, under Program and Debug, select **Generate Bitstream** Generate Bitstream, select **Save**, check **Select an existing file**, set the filename as **`xapp1283.xdc`** (for UltraScale FPGAs) or **`xapp1283_usp.xdc`** (for UltraScale+ FPGAs), click **OK**, and click **Overwrite** (Figure 39 and Figure 40). This adds the encryption options and key value to the constraints file.

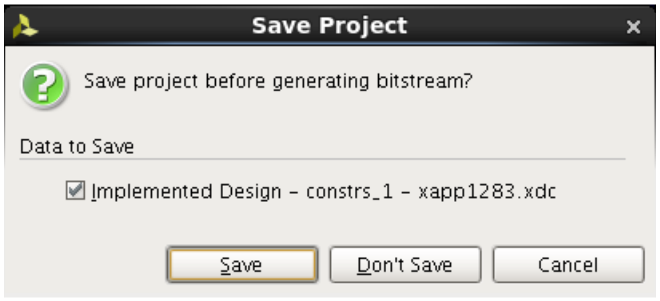
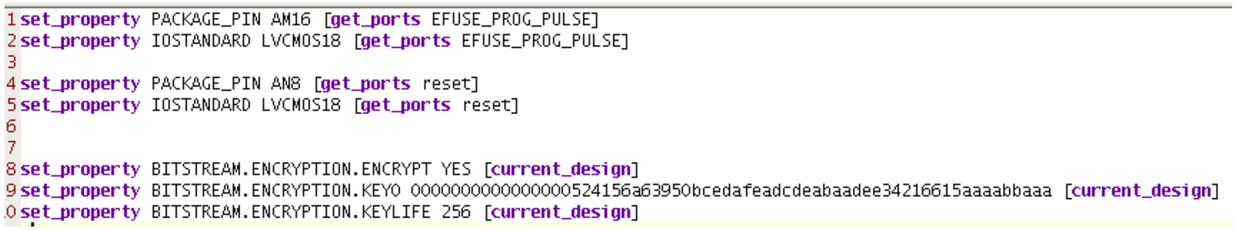

Save Project
Save project before generating bitstream?
Data to Save
☑ Implemented Design – constrs_1 – xapp1283.xdc
Save
Don't Save
Cancel
X18354-011117

Figure 39: **Save Project**



```
set_property PACKAGE_PIN AM16 [get_ports EFUSE_PROG_PULSE]
set_property IOSTANDARD LVCMOS18 [get_ports EFUSE_PROG_PULSE]

set_property PACKAGE_PIN AN8 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports reset]


set_property BITSTREAM.ENCRYPTION.ENCRYPT YES [current_design]
set_property BITSTREAM.ENCRYPTION.KEY0 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa [current_design]
set_property BITSTREAM.ENCRYPTION.KEYLIFE 256 [current_design]
```

Figure 40: **set_property**

26. Click **Cancel** when the bitstream generation is completed.

27. Using a file browser, navigate to `<project_path>/ internal_prog_proj/internal_prog_proj.runs/impl_1`.

28. Rename `base_mb_wrapper.bit` to `hello_encrypted.bit`. Copy `hello_encrypted.bit` to the same temporary folder as `prog_bbram.bit` and `prog_efuse.bit`.

Verification

After the three bitstreams `prog_bbram.bit`, `prog_efuse.bit`, and `hello_encrypted.bit` are successfully generated, you can load each onto the Kintex UltraScale or UltraScale+ device located on the KCU105 or KCU116 evaluation kit respectively and verify their functionality. The following steps outline the verification of each of these bitstreams:

1. Power up and connect the KCU105 (or KCU116) evaluation board to a computer using the USB JTAG and USB UART cables. For more information when using the UltraScale device, refer to the *KCU105 Quick Start Guide* (XTP391) [Ref 17] and *KCU105 Board User Guide* (UG917) [Ref 18]. For more information when using the UltraScale+ device, refer to the *KCU116 Evaluation Kit Quick Start Guide* (XTP471) [Ref 19] and *KCU116 Evaluation Board User Guide* (UG1239) [Ref 20].
2. On the KCU105 evaluation board, set dip switch **SW15** to the following settings for the JTAG configuration mode:
    - Positions **1** through **5** = **OFF**
    - Position **6** = **ON**

   On the KCU116 evaluation board, set dip switch **SW21** to the following settings for the JTAG configuration mode:
    - Position **6** = **ON**
3. Determine the COM port that connects to the MicroBlaze processor's UART using Windows device manager. The USB UART connection establishes two COM ports (standard and enhanced) on the computer (the Silicon Labs serial driver must be installed on the system). The computer COM port associated with the Silicon Labs Standard COM port connects to the MicroBlaze processor's UART.
4. Start up the serial communications terminal program. Following is the serial port settings:
    - COM Port = Silicon Labs Standard COM Port
    - Baud rate: 115200
    - Data: 8
    - Parity: none
    - Stop: none
    - Flow control: none
5. Press and release the **PROGRAM_B** switch **SW4** on the KCU105 board (or switch **SW5** on the KCU116 board).

IMPORTANT: *The PROGRAM_B switch must be pressed between each bitstream load because the design contains the MASTER_JTAG primitive. Failure to do so blocks the external JTAG path and the computer's USB JTAG communications fail.*

6. Select **Hardware Manager** Hardware Manager in the Vivado tools from the flow navigator window on the left of the screen, under Program and Debug. Select **Open Target** Open Target, and click **Auto Connect**.
7. Right-click the **xcku040_0** device (or **xcku5p_0** device) in the center hardware window and select **Clear BBR Key**. Click **OK** (Figure 41).

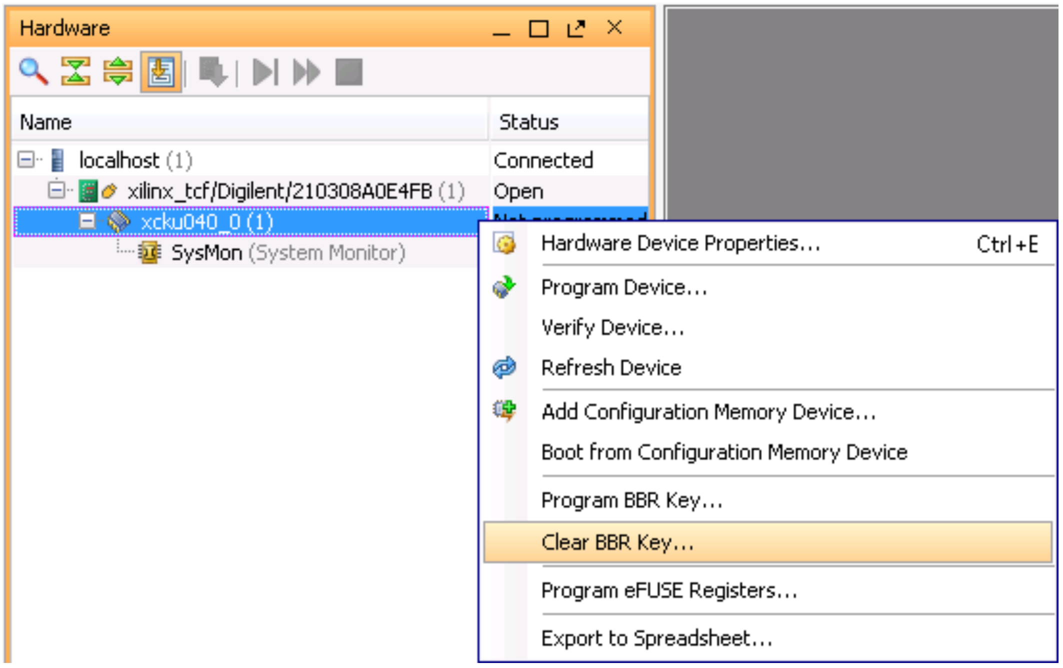


Figure 41: **Hardware Manager BBR Key**

8. Right-click the **xcku040_0** device (or **xcku5p_0** device) in the center hardware window, and select **Program Device** (Figure 42).

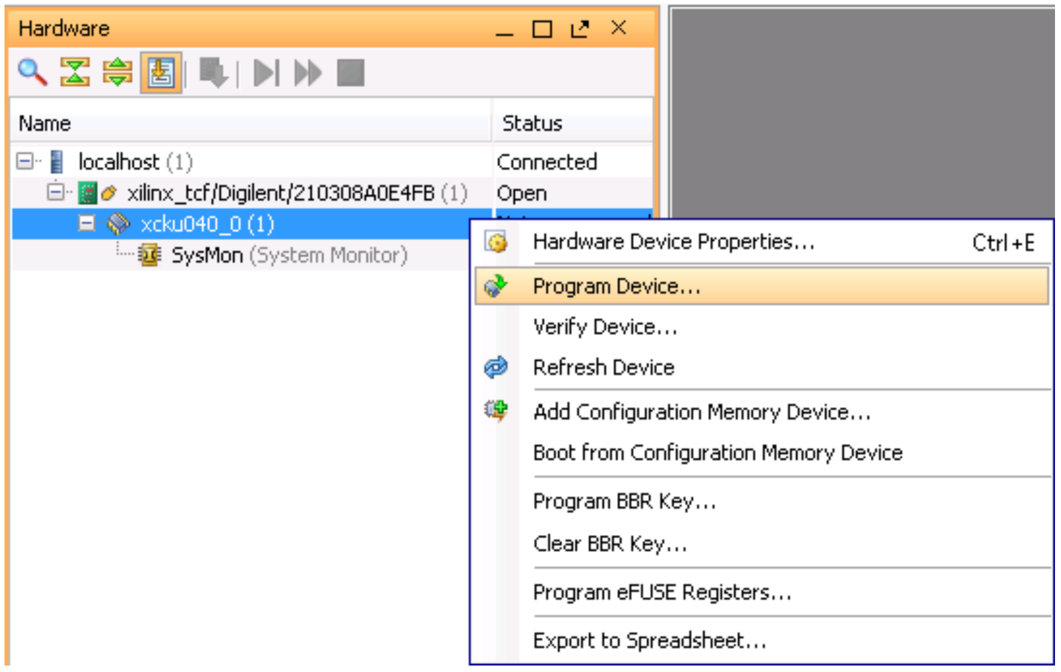
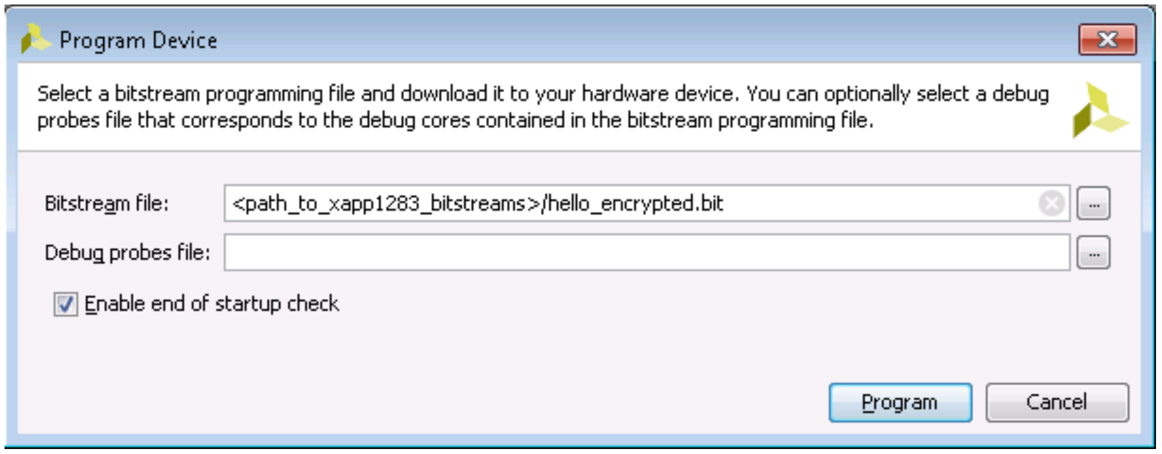


Figure 42: **Hardware Manager Program Device**

9. Set **Bitstream file: <path_to_bitstreams>/`hello_encrypted.bit`,** and click **Program** (Figure 43).


Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: <path_to_xapp1283_bitstreams>/hello_encrypted.bit

Debug probes file:

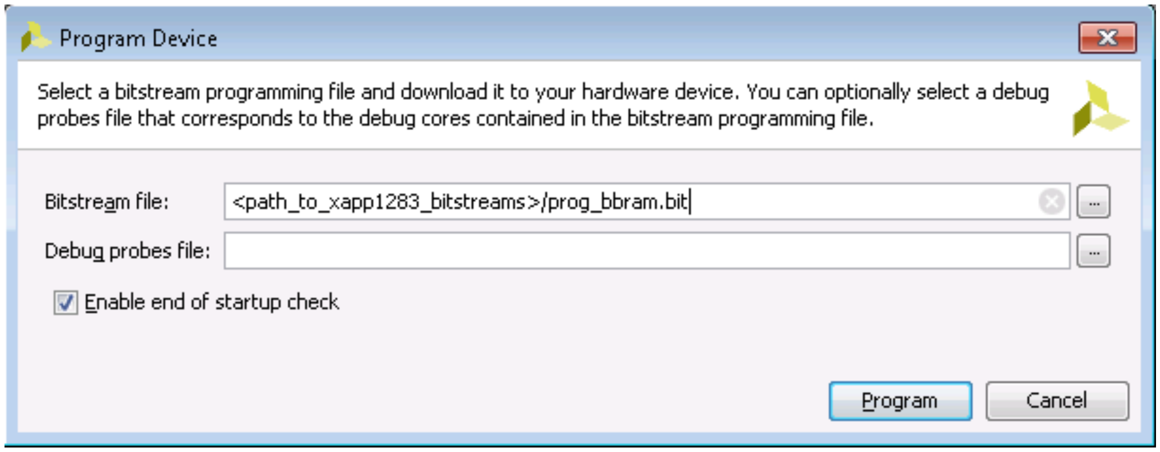
[✓] Enable end of startup check

Figure 43: **Program Device (hello_encrypted.bit)**

10. The hardware manager attempts to program the device. However, because there is no BBRAM key, the configuration fails. When the bitstream finishes loading, verify that the DONE LED (DS34 for the KCU105 evaluation board and DS32 for the KCU116 evaluation board) is OFF.

11. Power cycle the board with the main power switch **SW1**. Then press and release the **PROGRAM_B** switch **SW4** on the KCU105 board (**SW5** on the KCU116 board).

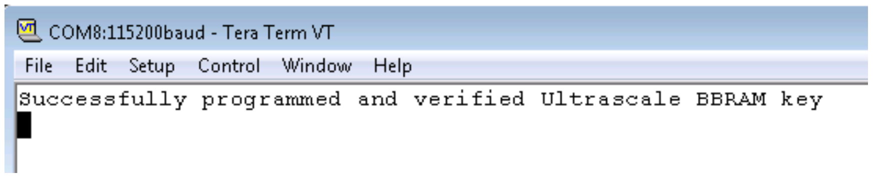
12. Right-click the **xcku040_0** device (or **xcku5p_0** device) in the center hardware window. Select **Program Device**, set **Bitstream file: <path_to_bitstreams>/`prog_bbram.bit`**, and click **Program** (Figure 44).


Program Device
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.
Bitstream file: <path_to_xapp1283_bitstreams>/prog_bbram.bit
Debug probes file:
☑ Enable end of startup check
Program
Cancel

X18360-011117

Figure 44: **Program Device (prog_bbram.bit)**

13. When the bitstream finishes loading, the DONE LED (DS34 for the KCU105 board and DS32 for the KCU116 board) turns ON. It blocks the FPGA's external JTAG port and causes communication errors in the Vivado Tcl Console because the MASTER _JTAG is instantiated in the design.These errors are expected and can be safely ignored.
14. The serial terminal window displays a message for a successfully programmed and verified BBRAM (Figure 45).


COM8:115200baud - Tera Term VT
File Edit Setup Control Window Help

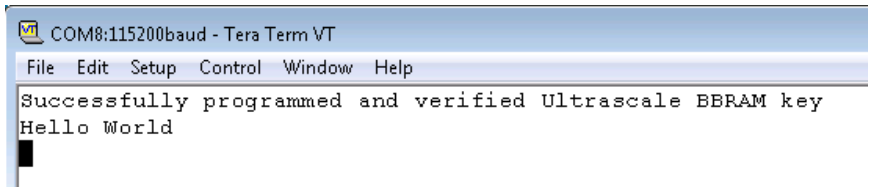
```
Successfully programmed and verified Ultrascale BBRAM key
```

X18361-011117

Figure 45: **Programmed and Verified BBRAM Key**

15. Press and release the **PROGRAM_B** switch **SW4** on the KCU105 board (**SW5** on the KCU116 board).
16. Right-click the **xcku040_0** device (or **xcku5p_0** device) in the center hardware window. Select **Program Device,** set **Bitstream file: <path_to_bitstreams>/`hello_encrypted.bit`**, and click **Program**.
17. When the bitstream finishes loading, the DONE LED (DS34 for the KCU105 board and DS32 for the KCU116 board) turns ON because the BBRAM key is loaded internally by the last bitstream (ignore the errors in the Vivado Tcl Console).

18. The serial terminal window displays a Hello World message that is appended to the last message. This indicates that the `prog_bbram.bit` bitstream (loaded earlier), wrote the BBRAM key successfully (Figure 46).


COM8:115200baud - Tera Term VT
File Edit Setup Control Window Help

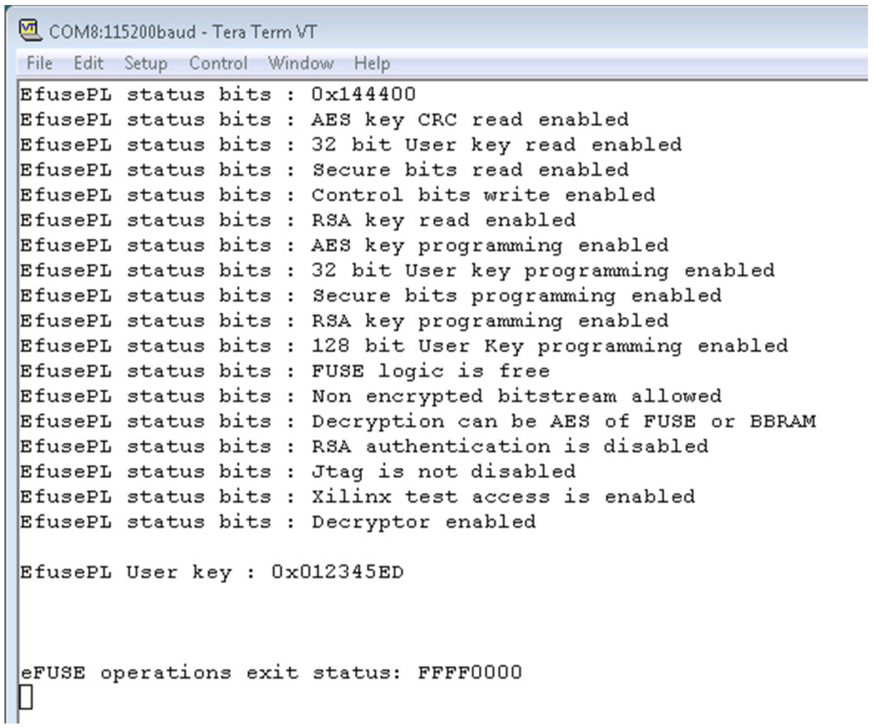
```
Successfully programmed and verified Ultrascale BBRAM key
Hello World
```

Figure 46: **Hello World**

19. Press and release the **PROGRAM_B** switch **SW4** on the KCU105 board (**SW5** on the KCU116 board).
20. Clear the serial terminal program screen.
21. If you instantiated the optional JTAG_MONITOR Module, connect an oscilloscope probe to J53-2 (PACKAGE_PIN AM16) for the KCU105 board and J87-1 (PACKAGE_PIN D13) for the KCU116 board. This is the EFUSE_PROG_PULSE signal that mirrors the actual eFUSE programming phase. You will see a 5 µs pulse for each eFUSE bit that is programmed from a 0 to a 1.
22. Right-click on the **xcku040_0** device (or **xcku5p_0** device) in the center hardware window. Select **Program Device**, set **Bitstream file: <path_to_bitstreams>/`prog_efuse.bit`**, and click **Program**.

    Note: The `prog_efuse.bit` file is not encrypted. It can be encrypted (if required) using the BBRAM key because the key is already loaded in the device.
23. When the bitstream finishes loading, the DONE LED (DS34 for the KCU105 board and DS32 for the KCU116 board) turns ON (ignore the errors in the Vivado Tcl console).

24. The serial terminal window displays a message for a successfully programmed eFUSE. The 32-bit FUSE_USER register contains the value that you entered earlier in the header file for XSK_EFUSEPL_USER_KEY (Figure 47).


COM8:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
EfusePL status bits : 0x144400
EfusePL status bits : AES key CRC read enabled
EfusePL status bits : 32 bit User key read enabled
EfusePL status bits : Secure bits read enabled
EfusePL status bits : Control bits write enabled
EfusePL status bits : RSA key read enabled
EfusePL status bits : AES key programming enabled
EfusePL status bits : 32 bit User key programming enabled
EfusePL status bits : Secure bits programming enabled
EfusePL status bits : RSA key programming enabled
EfusePL status bits : 128 bit User Key programming enabled
EfusePL status bits : FUSE logic is free
EfusePL status bits : Non encrypted bitstream allowed
EfusePL status bits : Decryption can be AES of FUSE or BBRAM
EfusePL status bits : RSA authentication is disabled
EfusePL status bits : Jtag is not disabled
EfusePL status bits : Xilinx test access is enabled
EfusePL status bits : Decryptor enabled

EfusePL User key : 0x012345ED


eFUSE operations exit status: FFFF0000
```

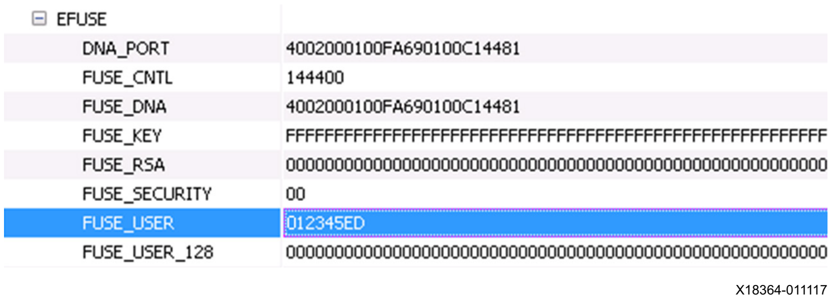
X18363-011117

Figure 47: **Programmed eFUSE**

25. The eFUSE can be further verified using the Vivado Hardware Manager.
26. Press and release the **PROGRAM_B** switch **SW4** on the KCU105 board (**SW5** on the KCU116 board).
27. In the center hardware window, right-click the **xcku040_0** device (or **xcku5p_0** device) > **Refresh Device**.
28. In the center hardware window, right-click the **xcku040_0** device (or **xcku5p_0** device) > **Hardware Device Properties**.

Send Feedback

29. Click the **Properties** tab in the hardware device properties window (directly below the hardware window), expand the properties tree at **REGISTER,** and **EFUSE**. The FUSE_USER field contains the recently programmed eFUSE value (Figure 48).



EFUSE	
DNA_PORT	4002000100FA690100C14481
FUSE_CNTL	144400
FUSE_DNA	4002000100FA690100C14481
FUSE_KEY	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FUSE_RSA	00000000000000000000000000000000000000000000000000000000
FUSE_SECURITY	00
FUSE_USER	012345ED
FUSE_USER_128	00000000000000000000000000000000000000000000000000000000

X18364-011117

Figure 48: **FUSE_USER**

Conclusion

This application note describes a procedure to internally program the BBRAM and eFUSE memory of an UltraScale and UltraScale+ FPGA. Internal programmability is valuable, especially in the context of security-critical applications. The method presented takes advantage of the MASTER_JTAG primitive, an existing MicroBlaze-based hardware example design, AMD-supplied HDL modules, a software library (XILSKEY), and software examples that are used to create functional applications. The procedure explained in this application note modifies and enhances examples to fit various use models including a tamper event driven model to impose penalties and a field update of a BBRAM AES key over a secure link.

Reference Design

You can download the reference design files for this application note from the AMD website.

Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Ed Peterson
Target devices	UltraScale and UltraScale+ FPGAs
Source code provided	Yes
Source code format	VHDL
Design uses code and IP from existing AMD application note and reference designs or third party	No
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No (Verification using ILA debugger)
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado design suite 2016.3 for UltraScale FPGAs Vivado design suite 2017.3 for UltraScale+ FPGAs
Implementation software tools/versions used	Vivado design suite 2016.3 for UltraScale FPGAs Vivado design suite 2017.3 for UltraScale+ FPGAs
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KCU105 Evaluation Board for UltraScale FPGAs KCU116 Evaluation Board for UltraScale+ FPGAs

References

1. MicroBlaze Soft Processor Core
2. *UltraScale Architecture Configuration User Guide* (UG570)
3. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
4. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
5. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
6. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
7. *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs Application Note* (XAPP1098)
8. *Standalone Library Documentation: BSP and Libraries Document Collection* (UG643)
9. AMD Kintex UltraScale FPGA KCU105 Evaluation Kit
10. Vivado Design Suite
11. Xilinx Software Development Kit
12. Tera Term https://teratermproject.github.io/index-en.html
13. AMD Kintex UltraScale+ FPGA KCU116 Evaluation Kit
14. Soft Error Mitigation Core
15. UltraScale device data sheets:
    - *UltraScale Architecture and Products Overview* (DS890)
    - *Kintex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS892)
    - *Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS893)
16. Security Monitor IP Core Product Brief
17. *KCU105 Quick Start Guide* (XTP391)
18. *KCU105 Board User Guide* (UG917)
19. *KCU116 Evaluation Kit Quick Start Guide* (XTP471)
20. *KCU116 Evaluation Board User Guide* (UG1239)
21. *Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide* (UG953)
22. *Programming BBRAM and eFUSEs* (XAPP1319)
23. Vitis Unified Software Platform
24. Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream (XAPP1267)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/21/2025 Version 1.2.1	
General updates	Editorial updates only. No technical content updates.
07/31/2020 Version 1.2	
Added support for stacked silicon interconnect (SSI) technology devices and the Zynq UltraScale+ programmable logic (PL). Added new link for Using Encryption and Authentication to Secure an UltraScale + FPGA Bitstream XAPP1267. Added Figure 24 that exemplifies xilskey settings.	
02/15/2018 Version 1.1	
Added support for UltraScale+ FPGAs.	
03/08/2017 Version 1.0	
Initial AMD release.	N/A

Please Read: Important Legal Notices