IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received November 10, 2020, accepted November 24, 2020, date of publication November 27, 2020,
date of current version December 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3041177

FPGA Based Real Time Simulations of the Face Milling Process

MICHAŁ R. MAZUR, MAREK A. GALEWSKI, AND KRZYSZTOF J. KALIŃSKI
Faculty of Mechanical Engineering, Gdańsk University of Technology, 80-233 Gdańsk, Poland

Corresponding author: Michał R. Mazur (micmazur@pg.edu.pl)

This work was supported in part by the Polish National Center for Research and Development (Application of selected mechatronic solutions to supervise the cutting process of large-size workpieces on multi-axis machining centers) under Grant TANGO1/266350/NCBR/2015.

ABSTRACT The article presents a successful implementation of the milling process simulation at the Field-Programmable Gate Array (FPGA). By using FPGA, very rigorous Real-Time (RT) simulation requirements can be met. The response time of the FPGA simulations is significantly reduced, and the time synchronization is better than in a typical RT system implemented in software. The FPGA-based approach is characterized by enormous flexibility when it comes to input and output operations that can be implemented deterministically in RT. Complex simulation software has been implemented using the High Level Synthesis technique, which is a relatively easy and fast approach for FPGA programming without using complex Hardware Description Languages. The hardware functions are based on procedures written in high-level C programming language. The mathematical descriptions of simulations, results of computer simulations, Hardware-in-the-Loop Simulation experiments, and real experiments are presented. The approach presented in this paper can be used to simulate the dynamics of various mechatronic systems.

INDEX TERMS Field programmable gate arrays, high level synthesis, systems simulation.

I. INTRODUCTION

The paper concerns the simulation of the milling process with simulation procedures implemented in the FPGA (Field-Programmable Gate Array) system using the High Level Synthesis (HLS) technique [1]–[3]. This is an example of Real Time (RT), hardware simulations [4] of complex system dynamics that are able to meet very narrow time limits (at the level of several microseconds) with very low uncertainty (at the level of several nanoseconds) while maintaining accurate simulation uniformity and physical time. Thanks to the use of HLS and FPGA, it is possible to meet such time limits for various types of simulations of different mechatronic systems [5], [6]. The main advantage of the proposed approach to FPGA-based simulations is the possibility to obtain very accurate time and short delayed responses. Input and output signals can be processed in an individual integration time step, greatly expanding the application of Hardware-in-the-Loop Simulation (HiLS) techniques.

The HiLS technique is used to emulate at least a part of a controlled mechanical structure or process [7], [8]. It is possible to simulate or emulate some system components in such a way that they look and behave like they were real for other parts of the systems. Thanks to this, it is possible, for example, to simulate an object for the purposes of building a control system without the need to connect it with a real object. It also increases the security of the development process as there is no risk of damaging the object while testing an unfinished control system. As modern machines and their control systems are increasingly complex, the number of potential errors increases. Detecting and isolating such errors in a complex system can be very difficult. With HiLS it is possible to speed up the process of creating a control system and software. It also allows for quick verification of the impact of parameters' values changes on the examined system. Additionally, it is also a cost-effective approach as the need for experiments in the target environment can be greatly reduced.

The combination of HiLS techniques with the advantages of real-time FPGA simulation enables the construction of a HiLS system taking into account very tight time constraints. For example, instead of building a complete prototype of a complex structure for vibration or fatigue analysis, a part of the structure can be simulated and its dynamics can influence

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

and interact with the dynamics of a physical part of the system. In a large number of HiLS applications, the simulated part responses are limited by time constraints, which can often be difficult to meet [9]. Therefore, special techniques are used to obtain the response time accuracy and reduce its uncertainty (jitter) [10]. Depending on the application, the RT system of the software may serve as the target development platform for HiLS. Such a system can be defined as *hard RT* or *soft RT* which describes its ability to always, or not always meet exact time constraints. However, the exact definitions of the RT system and how to classify such systems differ depending on the literature and on the application [11]–[13]. The problem of *jitter* occurs in every modern RT system [14]. This problem should be limited, as the changing delay in the control loop can have a large impact on the properties of the controlled system [15]. In some cases, a controlled or simulated system may become unstable due to high *jitter* values, even when a limited fixed time delay can be effectively compensated [16]. FPGA based RT systems have very short response times and low *jitter* values that can be limited to a few nanoseconds.

When using an RT system for HiLS, the exact time constraints (requirements) for the simulation loop must be specified. Unfortunately, a common misconception is the expectation that the RT system will compute (simulate) faster than a typical computer system. Even though typical (non-RT) computer systems can be efficiently used to perform complex computational tasks, this does not mean that they can switch between such tasks fast enough, synchronize them accurately, and respond quickly to input signals. Critical RT tasks may require special hardware to reduce response time and increase system reliability so that computation time can be adjusted to real time. In typical software executed RT applications response time are measured in milliseconds and uncertainty can be achieved at the level of microseconds. However, many simulations of the dynamics of mechatronic systems may require the solution of Ordinary Differential Equations (ODE) [17] with an integration step measured in microseconds and floating point double precision is needed as well [18], [19]. To achieve the desired RT requirements, the selection of appropriate algorithms and their further modifications should be considered [10]. Meeting the required time constraints can be difficult for typical processors, even those designed for RT applications. Hence, hardware solutions (especially FPGA based ones) may be required to achieve better response time and better system reliability and repeatability.

In some solutions, mechanical systems are simulated with the use of analog electronic systems [20], [21]. Analog systems can achieve very good response times, but they can be susceptible to various and difficult to eliminate errors (e.g., presence of analog noise). Additionally, making changes to the simulated system may be restricted to a limited range of parameters only because major changes may even require redesigning the analog subsystem or replacing its components.

To solve this problem, the hardware implementation of the computational simulation algorithm can be realized using CPLDs (Complex Programmable Logic Devices) or FPGAs. When operating FPGAs with a relatively low frequency (e.g. 100 MHz), it is possible to obtain response time (excluding the time needed to propagate signals in additional input and output circuits) at the microsecond level and with *jitter* limited to a few nanoseconds. In this way, FPGA-based simulation systems provide a response time that can be compared with analog systems, while maintaining the reliability of digital systems.

The list of possible applications of FPGAs is long and constantly growing [22]. They are mainly used as control units, for signal processing, as computing accelerators and for prototyping Application-Specific Integrated Circuits (ASICs). Currently, FPGAs are used for more complex tasks for two reasons. First, the top-of-the-line FPGAs on the market contain a large number of computationally efficient Digital Signal Processing (DSP) blocks that are important in floating point computations. The computing power of modern FPGAs reaches the levels of Tera Floating Point Operations Per Second (TFLOPS), which makes them competitive for parallel computing compared to typical processors and even Graphical Processing Units (Graphical Processing Units). Moreover, unlike the GPU, FPGA is able to efficiently perform many different tasks simultaneously. The second reason is the growing availability and reliability of HLS tools that allow engineers to effectively implement code written in high-level languages such as C, C++, OpenCL or SystemC on FPGA hardware without having to use rather cumbersome languages such as VHDL or Verilog.

The hardware simulation of the milling process presented in this paper can be used, for example, to develop a device that detects *chatter* vibrations [23]–[25]. Self-excited *chatter* vibration is a phenomenon observed, for example, during milling operations. The *chatter* phenomenon should be avoided, because it leads to a reduction in the quality of the milled surface, increased tool wear, and even to the destruction of the processed part or tool [26], [27]. Thanks to the methods used and their implementation, this risk can be minimized. This may be advantageous, for example, when developing a vibration detecting device, as parts of the systems that may be damaged can be simulated in real time, which is an example of HiLS. Due to the use of HLS, various control procedures can be implemented relatively quickly on FPGA [28], therefore the knowledge presented in this document may be applicable not only to the simulation of the milling process, but also to other HiLS applications or control problems in general [29].

II. MILLING SIMULATIONS

In order to simulate the dynamics of the milling process, an appropriate model of the workpiece, tool and cutting process should be derived and implemented on the FPGA. However, synthesis and implementation times can be very long (usually up to several hours), which has a significant impact on the overall uptime, especially when a significant amount of debugging is required or many consecutive iterations of the algorithm are being developed. This is in contradiction to compile time, as for the classic CPU it is usually much shorter (usually a few to tens of seconds). For this reason, in order to shorten the total development time, a software version had to be developed and tested before the hardware implementation of the simulation algorithm on the FPGA. Additionally, debugging is a much easier task when the algorithm is implemented as a classic program. The results from the software version of the algorithm can later be used as a reference when implementing a similar algorithm in FPGA. The software version can be profiled, divided into parts and then parts intended to be implemented in ''hard'' RT can be selected. By analyzing the results and performance of software simulations, time constraints for RT can be defined and investigated. Later, simplifications of the model can be introduced to the code and analyzed. Choosing different algorithms for solving specific tasks can be considered when developing a software version. Some algorithms may be more suitable for high-frequency CPU, but at the same time they are not as effective when used in GPU or FPGA implementations.

Making further changes is also easier and faster for the software version than for the hardware version, so there is a need to carefully analyze the software version before developing the hardware one.

In the presented work, the software version was based on the *Amikro4* program, which was previously developed to simulate the dynamics of the face milling process. This program was originally written in the FORTRAN77 programming language and performs the calculations described in the following chapters [30].

A. DESCRIPTION OF THE CUTTING PROCESS DYNAMICS

For the purposes of simulating the dynamics of the face milling process, the following assumptions were made [31], [32].

- The spindle and workpiece are separated from the machine structure. The rest of the milling machine is neglected [33]–[35].
- The flexibility of the tool and the workpiece was taken into account [36].
- Coupling elements (CEs) were used to model the dynamics of the cutting process.
- The effect of the first pass of the edge along the cutting layer causes proportional feedback, and the effect of multiple passes additionally the delayed feedback (these two feedback are considered the main cause of *chatter* vibration [33], [27]).

For the instantaneous point of contact between the selected edge of the tool and the workpiece (idealized by CE no. *1*), a proportional model of cutting dynamics was taken into account [30], [31]. Based on this model, the cutting forces depend proportionally on the instant thickness of the cutting layer $h_l(t)$, as well as on the instant width of the cutting layer $b_l(t)$. According to the direction of action, the cutting force component F_{yl1} acting along the nominal cutting speed, the cutting force component F_{yl2} acting along the thickness of the cutting layer, and additionally the cutting force component F_{yl3} acting along the width of the cutting layer are separated (Fig. 1).

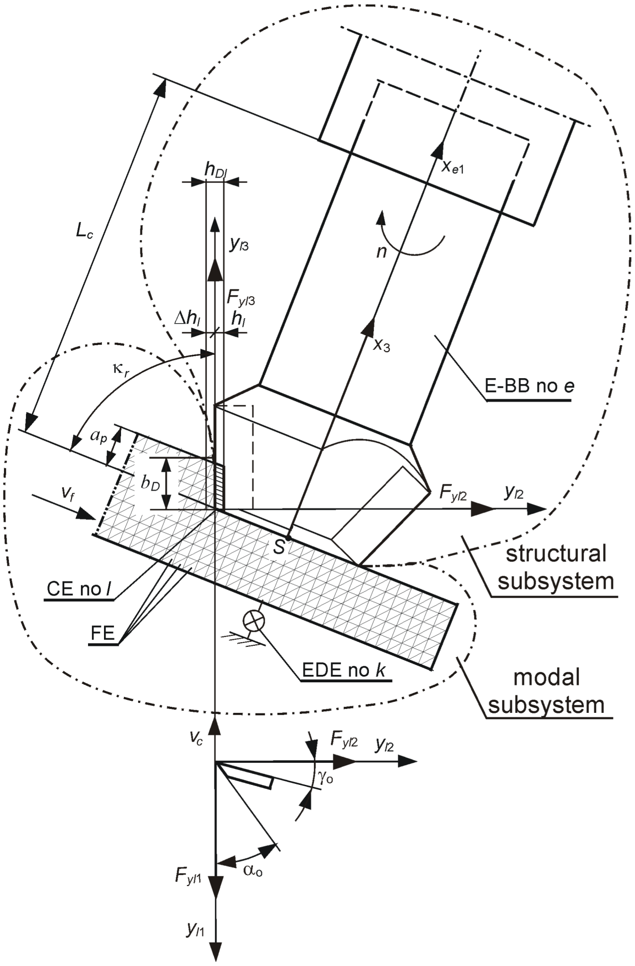

FIGURE 1. Scheme of a face milling of a flexible workpiece.

$$F_{yl1}(t) = \begin{cases} k_{dl} b_l(t) h_l(t), & h_l(t) > 0 \wedge b_l(t) > 0, \\ 0, & h_l(t) \leq 0 \vee b_l(t) \leq 0, \end{cases} \tag{1}$$

$$F_{yl2}(t) = \begin{cases} \mu_{l2} k_{dl} b_l(t) h_l(t), & h_l(t) > 0 \wedge b_l(t) > 0, \\ 0, & h_l(t) \leq 0 \vee b_l(t) \leq 0, \end{cases} \tag{2}$$

$$F_{yl3}(t) = \begin{cases} \mu_{l3} k_{dl} b_l(t) h_l(t), & h_l(t) > 0 \wedge b_l(t) > 0, \\ 0, & h_l(t) \leq 0 \vee b_l(t) \leq 0, \end{cases} \tag{3}$$

where:

$$b_l(t) = b_D(t) - \Delta b_l(t),$$
$$h_l(t) = h_{Dl}(t) - \Delta h_l(t) + \Delta h_l(t - \tau_i),$$

$b_D(t)$ – desired cutting layer width; $b_D(t) = a_p(t)/\sin(\kappa_r)$,
$\Delta b_l(t)$ – dynamic change in cutting layer width for CE no. l,

$h_{Dl}(t)$ – desired cutting layer thickness for CE no. l; $h_{Dl}(t) \cong f_z \sin(\kappa_r) \cos\varphi_l(t)$,

Δh_l – dynamic change in cutting layer thickness for CE no. l,

k_{dl} – average dynamic specific cutting pressure for CE no. l,

μ_{l2}, μ_{l3} – cutting force ratios for CE no. l, as quotients of forces F_{yl2} and F_{yl1}, and forces F_{yl3} and F_{yl1},

τ_l – time-delay between the same position of CE no. l and of CE no. l–1,

$a_p(t)$ – desired depth of cutting,

κ_r – edge angle,

f_z – feed per edge,

$\varphi_l(t)$ – angular position of CE no. l [31].

Relationships (1) – (3), which describe cutting forces for CE no. l in case of 3-dimensional proportional model, may be presented with the use of matrix notation (4), as shown at the bottom of the next page, or using the abbreviated notation (5), as shown at the bottom of the next page, where:

$\breve{\mathbf{F}}_l\,(t)$ – vector of cutting forces of CE no. l,

$\breve{\mathbf{F}}_l^0\,(t)$ – vector of cutting forces of CE no. l, resulted from a desired cutting geometry and kinematics,

$\breve{\mathbf{D}}_{Pl}^{(l)}\,(t)$ – matrix of linear proportional feedback interactions,

$\breve{\mathbf{D}}_{Pl}^{(n)}\,(t)$ – matrix of nonlinear proportional feedback interactions,

$\breve{\mathbf{D}}_{Ol}^{(n)}\,(t)$ – matrix of linear time-delayed feedback interactions,

$\breve{\mathbf{D}}_{Ol}^{(n)}\,(t)$ – matrix of nonlinear time-delayed feedback interactions,

$\Delta\breve{\mathbf{w}}_l\,(t)$ – vector of deflections of CE no. l at instant of time t,

$\Delta\breve{\mathbf{w}}_l\,(t-\tau_l)$ – vector of deflections of CE no. l at instant of time $t - \tau_l$,

$q_{zl}\,(t)$ – relative displacement of edge and workpiece along direction y_{l1} at instant of time t,

$q_{zl}\,(t-\tau_l)$ – relative displacement of edge and workpiece along direction y_{l1} at instant of time $t - \tau_l$.

Vector (5) can also be described in six-dimensional space, (6), as shown at the bottom of the next page, where:

$$\mathbf{F}_l\,(t) = col\left(\breve{\mathbf{F}}_l\,(t)\,,\mathbf{0}_{3\times1}\right), \tag{7}$$

$$\Delta\mathbf{w}_l\,(t) = col\left(\Delta\breve{\mathbf{w}}_l\,(t)\,,\mathbf{0}_{3\times1}\right), \tag{8}$$

$$\mathbf{F}_l^0\,(t) = col\left(\breve{F}_l^0\,(t)\,,\mathbf{0}_{3\times1}\right), \tag{9}$$

$$\mathbf{D}_{Pl}^{(l)}\,(t) = \begin{bmatrix} \breve{\mathbf{D}}_{Pl}^{(l)}\,(t) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{6\times6}, \tag{10}$$

$$\mathbf{D}_{Pl}^{(n)}\,(t) = \begin{bmatrix} \breve{\mathbf{D}}_{Pl}^{(n)}\,(t) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{6\times6}, \tag{11}$$

$$\mathbf{D}_{Ol}^{(l)}\,(t) = \begin{bmatrix} \breve{\mathbf{D}}_{Ol}^{(l)}\,(t) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{6\times6}, \tag{12}$$

$$\mathbf{D}_{Ol}^{(n)}\,(t) = \begin{bmatrix} \breve{\mathbf{D}}_{Ol}^{(n)}\,(t) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{6\times6}. \tag{13}$$

In order to simplify further notation, relationship (6) takes the form:

$$\mathbf{F}_l\,(t) = \mathbf{F}_l^0\,(t) - \mathbf{D}_{Pl}\,(t)\,\Delta\mathbf{w}_l\,(t) + \mathbf{D}_{Ol}\,(t)\,\Delta\mathbf{w}_l\,(t-\tau_l)\,, \tag{14}$$

where:

$$\mathbf{D}_{Pl}\,(t) = \mathbf{D}_{Pl}^{(l)}\,(t) - \mathbf{D}_{Pl}^{(n)}\,(t)\,, \tag{15}$$

$$\mathbf{D}_{Ol}\,(t) = \mathbf{D}_{Ol}^{(l)}\,(t) - \mathbf{D}_{Ol}^{(n)}\,(t)\,. \tag{16}$$

As a result of modeling the milling process, a hybrid system is obtained, which consists of separate subsystems (Fig. 1). These subsystems are:

- modal subsystem, i.e. a stationary model using the Finite Element Method (FEM) of a flexible workpiece supported by Elastic-Damping Elements (EDE), which moves at a given feed rate v_f. Initially, the subsystem is idealized as a set of tetragonal 10-node finite elements (FE) [30] and has a large number of degrees of freedom. However, after the modal transformation [31], [34], the behavior of this subsystem is described by the vector of its modal coordinates **a**, the number of which is in practice much smaller than the corresponding number of degrees of freedom. Therefore, when we consider the finite number of normal modes of the subsystem, we define its dynamic properties with:

  $\mathbf{\Omega} = diag(\omega_{0i})$ – matrix of angular natural frequencies of the modal subsystem; $i = 1, \ldots, mod$. This is also called the stiffness modal matrix;

  $\mathbf{\Psi} = [\mathbf{\Psi}_1 \ldots \mathbf{\Psi}_{mod}]$ – matrix of the considered mass normalised normal modes of the modal subsystem; $i = 1, \ldots, mod$;

  $\mathbf{Z} = diag\,(\zeta_i)$ – matrix of dimensionless damping coefficients (also called, *modal damping*) of the modal subsystem; $i = 1, \ldots, mod$;

- structural subsystem, i.e. a non-stationary discrete model of a rotating spindle (with a given spindle speed n) of a face milling cutter (i.e. a flexible finite element like Euler-Bernoulli Bar (E-BB) no. e [31], [34], having the x_{e1}, x_{e2}, x_{e3} coordinate system) and a cutting process (i.e. Coupling Element (CE) no l [31], [32] placed at the instantaneous position of the "active" cutting edge [31]). The edges are "active" when they are in contact with the workpiece, the others are called "inactive". The behavior of a subsystem is described by the vector of its generalized coordinates **q**. The dynamic properties of the decoupled structural subsystem (i.e. E-BB modeling the tool itself) are determined by the matrices of inertia **M**, damping **L** and stiffness **K**;
- abstract connecting subsystem as a conventional S contact point between tool and workpiece. Its generalized

coordinates are related to the remaining equations with time-dependent constraints [31], [37]. The latter allows to eliminate these generalized coordinates from the description of the behavior of the hybrid system.

B. DESCRIPTION OF THE DYNAMICS OF FLEXIBLE DETAIL IN HYBRID SYSTEM COORDINATES

Vector of deflections of CE no. l is expressed as a function of vector of generalised coordinates $\mathbf{q}$ and vector of modal coordinates $\mathbf{a}$. Hence, the relationship (17), as shown at the bottom of the next page, is obtained [31], [34], where:

$\boldsymbol{\xi} = \begin{Bmatrix} \mathbf{q} \\ \mathbf{a} \end{Bmatrix}$ – vector of hybrid coordinates of the hybrid system,

$\mathbf{T}_l(t)$ – transformation matrix of displacements' vector $\mathbf{q}$ from coordinate system x_{e1}, x_{e2}, x_{e3} of E-BB no. e, to coordinate system y_{l1}, y_{l2}, y_{l3} of CE no. l [30]–[32],

$\mathbf{W}_l(t)$ – matrix of constraints between displacements' vector in modal coordinates $\mathbf{a}$, and displacements in coordinate system y_{l1}, y_{l2}, y_{l3} of CE no. l [30], [31].

After transformation of the vector of force interaction of CE no. l (14) to hybrid coordinates, (18), as shown at the bottom of the next page, is obtained:

As the result of the hybrid system's consideration, the matrix equation of dynamics of non-stationary model of the milling process in hybrid coordinates will have the form (19), as shown at the bottom of the next page, [31], [34], [37] where:

i_l – number of "active" coupling elements [31], [34]. In order to identify modal model of the flexible workpiece (which is a part of Eq. (19)), the matrix of normal modes $\boldsymbol{\Psi}$ and matrix of corresponding angular natural frequencies $\boldsymbol{\Omega}$ of the modal subsystem must be determined. Separating the modal subsystem from the whole non-stationary structure makes it possible to reduce the finite element model to a few modes only. Its number depends on the importance and necessity of choosing the modes for further analysis. As the result, the size of the model is significantly reduced.

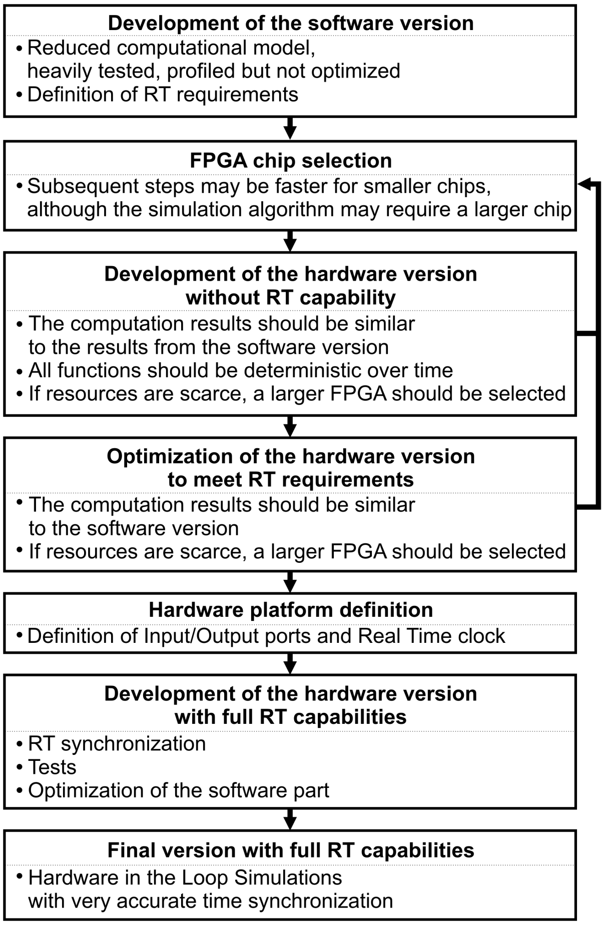
Development of the software version
• Reduced computational model, heavily tested, profiled but not optimized
• Definition of RT requirements
FPGA chip selection
• Subsequent steps may be faster for smaller chips, although the simulation algorithm may require a larger chip
Development of the hardware version without RT capability
• The computation results should be similar to the results from the software version
• All functions should be deterministic over time
• If resources are scarce, a larger FPGA should be selected
Optimization of the hardware version to meet RT requirements
• The computation results should be similar to the software version
• If resources are scarce, a larger FPGA should be selected
Hardware platform definition
• Definition of Input/Output ports and Real Time clock
Development of the hardware version with full RT capabilities
• RT synchronization
• Tests
• Optimization of the software part
Final version with full RT capabilities
• Hardware in the Loop Simulations with very accurate time synchronization

FIGURE 2. Development stages of hardware (FPGA) Real Time simulations of a mechatronic system.

III. IMPLEMENTATION

The entire process of computer hardware development carried out in the simulation of the mechatronic system is presented in general in Fig. 2. Each stage of this process is described in detail in the following chapters of thie paper.

$$\begin{aligned}
\underbrace{\begin{Bmatrix} F_{yl1} \\ F_{yl2} \\ F_{yl3} \end{Bmatrix}}_{\breve{\mathbf{F}}_l(t)} = \underbrace{\begin{bmatrix} k_{dl} b_D(t)\, h_{Dl}(t) \\ \mu_{l2} k_{dl} b_D(t)\, h_{Dl}(t) \\ \mu_{l3} k_{dl} b_D(t)\, h_{Dl}(t) \end{bmatrix}}_{\breve{\mathbf{F}}_l^0(t)} - \left\{ \underbrace{\begin{bmatrix} 0 & k_{dl} b_D(t) & k_{dl} h_{Dl}(t) \\ 0 & \mu_{l2} k_{dl} b_D(t) & \mu_{l2} k_{dl} h_{Dl}(t) \\ 0 & \mu_{l3} k_{dl} b_D(t) & \mu_{l3} k_{dl} h_{Dl}(t) \end{bmatrix}}_{\breve{\mathbf{D}}_{Pl}^{(l)}(t)} - \underbrace{\begin{bmatrix} 0 & k_{dl} \Delta b_l(t) & 0 \\ 0 & \mu_{l2} k_{dl} \Delta b_l(t) & 0 \\ 0 & \mu_{l3} k_{dl} \Delta b_l(t) & 0 \end{bmatrix}}_{\breve{\mathbf{D}}_{Pl}^{(n)}(t)} \right] \cdot \underbrace{\begin{Bmatrix} q_{zl}(t) \\ \Delta h_l(t) \\ \Delta b_l(t) \end{Bmatrix}}_{\Delta \breve{\mathbf{w}}_l(t)} \\
+ \left\{ \underbrace{\begin{bmatrix} 0 & k_{dl} b_D(t) & 0 \\ 0 & \mu_{l2} k_{dl} b_D(t) & 0 \\ 0 & \mu_{l3} k_{dl} b_D(t) & 0 \end{bmatrix}}_{\breve{\mathbf{D}}_{Ol}^{(l)}(t)} - \underbrace{\begin{bmatrix} 0 & k_{dl} \Delta b_l(t) & 0 \\ 0 & \mu_{l2} k_{dl} \Delta b_l(t) & 0 \\ 0 & \mu_{l3} k_{dl} \Delta b_l(t) & 0 \end{bmatrix}}_{\breve{\mathbf{D}}_{Ol}^{(n)}(t)} \right] \cdot \underbrace{\begin{Bmatrix} q_{zl}(t-\tau_l) \\ \Delta h_l(t-\tau_l) \\ \Delta b_l(t-\tau_l) \end{Bmatrix}}_{\Delta \breve{\mathbf{w}}_l(t-\tau_l)}
\end{aligned} \tag{4}$$

$$\breve{\mathbf{F}}_l(t) = \breve{\mathbf{F}}_l^0(t) - \left(\breve{\mathbf{D}}_{Pl}(t) - \breve{\mathbf{D}}_{Pl}^{(n)}(t) \right) \Delta \breve{\mathbf{w}}_l(t) + \left(\breve{\mathbf{D}}_{Ol}(t) - \breve{\mathbf{D}}_{Ol}^{(n)}(t) \right) \Delta \breve{\mathbf{w}}_l(t-\tau_l) \tag{5}$$

$$\mathbf{F}_l(t) = \mathbf{F}_l^0(t) - \left(\mathbf{D}_{Pl}^{(l)}(t) - \mathbf{D}_{Pl}^{(n)}(t) \right) \Delta \mathbf{w}_l(t) + \left(\mathbf{D}_{Ol}^{(l)}(t) - \mathbf{D}_{Ol}^{(n)}(t) \right) \Delta \mathbf{w}_l(t-\tau_l) \tag{6}$$

A. DEVELOPMENT OF THE SOFTWARE VERSION

The *Amikro4* software version was developed in the FORTRAN77 language and was based on (19). This simulation software uses the Newmark-Beta algorithm (ODE solving), a Gaussian elimination algorithm and several linear matrix operations. In order to develop the hardware version of *Amikro4*, the software version has been used as reference.

When using ODE resolution procedures, it is important to select the appropriate time step. Shorter time steps require more computation and lead to longer simulation times, while longer time steps usually lead to less accurate results but acquired in a shorter time. For the *Amikro4* simulation software, the fixed time step Δt was set to 42 μs. After each time step Δt, the input and output signals are synchronized. It is then assumed that the input and output signals may not change during this time step. However, input signals may appear during the Δt time step. Thus, the time step Δt defines the maximum time delay of the RT version of the *Amikro4* simulation software. Signal propagation times and timing uncertainties are generally small, i.e., a few nanoseconds, and are thus ignored.

The optimized version of the *Amikro4* software was compiled and run on the computer system of the Zynq UltraScale+MPSoC ZCU102 development board, which was running under the control of Ubuntu Linux 16.04. This version worked correctly in terms of numerical results, but it was not possible to achieve even soft RT expectations on the CortexA53 CPU at the standard frequency of 1.2 GHz. The total execution time (350.712 s) was almost 3 times as long as the simulated process time (120 s). Thanks to the program profiling (using the *gprof* tool), it was possible to assess the time needed for each of the *Amikro4* program functions, and then decide whether it is possible to accelerate it (Table 1).

TABLE 1. Profile analysis of the five most time-consuming functions of the Amikro4 software version (CortexA53 Linux).

Function name	Description	Number of calls	Self-time [s]	% of total time
rozw	Solves linear equations by Gaussian elimination	2993343	184.62	58.52
mawie	Finds the active node in the FEM model	2374137	43.22	13.70
newmar	Solves ODE with the Newmark-Beta method	2993342	37.42	11.86
mn2ma4	Matrix multiplication	15536113	33.14	10.51
mlkf1	Matrix composition	2993343	13.45	4.26

The most time-consuming function is *rozw*, which solves linear equations by Gaussian elimination. To simulate the milling process, which lasted 120 s, approximately 184.62 s with the CortexA53 (1.2 GHz) was only required for the calculation of the *rozw* function.

B. FPGA SELECTION

The software version of the *Amikro4* did not meet the RT requirements for CortexA53 (1.2 GHz), because the computational time t_c (Fig. 3) was greater than the minimum time step Δt. For comparison, on a PC with an Intel i7-6700 processor, the total computation time was about 7.75 seconds for 120 seconds of simulated process time, however, it is very difficult to synchronize the input and output operations with the RT capabilities of the PC.

The decision on the final selection of a specific FPGA chip can be confirmed after a successful computer simulation of the tested algorithm operating on this part of the FPGA (Fig. 2). Thanks to the use of IDE (Integrated Development Environment), such as Xilinx SDSoC (Software-Defined System-On-Chip), software development and integration of hardware acceleration functions (performed in the FPGA Programmable Logic) is very simplified and automated. Usually, there is no need to supervise the entire synthesis and implementation process of the automatically generated HDL

$$\Lambda \mathbf{w}_i(t) = \mathbf{T}_l(t)\mathbf{q} - \mathbf{W}_i(t)\mathbf{a} = [\mathbf{T}_l(t) - \mathbf{W}_i(t)] \begin{Bmatrix} \mathbf{q} \\ \mathbf{a} \end{Bmatrix} = [\mathbf{T}_l(t) - \mathbf{W}_l(t)]\,\xi \tag{17}$$

$$\begin{aligned} \begin{bmatrix} \mathbf{T}_l^T(t) \\ -\mathbf{W}_l^T(t) \end{bmatrix} \mathbf{F}_l(t) &= \begin{bmatrix} \mathbf{T}_l^T(t) \\ -\mathbf{W}_l^T(t) \end{bmatrix} \mathbf{F}_l^0(t) + \begin{bmatrix} \mathbf{T}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{T}_l(t) & -\mathbf{T}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{W}_l(t) \\ -\mathbf{W}_l^T\mathbf{D}_{pl}(t)\mathbf{T}_l(t) & \mathbf{W}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{W}_l(t) \end{bmatrix} \begin{Bmatrix} \mathbf{q}_s \\ \mathbf{a}_m \end{Bmatrix} \\ &\quad + \begin{bmatrix} \mathbf{T}_l^T(t)\mathbf{D}_{ol}(t) \\ -\mathbf{W}_l^T(t)\mathbf{D}_{ol}(t) \end{bmatrix} \Delta \mathbf{w}_l\,(t - \tau_l) \end{aligned} \tag{18}$$

$$\begin{aligned} &\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \ddot{\xi} + \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & 2\mathbf{Z}\boldsymbol{\Omega} \end{bmatrix} \dot{\xi} + \begin{bmatrix} \mathbf{K} + \sum_{l=1}^{i_i} \mathbf{T}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{T}_l(t) & -\sum_{l=1}^{i_i} \mathbf{T}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{W}_l(t) \\ -\sum_{l=1}^{i_i} \mathbf{W}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{T}_l(t) & \boldsymbol{\Omega}^2 + \sum_{l=1}^{i_i} \mathbf{W}_l^T(t)\mathbf{D}_{pl}(t)\mathbf{W}_l(t) \end{bmatrix} \xi \\ &= \begin{bmatrix} \sum_{l=1}^{i_i} \mathbf{T}_l^T(t)\mathbf{F}_l^0(t) + \mathbf{T}_l^T(t)\mathbf{D}_{ol}(t)\Delta\mathbf{w}\,(t - \tau_l) \\ -\sum_{l=1}^{i_i} \mathbf{W}_l^T(t)\mathbf{F}_l^0(t) + \mathbf{W}_l^T(t)\mathbf{D}_{ol}(t)\Delta\mathbf{w}\,(t - \tau_l) \end{bmatrix} \end{aligned} \tag{19}$$

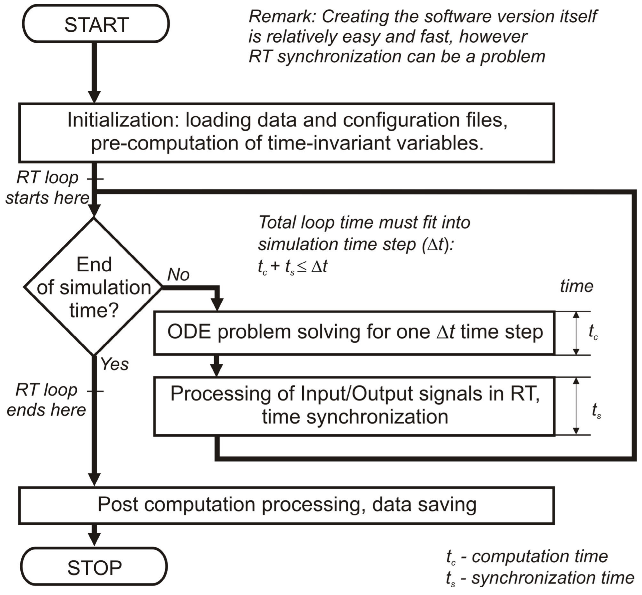
START
Remark: Creating the software version itself is relatively easy and fast, however RT synchronization can be a problem
Initialization: loading data and configuration files, pre-computation of time-invariant variables.
RT loop starts here
End of simulation time?
No
Yes
Total loop time must fit into simulation time step (Δt):
$t_c + t_s \leq \Delta t$
time
ODE problem solving for one Δ*t* time step
t_c
RT loop ends here
Processing of Input/Output signals in RT, time synchronization
t_s
Post computation processing, data saving
STOP
t_c - computation time
t_s - synchronization time

FIGURE 3. **Simplified software-only algorithm for Real Time simulation of a mechatronic system.**

code. However, the function selected for hardware acceleration must match the physical limits of the FPGA. If it requires more resources (i.e., logical cells or DSP blocks) than the actual selected FPGA provides, the automatic integration process will fail. Therefore, it is recommended to start the implementation with the appropriate software, including a simulator of various parts of the FPGA, which helps in choosing the right target FPGA. Knowing the computational complexity of the implemented algorithm, one can try to predict how the requirements for FPGA resources will change depending on the size of the task, but in general it is very difficult to assess exact needs.

Note that it is not recommended to select the largest FPGA available. Such FPGAs will not only be expensive and require more energy to operate, but the synthesis of larger FPGAs is also very time consuming. If the FPGA chip is already selected (for different reasons, for example), it may be advantageous to perform the initial layout using a smaller chip (preferably from the same family) at an early stage of development, and then switch to a larger layout when needed or at the final stage of development. This approach saves a lot of work time.

Currently, there are various FPGAs available on the market from different vendors (e.g. Altera, Xilinx). Each retailer offers different families of chips and they are designed for different purposes. Unfortunately, probably none of the chips currently available are dedicated to RT simulation of mechatronic systems. The most similar applications seem to be RT control and signal processing, and the use of FPGAs as accelerators. However, the amount of resources of a particular FPGA is much more important than the actual target. FPGAs are generally very flexible when it comes to their configurability. The number of I/O ports is usually not as important for a mechatronic system RT simulation as the number and types of integrated DSP blocks.

In general, the largest FPGAs are the richest in terms of available resources, which makes them better suited as a basis for implementing a complex simulation algorithm. It is tempting to choose a larger and faster FPGA chip, because they require less work during implementation. Unfortunately, the synthesis times of larger FPGAs are much longer. Such chips are usually available as standalone units or as part of a PCIe card installed in a personal computer to act as an accelerator. For standalone units, it may be required to implement all additional components in the hardware (e.g., a serial port used for communication between FPGA and PC) or to deploy some FPGA resources for a programmable processor that can be used for input/output operations, initialization, or performance of various tasks that are not carried out directly by the equipment. In the case of a PCIe card, some FPGA resources can be used to manage the PCIe interface and provide a base (hardware platform) for the reconfigurable part that can be used for code created with the OpenCL toolkit. Dedicated software tools are needed for this. In this case, it is a matter of design whether OpenCL is only used to initiate and control the simulation, but all logic is implemented as a defined hardware platform or the simulation is directly implemented by OpenCL which is based on a C-like programming language. Accelerator seems to be a simple and attractive solution, but the problem is the delay in making OpenCL calls to the card from the host CPU. When an FPGA or GPU is used as an accelerator, the number of such calls should be limited to achieve the desired performance [38]. In the case of RT simulations, this usually means that the entire simulation algorithm must be implemented in hardware, as the time needed for synchronization and data transfer between the CPU and the accelerator may be too long. The medium-sized FPGA chips also appear as System on Chip (SoC) units that are equipped with real hardware processors and other required resources to form the basis of an independent computer system. In general, such systems have much lower latency in communication between the CPU and the functions performed on the FPGA, however, this latency may be still too high for some RT requirements. SoC-based FPGAs are much more common and therefore it may be easier and more reliable to use them to implement a simulation code.

Xilinx Zynq UltraScale + MPSoC ZCU102 Evaluation Kit [39] was selected for the purposes of the presented work. It is based on the Xilinx XCZU9EG-2FFVB1156I SoC [40] chip, which includes four 64-bit processors with the Cortex A53 core (maximum clock frequency 1.5 GHz) and two 32-bit Cortex R5 cores (maximum clock frequency 600 MHz) designed for RT applications. The integrated FPGA chip has 599550 logical cells and 2520 DSP48 slices, so at the time of project development (year 2018/2019) it was classified as a mid-range product. The ZCU102 system ran software from an SD memory card. Linux-based Ubuntu 16.04 Base (AArch64 OS architecture) was used as the Operating System (OS) for the ZCU102. A similar OS worked on the

DELL Precision 3620 Workstation (x86_64 system architecture), which was used to develop software for this project. Various free and commercial software tools were used during the development process. Xilinx products were used in the FPGA software to configure and implement simulation algorithms. It was: Vivado HLx 2017.4 – software package for the synthesis and analysis of Hardware Description Languages (HDL) and HLS projects, PetaLinux 2017.4 – tools for customizing, building and implementing a Linux-based operating system in the embedded system, SDK 2017.4 – Software Development Kit and Integrated Design Environment (IDE) for creating embedded applications and SDSoC 2017.4 – dedicated IDE software in High Level Languages (C, C ++, SystemC or OpenCL) with hardware accelerated functions (using Programmable Logic). SDSoC is designed to simplify and automate software development with hardware accelerated functions, and its functionality relies on Vivado HLx, PetaLinux and SDK. In this way, similar effects could be achieved using only these three tools. Additionally, some functions can only be implemented using one of these three tools, such as defining the I/O interfaces for part of the FPGA system. However, SDSoC automates the integration of hardware-implemented functions (Vivado's block design process) and provides drivers and runtimes for various runtime environments: Linux, FreeRTOS, and Baremetal (*libmetal*).

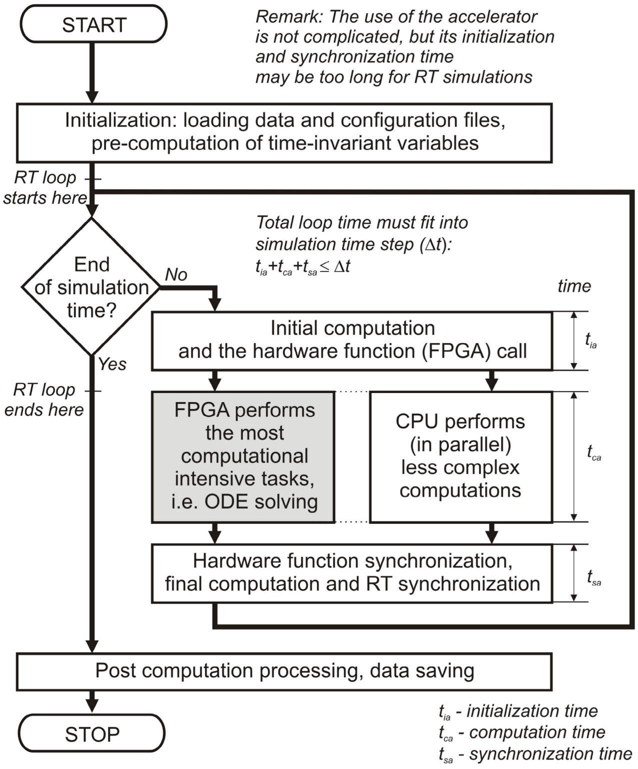
START
Remark: The use of the accelerator is not complicated, but its initialization and synchronization time may be too long for RT simulations
Initialization: loading data and configuration files, pre-computation of time-invariant variables
RT loop starts here
End of simulation time?
No
Yes
Total loop time must fit into simulation time step (Δt): $t_{ia}+t_{ca}+t_{sa} \leq \Delta t$
time
Initial computation and the hardware function (FPGA) call
t_{ia}
RT loop ends here
FPGA performs the most computational intensive tasks, i.e. ODE solving
CPU performs (in parallel) less complex computations
t_{ca}
Hardware function synchronization, final computation and RT synchronization
t_{sa}
Post computation processing, data saving
STOP
t_{ia} - initialization time
t_{ca} - computation time
t_{sa} - synchronization time

FIGURE 4. **Simplified algorithm of mixed hardware and software implementation of mechatronic system simulation in Real Time – acceleration approach.**

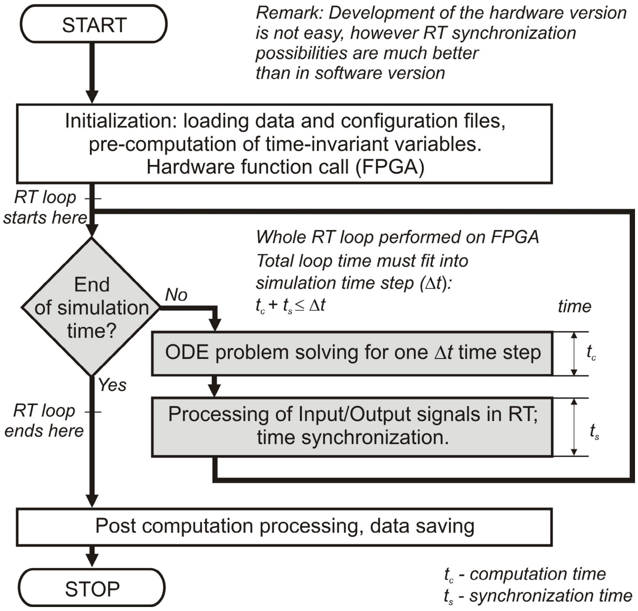
START
Remark: Development of the hardware version is not easy, however RT synchronization possibilities are much better than in software version
Initialization: loading data and configuration files, pre-computation of time-invariant variables. Hardware function call (FPGA)
RT loop starts here
End of simulation time?
No
Yes
Whole RT loop performed on FPGA
Total loop time must fit into simulation time step (Δt): $t_c + t_s \leq \Delta t$
time
ODE problem solving for one Δt time step
t_c
RT loop ends here
Processing of Input/Output signals in RT; time synchronization.
t_s
Post computation processing, data saving
STOP
t_c - computation time
t_s - synchronization time

FIGURE 5. **Simplified hardware algorithm implementation of mechatronic system in Real Time.**

C. DEVELOPMENT OF THE HARDWARE VERSION

Based on the software version of the implemented algorithm, a preliminary decision can be made before implementation. The first one is the choice of whether the entire algorithm is to be implemented on hardware (FPGA) or whether some procedures can be implemented by software running on the CPU (Fig. 4). It must also be decided how the data should be managed and whether the code can work simultaneously on different processor cores and the FPGA accelerator, or it must be implemented only by the hardware (Fig. 5). The required amount of FPGA resources, latency in communication between the processor and the implemented hardware functions must be analyzed as well.

In the presented example, using Xilinx SDSoC, two functions *newmar* and *rozw* (see Table 1) were integrated into one and implemented as a function of hardware acceleration on FPGA (as in Fig. 4). The optimized FPGA implementation of this function worked fine, and it only took 345 FPGA cycles to complete it. FPGA worked at 100 MHz.

Given the number of 2993342 calls to this function, the total calculation time for *rozw* and *newmar* functions was estimated at 10.33 s, which is 21.6 times faster than the version of the software running on CortexA53. Standard Ubuntu 16.04 *gcc* compiler was used and all relevant optimization options were enabled. However, the number of calls to the *rozw* and *newmar* functions of 2993342 is very high for the simulation of 120 s of the milling process. In this case, calling the hardware function takes more time than its performance and therefore the complete execution of the *Amikro4* program with hardware accelerated *rozw* and *newmar* functions took several minutes longer than the software version. The reason is that each hardware function call requires many time-consuming operations. This problem is further investigated in Appendix A.

The number of calls for functions that require acceleration is very large. The amount of data that needs to be copied from, and to the FPGA is also too large for the specified time interval. Even twenty times faster execution of individual functions would not be enough to obtain the execution of the code in defined RT expectations. Therefore, it was decided that all program initialization will be performed in the program part operating in the Linux environment, however, the entire main loop with all necessary functions must be transferred to FPGA (Fig. 5). Since the original version of *Amikro4* was written in FORTRAN77, all these functions were rewritten in the C programming language to enable their further processing using HLS tools. The hardware-accelerated main loop of the program was exported as a dynamically linked library by the Xilinx SDSoC tool, and then connected to the remaining non-hardware-accelerated part of the program written in FORTRAN77.

Using the HLS technique, it is relatively easy to transfer the implementation of selected functions to the FPGA. The problem is to match the hardware implemented functions to the physical limitations of the selected FPGA and at the same time to meet the required time limits. The program code requires appropriate modification so that its performance is time-deterministic. This means that some parts of the algorithm must be corrected and rewritten. Due to the limited resources of FPGAs, wherever possible, single-precision floating-point or even fixed-point should be used instead of double-precision floating-point variables. For example, loops should have a maximum number of iterations defined as constants instead of variables, and conditional execution based on nondeterministic input values should be avoided. For example, if a computational exception (such as division by zero) is selected, computations should continue even if there is Not a Number (NaN) or Infinity (Inf) result. It is possible to return a certain *Boolean* control value if an exception occurs instead of the abort. Such changes and analysis of the complex simulation algorithm needed an in-depth knowledge and understanding of how the algorithm works. For example, an analysis of the required FPGA cycles to perform a specific function may show that it would not meet the predefined RT limits if it had to calculate cutting forces for all cutting edges of a milling tool, but during simulation it is known that this situation will not occur at all (one or more edges have no contact with the workpiece, at every stage of the simulation). Moreover, some changes can be effectively introduced with this knowledge, for example the task dimension (modal model dimension) can be reduced based on the concept of effective mass. Finally, it is worth analyzing which of the variable parameters can be defined as software constants for hardware implementation. Constant parameters are optimized during the implementation of FPGA, and thus effectively reduces the consumption of its resources, so considering different FPGA implementations for different values of a particular parameter is advised.

D. DEFINITION OF THE HARDWARE VERSION

Several optimization techniques can be used in HLS tools [41]. The entire task that must be performed on the hardware can be divided into modules and data can be transferred between modules using First-In-First-Out (FIFO) queues. Each module performs a specific task at a different time. However, it is worth noting that FPGAs have limited on-line reconfiguration possibilities, and dedicating FPGA resources to a specific task will make them unavailable for other tasks. Program functions and loops can be pipelined or unrolled. The internal FPGA memory should be properly partitioned and distributed to enable efficient parallel execution of individual functions. Further changes to the software code can be made, for example by adding additional variables that will enable the pipeline of the selected loop. Loops can be divided into blocks that will be processed in parallel. Different operating frequencies can be selected for certain functions. Typically, when using one optimization technique, it is difficult to predict its impact on code execution speed and resource consumption. It is more or less a trial and error method, however, understanding the fundamental differences between executing software code on the processor and hardware implementation with FPGA is very helpful. In the case of a complex program with numerous small-size tasks, choosing the right optimization strategy is not easy. Current versions of tools like Xilinx Vivado HLS and Xilinx SDSoC are dedicated to transferring small (uncomplicated) functions to hardware and have limited ability to deal with problems that arise when working with more complex algorithms. Due to these limitations, a top-down approach to optimizing a complex program on an FPGA is not recommended.

By using Xilinx SDSoC, it was possible to match the main simulation loop to the limited resources of the selected FPGA and achieve a run time of each step of this loop shorter than the actual simulation time step. The working implementation used only 49.25% of the available DSP48 blocks and 58.74% of the available Lookup Tables (LUTs), so in theory, only about half of the key FPGA resources were used. However, the real limiting factor was found during deeper analysis of the Xilinx SDSoC logs. It was the use of Configurable Logical Blocks (CLBs) that reached 85.98% in the final working implementation. Initial designs suffered from too much condensation: the logic distribution during the FPGA resource deployment process was limited by routing capabilities.

E. DEFINITION OF THE HARDWARE PLATFORM WITH RT CAPABILITIES

The Xilinx SDSoC tool is not intended for building RT applications and therefore it was necessary to add some modifications to the software and directly on the hardware. Custom hardware platform (initial SoC configuration, mainly FPGA part) was defined with Xilinx Vivado tool (Fig. 6) for later use with Xilinx SDSoC.

These modifications included a 64-bit clock counter and corresponding register used to synchronize the RT process, a discrete input register and an output register with hardware DAC support to generate simulation results as an analog output signal. The definition of the clock and the discrete input

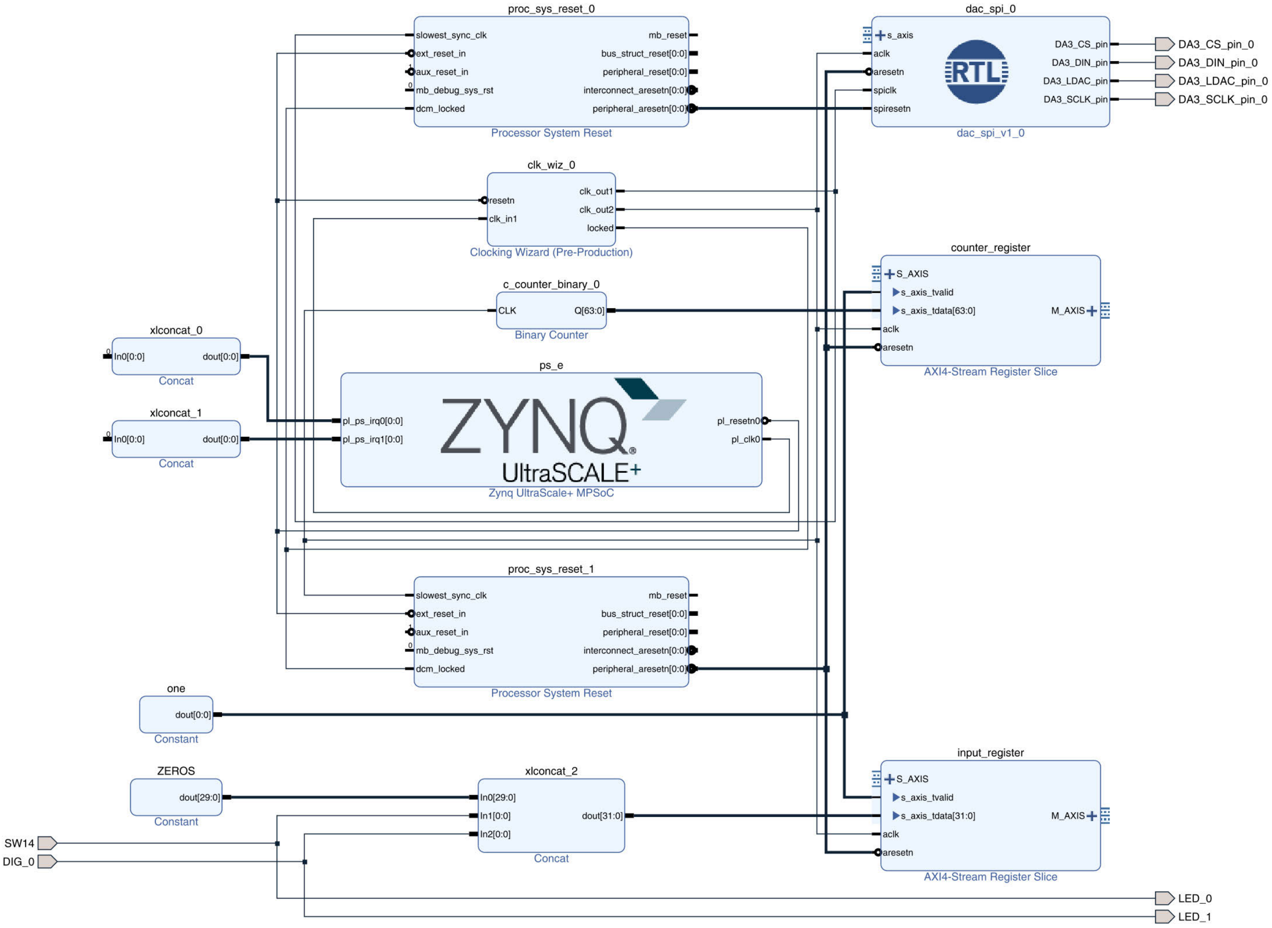
proc_sys_reset_0
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
dac_spi_0
s_axis
aclk
aresetn
spiclk
spiresetn
RTL
DA3_CS_pin
DA3_DIN_pin
DA3_LDAC_pin
DA3_SCLK_pin
dac_spi_v1_0
DA3_CS_pin_0
DA3_DIN_pin_0
DA3_LDAC_pin_0
DA3_SCLK_pin_0
clk_wiz_0
resetn
clk_in1
clk_out1
clk_out2
locked
Clocking Wizard (Pre-Production)
counter_register
S_AXIS
s_axis_tvalid
s_axis_tdata[63:0]
aclk
aresetn
M_AXIS
AXI4-Stream Register Slice
c_counter_binary_0
CLK
Q[63:0]
Binary Counter
xlconcat_0
In0[0:0]
dout[0:0]
Concat
xlconcat_1
In0[0:0]
dout[0:0]
Concat
ps_e
pl_ps_irq0[0:0]
pl_ps_irq1[0:0]
ZYNQ
UltraSCALE+
pl_resetn0
pl_clk0
Zynq UltraScale+ MPSoC
proc_sys_reset_1
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
one
dout[0:0]
Constant
ZEROS
dout[29:0]
Constant
xlconcat_2
In0[29:0]
In1[0:0]
In2[0:0]
dout[31:0]
Concat
input_register
S_AXIS
s_axis_tvalid
s_axis_tdata[31:0]
aclk
aresetn
M_AXIS
AXI4-Stream Register Slice
SW14
DIG_0
LED_0
LED_1

FIGURE 6. Block diagram of the Xilinx Vivado hardware platform for HiLS applications.

register was quite simple and was made directly in the block structure (Fig. 6), however, the hardware Digital-to-Analog (DAC) converter (Digilent Pmod DA3 with 16 bit Analog Devices AD5541A) was implemented using the VHDL language, and thus was more complicated. The defined registers were connected to the part described in the software language C by defining the appropriate system port (*sys_port*) in Xilinx SDSoC and the hardware acceleration function was integrated (Fig. 7). Appropriate pragmas have been defined [41] to limit the use and length of the default very long FIFOs used for the I/O registers by Xilinx SDSoC during integration. Simple functions for emptying FIFO queries and real-time synchronization were written in the C language and using the HLS technique (Xilinx SDSoC) were integrated with the final hardware version of the *Amikro4* simulation program.

IV. SIMULATION AND EXPERIMENTAL RESULTS

This chapter presents software and hardware simulations and experimental results of the face milling process of the sample workpiece.

A. MACHINED WORKPIECE

The algorithm implemented in FPGA simulated the milling process of the real workpiece, which was made of EN-GJS-400-15 cast iron (Fig. 8b) and clamped on the table of the MIKROMAT 20V portal milling center. The FEM model (Fig. 8a) of the workpiece was properly tuned to obtain a good correlation with the experimentally obtained modal model. The milled surface is marked in Fig. 8b. The first five modal modes were examined during the simulation.

B. MODEL IDENTIFICATION

Modal tests were carried out for the workpiece shown in Fig. 8. The 6 vibration modes were identified by the pLSCF-D method [42]. The FEM model presented in Fig. 8a was appropriately fixed (to represent the holder used during actual milling operations on the milling center) with twelve finite elements representing springs (each with six degrees of freedom) to obtain satisfactory correlation of natural frequencies and modes. The values of the Modal Assurance Criterion (MAC) [43] are presented in Table 2. In further simulations the modal model, reduced to five forms with the lowest frequencies, was used.

C. MILLING PARAMETERS

A SECO face milling cutter with a diameter of 63 mm and 6 edges was used. The standard (same as for real process) parameters used to simulate the milling process were: $n = 1112$ rev/min, $v_f = 1112$ mm/min, $a_p = 1$ mm.

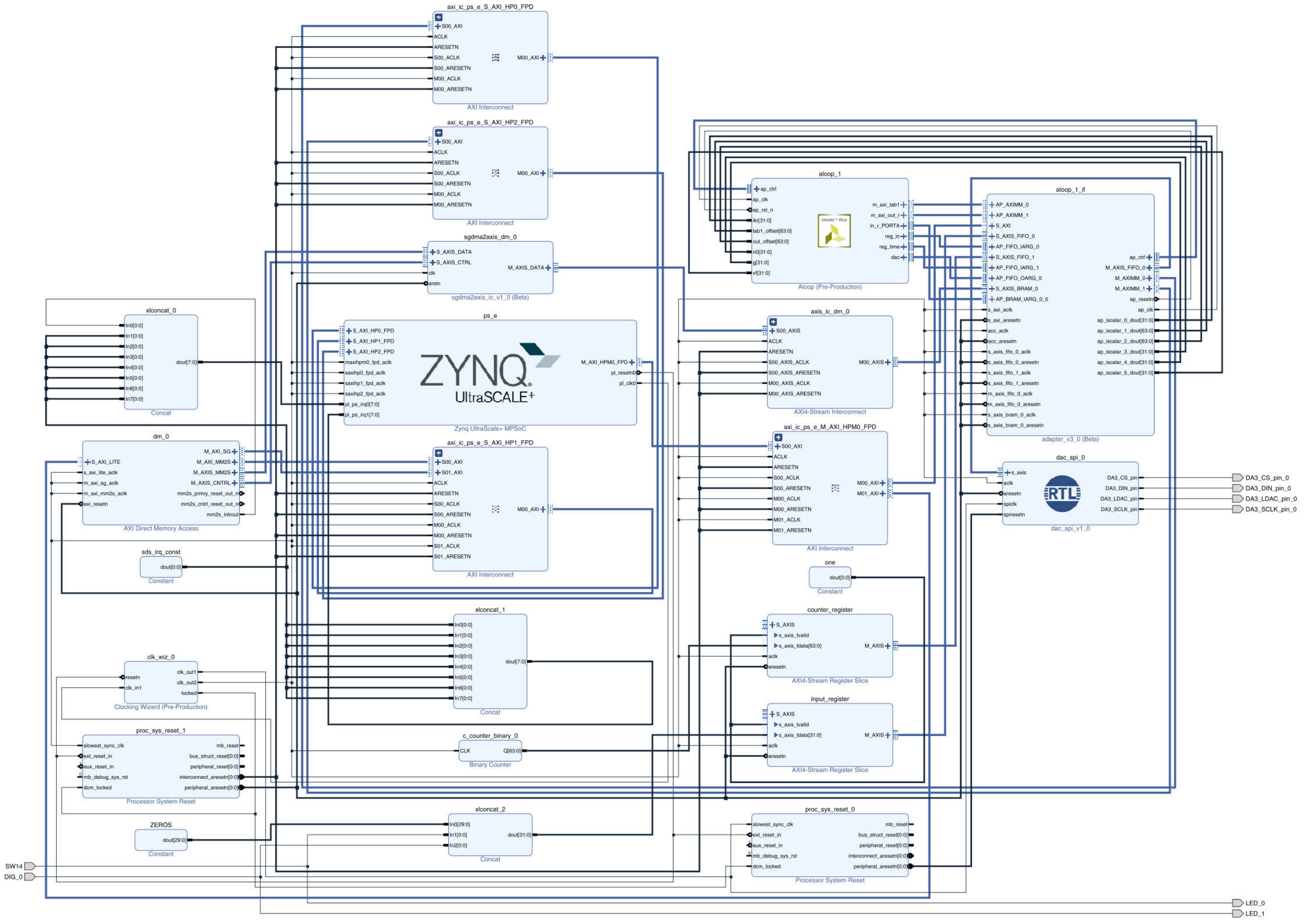
axi_ic_ps_e_S_AXI_HP0_FPD
AXI Interconnect
axi_ic_ps_e_S_AXI_HP2_FPD
AXI Interconnect
sgdma2axis_dm_0
sgdma2axis_ic_v1_0 (Beta)
xlconcat_0
Concat
ps_e
ZYNQ UltraSCALE+
Zynq UltraScale+ MPSoC
dm_0
AXI Direct Memory Access
sds_irq_const
Constant
axi_ic_ps_e_S_AXI_HP1_FPD
AXI Interconnect
aloop_1
Aloop (Pre-Production)
aloop_1_if
adapter_v3_0 (Beta)
axis_ic_dm_0
AXI4-Stream Interconnect
axi_ic_ps_e_M_AXI_HPM0_FPD
AXI Interconnect
dac_spi_0
dac_spi_v1_0
one
Constant
counter_register
AXI4-Stream Register Slice
input_register
AXI4-Stream Register Slice
xlconcat_1
Concat
clk_wiz_0
Clocking Wizard (Pre-Production)
proc_sys_reset_1
Processor System Reset
c_counter_binary_0
Binary Counter
ZEROS
Constant
xlconcat_2
Concat
proc_sys_reset_0
Processor System Reset
SW14
DIG_0
DA3_CS_pin_0
DA3_DIN_pin_0
DA3_LDAC_pin_0
DA3_SCLK_pin_0
LED_0
LED_1

FIGURE 7. Scheme of integration with Xilinx SDSoC hardware to simulate the milling process.

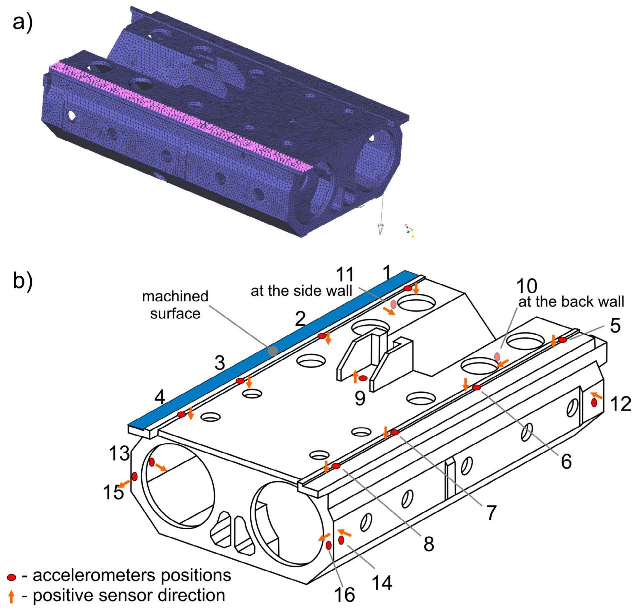
a)
b)
machined surface
at the side wall
at the back wall
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
● - accelerometers positions
↑ - positive sensor direction

FIGURE 8. Tested workpiece: a) FEM model of the real workpiece, b) schematics of the workpiece with marked sensor mounting points.

Additionally, artificially modified parameters of the milling process were also tested, i.e. $n = 1212$ rev/min, $v_f = 1512$ mm/min and $a_p = 1$ mm.

TABLE 2. Modal Assurance Criterion (MAC) values between FEM model modes and modal test modes.

pLSCF-D FEM	185.0 Hz	211.3 Hz	242.4 Hz	435.7 Hz	585.0 Hz	631.2 Hz
184.6 Hz	**0.96**	0.05	0.08	0.01	0.09	0
211.4 Hz	0.01	**0.98**	0.24	0	0.01	0.03
242.2 Hz	0.08	0.18	**0.95**	0.03	0.05	0.01
295.5 Hz	0.20	0	0	0	0.03	0
434.3 Hz	0.13	0.08	0.03	0.01	0.18	0
434.4 Hz	0.08	0	0	**0.94**	0.06	0.01
571.5 Hz	0.05	0.03	0.03	0.05	**0.87**	0.01
630.2 Hz	0	0.03	0	0.01	0.01	**0.94**

D. SOFTWARE SIMULATION RESULTS

The results of software simulations (i.e. performed on a PC) of the displacements of the workpiece surface in the direction of the x_3 axis (Fig. 1), for the standard parameters of the milling process, are shown in Fig. 9a, and for the modified parameters in Fig. 9b.

E. HARDWARE SIMULATION RESULTS

The same simulation program compiled with different compiler optimization options may produce slightly different results (up to 5×10^{-6} mm) due to underflow and rounding

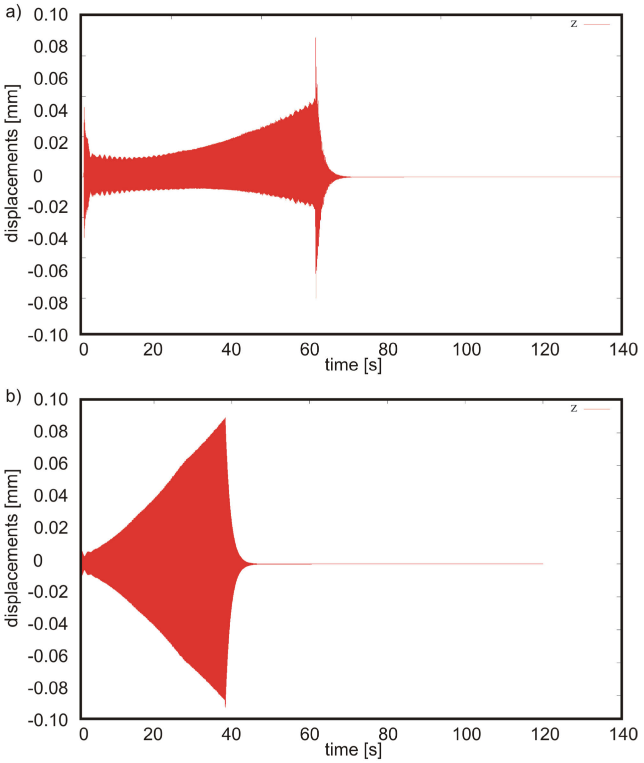

FIGURE 9. **Reference – results of software simulations – displacements along the x3 axis: a) standard parameters, b) modified parameters.**

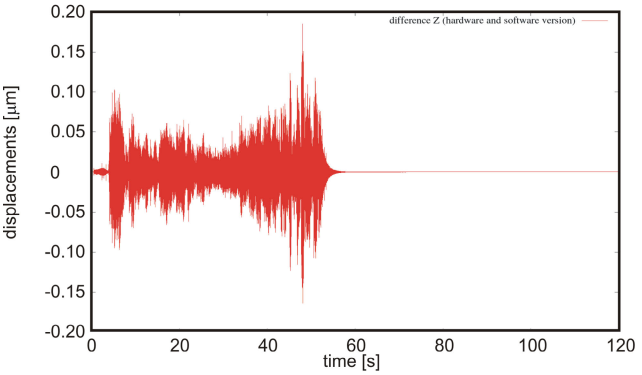

FIGURE 10. **The difference between the results of software and hardware simulations – displacements along the x_3 axis.**

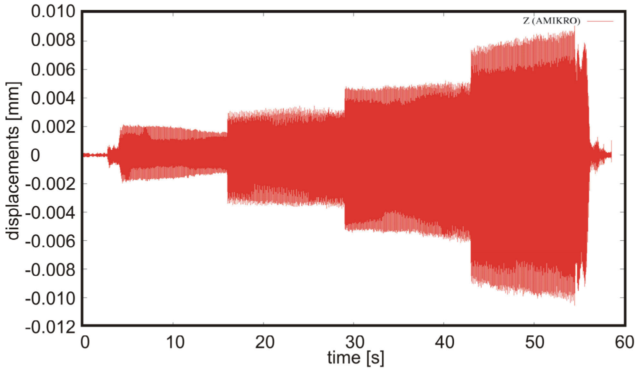

FIGURE 11. **Experimental results - displacements of the workpiece surface along the x_3 axis at measuring points from 4 to 1 (see Fig 8).**

errors when calculating using floating point variables. Such insignificant (compared to the values presented in Fig. 9a) differences were observed between software and hardware (FPGA) implementations of the *Amikro4* simulation program. This version of the hardware implemented in the *Amikro4* simulations performed the calculations in RT in all tests, however the number of theoretically required FPGA cycles was slightly higher than the RT requirements. If all cutting edges of a milling tool are cutting simultaneously in the same time step, the calculation will take slightly longer than the minimum time step Δt (42 μs) because FPGA operated at a relatively low frequency (100 MHz) compared to modern CPUs. In this case, a problem was identified with the *mawie* function algorithm that finds the active node in the FEM model (Table 1). The optimized version of this algorithm required 593 FPGA cycles. In order to be sure that the computation would be performed at RT in every possible state, the *mawie* function had to be rewritten to reduce the required number of cycles to compute it. Instead of finding (in each simulation step) which node is closest to the tool, the hardware implementation of the *mavie* function used a pre-computed map (which only requires a few FPGA cycles). Both functions give slightly different results at the corners of the map cells. The differences in the results of *Amikro4* with the *mawie* function mapped version are shown in Fig. 10. It is still possible to improve the results by introducing a map with a higher density in the *mawie* function, but for HiLS both the quality (stability) and the quantity (the differences are still small compared to simulation results) of the presented solution was considered satisfactory.

F. EXPERIMENTAL RESULTS FOR THE REAL MILLING PROCESS

The results (measured with accelerometers from 4 to 1 in Fig. 8b) of the real milling process of a workpiece with standard parameters (chapter IV.C) are shown in Fig. 11. It should be noted that the figure shows the displacements observed at the closest measurement points, fixed to the workpiece surface (i.e. the workpiece-tool contact point moves near and over a given sensor). Meanwhile, the simulation results presented in Fig. 9 show displacements at the point of contact between the tool and the workpiece (i.e. measurement point moves with the tool). For example, the time interval from 5 to 16 seconds was calculated over the period from the start of the milling operation (excluding the time while the tool entered the material) to the time while the tool was in the center position between sensors 4 and 3 (see Fig. 8 for sensor positions). The comparison of the real simulation results with the results of the *Amikro4* FPGA RT simulation in the frequency domain is shown in Fig. 12. The different amplitudes are mainly the result of different assumed parameters in equations (1)-(3)

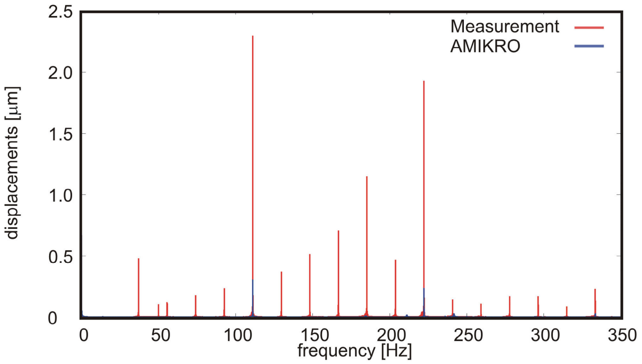

FIGURE 12. The comparison of the real simulations results (point 2) and the Amikro4 FPGA RT simulations results in frequency domain – displacements along x3 axis.

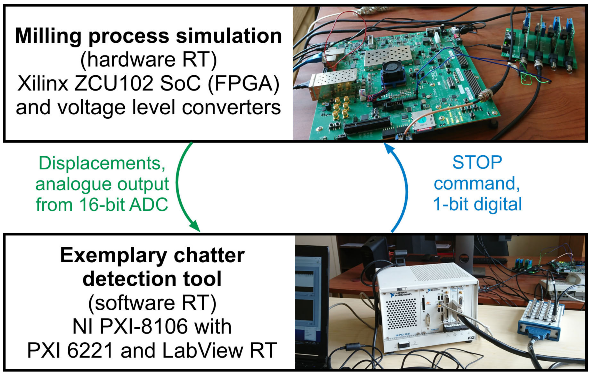

FIGURE 13. Example HiLS application with hardware RT (FPGA).

and the introduction of a reduced model (only five modal workpiece modes were used). The two main harmonics are present in the simulation results (Fig. 12), but many other harmonics are missing. During the simulation, a simplified model of the tool was assumed (all cutting edges are exactly the same), and some effects (appearing in the experiment) were ignored (for example, the effects of balancing and tool bending [44]).

It should be emphasized that both the most important components of the vibration spectrum, as well as the general outline of the displacement and the evolution of vibrations obtained from RT simulations made on FPGA are consistent with the experimental results.

G. RT EXPERIMENTAL VALIDATION

It was assumed that the hardware implementation of the RT milling process simulation would be part of the HiLS system designed to develop a tool for detecting self-excited *chatter* vibrations. This is only an example application, hence a simplification of the HiLS system - the relative tool-workpiece vibrations (displacements) are ‘‘published’’ from the RT as an analog signal generated by a 16-bit D/A converter. It is possible to generate various signals resulting from RT simulation, such as tool vibrations (not relative), but to simplify the comparisons, the original form of the results from the *Amikro4* program has not been modified. The FPGA monitors one digital input signal which is a command (high signal level) to stop the milling process. In addition, simulation results are saved in FPGA memory and then copied to CPU memory after simulation is complete, where they are finally saved as text files for further, off-line, non-RT analysis.

The *chatter* detection tool has been simplified (and it is not the topic of this article). Appropriate software was implemented in LabView Real Time at the NI PXI-8106 RT controller system with a 16-bit DAQ PXI 6221 card. The system monitors the vibration level, i.e. the signal generated on-line by the FPGA chip performing RT simulations. This signal mocks the signal that can be obtained from some vibration sensor, for example an accelerometer or a proximity probe. When the monitored signal exceeds any selected value, the PXI generates a stop signal. The milling process is then stopped (the feed rate in the simulated process is set to $v_f = 0\,\mathrm{m/s}$). A schematic description of the stand is presented in Fig. 13. For the conversion of input and output signals of the FPGA (level 0 - 3.3 V) to a voltage level (-10 – 10 V), voltage converters with low propagation time (1 us) were used between the FPGA and the PXI system.

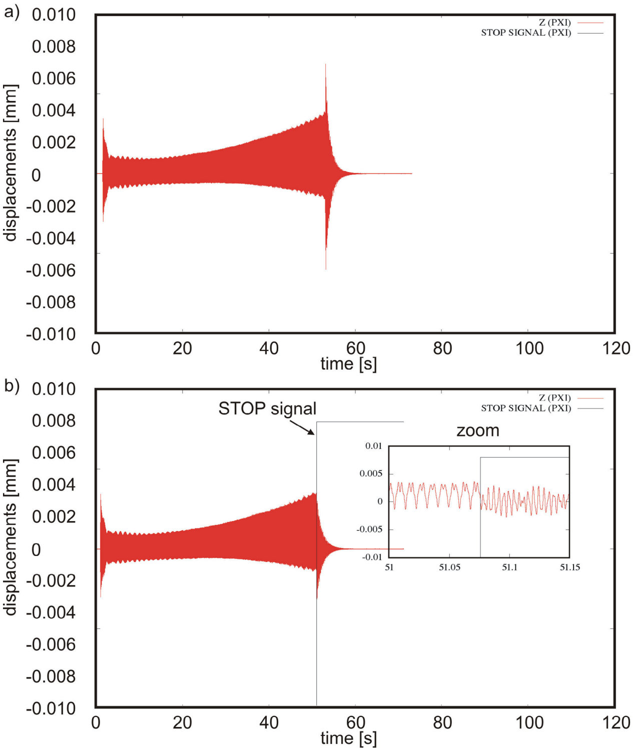

FIGURE 14. Hardware simulations results with standard parameters – displacements along the x_3 axis: a) uninterrupted, b) interrupted – a stop signal is generated.

The results of the hardware simulations were appropriately scaled and shown in Fig. 14 for the simulation performed with standard milling parameters and with modified milling parameters in Fig. 15. The observed analog signals that were

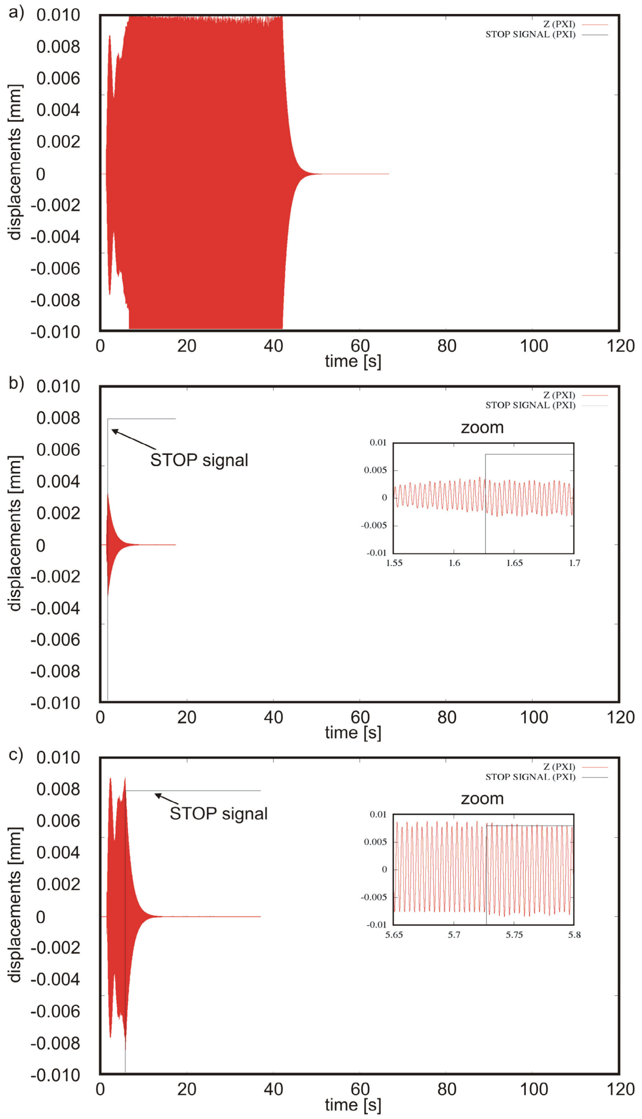

FIGURE 15. **Hardware simulations results with modified parameters – displacements along the x_3 axis: a) uninterrupted, b) interrupted (low trigger value) – stop signal generated, c) interrupted (high trigger value) – stop signal generated.**

generated in RT are similar to the results obtained with the software, off-line version simulation: Fig.14a is similar to Fig.9a and Fig.15a is similar to Fig.9b, but the last two have a limited range because the analog signals are limited (maximum displacement is ± 0.01 mm) and thus observed signals are saturated. The result of the hardware simulation presented in Fig. 15a is unstable, so some aliasing effects are observed (even though the sampling time of the LabView-based acquisition system is relatively small (0.000025 s)), because the maximum values of the generated signals are not synchronized in time with the sampling period.

The trigger values for the STOP signal have been set arbitrarily to the maximum allowed value of the observed displacement in a LabView based RT system. The STOP signals were generated with a delay of one sampling period, i.e. 25 us. The STOP signal effects could be observed in Figs. 14b, 15b and 15c. The hardware simulated system reacted in real time to the given STOP signals during the time integration step, i.e. 42 μs. To make the simulations more realistic, the response time can be increased in this case.

In real applications, much more sophisticated techniques and algorithms for *chatter* detection can be used [45]–[48]. The example in the paper only confirms the validity of the FPGA-based RT simulation application concept.

V. SUMMARY AND CONCLUSION

The presented RT hardware simulation results allow for the conclusion that it is possible to create RT simulations of a complex process, e.g. dynamic milling systems using FPGA and HLS techniques. The possibility of very sophisticated time synchronization along with very short response delays and very little jitter make hardware simulations better than the typical software RT approach. It is also possible to implement more complex control tasks that run in parallel, especially when executing HiLS. The FPGA-based approach is characterized by higher computing power and enormous flexibility when it comes to input and output operations that can be implemented deterministically in RT. In this example project, you could define various input and output signals, which can be analog or digital. Thus, it is possible, for example, to study the simulations of the influence of active damping systems on the simulated milling process. The hardware functions are based on procedures written in high-level C programming language, which makes it easy to introduce changes to the presented algorithm. The approach presented in this paper can be used to simulate the dynamics of various mechatronic systems.

APENDIX

Data must be copied to and from the accelerator, and accelerator operations must be synchronized. The Xilinx SDSoC software tries to estimate this time, but to investigate this problem further, a simple test function *add* was written in the C language. The *add* function returns the sum of two input floating point numbers:

```
float add(float a, float b)
{
  return a+b;
}
```

The *add* function has been called from a simple test program loop one million times:

```
float a = 1.f, b = 1.f;
for(;a<1000000.f;)
{
  a = add(a,b);
}
```

The program containing this loop was compiled with the Xilinx SDSoC utility. The standard optimization options that are used by the Xilinx SDSoC compilers (*clang* and *gcc*) are enabled. The test program execution time is shown in Table 3. It did not matter whether the operating system (Linux) was

TABLE 3. Simple test program execution time.

Variant	Linux CPU	Linux with FPGA	Baremetal CPU	Baremetal with FPGA
Time [s]	0.00	1.94	0.00	1.84

used or not (Baremetal), the execution time was always significantly shorter compared to the version with the hardware accelerated *add* function.

Based on the results presented in Tables 1 and 3, the final design decisions were made for the RT version of the *Amikro4* simulation program.

The execution time for the Linux version of *Amikro4* was only slightly longer than for the *Baremetal* version, which is a much less convenient development environment. All input, output, and network operations would probably run faster and more reliably on Linux. Xilinx SDSoC provides three major software platforms: Linux, FreeRTOS, and Baremetal. The standard version of Linux, unlike FreeRTOS and Baremetal, could not serve as the basis for the RT program, but could be a very good platform for performing all the necessary operations, which does not have to meet RT expectations.

REFERENCES

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[2] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Des. Test. Comput.*, vol. 26, no. 4, pp. 18–25, Jul. 2009.

[3] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[4] C. Dufour, S. Abourida, and J. Belanger, "Real-time simulation of permanent magnet motor drive on FPGA chip for high-bandwidth controller tests and validation," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jul. 2006, pp. 4581–4586.

[5] S. Usenmez, R. A. Dilan, M. Dolen, and A. B. Koku, "Real-time hardware-in-the-loop simulation of electrical machine systems using FPGAs," in *Proc. Int. Conf. Electr. Mach. Syst.*, Tokyo, Japan, 2009, pp. 1–6.

[6] E. Duman, H. Can, and E. Akin, "FPGA based Hardware-in-the-Loop (HIL) simulation of induction machine model," in *Proc. 16th Int. Power Electron. Motion Control Conf. Expo.*, Sep. 2014, pp. 616–621.

[7] K. J. Kaliński and C. Buchholz, "Mechatronic design of strongly nonlinear systems on a basis of three wheeled mobile platform," *Mech. Syst. Signal Process.*, vols. 52–53, pp. 700–721, Feb. 2015.

[8] K. J. Kaliński and M. A. Galewski, "Vibration surveillance supported by Hardware-In-the-Loop simulation in milling flexible workpieces," *Mechatronics*, vol. 24, no. 8, pp. 1071–1082, Dec. 2014.

[9] F. Naets, T. Tamarozzi, G. H. K. Heirman, and W. Desmet, "Real-time flexible multibody simulation with global modal parameterization," *Multibody Syst. Dyn.*, vol. 27, no. 3, pp. 267–284, Mar. 2012.

[10] J. Garcia de Jalon, E. Bayo, *Kinematic and Dynamic Simulation of Multibody Systems the Real-Time Challenge*. New York, NY, USA: Springer-Verlag, 1994.

[11] J. H. Lala and R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proc. IEEE*, vol. 82, no. 1, pp. 25–40, Dec. 1994.

[12] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes," in *Proc. 24th IEEE Real-Time Syst. Symp.*, Cancun, MN, USA, 2003, pp. 396–407.

[13] R. Cortesao, Jaehe, and O. Khatib, "Real-time adaptive control for haptic telemanipulation with Kalman active observers," *IEEE Trans. Robot.*, vol. 22, no. 5, pp. 987–999, Oct. 2006.

[14] B. Wittenmark, J. Nilsson, and M. Torngren, "Timing problems in real-time control systems," in *Proc. ACC*, Seattle, WA, USA, vol. 3, 1995, pp. 2000–2004.

[15] A. Cervin, B. Lincoln, J. Eker, K-E. Årzén, G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proc. 10th Int. Conf. Real-Time Embedded Comput. Syst. Appl.*, 2004, pp. 1–10.

[16] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, "Jitter compensation for real-time control systems," in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, London, U.K., 2001, pp. 39–48.

[17] A. Schmitt and R. Seifried, "Comparison of various models and integration method for real-time simulation of complex vehicle models with structural flexibility," in *Proc. Int. Conf. Noise Vib. Eng.*, P. Sas, D. Moens, and A. van de Walle, Eds., 2016, pp. 1–5.

[18] D. H. Bailey, R. Barrio, and J. M. Borwein, "High-precision computation: Mathematical physics and dynamics," *Appl. Math. Comput.*, vol. 218, no. 20, pp. 10106–10121, Jun. 2012.

[19] R. W. Robey, J. M. Robey, and R. Aulwes, "In search of numerical consistency in parallel programming," *Parallel Comput.*, vol. 37, nos. 4–5, pp. 217–229, Apr. 2011.

[20] G. Cockerham and M. Cole, "Stick-slip stability by analogue simulation," *Wear*, vol. 36, no. 2, pp. 189–198, Feb. 1976.

[21] R. S. Fayose, "Development of analogue computer for the simulation of linear circuits and systems," *Int. J. Res. Appl. Sci. Eng. Tech.*, vol. 3, pp. 97–106, Dec. 2015.

[22] J. J. Rodríguez-Andina, M. D. Valdés-Peña, and M. J. Moure, "Advanced features and industrial applications of FPGAs—A review," *IEEE Trans. Ind. Informat.*, vol. 11, pp. 853–864, Jul. 2015.

[23] Z. Yao, D. Mei, and Z. Chen, "On-line chatter detection and identification based on wavelet and support vector machine," *J. Mater. Process. Technol.*, vol. 210, no. 5, pp. 713–719, Mar. 2010.

[24] Y. Fu, Y. Zhang, H. Zhou, D. Li, H. Liu, H. Qiao, and X. Wang, "Timely online chatter detection in end milling process," *Mech. Syst. Signal Process.*, vol. 75, pp. 668–688, Jun. 2016.

[25] E. Kuljanic, G. Totis, and M. Sortino, "Development of an intelligent multisensor chatter detection system in milling," *Mech. Syst. Signal Process.*, vol. 23, no. 5, pp. 1704–1718, Jul. 2009.

[26] M. Nouari, G. List, and F. Girot, "Wear mechanisms in dry machining of aluminium alloys," *Int. J. Mech. Prod. Syst. Eng*, vol. 4, pp. 22–29, Dec. 2003.

[27] G. Quintana and J. Ciurana, "Chatter in machining processes: A review," *Int. J. Mach. Tools Manuf.*, vol. 51, no. 5, pp. 363–376, May 2011.

[28] Y. Shu, H. Li, and Q. Wu, "Expansion application of dSPACE for HILS," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jun. 2008, pp. 2231–2235.

[29] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control Systems—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.

[30] K. J. Kaliński, M. A. Galewski, M. R. Mazur, and N. Morawska, "A technique of experiment aided virtual prototyping to obtain the best spindle speed during face milling of large-size structures," *Meccanica*, vol. 5, pp. 1–6, Jul. 2020.

[31] K. J. Kaliński, *A Surveillance of Dynamic Processes in Mechanical Systems*. Gdansk, Poland: Gdansk Univ. Technol., 2012 (in Polish: Nadzorowanie procesów dynamicznych w układach mechanicznych).

[32] K. J. Kaliński, "The finite element method application to linear closed loop steady system vibration analysis," *Int. J. Mech. Sci.*, vol. 39, no. 3, pp. 315–330, Mar. 1997.

[33] J. Tomkow, *Vibrostability of Machine Tools*. Warsaw, Poland: The Scientific and Technical, 1997.

[34] K. J. Kaliński and M. A. Galewski, "Optimal spindle speed determination for vibration reduction during ball-end milling of flexible details," *Int. J. Mach. Tools Manuf.*, vol. 92, pp. 19–30, 2015.

[35] L. Uriarte, M. Zatarain, D. Axinte, J. Yagáe-Fabra, S. Ihlenfeldt, J. Eguia, and A. Olarra, "Machine tools for large parts," *CIRP Ann.*, vol. 62, no. 2, pp. 731–750, 2013.

[36] A. D. Sarhan, S. R. Besharaty, and J. M. Akbaria Hamdi, "Improvement on a CNC gantry machine structure design for higher machining speed capability," *Int. J. Mech., Aerosp., Ind., Mech. Manuf. Eng.*, vol. 9, pp. 534–538, Dec. 2015.

[37] M. Chodnicki, K. J. Kaliński, and M. A. Galewski, "Vibration surveillance during milling of flexible details with the use of active optimal control," *J. Low Freq. Noise Vib. Act. Control*, vol. 32, pp. 145–156, 2013.

[38] S. Kestur, J. D. Davis, and O. Williams, "BLAS comparison on FPGA, CPU and GPU," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2010, pp. 288–293.

[39] *ZCU102 Evaluation Board User Guide, UG1182 (v1.4)*, Xilinx, San Jose, CA, USA, 2018. [Online]. Available: www.xilinx.com

[40] *Zynq UltraScale+ MPSoC Data Sheet: Overview, DS891 (v1.7)*, Xilinx, San Jose, CA, USA, 2018. [Online]. Available: www.xilinx.com

[41] *Vivado HLS Optimization Methodology Guide, UG1270 (v2017.4)*, Xilinx, San Jose, CA, USA, 2017. [Online]. Available: www.xilinx.com

[42] P. Guillaume, P. Verboven, S. Vanlanduit, H. Van der Auweraer, and B. Peeters, "A poly-reference implementation of the least-squares complex frequency-domain estimator," in *Proc. IMAC*, 2003, pp. 1–9.

[43] R. J. Allemang, "The modal assurance criterion—Twenty years of use and abuse," *Sound Vib.*, vol. 15, pp. 14–21, Aug. 2003.

[44] G. Totis, P. Albertelli, M. Torta, M. Sortino, and M. Monno, "Upgraded stability analysis of milling operations by means of advanced modeling of tooling system bending," *Int. J. Mach. Tools Manuf.*, vol. 113, pp. 19–34, Feb. 2017.

[45] P. Albertelli, L. Braghieri, M. Torta, and M. Monno, "Development of a generalized chatter detection methodology for variable speed machining," *Mech. Syst. Signal Process.*, vol. 123, pp. 26–42, May 2019.

[46] Y.-C. Yao, Y.-H. Chen, C.-H. Liu, and W.-P. Shih, "Real-time chatter detection and automatic suppression for intelligent spindles based on wavelet packet energy entropy and local outlier factor algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 103, nos. 1–4, pp. 297–309, Jul. 2019.

[47] K. Yang, G. Wang, Y. Dong, Q. Zhang, and L. Sang, "Early chatter identification based on an optimized variational mode decomposition," *Mech. Syst. Signal Process.*, vol. 115, pp. 238–254, Jan. 2019.

[48] H. Caliskan, Z. M. Kilic, and Y. Altintas, "On-line energy-based milling chatter detection," *J. Manuf. Sci. Eng.*, vol. 140, no. 11, pp. 1–12, Nov. 2018.

MICHAŁ R. MAZUR received the M.Sc. and Ph.D. degrees in machine construction and operation from the Gdańsk University of Technology (GUT), Gdańsk, Poland, in 2005 and 2010, respectively.

Since 2010, he has been an Associate Professor with the Faculty of Mechanical Engineering, GUT. He is the author or the coauthor of over 38 articles. His research interests include problems of robotics, especially mobile robots, mechanical vibration, modal analysis, finite-element methods, theoretical and experimental modal analysis, mechatronics, signal processing, and real-time systems and programming.

MAREK A. GALEWSKI received the M.Sc. degree in automatic control and robotics, and the Ph.D. degree in machine construction and operation from the Gdańsk University of Technology (GUT), Gdańsk, Poland, in 2003 and 2007, respectively, and the D.Sc. degree for his works on mechatronic solutions for vibration reduction in milling with slender tools, in 2016.

He has been an Associate Professor with the Faculty of Mechanical Engineering, GUT, since 2016. He is the author or the coauthor of over 65 articles, a book, and two handbooks. His research interests include problems of vibration reduction in dynamical objects, especially during milling processes, experimental modal analysis and modal identification, signal processing, and vibrations measurement systems integration and programming (mainly in C, LabView, and LabWindows), including real-time systems.

KRZYSZTOF J. KALIŃSKI received the M.Sc. degree from the Faculty of Production Engineering, Gdańsk University of Technology (GUT), Gdańsk, Poland, in 1980, the Ph.D. degree from the Faculty of Machine Building, GUT, in 1988, and the D.Sc. degree from the Faculty of Mechanical Engineering, GUT, in 2002. He received a Professor's title, in 2013.

He is currently a Full Professor with the Faculty of Mechanical Engineering, GUT. He is the author or the coauthor of over 250 publications. He was invited as a Visiting Professor at the Ecole Nationale d'Ingénieurs de Metz, France, and Northwestern University, Evanston, USA. He gave lectures at Basrah University, Iraq, and London University, U.K. His research interests include theoretical and applied mechanics, machine dynamics, vibration engineering, dynamics of machine tools and production processes, robotics and automation, finite-element methods, theoretical and experimental modal analysis, mechatronics, and biomechanics of a mandible.

• • •