Power Electronics Control with System-on-a-Chip-Based Platforms

Marco Guerreiro, Shrikant Kharade, Pedro dos Santos and Steven Liu
Chair of Control Systems, Technische Universität Kaiserslautern, Kaiserslautern, Germany
Corresponding author: guerreiro@eit.uni-kl.de

***Abstract*—An emerging trend in power electronics control is the use of controllers based on a system-on-a-chip (SoC). SoCs provide a high-performance computing platform by embedding one or multiple processors and a field-programmable gate array (FPGA) on the same chip. This paper reviews SoCs for power electronics control applications, discussing aspects such as task allocations and available hardware and software tools. A design and experimental results of an SoC-based controller are discussed, and some of the SoC-based platforms built by researchers in power electronics control are reviewed.**

***Index Terms*—power electronics, embedded control, system-on-a-chip, field-programmable gate array**

I. Introduction

Today and for the coming decades, the power grid is seeing a shift in its operational paradigm. The next generation power grid will allow bidirectional flow of information and power, distributed generation, and high-voltage DC (HVDC) transmission. A key enabler of the next generation power grid are power electronic converters. These embed many features that are important for the new power system [1]. However, operation and control of these converters can be challenging, due to the complex structure of modern power electronic systems and the required behavior for grid-connected operation. One example is the modular multilevel converter (MMC), a topology that has been gaining increasingly attention due to its application in HVDC. For proper operation, the MMC requires several control loops, such as submodule voltage balancing, circulating current suppression, output power control and energy control [2]. Moreover, power electronic converters require fast-acting protection mechanisms to ensure that the converter operates within safe levels.

For the next generation power system, power converters need to quickly react to changing set-points and system disturbances. Traditionally, cascaded PI-based control loops are the main solution for converter control. These controllers are simple to realize; however, the cascaded structure limits their performance. Today, advanced multi-objective control techniques are an appealing alternative to cascaded structures [3], [4]. For instance, optimization-based methods, such as model predictive control (MPC), not only allow multiple-input, multiple-output (MIMO) systems to be controlled with a single loop, but also allow for system constraints to be taken into account during the controller design. The main disadvantages of these advanced control techniques lie in their much higher computational complexity and the need for more system measurements.

Currently, digital-signal processors (DSPs) are paired with field-programmable gate arrays (FPGAs) to deal with the complexity imposed by the converter control objectives [5]. The parallel nature of FPGAs allows simultaneous coordination of signal acquisition, protection and actuation systems, while DSPs execute complex and conditional control tasks in the sub-millisecond range. The bottleneck of this approach lies in the coordination between the two parts, which is usually limited by the DSP.

To cope with the increasing control complexity and timing requirements demanded for converter control, particularly converters in grid-connected operation, researchers are moving towards devices that embed a high-performance multi-core processor and an FPGA on the same chip, denoted as a system-on-a-chip (SoC) [6]–[8]. This silicon-level integration improves on the bottleneck of processor/FPGA communication, as well as leading to a controller that is smaller, faster, more reliable, easier to maintain, and cheaper.

This papers reviews SoCs for embedded control applications, with a focus on converter control problems. The rest of this paper is organized as follows. Section II discusses how SoCs can be used for control applications. Section III discusses the main aspects to consider when building an SoC controller and shows a practical example. Section IV presents some of the SoC-based platforms developed by power electronics control researchers, and Section V concludes this paper.

II. Power Electronics Control with SoCs

Fig. 1 shows an example of a cascaded control strategy for a power converter. The inner loop is usually responsible for current control, while the outer loop controls power or voltage. The actual implementation of the control strategy can be realized through an embedded system, which receives set-point signals and interacts with the power converter through actuators and sensors. The actual implementation of the embedded controller is shown by the block diagram of Fig. 2. The controller interfaces with the power converter mainly through a low-level actuation system (e.g. gate drivers) and a signal acquisition system (e.g. voltage and current sensors).

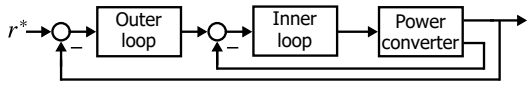

Fig. 1: Cascaded structure for power converter control.

 DOI 10.1109/PEMC51159.2022.9962949

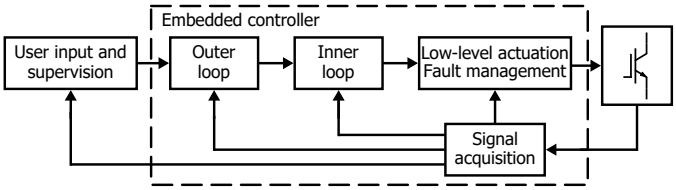

Fig. 2: Power converter control with an embedded controller.

Moreover, the controller must include a fault management system that keeps the converter operating within safe levels.

The diagram shown in Fig. 2 introduces some of the tasks that a embedded controller must perform: provide a user interface, drive the actuation system, acquire sensor signals, monitor the converter and run the control algorithm. Some of these tasks are better implemented as hardware or software. Hardware allows parallel processing and tasks that are better implemented in this way are the low-level actuation signals, fault management and signal acquisition. These have a fixed structure and are time-critical. Other tasks showing more complex and dynamic structure, such as a user interface, are better implemented as software, since they show a more complex and dynamic structure. Certain tasks can be implemented in software or hardware, or may even benefit from a mixture of both. Control algorithms with a cascaded structure are one such example. The inner loop, which is usually simpler, can be implemented directly by a digital circuit, while a complex and conditional algorithm is better implemented as software.

SoCs provide a promising architecture for the the hybrid software/hardware nature of embedded controllers. SoCs are devices that embed a processing system (PS) and programmable logic (PL) on the same chip [9]. The PS runs software tasks, while the PL is a programmable hardware that allows building custom digital circuits. Fig. 3 shows a generic representation of an SoC, as well as task allocation for a power converter control application. The PS can be a single processor with one or multiple cores, or multiple processors with multiple cores. Moreover, the PS provides its own set of peripherals, memory interfaces and input/output (I/O) ports. The PL is equivalent to an FPGA chip in terms of architecture and programmable resources, and also provides its own set of blocks, such as multiply-and-accumulate (MAC) units, dedicated memory and dedicated I/O ports. The PS and PL communicate through high-speed buses.

A. Task allocation on SoCs

Although there are many configurations for the PS, this paper focuses on SoCs where the PS is a single dual-core processor, a topology more commonly found in SoC-based platforms. There is no standard design methodology to allocate all of the controller's tasks on an SoC [6]–[8]. This process is highly dependent on the application and on the system. Fig. 3b shows one possible way to distribute the tasks on an SoC with a single dual-core processor, PL and shared memory. The dual-core processor is represented by the CPU 1 and CPU 2 blocks, and the PL is represented by the FPGA block. In the structure shown in Fig. 3b, one core is dedicated to providing a user interface, the second core is dedicated to the main control task and the PL interfaces with the power converter.

In SoC-based controllers, it is common and advantageous to dedicate one core for the user interface [6]–[8]. The interface task can be executed without disrupting the other core and the PL, and an operating system (OS) can be used. OSs support many common interfaces (HDMI, USB, Ethernet) and high-level tools, which simplifies development of the user interface.

There are many ways to achieve the user interface. One possibility is to have a monitor and input peripherals (e.g., mouse and keyboard) connected directly to the controller. Alternatively, it is possible to establish a communication channel between a host computer and the controller, where commands are sent through a command-line interface (CLI) or a graphical user interface (GUI). The core dedicated to the interface task also accesses regions of the shared memory that may be shared by the other core and the PL.

The main control task is executed by another core, represented by CPU 2 in Fig. 3b. This core also executes additional tasks, such as interfacing with the first core and with the PL. Usually, the control task is executed periodically and must be completed inside one sampling period, so that the control signals are properly updated. The other tasks may be periodic or triggered by events. Due to strict timing requirements, it is common to avoid an OS to schedule the execution of tasks on this core. Instead, the tasks are implemented following an interrupt-based mechanism (also known as bare-metal implementation). In this case, the interrupts are tied to signals such as timers, completion of analog-to-digital conversions and reception or transmission of data. These interrupts are assigned different priority levels, such that the main control task can be executed without disruption.

Interfacing with the power electronic system is a task of the PL. This interface consists of low-level control of actuators and analog-to-digital converters (ADCs). Using the PL for this interface offers many advantages. Due to its parallel nature, the PL is able to perform measurements simultaneously, which is specially important in systems containing many sensors. This simultaneous acquisition also allows quick detection of faults or violation of operating conditions. The PL can likewise control many actuators simultaneously and independently. It is also common to run simpler control algorithms on the PL. In cases where the overall controller presents a cascaded structure, inner and simpler loops can be implemented by the PL. Additional tasks of the PL include interfacing with the core executing the main control task and accessing shared memory. Such access allows the PL to directly log measurements and events, such as faults.

B. Variations on task allocation

The structure shown in Fig. 3b is only one possible way to distribute tasks on an SoC. Other possibilities are to execute the control task entirely on the PL, and use the cores only for interfacing and system monitoring. The advantage of this approach is that no synchronization between the PL and the

 DOI 10.1109/PEMC51159.2022.9962949

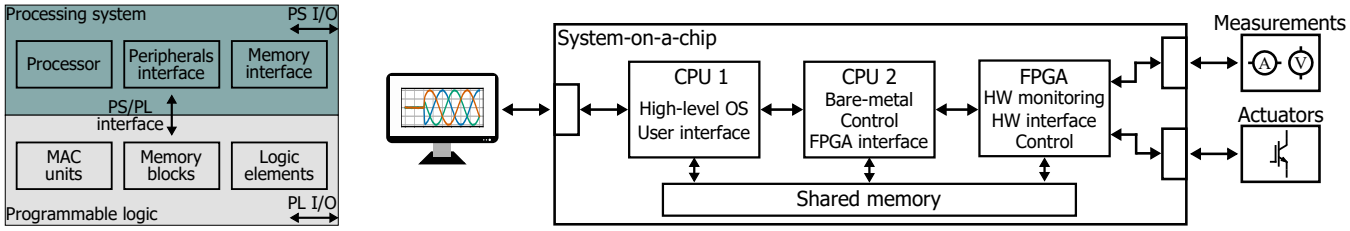

Fig. 3: System-on-a-chip (SoC) for control applications. (a) Generic structure of an SoC consisting of a processing system (PS) and programmable logic (PL). (b) Task allocation on an SoC with a dual-core processor and PL.

PS is required at every control period. However, the control algorithm must be realizable with the limited PL resources, which are also used for data acquisition and actuation.

The true potential of SoCs, however, lies in its flexibility. Co-design allows software and hardware tasks to complement one another and each control application can have a tailored implementation. One example is mixing execution of the control algorithm on the PS and on the PL. For instance, computationally intensive operations that are parallel in nature can be offloaded to the PL, while the PS executes more complex and conditional operations [9]. Moreover, the control algorithm can be split between the cores of the SoC.

In general, suitable operations to be executed on the PL are data filtering, data pre-processing, phase-locked loops, and common linear transformations (e.g. Clarke and Park transformations) [8]. These operations precede the main control algorithm, and can be executed in fixed-point format with samples directly from the ADC. Filtering can be performed independently and simultaneously for each ADC, and three-phase transforms for current and voltage quantities can also be accomplished in parallel. Additional operations suited for PL implementation are matrix-matrix or matrix-vector operations.

On the PS, suitable operations are those requiring floating-point arithmetic, due to the high dynamic range of the involved variables. Iterative algorithms, where the next iteration can only start when the current iteration has finished, may benefit if executed on the PS. Since the PS clock is usually much higher than the PL clock, execution of these iterations can be faster on the PS. Moreover, the PS may be the only possibility to execute complex algorithms, since the code can be stored in random-access memory (RAM), which is usually large for SoCs (>512 MB), while the PL has a limited amount of programmable resources.

Despite the immense potential provided by the flexibility of SoCs, its actual realization faces various technical challenges. Splitting the control algorithm among several cores and the PL requires a strict coordination effort. Instead of synchronization once at every control period, the cores and the PL may need to synchronize multiple times inside this period, and the design effort of such system increases considerably. Moreover, a detailed analysis of the control algorithm must be carried out, in order to identify how the algorithm should be split, and which parts would run better on the PL or on the PS.

III. SoC-Based Controller Design

This section is divided into two parts. The first part describes the main aspects to consider when building an SoC-based controller and shows some of the available hardware and software tools. The second part shows an example of designing an SoC controller for a line-side converter, discussing task allocation, memory, PL resource usage, and timing measurements.

A. Building an SoC-based controller

1) Hardware: The quickest way to start developing SoC-based controllers is with off-the-shelf boards. Table I summarizes some of the available boards that are potentially interesting for power electronics control. The prices are listed as of June, 2022, obtained from each manufacturer's website. Most of the SoC boards provide at least RAM, flash memory, clock oscillators and a complete power supply solution.

For power electronics control applications, a few configurations and peripherals are of main interest. The PS is responsible for the user interface and control tasks; thus, it is important to choose an SoC with a high-performance PS. RAM is also important, since it holds the code and stack for the CPUs, and may additionally record measurements and internal signals. It is also important to choose a board with the relevant connectivity peripherals. These are mainly dictated by the user interface and can range from just Ethernet to many peripherals such as HDMI and several USBs.

One of the most critical aspects when choosing an SoC board is the number of available I/Os [8]. Power converter control usually requires several measurements and actuator signals. When independent ADCs are used for each measurement, each ADC may require up to four I/Os. In converters where three-phase voltage and current measurements on the converter and on the grid-side are required, the number of I/Os needed can be quite high. Moreover, in addition to the switch signals required to control the converter, there are other important inputs and outputs, such as precharge switches, temperature sensors and diagnostic signals.

2) Software: Converting the controller's tasks to code for the PS and the PL can be achieved mainly by using code generators that are based on abstract models, by hand-coding or by a mixture of both approaches. Abstract models are usually found in control design software tools. In this case, control blocks are exported to different coding languages that are executed on the

TABLE I: List of selected low-cost SoC boards ($\leq$ 500 USD).

Board	SoC	PS	MAC/DSP slices	RAM	Connectivity	I/Os	Price (USD)
PicoZed	Xilinx XC7Z020 (Z030)	Dual-core ARM A9, 667 MHz	220 (400)	1 GB DDR3	Gigabit Ethernet PHY, USB 2.0 PHY	138	255.0 (450.00)
MicroZed	Xilinx XC7Z020	Dual-core ARM A9, 667 MHz	220	1 GB DDR3	Gigabit Ethernet, USB 2.0	123	255.0
Pynq-Z2	Xilinx XC7Z020	Dual-core ARM A9, 667 MHz	220	512 MB DDR3	Gigabit Ethernet, USB 2.0, HDMI	70	157.73
Z-turn board	Xilinx XC7Z020	Dual-core ARM A9, 766 MHz	220	1 GB DDR3	Gigabit Ethernet, USB 2.0, HDMI	106	179.00
Kria K26 SOM	Xilinx XCZU5EV	Quad-core ARM A53, 1.5 GHz, Dual-core ARM R5, 600 MHz	1248	4 GB DDR4	Gigabit Ethernet, USB 3.0, HDMI	240	392.73
SmartBerry	Microsemi SmartFusion2	Single-core ARM M3, 166 MHz	22	256 MB DDR3	Gigabit Ethernet	60	135.62
DE10-Nano	Intel Cyclone 5CSEB6	Dual-core ARM A9, 800 MHz	112	1 GB DDR3	Gigabit Ethernet, USB 2.0, HDMI	52	180.00

PS or on the PL. These tools are intended to convert control blocks to coding languages without a specific target hardware. Usually, SoC vendors have their own add-ons that integrate with control design software, and provide target-oriented code generation for their specific hardware.

For the PL, there is an additional abstract code generation method, denoted as high-level synthesis (HLS) [9], [10]. HLS allows the user to synthesize hardware description language (HDL) code from C/C++ code. Thus, tasks and algorithms initially developed in C can be executed on the PL without the need to code it to HDL. HLS provides several optimization features such as loop unrolling, pipelining and array partitioning, which allow the user to exploit the parallelization capabilities of the PL.

The PL and the PS can also be hand-coded. For the PL, common languages are VHDL, Verilog and SystemVerilog. These are vendor-independent. Hand-coding the PS is usually done with C/C++ and, in some special instances, using Assembly.

A hybrid approach combines code generation and hand-coding. In this case, a framework is first built, where specific tasks and blocks are hand-coded for efficiency or for custom features. Examples are pulse-width modulation (PWM) blocks, ADC blocks, and the user interface. Then, the controller or specific parts of the controller are automatically generated and integrated to the built framework. Table II outlines some of the main design tools provided by SoC vendors. The table shows tools for PS and PL coding, HLS tools and Matlab add-ons.

B. Design example

This section shows a design example of an SoC-based controller for control of a line-side converter. This type of converter is used in railway applications, where it interfaces the AC railway traction supply system to the DC traction drive system of locomotives [11]. The diagram of a scaled-down laboratory prototype of the converter is shown in Fig. 4. The system consists of an input AC source, filter inductors (L),

TABLE II: SoC design tools.

Vendor	PS/PL IDE	HLS	Matlab add-on
Xilinx	Vitis/Vivado	Vitis HLS	Model Composer
Intel	SoC EDS	HLS compiler	DSP builder
Microsemi	Libero SoC DS	SmartHLS	-

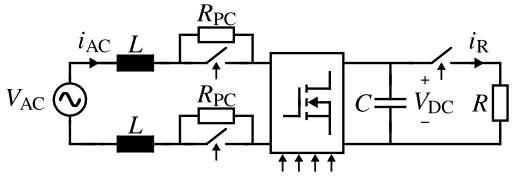

Fig. 4: Single-phase active rectifier.

precharge resistors (R_{PC}), AC-side switches, a full bridge, an output filter capacitor (C), a DC-side switch and a load (R). Although IGBTs are more commonly used for the full bridge, MOSFETs were used for the scaled-down prototype. The control objective is to regulate the output voltage while presenting a sinusoidal current waveform for the AC source. Four measurements are taken: the grid and load voltages (V_{AC} and V_{DC}) and currents (i_{AC} and i_R).

To prevent peak currents at startup, a precharge scheme is used to allow the output capacitor to charge at a predetermined current. The controller used in this example is based on a cascaded PI and proportional resonant (PR) loop, as described in [12]. The controller was discretized using the Tustin transform. For signal acquisition, SPI-based ADCs were used. Each ADC requires 3 pins; resulting in 12 pins to interface with all 4 ADCs. To actuate on the converter, 7 pins are required: 4 pins for the full-bridge MOSFETs and 3 pins for the input and output switches. Additional output pins were used for diagnostic signals. The switching frequency was set to 5 kHz.

For this example, the Pynq-Z2 board is used as the SoC platform. This board has a Xilinx Zynq-7020 SoC, containing a dual-core ARM A9 processor and an Artix-7-equivalent FPGA. Moreover, the board contains an embedded programmer, Ethernet and USB connectors, 512 MB of RAM and prototype-friendly pin headers. The tasks of the controller are allocated based on the scheme shown in Fig. 3b.

The actual implementation of the SoC controller is illustrated by the diagram shown in Fig. 5. The most critical aspect for implementing the controller's tasks on the SoC is the coordination of both CPUs and the PL. Here, this coordination is achieved by two mechanisms: memory sharing and interrupt requests. The basic principle is that the sender first writes the data to be sent on a shared memory location, and uses an interrupt signal to notify the receiver that new data is available.

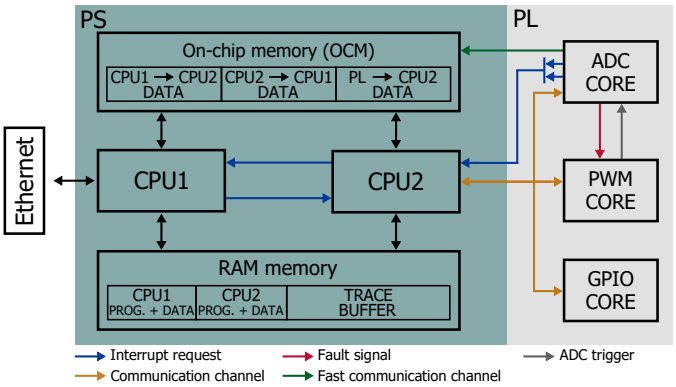

Fig. 5: SoC layout used in this example.

In the SoC controller designed in this example, there are three sender-receiver scenarios: CPU1 to CPU2, CPU2 to CPU1 and PL to CPU2. These scenarios are tied to three different interrupt signals, according to the data transfer direction. These signals are indicated by the blue arrows in Fig. 5. For each interrupt signal there is an associated memory location, that holds the data being transferred. Here, the on-chip memory (OCM) of the processor was used for this purpose, to allow a low processor access latency [9]. The SoC used has 256 kB of OCM, and three 16 kB sections were reserved for data exchange (the rest is left unused). However, in practice, the most data intensive operation is sending the ADC readings to CPU2, which uses only 16 bytes.

The RAM of the system holds the code and stack for each CPU and a trace buffer. Thus, the RAM was segmented in three sections: 7 MB are reserved for CPU1, 3 MB are reserved for CPU2 and 128 MB are reserved for the trace buffer. The rest is left unused. Next, the user interface and the tasks allocated for each processor and the PL are discussed, as well as their implementation results.

1) User interface: In this example, the user interacts with the controller through a simple CLI, by sending commands such as "enable control", "set controller parameters" and "read logged data". To provide flexibility and high data rates, TCP/IP sockets are used to exchange data between a host (the user) and the controller. In this case, the controller acts as a server and the user is a client. On the host side, Python is used to handle the client socket, while on the controller side, the lightweight IP (lwIP) framework is used to manage the server socket.

2) CPU1: this CPU is responsible for providing the user interface, by managing the server socket and decoding commands received from the user. When necessary, CPU1 communicates with CPU2 and accesses the trace buffer. For these tasks, FreeRTOS and the lwIP framework were used, resulting in small code footprint and stack size.

3) CPU2: this CPU is responsible for execution of the main control algorithm, interfacing with the PL, interfacing with CPU1 and saving selected signals to the trace buffer. This CPU works on an interrupt-based mechanism without an OS. As shown in Fig. 5, two interrupt signals are connected to CPU2: one from CPU1 and one from the PL. The first is used to process commands received from CPU1, while the second interrupt actually contains two independent interrupt signals. One signal indicates the availability of new ADC samples, while the other signal indicates a fault detection. The signal related to ADC conversions is periodic and is used to trigger execution of the control algorithm.

The main control algorithm is governed by a state machine that checks the precharge condition and controls the input and output switches. When the precharge is complete, these switches are activated and the main control process is enabled, which runs the actual control algorithm for the converter.

CPU2 has also a communication channel with the PL, as indicated by the orange arrow in Fig. 5. This channel is used to configure the PL by setting the sampling frequency, the ADC clock frequency and the state of outputs. This is implemented as an AXI4-lite interface, provided by Xilinx [9]. It is the simplest and least resource demanding interface, but it is also the slowest one. However, the speed was sufficient in this case. After completion of the control algorithm, selected signals are stored in the trace buffer.

4) PL: The PL consists of three main blocks, or cores: a GPIO core, a PWM core and an ADC core. The GPIO core is mainly used to drive output pins. This core is controlled by the PS and is used for the precharge signals and additional signals for system diagnostics.

The PWM core generates the gate signals for the converter based on a modulation index set by CPU2. This core produces an additional signal that is used to synchronize the PWM signals with the ADC measurements, as shown by the gray arrow in Fig. 5. Moreover, this block has an additional input that allows the PWM output signals to be disabled. This is shown by the red arrow in Fig. 5 and is used to disable the gate signals in case of a fault.

The ADC core controls the external ADCs through a SPI interface and writes the measurements to the OCM. Currently, up to 8 ADCs are supported. The ADC core is triggered by the PWM core; at each trigger, it reads all external ADCs simultaneously, compares the measurements to minimum and maximum values, writes the measurements to the OCM and generate interrupt signals. If any measurement exceeds its bounds, the ADC core generates a fault signal that is used to disable the PWM core. Data is written to the OCM through a fast communication channel, indicated by the green arrow in Fig. 5. This allows a lower latency when transferring data from the PL to CPU2. The fast communication channel is implemented by the AXI4-full interface provided by Xilinx [9].

5) Experimental results: CPU1 and CPU2 were hand-coded and integrated with FreeRTOS, the lwIP framework and drivers from Xilinx. The total code and stack size was approximately 5.5 MB for CPU1 and under 100 kB for CPU2. The main process executed by CPU2 consists of checking the precharge condition, execution of the control algorithm, updating the modulation index on the PL and saving 12 floating-point values to the trace buffer. This entire process executes in approximately 2.45 µs; execution of the discrete PR and PI controllers alone take approximately 0.45 µs.

 DOI 10.1109/PEMC51159.2022.9962949

The ADCs used here have a 12 bit resolution and require 16 clock cycles to transmit a sample. The ADCs were clocked at 1.25 MHz; reading all samples required 13.20 µs. Transferring eight 16-bit values from the PL to the OCM took approximately 0.56 µs. The time between the data being written to the OCM and CPU2 being notified was approximately 0.34 µs.

The ADC and PWM cores were hand-coded and integrated with the AXI interface provided by Xilinx. For the GPIO core, a generic block from Xilinx was used. The Vivado software reports the utilization of the programmable resources of the PL. In this example, the utilization was: 4.2% of look-up tables (LUTs), 0.6% of LUTRAM, 2.7% of flip-flops (FFs) and 24.0% of the available IOs (125 in total).

Figs. 6 and 7 show steady-state and transient waveforms stored on the controller's RAM. All waveforms and additional signals were captured in a 40-second time span. In total, capturing 12 floating-point signals for 40 seconds at 5 kHz required approximately 10 MB of the 128 MB reserved for the trace buffer. Fig. 6 shows the AC-side voltage and current waveforms before and after the controller is enabled. Before the controller is enabled, the gate signals are off and the converter shows the behavior of a passive rectifier. After the controller is enabled, the MOSFETs are controlled and the AC input sees a sinusoidal waveform. Fig. 7 shows the AC- and DC-side transient when the controller is enabled. Before the controller is enabled, the DC-side voltage is given by the passive rectifier. After the controller is enabled, the voltage is regulated to its reference value (20 V), and the AC-side current becomes sinusoidal.

IV. SoC Platforms in Power Electronics Control

Many research groups are moving towards the use of SoCs to build custom embedded controllers for power electronics control. This section discusses some of the platforms that were reported in the literature.

A. UltraZohm

UltraZohm is a platform developed jointly by researchers from TH Nürnberg, TU München and Universidad de Santiago de Chile [6]. The platform is based on a modular hardware design, consisting of a processing board, digital and analog adapter boards, and a carrier board. The processing board is based on a module by Trenz Electronics, hosting a Xilinx multiprocessor SoC. The SoC has one quad-core ARM A53, a dual-core ARM R5 and a large FPGA. The digital and analog boards provide drivers and ADCs, while the carrier board is used to connect all boards together and to provide power supplies and communication channels. UltraZohm is an open source project and all project files are available online.

Task allocation on the UltraZohm is similar to the one showed in Fig. 3b. The user interface is provided by a host computer, which runs a Java-based GUI that communicates with the hardware through TCP/IP. One core of the quad-core ARM A53 is dedicated to managing the TCP/IP connection, and uses FreeRTOS as the OS. This core also exchanges data with the PL, by setting and reading set-points. The other three cores are not used. Slow control algorithms, initialization of the PL and safety functions are executed by one core of the ARM R5 processor, and the other is left unused. The PL is responsible for signal acquisition, actuation, fault detection, data logging and control algorithms.

Using the UltraZohm, Baltruweit et al. (2021) applied finite control set model predictive control (FCS-MPC) to a three-level inverter driving an induction machine [13]. They used a branch-and-bound method to solve the related integer programming problem, and were able to attain a prediction horizon of 12, which is executed within 25 µs.

B. uCube

Researchers from the Power Electronics, Machine and Control group of the University of Nottingham have developed

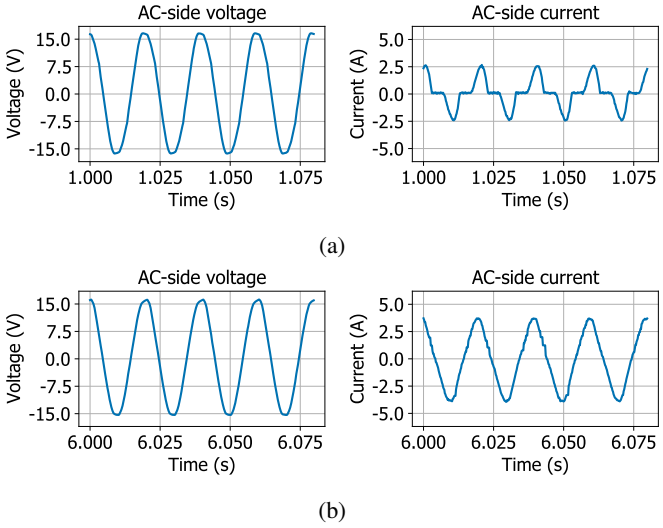

Fig. 6: Steady-state AC-side voltage and current (a) before and (b) after the controller is enabled.

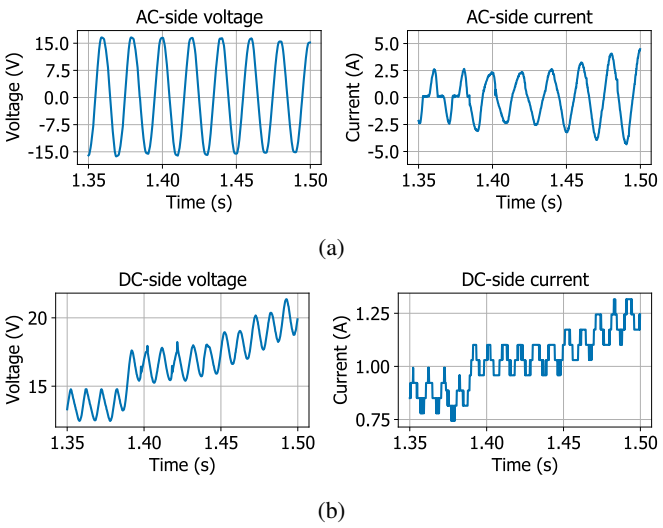

Fig. 7: Transient of (a) the AC- and (b) DC-side when the controller is enabled.

an SoC-based platform denoted as uCube [7]. The platform is based on a board from Avnet, called MicroZed. This board has a Xilinx SoC consisting of a dual-core ARM A9 processor and an Artix-7 FPGA. Three additional boards were designed by the researchers: a digital I/O board with isolated low-level communication interfaces, an analog board and an encoder board. All these boards are stacked on top of each other.

Task allocation on the uCube is also similar to the one showed in Fig. 3b. On a host computer, a Matlab-based GUI allows the user to retrieve logged data and change controller parameters. Communication between the host computer and uCube uses the Modbus protocol, which is carried out either by TCP/IP or an asynchronous serial transmission media. On the controller side, the core dedicated to the interface runs PetaLinux, a Linux-based OS developed by Xilinx for its SoC modules. The second ARM A9 core is used for the control application, to log data and to initialize the PL. This core works on an interrupt-based mechanism that is triggered by new measurements and additional asynchronous events. The PL is used for signal acquisition, PWM generation, fault detection, system protection and control algorithms.

Rovere et al. (2019) used uCube to implement a deadbeat-based controller for a permanent-magnet synchronous motor [14]. The algorithm, which includes a disturbance observer, ran in less than 700 ns.

C. ETI-SoC system

The ETI-SoC system is an embedded controller developed by researchers of the Institute of Electrical Engineering, at the Karlsruhe Institute of Technology [8]. Their platform is based on the PicoZed board, which has a Xilinx SoC with a dual-core ARM A9 processor and a Kintex-7 FPGA. The researchers developed a carrier board for the PicoZed, containing ADCs, communication channels and digital I/Os.

The controller tasks are allocated according to the scheme shown in Fig. 3b. A host computer is connected to the platform through Ethernet and runs a Labview-based GUI. The core servicing the interface runs FreeRTOS, accesses shared memory and interacts with the second core. The second core follows a bare metal application and works through an interrupt-based mechanism. This core has access to shared memory, interfaces directly with the PL and executes the main control algorithm. The PL is responsible for signal acquisition and actuation, and a few processing tasks, such as three-phase-related transforms.

The ETI-SoC system has been used control of a power-hardware-in-the-loop platform [15] and control of active filters for grid-connected converter applications [16].

V. Conclusions

The control of modern power electronic systems faces many challenges, due to the requirement of sub-millisecond execution of complex control strategies. This paper showed how SoCs are a viable solution to these challenges. By providing silicon-level integration of multi-core processors and an FPGA, SoCs enable the development of high-performance control units capable of executing complex control algorithms within small periods of time. Moreover, cheaper and more accessible hardware tools, as well as more mature software tools, have enabled researchers to build their own SoC-based controllers. The main advantage of this approach is the greater flexibility and the possibility to tailor the controller to specific power electronics control problems.

References

[1] B. K. Bose, "Power electronics, smart grid, and renewable energy systems," *Proc. IEEE*, vol. 105, no. 11, pp. 2011–2018, 2017.

[2] S. Debnath, J. Qin, B. Bahrani, M. Saeedifard, and P. Barbosa, "Operation, control, and applications of the modular multilevel converter: A review," *IEEE Trans. Power Electron.*, vol. 30, no. 1, pp. 37–53, 2015.

[3] J. Rodriguez, M. P. Kazmierkowski, J. R. Espinoza, P. Zanchetta, H. Abu-Rub, H. A. Young, and C. A. Rojas, "State of the art of finite control set model predictive control in power electronics," *IEEE Trans. Ind. Informat.*, vol. 9, no. 2, pp. 1003–1016, 2013.

[4] S. Kouro, M. A. Perez, J. Rodriguez, A. M. Llor, and H. A. Young, "Model predictive control: MPC's role in the evolution of power electronics," *IEEE Ind. Electron. Mag.*, vol. 9, no. 4, pp. 8–21, 2015.

[5] J. Ismail, Y. Wan, H. K. Iqbal, P. L. dos Santos, and S. Liu, "Experimental evaluation of a high power medium voltage converter for a DC grid connected agricultural machine," in *PCIM Europe digital days; Int. Exhibition and Conf. for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*, 2020, pp. 1–7.

[6] S. Wendel, A. Geiger, E. Liegmann, D. Arancibia, E. Durán, T. Kreppel, F. Rojas, F. Popp-Nowak, M. Diaz, A. Dietz, R. Kennel, and B. Wagner, "UltraZohm - a powerful real-time computation platform for MPC and multi-level inverters," in *IEEE Int. Symp. on Predictive Control of Electrical Drives and Power Electron. (PRECEDE)*, 2019, pp. 1–6.

[7] A. Galassini, G. Lo Calzo, A. Formentini, C. Gerada, P. Zanchetta, and A. Costabeber, "uCube: Control platform for power electronics," in *IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD)*, 2017, pp. 216–221.

[8] R. Schwendemann, S. Decker, M. Hiller, and M. Braun, "A modular converter- and signal-processing-platform for academic research in the field of power electronics," in *Int. Power Electronics Conf. (IPEC-Niigata ECCE Asia)*, 2018, pp. 3074–3080.

[9] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book*. Strathclyde Academic Media, 2014.

[10] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "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, 2019.

[11] J. Xiang, J. Xu, H. Wang, C. Li, G. Cui, and Y. Peng, "Reconfigurable line-side converter for DC voltage matching and ripple suppression in multisystem locomotives," *IEEE Trans. Power Electron.*, vol. 36, no. 5, pp. 5832–5844, 2021.

[12] H.-S. Song, R. Keil, P. Mutschler, J. van der Weem, and K. Nam, "Advanced control scheme for a single-phase PWM rectifier in traction applications," in *38th IAS Annual Meeting on Conf. Record of the Industry Applications Conf.*, vol. 3, 2003, pp. 1558–1565 vol.3.

[13] S. Baltruweit, E. Liegmann, P. Karamanakos, and R. Kennel, "FPGA-implementation friendly long-horizon finite control set model predictive control for high-power electronic systems," in *IEEE 12th Energy Conversion Congress Expo. (ECCE-Asia)*, 2021, pp. 1823–1828.

[14] L. Rovere, A. Formentini, and P. Zanchetta, "FPGA implementation of a novel oversampling deadbeat controller for PMSM drives," *IEEE Trans. Ind. Electron.*, vol. 66, no. 5, pp. 3731–3741, 2019.

[15] L. Stefanski, R. Schwendemann, D. Bernet, M. Widenmeyer, A. Liske, and M. Hiller, "Cascaded H-bridge based parallel hybrid converter – a new voltage source for power-hardware-in-the-loop emulation systems," in *IEEE 21st Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2020, pp. 1–8.

[16] D. Bernet, L. Stefanski, and M. Hiller, "Integrating voltage-source active filters into grid-connected power converters—modeling, control, and experimental verification," *IEEE Trans. Power Electron.*, vol. 36, no. 11, pp. 12 218–12 233, 2021.

 DOI 10.1109/PEMC51159.2022.9962949