Received August 27, 2020, accepted September 6, 2020, date of publication September 9, 2020,
date of current version September 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022955

Automation of Dynamic Power Management in FPGA-Based Energy-Constrained Systems

MICHAL ŠKUTA, DOMINIK MACKO, (Member, IEEE), AND KATARÍNA JELEMENSKÁ, (Member, IEEE)
Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, 842 16 Bratislava, Slovakia

Corresponding author: Dominik Macko (dominik.macko@stuba.sk)

This work was support in part by the Ministry of Education, Science, Research and Sport of the Slovak Republic (Incentives for Research and Development, under Grant 2018/14427:1-26C0, and in part by the Eset Research Centre.

ABSTRACT The era of the Internet of Things comes with a huge number of interconnected communicating devices, which are often rather limited on the energy supply (e.g. battery powered or energy harvesting). Therefore, the pressure on energy efficiency of their operation (influencing lifetime or amount of functions) is especially crucial. In spite of a growing number of IoT devices, there still are many applications that are very specific and their market is quite limited. This is where the FPGAs offer a good alternative to dedicated application-specific chips, which would be too costly for such a purpose. Therefore, we target the power-management automation that simplifies energy-efficient design for FPGA platforms. A designer is then able to specify just an abstract power management in the commonly used SystemC model and it is automatically transformed in the more-complex form acceptable by a specific FPGA device. The proposed simplification and automation shortens the time-to-market of energy-efficient IoT products and prevents possible human-errors that could be otherwise introduced to the design. The alleviated verification and debugging spare even more time in the development process. The experiments have proved the benefits of the proposed automation method.

INDEX TERMS Design automation, energy efficiency, FPGA design, Internet of Things, power management.

I. INTRODUCTION

The market forces regarding cost reduction and time-to-market of the products increase popularity of the FPGA (Field-Programmable Gate Array) devices. These devices can be programmed to perform some function in hardware (i.e. at hardware speed); however, they are premanufactured, thus the hardware designers do not need to wait for their designs to be manufactured as a chip to integrate them in their products. Therefore, FPGAs are often used for hardware acceleration of some specific tasks. In contrast to ASIC (Application-Specific Integrated Circuit) or PLD (Programmable Logic Device) chips, the hardware function of the FPGA device can be changed by its reconfiguration. This is very useful for space applications (e.g. in satellites or in space-exploration missions devices), since it provides a way to update the device function remotely [2]–[4]. The reconfiguration possibility also widens a high interest of using the FPGA technology as a prototyping platform, for functional verification and characterization of hardware designs that will be later fabricated as ASIC chips. To "try-out" some new circuit is just not feasible in ASIC technology, mainly due to time and cost resources to manufacture single or a few chips. Modern FPGAs can be very complex circuits, which often integrate various special-purpose components (e.g. multipliers, memories, specialized processors) to support as many applications as possible (to widen market of their own). These additional components are not always used (in each configuration); however, they increase power requirements of FPGA devices.

Energy costs, as well as the green-computing ideology, put the energy efficiency of the hardware designs to the forefront [5], [6]. Also, the rapidly growing market of the Internet of Things (IoT) strictly requires low-power operation of the devices limited by the energy source (e.g. battery powered or energy harvesting) [7], [8]. Therefore, the energy demands of FPGA-based systems have to be reduced as much as possible and each hardware design should focus on energy

The associate editor coordinating the review of this manuscript and approving it for publication was Kim-Kwang Raymond Choo.

efficiency – if not for low-power application market, then to reduce costs, or at least for sustainability reasons. The usage of well-known power-reduction techniques, such as power gating of voltage scaling, is limited by the hardware architecture of the FPGA device itself. For example, a multivoltage design technique can be used only if the FPGA platform supports multiple voltages and the power-gating technique cannot be used in the application design if not enabled by the FPGA device. Nowadays, the most efficient power-reduction techniques are commonly applied by some sort of dynamic power management, enabling to control power in various parts of the system depending on current tasks executed by the application (i.e. during runtime). However, it complicates the design and is often just too difficult to be applied by inexperienced designers. Therefore, various power-reduction techniques are not used, even if they could be, wasting the valuable energy.

There are various research works focused on increasing the energy efficiency of FPGA-based systems. For example, a modification of the place-and-route process can reduce power requirements of the FPGA interconnect, as proposed in [9]. This method is useful when synthesizing the application design for the selected FPGA platform; however, it cannot optimize the power consumption during runtime. Another method [10] utilizes the autonomous power-gating technique in a fine-grained manner, targeting LUTs (Look-Up Tables) in asynchronous FPGAs. It thus focuses on the FPGA-chip architecture rather than on reducing power in the application design. A similar focus was preferred by [11] that uses power gating in so-called mega cells to optimize power requirements. However, it also cannot be utilized for dynamic power management by the application design. Another modification of the FPGA architecture was proposed in [12]. Although it enables a sort of power management, it cannot be used on commercial FPGA devices. Dynamic power management, applying software-driven power gating, was used in [13], which targeted the Xilinx ZYNQ platform. However, it cannot be used on devices that do not contain any embedded processor in addition to the FPGA chip. A method utilizing clock domains and algorithm partitioning was applied in [14]. Although such architectural modifications can indeed help to reduce power requirements, they do not enable managing power dynamically during runtime. There also exist FPGA-based methods that enable to estimate power consumption of system prototypes [15], or methods that demonstrate benefits of various technologies in low-power system design [16]. Neither one although enables dynamic power management.

Incorporating dynamic power management is definitely a challenge for design teams, since it substantially increases complexity of the system. To improve design productivity of complex systems, the International technology roadmap for semiconductors [17] suggested the adoption of higher abstraction in the design process. The ESL (Electronic System Level) progressively becomes the industrial design starting point, especially in FPGA-based systems [18]–[22]. Although the ESL-based design methods definitely increase the productivity in programming FPGAs, they tend to omit the energy point in the high-level synthesis, focusing on the functional aspect of the design. There were some methods already developed for adoption of power management into ESL specification [23]–[26]. However, all existing methods are focused on automatically generating power intent at lower levels in a standard format, suitable only for the ASIC technology. In our previous work, we have already outlined the idea of power-management automation in FPGA-based systems [27]. It was mainly focused on the power-management unit (PMU) automated generation, based on an ESL specification. However, important parts of the power management, such as clock gating, isolation cells, or other support logic, have not been generated. Thus, a high amount of manual effort was still required to successfully finish power-management integration into the design.

In this work, we extend our previous results [1] into a new method, which enables to analyze the abstract ESL specification of the power-managed system described in SystemC/PMS [26] and automatically synthesizes Verilog model of the PMU that enables to scale the frequency of the system, as well as other required logic, such as synchronization elements between clock domains of the system. In comparison to [1], which proposed the method and provided early experimental verification of manual power-management techniques application to an FPGA-based design, this work is accompanied by the fully functional automation tool that has automatically synthesized the power management comparable to the previously used manual one (a golden model). The key contribution of the work is that the synthesized power management is application-specific (i.e. optimized, to accomplish just the intent specified in SystemC/PMS), instead of the use of general-purpose power controllers. Just to be clear, this article does not introduce a new power-reduction technique, it focuses on simplified application of the existing popular techniques that are rather complicated for introduction into an FPGA-based system design. Usage of abstract power-management specification significantly simplifies the ESL modeling, and thus shortens the design time. Moreover, the automated transition from ESL to RTL (Register-Transfer Level) abstraction levels prevents possible human errors that could be introduced to the design, and thus reduces debugging effort. The proposed method simplifies and speeds-up the design process of low-power and energy-efficient FPGA-based systems. This way, it enables even designers unfamiliar with power-reduction techniques to target energy efficiency, making their products to run longer on batteries, integrate more functions into the products while not exceeding energy limits, and contribute to sustainability.

The remaining part of the article is structured as follows. The next section includes an overview of the background, regarding the SystemC/PMS specification. Section III describes the proposed power-management automation method for FPGA-based systems. In Section IV, the experimental results supporting the benefits of the proposed method are reported and discussed. And in the last section, the work is concluded.

II. SystemC/PMS SPECIFICATION

The abstract power-management specification in SystemC/PMS was introduced in [26]. It is based on the standard power concepts defined by the UPF (Unified Power Format) standard [28], which are introduced into a system-level model described in SystemC [29] in an easy-to-use way. The key utilized concepts include power states, power domains, power modes, and power policy. There are five abstract states predefined by SystemC/PMS, which are summarized in Table 1 along with the corresponding power-reduction techniques that are expected to be applied by the power states. A power domain groups multiple components that are always operating in the same power states (the state changes simultaneously in all components). Power modes represent allowed combinations of power states in all power domains (one power state for each power domain) – i.e. like a system-wide power state. The power policy specifies when and how the system switches between power modes. Unlike in UPF, the SystemC/PMS abstract power state is not limited to the specification of the supply-voltage level only, but also the frequency level. Such a frequency-voltage pair is called a performance level, which must be specified for each active power state (i.e. either NORMAL or DIFF_LEVEL).

TABLE 1. The predefined abstract power states in SystemC/PMS.

Power state	Expected power-reduction techniques to be used	Description
NORMAL	no	The power domain operates at the basic voltage and frequency levels.
DIFF_LEVEL(#)	voltage scaling frequency scaling voltage islands clock islands	The power domain operates at the adjusted voltage and/or frequency level. The number # enables to specify multiple states of this kind.
HOLD	clock gating operand isolation	The power domain suspends its operation (i.e. the state is not changing).
OFF	power gating	The power domain is switched off.
OFF_RET	power gating with state retention	The state of the power domain is retained, while it is not powered.

To illustrate how such SystemC/PMS specification of architectural power management looks like, we provide an example in Fig. 1. The abstract power management is specified in the top module called *example_top*. The declaration part of the module contains a declaration of available power domains and power modes. However, a specification of power states for power domains and power modes, along with the assignment of components to the power domains, is contained in the functional part of the module, such as

```
#include "systemc.h"
#include "pms.h"

SC_MODULE(example_top){
    PowerDomain PD_CPU,PD_UART;
    PowerMode PM_High, PM_Low;
    processor_component CPU;
    interface_component UART;
    ...
    SC_CTOR(example_top):CPU("CPU"), UART("UART"){
        PD_CPU = PD(OFF,DIFF_LEVEL(1));
        PD_UART = PD(NORMAL,DIFF_LEVEL(1));
        Set_Level(DIFF_LEVEL(1), 1 V, 20 MHz);
        Set_Level(NORMAL, 0.7 V, 50 kHz);
        PD_CPU.AddComponent("CPU");
        PD_UART.AddComponent("UART");
        PM_Low = PM(OFF,NORMAL);
        PM_High = PM(DIFF_LEVEL(1),DIFF_LEVEL(1));
        POWER_MODE = PM_Low;
        ...
    }
}
```

FIGURE 1. A partial example of power-management specification in SystemC/PMS.

its constructor. The special variable *POWER_MODE* holds information about the current power mode of the system, and it can be switched in some other SystemC process of the functional model.

It must be noted that SystemC/PMS represents just an abstract specification of power management, it does not model the effects of power management. It means that it specifies "what" should be achieved, but "how" it is achieved must be modeled and implemented at lower abstraction levels.

As previously mentioned, the SystemC/PMS specification is based on UPF concepts. The original goal was to simplify power-intent specification by abstracting from unnecessary details at the system level, and then automatically synthesize a more-complex equivalent specification in UPF. However, the UPF standard is intended for ASIC-based chips (Application Specific Integrated Circuits) and its concepts are not easily applicable to FPGA-based systems. The whole power-management synthesis algorithm must be adjusted to this kind of devices.

III. THE PROPOSED POWER-MANAGEMENT AUTOMATION

An initial idea of adopting power-management simplification benefits offered by SystemC/PMS to FPGA-based system was presented in [27]. This work extends the previous idea by automated synthesis of FPGA-supported power-management execution logic, such as clock gating, isolation or synchronization elements, in addition to the power-management unit. The proposed low-power FPGA-based application design flow is illustrated in Fig. 2. The red dashed line marks the process in the design flow, targeted by the proposed automation method (the flow step 4b in the figure). Similarly to the ASIC-based flow [26], we expect the design process to start at the highly abstract ESL, as a crude model specification (step 1). The abstraction-refinement process (step 2)

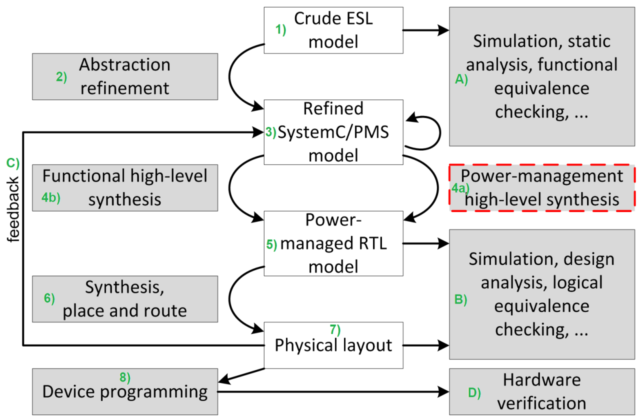
1) Crude ESL model
2) Abstraction refinement
Refined 3) SystemC/PMS model
C) feedback
Functional high-level 4b) synthesis
Power-management 4a) high-level synthesis
Power- 5) managed RTL model
6) Synthesis, place and route
7) Physical layout
8) Device programming
A) Simulation, static analysis, functional equivalence checking, ...
B) Simulation, design analysis, logical equivalence checking, ...
D) Hardware verification

FIGURE 2. The proposed low-power FPGA-based application design flow.

is then used to specify abstract power management directly into the functional model using SystemC/PMS (step 3). Since the abstract power-management specification does not affect the system function, the designer does not need to worry that it will corrupt the simulation results (the optional step A). The designer can rely on the previously developed abstract power-management static analysis [30] to validate the specification. Just before the functional high-level synthesis (HLS) takes place (step 4b), the power-management HLS extracts the power-related information from the ESL model (step 4a), in order to be synthesizable by commonly used HLS tools (e.g. Vivado HLS). After the functional RTL model is synthesized, the power-management HLS automatically generates the appropriate power-management components and integrates them into the functional model (step 5).

The synthesized power-managed RTL model can then be verified during a functional simulation (the optional step B). After logical synthesis and place and route process (step 6), the design can be analyzed for power consumption and resource utilization (the optional step B). The analyzed information (the optional step C) can be then used (while taking into account the trade-off between power, performance, and area) to adjust the abstract power-management specification (step 3) and resynthesize the model (steps 4 and 5). If the functionality was not modified, the functional HLS process (step 4b) does not need to be run again. This speeds-up power-management exploration.

The proposed method was implemented into a tool, called pmuToFPGA, which automates the power-management synthesis process. It was implemented in the Python programming language, version 3.7, using IDE editor PyCharm CE at the Mac OS X platform. Since Python is a multiplatform language, the tool is also usable at different platforms, such as Linux or Windows. The modular architecture of the tool is illustrated in Fig. 3. It consists of five separate components with dedicated functions. Such a modular design makes the tool flexible for extension and future modifications.

The Controller component interconnects all the other components and enables to exchange data among them in an efficient and meaningful way. For example, it enables

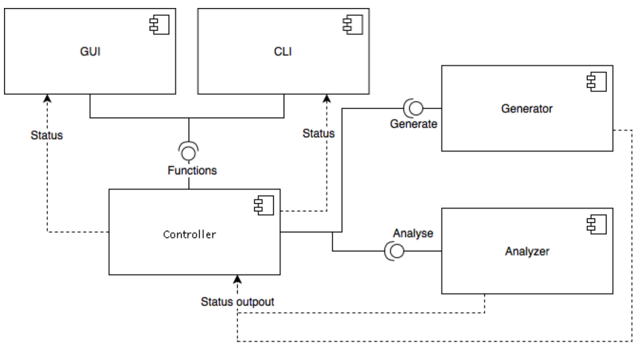

FIGURE 3. Components of the automation tool.

to obtain user-defined inputs via a user interface (either GUI - Graphical User Interface or CLI - Command Line Interface), redirects them to the other components, and provides the synthesized model to the user again via a user interface.

The Analyzer component is used to load the input ESL specification of the system model in SystemC/PMS, extract the power-related data from the specification, and fill them into an intermediate structure. For this structure to be kept in a simple and easy-to-use form (for a human as well as for a computer), we have proposed a JSON-based format of the internal structure, as illustrated in Fig. 4.

```
{
"levels": {
    "NORMAL": [ 5.0, 12.0 ],
    "DIFF_LEVEL(1)": [ 5.0, 0.12 ]
},
"power_domains", {
    "PD_1": [ [ "RS232" ], [ "NORMAL", ] ],
    "PD_2": [ [ "CPU" ],
        [ "DIFF_LEVEL(1)", "OFF" ] ],
    "PD_GEN": [ [ "RS232_copy" ], [ "NORMAL", ] ]
},
"signals": {
    "cpu_dout": {
        "PD_2": [
            [ "CPU", "data_out", "OUTPUT" ] ],
        "PD_1": [
            [ "RS232", "tx_byte", "INPUT" ] ],
        "PD_GEN": [
            [ "RS232_copy", "tx_byte", "INPUT" ] ]
    },
    ...
},
"components": {
    "RS232": [ "PD_1", "serialType" ],
    "CPU": [ "PD_2", "cpuType" ],
    "RS232_copy": [ "PD_GEN", "serialType" ]
},
"power_modes": {
    "on_mode": [
        [ "PD_1", "NORMAL" ],
        [ "PD_2", "DIFF_LEVEL(1)" ],
        [ "PD_GEN", "NORMAL" ]
    ]
}}
```

FIGURE 4. An example of the internal JSON structure.

Firstly, Analyzer has to find all the specified performance levels in SystemC/PMS model. The information about the power state name, the supply voltage, and the frequency value, is stored. Since the performance levels of inactive power states (i.e. HOLD, OFF, OFF_RET) are not specified explicitly, the Analyzer must deduce them. Then, the Analyzer component finds all the specified power domains, along with the assigned components and the allowed power states for each domain. Also, all the signals interconnecting the components of the modeled system must be indexed, and the information about which components are connected and the direction of the communication are stored. It is inevitable for synchronization and isolation purposes. Last but not least, Analyzer must find all the specified power modes and store the information about the selected power states for each domain in each mode.

The analyzed data in the internal JSON structure are then forwarded to the Generator component. Using the internal structure, Generator synthesizes the power-management unit, driving all the clock signals according to the specification, as well as the control signals for other power-management elements. Generator also synthesizes the power-management support logic, such as synchronizers. For the code synthesis, the Mako-based templates [31] are used. To show an example, a portion of the template for the PMU synthesis is provided in Fig. 5. It is used to create the main body of the synthesized PMU; however, individual parts of the PMU are synthesized by utilization of another more-complex template (the whole synthesis code is available on GitHub [32]).

The Generator component is configured by a configuration file, which is used by the designer to set some synthesis parameters for a specific FPGA device (e.g. whether the PLL module should be used or not, or what are the acceptable boundaries of the generated clock frequencies). An example of the configuration file is provided in Fig. 6. The synthesized PMU uses the PLL unit, if *use_pll* attribute is activated. Input parameters for the PLL to generate a specific frequency are computed using Algoritm 1. If the PLL cannot generate the clock frequency in the specified boundaries, the divider is synthesized to adjust the frequency value of the main clock signal (the *divide_clock* attribute must be activated). The *divide_pll* attribute enables dividing the PLL output signal. The main clock signal has precedence before the PLL-generated signal. Using the *strict_freq* and *all_freq* attributes, it is possible to specify whether the power domains are limited only to the frequencies deduced from the specified abstract power states, or they can use all the generated frequencies.

The configuration file also enables to activate the reconfiguration support. The synthesized PMU is then able to not only switch between power modes, but also switch between multiple configurations. It is usable, for example, as a replacement for unsupported power gating by some FPGA devices (like the selected FPGA device iCE40 for our prototype). Instead of powering down some component, it is possible to switch to the alternative design, not including that component. Although this functionality is fully supported by the synthesized PMU, the alternative designs (i.e. configurations)

```
<%namespace name="pmu" file=
"pmu_inner_func.mako"/>
module power_manager(
    input clk,
    input reset,
${pmu.make_control_input()}\
${pmu.make_pds_clock()}
);
// wire and pll module
wire pll_clk;
${pmu.make_pll()}
// reg and warmboot module for
// reconfiguration
${pmu.make_reconf_setter()}
${pmu.make_warmboot()}
// Generated levels
${pmu.make_levels_defines()}
// Inner registers for divider
${pmu.make_counter_reg()}
${pmu.make_counters()}
// Power domains registers and assigns
% if strict_freq == True:
${pmu.make_strict_setters()}
${pmu.make_strict_assigns()}
% else:
${pmu.make_setters()}
${pmu.make_assigns()}
% endif
// Synchronous sequential logic
% if divide_clock:
always @ (posedge clk) begin
    if (reset) begin
${pmu.reset_counter(2,False)}\
    end
    else begin
${pmu.make_counting(2,False)}
        // Controllers
${pmu.make_controller(2)}
    end
end
% endif
% if divide_pll:
always @ (posedge pll_clk) begin
    if (reset) begin
${pmu.reset_counter(2,True)}
    end
    else begin

${pmu.make_counting(2,True)}
    end
end
% endif
endmodule
```

FIGURE 5. **The Mako-based template for PMU synthesis.**

must be created by the designer – their creation is not yet automated. After the synthesis of PMU and synchronizers, the Generator component must find a suitable place in the top module of the modeled system for their automated integration. According to the direction of a specific signal of some component, it is renamed to the signal generated by the synchronization element (e.g. signal_synced).

After the power-managed model is automatically synthesized, a few manual modifications are required. The designer needs to find the place in the code, where the synthesized power-management modules have been inserted. To simplify the code search, we have identified such a place by a commentary of ‘‘Start of auto-generated components PMU + synchronizers’’. Since the main clock signal can have various

```
{
    "__class__": "DeviceConf",
    "__module__": "structs.device",
    "name": "dev_conf_0",
    "clk_freq": 12.0,
    "device_type": "ice40",
    "use_pll": false,
    "pmu_type": "COMBINED",
    "pll_clk_in_min": 10,
    "pll_clk_in_max": 133,
    "pll_clk_out_min": 16,
    "pll_clk_out_max": 275,
    "divide_clock": true,
    "divide_pll": false,
    "use_explicit_clock_buffers": false,
    "strict_freq": false,
    "all_freq": false,
    "ice40_reconfiguration": false,
    "ice40_confs": [
        false,
        false,
        false,
        false
    ],
    "sync_control": false,
    "accepted_freq": 0.0
}
```

FIGURE 6. **An example of configuration file contents.**

Algorithm 1 An Algorithm to Calculate PLL Parameters to Generate a Given Frequency

```
best_fout = 0
for divr in range(16):
    f_pfd = device.clk_freq / (divr + 1)
    for divf in range(128):
        f_vco = f_pfd * (divf + 1)
        for divq in range(1, 7):
            fout = f_vco * math.pow(2, -divq)
            if math.fabs(fout -
                    freq_setting.frequency) <
                    math.fabs(best_fout -
                    freq_setting.frequency) or
                    not found_something:
                best_fout = fout
                best_divr = divr
                best_divf = divf
                best_divq = divq
```

identifiers, the designer must connect it with the *clk* port of the PMU module. If the *sync_control* attribute has been activated in the Generator configuration file, the synchronizers for PMU inputs are also synthesized. It is necessary to connect two input clock signals, one for the controlling component (requesting power-mode switching, e.g. CPU) and one for the PMU. Also, two control signals driven by the controlling component have to be connected manually. The power-mode switching request signal must be connected to the *FlagIn_clkA* port, and the switching vector must be connected to the *BusIn* port. Also, in case of other synchronizers (between other communicating components), it is necessary to connect a suitable signal to the *FlagIn_clkA* port. There is an optional *FlagOut_clkB* port that informs about new value at the *BusOut* output, which can also be connected. Another optional output port is *Busy_clkA*, which can be connected to inform the input power domain (generating the signal) that the signal has not been processed yet by the output domain. All modules have a reset signal, preset to the static zero value. It is recommended to replace it so that the signal is driven dynamically. In the prototype, we have activated reset signal upon the start for the period of one clock cycle and the system worked correctly. The last required modification is to connect clock signals of the functional modules to the clock signals generated for their power domains. To assist with this, the commentaries are generated in the code, e.g. "Change clock to pd_clk_0".

All of the specified functional and non-functional requirements have been met, which can be summarized as follows. The developed tool is able to load SystemC/PMS specification and extract the information about power management. It is able to synthesize PMU that applies power-reduction techniques supported by the target FPGA device. The tool is multi-platform and is able to run by various operating systems. The tool is available as an easy to use tool offering both, command line and graphical, user interfaces. Besides, the created source code can be used as a module for other Python projects. The tool supports simple user information about the current state (i.e. progress) and it is also able to generate a detailed SystemC/PMS analysis report. Configurability of the tool is achieved mainly by the JSON-based configuration file.

IV. RESULTS AND DISCUSSION

To evaluate the proposed method, the experiments have consisted of four parts. Firstly, the ability of the developed tool (implementing the proposed method) to analyze a SystemC/PMS model and to extract the power-management information has been verified. The second part has verified the ability of the tool to create a PMU, including the power-reduction techniques available on the selected FPGA device, based on the extracted power-management information. The third part evaluated the simplification of the design process offered by the proposed automation method. And last but not least, it has been verified that the generated code is accepted by the development environment for the selected FPGA and that the energy requirements of the final hardware device (i.e. the FPGA with the running application including automated power management) can be reduced using the proposed method.

Although the pmuToFPGA tool has been developed at the Mac OS X platform, which was also used in the following experiments, the tool is multiplatform thanks to the Python nature. The environment requirements are Python version 3.7 or higher with the following libraries: pyparsing, Mako, and PyQt5. Besides the Mac platform, we have successfully tested the tool using the Windows 10 Home 64-bit operating system running at the machine with the Intel Core i5 processor and 8 GB of RAM.

In the first part of the experiments, we have used various SystemC/PMS specifications described manually. The basic SystemC model described a simple FPGA application with one Intel 8080-compatible processor and the RS232 interface. There were three performance levels specified. Other models described small variations of the basic model. For example, an additional power state was used or there was another power domain with an extra component specified. The intermediate JSON files with extracted power-management information have then been manually checked whether they correspond to the specifications. The result of this part of the experiments is that the analysis and extraction features of the tool worked correctly.

In the second part, six intermediate JSON files specifying various power management have been used to verify the synthesis function of the developed tool. In these files, the differences were, for example, in the specified frequencies or in the number of power domains. These six power-information JSON files were accompanied by another 13 configurations for FPGA, testing various possible parameters. All possible combinations of these files and configurations have been used to synthesize the PMUs and other code required by the FPGA to successfully use the power-reduction techniques (e.g. synchronizers). Simple testbenches were also generated to run simple Verilog simulations to verify the syntactical correctness. The logical correctness of the automatically generated code has been manually checked whether it corresponds to the original PMS specifications. For a single randomly selected combination, the manually created more complex testbench was used to verify all possible states during the simulation. The results confirmed that the automatically synthesized Verilog code is generated correctly.

The main goal of the proposed method was to simplify and speed-up the design process for FPGA-application designers, who need to develop energy-efficient systems. To evaluate these benefits of the method, we have summarized the statistics (concerning the code size) from the previous experiments. Six samples of different power-management specifications were used and the average values have been compared. For the abstract power management, only the SystemC/PMS information about power management is counted. For the synthesized power management, the Verilog code describing the PMU and generated synchronization and isolation nodes are counted, as well as the modifications in the top-level Verilog module. Since various code-style discipline could influence a single-parameter comparison, a number of lines, a number of words, and a number of characters have been compared. The results reported in Table 2 show that the synthesized power management in Verilog is approximately 6.6 times more complex than the abstract power management in SystemC/PMS. It means that the manual effort of designers regarding power management is 6.6 times smaller using SystemC/PMS, and thus the design is faster (since the automated-synthesis time is negligible). Such a result

TABLE 2. **Abstract and synthesized power management comparison.**

Code size	Abstract power management	Synthesized power management	Difference
Characters	993	6555	660%
Words	92	609	662%
Lines	38	260	684%

represents a significant simplification for the power-managed FPGA-applications design process.

To evaluate the proposed method in real hardware, we have selected the Lattice iCEstick Evaluation Kit [33] (the iCE40 device, specifically suitable for low-power designs), mainly due to good support by open-source tools and small purchase costs. A drawback of the selected FPGA is its support of just a single voltage level for the application logic. Due to this fact, we were limited to clock gating and frequency scaling techniques to increase energy efficiency. However, it was sufficient to show the benefits and applicability of the proposed method.

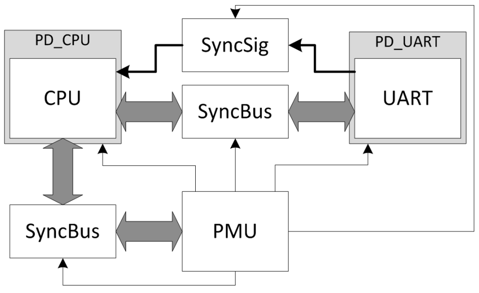

FIGURE 7. **The implemented case-study system overview.**

As a demonstration application to be running on the FPGA, a simple open-source 8-bit microprocessor (CPU) was interconnected with the UART (Universal Asynchronous Receiver-Transmitter) interface. We have created the abstract model of the top module of this system using SystemC/PMS and specified power management for such a system. The system was split into two clock domains, one for CPU (*PD_CPU*) and the other for UART (*PD_UART*). Four power states were specified for the CPU domain: the first one to stop its operation, the second to scale its operating frequency to 1.2 kHz, the third to scale the frequency to 12 MHz, and the last to scale the frequency to 48 MHz. Based on this specification, a new top-module Verilog code along with the PMU have been synthesized automatically. We have made a few small manual modifications (to identify clock signals of the components) and used the freely available synthesis tools of the IceStorm project [34] to program the FPGA device with the designed system. In order to visually see in which power state the CPU is operating, we have created a simple assembly program writing specific text strings via the UART interface. Special keys were preconfigured to switch the power states. An abstract overview of such a setup is illustrated in Fig. 7. The synchronizers are automatically introduced into the design by the proposed method. These are required for signals and buses interconnecting the two power domains (clock domains) and to control the PMU by the CPU.

The result of this experiment was the correctly functioning FPGA application – i.e. the speed of text-strings appearing while reading the UART interface by the laptop was noticeable when the special key was pressed. Although the chosen application was pretty simple, it was enough to illustrate the benefits of the proposed method. The proposed automation method helped to scale the frequency of multiple clock domains, while the automatically generated synchronizers successfully avoided the metastability issues.

In order to ensure that the developed tool and the implemented method are not limited to specifically selected iCE40 FPGA device, we have used the tool to implement the same application on commonly known Xilinx Spartan 3 FPGA [35]. Both the Vivado synthesis tools and the Spartan 3 device accepted the pmuToFPGA-generated RTL model without any problems. Thus, the tool is not suited just to the specifically selected prototype device – since the proposed automation method is general enough, the tool is easily extendible for other target FPGA devices. The device-specific components (such as PLL) must be predefined in the tool to correctly generate the enriched RTL model for a given platform. If not, a general RTL code is synthesized, acceptable by any FPGA device – however, it might not be optimal (e.g. a common counter is used to generate the required clock frequencies instead of dedicated clock-management circuitry, such as PLL).

Using the Kkmoon RD Tester UM24C [36] device, the current consumption in the four specified power states have been measured (see Table 3). Since the current consumption directly relates to the power consumption ($P = I \times V$), the energy requirements of the device can be thus deduced. The calculated power in the HOLD state can be approximated to the static power of the device (no operation). Thus, the dynamic power can be assumed as the remaining amount of the calculated power.

TABLE 3. Current measurements in individual power states.

Power state	Measured current	Computed power (5 V supply voltage)	Dynamic power
HOLD (frequency = 0 Hz)	122 mA	610 mW	0 mW
DIFF_LEVEL(1) (frequency = 1.2 kHz)	124 mA	620 mW	10 mW
DIFF_LEVEL(2) (frequency = 12 MHz)	127 mA	635 mW	25 mW
DIFF_LEVEL(3) (frequency = 48 MHz)	130 mA	650 mW	40 mW

To derive the energy, we would need to measure also the time, which the device has spent in each power state. The dynamic power calculations (provided in the last column of the table) however are enough to show that the applied frequency scaling can indeed reduce power consumption, and thus the energy can be saved (e.g. when frequency scaling is applied to reduce idling of the components). Since the proposed method was not targeted to any new power-reduction technique, but to easier application of existing ones, this evaluation was focused on showing that the automatically applied technique works. This experiment has proved that the automatically synthesized power management using the proposed method can be used to reduce power, and thus it contributes to energy-efficient systems design.

Although there is still a small amount of manual effort needed to finish the integration of the automatically synthesized power-management code into the Verilog model, it is insignificant. In the used case-study system, the pure-manual introduction of power management has taken several hours and subsequent adding/modification of the frequency has taken another hour of work. Using the developed automation tool, it has taken up to an hour of complete time (including the mentioned post-synthesis manual modifications), while the synthesis itself has taken few seconds. We expect that the designer can save hours of design time using the proposed automation method. Even more, if we take into account possible introduction of human errors into the design and subsequent debugging effort.

The whole project implementing the proposed automation method along with the created/modified source code and the used test samples are available on GitHub [32], [37] to increase the reproducibility of our results. To summarize, the benefits and limitations of the proposed method are briefly stated in Table 4.

TABLE 4. Advantages and disadvantages of the proposed automation method.

Advantages	Disadvantages
simplified application of power management	limited to ESL-based design flow
reduced power consumption of FPGA-based systems	limited to SystemC/PMS specification
increased energy efficiency	still some manual effort required
reduced verification and debugging effort	limited support of power-reduction techniques
shorter time-to-market	

It must be noted that the amount of actually reduced power consumption and increased energy efficiency depends on the used techniques and their actual implementation on the FPGA device done by logic synthesis and place & route tools (not targeted in our work). The proposed automation method helps to create an RTL model of the power-managed system based on the abstract specification and it relies on existing tools (provided by the FPGA vendor) to implement the model into the device. Thus, different tools can provide different results.

V. CONCLUSION

This work was focused on the simplification of application of power-reduction techniques in FPGA-based designs using power management. Especially, it was focused on the automated insertion of power-management elements (the control unit as well as the supporting logic) required by the FPGA device based on the abstract specification in SystemC/PMS. The power-intent specification in SystemC/PMS is simple and intuitive, thus usable even by inexperienced designers. The complicated more-detailed power management is synthesized automatically; therefore, the design process is faster and verification effort is minimized (less debugging due to a limited number of human errors). The proposed method is useful especially in low-power application design for FPGA platforms, but also in other FPGA-based designs that require energy efficiency (such as energy-constrained IoT applications). The experimental results confirmed the benefits of the proposed automation method, which simplified the power-management specification approximately 6.6 times and reduced development time by hours of effort.

In the future, the method can be extended by other power-reduction techniques (such as power gating or voltage scaling); however, another FPGA device supporting them would need to be used. There is also the possibility to fully automate power management (outlined in [38]), in order to be completely transparent to the designer. It would enable the designer to fully focus on the system function by automatically manage power to accomplish the specified function with minimal spent energy. It is however complex task, which would require automated splitting of the system into power domains and assignments of appropriate power states and power modes to the domains. These cannot be optimally determined without further (dynamic) information about the system and without knowledge of the target FPGA device. But maybe some computational-intelligence methods could statistically predict close-to-optimal management based on the abstract functional simulation.

ACKNOWLEDGMENT

This is an extended version of the paper presented at the DDECS 2019 conference [1].

REFERENCES

[1] M. Škuta and D. Macko, "Automated integration of dynamic power management into FPGA-based design," in *Proc. IEEE 22nd Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2019, pp. 1–4.

[2] H. Hihara, A. Iwasaki, M. Hashimoto, H. Ochi, Y. Mitsuyama, H. Onodera, H. Kanbara, K. Wakabayashi, T. Sugibayashi, T. Takenaka, H. Hada, M. Tada, M. Miyamura, and T. Sakamoto, "Atomic switch FPGA: Application for IoT sensing systems in space," in *Atomic Switch*, M. Aono, Ed. Cham, Switzerland: Springer, 2020, pp. 33–58.

[3] J. X. Qin, J. Yang, Z. Qu, and Y. X. Wang, "A mission oriented reconfiguration technology for spaceborne FPGA," *J. Phys., Conf. Ser.*, vol. 1195, Apr. 2019, Art. no. 012012, doi: 10.1088/1742-6596/1195/1/012012.

[4] D. Liu, G. Zhou, J. Huang, R. Zhang, L. Shu, X. Zhou, and C. Xin, "Onboard georeferencing using FPGA-based optimized second-order polynomial equation," *Remote Sens.*, vol. 11, no. 2, p. 124, Jan. 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/2/124

[5] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, "Energy-efficient acceleration of big data analytics applications using FPGAs," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 115–123.

[6] D. Weller, F. Oboril, D. Lukarski, J. Becker, and M. Tahoori, "Energy efficient scientific computing on FPGAs using OpenCL," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, New York, NY, USA, 2017, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021730

[7] T. Gomes, S. Pinto, T. Gomes, A. Tavares, and J. Cabral, "Towards an FPGA-based edge device for the Internet of Things," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2015, pp. 1–4.

[8] D. Chen, J. Cong, S. Gurumani, W.-M. Hwu, K. Rupnow, and Z. Zhang, "Platform choices and design demands for IoT platforms: Cost, power, and performance tradeoffs," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 1, no. 1, pp. 70–77, Dec. 2016.

[9] S. Huda and J. H. Anderson, "Power optimization of FPGA interconnect via circuit and CAD techniques," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2016, pp. 123–130.

[10] S. Ishihara, M. Hariyama, and M. Kameyama, "A low-power FPGA based on autonomous fine-grain power gating," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1394–1406, Aug. 2011.

[11] A. Ahari, B. Khaleghi, Z. Ebrahimi, H. Asadi, and M. B. Tahoori, "Towards dark silicon era in FPGAs using complementary hard logic design," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 518–523.

[12] A. A. M. Bsoul, S. J. E. Wilton, K. H. Tsoi, and W. Luk, "An FPGA architecture and CAD flow supporting dynamically controlled power gating," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 178–191, Jan. 2016.

[13] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time power gating in hybrid ARM-FPGA devices," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 512–517.

[14] P. P. Czapski and A. Śluzek, "System-level approaches to power efficiency in FPGA-based designs (data reduction algorithms case study)," *J. Automat. Mobile Robot. Intell. Syst.*, vol. 5, pp. 49–59, Jan. 2011.

[15] G. Patrigeon, P. Benoit, and L. Torres, "FPGA-based platform for fast accurate evaluation of ultra low power SoC," in *Proc. 28th Int. Symp. Power Timing Modeling, Optim. Simulation (PATMOS)*, Jul. 2018, pp. 123–128.

[16] A. Schwandt and M. Winzker, "Modular evaluation system for low-power applications: Educating undergraduate students in advanced digital design," in *Proc. 24th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2017, pp. 364–367.

[17] ITRS 2.0. (2015). *International Technology Roadmap for Semiconductors 2.0.* [Online]. Available: http://www.itrs2.net/itrs-reports.html

[18] D. Bacon, R. Rabbah, and S. Shukla, "FPGA programming for the masses," *Queue*, vol. 11, no. 2, pp. 40–52, Feb. 2013, doi: 10.1145/2436696.2443836.

[19] S. Lahti, P. Sjovall, J. Vanne, and T. D. Hamalainen, "Are we there yet? A study on the state of high-level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, May 2019.

[20] S. Windh, X. Ma, R. J. Halstead, P. Budhkar, Z. Luna, O. Hussaini, and W. A. Najjar, "High-level language tools for reconfigurable computing," *Proc. IEEE*, vol. 103, no. 3, pp. 390–408, Mar. 2015.

[21] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis, and K. Olukotun, "Spatial: A language and compiler for application accelerators," in *Proc. 39th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 296–311, doi: 10.1145/3192366.3192379.

[22] B. Pauget, D. J. Pearce, and A. Potanin, "Towards compilation of an imperative language for FPGAs," in *Proc. 10th ACM SIGPLAN Int. Workshop Virtual Mach. Intermediate Lang. (VMIL)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 47–56, doi: 10.1145/3281287.3281291.

[23] A. Qamar, F. B. Muslim, J. Iqbal, and L. Lavagno, "LP-HLS: Automatic power-intent generation for high-level synthesis based hardware implementation flow," *Microprocessors Microsyst.*, vol. 50, pp. 26–38, May 2017.

[24] K. Gagarski, M. Petrov, M. Moiseev, and I. Klotchkov, "Power specification, simulation and verification of SystemC designs," in *Proc. IEEE East-West Design Test Symp. (EWDTS)*, Oct. 2016, pp. 1–4.

[25] D. Lemma, M. Goli, D. Grose, and R. Drechsler, "Towards generation of a programmable power management unit at the electronic system level," in *Proc. 23rd Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2020, pp. 1–6.

[26] D. Macko, K. Jelemenská, and P. Čičák, "Simplifying low-power SoC top-down design using the system-level abstraction and the increased automation," *Integration*, vol. 63, pp. 101–114, Sep. 2018.

[27] D. Macko, "Adoption of abstract power-management specification to FPGA-based design," in *Proc. Int. Conf. Emerg. eLearn. Technol. Appl. (ICETA)*, Nov. 2016, pp. 199–204.

[28] IEEE. *IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems*, IEEE Standard 1801-2018, 2019.
[29] IEEE. *IEEE Standard for Standard SystemC Language Reference Manual*, IEEE Standard 1666-2011, 2012.
[30] D. Macko, K. Jelemenská, and P. Čičák, "Verification of power-management specification at early stages of power-constrained systems design," *J. Circuits, Syst. Comput.*, vol. 26, no. 8, Aug. 2017, Art. no. 1740002.
[31] M. Bayer. *Mako Templates for Python*. Accessed: Dec. 4, 2018. [Online]. Available: https://www.makotemplates.org/
[32] M. Škuta. (2019). *pmuToFPGA*. [Online]. Available: https://github.com/mintos5/pmuToFPGA
[33] Lattice Semiconductor. (2019). *iCEstick Evaluation Kit: Rapid Development for Affordable Innovation*. [Online]. Available: https://www.latticesemi.com/icestick
[34] C. Wolf and M. Lasser. *Project IceStorm*. Accessed: May 20, 2019. [Online]. Available: http://www.clifford.at/icestorm/
[35] Xilinx. (2019). *Spartan-3 FPGA Family*. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html
[36] KKmoon. (2016). *KKmoon UM24C USB 2.0 Color LCD Display Tester Voltage Current Meter Voltmeter Ammeter Battery Charge Cable Impedance Measurement Communication Version*. [Online]. Available: https://www.kkmoon.com/p-e3358-2.html
[37] M. Škuta. (2019). *iCE40HX1K Demos*. [Online]. Available: https://github.com/mintos5/iCE40HX1K-demos
[38] D. Macko, "Contribution to automated generating of system power-management specification," in *Proc. IEEE 21st Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2018, pp. 27–32.

MICHAL ŠKUTA received the bachelor's and master's degrees in computer engineering from the Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, in 2017 and 2019, respectively.

In his work, he focused on low power IoT devices, specifically the development of an access point for the LoRa technology and automation of power management on FPGA platforms.

DOMINIK MACKO (Member, IEEE) received the master's degree in computer engineering and the Ph.D. degree in applied informatics from the Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, in 2011 and 2015, respectively.

He is currently with the Institute of Computer Engineering and Applied Informatics of his Alma Mater. His research interests are in the area of low-power digital-systems design and energy-efficient communications within the Internet of Things.

KATARÍNA JELEMENSKÁ (Member, IEEE) received the Ph.D. degree in computer science from the Slovak University of Technology in Bratislava, in 1995.

She is currently the Director of the Faculty of Informatics and Information Technologies, Institute of Computer Engineering and Applied Informatics, Slovak University of Technology in Bratislava. She has been working with the Slovak University of Technology in Bratislava, since 1986. Her research interests include digital systems design, modeling, and verification, means of hardware specification, as well as efficient use of information and communication technologies in education.