Optimizing the use of an SPI Flash PROM in Microblaze-Based Embedded Systems

Ahmed Hanafi
University Sidi Mohammed Ben Abdellah
Fès, Morocco

Mohammed Karim
University Sidi Mohammed Ben Abdellah
Fès, Morocco

***Abstract*—This paper aims to simplify FPGA designs that incorporate Embedded Software Systems using a soft core Processor. It describes a simple solution to reduce the need of multiple non-volatile memory devices by using one SPI (Serial Peripheral Interface) Flash PROM for FPGA configuration data, software code (Processor applications), and miscellaneous user data. We have thus developed a design based on a MicroBlaze soft processor implemented on a Xilinx Spartan-6 FPGA SP605 Evaluation Kit. The hardware architecture with SPI flash was designed using the Xilinx Platform Studio (XPS) and the software applications, including the bootloader, was developed with Xilinx Software Development Kit (SDK). ISE Design Tools prepared by Xilinx Company, is employed to create the files used to program flash memory which are SREC (S-record) file associated with software code, Hexadecimal file for user data, and bootloader file to configure the FPGA and allows software applications stored in flash memory to be executed when the system is powered on. Reading access to the SPI Flash memory is simplified by the use of Xilinx In-System Flash (ISF) library.**

***Keywords*—Microblaze; ISF; Bootloader; SREC; Configuration; Bitstream SPI Flash**

I. Introduction

SRAM-FPGAs such as Spartan-6 from Xilinx are configured by loading application-specific configuration data (bitstream) into internal memory (CMOS Configuration Latches), then it must be reconfigured after it is powered down. Spartan-6 can configure itself from an external nonvolatile memory device or they can be configured by an external smart source, such as a DSP processor, or microcontroller. In the Master Mode configuration [1], the FPGA automatically loads itself with configuration data from an external SPI (Serial Peripheral Interface) Flash PROM as shown in figure 1.

On the other hand, many FPGA designs in space applications, automobiles, medical field and industrial control system, incorporate Embedded Software Systems using soft core Processor such as Microblaze and utilize external volatile memory to execute software code. This kind of system must also include a non-volatile memory to store the software code and small amounts of user data.

To simplify system design and reduce cost and consumption, we propose a Microblaze system that stores software code, user data, and configuration data in one SPI Flash device.

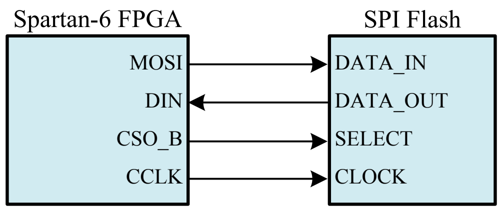


Fig. 1. Master Serial/SPI Mode with SPI Flash memory.

The figure 2 shows the memory organization used to store multiple blocks of data in the SPI flash PROM:

- The configuration section including the bitstream and the bootloader must be stored at address 0x0.
- The software application section can be anywhere in the SPI Flash based on the size of the bitsteam file. The software section start address will be usefull for the boot loader to work.
- The user data section is defined by a synchronization word followed by data. The synchronization word is used because the space memory between software application section and user data section is random.

Our work is based on the concepts described in documentation of Xilinx and Avnet, namely, how to create a Microblaze design with AXI (Advanced eXtensible Interface) system [2][3][4] and how to bootload a software application in BPI (Byte Peripheral Interface) configuration mode and SPI configuration mode [5][6].

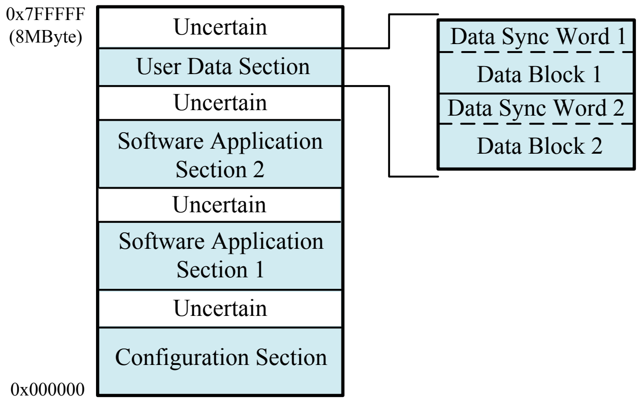


Fig. 2. SPI Flash memory map.

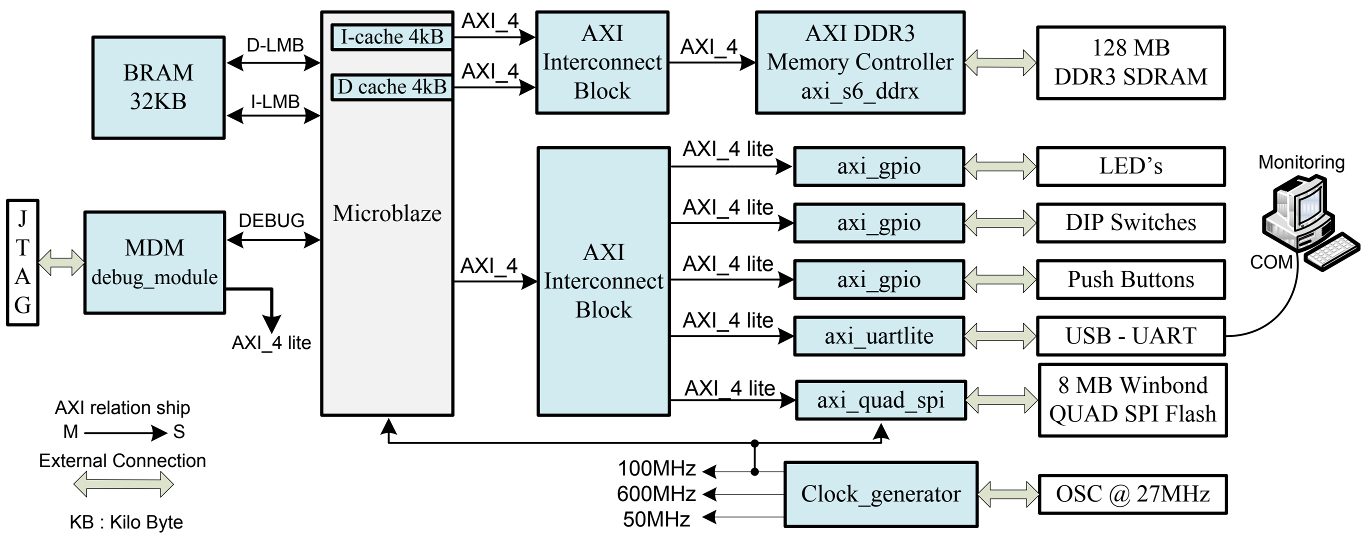


Fig. 3. Reference System Hardware Platform.

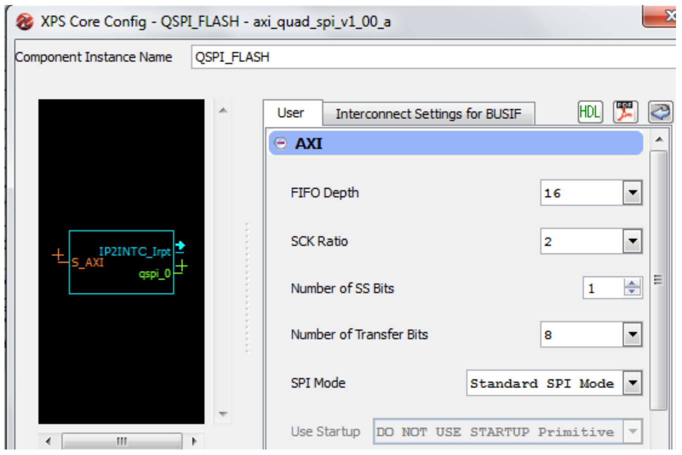
Our contribution lies in:

- Defining software flows for appending a PROM file with multiple software sections and one user data section with separate blocks of data for each application.
- Improving the use of Xilinx In-System Flash (ISF) library to read software data and user data from SPI Flash.

II. Hardware Design

The MicroBlaze reference system is implemented on the Xilinx Spartan-6 FPGA SP605 Evaluation Kit using Xilinx Platform Studio (XPS) Tool Suite provided by the Embedded Development Kit (EDK) of Xilinx. The architecture shown in figure 3 is created using the Base System Builder (BSB) wizard within XPS and it is based on the AXI interconnect, which operates at 50 MHz. Our design includes the main following IPs cores:

- DDR3 SDRAM interface (axi_s6_ddrx) operating at 600MHz. The on-board SDRAM will contain the application program chosen and copied from SPI Flash by bootloader.
- SPI Flash interface (axi_quad_spi) operating at 100MHz. The used Winbond SPI Flash memory W25Q64BV can run up to 80MHz in standard SPI mode, then we changed the C_SCK_RATIO parameter in AXI Quad SPI core to 2 as described in [7] (see figure 4). This will run SPI Flash at 50MHz.
- General Purpose Input Output (axi_gpio) operating at 50MHz and used to communicate with LEDs, DIP switches and Push Buttons. DIP switches will trigger the download and execution of one of the software codes stocked in SPI Flash memory.
- On-chip dual-port blocks RAM (BRAM) using to store the bootloader file. It is designed to be small not to exceed the BRAM limits.



XPS Core Config - QSPI_FLASH - axi_quad_spi_v1_00_a

Component Instance Name QSPI_FLASH

User | Interconnect Settings for BUSIF

AXI

FIFO Depth	16
SCK Ratio	2
Number of SS Bits	1
Number of Transfer Bits	8
SPI Mode	Standard SPI Mode
Use Startup	DO NOT USE STARTUP Primitive

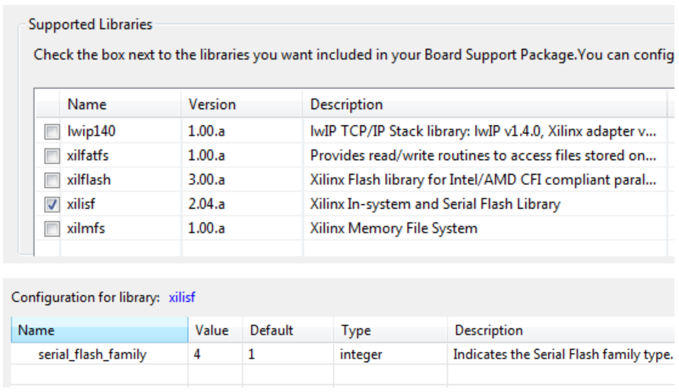
Fig. 4. Updating SPI Interface Clock Rate.

This hardware platform has been exported to SDK to be the basis of all software work.

III. Software Applications And Their Flash Images

The software applications consist of two C projects based on the Peripheral test application template provided in Software Development Kit (SDK). The template will be customized as follows:

- For both applications, we used the same Board Support Package (BSP) after having added and configured the ISF library as shown in figure 5 and described in [8].
- The difference between the two applications lies in the messages sent via UART and the blinking of LEDs.
- The linker script of each application project was updated to allow them to be executed from DDR3 SDRAM memory.
- In each application, we used a set of functions to handle the interaction with the corresponding block of user data. This part will be detailed in section 5 of this paper.


Supported Libraries

Check the box next to the libraries you want included in your Board Support Package.You can config

	Name	Version	Description
☐	lwip140	1.00.a	lwIP TCP/IP Stack library: lwIP v1.4.0, Xilinx adapter v...
☐	xilfatfs	1.00.a	Provides read/write routines to access files stored on...
☐	xilflash	3.00.a	Xilinx Flash library for Intel/AMD CFI compliant paral...
☑	xilisf	2.04.a	Xilinx In-system and Serial Flash Library
☐	xilmfs	1.00.a	Xilinx Memory File System

Configuration for library: xilisf

Name	Value	Default	Type	Description
serial_flash_family	4	1	integer	Indicates the Serial Flash family type.

Fig. 5. Setting ISF Library with SPI Flash family.

Then, we create a flash image of each application project that we can store in SPI Flash, and we chose the S-record (SREC) file format. This is the most flexible format since it consists of ASCII character strings, specially formatted for loading software data into memory. This format also allows using the bootloader application template provided in SDK. mb-objcopy command is used to generate a SREC file from the (executable file) ELF of each application as follows:

```
mb-objcopy -O srec  Application.elf  Application.srec
```

IV. The Bootloader File

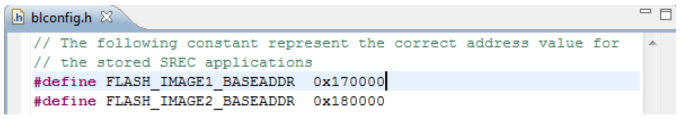
The hardware bitstream including the bootloader is stored in SPI Flash memory and used to configure the Spartan-6 FPGA. After the FPGA configuration, the processor starts executing the bootloader that will select and copies the executable software from a pre-determined location in SPI Flash to DDR3 SDRAM (Depending on the condition of the DIP switches).

Based on the Spartan-6 Configuration User Guide [1], bitstream file of the used Spartan-6 LX45T has a size of 11,939,296 bits (1458 KB). The Winbond SPI Flash contains 2048 x 4KB sectors, then the bit file will use the first 365 sectors of the flash and the software data sections can be stored from the 366th sector (from the @ 0x16E000).

SDK provides a bootloader template which is used for parallel NOR Flash memory. Based on Xilinx documentation [8] and Avnet tutorial [6], we modified this template to read from SPI Flash using the Xilinx In-System Flash (ISF) library. The SPI drivers can be used in polled mode or interrupted mode if interruption was enabled in the AXI Quad SPI core.

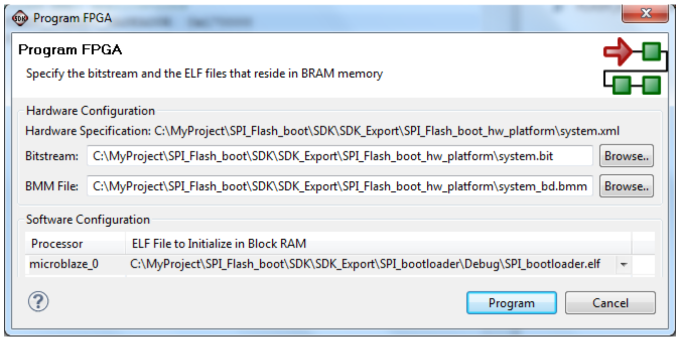
In order to create the bitstream file including configuration data and the bootloader executable file (ELF), the following steps are adopted:

- We modified bootloader.c, the main source file in project, with the initialization of SPI device and ISF library, and the update of Flash memory reads with ISF Read commands.
- We updated the flash memory offsets for the stored SREC applications in the blconfig.h file (figure 6).
- The linker script was updated to allow the boot loader to be executed from FPGA BRAMs.


blconfig.h

```
// The following constant represent the correct address value for
// the stored SREC applications
#define FLASH_IMAGE1_BASEADDR  0x170000
#define FLASH_IMAGE2_BASEADDR  0x180000
```

Fig. 6. The blconfig.h file.


Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Specification: C:\MyProject\SPI_Flash_boot\SDK\SDK_Export\SPI_Flash_boot_hw_platform\system.xml

Bitstream: C:\MyProject\SPI_Flash_boot\SDK\SDK_Export\SPI_Flash_boot_hw_platform\system.bit

BMM File: C:\MyProject\SPI_Flash_boot\SDK\SDK_Export\SPI_Flash_boot_hw_platform\system_bd.bmm

Software Configuration

Processor	ELF File to Initialize in Block RAM
microblaze_0	C:\MyProject\SPI_Flash_boot\SDK\SDK_Export\SPI_bootloader\Debug\SPI_bootloader.elf

Fig. 7. Program FPGA window.

Finally, we generate the configuration file with the embedded bootloader by running the data2mem application in command line mode. This command provides the facility to iterate new block RAM data into a bit file without the need to rerun Xilinx implementation tools. In conjunction with a new ELF file and the Block RAM Memory Map (BMM) file, data2mem command (described in [9]) updates the block RAM initialization in a BIT file image and outputs a new bit file. This facility is invoked as follows :

```
data2mem -bm my.bmm -bd code.elf -bt my.bit -o b new.bit
```

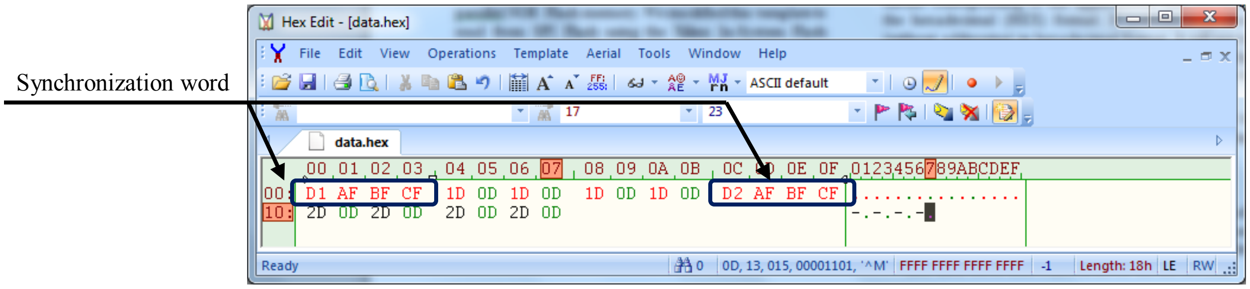
Another possibility is to run the download step in SDK. The program operation (see figure 7) will generate the new bit file (download.bit) even if the board is not connected.

V. The User Data File

One flash image will be used to regroup the two user data blocks corresponding to the applications, and we chose the hexadecimal (HEX) format. It contains only data (without addressing) in hexadecimal format. It will not be checked by the check sum as it is not really part of the bitstream file or SREC file, and we may do whatever we wish.

We used a hex editor (the shareware HexEdit [10]) to populate the data.hex file with respect for the following specific requirements as shown in figure 8:

- Every data line must be 16 bytes long.
- Every data number must be represented in hex.
- Put synchronization word (4Bytes) at the start of each user data block. In the example of figure 8, we used the synchronization word 0xD1AFBFCF for the first user data block and 0XD2AFBFCF for the second.
- Each block of data must be 8 bytes long. To use blocks of data with variable size, we can add immediately after the synchronization word, a word which indicates the size of the block followed by the data. This word will be recovered by the software application to set up the reading data routine.



```
    00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F 0123456789ABCDEF
00: D1 AF BF CF  1D 0D 1D 0D  1D 0D 1D 0D  D2 AF BF CF ................
10: 2D 0D 2D 0D  2D 0D 2D 0D                           -.-.-.-.
Ready   0   0D, 13, 015, 00001101, '^M'   FFFF FFFF FFFF FFFF   -1   Length: 18h   LE   RW
```

Fig. 8. The data.hex file.

The use of the synchronization words allows storing user data file anywhere in the SPI Flash after the software section, and also allows each application to identify its block of user data. Indeed, the software reads through the SPI Flash 1 Byte at a time until it finds a 32-bit word matching the data synchronization word. Once the data synchronization word is found, the software recovers the Start Address of user data block and the first 8 bytes following the synchronization word. The size of 8 bytes is given by way of example and can be modified depending on the application requirements.

VI. Programming The SPI Flash Device

To program the memory, we have to generate an MCS file (Intel MCS-86 Hexadecimal Format) from the bitstream with boatloader (downoald.bit), the SREC flash images (First_application.srec and Second_application.srec), and the user data file (data.hex).

The MCS file contains ASCII strings that define the storage address and data file. It can be created by the PROMGen command [11] or the iMPACT software. The figure 9 summarize the software flow for creating a MCS file.

We chose iMPACT because it provides a graphical, step-by-step approach. The following steps are followed to create our PROM file:

1) Launch the iMPACT GUI and create a new project with Prepare a PROM file option.

2) In the PROM File Formatter dialog box (figure 10) :

- Step1 : select Configure Single FPGA under SPI Flash in Storage Device type.
- Step 2 : select 64M and click on the Add Storage Device button. The choice depends on the on board SPI Flash memory.
- Step 3 : select a file name, directory location, and Yes for Add Non-Configuration Data Files pull-down.

3) In the next dialog boxes :

- Add one device file by browsing to the your_hw_platform directory and selecting the download.bit file.
- Add the first data file First_application.srec by indicating the start address 0x170000 (must be greater then 0x16E000).

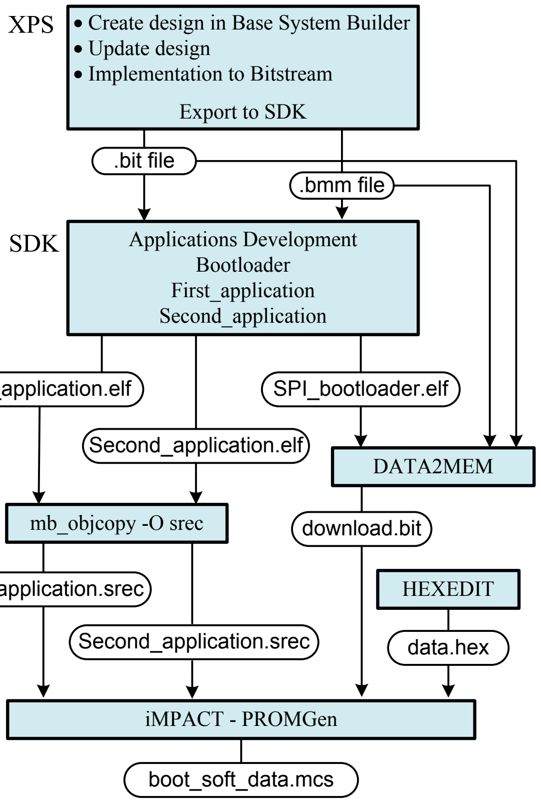


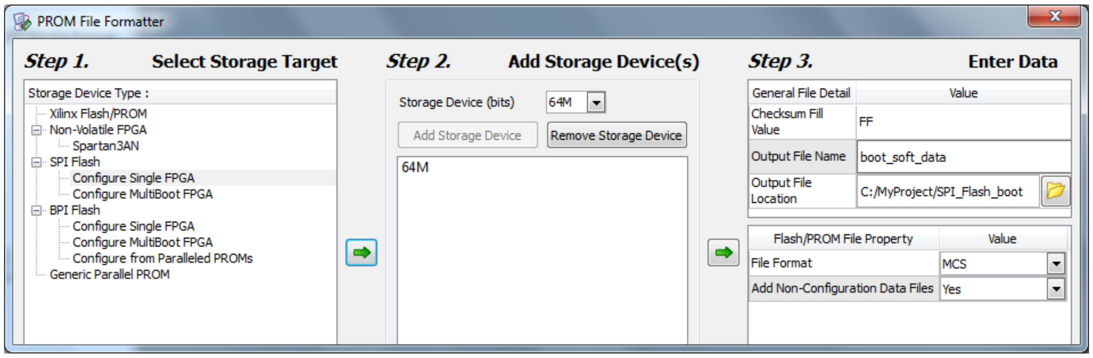
Fig. 9. Software flow for creating a MCS file.

- Add the second data file Second_application.srec by indicating the start address 0x180000 (depends on the size of the first data file).
- Add the third data file data.hex anywhere after the second datafile.

4) After validating the dialog box Data File Assignment shown in figure 11, double-click Generate File to create the MCS file : boot_soft_data.mcs.

To detect the Spartan-6 FPGA of SP605 Evaluation Kit and program the on board SPI Flash memory, the following steps are followed :

- Connect the PC to the USB JTAG connector on the SP605 board, set the mode pins for SPI Flash (M0=1, M1=0), and turn the board power on (see figure 12).



PROM File Formatter

Step 1. Select Storage Target

Storage Device Type :
- Xilinx Flash/PROM
- Non-Volatile FPGA
  - Spartan3AN
- SPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
- BPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
  - Configure from Paralleled PROMs
- Generic Parallel PROM

Step 2. Add Storage Device(s)

Storage Device (bits): 64M

Add Storage Device | Remove Storage Device

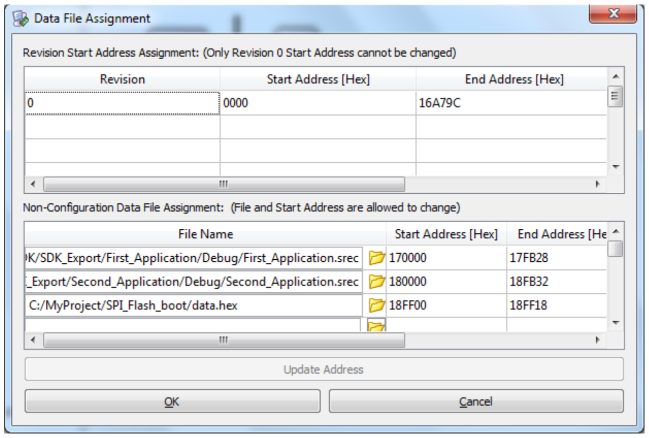
64M

Step 3. Enter Data

General File Detail	Value
Checksum Fill Value	FF
Output File Name	boot_soft_data
Output File Location	C:/MyProject/SPI_Flash_boot

Flash/PROM File Property	Value
File Format	MCS
Add Non-Configuration Data Files	Yes

Fig. 10. PROM File Formatter dialog box.



Data File Assignment

Revision Start Address Assignment: (Only Revision 0 Start Address cannot be changed)

Revision	Start Address [Hex]	End Address [Hex]
0	0000	16A79C

Non-Configuration Data File Assignment: (File and Start Address are allowed to change)

File Name	Start Address [Hex]	End Address [He
K/SDK_Export/First_Application/Debug/First_Application.srec	170000	17FB28
_Export/Second_Application/Debug/Second_Application.srec	180000	18FB32
C:/MyProject/SPI_Flash_boot/data.hex	18FF00	18FF18

Update Address

OK | Cancel

Fig. 11. Data File Assignment dialog Box.

- In iMPACT interface, double-click on the Boundary Scan flow, right-click on the Boundary Scan windows, and select Initialize Chain.
- Right-click on the Xilinx device detected, and select Add SPI/BPI Device. Browse to directory location selected later in the PROM File Formatter dialog box, and select the generated MCS file.
- In the Select Attached SPI/BPI dialogue box, select the on board SPI Flash memory (W25Q64BV/CV).
- To start the programming with the generated MCS file (boot_soft_data.mcs), right-click on the Flash device and select Program.

VII. Results And Discussions

A serial terminal program, such as Tera Term or Hyper Terminal, must be set to view the output of the bootloader and the test applications.

Make sure before starting the test that, the SPI mode is set, the DIP switches is set at 0x0001 (see figure 13) to run the first application test or 0x1000 to run the second application test.

Press the PROG button (see figure 12) every time we want to re-configure the FPGA. The figure 14 shows running results of the boot loader :

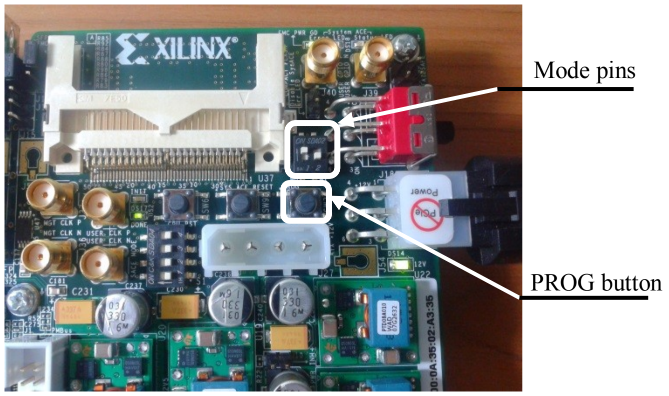


Fig. 12. Configuration mode pins and PROG button.

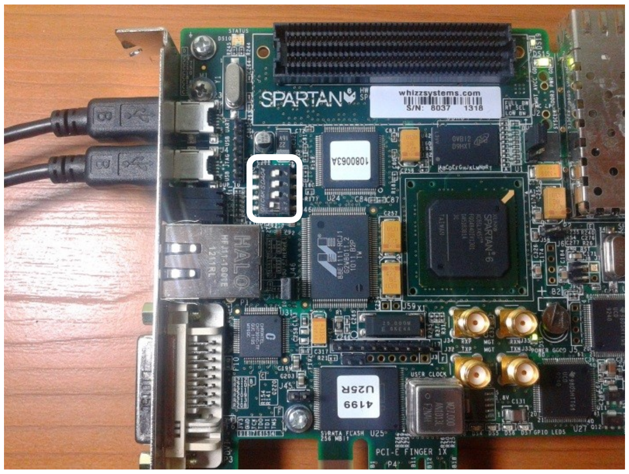
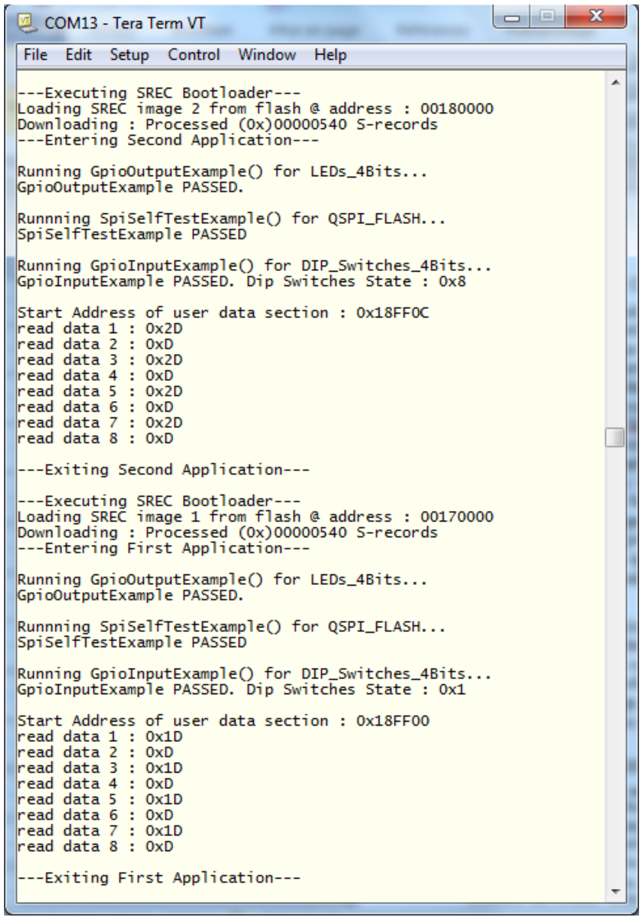


Fig. 13. DIP switches setting for First application test.

- After the FPGA configuration, the bootloader downloads one of the test applications in DDR3 SDRAM memory (depending of DIP switches state).
- The execution of each test application includes peripheral tests, LEDs blinking, and recovery of the specific user data. The start address of application data section and user data block is recovered and printed.

If the DIP switches setting differ from 0x0001 and 0x1000 values, a message is printing to indicate no SREC file to load.



```
---Executing SREC Bootloader---
Loading SREC image 2 from flash @ address : 00180000
Downloading : Processed (0x)00000540 S-records
---Entering Second Application---

Running GpioOutputExample() for LEDs_4Bits...
GpioOutputExample PASSED.

Runnning SpiSelfTestExample() for QSPI_FLASH...
SpiSelfTestExample PASSED

Running GpioInputExample() for DIP_Switches_4Bits...
GpioInputExample PASSED. Dip Switches State : 0x8

Start Address of user data section : 0x18FF0C
read data 1 : 0x2D
read data 2 : 0xD
read data 3 : 0x2D
read data 4 : 0xD
read data 5 : 0x2D
read data 6 : 0xD
read data 7 : 0x2D
read data 8 : 0xD

---Exiting Second Application---

---Executing SREC Bootloader---
Loading SREC image 1 from flash @ address : 00170000
Downloading : Processed (0x)00000540 S-records
---Entering First Application---

Running GpioOutputExample() for LEDs_4Bits...
GpioOutputExample PASSED.

Runnning SpiSelfTestExample() for QSPI_FLASH...
SpiSelfTestExample PASSED

Running GpioInputExample() for DIP_Switches_4Bits...
GpioInputExample PASSED. Dip Switches State : 0x1

Start Address of user data section : 0x18FF00
read data 1 : 0x1D
read data 2 : 0xD
read data 3 : 0x1D
read data 4 : 0xD
read data 5 : 0x1D
read data 6 : 0xD
read data 7 : 0x1D
read data 8 : 0xD

---Exiting First Application---
```

Fig. 14. Terminal Window showing test running

VIII. CONCLUSION AND FUTUR WORKS

This paper demonstrates a method concerning the optimization of the SPI flash memory in FPGA designs that incorporate Embedded Software Systems using Microblaze. It is simple to implement and can be used in any embedded system with limited memory resources.

Successful achievement of this work will encourage us to use an SPI Flash memory as mass memory of our on-board computer (for nano-satellite) that will take an SRAM-FPGA as central processor. The embedded platform can be enriched by proposing a fallback and multiboot technique to create a multiple embedded designed systems (multiple configuration sections), always by using a single SPI Flash memory.

REFERENCES

[1] Xilinx Company "Spartan-6 FPGA Configuration User Guide" *UG380 (v2.5) January 23, 2013*. [internet] Available at
http://www.xilinx.com/support/documentation/user_guides/ug380.pdf

[2] Xilinx Company "MicroBlaze Processor Reference Guide" UG081 (v13.4) January 18, 2012. [internet] Available at
http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/mb_ref_guide.pdf

[3] Xilinx Company "Spartan-6 Family Overview" DS160 (v2.0) October 25, 2011. [internet] Available at
http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[4] Xilinx Company "Embedded System Tools Reference Manual EDK" UG111 (v13.4) January 18, 2012. [internet] Available at
http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/est_rm.pdf

[5] Casey Cain "FPGA Configuration from Flash PROMs on Spartan-3E 1600E Board" XAPP978 (v1.2) November 5, 2010.

[6] Avnet Reference Design "Creating a Microblaze SPI Flash Bootloader" Version 13.2.01 September 22, 2011.

[7] Xilinx Company "LogiCORE IP AXI Quad Serial Peripheral Interface (AXI Quad SPI) V1.00a" DS843 October 19, 2011.
http://www.xilinx.com/support/documentation/ip_documentation/axi_quad_spi/v1_00_a/ds843_axi_quad_spi.pdf

[8] Xilinx Company "OS and Libraries Document Collection" UG643 July 27, 2012. [internet] Available at
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/oslib_rm.pdf

[9] Xilinx Company "Data2MEM User Guide" UG437 (v2.0) April 04, 2007. [internet] Available at
http://www.xilinx.com/itp/xilinx10/books/docs/d2m/d2m.pdf

[10] http://www.hexedit.com/

[11] Xilinx Company "Command Line Tools User Guide" UG628 (v 13.4) January 18, 2012. [internet] Available at
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/devref.pdf