Information Technology for Configuration of System-on-Chip in Cloud Environment

Yaroslav Krainyk[1][0000-0002-7924-3878], Oleksii Denysov[2] and Yevhen Darnapuk[1][0000-0002-7099-5344]

[1] Petro Mohyla Black Sea National University, Mykolaiv, Ukraine
[2] EnvisionTEC, Kyiv, Ukraine
yaroslav.krainyk@chmnu.edu.ua, maildenisov@gmail.com,
yevhen.darnapuk@chmnu.edu.ua

Abstract. The presented paper conducts the research on the topic of System-on-Chip (SoC) configuration in cloud environment. The main focus is devoted to the configuration of the SoC that includes Field-Programmable Gate Array as a composing part of the chip. The proposed approach is based on the assumption that each part of the system should support remote configuration by matching software artifact. We consider each component for SoC individually and provide a complex solution for its configuration when all of them are running as a system. Moreover, the settings of running machine can be changed dynamically according to the devised information technology. Peculiar attention is paid to the configuration of programmable logic within the framework. As cloud providers also establish services that involve configuration of FPGA and its further usage of an accelerator for specific tasks, the information technology can be used as a basis for software manager responsible for configuration of the SoC.

Keywords: Configuration, System-on-Chip, Field-Programmable Gate Array.

1 Introduction

Modern cloud infrastructure provides plethora of services [1-4] that can be applied in general situations as well as for solutions of specific problems. In fact, it has developed to such a degree where complete ready-to-use solutions can be found in marketplace and immediately deployed on the level of infrastructure, platform or software. Cloud environment exposes almost unlimited computation and storage resources and offers them to customers on the terms of pay-as-you-use license agreement.

While the major part of computations in the cloud infrastructure are executed by the means of general purpose Central Processing Units (CPU) or Graphical Processing Units (GPU), System-on-Chip (SoC) architecture is gaining momentum in the market segment of end-user devices Despite the fact that combination of CPU and GPU is predominant for SoC design, Field-Programmable Gate Array (FPGA) part in junction with CPU is one of the possible alternatives. Hence, the main point of interest of this paper is the development of the information technology that allows configuring SoC composed of CPU and FPGA parts in the cloud environment.

The rest parts of the paper are organized as follows. The second section provides a review on the state-of-the-art developments in the cloud technologies that involve SoC and FPGA. The third section contributes the devised information technology for SoC configuration in the cloud infrastructure. The fourth section demonstrates practical usage of the technology on the testbed hardware.

2 Related work review and background

The main advantage of the SoC [5] is that the developer can exploit one of the available computational units that is more suitable for the specific task. To advance this paradigm even further, FPGA part of the SoC is configurable and may be programmed with different configurations in runtime mode. For instance, programmable logic is suitable for data processing in stream mode [6], image and sensor processing [7, 8], etc. At the same time, due to the more efficient architecture, FPGA also outperforms other computational modules in energy efficiency.

The paper [9] establishes an extensive review of FPGA usage in the cloud environment. It considered solution for different scenarios involved FPGA from the low level of development (e.g., Intellectual Property cores design and selection) to the level of the system integration. However, it concentrated on the FPGA part only while SoC architecture was not discussed in the work. In opposite, this paper is oriented to provide the solution for SoC with FPGA part specifically.

The authors of papers [10, 11] investigated deployment of FPGA and its availability as a service. The devised architecture allows the user to access services run on FPGA. The solution is based on usage of PCIe interface to communicate with programmable logic part. However, it does not support configuration of a processor part of the SoC and, therefore, requires improvement.

One of the biggest cloud providers, Amazon, offers F1 instances that employ FPGA as a computational element [12]. Such instances are configured via Amazon console Graphical User Interface (GUI). Each FPGA instance is associated with an FPGA image. The image is an output from the development environment established by Amazon. Henceforth, this whole infrastructure can be considered as a complete solution for deployment of FPGA-based solutions in the cloud.

In case when SoC is used as a computational unit in the cloud, the developer have to deal with software part alongside with hardware part. In contrary to the situation with single FPGA accelerator that can be simply programmed, SoC also hosts user-space software for communication with accelerator.

Linux operating system (OS) [13] is a prevalent choice for cloud providers. Thus, information technology under development should be tailored to the presence of Linux as a software environment [14]. While for processor part Linux is an essential OS and multiple solutions are ready-to-use, presence of the FPGA part imposes the problem of communication between processor and FPGA on the level or running Linux OS.

The main contribution of the presented paper is in the developed information technology that allows performing configuration of SoC with programmable logic part at the boot stage and on the running instance. As a matter of fact, it supports configuration of the most elemental components and, as a result, the whole system. The particular role in the investigation is prescribed for FPGA that can be configured at the initial stage as well as during the runtime.

3 Information technology for SoC configuration

Let us assume that the system under investigation comprises a SoC with processor and FPGA parts. Contemporary SoCs support not only configuration during the boot process but also dynamic configuration with all components launched. Therefore, there are two options to consider from the point of view of FPGA. As FPGA is primarily employed as an accelerator for operations that support high degree of parallelism, option of dynamic configuration provides a clear advantage.

The work of hardware components depends on the software artifacts and software organization. In this work, we investigate the system that runs under control of the Linux operating system with minimum set of software components. The set includes:

1. First-stage bootloader (FSBL).
2. Second-stage bootloader (SSBL).
3. Boot script.
4. Device-tree file.
5. Kernel image.
6. File system to work with when the system is running.

In addition, the list can be completed with FPGA bitstream configuration file for programmable logic part programming. Regarding the possibility of remote configuration that is obligatory in the cloud environment, most of the mentioned components should support substitution with the newer version of its counterparts.

Typically, the enlisted components are stored in the Flash-memory device (e.g. SD-card) that is mounted on the SoC board. When the boot process is initiated, necessary parts are invoked in a sequential manner to further run FPGA accelerator. In cloud infrastructure, devices should be configured remotely without direct physical access to them.

3.1 System configuration at the boot stage

First-stage bootloader executes preliminary initialization of the low-level devices and invokes second-stage bootloader. Being important part of the whole process, first-stage bootloader cannot access more complex elements due to bootloader’s constraints. As a matter of fact, this bootloader is to be written into the specific memory region and cannot be configured according to the proposed workflow. On the other side, the SSBL is a key component from the point of view of configuration during the boot stage.

To automate booting process, SSBL relies on the boot script that displays typical scenario to perform initialization of the whole system. It is important to mention that when SSBL receives control over the system, network interfaces are initialized and can communicate with internal network elements of the cloud via basic protocols. This also implies that data from the cloud network is accessible and can be written into the memory of the device.

This way the content and functions of the SoC can be changed completely at the next reboot. On the basic level, to conduct this process SSBL needs to retrieve system files and to write them into dedicated memory regions.

As soon as SoC finishes launch sequence for all components, it is required to invoke the application software part. While it is still possible that FPGA part works independently, the most probable scenario in the cloud is that processor part receives data from the internal network and sends it to the FPGA accelerator. Thus, both parts are connected with each other in terms of data interchange.

To make dependency between hardware platform and software even looser, the partition with file system for Linux libraries and executables may be established as a Network File System (NFS).

The complete sequence of actions required by the devised information technology is shown in Fig. 1 as a sequence diagram.

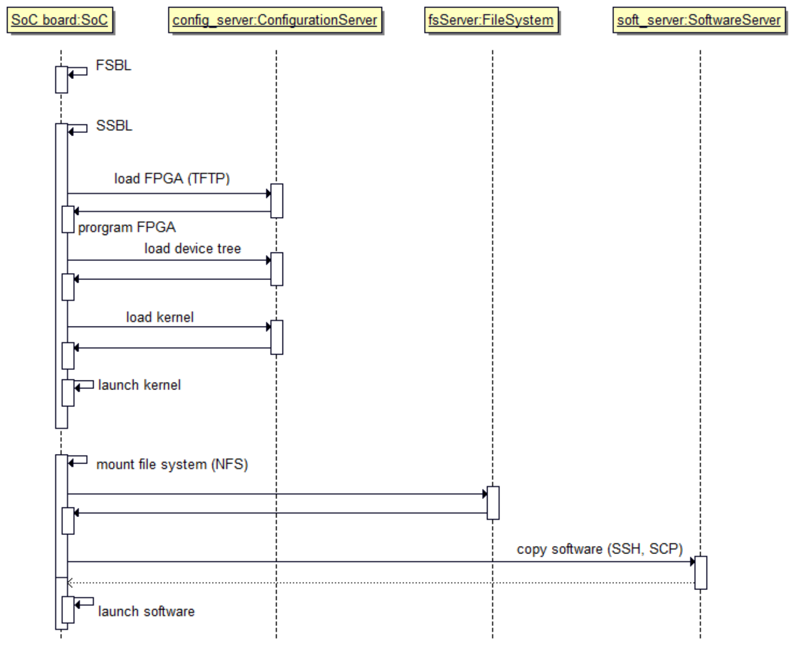


Fig. 1. Sequence diagram of the communication between participating hardware components.

The major disadvantage of the solution that relies only on boot process is that change of configuration demands reboot operation. It increases latency of the system when new functions are available for the use. Therefore, dynamic configuration of the system without rebooting.

3.2 Dynamic configuration of the system

As dynamic configuration supposes that current running software components are stopped and further substituted with new ones, this mechanism should be formalized for all the involved components.

In the kernel version 3.xx of Linux OS, configuration port of FPGA is accessed via specific device (/dev/fpga). However, newer version of the OS assumes different mechanisms for the FPGA programming. The list of possible options includes:

1. Configuration via a dedicated programming interface using software that just writes data from bitstream file into specified address.
2. Configuration using overlays for device tree and configfs file system.

In general, the first option is similar to the previous approach and demands a user-space application with appropriate privileges that executes necessary actions. In contrary, the second option relies on the functions available in the Linux kernel, however, it is not as obvious as the mentioned one.

From the point of view of the Linux kernel, FPGA is assigned to the specific address space. It is controlled by fpga-manager instance accessible from the sysfs file system. To apply necessary changes to the system and load new configuration into programmable logic part, overlay mechanism may be applied.

To activate the overlay, it should be compiled into binary form and then inserted into existing structure using configfs. Overlay is a part of device tree, so, initially, it has a form of device tree include file. The sample file that describes overlay with FPGA firmware is shown in the listing below.

```
/fpga_config/;
/ {
fragment@0 {
target-path = “/soc/base-fpga-region”;
#address-cells = <1>;
#size-cells = <1>;
overlay {
firmware-name = “reconfig.rbf”;
#address-cells = <1>;
#size-cells = <1>;};};};
```

The parameter “target-path” indicates the part of the original device tree for substitution. Configuration of the file for FPGA programming is assigned using “firmware-name” parameter. The file should be visible from the current work directory.

For single FPGA configuration, it can be achieved with the following sequence of commands:

```
mount -t configfs configfs /root/config
rmdir /config/device-tree/overlays/fpga
mkdir /config/device-tree/overlays/fpga
echo fpga_config.dtbo > /config/device-tree/overlays/fpga/path
```

As a result of this action, FPGA section is programmed with a new rbf-file. To extend the proposed concept even further, on the level of OS, the developer can allocate specific directory for rbf-files (they can be previously retrieved from the server). Each configuration file is matched by specific overlay file that is enabled according to the requests from the customer. The general presentation of the proposed solution is shown in Fig. 2.

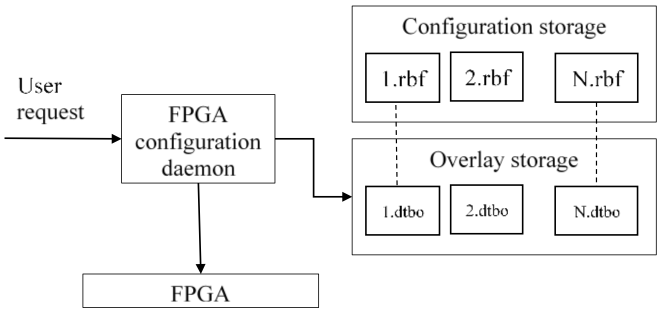


Fig. 2. The general diagram of overlay-based FPGA-configuration.

Therefore, the customer can select necessary functionality from the prepared configurations and change it at the runtime stage without the need for reboot. This way the accelerator is enabled almost immediately after programming.

4 Results and Discussions

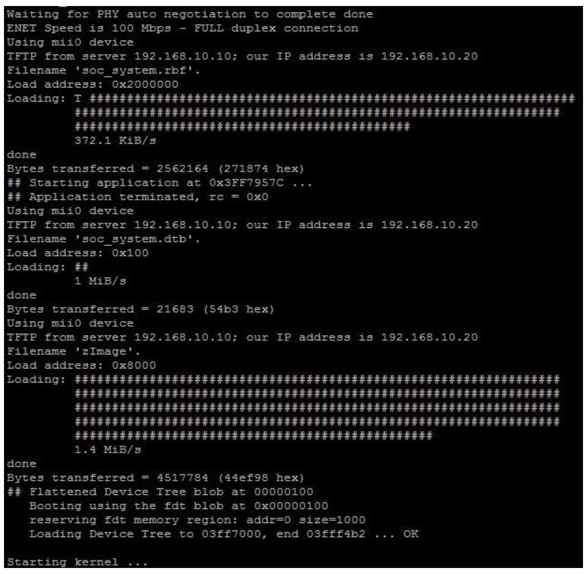
To demonstrate the correctness of the workflow according to the information technology, we employed DE0-Nano SoC Kit developed by Terasic. The board facilitates Cyclone V SoC with FPGA part and two Cortex A9 cores. The testbed was integrated into a local network and assigned a static IP-address. The board was configured connect to the main server machine with running TFTP-server and open SSH-port.

The u-boot script was prepared with following parameters and commands (only significant settings in context of the boot process are shown):

```
mmcroot=/dev/mmcblk0p2
fdtimage=socfpga.dtb
bridge_enable_handoff=mw $fpgaintf ${fpgaintf_handoff}; go $fpga2sdram_apply; mw
$fpga2sdram ${fpga2sdram_handoff}; mw $axibridge ${axibridge_handoff}; mw
$l3remap ${l3remap_handoff}
xfpga=tftpboot 0x2000000 soc_system.rbf;fpga load 0 0x2000000 0x700000; run
bridge_enable_handoff;tftpboot 0x00000100 soc_system.dtb
xload=run xfpga;tftpboot 0x8000 zImage;setenv bootargs console=ttyS0,115200
root=${mmcroot} rw rootwait;bootz 0x8000 - 0x00000100
```

Actually, the script just executes single command (run xload) to configure the system. By the end of the command execution, we can observe the situation when the FPGA is programmed with bitstream file while device tree is loaded and operating system is launched. The final step is to run corresponding software. As soon as the kernel is launched, copy command is executed from the startup script and, then, copied software starts.

In Fig. 3, the output received using terminal program putty is shown.



```
Waiting for PHY auto negotiation to complete done
ENET Speed is 100 Mbps - FULL duplex connection
Using mii0 device
TFTP from server 192.168.10.10; our IP address is 192.168.10.20
Filename 'soc_system.rbf'.
Load address: 0x2000000
Loading: T #################################################################
         #################################################################
         #############################################
         372.1 KiB/s
done
Bytes transferred = 2562164 (271874 hex)
## Starting application at 0x3FF7957C ...
## Application terminated, rc = 0x0
Using mii0 device
TFTP from server 192.168.10.10; our IP address is 192.168.10.20
Filename 'soc_system.dtb'.
Load address: 0x100
Loading: ##
         1 MiB/s
done
Bytes transferred = 21683 (54b3 hex)
Using mii0 device
TFTP from server 192.168.10.10; our IP address is 192.168.10.20
Filename 'zImage'.
Load address: 0x8000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         ################################################
         1.4 MiB/s
done
Bytes transferred = 4517784 (44ef98 hex)
## Flattened Device Tree blob at 00000100
   Booting using the fdt blob at 0x00000100
   reserving fdt memory region: addr=0 size=1000
   Loading Device Tree to 03ff7000, end 03fff4b2 ... OK

Starting kernel ...
```

Fig. 3. An example of terminal text output during the components load stage.

Previously mentioned actions regarding installation of user-space applications and programming via overlays were executed in manual mode. However, it can be easily automated using bash scripts toolset and remote automatized SSH-sessions.

All the artifacts used during the experiments campaign for Terasic development board are available at the GitHub repository [15].

5 Conclusions

The developed information technology declares how SoC with programmable logic part can be configured in the cloud environment. The main advantage of the technology is that it supports configuration of the most important components. The technology supposes configuration during the boot process as well as configuration of a running instance. From the point of view of cloud technologies, that means that it can be sufficiently integrated into cloud infrastructure and automatize configuration of SoC and FPGA as its hardware part. The test performed on the testbed proved that configuration at the boot stage and configuration without reboot work correctly and all mentioned components are configured successfully.

References

1. Khethavath, P., Thomas, J.P. & Chan-tin, E.: Towards an efficient distributed cloud computing architecture. Peer-to-Peer Networking and Applications 10, 1152–1168 (2017).
2. Cornetta, G., Mateos, J., Touhafi, A., Muntean, G.M.: Design, simulation and testing of a cloud platform for sharing digital fabrication resources for education. Journal of Cloud Computing 8:12 (2019). doi: https://doi.org/10.1186/s13677-019-0135-x.
3. Talia, D.: A view of programming scalable data analysis: from clouds to exascale. Journal of Cloud Computing 8:4 (2019). doi: https://doi.org/10.1186/s13677-019-0127-x.
4. Banerjee, S., Adhikari, M., Biswas, U.: Design and analysis of an efficient QoS improvement policy in cloud computing. Serv. Oriented Comput. Appl. (2017). doi: https://doi.org/10.1007/s11761-016-0196-3.
5. Pullini, A., Conti, F., Rossi, D., Loi, I., Gautschi, M., Benini, L.: A heterogeneous multicore system on chip for energy efficient brain inspired computing. IEEE Trans. Circuits Syst. II Express Briefs. (2018). doi: https://doi.org/10.1109/TCSII.2017.2652982.
6. Musiyenko, M., Krainyk, Y., Denysov, O.: Reconfigurable decoder for irregular random low density parity check matrix based on FPGA. In: 2015 IEEE 35th International Conference on Electronics and Nanotechnology, ELNANO 2015 - Conference Proceedings. pp. 498–503 (2015). doi: https://doi.org/10.1109/ELNANO.2015.7146937.
7. Krainyk, Y., Darnapuk, Y.: Configurable Description of FPGA-based Control System for Sensor Processing. In: 2019 11th International Scientific and Practical Conference on Electronics and Information Technologies, ELIT 2019 - Proceedings. pp. 210–213 (2019). doi: https://doi.org/10.1109/ELIT.2019.8892313.
8. Krainyk, Y., Darnapuk, Y., Stelmakh, S.: Dataflow and system organization for image sensor data processing. In: 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering, UKRCON 2019 - Proceedings. pp. 689–694 (2019). doi: https://doi.org/10.1109/UKRCON.2019.8879934.
9. Skhiri, R., Fresse, V., Jamont, J.P., Suffran, B., Malek, J.: From FPGA to support cloud to cloud of FPGA: State of the art. International Journal of Reconfigurable Computing 2019 (2019). doi: https://doi.org/10.1155/2019/8085461.
10. Kolesnyk, I., Perepelitsyn, A., Kulanov, V.: Providing of FPGA Resources as a Service: Technologies, Deployment and Case-Study. In: CEUR Workshop Proceedings. pp. 63–68 (2017).
11. Kolesnyk, I., Kulanov, V., Perepelitsyn, A.: Markov model of FPGA resources as a service considering hardware failures. In: CEUR Workshop Proceedings. pp. 56–62 (2018).
12. Amazon EC2 F1 Instances, https://aws.amazon.com/en/ec2/instance-types/f1, last accessed 2020/02/20/.
13. Linux kernel source tree, https://github.com/torvalds/linux, last accessed 2020/02/20.
14. Tian, D., Choi, J.I., Hernandez, G., Traynor, P., Butler, K.R.B.: A practical intel SGX setting for linux containers in the cloud. In: CODASPY 2019 - Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (2019). doi: https://doi.org/10.1145/3292006.3300030.
15. Configuration of FPGA part in SoC, https://github.com/codebreaker7/SoCFPGAConfig, last accessed 2020/02/24.