energies

MDPI

Article

Flexible Matrix of Controllers for Real Time Parallel Control

Patryk Chaber * and Andrzej Wojtulewicz

Institute of Control and Computation Engineering, Faculty of Electronics and Information Technology, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland; andrzej.wojtulewicz@pw.edu.pl

* Correspondence: patryk.chaber@pw.edu.pl

Abstract: This work aims to develop a novel system, including software and hardware, to perform independent control tasks in a genuine parallel manner. Currently, to control processes with various sampling periods, distributed control systems are most commonly utilized. The main goal of this system is to propose an alternative solution, which allows simultaneous control of both fast and slow processes. The presented approach utilizes FPGA (Field Programmable Gate Array) with Nios II processor (Intel Soft Processor Series) to implement and maintain instances of independent controllers. Instances can implement FDMC (Fast Dynamic Matrix Control) and PID (Proportional-Integral-Derivative) control algorithms with various sampling times. The FPGA-based design allows for true independence of controllers' execution both from one another and the managing processor. Also, pure parallel execution allows for implementing slow and fast controllers in the same device. The complete flexible system with a matrix of controllers working in parallel in real-time was tested with both simulated and actual control processes (servomotor), yielding the same results as fully simulated experiments.

Keywords: Model Predictive Control; Fast Dynamic Matrix Controller; field programmable gate array; servomotor; matrix of controllers

Citation: Chaber, P.; Wojtulewicz, A. Flexible Matrix of Controllers for Real Time Parallel Control. *Energies* **2022**, *15*, 1833. https://doi.org/10.3390/en15051833

Academic Editor: Elyas Rakhshani

Received: 28 January 2022
Accepted: 26 February 2022
Published: 2 March 2022

1. Introduction

As the Model Predictive Control (MPC) [1,2] approach is expanding from industrial processes with long sampling periods to processes with sampling periods of single milliseconds [3–5] or even microseconds [6,7], there is a demand for controllers executing and calculating new control signals on time. There are research papers concerning the process of solving control problems using fast solvers [8–10], as well as tuning MPC to reach lower control error, which further influences control time [11]. On the other hand, some papers aim for the robustness of the control algorithms despite uncertainties [12,13], unknown disturbances [14] or unknown model of the controlled process [15]. This phenomenon is also coupled with a still-growing interest in MPC in embedded systems, including microcontrollers [6,16] and Field Programmable Gate Arrays [17–19], which, despite having much more limited resources, can be utilized for controlling complex, nonlinear processes [20,21].

The application of FPGAs is very wide. The general computational capabilities are presented in the article [22]. Based on numerous examples, it can be concluded that FPGAs allow much better performance than very fast classical processors [23]. In many cases, it is possible to achieve more than 100 times algorithm speedup by using process decomposition and computation parallelization [24,25]. The use of FPGAs in the multiplication of integer with notation matrices was presented in the work of [26], where the optimal use of multiplication circuits reduced the use of logic resources and increased the speed of computation. In the field of pseudorandom number generators, attention can be drawn to the dissertation of [27], where the applicability of FPGAs is described in detail. FPGAs are frequently used in cryptography [28]. Also, applications in space vehicles find many opportunities from the point of view of computation and their speed using low clock frequencies [29,30]. The

implementation of a numerical predictive control algorithm based on a linear model in the state space is discussed in the paper [31]. A hardware block solver for the quadratic programming task (active constraint method) was used. The algorithm was applied to a synchronization problem between two motors. An interesting item [32] deals with the application of a numerical predictive control algorithm, based on linear models in the state space, to an embedded system (for a simulated Cessna aircraft). Parallel operations were used to reduce computation time. Optimization of operations in solving the quadratic programming task was performed. Results are presented showing that the parallel implementation allowed practically a twofold reduction in computation time. Solutions for the military industry are discussed in the work of [33]. Although not common, there are applications of FPGAs in automation of industrial processes. Examples of applications are presented in the paper describing CNC control [34] or the paper [35], which describes a proprietary PLC implemented using an FPGA chip.

The control of a DC motor with Proportional-Integrate-Derivative (PID) and Internal Model Control algorithms, realized in hardware on an FPGA chip, is presented in the paper [36]. An example of use for industrial robot applications can be found in work [37]. FPGA based predictive controllers are also utilized for winch servomotors [38] and power converters [23,39,40]. There are applications of FPGAs in medicine, such as in computed tomography imaging tasks [41,42]. FPGAs have dedicated versions for military applications, where, as an example from Xilinx, encrypted radio links can be used. Modern unmanned aerial vehicles use FPGA technology for trajectory planning [43]. FPGAs can also be used to control the electronic keying circuitry of capacitor batteries [44]. An interesting and giving very illustrative comparison is the work of [45], which describes the implementation of a numerical Dynamic Matrix Controller (DMC) algorithm in an FPGA.

Also noteworthy are the FPGA-based applications that have been used for distributed systems [46], as well as real-time parallel systems [47], or distributed systems with Petri nets [48]. Also worth mentioning are control and communication systems for distributed power systems for space applications [49], or even high-speed multichannel distributed data acquisition systems [50] used in high energy physics experiments.

The proposed system structure represents an original approach that allows simultaneous control of multiple high-speed processes requiring very short sampling periods, where the sampling periods can be different for each control loop. The control loops are fully realized in hardware as dedicated digital circuits. In addition to selecting the needed sampling period, it is also possible to define which analog input of the real process is to be measured by a given controller. In addition to the real processes, it is also possible to connect a simulation block, which in particular can reproduce the behavior of the real process with the accuracy of the proposed model, where first order inertia is used in the considered case. The output of the controller—that is, the control value of the process—can be directed to a number of ports that correspond to the physical output of the PWM, the analog output or the simulation block. With this approach, it is possible to realize a comparison of the control result of a real process by one controller with a second controller, which can be used to control a simulated process based on the real process model. This will allow to study the differences between simulation and real object behavior, where it will be possible, for example, to monitor or diagnose the real controller. In addition to parallel connection, it is also possible to cascade connection of up to three regulators, where there will be inertia in the intermediate path, which will allow filtering the signal from individual stages of the regulators. The considered cascade will follow the problem of simultaneous regulation of slow and fast processes. This solution will also be characterized by universality, which is lacking in the cited publications, because the control algorithms will not be limited to a specific control process and each of them has the possibility of dynamically variable parameters from the level of the managing processor. There are many possible applications of the proposed system. The main targets are complex processes in terms of a structure yet can be decomposed into many simpler control loops, even if requiring different sampling control intervention frequencies, e.g., power plants. This approach would be beneficial

to manage those processes better, as there would not be a necessity for multiple separate controllers to control one process. The solution considered in this paper is as functional as different controllers yet combine all of them with an integrated layer of management. Nevertheless, more complex control loops would be considered as the system is developed.

The main contributions of this paper are summarized as follows: (i) a system allowing for parallel execution of control algorithms is developed, (ii) the robust structure of this control system is justified and discussed, (iii) simulation and experimental tests are performed using the proposed system to validate its applicability.

This paper is organized as follows. In Section 2, the theoretical formulation of DMC, specifically its fast version Fast Dynamic Matrix Controller (FDMC) is given. FPGA is described in Section 3, giving a foundation for system design description, which starts in Section 4 with the description of testbench. In this section, there are given details of the considered hardware interface between FPGA and the external devices. System design description is then continued through Section 5, where the implementation details in Verilog Hardware Description Language (VHDL) are given and discussed. Section 6 focuses on the last part of the presented system, i.e., the software Nios II processor, which manages all the aforementioned elements of the system. The overall testing environment is described in Section 7, presenting how separately discussed elements of the system are connected into a whole solution. Sections 8 and 9 present simulation and real-life results, respectively. Paper is concluded in Section 10, main advantages of the proposed system are listed, and potential further development routes are shortly discussed.

2. Fast Dynamic Model Control

This chapter presents the theoretical basis of the predictive control algorithm with a derivation of the formulas of the computationally efficient version.

A Dynamic Matrix Control is a Model Predictive Control algorithm used for asymptotically stable processes. The control signal is determined by minimizing some cost function, assuming that the optimization variable is a trajectory of future control increments. Further on, only discrete control is considered. The following notation will be used: manipulated variable $y(k)$ denotes measured output signal of the process of control, controlled variable $u(k)$ denotes signal that affects the process of control, and $y^{\mathrm{sp}}(k)$ denotes desired output signal value, i.e., setpoint. In the introduced signals, discrete-time instants $k = 0, 1, 2, \ldots$ are used. To stress some signals' predictive nature, a notation $x(p|k)$ is used, which should be read as a value of signal x determined at time instant k, for the time instant p, where $p \geq k$. Also a abbreviation $\|x\|^2_\Theta = x^{\mathrm{T}}\Theta x$ is commonly used in following formulas. Only controllers with single input and single output are considered in this paper, as the presented algorithm is easily extended for larger problems [6].

To determine a quality of control based on the future control trajectory increments, i.e., optimization variable, $\triangle u(k) = u(k) - u(k-1)$):

$$\triangle \boldsymbol{u}(k) = [\triangle u(k|k) \ \ldots \ \triangle u(k + N_{\mathrm{u}} - 1|k)]^{\mathrm{T}} \tag{1}$$

where N_{u} is the control horizon, a following cost function is introduced:

$$J(k) = \sum_{p=1}^{N} \left(y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k)\right)^2 \tag{2}$$

$$+ \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} \left(\triangle u(k+p|k)\right)^2 \tag{3}$$

In this formula $\hat{y}(k+p|k)$ denotes the predicted values of the controlled variable (the process output), N ($N \geq N_{\mathrm{u}}$) is the prediction horizon, $\lambda > 0$ is a penalty factor. DMC algorithm repeatedly, at each time instant k, determines control signal increments (1) as the solution of minimization of the cost function (3). However, only the first element of

this vector is ultimately applied to the control process. In the next iteration of the control algorithm, all calculations have to be conducted again.

The prediction of the process' output signal is determined based on the step-response model, in a form of a vector of coefficients $s_1, s_2, \ldots$ of length D. Value of D is called dynamics horizon is determined as an index of step-response coefficient, which satisfies relation $s_i \approx s_D$ for $i > D$.

The cost function (3) can be written in a form of:

$$J(k) = \|\boldsymbol{y}^{\mathrm{sp}}(k) - \hat{\boldsymbol{y}}(k)\|^2 + \|\triangle \boldsymbol{u}(k)\|^2_{\boldsymbol{\Lambda}} \tag{4}$$

where vector $\boldsymbol{y}^{\mathrm{sp}}(k) = [y^{\mathrm{sp}}(k+1|k) \ldots y^{\mathrm{sp}}(k+N|k)]^{\mathrm{T}}$ is of length N, the same as vector

$$\hat{\boldsymbol{y}}(k) = [\hat{y}(k+1|k) \ \ldots \ \hat{y}(k+N|k)]^{\mathrm{T}} \tag{5}$$

The matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda, \ldots, \lambda)$ has a dimensionality of $N_{\mathrm{u}} \times N_{\mathrm{u}}$. Thanks to the new form of the cost function (4) it is suitable for both single input single output as well as multiple inputs multiple outputs processes, assuming dimensions of the matrices used in this function are appropriate to the extended problem.

Because of the quadratic dependency of cost function on the control signal increments, i.e., optimization variable, the solution to this problem can be calculated analytically

$$\triangle \boldsymbol{u}(k) = \boldsymbol{K}(\boldsymbol{y}^{\mathrm{sp}}(k) - \boldsymbol{y}^0(k)) \tag{6}$$

where the $N_{\mathrm{u}} \times N$ matrix $\boldsymbol{K}$ is defined as

$$\boldsymbol{K} = (\boldsymbol{M}^{\mathrm{T}}\boldsymbol{M} + \boldsymbol{\Lambda})^{-1}\boldsymbol{M}^{\mathrm{T}} \tag{7}$$

$$= \begin{bmatrix} \overline{\boldsymbol{K}}_{1,1} \\ \vdots \\ \overline{\boldsymbol{K}}_{N_{\mathrm{u}},1} \end{bmatrix} = \begin{bmatrix} K_{1,1} & \ldots & K_{1,N} \\ \vdots & \ddots & \vdots \\ K_{N_{\mathrm{u}},1} & \ldots & K_{N_{\mathrm{u}},N} \end{bmatrix} \tag{8}$$

The step-response matrix

$$\boldsymbol{M} = \begin{bmatrix} s_1 & 0 & \ldots & 0 \\ s_2 & s_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ s_N & s_{N-1} & \ldots & s_{N-N_{\mathrm{u}}+1} \end{bmatrix} \tag{9}$$

is of dimensionality $N \times N_{\mathrm{u}}$. The free trajectory $\boldsymbol{y}^0(k)$ is defined as

$$\boldsymbol{y}^0(k) = \boldsymbol{y}(k) + \boldsymbol{M}^{\mathrm{p}} \triangle \boldsymbol{u}^{\mathrm{p}}(k) \tag{10}$$

where the matrix $\boldsymbol{M}^{\mathrm{p}}$, with dimensionality of $N \times (D-1)$ is

$$\boldsymbol{M}^{\mathrm{p}} = \begin{bmatrix} s_2 - s_1 & \ldots & s_D - s_{D-1} \\ s_3 - s_1 & \ldots & s_{D+1} - s_{\mathrm{D}-1} \\ \vdots & \ddots & \vdots \\ s_{N+1} - s_1 & \ldots & s_{N+D-1} - s_{D-1} \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} \boldsymbol{M}^{\mathrm{p}}_1 \ \ldots \ \boldsymbol{M}^{\mathrm{p}}_{D-1} \end{bmatrix}^{\mathrm{T}} \tag{12}$$

the vectors $\boldsymbol{y}(k) = [y(k) \ \ldots \ y(k)]^{\mathrm{T}}$ and $\triangle \boldsymbol{u}^{\mathrm{p}}(k) = [\triangle u(k-1) \ldots \triangle u(k-(\mathrm{D}-1))]^{\mathrm{T}}$ are of length N and $D-1$, respectively.

It is worth pointing out that the classical definition of the DMC algorithm assumes that all elements of the control increment vector have to be calculated. At the same time, only the first one is ultimately used, and the others are discarded. In the FDMC, calculations, which result is not useful, are omitted to shorten calculation time, allowing it to be independent on the control and prediction horizon. This is achieved by simplifying control law (6):

$$\triangle u(k|k) = K^{\mathrm{e}}(y^{\mathrm{sp}}(k) - y(k)) - \sum_{i=1}^{D-1} K_i^{\mathrm{u}} \triangle u(k-i) \tag{13}$$

where

$$K^{\mathrm{e}} = \sum_{p=1}^{N} K_{1,p} \qquad K_i^{\mathrm{u}} = \overline{\boldsymbol{K}}_1 \boldsymbol{M}_i^{\mathrm{p}} \tag{14}$$

It is necessary to underline that this simplification is possible only if the setpoint value is constant on the prediction horizon, i.e., $y^{\mathrm{sp}}(k+1|k) = \ldots = y^{\mathrm{sp}}(k+N|k)$.

3. Field Programmable Gate Array

The chapter presents basic information about FPGAs taking into account their design and features particularly relevant to the present work.

Since 1980 there has been a great breakthrough of MOS technology and it has begun to dominate the entire integrated circuit market. The year 1980 celebrated the first one million transistors in an integrated circuit, today there are hundreds of millions of transistors in one integrated circuit bone using MOS technology.

The breakthrough came with the development of a reprogrammable integrated circuit. In 1984, Altera first provided the market with a reprogrammable logic IC with symbol EP300. The programming of the logic was accomplished using EPROM memory. Today several leading companies offer FPGAs on the market: Intel (former Altera), Xilinx, Lattice, Microsemi. The main principle of operation is the same with all manufacturers. Using VHDL or Verilog hardware description languages allows design portability between different vendors, unfortunately using vendor-specific function blocks does not allow full code portability as you have to adapt those blocks to the new vendor or even use their dedicated functions. Differences also appear in peripheral hardware and hardware capabilities. Advanced FPGA ICs contain a very large number of programmable logic cells in their structure, typically from 3000 to as many as 9 million. A distinguishing feature of FPGAs is that they can be repeatedly configured directly by the user. They are ideally suited for concept testing in phase testing. Application Specific Integrated Circuit (ASIC) chips, which have their configuration permanently written into their structure, are then selected for volume production.

The three most important elements of FPGAs are as follows: logic blocks, input and output blocks and internal interconnect matrix. The logic blocks are user-configurable. Similarly, the input and output blocks are fully configurable. For the entire system to function properly, an internal connection matrix still needs to be configured, where the appropriate connections between the logic blocks and the input-output blocks can be set. The entire configuration of the FPGA chip is stored in configuration memory. Most commonly, this memory is static RAM. Upon power-up, the configuration is loaded from the FLASH memory and only later the RAM is used.

Each logic cell consists of look-up tables (LUTs), multiplexers, arithmetic logic, and registers. These blocks provide an optimal representation of the logic functions required by the user. It should be noted that logic blocks operate simultaneously and can perform independent operations. They can also perform mathematical operations. If we combine both of these properties then programmable circuits allow different tasks to be performed at the same time in many arithmetic units. The limitation, of course, is the number of logic units available, but with current technology this limitation is insignificant.

The input and output blocks allow you to use each pin as an input or as an output. If the input option is selected, it is possible to use resistors to pull the line to supply voltage or ground. The inputs are also protected against electrical charge-discharge. The output configuration can use the built-in D flip-flop, a three-state buffer, synchronous or asynchronous operation capability.

The internal connection matrix provides optimal connections between internal blocks through three connection types. Available are general purpose connections, long lines, and direct connections. General-purpose connections use metallization paths grouped into segments. Switch matrices occur at intersections, and every distance there are also built-in buffers for the signal regency. Long lines are dedicated routes that bypass the switch matrices and are dedicated to connecting distant blocks. This achieves a small phase shift of the clock signal, which is especially important in the synchronous operation of logic circuits. Most often these connections are used to distribute the clock signal within the logic circuit structure. Direct connections are used to connect adjacent blocks, thus achieving minimal delay in propagation of signals bypassing switch matrices.

The advantages of FPGAs include highly parallelized signal processing. Due to the functional representation of the algorithm in the structure employing an appropriate HDL code, it is possible to completely transfer the algorithm to the FPGA structure. Hardware realizations of control algorithms will be developed. What is more, FPGAs enable parallel operation, which will allow operating many regulation algorithms simultaneously. The possibility of parallel work of many circuits in FPGA structure will allow for a large increase in computational speed. It is possible to implement several processors operating completely independently. Thanks to the high speed and computational capabilities of FPGAs, it will be possible to apply predictive control algorithms to complex processes with very short sampling periods, of the order of a single millisecond or even less than one millisecond.

4. System Design—Testbench

This chapter describes the interface board assumptions, electronic design details and hardware implementation.

The selected FPGA board provides universal connectors to which you can connect your own devices. The allowable signal voltage levels are between 0 and 3.3 V. Such a standard is, of course, completely impractical for use in industrial process control. This can very often be a big problem, while e.g., PLCs offer interfaces that meet industrial standards. To solve this problem, an interface board dedicated to the FPGA under consideration was designed and fabricated. Each input and output is galvanically isolated using ADUM circuits to provide magnetic isolation. This protects the ports of the FPGA chip from being damaged by abnormal voltages. The following interfaces have been prepared, with the main focus on industry standards:

- analog input $\pm$10 V: signals from sensors, servo speed measurement,
- analog output $\pm$10 V: servo speed control, proportional valves,
- digital input +24 V: binary signals, sensors, incremental encoders,
- digital input differential A, /A, B, /B: incremental encoders from servo,
- digital input +3.3 V: other low voltage sets,
- digital output +24 V: PWM, binary outputs, stepper motor or servo control,
- digital output +3.3 V: other low voltage sets.

On the FPGA side, an IDC connector is available to connect the FPGA pins to the corresponding signals on the interface board. A +5 V and +3.3 V power supply is also provided on the FPGA side of the board to supply power to the galvanic isolation circuits. There is a power supply on the interface board whose main task is to provide all the required voltages on the isolated side. The main power input requires 24 V. There is a 10 V stabilizer, a 5 V stabilizer, a 3.3 V stabilizer for analog circuits, a 3.3 V stabilizer for digital circuits, and a 2.5 V stabilizer as a reference for analog-to-digital converters. In addition, a voltage converter module is available to produce $+10$ V and -10 V voltages for the analog signal outputs. Each of the stabilizers is enclosed by a capacitor bank to

provide adequate suppression of power supply ripple. The power supply is equipped with LEDs confirming its operation. In Figure 1 shows a schematic of a single ±10 V analog input, which is implemented as a resistive divider with correction capability. The voltage at the output can vary in range from 0 to 2.5 V, which is required by the analog-to-digital converter. Schottky diodes watch over the protection of the allowable voltage range. The input signal can be applied to an ARK screw connector or two pin connectors. The output of this simple circuit is given to a secondary built on an operational amplifier MCP6004. From here, the protection diode is fed to a +3.3 V supply.

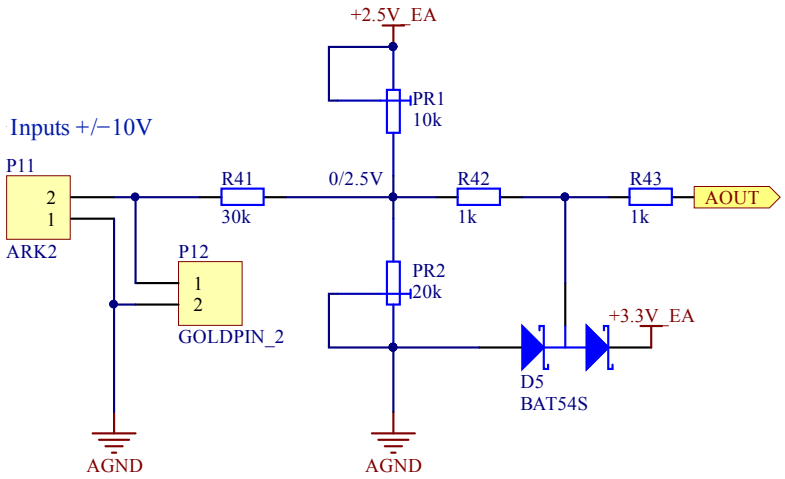

Figure 1. Schematic of a single ±10 V analog input.

Figure 2 shows a further part of the analog-to-digital conversion circuit. Each input voltage divider is connected to an operational amplifier whose purpose is to realize high input impedance, and also to provide additional separation between the physical input and the ADC. The converter is equipped with 8 channels, of which 4 are permanently connected to the outputs of the operational amplifiers. The four remaining inputs can be wired according to the user's needs, e.g., they can be other operational amplifiers or other external circuits dedicated to the application. The ADS8344EB ADC has an internal multiplexer that selects an input for signal processing by successive SAR approximation. The result is available on a digital bus that is controlled from the FPGA chip. A galvanic isolation circuit, ADUM163N0BRZ, was used to transfer the signals. This circuit has internal coding and decoding circuits, so that it is able to carry both the variable component and the constant component.

A section of the digital-to-analog processing circuit has also galvanic separation in the form of the ADUM160N0 chip, which passes signals to 4 independent DACs. The DACs require only the SCLK clock signal, DIN data signal, and SYNC selection signal. The control of the corresponding signal states is handled by the dedicated logic implemented in the FPGA structure. A further digital-to-analog processing path has DAC8560 signal converter itself and the OPA277U operational amplifier, which converts the analog voltage from the 0/3.3 V range to −10/+10 V. A single binary input for a 24 V signal consists of a simple voltage divider. Additionally, a light-emitting diode is included for quick verification of the current input state. After the divider there is also a voltage range protection with Schottky diodes. A further processing of the 24 V binary signal goes to the galvanic isolation system ADUM160N. The 3.3 V binary input has been solved similarly. A 24 V binary output solution provides a possibility to manually control the output with the PB1 button. It is possible to operate the output in two variants of logic. Transistor allows to control the signal by shorting to ground the system, another transistor allows to control the signal by shorting to power supply the system. The control type can be selected by jumpers. In addition, the output is equipped with a light-emitting diode that indicates its current state. There

are also diodes to protect against overvoltages that may occur when controlling actuators with high inductance such as valve coils. A differential input circuit has 4 differential channels, converted to digital form by the AM26LS32ACD chip. These signals then go to the ADUM160N0 galvanic isolation circuit. Additional LEDs allow for quick verification of proper connection. For the case under consideration, it is possible to connect two servo motors with differential signals with A and B outputs. This allows the current position of the motor or an independent incremental encoder to be counted. The designed PCB has at the center an IDC connector for connection to the main FPGA board. Around the connector are ADUM circuits for galvanic isolation. At the bottom is the power supply for external signals. On the left are the 3.3 V output circuits, above on the right are the 3.3 V inputs, in next are the differential inputs, then the $\pm$10 V analog output circuits, on the top right are the 24 V binary inputs, below are the $\pm$10 V analog inputs. Between the IDC connector and the power supply are the 24 V binary outputs. The designed board is double-sided, where the bottom layer is mainly a ground plane. The analog ground is separated from the digital ground to minimize interference. Due to the size of the designed board, the density of component packing, and the very small ICs requiring soldermask for soldering, the board was commissioned. All components were purchased and the assembly process began. First, the power supply was soldered and all necessary voltages were verified. When they were found to be working properly, the connection part to the FPGA board was soldered: the connector, capacitors, resistors, and ADUM chips. The operation of the inputs and outputs of the ADUM circuits was tested, in particular, the correctness of the DC transfer was verified. The maximum transmitted frequency is about 100 kHz. The next step was to solder the inputs and outputs in standard 3.3 V and test them. It was already possible to connect the board to a PC via a USB-RS232 converter and test the communication capability in both directions. In addition, communication with the thermal bench was tested, in which communication follows the RS485 protocol. The last binary signals to be run were the 24 V inputs and outputs. The corresponding circuits were soldered and connected to several input test signals.

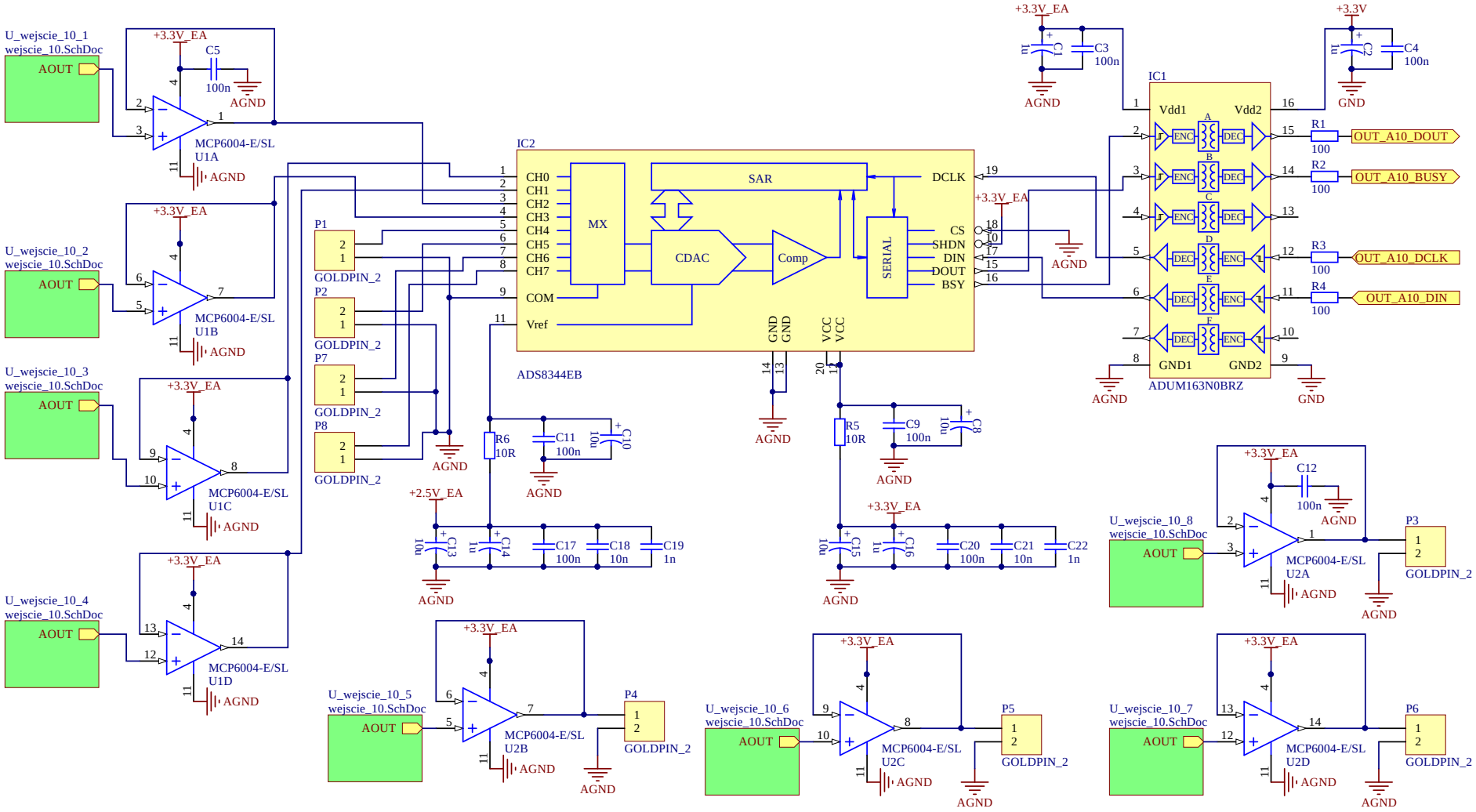

Figure 2. Analog input circuit diagram.

5. System Design—Hardware Description Language

This chapter presents a description of the implementation of the digital layer, including support for hardware interfaces and individual control algorithms.

The digital part design is realized in a dedicated environment Quartus Prime in version 18.0 by Intel. It is a comprehensive tool for FPGA, SoC (System on Chip) and CPLD (Comple Programmable Logic Devives) projects. The implementation in hardware description language is done in order, then the process of verification, logic synthesis and optimization of the interconnect layout structure is started. The software has a rich database of basic logic blocks. It is also possible to perform simulations in the related ModelSim package. In this paper, a basic configuration of the FPGA circuit was performed based on the VHDL language. The main unit contains, among others, instances of the Nios II processor, process controllers, inertia modules, PLL modules, hardware counters, analog interfaces and PWM interfaces. Both the PWM blocks and the analog signals are prepared to operate at sampling rates on the order of 1 kHz to provide 1ms signal sampling. No digital filtering operations are performed on the signals coming from the analog inputs. The logic diagram of the analog input control module generated during synthesis is shown. The circuit is designed so that values from four analog-to-digital converters (A, B, C, D) are acquired and placed in registers within 1 ms. The data coming from the converter are fed in series with one pin, so later they have to be processed by 16-bit registers and placed in appropriate output flip-flops. The whole operation is controlled by a completely independent state machine realized only by logic circuits using the process mechanism in VHDL language. The logic diagram of the analog output control module generated during synthesis is shown. The circuit is designed in such a way that values from four digital registers (A, B, C, D) are set at each analog output within 1ms. The data from the registers are fed in parallel and must be processed into a single signal fed to the converter. The whole operation is controlled by a completely independent state automaton realized only by means of logic circuits using the process mechanism in VHDL language. The automaton sets synchronization outputs in appropriate moments in correlation with a selected digital register. The logic diagram of the PWM output control module generated during synthesis is presented. The circuit is designed to update the signal fill value within 1 ms. The carrier frequency of the PWM signal is 1 kHz. The fill control is implemented by a 10-bit counter. The smallest quantum that can be set is 1 ms/1024. The physical implementation of this circuit has been realized using a combiner, a counter and a comparison circuit. It should be noted that by using digital circuits the various operations are performed in parallel and thus it is possible to obtain current measurements from each analog input at any sampling time (even every 1 ms), send new values of control signals to each analog output and new values of signals to PWM outputs. The implementation and assembly of the entire system was done in a single VHDL file. It contains all the instances of the needed logic blocks and the communication signals between them. As a result of compilation and synthesis of the whole project, the logic diagram of the system was obtained. Looking for an error in the obtained logic diagram is a big challenge. The compilation will naturally not hint at the functional error of the logic circuits. With incorrect implementation of hardware description code, it may happen that it is not possible to synthesize a given algorithm. Then the compilation and synthesis will be successful, but the generated logic gate structure will not resemble the designed circuit at all, the obtained result will not meet the assumed functional requirements. In order to effectively verify such situations, it is necessary to use a tool for simulating logic circuits. For Intel company the built-in tool is ModelSim program. It is a powerful environment allowing to fully simulate VHDL code. It is possible to simulate the original code written by the user and the code which is available after compilation and synthesis in the Quartus system. This version of the code is an accurate representation of the logic that will be on the chip. It takes into account the delays of signal transmission and the operation of individual gates. This is why it is important to verify your own code and the code generated by the Quartus program to be absolutely sure that the final circuit will work. To facilitate the verification of the circuits in ModelSim you can

use the test automation scripts. In such scripts it is possible to perform automatic code compilation, add signals for drawing, give appropriate overrides on the set signals and finally run the simulation for a specified time.

The Figure 3 shows a sample of the implementation of the computational part of the algorithm including parameter registers, multiplication and summing circuits. All elements have been duplicated according to the required dynamics horizon using the generation function which allows to easily scale the solution. After the compilation, appropriate connections of digital circuits were obtained and presented in the form of schemes as shown in Figure 4. It is possible to check the correctness of the realized control system.

```
--Generate the modules
--storing previous control increments
gen_Du_array: for i in 0 to 31 generate
Du : D_P
PORT MAP
(
    D_in    => TempAFLT(i),
    Clk     => InClkSampling,
    Rst     => InReset,
    Q_out   => TempAFLT(i+1)
);
end generate gen_Du_array;

--multiplying parameter by previous control increments
gen_mult_array: for i in 0 to 18 generate
mult_module : ALTERA_FP_MULT
PORT MAP
(
    clk     => InClk,
    areset  => InReset,
    a       => SigParamData(i),
    b       => TempAFLT(i+1),
    q       => TempBFLT(i)
);
end generate gen_mult_array;

--summation of intermediate results
gen_addder_array: for i in 0 to 18 generate
adder_module : ALTFP_ADD
PORT MAP
(
    clock   => InClk,
    dataa   => TempBFLT(i),
    datab   => TempCFLT(i),
    result  => TempCFLT(i+1)
);
end generate gen_addder_array;
```

Figure 3. Code snippet in Verilog Hardware Description Language describing a set of automatically generated multipliers with summation.

This allows to implement PID algorithms using the structure presented in Figure 5. Model Predictive Control algorithms are also feasible, and thus a fast variant of the DMC algorithm is implemented using the scheme visible in Figure 6. It is worth mentioning that further optimization can be done to lower computational time so that it grows in a logarithmic manner depending on the dynamics horizon, as it is shown in Figure 7.

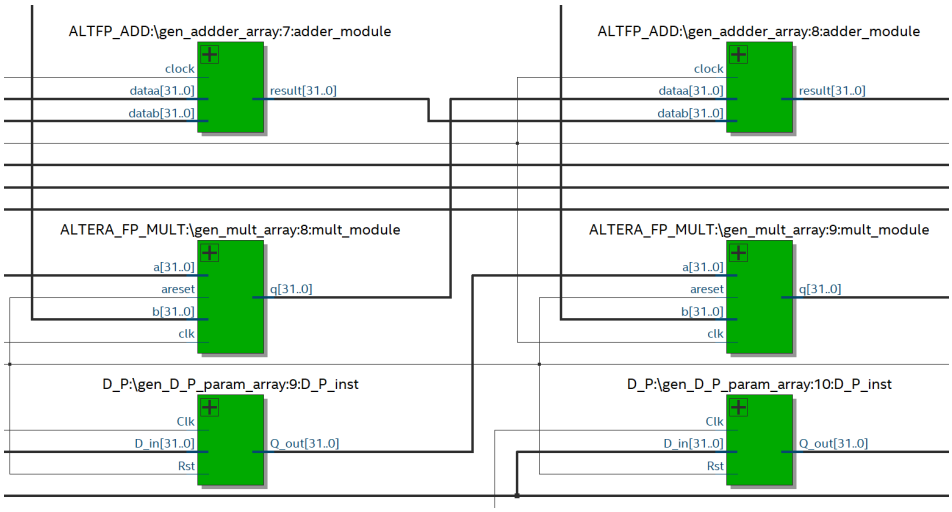

Figure 4. Automatically generated multipliers with summation as seen in the Quartus software.

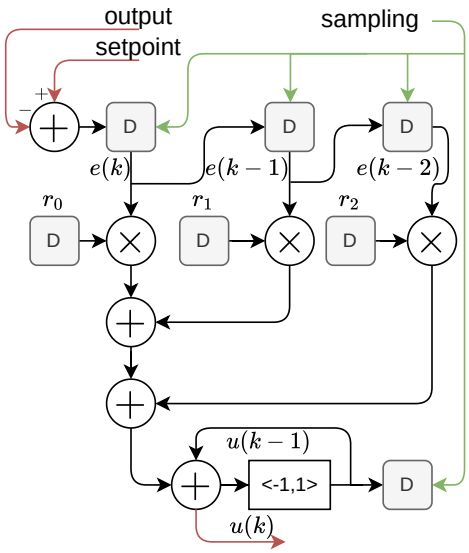

Figure 5. Current PID design implemented in the Matrix of Controllers.

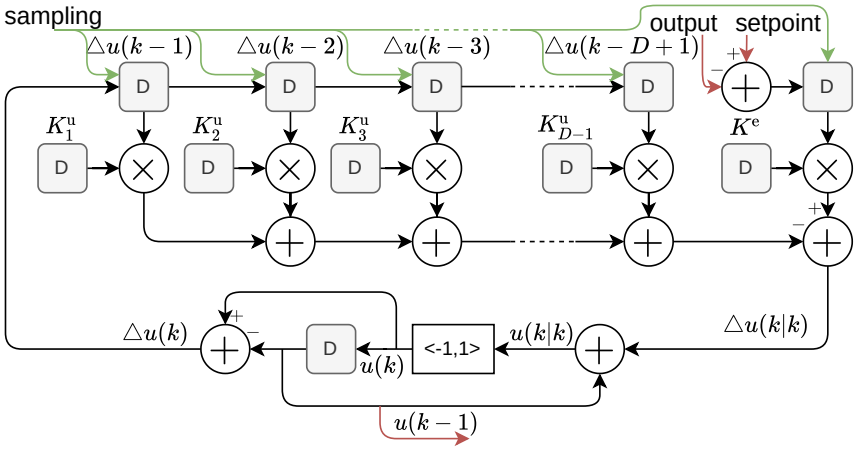

Figure 6. Design of Fast Dynamic Matrix Controller algorithm for FPGA implementation.

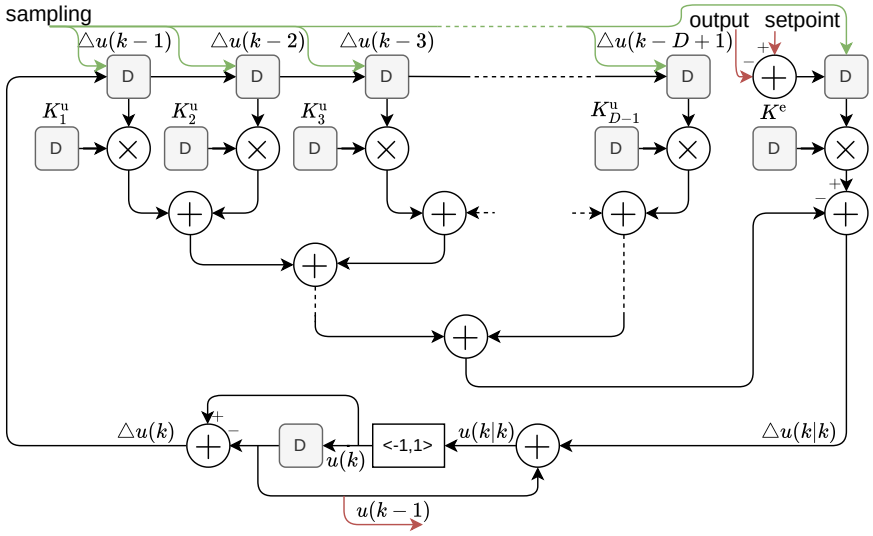

Figure 7. Improved design of FDMC with a logarithmic computational complexity.

6. System Design—Software CPU

In the following chapter, a managing processor (Nios II) is described, and configuration possibilities are discussed.

Two processors are considered in the presented approach. One is designed to oversee the control algorithms:

- Maintain the safety of internal data.
- Perform synchronization of externally supplied parameters.
- Check if the overall configuration is without issues.

The other one is considered a simplified SCADA, i.e., its tasks are to transfer and visualize data from the controllers. As the system is still under development, the functionality of both processors is contained in a single instance of a Nios II processor.

There are many tasks for which this processor is responsible. One of those is to maintain the consistency of the simulations running in the FPGA. Considering that the presented system may be used to design the control system, simulation functionality is incorporated into the described solution. The simulated process implemented on the FPGA currently considers ARX models, which allow modeling most of the commonly considered processes for industrial applications. State-space and neural models will be further implemented as the system is still in development. In the following experiments, two simulated processes with ARX models are presented.

The main concern of this processor is to maintain a safe working environment for the controllers, which includes configuring controllers. Specifically, there are three main steps of controller configuration: routing its input signals, routing its output signals, and configuring its internal parameters. All those tasks have to be carefully planned so that no uncontrolled signals from the controller are generated, as those could damage the control process.

Currently, there are two sets of input signals which can be directed as an input of the controller: measurements performed based on the analog-to-digital converters and simulated outputs from the internal simulations. The configuration is performed by the Nios II processor through a selector register, which allows deciding which source signal will be used as an input to the controller block (Figure 8 (left)). This approach allows for easy reconfiguration and even performing online changes to the signal routes. This approach will be extensively utilized in further research where redundant controllers will be tested as well as voting mechanisms. Apart from the input signal source, the intervention time of the controller can be configured dynamically. In Figure 8 (right), there is another selector which determines which clock signal will be used as the intervention signal for the controller.

The set of potential input clock signals contains both constant, predefined frequencies, as well as signals with configurable prescalers and PLL to allow customization of input clock frequency, where it is necessary.

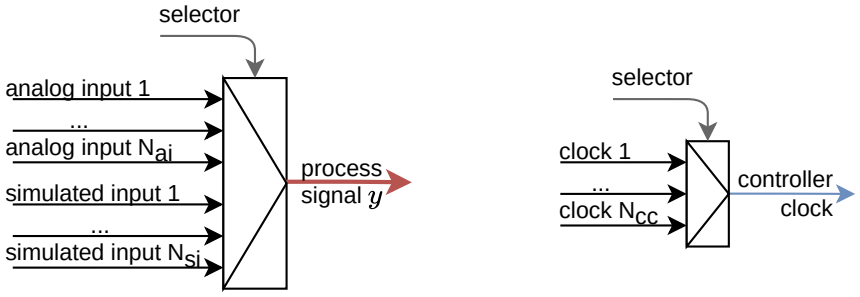

Figure 8. Selectors for input signals of the controller: process signal (**left**) and controller clock (**right**) in the considered system.

The analogous configuration has to be performed at the output side of the controller. Each output signal can have its source selected as the output of one of the controllers to prevent collisions in signal routes. This configuration is presented in Figure 9, where, as it is currently implemented, there is a set of analog output signals, a set of PWM output signals, and a set of signals that are connected with simulated processes implemented in the same FPGA unit. Each of those inputs has at most one of the controllers' outputs selected as its source; therefore, it is not possible to direct the outputs of two controllers into the same output signal.

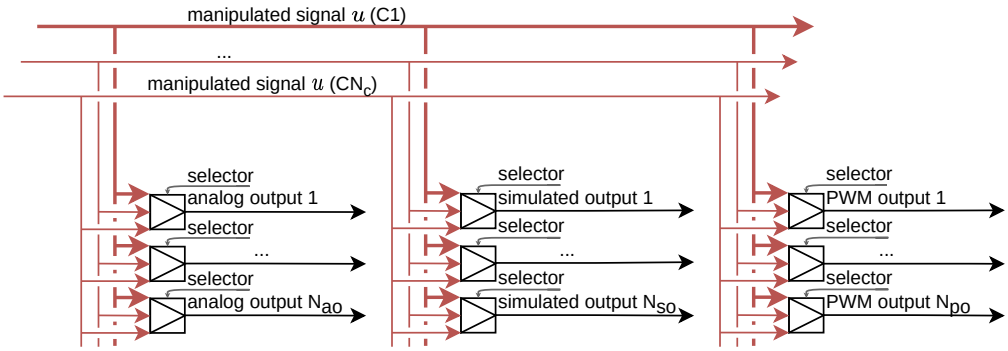

Figure 9. Selectors for output signals of the controller directing manipulated signal to one of many interfaces.

The controller itself (Figure 10) consists of a set of input and output signals, of which some are determined through a Nios II processor. Input signal named "process signal y" and "controller clock" was already discussed, as well as output signal "manipulated signal u". Also, some signals are predefined and cannot be modified, as is the case with the computation clock, which is used as the clock base for all internal mathematical operations performed inside a specific controller implementation. Nevertheless, parameters of the controller and setpoint value can and have to be configured through the considered in this section processor. There is a predefined number of parameters, although because of the automatic generation utilized in the controllers, this number can be easily modified in the HDL description. Before the controller is used, the processor has to set each one of the parameters to the correct value by setting its consecutive indices and values of parameters and sending a pulse signal through the latching line. At this point in development, those parameters are precalculated outside of the Nios II processor, and the processor is used only to transfer those values, but the aim is to perform as many calculations as possible in the FPGA itself. A similar concept is utilized with setpoint value – this time, however, there is no need for index information as there is only one value for a single output signal. Again latch signal is used to effectively store new setpoint value in the controller. There are also

control and status signals that are used to perform additional operations on the controller (e.g., stop or force its execution) and get information about its current state, respectively. The latter is used to determine which controller performed all its operations and, therefore, which data are ready to collect and transfer or visualize.

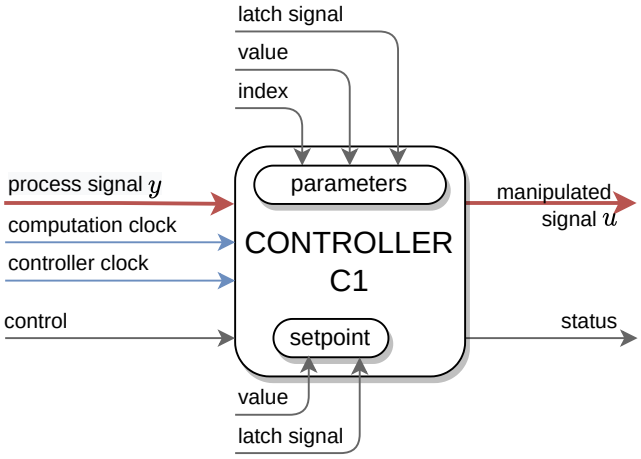

Figure 10. General controller structure as it is implemented in the considered system.

The last task of this processor is to collect data from controllers. The processor obtains data independently of the intervention periods of the controllers. In the following experiments this period was set to 10 and 25 milliseconds. This allows collecting data with a granularity fitting dynamics of the whole process instead of focusing on the particular controller and its data. This approach makes it possible to first observe each control loop in detail in the stage of controller tuning, and when it is finished, one can shift focus on the whole plant, where there are more complex relations and more advanced controllers.

Data that are collected by the processor should also be accessible further in time for the sake of analysis. Two approaches are considered: constant transfer of data as soon as they are obtained and transfer of data in bulks. The first one is a proper approach when sampling time is higher than the transfer time of a single set of data. In this case, there are no issues, and thus a simple solution is the best one. In case data are collected faster than the processor can send them, such a policy of transferring data would result in frequent data inconsistency. If the inconsistency is imminent, it is important to have them under control. Therefore it is advised to gather all data and buffer them inside the processor, and when the amount of information is big enough, all of it should be sent in one transfer. This would obviously generate a period of no measurements, as the processor is busy with transferring data instead of collecting them, but the data that is already collected and is being transferred is consistent in itself, which is critical in further analysis.

This approach can be further extended with the compression algorithms or introduction of a threshold of changes in signals after which the signal is determined to have changed, as it is commonly used in SCADA systems.

As the processor has many tasks to handle, no visualization is currently performed on board. As this is a task of lower priority, it should be performed with another processor specifically designed for this task only, and it should be connected to the data bus of the considered in this section processor.

7. Testing Environment

This chapter describes the hardware bench built and lists the major components included. The testing environment considered in this research is shown in Figure 11.

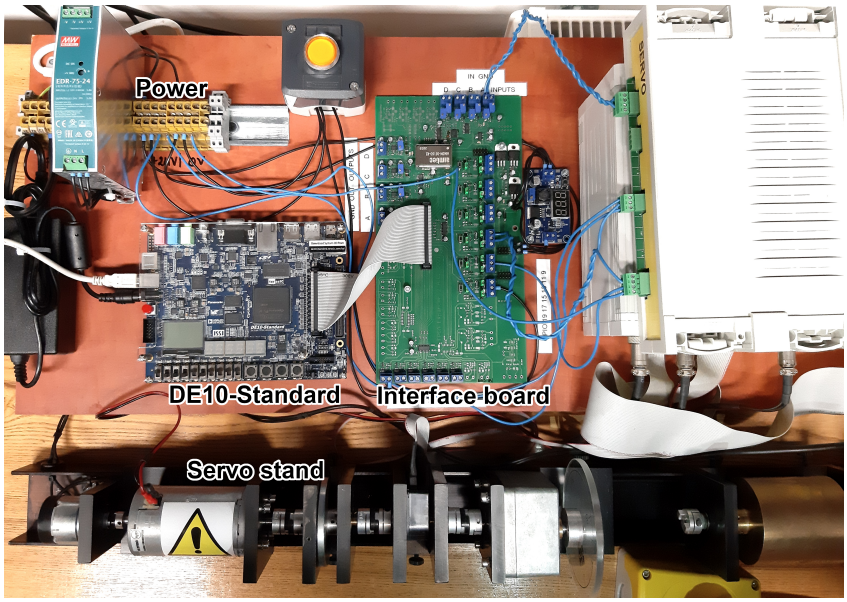

Figure 11. The testing environment, which includes: Power supply, FPGA development board DE10-Standard, Interface board and servo motor—the process of control.

DE10-Standard board is connected with the interface expansion board, which is further connected with a modular servomotor. This servomotor is made by Inteco and allows the use of a PWM signal to influence the rotational speed of the servo. As the measurements, readings from the tachogenerator are used. As the power cannot be given from the development board directly, an amplifier is necessary. Technical details are gathered in the Table 1. Servomotor's speed is controlled with an FDMC controller with an intervention period of 0.01 s.

Table 1. Test enviroment equipment summary.

Section	Component	Additional Information
Terasic DE10-Standard development board	FPGA chip	Cyclone V SoC SCSXFC6D6F31C6N 110,000 logic elements
	Memory	64 MB SDRAM
	Interface	USB, Ethernet, JTAG, USB
	User interface	Switches, Buttons, LEDs
	I/O interface	40 pin connector
Interface board	Power supply Analog inputs	10 V, 5 V, 3.3 V DC 4 ports $\pm$ 10 V
	Analog outputs	4 ports $\pm$ 10 V
	Digital inputs	6 port 24 V
	PWM outputs	6 port 24 V
Servo stand	Control Unit Interface Unit	Power supply, Safety Conversion signals to 24 V standard
	Main Motor	24 V DC motor, 3000 rpm, 3.1 A
	Measurements	Tachogenerator, Incremental Encoder
Power	Power supply	24 V DC 3.2 A
	Fuse	short circuit protection

Two additional simulated processes are introduced to emphasize the ability of the presented system to control processes with various controller intervention periods. Those processes are modeled as transfer functions, namely:

$$G_1 = \frac{1}{0.25\,\mathrm{s} + 1} \qquad G_2 = \frac{1}{0.50\,\mathrm{s} + 1} \tag{15}$$

where G_1 and G_2 are simulated both with a frequency of 1 kHz, and controllers are configured to have an intervention period of 0.1 s and 1 s, respectively. Process G_1 is controlled with FDMC while G_2 is in loop with PID controller. The selection of control algorithms for those processes is arbitrary as both FDMC and PID are equally good choices. The development board is connected to the PC computer, where it periodically sends measurements in sets of 3000 because of the short sampling time. It is worth mentioning that the Nios II processor is used to set controllers' setpoint values and does it based on the number of iterations of the sampling loop. This aspect will be described further in the following sections.

To configure the system described in this paper, one has to perform the following steps:

1. determine which algorithm is appropriate for the process,
2. perform configuration of the controller:
   (a) choose one of the free instances of the appropriate control algorithm,
   (b) select the input signal for the controller (ADC/output of simulated process),
   (c) select output of the considered controller as a source for the output signal (DAC/PWM/input of simulated process),
   (d) set intervention period of the controller,
   (e) determine and set parameters of the control algorithm:
      - for the FDMC first 63 of 64 registers represent values of K_i^{u}, while on the last position, a K^{e} is stored.
      - for the PID algorithm, values of r_0, r_1 and r_2 are stored at registers with indices 0, 1 and 2, respectively.
   (f) define setpoint values based on the current requirements.

In the case of the PID algorithm, tuning of the parameters can be performed using one of many commonly used methods, e.g., step response method, relay method. In the case of the FDMC algorithm, to determine parameters of the controller, one has to obtain step response coefficients and determine values of K^{e} and K_i^{u}:

1. set control values as constant in time (e.g., connect source for the output signal directly to Nios)
2. when the output signal stabilizes, change the control signal, generating a step input on the process of control
3. from this point in time onward, store all measurements in a reference to the stable signal from 2 as long as the output signal changes
4. filter obtained measurements or generate ideal approximation of the step response
5. store step response coefficients in a form of vector $s = [s_1, s_2, \ldots, s_D]^{\mathrm{T}}$
6. calculate final parameters of the FDMC controller:
   (a) calculate matrix $\boldsymbol{M}$ using Equation (9)
   (b) calculate matrix $\boldsymbol{M}^{\mathrm{P}}$ using Equation (11)
   (c) calculate matrix $\boldsymbol{K}$ using Equation (7)
   (d) using Equations (8), (12) and (14) determine values of K^{e} and K_i^{u}.

After those steps, the controllers are ready to be utilized. Nevertheless, even during runtime, it is allowed to change the controller's parameters and thus perform online tuning, or even reroute signals, e.g., routing redundant measurement to the controller in case of problems with the original signal.

8. Simulation Results

In this section a reference control trajectories are determined based on the MATLAB simulation.

Before the experiments on the testing environment were conducted, a MATLAB simulation was implemented and executed. A model of the servomotor was determined based on experimental data:

$$G_{\text{servo}} = \frac{-0.3876}{0.0387\ \text{s} + 1} \tag{16}$$

and an offset of 0.9 was determined for the control signal.

Three controllers are defined to control output signals of the simulated processes. Namely, FDMC is used to control the output signal (rotational speed) of simulated servo—this controller will be called C1. Another FDMC is used to regulate simulated process based on a transfer function G_1, and lastly, the PID controller regulates the output signal of a simulated process G_2 (Figure 12 (left)).

As the simulated processes are updated each millisecond on the FPGA, in the simulation, the same 1ms was used as a period after which all three considered processes determine the new measurement. As the sampling time, 10 ms was used, and the controller intervention period was set to 10 ms, 100 ms, and 1000 ms for C1, C2, and C3, respectively. The setpoint value for each controller is determined and configured in a sampling subroutine, which is executed each 10ms. Setpoint changes occur in a regular fashion for each controller: for C1 each 150 sampling periods (1.5 s), for C2 each 300 sampling periods (3 s), and each 600 sampling periods (6 s) for C3. Results of the simulation are presented in Figure 13. The output signal of the process is denoted as y, while the control signal from the controller is written as u. The number of the control loop (for C1, it is 1, for C2 – 2, and for C3, it is 3) is visible in the subscript of the signal name. On the left-hand side, there are results of the whole experiment, while on the right-hand side, only a selected period of time is visible. This allows us to see how the frequency of controller interruption varies between defined controllers.

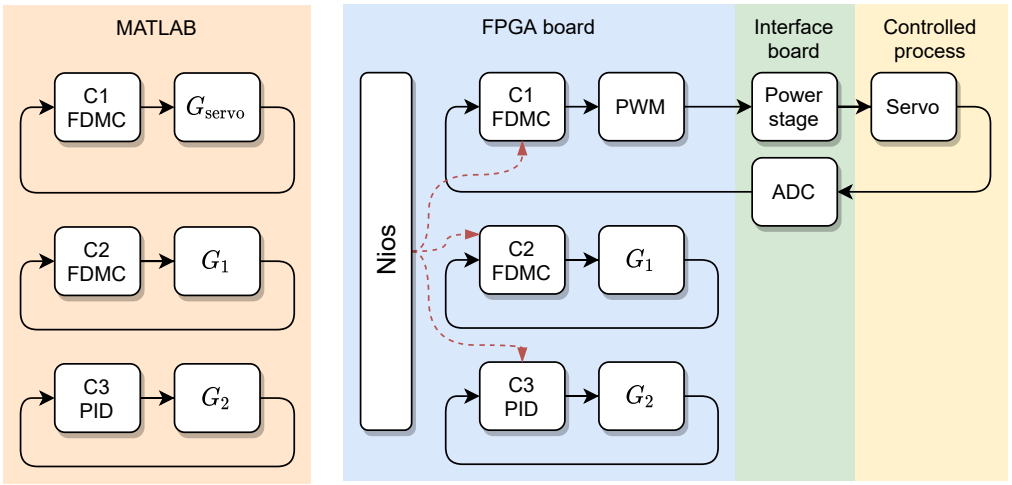

Figure 12. The structure of control loops in the simulated (MATLAB) environment (**left**) and FPGA with servomotor as a process of control (**right**).

It is worth noting that due to the artificially introduced offset between the setpoint changes of C3 and its intervention periods, there is a delay before controller C3 reacts to the change of control error. Nevertheless, this has to be underlined that this behavior is expected and should not be considered as an error.

There is a drawback to this simulation, as all the controllers are executed in synchronization to one global clock, thus executing in the worst-case scenario all three controllers in the same iteration of the simulation. Although in this example there are no complex calculations, the total time of execution of all controllers equals the sum of time required

to run each of the controllers. With more complex or more dynamic processes, this time would lengthen, whereas, in the proposed system, all controllers work independently, and thus execution time of any controller does not depend on the time of any other controller.

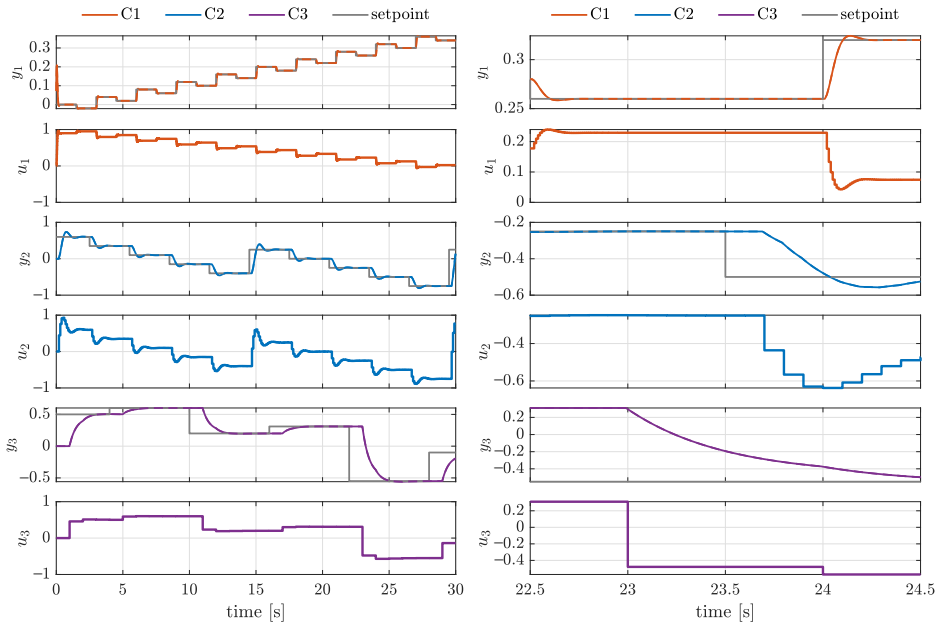

Figure 13. Results of fully simulated experiment implemented in MATLAB: full experiment (**left**) and zoomed (**right**).

9. Experiment Results

Experimental results are presented in this section and the configuration utilized to obtain those results.

The analogous experiment was performed with the testing environment, and thus with a real-life servomotor connected via ADC and PWM, and two internally implemented in the FPGA simulated processes. All of those are controlled with independent controllers implemented inside of FPGA (Figure 12 (right)).

Time periods for updating simulated processes, controller interruption, and sampling are the same as in the simulated environment. Also, setpoint trajectories are kept the same for the sake of a clear comparison between expected results and collected ones.

Results are presented in Figure 14. There is visible noise in the output signal of the servomotor (y_1), which is connected with the inaccuracies of the mechanical connections between the servomotor shaft and the tachometer. Also, a shift in the control signal is visible, as the servomotor is controlled using reverse polarity PWM, and thus with greater width of the PWM, the power delivered to the servomotor decreases.

It is important to notice that the control quality of the second and third control loop is the same as the quality presented in simulated results. Even though this observation might be expected, it implies that the execution of a number of controllers in parallel can be executed in such a way that they indeed do not interfere with each other keeping their control quality as if they were executed one after another.

To further stress independence between the process of control and process of data gathering, Figure 15 shows data collected with a sampling period of 25 ms. As there are still samples from 3000 discrete time instants, the overall time increased to 75 s even though the setpoint trajectory did not change (because it is dependent on the sampling iteration) measured output signals of processes of control change faster, which is to be expected.

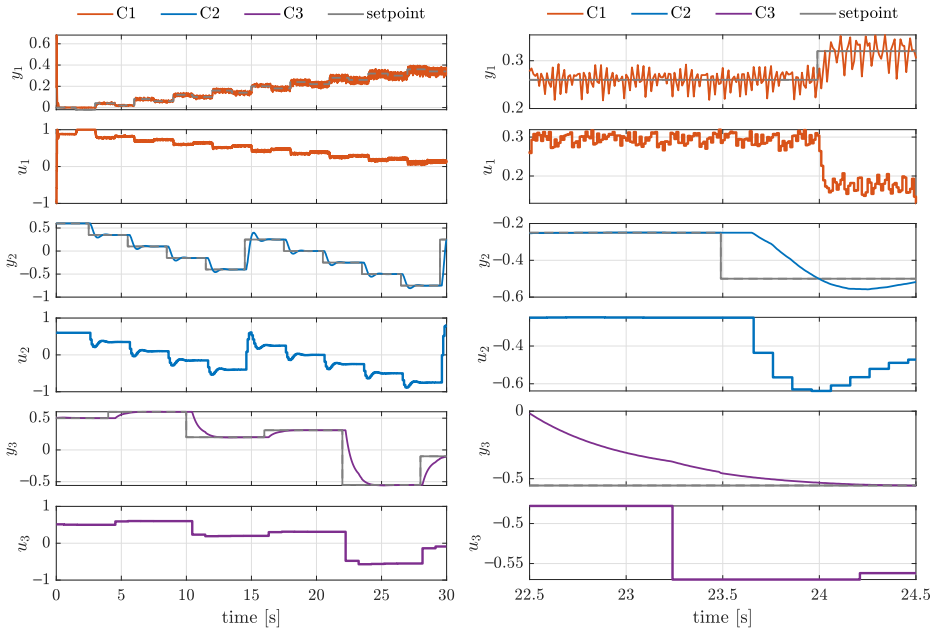

Figure 14. Results of experiment conducted with the servomotor and simulated on FPGA processes, with a sampling period of 10 ms: full experiment (**left**) and zoomed (**right**).

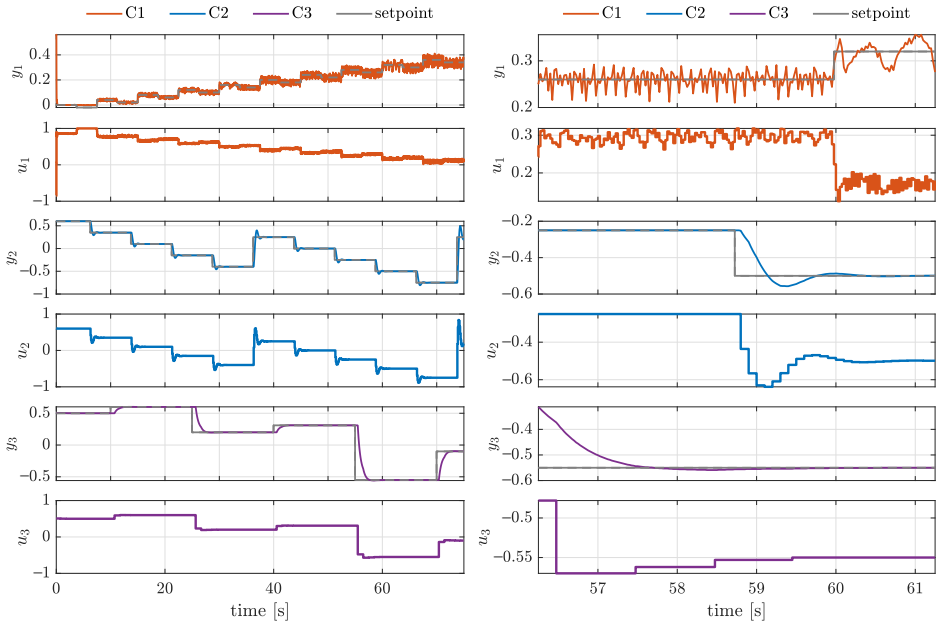

Figure 15. Results of experiment conducted with the servomotor and simulated on FPGA processes, with a sampling period of 25ms: full experiment (**left**) and zoomed (**right**).

10. Conclusions

In this paper, the authors presented the novel design of the control system based on the FPGA unit, which allows running independent controllers with various intervention times in an organized manner. This approach gives the ability to perform data gathering in yet another process, which is designed to mimic a simplified SCADA, which is implemented using the Nios II processor implemented in the same FPGA. Simulations were performed and compared with the experiments in the testing environment to determine if the quality of the controllers was hindered by the presented approach, and results show no such effects.

The main advantages of the solution proposed in this paper compared to the alternatives include.

1. the ability of the proposed system to simultaneously control both fast and slow processes,
2. the scalability of the system in terms of number of controlled processes—new controller instance does not influence already existing control loops,
3. the flexibility of input and output signals routing, which can be modified during runtime and allow redundancy in controllers by connecting one measurement signal to multiple controllers,
4. the ability to store parameters as mutable registers shared between Nios processor and controller instances,
5. the separation of the control loops from the controller manager unit (Nios processor), allowing its code modification without affecting the stability of the running control loops,
6. the modular approach to the configuration of control loops, allow for easy development of defining operations on input/output signals.

The authors plan to further develop this system—main areas of development will include incorporating a larger set of control algorithms and designing a user interface for easier controller configuration and maintenance. In addition, the possibility of developing the system with control algorithms such as General Predictive Controller is considered. The authors plan of developing the system towards control signal mixing, and thus allowing implementation of fuzzy systems. Other goals include using solutions that do not require expensive licenses and increasing the number of controllers and connecting more real control processes.

Author Contributions: All authors have equally contributed to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research was financed by Warsaw University of Technology in the framework of the project for the scientific discipline automatic control, electronics and electrical engineering.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Maciejowski, J. *Predictive Control with Constraints*; Prentice-Hall, Pearson Education Limited: Harlow, UK, 2002.
2. Tatjewski, P. *Advanced Control of Industrial Processes, Structures and Algorithms*; Springer: London, UK, 2007.
3. Wojtulewicz, A. Implementation of Dynamic Matrix Control Algorithm Using Field Programmable Gate Array: Preliminary Results. *Adv. Intell. Syst. Comput.* **2017**, *577*, 325–334.
4. Wojtulewicz, A.; Ławryńczuk, M. Implementation of multiple-input multiple-output dynamic matrix control algorithm for fast processes using field programmable gate array. *Ifac Pap.* **2018**, *51–56*, 324–329. [CrossRef]
5. Zarzycki, K.; Ławryńczuk, M. Fast Real-Time Model Predictive Control for a Ball-on-Plate Process. *Sensors* **2021**, *21*, 3959. [CrossRef] [PubMed]
6. Chaber, P.; Ławryńczuk, M. Fast Analytical Model Predictive Controllers and Their Implementation for STM32 ARM Microcontroller. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4580–4590. [CrossRef]
7. Chaber, P.; Ławryńczuk, M. AutoMATiC: Code generation of model predictive control algorithms for microcontrollers. *IEEE Trans. Ind. Inform.* **2020**, *16*, 4547–4556. [CrossRef]
8. Richards, A. Fast model predictive control with soft constraints. In Proceedings of the 2013 European Control Conference (ECC), Zurich, Switzerland, 17–19 July 2013; pp. 1–6. [CrossRef]
9. Bemporad, A.; Oliveri, A.; Poggi, T.; Storace, M. Ultra-Fast Stabilizing Model Predictive Control via Canonical Piecewise Affine Approximations. *IEEE Trans. Autom. Control* **2011**, *56*, 2883–2897. [CrossRef]
10. Stellato, B.; Banjac, G.; Goulart, P.; Bemporad, A.; Boyd, S. OSQP: An operator splitting solver for quadratic programs. *Math. Program. Comput.* **2020**, *12*, 637–672. [CrossRef]
11. Nebeluk, R.; Ławryńczuk, M. Tuning of Multivariable Model Predictive Control for Industrial Tasks. *Algorithms* **2021**, *14*, 10. [CrossRef]
12. Yang, G.; Wang, H.; Chen, J. Disturbance compensation based asymptotic tracking control for nonlinear systems with mismatched modeling uncertainties. *Int. J. Robust Nonlinear Control.* **2021**, *31*, 2993–3010. [CrossRef]

13. Yang, G.; Yao, J.; Ullah, N. Neuroadaptive control of saturated nonlinear systems with disturbance compensation. *ISA Trans.* **2021**. [CrossRef]
14. Wang, S.; Li, S.; Chen, Q.; Ren, X.; Yu, H. Funnel tracking control for nonlinear servo drive systems with unknown disturbances. *ISA Trans.* **2021**. [CrossRef] [PubMed]
15. Wang, S. Approximation-Free Control for Nonlinear Helicopters with Unknown Dynamics. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**. [CrossRef]
16. Short, M.; Abugchem, F. A Microcontroller-Based Adaptive Model Predictive Control Platform for Process Control Applications. *Electronics* **2017**, *6*, 88. [CrossRef]
17. Mohamed, K.; El-Mahdy, A.; Refai, M. Model predictive control using fpga. *Int. J. Control. Theory Comput. Modeling (IJCTCM)* **2015**, *7*, 2. [CrossRef]
18. Gulbudak, O.; Gokdag, M. Efficient FPGA-Based Real-Time Implementation of Model Predictive Control for Single-Phase Direct Matrix Converter. *Electr. Power Components Syst.* **2020**, *48*, 1773–1785. [CrossRef]
19. Wojtulewicz, A. Hardware Accelerators for Fast Implementation of DMC and GPC Control Algorithms Using FPGA and Their Applications to a Servomotor. In *Advanced, Contemporary Control*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 48, pp. 1079–1091. [CrossRef]
20. Patne, V.; Ingole, D.; Sonawane, D. FPGA Implementation Framework for Low Latency Nonlinear Model Predictive Control. *IFAC-PapersOnLine* **2020**, *53*, 7020–7025. [CrossRef]
21. Ndje, M.; Bitjoka, L.; Boum, A.T.; Fotsa Mbogne, D.J.; Buşoniu, L.; Kamgang, J.C.; Tchané Djogdom, G.V. Fast constrained nonlinear model predictive control for implementation on microcontrollers. *IFAC-PapersOnLine* **2021**, *54*, 19–24. [CrossRef]
22. Wain, R.; Bush, I.; Guest, M.; Deegan, M.; Kozin, I.; Kitchen, C. An overview of FPGAs and FPGA programming; Initial experiences at Daresbury. *Counc. Cent. Lab. Res. Counc.* **2006**. Available online: https://epubs.stfc.ac.uk/manifestation/1167/DL-TR-2006-010.pdf (accessed on 28 January 2022).
23. Sankar, D.; Syamala, L.; Chembathu Ayyappan, B.; Kallarackal, M. FPGA-Based Cost-Effective and Resource Optimized Solution of Predictive Direct Current Control for Power Converters. *Energies* **2021**, *14*, 7669. [CrossRef]
24. Craven, S.; Athanas, P. Examining the Viability of FPGA Supercomputing. *Eurasip J. Embed. Syst.* **2006** , 1–13. [CrossRef]
25. Edwards, M.D.; Forrest, J.; Whelan, A.E. Acceleration of software algorithms using hard-ware/software co-design techniques. *J. Syst. Archit.* **1996**, *42*, 697–707. [CrossRef]
26. Sonawane, D.N.; Sutaone, M.S.; Malek, I. Systolic Architecture for Integer Point Matrix Multiplication using FPGA. In Proceedings of the 2009 4th IEEE Conference on Industrial Electronics and Applications, Xi'an, China, 25–27 May 2009; pp. 3822–3825.
27. Jaworski, M. *Rozprawa Doktorska: Wykorzystanie Zasobów FPGA do budowy szybkich generatorów pseudolosowych*; POLITECHNIKA POZNAŃSKA WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI: Politechnika Poznańska, 2016.
28. Guneysu, T.; Kasper, T.; Novotny, M.; Paar, C.; Rupp, A. Cryptanalysis with COPACOBANA. *IEEE Trans. Comput.* **2008**, *57*, 498–1513. [CrossRef]
29. Ferguson, R.C.; Tate, R. Use of field programmable gate array technology in future space avionics. In Proceedings of the 24th Digital Avionics Systems Conference, Washington, DC, USA, 30 October–3 November 2005.
30. Habinc, S. Suitability of reprogrammable FPGAs in space applications. Technical report, Gaisler Research. 2002. Available online: http://www.gaisler.com/doc/fpga_002_01-0-4.pdf (accessed on 28 January 2022).
31. Yang, N.; Li, D.; Zhang, J.; Xi, Y. Model predictive controller design and implementation on FPGA with application to motor servo system. *Control Eng. Pract.* **2014**, *20*, 1229–1235. [CrossRef]
32. Ling, K.V.; Wu, B.F.; Maciejowski, J.M. Embedded Model Predictive Control (MPC) using a FPGA. In Proceedings of the IFAC Proceedings Volumes, Seoul, Korea, 6–11 July 2008; Volume 41, pp. 15250–15255.
33. Xilinx Aerospace and Defence. Available online: https://www.xilinx.com/applications/aerospace-and-defense/milcom.html (accessed on 28 January 2022).
34. Osornio-Rios, R.A.; Romero-Troncoso, R.; Herrera-Ruiz, G.; Castaneda-Miranda, R. The application of reconfigurable logic to high speed CNC milling machines controllers. *Control. Engneering Pract.* **2007**, *16*, 674–684. [CrossRef]
35. Puton, P. Opracowanie Systemu Cyfrowego Realizującego Sterownik PLC w Oparciu o układy FPGA. Implementacja Oprogramowania Narzędziowego do Realizacji Programów Wykonawczych. Master's Thesis, The Faculty of Electronics and Information Technology on Warsaw University of Technology, Warsaw, Poland, 2018 .
36. Ciganek, J.; Kocur, M.; Kozak, S. Hardware Realization of Advanced Controller Design Methods using FPGA. *IFAC-PapersOnLine* **2016**, *49*, 163–168. [CrossRef]
37. Petko, M.; Karpiel, G.; Uhl, T. Implementacja algorytmów sterowania w układach FPGA na przykładzie robota równoleglego. *Pomiary Autom. Kontrola* **2006**, *5*, 27–30.
38. Ren, Z.; Skjetne, R.; Gao, Z. A Crane Overload Protection Controller for Blade Lifting Operation Based on Model Predictive Control. *Energies* **2019**, *12*, 50. [CrossRef]
39. Zafra, E.; Vazquez, S.; Guzman Miranda, H.; Sanchez, J.A.; Marquez, A.; Leon, J.I.; Franquelo, L.G. Efficient FPSoC Prototyping of FCS-MPC for Three-Phase Voltage Source Inverters. *Energies* **2020**, *13*, 1074. [CrossRef]
40. Campos-Mercado, E.; Mendoza-Santos, E.F.; Torres-Muñoz, J.A.; Román-Hernández, E.; Moreno-Oliva, V.I.; Hernández-Escobedo, Q.; Perea-Moreno, A.J. Nonlinear Controller for the Set-Point Regulation of a Buck Converter System. *Energies* **2021**, *14*, 5760. [CrossRef]

41. Coric, S.; Leeser, M.; Miller, E.; Trepanier, M. Parallel-Beam Backprojection: An FPGA Implementation Optimized for Medical Imaging. *J. Vlsi Signal Process. Syst. Signal Image Video Technol.* **2002**, *39*, 295–311.
42. Jiang, R.M.; Crookes, D. FPGA Implementation of 3D Discrete Wavelet Transform for Real-Time Medical Imaging. In Proceedings of the 2007 18th European Conference on Circuit Theory and Design, Seville, Spain, 26–30 August 2007; pp. 519–522.
43. Allaire, F.C.J.; Tarbouchi, M.; Labonte, G.; Fusina, G. FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning. *J. Intell. Robot. Syst.* **2008**, *54*, 495–510. [CrossRef]
44. Thielemans, S.; Vyncke, T.J.; Jacxsens, M.; Melkebeek, J.A. FPGA implementation of online finite-set model based predictive control for power electronics. In Proceedings of the 2011 Workshop on Predictive Control of Electrical Drives and Power Electronics, Monachium, Niemcy, 14–15 October 2011 2011 ; pp. 63–69.
45. Lan, J.; Li, D.; Xi, Y. Implementation of Dynamic Matrix Control on FPGA. In Proceedings of the 29th Chinese Control Conference, Beijing, China, 29–31 July 2010; pp. 5970–5974.
46. Trabelsi, C.; Meftali, S.; Dekeyser, J.L. Distributed control for reconfigurable FPGA systems: A high-level design approach. In Proceedings of the 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), York, UK, 9–11 July 2012; pp. 1–8. [CrossRef]
47. Lee, S.; Son, C.; Jang, H. Distributed and parallel real-time control system equipped FPGA-Zynq and EPICS middleware. In Proceedings of the 2016 IEEE-NPSS Real Time Conference (RT), Padova, Italy, 6–10 June 2016; pp. 1–4. [CrossRef]
48. Bukowiec, A.; Mróz, P. An FPGA synthesis of the distributed control systems designed with Petri nets. In Proceedings of the 2012 IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA), Liverpool, UK, 13–14 December 2012; pp. 1–6. [CrossRef]
49. Ping, Z.; Gao, Z. An FPGA-based digital control and communication module for space power management and distribution systems. In Proceedings of the 2005, American Control Conference, Portland, OR, USA, 8–10 June 2005; Volume 7, pp. 4941–4946. [CrossRef]
50. Pan, T.; Zhang, G.; Xu, Y.; Song, Z.; Wang, J. Design of Multichannel High-speed Synchronous Data Acquisition System Based on Multi-FPGA and Distributed Control Strategy. In Proceedings of the 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, 12–14 October 2018; pp. 1639–1643. [CrossRef]