Run-time Fallback and Multiboot Technique for Embedded Platform using Low-Cost Spartan-6 FPGA

AHMED HANAFI, MOHAMMED KARIM
Department of Science and Technology of Information and Communication STIC
Faculty of Sciences Dhar el Mahraz
University Sidi Mohammed Ben Abdellah
B.P 1796, Fès-Atlas, 30003 Morocco.
ahmedhanafi72@gmail.com http://www.fsdmfes.ac.ma/

Abstract: - This paper aims at demonstrating the whole process allowing implementing a robust in-system update solution for Microblaze-based embedded systems using low-cost and low-power consuming Spartan-6 FPGA. In this work, we design a run-time full reconfigurable embedded platform based on the Spartan-6 Multiboot and fallback features. The FPGA Multiboot feature enables switching between two or more configuration files, on the fly (during normal operation), from an external SPI Flash memory. When an error or an interruption is detected during the Multiboot configuration process, the FPGA triggers fallback feature that ensures the configuration with a golden "safe" image.
Embedded development kit (EDK) prepared by Xilinx company is employed to implement the embedded platform on a Spartan-6 evaluation board (i.e., SP605). Based on the Internal Configuration Access Port (ICAP) primitive in the FPGA fabric, we used Xilinx LogiCORE IP AXI HWICAP (Advanced eXtensible Interface Hardware ICAP) core to write software programs that modify the circuit structure and functionality during run-time. This IP Core has been originally designed to support the Run-time Partial Reconfiguration (PR) feature for the Virtex-4, Virtex-5, Virtex-6 family FPGA. Xilinx added support for Spartan-6 family FPGA in 2010 and we decide to use it to facilitate the run-time full reconfiguration process.

Key-Words: - Run-time full reconfiguration, Multiboot, Fallback, ICAP, Microblaze, AXI HWICAP, FPGA

1 Introduction

By exploiting the Multiboot and fallback features of an FPGA such as Spartan-6, the designers can develop dynamic full reconfigurable embedded systems in a wide range of applications that have to limit development cost, include upgrade capability and manage configuration errors (Space and automobile applications, industrial control systems). This kind of application incorporate Embedded Software System based on a soft processor, such as Microblaze, that will be used to achieve and facilitate the self-configuration.

As far as we know, Xilinx is the main world's leading provider of run-time reconfiguration and soft error mitigation features. The low-cost and low-power consuming Spartan-6 family FPGAs provides a dedicated Internal Configuration Access Port (ICAP), which directly interfaces to the configuration memory. This port allows modification of the logic of the device autonomously at run-time. The ICAP_SPARTAN6 primitive, is the hardwired FPGA logic that provides logic access to the Spartan-6 configuration interface, allowing the designer to access configuration registers, readback configuration data, and partially or full reconfigure the device [1]. The work descripted in this paper provides an easy and fast Embedded Software solution to implement the Multiboot and fallback features, using the ICAP primitive.

The paper is organized as follows. In section 2, an overview of some related work on run-time full and partial reconfiguration is provided, followed by the specificities of our work. Section 3 gives a brief description of the Spartan-6 FPGA configuration control logic, and introduces the ICAP interface used for dynamic reconfiguration. In section 4, implementation of Multiboot and fallback features are described, while in sections 5 implementation details (both at hardware and software level) of the proposed run-time full reconfigurable embedded platform are described. Finally, section 6 gives the implementation results, while conclusions and future works are drawn in section 7.

2 Related Works

Most of related works use the ICAP port to the run-time partial reconfiguration. Therefore Koch et al. [2] have demonstrated systems based on Spartan-6 series FPGAs that provide full support for active partial run-time reconfiguration. They have used

ICAP with 16-bits mode at 100 MHz, but instead of relying on a self-reconfiguration, as is the case of this work, they used PC (UART interface) to access the ICAP and control the configuration.

Ming Liu et al. [3] investigate the performance of various ICAP architectures such as OPB_HWICAP and XPS_HWICAP. They use a Virtex-4 FPGA but the ICAP architecture AXI_HWICAP and the full run-time reconfiguration were not included in the study.

Otero et al. [4] design a partial reconfigurable platform based on the Spartan-6 FPGA including two implementation options. The first one is a software solution build on the top of both, software and hardware design XPS_HWICAP. The second one is a hardware solution based on an enhanced HWICAP peripheral core.

For work that focuses on the Multiboot and the full run-time reconfiguration, Khalil and Mohammed [5] use the STARTUP_SPARTAN3 primitive, provided by Xilinx to design a Multiboot embedded system in the Spartan-3E FPGA. But fallback feature was not provided and the self-configuration was achieved with the microcontroller core PicoBlaze.

Our synthesis work proposes to implement the IP core AXI_HWICAP in a Microblaze-based embedded system to achieve a run-time full reconfigurable embedded platform. The practical software solution described for implement Multiboot and fallback features, uses the low-level macros and functions of the AXI_HWICAP core.

3 Reconfiguration Details of Spartan6 Devices

3.1 Spartan-6 Configuration Control Logic

The functionality of each configurable element within the FPGA (CLBs, DSPs or BRAMs) is controlled by configuration data (organized as 16-bit words) stored in the configuration memory. The modification at run-time of its content is done by reading or writing to the configuration registers, using the ICAP_SPARTAN6 primitive which has a separate bus as shown in figure 1.

All configuration data (register writes) used in our implementation is encapsulated into a Type1 packet which contain 2 sections: Header and Data [6]. Table 1 shows the Header section (16-bit word) used in configuration data: only the Write command is used (Code Operation = 10).

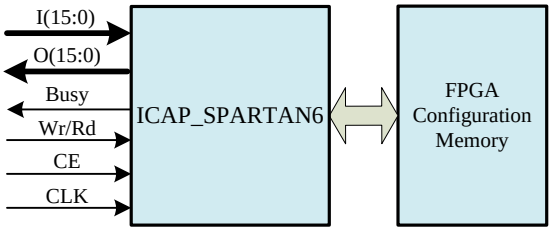

Fig.1: The ICAP primitive on Spartan-6 FPGA

Run-time full reconfiguration implies to provide some configuration commands. Table 2 summarizes the main configuration registers of interest for this purpose:

- COR register set the configuration options.
- MODE_REG register contains the mode setting which can be used for the reboot. We used bitstream mode (required for MultiBoot and Fallback), and buswidth SPI by 1.
- GENERAL1,2 registers set the start address of the Multiboot bitstream.
- GENERAL3,4 registers set the start address of the Golden bitstream.
- CMD register used to perform configuration functions.

Table 1: Type 1 Packet Header

Type 1	Operation	Register Address	Word count
[15:13]	[12:12]	[10:5]	[4:0]
001	10	xxxxxx	xxxxx

Table 2: Used Configuration Registers

Register	Address	Description
CMD	6'h05	Command register
COR2	6'h0B	Set configuration option
HC_OPT_REG	6'h10	House clean option register
GENERAL1	6'h13	Loadable program address (LPA)
GENERAL2	6'h14	LPA and SPI opcode
GENERAL3	6'h15	Golden bitstream address (GBA)
GENERAL4	6'h16	GBA and SPI opcode
MODE_REG	6'h18	Reboot mode

3.2 Xilinx IP Core AXI_HWICAP

In the Xilinx design support, the IP Core AXI_HWICAP is typically used to carry out dynamic and partial reconfiguration of the Spartan-6 FPGAs [7]. As shown in figure 2, this core has the structure of a peripheral that use the ARM AXI (Advanced eXtensible Interface) [8] bus technology as interface with the embedded processor, and integrates the ICAP reconfiguration port together with a finite state machine in charge of generating the necessary control signals. For writes to the ICAP (FPGA reconfiguration), the required configuration data loaded from external memory (SPI Flash

memory) are stored within a Write First In First Out (FIFO), from where it can be sent to the ICAP.

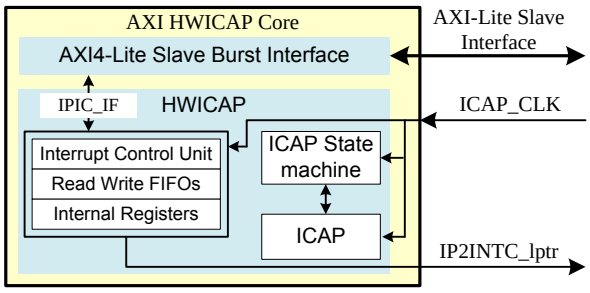

Fig. 2: Block Diagram for the AXI HWICAP

The software application is in charge of sending correct configuration commands to the FIFO. Thus, drivers are available to control the operations of the AXI_HWICAP core from the embedded processor. Functionality included on functions within these drivers will be used to read and write from ICAP.

4 Implementation of Multiboot and Fallback Features

Spartan-6 FPGA has dedicated Multiboot logic, which is used for both fallback and Multiboot reconfiguration. In the chosen Master Serial Configuration Mode, we used the Master Serial Peripheral (SPI) interface to allow the Spartan-6 FPGA to configure itself from a directly attached SPI serial Flash PROM [6]. We employ the on-board Winbond SPI Flash memory of the Xilinx Spartan-6 FPGA SP605 Evaluation Kit. Figure 3 show the connection used for an SPI configuration with data width of x1 and standard SPI mode.

The reconfiguration of FPGA can be triggers after the device is powered up, after the PROGRAM_B pin is pulsed low (SW3 Pushbutton), after the IPROG command (internal PROGRAM_B), or during a fallback retry configuration sequence. In our implementation, there are four images for Multiboot/fallback configuration, stored in the same SPI Flash memory as shown in Figure 4:

- The first image is the Header, a small bitstream who must start at address 0 to trigger the reconfiguration. It contains a set of commands sent to the configuration memory using the ICAP_SPARTAN6 primitive. Table 3 shows the sequence of commands used in our Header file.
- The second image is the fallback or golden bitstream, This image can reside at any address specified in GENERAL3,4 registers.
- The remaining two images are the two Multiboot bitstream. GENERAL1,2 registers values must be set to the location of the next bistream (one of the two images) that the user plans to configure first.

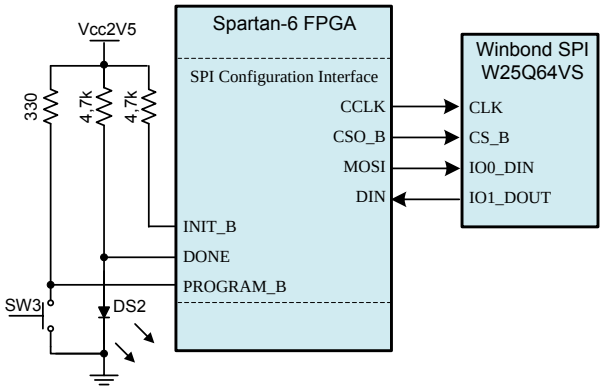

Fig. 3: Spartan-6 FPGA SPI Configuration Interface

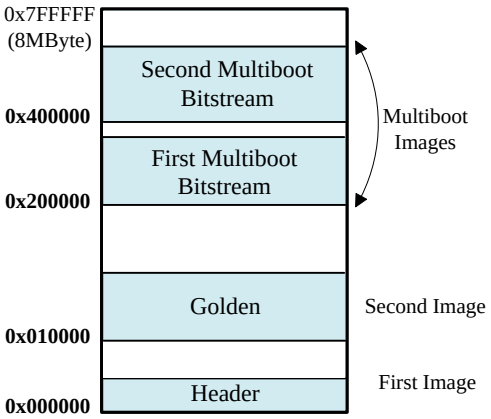

Fig. 4: SPI Flash memory map

Table 3: Header Bitstream for IPROG through ICAP

Configuration data (Hex)		Table Description
FFFF	FFFF	Dummy word
AA99	5566	Synchronization word
31E1	FFFF	Enable Reset on Error : COR2 register
3261 3281	**xxxx** 03**xx**	Set GENERAL1,2 registers with the start address of multiboot image and the SPI flash memory opcode (0x03 for Winbond)
32A1 32C1	**0000** 03**01**	Set the GENERAL3,4 registers with the start address of golden (fallback) image and the SPI flash memory opcode
3301	2100	Set Reboot mode: MODE_REG register
3201	001F	Do not skip initialization
30A1	000E	Send IPROG command : CMD register
2000	2000	No Operation

During configuration, each image has three "strike" allotted to it. If an error is detected (CRC error or watchdog timer time-out error), the strike count (BOOTSTS register) increments and configuration restart. The sequence of full reconfiguration is:

- At power-up, the configuration memory is cleared and the start address 0 is used during FPGA configuration. The Header image set the REGISTER1,2,3,4 and issues an IPROG command using ICAP.
- The special feature of IPROG is that it not reset the dedicated reconfiguration logic. So, the start address set in REGISTER1,2 (Multiboot image) is used during next reconfiguration instead of the default address (zero).
- If the configuration fallback occurs, the golden bitstream is reached. Its start address is defined by the values of GENERAL3,4 registers.

5 Spartan-6 FPGA Run-Time Full Reconfigurable Embedded Platform

5.1 Hardware Design

The embedded platform is implemented on the Xilinx SP605 Evaluation Kit using Xilinx Platform Studio (XPS) provided by Embedded Development Kit (EDK) of Xilinx.

The architecture shown in figure 5 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:

- 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. This will run SPI Flash at 50MHz.
- General Purpose Input Output (GPIO) operating at 50MHz and used to communicate with LEDs, and Push Buttons. Push Buttons will trigger the reprogramming and download of Mutliboot bitsream files from extenal SPI flash memory, and the corruption of the Multiboot images data.
- The AXI_HWICAP core used to carry out dynamic and full reconfiguration of the Spartan-6 FPGA.
- UART (Universal Asynchronous Receiver Transmitter) interface provide serial communication with a PC via an UART/USB converter. It will enable debugging and monitoring of our implementation.

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

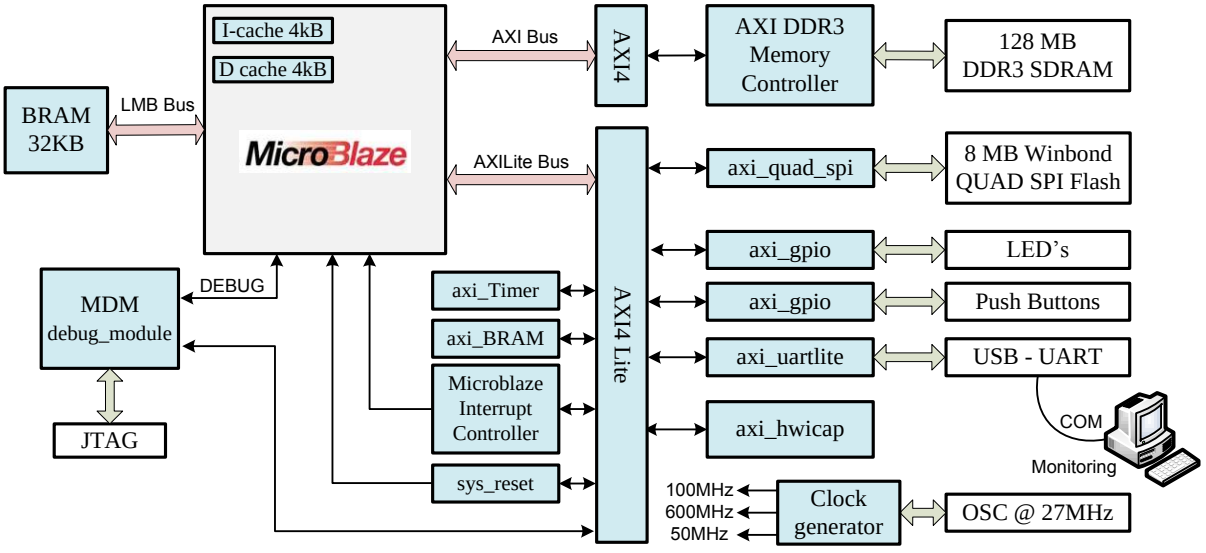

Fig. 5: Hardware Architecture of the reference system.

5.2 Software Design

5.2.1 Software Applications

The software applications consist of C projects developed in Software Development Kit (SDK). To implement and test Multiboot/fallback features, we used the low-level macros and functions of the AXI_HWICAP core. Thus, the corresponding software application to each hardware configuration (Golden, First and second Multiboot images) will include a low-level function named HwIcapLowLevel() and set with the base address of the update image. The function will enable FPGA reconfiguration by personalizing it with state machine for ICAP_SPARTAN6 reboot sequence based on the IPROG command as described in section 4. This low-level function will execute the following main tasks:

- Initiate the Abort sequence in the ICAP by setting the Abort bit in the Control Registers.
- Write the personalized command sequence to the Write FIFO Register.
- Start Transfer data from the FIFO to the ICAP device by writing "0x00000001" into the Control Register.
- Poll for Done bit in the Status Register, which indicates end of transfer.
- Wait till the Write bit is cleared in the CR register indicating the successful completion of the configuration.

Figure 6 shows part of the code of the HwIcapLowLevel() function corresponding to software application of the First Multiboot image.

5.2.2 Test of the fallback feature

To test the fallback feature, we used a function to corrupt the Multiboot image stored in the SPI flash memory by writing 0xFF data in the ten first addresses, to force an error when the image is loaded. Writing access to the SPI Flash memory is simplified by the use of Xilinx In-System Flash (Xilisf) library [9].

Thus, for all software applications, we used:

- The same Board Support Package (BSP) after having added and configured the ISF library to work with the used Winbond SPI Flash memory.
- A function named corrupt_multiboot_image() using a set of pre-defined functions of the Xilisf library to write to the Serial Flash memory device.

```
static u32 Sequence1[BITSTREAM_LENGTH] =
{
  0xFFFF,   /* Dummy Word */
  0xAA99,  /* Sync Word*/
  0x5566,   /* Sync Word*/
  …….
  0x30A1,
  0x000E,
  …….
};
u32 HwIcapLowLevel(u32 BaseAddress, u32 iprog_address)
{
  ……….
  /*Initiate the Abort sequence in the ICAP  */
  RegData = XHwIcap_ReadReg(BaseAddress, XHI_CR_OFFSET);
  XHwIcap_WriteReg(BaseAddress, XHI_CR_OFFSET, RegData |
                                XHI_CR_SW_ABORT_MASK);
  /*Write command sequence to the FIFO */
  ……….
    for (Index = 0; Index < BITSTREAM_LENGTH; Index++) {
   XHwIcap_WriteReg(BaseAddress, XHI_WF_OFFSET, Sequence1[Index]);
    }
  ……….
  /*Start the transfer of the data from the FIFO to the ICAP device */
  XHwIcap_WriteReg(BaseAddress, XHI_CR_OFFSET,
                                XHI_CR_WRITE_MASK);

  /* Poll for done, which indicates end of transfer */
  Retries = 0;
  while ((XHwIcap_ReadReg(BaseAddress, XHI_SR_OFFSET) &
      XHI_SR_DONE_MASK) != XHI_SR_DONE_MASK) {
      Retries++;
      if (Retries > XHI_MAX_RETRIES) {
            return XST_FAILURE;
      }
  }
  /* Wait till the Write bit is cleared in the CR register */
  while ((XHwIcap_ReadReg(BaseAddress, XHI_CR_OFFSET)) &
          XHI_CR_WRITE_MASK);
  ……….
}
```

Fig. 6: The HwIcapLowLevel() function.

5.2.3 Programming the SPI Flash memory

To program the SPI Flash memory, we have to generate an MCS file (Intel MCS-86 Hexadecimal Format) from:

- The "Header" file in hexadecimal (HEX) format. We used a hex editor (the shareware HexEdit) to populate it with the sequences of commands described in Table 3.
- The bistream files of the Golden image and the two Multiboot images.

```
cd <project_path>\implementation
bitgen -w -g next_config_register_write:Disable -g reset_on_err:Yes -g spi_buswidth:1 -bd <Project_path>\Debug\Embedded_app_1.elf system.ncd Embedded_app_1.bit
bitgen -w -g next_config_register_write:Disable -g reset_on_err:Yes -g spi_buswidth:1 -bd <Project_path>\Debug\Embedded_app_2.elf system.ncd Embedded_app_2.bit
bitgen -w -g next_config_register_write:Enable -g reset_on_err:Yes -g spi_buswidth:1 -bd <Project_path>\Debug\\Golden.elf system.ncd Golden.bit
promgen -w -p mcs -r header.hex -o header.mcs
promgen -w -p mcs -spi -s 16384 -u 010000 Golden.bit -u 200000 Embedded_app_1.bit -u 400000 Embedded_app_2.bit -o golden_multiboot.mcs
type header.mcs golden_multiboot.mcs > header_golden_multiboot.mcs
```

Fig. 7: Executable script to generate the final MCS file.

The MCS file contains ASCII strings that define the storage address and data file. It can be created by the iMPACT software [10] or the BitGen/PROMGen commands [11]. We chose to use the PROMGen and BitGen commands. Figure 8 summarize the software flow for creating a MCS file and Figure 7 show the build_script.bat file used to generate the final header_golden_multiboot.mcs file according with the SPI Flash memory map described in Figure 4.

Figure 9 resume the whole process allowing implementation of the Multiboot and Fallback features.

6 Implementation, Test and Results

A serial terminal program, such as Tera Term, will be set to debug and monitor the various steps of the implementation test. We used the GPIO Push Buttons as follows (see Figure 10 in the next page):

- SW7 and SW8 used to trigger the download of one of the two Multiboot images.
- SW5 used to corrupt the data of the Mulitboot images.

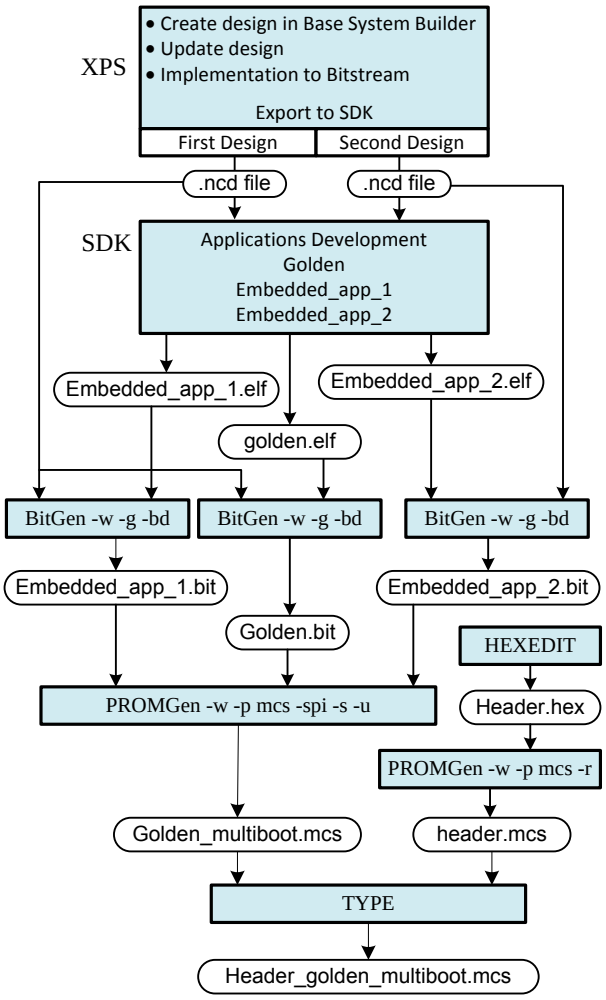

Fig. 8: Software flow for creating the MCS file

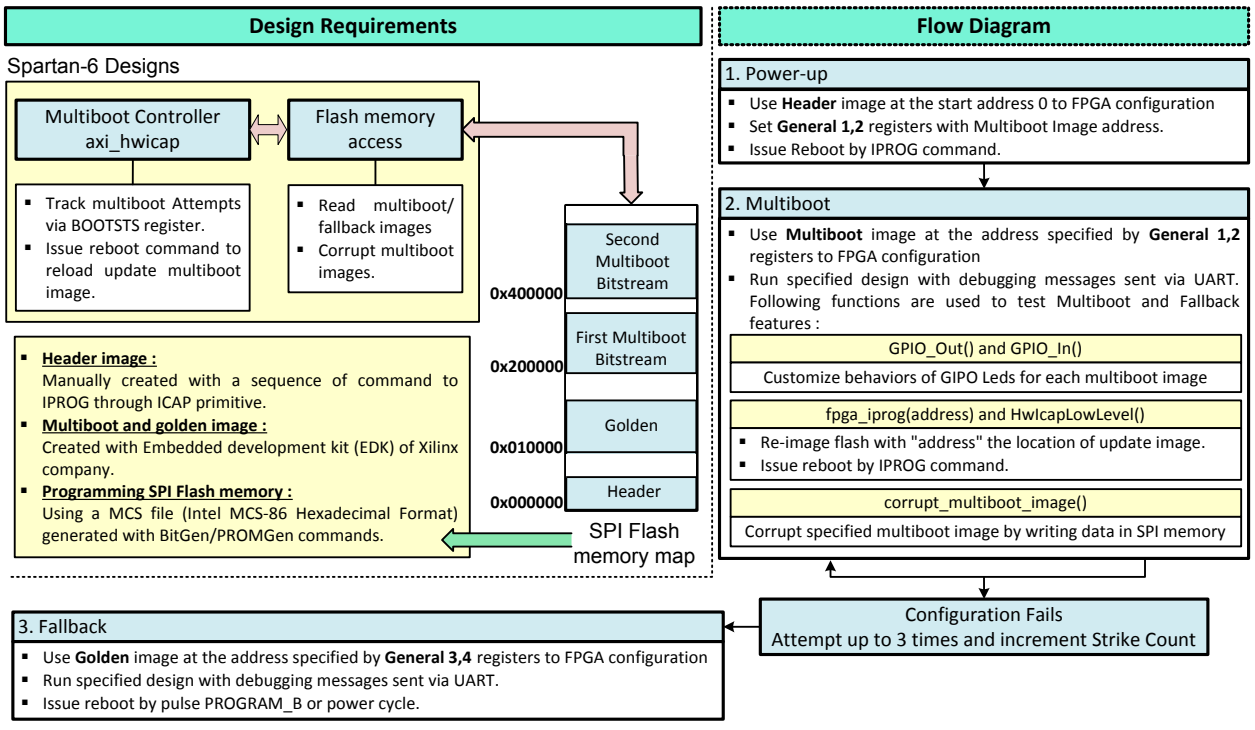

Fig. 9: Process logic model of the complete system.

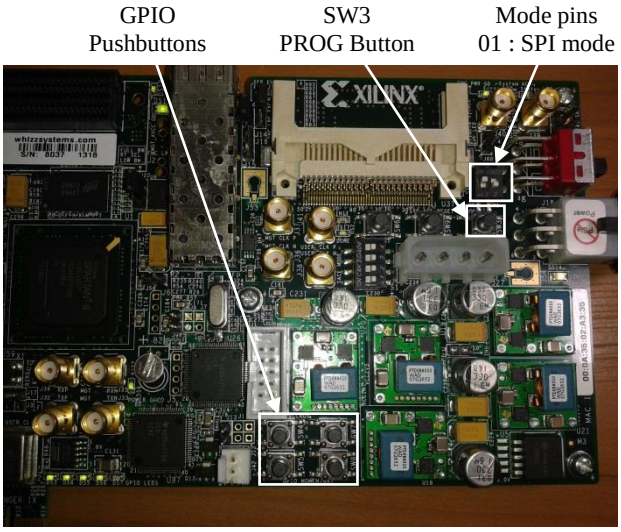

Fig. 10: The SP605 Platform used for implementation.

The execution of each test application includes customized behaviors of the GPIO Leds and messages sent via UART. Figure 11 shows an example of running results.

To test the run-time fallback feature, we used the iMPACT software to read the BOOTSTS configuration register that can store the potential errors of a MultiBoot:

- VALID_0 bit is updated with the current status.
- FALLBACK_1 bit is updated if fallback occurs.
- WTO_ERROR_1 bit is updated if watchdog time out error occurs.
- CRC_ERROR_1 bit is updated if CRC error occurs.

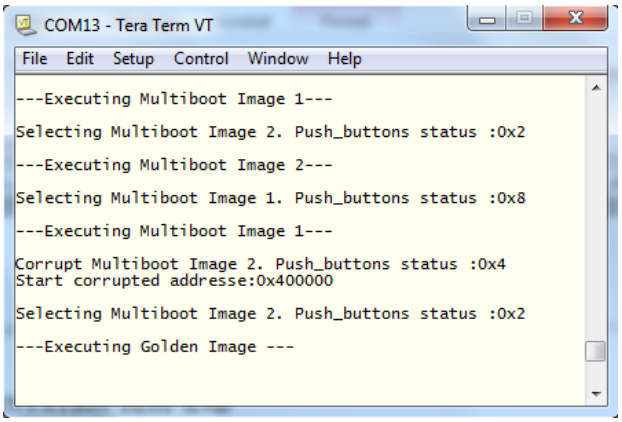

Fig. 11: Terminal Window showing debugging and monitoring

Figure 12 shows the contents of BOOTSTS register after a successful reconfiguration with a Multiboot image and figure 13 shows the contents of the same register when a fallback occurs.

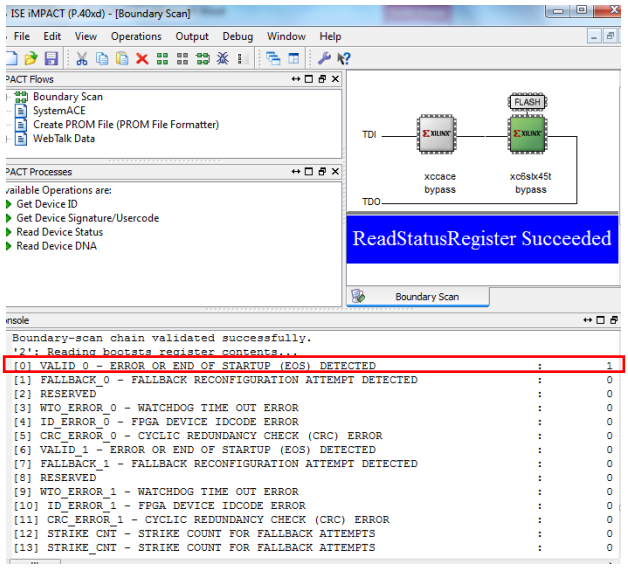

Fig. 12: The BOOTSTS status after successful reconfiguration.

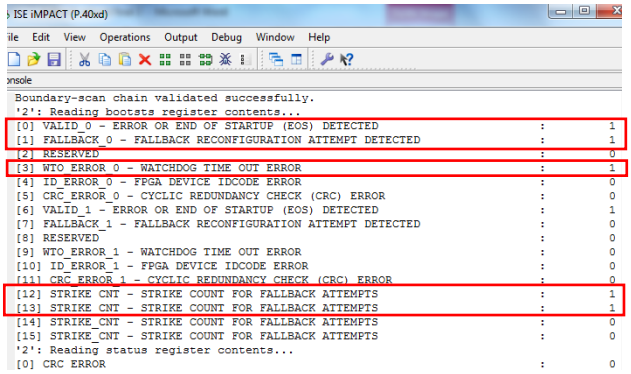

Fig. 13: The BOOTSTS status after a fallback occurs.

7 Conclusions and Future Works

In this paper, we have designed a run-time full reconfigurable embedded platform using a Fallback MultiBoot solution with a straightforward update bitstream overwrite method and a sequential try-and-recover configuration method. The FPGA tries to configure from the update bitstream, and if the FPGA detects a configuration error, it initiates a reconfiguration Fallback with a known good design (golden image). However, the configuration time for the Fallback case can be twice as long as the standard configuration time. The method also optimizes the use of an on-board SPI flash memory in any embedded system with limited memory resources.

Successful achievement of this work will allow us, as future work, design and implementation of an embedded platform with remote FPGA update capability that enables update systems with design patches or enhanced functionality. The new method of implementation will place the responsibility of error recovery on the programming operation via a simple adjustment to the programming algorithm for the bitstream update process. The aim is to propose a Fallback MultiBoot solution more robust, and quick to configure in all cases.

References:

[1] UG615 Xilinx Guide, *Spartan-6 Libraries Guide for HDL Designs*, 2012, pp: 121-122. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/spartan6_hdl.pdf.

[2] Koch, D., C. Beckhoff and J. Tørrison, *Advanced partial run-time reconfiguration on Spartan-6 FPGAs*. International Conference on Field-Programmable Technology, Dec. 8-10, 2010, IEEE Xplore Press, Beijing, pp: 361-364.

[3] Liu, M., W. Kuehn, Z. Lu, and A. Jantsch, *Run-Time Partial Reconfiguration Speed Investigation And Architectural Design Space Exploration.* Inter-national Conference on Field Programmable Logic and Applications, Aug. 31-Sept. 2, 2009, IEEE Xplore Press, Prague, pp: 498-502.

[4] Otero, A., M. Llinás, M. L. Lombardo, Jorge Portilla, E. de la Torre and T. Riesgo. *Cost and Energy Efficient Reconfigurable Embedded Platform Using Spartan-6 FPGAs*, Proceedings of the SPIE, Volume 8067, VLSI Circuits and Systems V, May 03, 2011.

[5] Khalil, M. R. and S. A. Mohammed. *Using Multi-boot Technique to Create a Multiple Embedded Designed Systems*, Journal of Theoretical and Applied Information Technology, Vol.34, No.1, 2011, pp. 80- 87.

[6] UG380 Xilinx Guide, *Spartan-6 FPGA Configuration User Guide,* January 23, 2013, pp: 24-27, 92-105 http://www.xilinx.com/support/documentation/user_guides/ug380.pdf.

[7] DS817 Xilinx IP Documentation. *LogiCORE IP AXI HWICAP*, 2012. http://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v2_03_a/ds817_axi_hwicap.pdf.

[8] UG761 Xilinx Guide, *AXI Reference Guide*, November 15, 2012. http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf.

[9] UG643 Xilinx Guide, *OS and Libraries Document Collection*, 2012, pp: 173-186. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/oslib_rm.pdf.

[10] Hanafi, A., and M. Karim. *Optimizing the use of an SPI Flash PROM in Microblaze-Based Embedded Systems*, International Journal of Advanced Computer Science and Applications, Vol.4 No.10, 2013, pp. 109-114.

[11] UG628 Xilinx Guide, *Command Line Tools User Guide*, 2012, pp: 173-186. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_4/devref.pdf.