QSPI Flash Memory Bootloading In Standard SPI Mode with KC705 Platform

Summary: KC705 platform has nonvolatile QSPI flash memory. It can be used to configure FPGA and store application image. This tutorial describes the method to store bistream and application image to qspi flash with Xilinx tools. And how to create a bootloader to copy the application image from flash memory to DDR3 memory and run it.

Included Systems: A reference design with standard SPI mode is attached. A custom IP and driver is built into this example, and the user application drives LED as the setting of DIP switch with the custom IP and its driver.

Hardware And Software Requirement:

- Xilinx ISE and EDK software, version 14.4
- Xilinx KC705 platform
- Digilent cable for JTAG
- Usb mini serial cable for UART

Objectives: The objectives of this design are:

- In XPS generate a simple BSB design with custom IP and driver.
- Export to SDK, and create an user application that uses the custom driver API to read and write to the custom IP.
- Use SDK utility to create application image bin file, and store the bitstream and application bin file to flash.
- Create a bootloader that will take the application from flash and copy it to DDR and execute from DDR.

Reference design:

Generating hardware

This section describes the creation of hardware design. The user can skip this step to "Generating application bin file". There is pre-built hardware in directory < design directory >\ image_translate.

To implement the embedded design and export it to SDK:

- Start XPS and open the embedded project at <design directory>\kc705_qspi_bootload\system.xmp.
- 2. Export the hardware project to SDK by selecting project >export_hardware_design_to_SDK. Check the Include bitstream and BMM file and click the Export & Launch SDK button. At this point, XPS exports the embedded system configuration via a system.xml file that is used by SDK to understand what peripherals are present in the design and what the base addresses are. The

file is automatically exported to <design directory>\kc705_qspi_bootload\SDK\SDK_Export\hw. After all finished, SDK opens a dialog box asking where the workspace is located. Create a new folder <design directory>\kc705_qspi_bootload\software. Browse to and select the directory <design directory>\kc705_qspi_bootload\software, click OK. Click OK again.

Generating application

Configuring SDK

After XPS exported and launched SDK, a hardware platform is added to SDK workspace. And the platform folder kc705_qspi_bootload_hw_platform is created in <design directory>\kc705_qspi_bootload\software.

Copy the folder < design directory>\kc705_qspi_bootload\drivers to < project directory>\kc705_qspi_bootload\software\kc705_qspi_bootload_hw_platform.

- 1. Start **SDK** and **open** the workspace at <design directory>\kc705_qspi_bootload\software. This step is not necessary if XPS was used with **Export & Launch SDK**.
- 2. Point SDK to the included repository that contains the custom driver(Figure 1):
 - Select Xilinx_tools > repositories.
 - Select **Relative** for local repositories.
 - Select folder kc705_qspi_bootload_hw_platform and click OK.
 - Click **Rescan Repositories** and click **OK**.

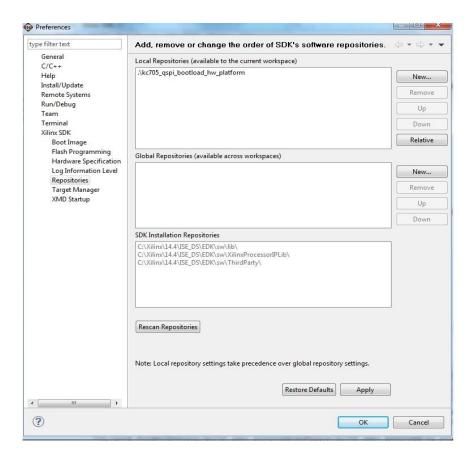


Figure 1

Creating user application

The application has already been compiled and is available at <design directory>\ image_translate

- 1. Select File > New > board_support_package.
- 2. Click Finish.
- 3. Select File > new > application_project.
- 4. Enter the project name user_application.
- 5. Change board support package to **Use existing** and select the BSP just created.
- 6. Click Next.
- 7. Choose the **Empty Application** template.
- 8. Click Finish.
- 9. In SDK's Project Explorer tab, expand **user_application** and right click on the **src** folder.
- 10. Select Import.
- 11. Select General > File_System.
- 12. Click Next.
- 13. **Browse** to and **select** the included directory <design directory>\src\apps\user_application.
- 14. In the left window pane, check the **user_application** folder.
- 15. Click Finish.

16. Select **yes** to overwrite lscript.ld.

After SDK completes compiling this application, the ELF is available at

<design

directory>\kc705_qspi_bootload\software\user_application\Debug\user_applicatio
n.elf.

Creating bootloader application

- 1. Select File > new > application_project.
- 2. Enter the project name bootloader.
- 3. Change board support package to **Use existing** and select the BSP just created.
- 4. Click Next.
- 5. Choose the **Empty Application** template.
- 6. Click Finish.
- 7. In SDK's Project Explorer tab, expand **bootloader** and right click on the **src** folder.
- 8. Select Import.
- 9. Select General > File_System.
- 10. Click Next.
- 11. Browse to and select the included directory <design directory>\src\apps\bootloader.
- 12. In the left window pane, check the **bootloader** folder.
- 13. Click Finish.
- 14. Select yes to overwrite lscript.ld.

After SDK completes compiling this application, program FPGA with the generated ELF of bootloader.elf. A new bitstream download.bit including bootloader in bram will be generated in directory <design directory>\kc705_qspi_bootload\software\kc705_qspi_bootload_hw_platform. Copy the user_application.elf and download.bit to <design directory>\ image_translate. There is prebuilt download.bit in directory <design directory>\ image_translate.

Generating application bin file

Open a ISE Command Prompt and **set** the directory to **<design directory>\ image_transfer**.

We can use mb-objcopy to generate bin files with application ELF image. Because the application's VECTOR sections locate in bram and the rest sections locate in DDR3 memory, we need separate the ELF image to two bin files.

To generate the bin file **vector_section.bin** for VECTOR sections, use below command:

mb-objcopy -O binary -j .vectors.reset -j .vectors.sw_exception -j .vectors.interrupt -j .vectors.hw_exception user_application.elf vector_section.bin

To generate the bin file **rest_section.bin** for the rest sections, use below command:

mb-objcopy -O binary -R .vectors.reset -R .vectors.sw_exception -R .vectors.interrupt -R .vectors.hw_exception user_application.elf rest_section.bin

If user wants to load custom application from flash to DDR3, this flow can generate bin files. We need know the byte numbers in these bin files, because the parameters **VECTOR_SECTION_BYTE_NUM** and **REST_SECTION_BYTE_NUM** in bootloader.c need be modified according to the byte numbers. We can open the bin files with some tools, for example UltraEdit, and count the byte numbers of the two bin files.

Storing bistream and bin image to flash

By now, we have **download.bit vector_section.bin** and **rest_section.bin**, we can use impact to generate a MCS image with them and program it to flash.

- Open iMPACT.
- 2. Double click Create PROM File(Figure 2).

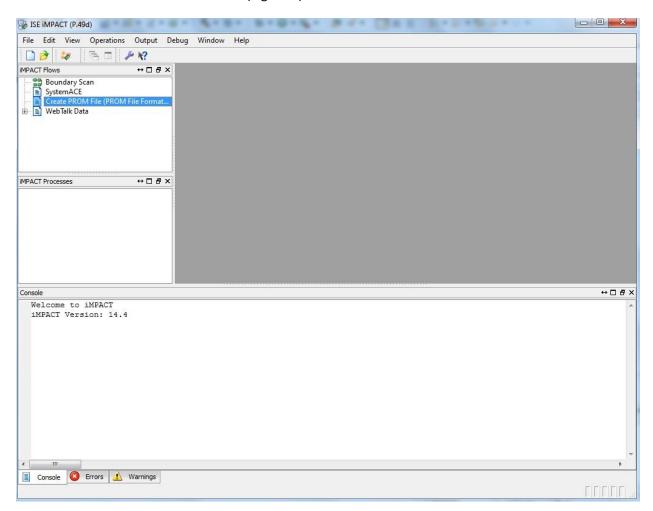


Figure 2

File Edit View Operations Output Debug Window Help IMPACT Flows ↔□♂× SystemACE
Create PROM File (PROM File Format... ⊕ WebTalk Data PROM File Formatter Select Storage Target Step 2. Add Storage Device(s) Step 3. **Enter Data** Step 1. General File Detail Storage Device Type: Value Storage Device (bits) 128M 🔻 **iMPA** Xilinx Flash/PROM Checksum Fill ■ Non-Volatile FPGA Add Storage Device Remove Storage Device Spartan3AN Output File Name | bootloader SPI Flash 128M Configure Single FPGA Output File C:\work_folder\project\image_tra Configure MultiBoot FPGA Location BPI Flash
 Configure Single FPGA Flash/PROM File Property Configure MultiBoot FPGA Configure from Paralleled PROMs -File Format Generic Parallel PROM Add Non-Configuration Data Files Yes Cons ₽× Auto Select PROM In this step, you will enter information to assist in setting up and generating a PROM file for the targeted storage device and mode. • Checksum Fill Value: When data is insufficient to fill the entire memory of a PROM, the value specified here is used to calculate the checksum of the unused portions. Output File Name: This allows you to specify the base name of the file to which your PROM data will be written
 Output File Location: This allows you to specify the directory in which the file named above will be created Cancel Help Console Errors Marnings

Select Configure Single FPGA → 128M → MCS with non-configuration data files YES(Figure 3).

Figure 3

- 4. Click OK
- 5. Add download.bit rest_section.bin and vector_section.bin(Figure 4). The start address of rest_section is 0xB00000, and vector_section is 0xC00000. If user changes the address, the parameter REST_SECTION_START_ADDR and VECTOR_SECTION_START_ADDR in bootloader.c need be modified accordingly.

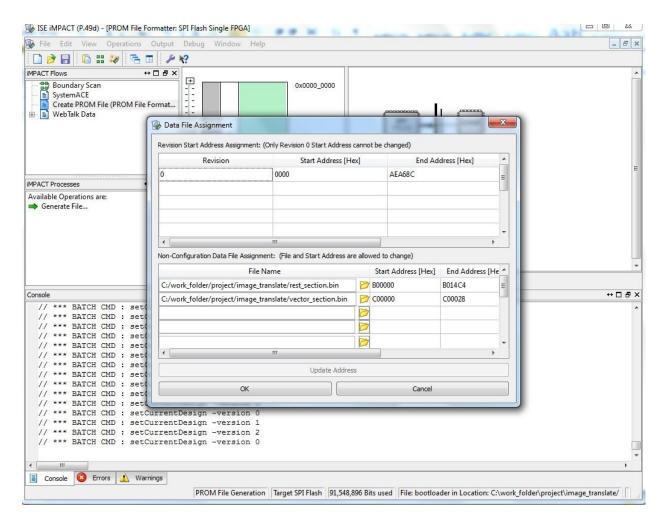


Figure 4

6. Click **Generate File**(Figure 5).

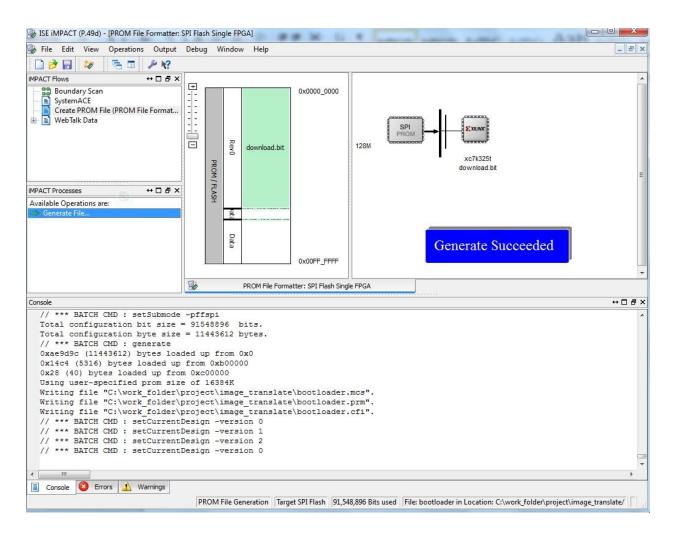


Figure 5

7. Add the generated MCS file to SPI flash(figure 6), and click OK.

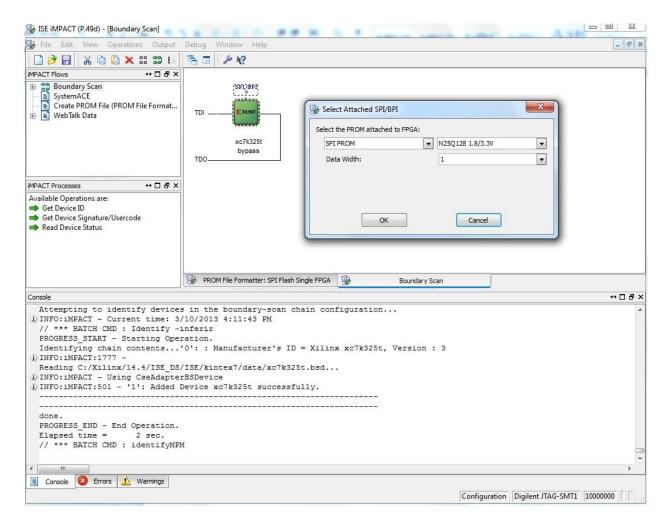


Figure 6

8. **Program** the flash.

Running the design

We have downloaded the whole design into flash now. Make sure pin 5(M0) is switched to 1 and USB mini serial cable is connected. Open Tera-Term(or other terminal application) and configure it with 9600 baud rate. Switch on the power of board. We will see the message in figure 7. And led will be lighting as the setting of dip switch.

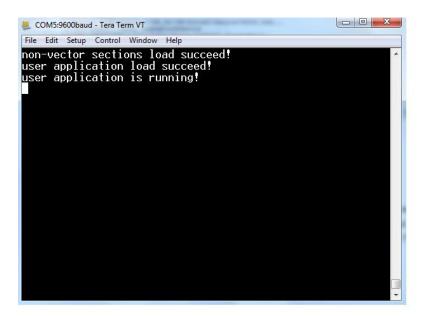


Figure 7

Debugging the design

SDK can debug application running. Here is an example of bootloader application.

- 1. Start debugging bootloader application:
 - a. In the SDK project explorer window, right click **bootloader** and select **debug_as >debug_configurations(Figure 8).**

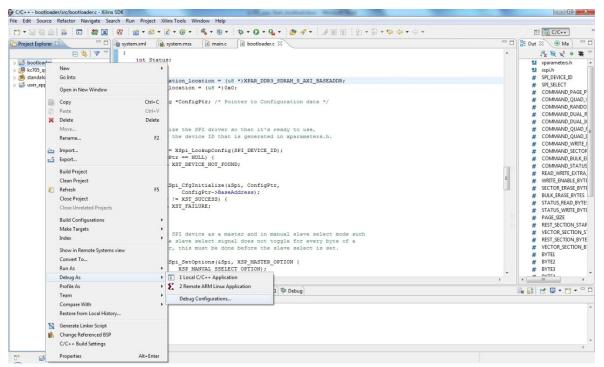


Figure 8

b. Highlight Xilinx C/C++ ELF and select the New launch configuration icon at the top left(Figure 9).

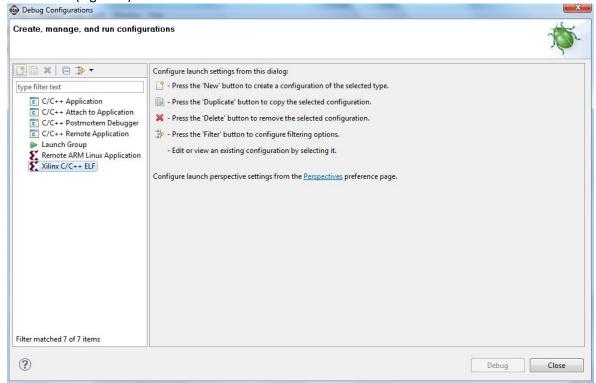


Figure 9

Debug Configurations Create, manage, and run configurations Name: bootloader Debug type filter text Main Source 🗞 Device Initialization 🍫 STDIO Connection 📓 Remote Debug 📓 Debugger Options 🔲 Common C/C++ Application C/C++ Application: C/C++ Attach to Application Search Project... Browse... Debug\bootloader.elf C/C++ Postmortem Debugger C/C++ Remote Application Project: Launch Group bootloader Browse... Remote ARM Linux Application
Xilinx C/C++ ELF Build (if required) before launching & bootloader Debug Build configuration: Debug Enable auto build Disable auto build Use workspace settings Configure Workspace Settings... $\ensuremath{ \ensuremath{ \begin{tabular} \ensuremath{ \ensuremath{ \begin{tabular} \ensuremath{ \ensuremath{ \begin{tabular} \ensuremath{ \ensure$ Apply Revert Filter matched 8 of 8 items ? Debug Close

c. The configuration name is automatically set to **bootloader Debug**(Figure 10).

Figure 10

- d. Click **Debug**. Click **Yes** to confirm the perspective switch.
- e. In the debug perspective, set two breakpoints at line 225 and 240. At line 225 rest section data is copied to DDR3 memory address 0xC0000000. At line 240, vector section data is copied to address 0x00000000. In Memory tab at bottom, create views for 0x00000000 and 0xC0000000(Figure 11).

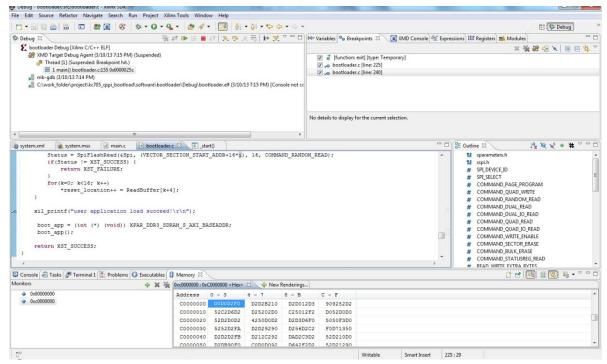


Figure 11

f. Click Resume button, the breakpoint at line 225 will be hit. The memory view for 0xC0000000 will change. Compare the data with rest_section.bin file, user can find out if any data is copied incorrectly(Figure 12).

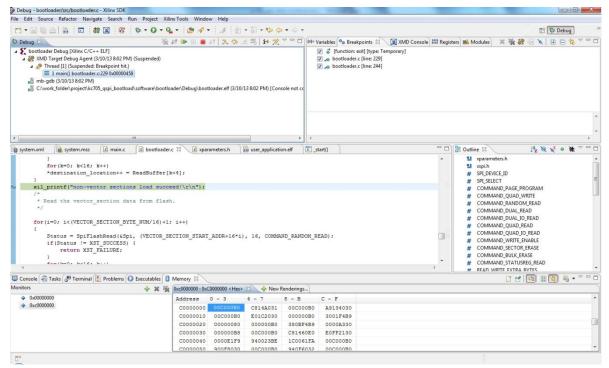


Figure 12

g. Click **Resume** button, the breakpoint at line 240 will be hit. The memory view for 0x00000000 will change. Compare the data with **vector_section.bin** file, user can find out if any data is copied incorrectly(Figure 13).

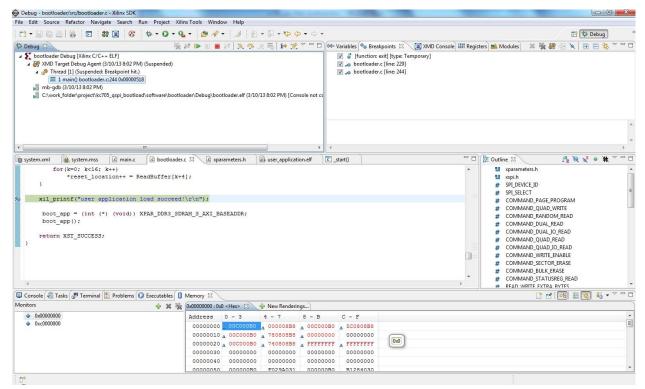


Figure 13