HARDWARE RECONFIGURATION OF A SOC

Alexandru DINU, Adrian CRĂCIUN, Marian ALEXANDRU

Transilvania University, Braşov, Romania
(alexandru.dinu.cor@gmail.com, adrian.craciun@unitbv.ro, marian.alexandru@unitbv.ro)

DOI: 10.19062/1842-9238.2018.16.1.8

Abstract: *The continuous advance of science often leaves behind devices, and makes their usage obsolete. This can be observed, for example, in the medical domain, where the performance of devices achieved tremendous capabilities, or in the latest increase of power of computing: today's top-ranked smartphones are comparable in performance with the best desktop computers of the previous decade. Therefore, significant research efforts are directed at the reusability of current technologies by updating them according to new discoveries. Among the existing solutions for device reusability, there are two concepts which are highly ranked: the usage of system-on-chip devices and partial reconfigurability implementation. This paper analyzes the benefits of using these solutions both independently and combined.*

Keywords: *SoC, partial reconfigurability, reusability, hardware, software, fabric area*

1. INTRODUCTION

Nowadays, technology evolves faster than ever. Some of the big steps in technology are represented by:

– the revolution in communications, where VoIP (Voice over IP) calls replace more and more business conversations which before were held mostly through landline phones;

– the development of social networks which allows one to get connected with more than one person simultaneously, due to the fact that a message can almost instantly reach a receiver located in a distant region of the planet;

– the widespread access to the Internet;

– all the facilities brought by a smartphone.

Among the aspects considered when developing a new item in technology, ease of use, performance, reduced costs, and novelty of the product are probably the most important. These are also the aspects further considered in this paper, in which the theory and application of two modern technological concepts relying on current technological advance in electronics is presented: the Partial Reconfigurability of Hardware (an idea that appeared in the sixties but was implemented in the eighties [1] and SoC (System on Chip), first SoC having appeared in 1974 [9].

Then, in terms of the implementation methods of algorithms, the software is appreciated for its flexibility and possibility to be easily updated, through a new release. Also, it is well-known that, in hardware, operational latencies are significantly decreased compared to software, therefore the algorithms run faster. Moreover, hardware equals to parallel computing and, as long as there is available area of the implementation fabric, it can accustom many independent functions which will not influence each other’s execution time. The SoC combines both hardware and software, and this is the reason why this kind of device is very appreciated.

Also, partial reconfigurability proves its benefits in saving hardware area and, by making possible the moving of an increased amount of operations from software into hardware, it demonstrates its benefits in terms of processing speed.

2. A STUDY OF PARTIAL RECONFIGURABILITY

A reconfigurable device allows on-the-fly modification of one of his functionalities, while the rest of its functions remain unaffected. The implementation of a project which uses partial reconfigurability is realized in a way which is similar to the implementation of several non-configurable projects which use the same resources. In order to create this kind of project, hardware reconfigurable circuitry of the system (often represented by FPGA – Field Programmable Gate Array) is split in partitions. Some partitions are dynamically reconfigured during device operation, and some of them remain untouched to keep other functionalities available. The last ones are called static partitions, and one of their roles is to maintain the basic functionality of the device (i.e. for a bus slave, the static partition contains the communication protocol, which must be always available in order to respond to the master requests; the dynamic partition could contain different algorithms which represent the behavior of the slave). Fig. 1 shows the way in which a reconfigurable partition was delimited by the rest of design through the tool PlanAhead developed by Xilinx when we implemented partial reconfigurability on Artix 7 FPGA device.

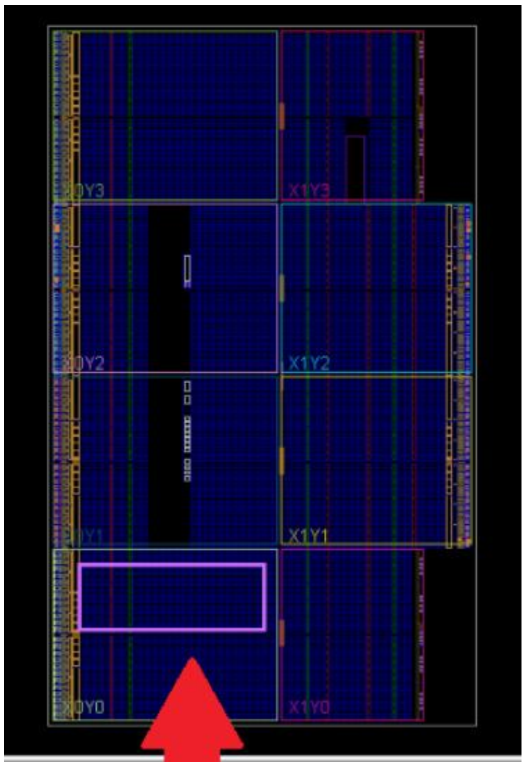

FIG. 1. Defining a reconfigurable partition (shown by the red arrow) in Artix 7 FPGA device using PlanAhead tool from Xilinx

One of the most important aspects is that the functions performed by the "static" partition will not be interrupted by reconfigurations of "dynamic" partitions of circuit. That creates a lot of advantages, some of them being in accordance with such principles as [7]:

– Reducing size of the required device to achieve a functionality, by implementing multiple functions which don't run in parallel on the same area.

– Because of this criterion, another two positive things result: reducing cost for device manufacturing; reducing power consumption by device while operating;

– Providing flexibility in the choices of algorithms or protocols available to an application;

– Enabling new techniques in design security;

– Improving the reconfigured devices' fault tolerance;

– Accelerating configurable computing;

– Implementing new devices, which can't be achieved in the absence of the partial re-configurability feature.

A proper environment to implement the principle of partial reconfigurability is represented by SoC devices, where the device operation is split between reconfigurable hardware and microprocessor. This way, the benefits come both from the flexibility of the software and the rapidity and parallel operation of hardware (according with [3]).

3. SYSTEM-ON-CHIP DEVICES

In a few words, a SoC consists of both hardware and software. Analyzing the details, according with [8], a SoC device represents an integrated circuit on whose logic fabric a lot of electronic system components are designed, from multiple domains: digital electronics, analog electronics, mixed-signal, radio-frequency elements and there can be other examples as well. A SoC device can be composed by a microcontroller (or microprocessor) and advanced peripherals such as a GPU (Graphics Processing Unit), wireless communication modules or another coprocessor. In general, there are three distinguishable types of SoC devices:

– systems which accommodate a microcontroller (i.e. *Xplained Evaluation Kit* for ATxmega128B1 microcontroller);

– systems which accommodate a microprocessor (a widespread example is the base board of smartphones);

– ASICs (Application Specific Integrated Circuit);

– systems which are user-configurable after their manufacturing, in order to perform a wide range of functions (for example PSoC devices from Cypress company or the boards which accommodate an embedded microprocessor on the FPGA fabric). The board *SoCKit - the Development Kit for New SoC Device* from Terasic company [5] is a member of the latter family of devices.

SoC devices are widely used in a lot of industry fields and for many purposes such as manufacturing of smartphones, tablet computers, wearables, digital cameras, wireless routers and the list can be continued (adapted after [10]).

The board mentioned before, *SoCKit - the Development Kit for New SoC Device,* is built around *FPGA Altera Cyclone V System-on-Chip.* A rough guide for the structure of this integrated circuit is presented in Fig. 2.

Between the two computational elements, embedded microprocessor and FPGA user-configurable logic, high communication speeds can be achieved. One of the reasons why Altera company (acquired by Intel Company) put the processor into the FPGA fabric is related to software limits: the applications which are built through a program which runs on a computing machine (in this case, the microprocessor) aren't very efficient, because these can produce an overhead to the processor even for simple tasks. Nevertheless, in software, different programs cannot run in parallel on the same core (however, these can run in pseudo-parallel, based on execution threads), and this fact creates another delay in achieving responses of the programs.

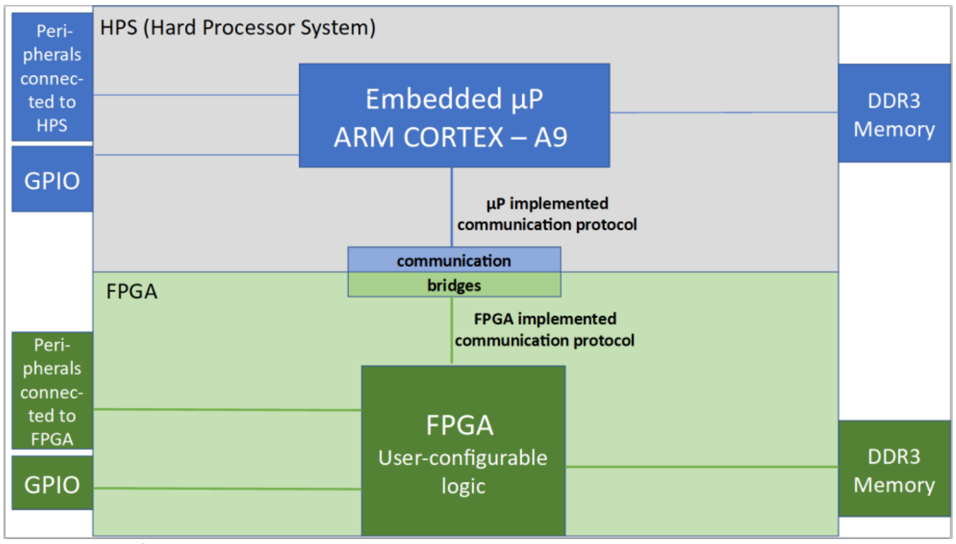
Peri-pherals connec-ted to HPS
GPIO
HPS (Hard Processor System)
Embedded µP ARM CORTEX – A9
DDR3 Memory
µP implemented communication protocol
communication bridges
FPGA
FPGA implemented communication protocol
Peri-pherals connec-ted to FPGA
GPIO
FPGA User-configurable logic
DDR3 Memory

FIG. 2. Structure of *FPGA Altera Cyclone V System-on-Chip* integrated circuit drawn on an indicative basis.

In hardware, it is possible for different algorithms to run in parallel (the condition is that the parallel running algorithms do not share the same resources, as a memory with only one communication port, for example), and the operations often can be implemented more effectively than in a software solution. Here, it is necessary to be stated that there are two major kinds of hardware implementation target devices: ASICs and FPGAs.

When there is a large production of devices (millions of items), the ASICs are more convenient because a low price per item is achieved. The downside is that ASIC isn’t upgradable, having a fixed structure and, thus, it can’t be updated with the evolution of technology. A lot of devices which sometimes were bought for a huge price, aren't used anymore now, because their technology is too old as compared to the newest discoveries and it can't be updated. For instance, the pneumatic extradural intracranial pressure monitor replaced in 1980 the existing device for detecting ICP (intracranial pressure) from a site outside the dura (the outermost and toughest membrane covering the brain) with its very complicated and fragile pressure sensors (adapted after [4]).

Furthermore, in both ASIC and FPGA devices, implementing complex algorithms becomes a problem because too much area is used. The FPGA devices are more expensive than ASIC devices, but these are flexible and these can be upgradable. Bringing together software (implemented on the microprocessor, which is very flexible, even in what concerns the configuration of the interface, and can be updated as many times as necessary) and hardware (the FPGA, where fast operations can be done) can represent a good solution to make a device more flexible, updatable and reconfigurable in order to achieve dynamic, high performance, and market-request-adaptable appliances.

4. PARTIAL RECONFIGURABILITY INSIDE FPGA DEVICES

Usually, the configuration methodology of an FPGA device implies the generation of a file called *bitstream*. It contains the instructions needed for specific configuration and interconnection of logic blocks (there are two types: CLB – Configurable Logic Blocks - and Configurable I/O blocks) inside the FPGA. Therefore, by creating only one *bitstream* file for entire configurable hardware, it is considered that the gate array represents an atomic entity (adapted after [2]).

In contrast to this idea, partial reconfigurability methodology assumes that an FPGA device is divided in a minimum of two regions, one being called the "static" region and the other being called the "dynamic" region. The "static" region is the FPGA part which is configured only at start-up and after that remains untouched during device operation. The "Dynamic region" is the FPGA part dynamically reconfigured, at multiple times, and with different algorithm versions or with different steps of the same algorithm.

In order to implement partial reconfigurability, the following steps must be considered: after programming the FPGA with a complete *bitstream* file (through this file all logic blocks on the fabric get configured), through partial *bitstreams* one or more partitions (declared "dynamic" before) can be modified in order to extend functionality of device, as shown in Fig. 3.

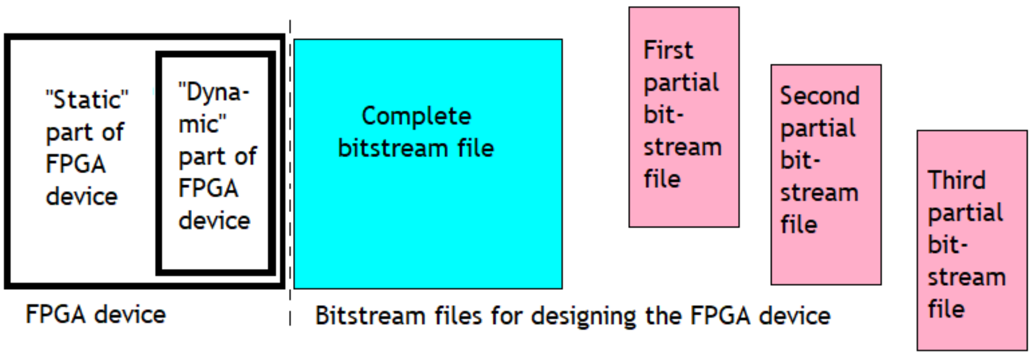
"Static" part of FPGA device
"Dyna-mic" part of FPGA device
FPGA device
Complete bitstream file
First partial bit-stream file
Second partial bit-stream file
Third partial bit-stream file
Bitstream files for designing the FPGA device

FIG. 3. Schematic of how partial reconfiguration is achieved

5. DIFFERENT WAYS TO IMPLEMENT PARTIAL RECONFIGURABILITY IN SOC

As written above, hardware partial reconfigurability allows the consecutive implementation of different functionalities in the same FPGA partition. Hardware configuration through partial *bitstream* files can be human-driven or realized automatically by microprocessor.

In accordance with [6], there are three possible ways to configure an FPGA: from an external configuration flash memory; with the Quartus Programmer tool; from HPS software.

In this paper, FPGA device configuring through HPS (Hard Processor System) which is on the same fabric was adopted. This method allows device configuration in a remote manner, through the modification of a register by a human, or even automatically, when the register is modified by hardware or software in SoC. In the latter situation, an automated process with negative feedback can be implemented.

6. LOGIC DISTRIBUTION BETWEEN HARDWARE AND SOFTWARE

The information provided so far shows that using both hardware and software in a project brings a lot of advantages. But how to split the logic of a project? Which are the parts which can be implemented efficiently in the hardware, and which parts could be better placed in the software? Below, some guidelines in order to answer these questions are listed.

Mainly, the hardware must contain drivers for peripherals used by designed system. Generally, these drivers communicate with microprocessor through addressable registers by the software; these are used as a medium were acquired data from inputs or data to be transmitted to outputs is stored.

Also, in hardware, the protocol which allows communication with the processor (often, this communication is realized through a bus) is also implemented. Also, the hardware can be used as a support for data processing algorithms. This way, the algorithms could run faster, but a withdrawal is represented by the used area which, as the other resources, is limited. If these algorithms are split in chunks, and every part of them is consecutively downloaded through a partial *bitstream* file into a reconfigurable partition on the FPGA, a big amount of fabric area is saved. In this case, a temporary memory must be used for saving the results produced by an algorithm step in order to be transmitted to the next algorithm step. Also, it must be mentioned that the logic which is in charge with the interface protocol of the microprocessor (it includes registers seen by software, too) must be maintained into the static hardware partitions. This way, registers are always available for software usage.

The software is tasked with the running of data processing algorithms. Also, it reads the data supplied by the hardware and sends back necessary data and commands to it. When necessary, updating its algorithms is a very simple process, consisting in creating a new programming file for the microprocessor.

The logic distribution concept between hardware and software is also represented in Fig. 4.

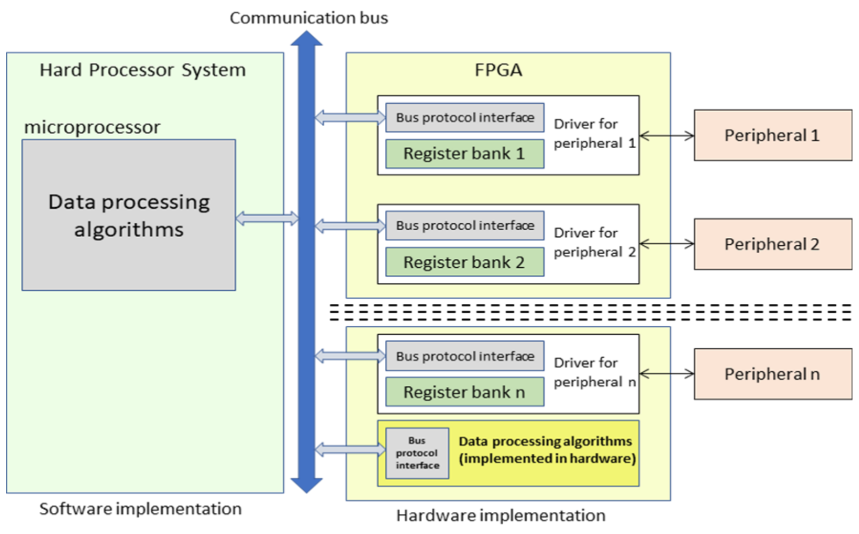

FIG. 4. Representation of the proposed distribution of logic between hardware and software

7. EXAMPLES OF PROJECTS THE EFFICIENCY OF WHICH CAN BE INCREASED BY USING HARDWARE RECONFIGURATION OF A SOC CONCEPT

The main advantages of partial reconfigurability implementation in a SoC relate to area saving, the continuity of device operation during configuration and the increased speed of algorithm execution by moving the logic from software to hardware. Below are some examples to demonstrate these advantages of the method.

1. A domain where partial reconfigurability can also positively influence an application is image processing. Let's suppose that a new version of intelligent military goggles is created. They must adapt instantly at every light changing event: when there is only darkness, infrared vision and thermal imaging must be activated, when a little amount of light appears, infrared vision is partially deactivated but thermal imaging is still working, when a heat source appears, both previous features are deactivated and an algorithm based on sound waves is activated.

The fast changing between algorithms can be achieved through partial reconfigurability. Also, supposing that these algorithms are very complex, their implementation will occupy a large fabric area. But great savings are realized, if these algorithms are implemented into FPGA fabric only during their active usage.

2. Another situation where partial reconfiguration can be really useful is using it into specialized communications terminals used by the military. Let us suppose that a device, which must be always connected to the GSM network in order to transmit critical information, is mounted on a fast car which gathers data from a large area. Because there is a lot of data which must be transmitted, the best available quality of transmission channel must be achieved. In this scope, the communication terminal must be able to fast switch from 4G to 3G network or even to 2G network, if only the last can represent at a given point a stable communication channel with a good signal strength. When a higher communication protocol becomes available, the terminal should be able to switch fast to it. Hereof, partial reconfigurability can be used, a single communication protocol being active at a time.

3. In another case, a SoC device is attached to a robot which works in a human-inaccessible place in order to analyze the environment. Depending on the ground type he is moving on, the robot needs to use continuous tracks, narrow wheels, wide wheels, climbing claws etc. Through infrared sensors, the robot captures data about ground type and analyze it. Therefore, the code of the ground is retained into a register, and based on it, the software is able to reconfigure the proper FPGA partition with the algorithm that the robot must use in order to keep going on.

In the situations above, partial reconfigurability is supposed to be automatically done by software based on specific input stimuli.

8. PROOF OF CONCEPT

To prove this concept on our side, we started to create a simple project using the board *SoCKit - the Development Kit for New SoC Device* consisting of a temperature regulator. The peripherals used are a fan driven by a simple DC motor, the DS18B20 temperature sensor with digital output and an electronic heating element (we have chosen a BD652 integrated circuit which becomes hot due to a current of 0,2 – 0,5A which is driven to it). A schematic of the project is shown in Fig. 5.

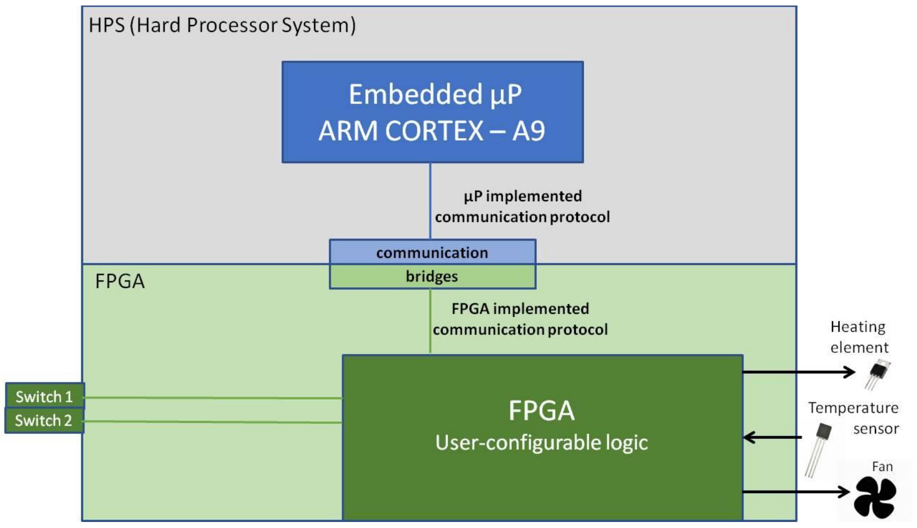
HPS (Hard Processor System)
Embedded µP ARM CORTEX – A9
µP implemented communication protocol
communication bridges
FPGA
FPGA implemented communication protocol
FPGA User-configurable logic
Switch 1
Switch 2
Heating element
Temperature sensor
Fan

FIG. 5. Schematic of our project started in order to prove partial reconfigurability of a SOC concept

We communicate with the digital temperature sensor DS18B20 through its 1-Wire® protocol. There are multiple operations which the sensor is able to accomplish [11], but we only used the following three: writing scratchpad memory of the sensor, converting temperature in digital representation and reading scratchpad memory of the sensor. These operations can be realized by following the steps presented in Fig. 6.

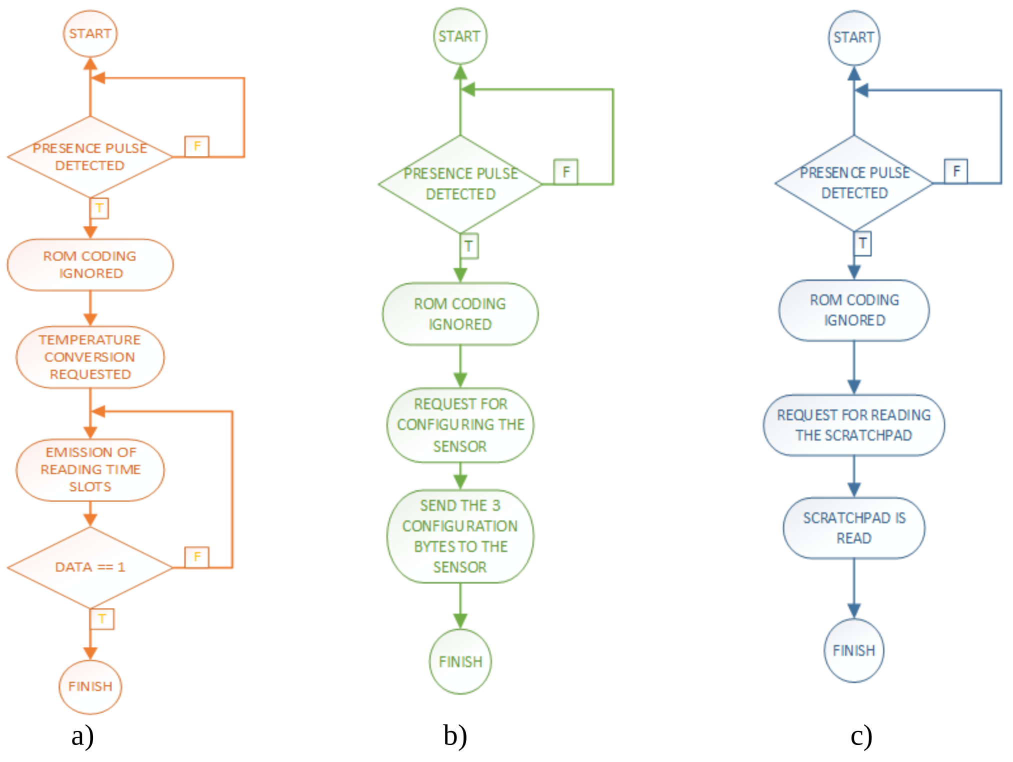

FIG. 6. Steps which must be followed, according to the sensor protocol, in order to:
a) initiate current temperature conversion by sensor to digital representation;
b) configure resolution of the temperature acquired;
c) read the temporary memory of the sensor.
The succession of these three steps makes possible reading the temperature from sensor.

In order to communicate with both microprocessor and temperature sensor in the three ways above, a "dynamic" hardware partition is created into FPGA, and a module for communication with the sensor is implemented inside it. It is connected with the data line of the temperature sensor via an input/output pin. Also, software sends commands to this module through a communication protocol (we have chosen APB – Advance Peripheral Bus – protocol by AMBA). In the example, the microprocessor sets a start bit which is used for starting current operation flow and reads the finish bit which is used to signal that the operation is finished. The start and finish bits belongs to a *control register*: the start bit can be only asserted by software and it is reset in the next clock cycle and the stop bit can be only read by software. Also, there are two more registers which can be accessed by software: the *temperature register* (read only by software) which is 16 bits width and contains valid data only when sensor memory reading operation is implemented and the *configuration register* (written only by software) which is 8 bits width and contains the resolution setting for temperature digital representation (its resolution can be 9, 10, 11 or 12 bits). This register must contain valid data when sensor memory writing operation is running.

Every 10 seconds, the microprocessor starts a reading temperature cycle. To do that, it firstly configures the dynamic partition with the algorithm that contains the operation steps for writing the scratchpad memory and assert the start signal. After finish signal becomes logic "1", the microprocessor loads into the "dynamic" partition the algorithm which contains the operation steps for converting temperature and monitors the finish signal as in the previous step. The last loaded algorithm is the one which reads the memory of the temperature sensor (called scratchpad memory). The data from the memory is saved into *scratchpad data register* which is also existent on the reconfigurable partition, but is used by hardware solely. The two least significant bytes from the *scratchpad data register* contain the value of temperature, are these are therefore copied in temperature register. After the finish signal is asserted again by the hardware, the microprocessor reads the temperature register and, depending of the temperature value, generates a convenient duty cycle for a PWM signal which is used to command the simple DC motor of the fan. The driving operation of the fan is implemented into a "static" partition of the FPGA, and this partition contains a register (written only by hardware) which contains a code of duty cycle value (that can vary between 0% (stopped) and 100% (full speed) with a resolution of 5%).

The heating element driver is implemented into another reconfigurable partition. The BD652 integrated circuit (which actually represents two power transistors in Darlington configuration) can be either heated or cooled (the cooling is realized by stopping the current which passes the integrated circuit) by software in a specific manner programmed by us, or the heating can be started and stopped manually through a switch (Switch 1 in Fig. 5) which is available on the project board. Also, the decision of reconfiguring the dynamic partition with one of the above methods is made by the software which reads a 1-bit register which is set and unset through a second switch (Switch 2 inside Fig. 5) available on the project board.

In Fig. 7 the structure of the FPGA which also was described above is represented. By implementing this project, we are able to prove that partial reconfiguration is a concept that saves FPGA fabric area. Also, because the project is implemented on a SoC device, the dynamic reconfiguration can be done automatically, through the software, in this case creating a well-operating independent device with negative feedback.

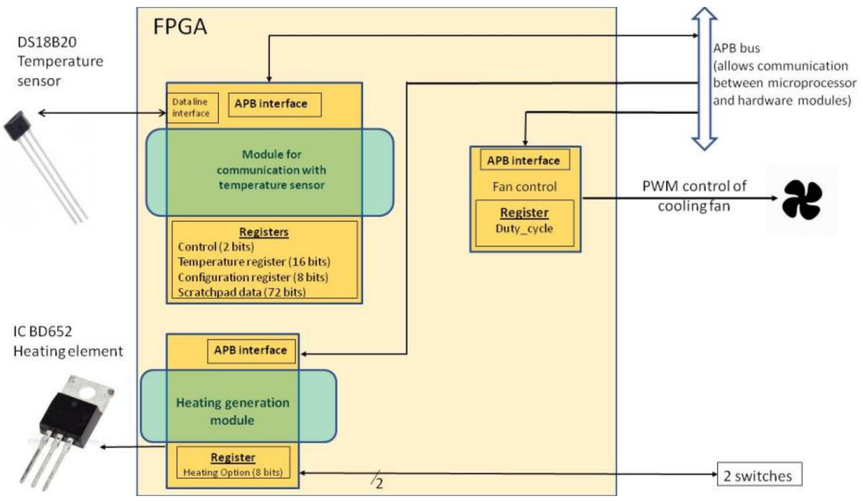
FPGA
DS18B20
Temperature
sensor
Data line interface
APB interface
Module for communication with temperature sensor
Registers
Control (2 bits)
Temperature register (16 bits)
Configuration register (8 bits)
Scratchpad data (72 bits)
APB bus
(allows communication between microprocessor and hardware modules)
APB interface
Fan control
Register
Duty_cycle
PWM control of cooling fan
IC BD652
Heating element
APB interface
Heating generation module
Register
Heating Option (8 bits)
2
2 switches

FIG. 7. The FPGA structure used in the project; reconfigurable partitions are colored in cyan.

9. CONCLUSIONS

For the increasing need of having updatable devices which are able to embed the latest discoveries, SoC devices can be chosen as a proper environment to develop competitive devices. And, in order to decrease logic fabric consumption, and therefore being able to move more logic from software to hardware (thus gaining processing speed), partial reconfigurability is always a solution which has to be considered.

This way, technological improvement will be welcomed not only by device manufacturers, who will benefit from shorter time to market cycles, but also by developers, who will be able to prove their concepts faster, and by end-users, who will benefit of enlarged time periods between the acquisitions of products created for the same purpose.

REFERENCES

[1] Radunovic, Bozidar, *An overview of advances in reconfigurable computing systems,* Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on. IEEE, 1999;

[2] Ashenden, Peter J. *The designer's guide to VHDL*. Vol. 3. Morgan Kaufmann, 2010;

[3] Popescu, V., – *Multimedia*. Editura Tehnică Bucureşti, anul 2000, p.78;

[4] National Academy of Engineering and Institute of Medicine, *New Medical Devices: Invention, Development, and Use,* Washington, DC: The National Academies Press, 1988, p.34;

[5] ***, *Overview page for SoCKit - the Development Kit for New SoC Device*, online resource accessed in April 24, 2018, at the address http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English &CategoryNo=205&No=816&PartNo=1;

[6] Martin, C., *GSRD v13.1 - Programming FPGA from HPS,* online resource accessed in April 24, 2018, at the address https://rocketboards.org/foswiki/Documentation/GSRD131ProgrammingFPGA*;*

[7] Xilinx, *Partial Reconfiguration User Guide (UG702),* (2012), p.8, online resource accessed in April 24, 2018 at the address https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf;

[8] Daga, A., *SOC (System On Chip),* online resource accessed in April 24, 2018, at the address https://techbuddiesbyajay.blogspot.ro/2017/10/soc.html;

[9] ***, *1974: Digital Watch is First System-On-Chip Integrated Circuit,* online resource accessed in April 24, 2018, at the address http://www.computerhistory.org/siliconengine/digital-watch-is-first-system-on-chip-integrated-circuit/;

[10] Neagu, C., *Întrebări simple: Ce este un SoC (System on a Chip)?,* online resource accessed in April 24, 2018, at the address https://www.digitalcitizen.ro/soc-system-chip;

[11] Maxim Integrated, *Datasheet for the Programmable Resolution 1-Wire Digital Thermometer,* online resource accessed in April 27, 2018 at the address https://datasheets.maximintegrated.com/en/ ds/DS18B20.pdf.