Methods for Booting an All Programmable System-on-Chip over PCI Express Link

Mrinal J Sarmah, Bokka Abhiram Saikrishna, Anil Kumar A. V., Kamalesh Vikramasimhan
Processors, Software and System Applications Group
Xilinx Inc.
Hyderabad, India

Syed Azeemuddin
Center for VLSI and Embedded Systems
IIIT, Hyderabad
Hyderabad, India

***Abstract*—An All Programmable System-on-Chip (SoC) solution consisting of a hardened processor and a programmable fabric solution has been identified as a growing heterogeneous programmable platform catering to variety of application areas including communication, remote radio head, and software-defined radio and so on. The process to boot such a SoC with embedded processor and programmable fabric has been a challenge considering the complexity involved in multiple stages of booting. The boot sequence typically involves loading of first stage loader, configuring the programmable fabric with configuration bitstream followed by loading of user boot loader. Booting with typical boot devices like Quad Serial Peripheral Interface (QSPI), NAND or NOR flash results in increased boot time. The embedded PCI Express block in such SoC can be used as a carrier for the boot image including the user boot loader and the bitstream. The boot loader can be transferred from a remote host PC to the SoC's on chip memory over PCIe link. Such boot process imparts significant improvement in boot time and ensures boot image security since the boot images can be stored in a secure processing system. Using boot over PCIe methodology, the authors have experimentally found 20x improvement in boot time over the state-of-the-art boot mechanisms.**

Keywords—All Programmable SoC, PCIe, boot, secondary boot, u-boot, bitstream, FPGA, Programmable Logic, Zynq

I. Introduction

With increasing complexity of the processors and peripherals used in System-on-Chip (SoC) solutions, the firmware associated with system boot is becoming inherently complex. The system level requirement is demanding significant portion of the peripherals to be operational after the first stage boot loader. In a multi-core system, the first stage boot loader image is responsible for initializing minimal peripherals as defined in the system level specification. The necessity to integrate the peripheral configuration sequence in the first stage boot loader is making the size of the boot loader considerably large. Since the boot loader files need to be stored in a non-volatile memory, with increase in size of the boot image, the cost of the memory device is adding up to the system cost.

The system security is gaining importance with the advancement in the embedded systems and their deployment in applications requiring secure access.

Peripheral Component Interconnect Express (PCIe) is a bus topology which is used to connect a peripheral to the PC system [1]. PCIe is integrated in most of the industry standard FPGA devices and the interface is used for FPGA based hardware acceleration. PCIe uses high speed serial IO as a means to communicate between the PCIe device and the host system. PCIe interface enables data transfer at 2.5 Gb/s, 5 Gb/s and 8 Gb/s lane rate depending on the PCIe revision used, thereby enabling massive throughput on the interface which makes it suitable for hardware acceleration usage. PCIe protocol defines clubbing of multiple PCIe lanes to aggregate more data throughput. A 4 lane configuration of 2nd generation PCIe interface can yield serial bandwidth of 20 Gb/s. The data transferred over PCIe link is scrambled and 8B/10B encoded which makes it difficult for an external intruder to easily decode the serial stream. External intrusion is possible only if a PCIe link partner is inserted in the serial data path which can easily be disabled while designing the Printed Circuit Board (PCB). In contrary, it is very easy to probe the I/O pins of a typical boot device and decode the information contained in the boot loader. The inherent protocol level security arising from the scrambling technique used and the massive throughput resulting from the high speed serial interface makes the interface suitable for transferring boot images to a SoC block.

With the advent of All Programmable SoC (AP SoC) device it is possible to integrate an embedded processor design with a programmable logic (PL) to provide a single chip solution to a programmable Integrated Circuit [1, 2]. An AP SoC device requires a configuration bitstream to program the PL part. The size of the bitstream depends on the size of the AP SoC device and typical bitstream size in Xilinx Zynq-7000 AP SoC ranges from 4 MB to 16 MB. The boot flow consists of loading of the FSBL followed by loading of the bitstream to configure the PL

978-1-5090-6621-6/17/$31.00 ©2017 IEEE

with user design. Storing the FSBL image and the configuration bitstream requires large external storage which adds to the overall system cost. Transferring the images over the external memory interface to the AP SoC is time consuming due to larger transfer size and limited frequency of operation of the memory interface. In order to reduce the image transfer time, a high speed interface is desired which is the primary motivation of the work described in this paper.

In this paper, the idea of transferring the boot image using PCIe interface which eliminates the necessity of storing a large image in a non-volatile memory is explored. The second stage boot loader image has been transferred over PCIe interface and the transfer time has been compared with standard boot devices.

In Section II, work related to processor boot is discussed. Section III elaborates the problem in more details. Section IV explains the proposed solution. In Section V, experimental results are discussed. We conclude the paper on Section VI.

II. Related Work

The existing literature discusses about fast boot methodologies and algorithms to reduce boot time in a multi-processing embedded environment. In [1, 2] the authors have discussed about a method to upgrade boot image remotely using a PCIe based serial link. The method explores using PCIe based link to fetch the programmable code from a Serial Peripheral Interface (SPI) based flash device. A hardware based state machine is designed to fetch the boot code from non-volatile memory and send it over PCIe link. However, the method does not involve a two CPU model to fetch the code from a PCIe based host. In [3] a fast boot methodology has been discussed using an external NAND device. The authors have observed 25% advantage in boot time with the methodology while the overhead associated is the use of a high density NAND device. The state-of-the art secure boot involves encryption and authentication of the boot image before storing the image in the external flash device. In [4] the authors have devised a mechanism to securely boot an embedded processing system deploying a secure processor and a co-processor. The overhead involved with the approach is 25% increase in boot time and the addition of processor logic to handle security aspects. In [5] a trusted computing mechanism has been discussed which enables security to the bitstream and the kernel images. The drawback of the approach is area overhead. In [6] an approach to use on board Non Volatile Memory (NVM) like Platform flash PROMs to store the secondary boot configuration and a soft core processor to load the boot configuration is discussed. The setback with this approach is, with increase in secondary boot image the size of the non-volatile memory will also increase and so is not cost effective.

III. Problem Definition

A conventional non-volatile memory device is capable of storing images ranging up to gigabyte range. With the help of file system the required boot images can be stored in the flash device and retrieved during boot process. One common problem associated with the flash devices is its limited storage capability. The device's memory retention ability may decay depending on the operating condition of the system and with ageing.

With the growing size of the boot image the time taken to load the boot image from primary boot devices increases substantially. Equation (1) defines the relation between boot image size, operating frequency of the boot device and time taken to boot [1].

$$T = k * S/(W * F) \quad (1)$$

Where T refers to time to boot, k is a proportionality constant, S is boot image size, W is the width of the memory interface used and F is the frequency of boot device. The clock frequency of the commonly used flash devices is limited due to the physical interface used for communication between the boot device and the controller. With continuously growing image size, the boot time will increase unless F is not proportionately increased.

The following sections describe different overheads associated with an external flash device used for booting a multi-processor system.

A. Area Overhead

The memory size to store the boot image and the boot image size have a direct relation. Hence, as the boot image size increases, the memory size increases causing area overhead on the overall system design.

B. Hardware and Power Overhead

The approach to reduce the boot time must utilize optimal hardware. Certain techniques to improve the boot time like using stacked QSPI device as boot source, using uncompressed boot image etc. have hardware overhead and hence are not cost effective. On increasing the hardware resources, the power consumption would also increase significantly. Increasing the clock frequency of the boot device to reduce the boot size would decrease the boot time considerably but leads to high dynamic power consumption.

C. Security of the Boot Images

The boot loader image is susceptible to security threats and it is required to be encrypted before storing the images in the external flash device. The decryption and authentication process is time consuming which adds to the overall boot time for security enabled systems. Also, the serial lines of the conventional boot sources like QSPI could be probed easily and pose security threat.

It is a common practice to upgrade the system firmware in SoC based systems. Upgrading the boot loader requires re-programming of the non-volatile memory device using standard programming tool. Such field up-gradation of systems involves cost overhead.

The technique to reduce the boot time must offer a cost effective solution to load the boot image utilizing optimal hardware resources and not compromising the security of the boot image.

IV. Proposed Solution

PCIe can be used as a candidate for booting a multi-processing SoC and can eliminate some of the limitations mentioned in section III. Using PCIe interface for boot offers the following advantages:

1. PCIe interface offers very high serial bandwidth. An x4Gen2 configuration can offer 20 Gb/s serial throughput. After eliminating 8B/10B encoding overhead used in PCIe, it is possible to achieve 16 Gb/s of effective throughput over the bus [1]. Thus equation (1) can be modified to:

$$T = k * S/16 * 10^{-9} \quad (2)$$

   If clock frequency in equation (1) is expressed in megahertz, equation (2) results into 20X improvement in T for a 16 MHz, 4 bit wide conventional memory device.

2. PCIe defines the serial characters to be scrambled and 8B/10B encoded which imposes an inherent security to the boot image transferred over PCIe link. Unless the intruder implements a PCIe based link partner it may be impossible to decode the data transferred over the PCIe link. The system design can prevent insertion of a PCIe link partner in the data path.

3. Using boot over PCIe enables easy update to the boot image. The upgradable image can easily be transferred over to the PCIe host using Ethernet interfaces. Once the new image is available, the PCIe host system software can deploy the image during the SoC boot. The system software can perform authenticity of the upgradable image before deploying them on the SoC system.

4. PCIe interface offers Quality of Service (QoS) option which ensures guaranteed transfer time [1]. For boot time critical system applications, it is possible to use QoS option to ensure guaranteed latency.

In the conventional boot flow, after power-up-sequence is executed, the SoC hardware hands off the control to the Boot Read Only Memory (BootROM) block. The BootROM samples the boot mode pin to know the boot device used in the system. The bootROM fetches the first stage boot loader (FSBL) image header from the external flash memory and based on the location and transfer size of the boot image copes the image on an on-chip-memory present in the SoC. The BootROM passes the CPU control to FSBL and FSBL performs initialization of the minimal peripherals. FSBL hands off control to the second stage boot image or the application as depicted in Figure 1.

PCIe is present as a hard block in Zynq-7000 AP SoC. Since it is part of the PL part of the chip, bringing up the PCIe link requires the PL to be programmed before using boot over PCIe option. In SoC devices, where PCIe is part of the Processing System (PS) and hence does not require PL to be programmed, it is possible to transfer the FSBL and PL bitstream over PCIe interface and the solution can be used for primary boot option over PCIe. The work presented here deals with secondary boot option over PCIe and requires PL to be programmed by the FSBL image.

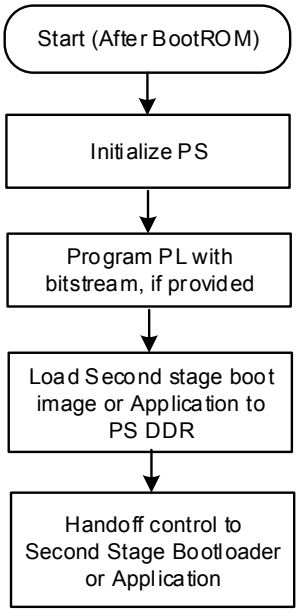

Figure 1: Original FSBL Boot Flow

This method is different from the actual FSBL flow in the way second stage boot image is transferred. The second stage boot image is read into an internal buffer by an application running on the host machine and the data is sent to the kernel space PCIe driver in chunks of one double word (DW). After the entire file is transferred into the PS-DDR, FSBL hands off control to the second stage boot image as shown in Figure 2.

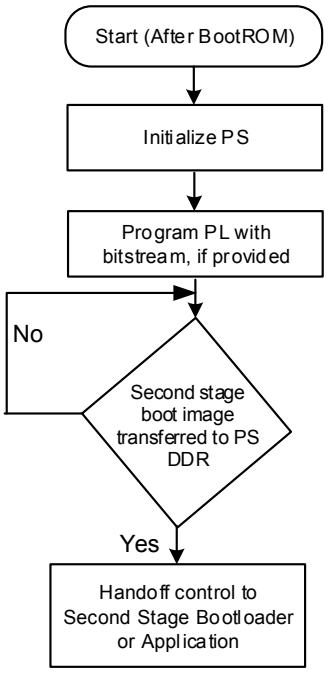

Figure 2: Proposed FSBL Boot Flow

v. Experimental Results

The focus of this paper is to show improvement in boot time by means of transferring the application over PCIe links of the SoC. As a proof of concept, the proposed solution is implemented on Xilinx All Programmable SoC platform called Zynq®-7000.

ZC706 is a Zynq®-7000 All Programmable SoC evaluation board featuring the XC7Z045 FFG900 -2 SoC. It is optimized for quickly prototyping embedded applications using Zynq-7000 SoCs. It has an advanced memory interface with 1GB DDR3 SODIMM Memory. The evaluation board has x4Gen2 PCIe edge which can be used to connect to a PCIe based host system. In the experimental setup, PCIe has been configured as an end point block.

Figure 3 shows the functional block diagram of the methodology discussed here and data flow from host computer image is transferred to the PS DDR and then the FSBL hands off control to the application.

The Processing System 7 IP is the software interface around the Zynq Processing System. PS7 block is configured such that the Programmable Logic (PL) fabric clock frequency is 250 MHz. The PS7 hard block in the SoC has high performance ports by which it can communicate with the PL. In this design, one AXI HP port and one GP Port are used.

B. Software Implementation

The secondary boot image which resides in the host computer is parsed into double words (DW) by an application running on the host and transmitted over PCIe. One DW is 4 bytes. These DWs are written at a predefined location in the PS DDR by the AXI Master. After the secondary image is transferred, the handoff address is passed to the FSBL, which then picks up the boot image from the address passed in the handoff routine and then executes it.

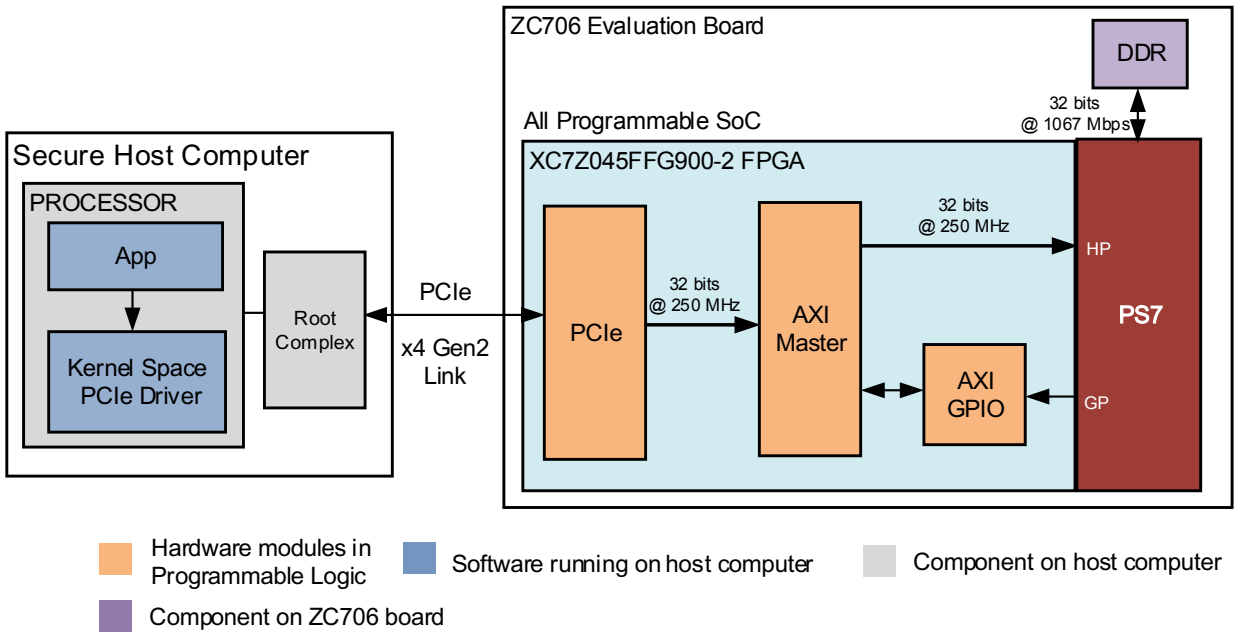

Figure 3: System level block diagram of the proposed solution

to the DDR memory of the System-on-Chip.

A. Hardware Implementation

The PCIe IP block is configured in 4 lane Gen2 mode. The AXI Master Module implements an AXI Memory Mapped interface and transfers the data coming from the streaming interface of PCIe to PS-DDR. It also monitors for the number of bytes transferred. Once the entire image is transferred, it signals the completion of the image transfer to the PS through the AXI GPIO module. There is a handshake mechanism implemented in the AXI Master module to indicate the completion of image transfer. In the proposed solution, the original FSBL code is modified to wait until the secondary

The default FSBL code can be modified using the Xilinx SDK. The SDK comes with the standard FSBL implementation for Zynq SoC. In the current implementation, the standard FSBL is modified to suit the proposed requirement. After all the initialization is done by the FSBL, the FSBL is put in polling mode, waiting for the image to be transferred into PS DDR. Once the FSBL receives the transfer complete indication, it assigns initial DDR address where the image is written to the next handoff address.

C. System Level Measurement

Using the proposed solution, the boot time for various image sizes are measured on the Asus Maximus 5 Gene machine as host PC with Intel i7 processor.

Table 1 gives the comparison of boot time (in ms) between PCIe and Dual QSPI with interface frequency of 200 MHz,

NAND at 100 MHz and Secure Digital (SD) operating at 50 MHz speed. PCIe interface is configured for x4Gen2.

TABLE I PCIe VS OTHER BOOT DEVICES BOOT TIME

Image Size	PCIe	Dual QSPI	NAND	SD
182 KB	0.278 ms	5.308 ms	8.421 ms	19.77 ms
1 MB	1.61 ms	29.934 ms	47.13 ms	121.8 ms
13 MB	20.413 ms	388.15 ms	615.2 ms	1541.1ms

Figure 4 shows the the graphical representation of the boot time for QSPI and PCIe boot mode option. Here horizontal axis is image size and vertical axis is time (in ms).

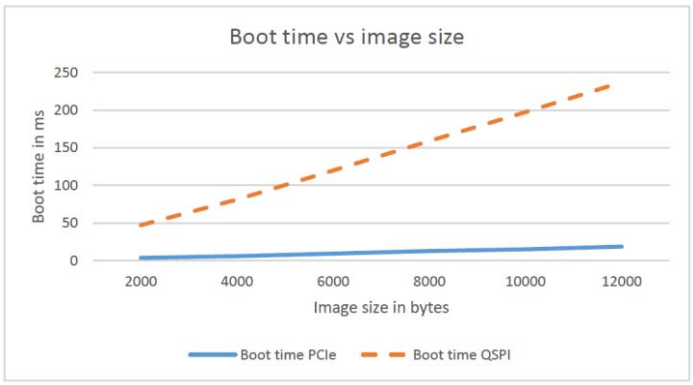

Figure 4: PCIe vs QSPI Boot Time

As the above graph depicts the boot time taken in case of PCIe as boot source is significantly less as compared to that of QSPI boot mode.

VI. Conclusion and Future Work

There is significant improvement in the boot time between standard boot mechanism and the boot mechanism proposed in this paper. The improvement of booting over PCIe is 20 times faster when compared to standard boot mechanism. The proposed solution also provides security feature to the application to be transferred, and thereby reduces time and complexity involved in employing the encryption techniques also. In systems using PCIe block for hardware acceleration, the same interface can be used to boot the processors in the AP SoC block which is used for acceleration.

In future the methodology will be extended to boot the next generation SoC device in the primary boot mode.

References

[1] Anna Fernandes, Rita C Pereira, Jorge Sousa, Paulo F Carvalho, Miguel Correia, Antonio P Rodrigues, Bernardo B Carvalho, Carlos M B A Correia, Bruno Goncalves, "FPGA Remote Update for Nuclear Encirnments", IEEE Trans. on Nuclear Science, Year 2016, Volume:63, Issue:3.

[2] Anna Fernandex, Rita C. Pereira, Jorge Sousa, Paulo F Carvalho, Miguel Correia, Antonio P Rodrigues, Bernardo B Carvalho, Carlos M B A Correia, Bruno Goncalves, "FPGA remote update for nuclear environments", 2015 4th International Conference on Advancements in Nuclear Instrumentation Measurement Methods and their Applications, 2015, pp. 1-4.

[3] Dey S., "Fast Boot User Experience Using Adaptive Storage Partitioning", Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. Computation World,2009, pp. 113-118

[4] Obaid Khalid, Carsten Rolfes, Andreas Ibing, "On implementing trusted boot for embedded systems", 2013 International Symposium on Harvard-Oriented Security and Trust, 2013, pp. 75-80.

[5] Devic,F.,Torres, L. ; Badrignans, B., "Securing Boot of an Embedded Linux on FPGA", IEEE International Symposium on Parallel and Distributed Processing, 2011, pp. 189-195.

[6] Shalin Sheth, "Microblaze Platform Flash/PROM boot loader and User Data Storage.", Xilinx Application Note, XAPP-482. http://www.xilinx.com/support/documentation/application_notes/xapp482.pdf