Wydział Elektroniki i Technik Informacyjnych

Politechnika Warszawska

Rozprawa doktorska

Projektowanie systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej i struktury FPGA

mgr inż. Andrzej Wojtulewicz

Warszawa, 2020

Spis treści

Streszczenie 5

Abstract 7

Rozdział 1. Wstęp 9

1.1. Cel, teza i zakres rozprawy 9
1.2. Układ rozprawy 12

Rozdział 2. Algorytmy regulacji predykcyjnej 15

2.1. Klasyczna regulacja PID 15
2.2. Zasada regulacji predykcyjnej 16
2.3. Algorytm DMC 23
2.3.1. Predykcja w algorytmie DMC 23
2.3.2. Algorytm DMC w wersji numerycznej 25
2.3.3. Algorytm DMC w wersji analitycznej 26
2.4. Algorytm GPC 29
2.4.1. Predykcja w algorytmie GPC 30
2.4.2. Algorytm GPC w wersji numerycznej 32
2.4.3. Algorytm GPC w wersji analitycznej 33
2.5. Zalety algorytmów regulacji predykcyjnej oraz przykładowe zastosowania . 35

Rozdział 3. Implementacja algorytmów regulacji predykcyjnej w systemach sterujących: przegląd istniejących rozwiązań 39

3.1. Wymagania 39
3.2. Sterowniki programowalne 39
3.3. Sterowniki przemysłowe 49
3.4. Mikrokontrolery 53

Rozdział 4. Układy FPGA 59

4.1. Geneza układów FPGA 59
4.2. Struktura układów FPGA 63

4.3. Programowanie układów FPGA 66
4.4. Typowe zastosowania układów FPGA 81

Rozdział 5. System do regulacji predykcyjnej oparty na strukturze FPGA: warstwa programowa 85

5.1. Ogólna struktura systemu 85
5.2. Przykładowa implementacja procesora NIOS II 88
5.3. Implementacja warstwy sprzętowej VHDL 94
5.3.1. Sprzętowy akcelerator analitycznego algorytmu DMC 100
5.3.2. Sprzętowy akcelerator analitycznego algorytmu GPC 102
5.4. Implementacja warstwy programowej C 107
5.4.1. Inicjalizacja funkcji sprzętowych 107
5.4.2. Ogólna organizacja pracy systemu 110
5.4.3. Algorytmy analityczne w wersji oszczędnej 112
5.4.4. Algorytmy analityczne (w wersji pełnej) i numeryczne 116

Rozdział 6. System do regulacji predykcyjnej oparty na strukturze FPGA: warstwa sprzętowa 131

6.1. Płyta FPGA 131
6.2. Projekt platformy interfejsów 132
6.3. Wykonanie i uruchomienie platformy interfejsów 136

Rozdział 7. Przygotowanie stanowisk testowych 149

7.1. Stanowisko grzejąco-chłodzące 149
7.2. Stanowisko świetlne 159
7.3. Serwomechanizm 163

Rozdział 8. Wyniki eksperymentów 173

8.1. Stanowisko grzejąco-chłodzące 176
8.2. Stanowisko świetlne 187
8.3. Serwomechanizm 193

Rozdział 9. Podsumowanie i wnioski 201

Bibliografia 205

Streszczenie

Algorytmy regulacji predykcyjnej (ang. Model Predictive Control, MPC) są często stosowane w przemysłowych systemach automatyki zaawansowanej, np. w przemyśle chemicznym, petrochemicznym, spożywczym i papierniczym. Przykładowe procesy to reaktory chemiczne oraz kolumny destylacyjne. Cechą szczególną takich procesów jest dość wolna dynamika, co oznacza, że okres próbkowania algorytmu regulacji jest rzędu sekund lub nawet minut. Do implementacji stosuje się programowalne sterowniki logiczne lub sterowniki przemysłowe. Rozwój elektroniki, dostępność szybkich i tanich mikrokontrolerów, umożliwia zastosowanie zaawansowanych algorytmów regulacji do szybkich procesów, charakteryzujących się krótkimi okresami próbkowania, rzędu ułamków sekund, dziesiątek milisekund lub nawet pojedynczych milisekund. Przykładowe zastosowania to sterowanie silnikami elektrycznymi i spalinowymi, sterowanie pojazdami autonomicznymi, sterowanie bezzałogowymi statkami powietrznymi lub pływającymi.

Przedmiotem rozprawy jest programowanie systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej oraz bezpośrednio programowalne macierze bramek układów (ang. Field Programmable Gate Arrays, FPGA), które do tej pory są znacznie mniej popularne jako platformy obliczeniowe automatyki niż mikrokontrolery. Zaproponowano spójną metodykę projektowania systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej i struktury FPGA, które umożliwiają sterowanie szybkich systemów dynamicznych, charakteryzujących się krótkimi okresami próbkowania. Proponowane podejście składa się z warstwy sprzętowej (projekt układów elektronicznych) oraz programowej (implementacja i testowanie), które są ze sobą ściśle związane.

Praca podzielona jest na dziewięć rozdziałów. Pierwszy rozdział jest wprowadzeniem, zdefiniowano obszar badawczy, cel, tezę i zakres rozprawy. W drugim rozdziale krótko opisano wykorzystywane algorytmy regulacji predykcyjnej: algorytm Dynamic Matrix Control (DMC) oraz Generalized Predictive Control (GPC), oba w wersji analitycznej oraz numerycznej. W trzecim rozdziale zdefiniowano wymagania stawiane algorytmom regulacji stosowanym w systemach wbudowanych, a następnie omówiono cechy szczególne trzech platform sprzętowych, służących do implementacji algorytmów regulacji: programowalnych sterowników logicznych, sterowników przemysłowych oraz mikrokontrolerów. W czwartym rozdziale przedstawiono układy FPGA: genezę powstania, strukturę oraz programowanie. W piątym rozdziale omówiono warstwę programową opracowanego systemu do regula-

cji predykcyjnej: koncepcję systemu i jego ogólną strukturę, a także implementację warstw: procesora NIOS II, VHDL oraz warstwy programowej C. W szóstym rozdziale omówiono warstwę sprzętową opracowanego systemu, w tym przedstawiono szczegóły dotyczące zaprojektowanej płytki interfejsowej, projekt interfejsów oraz aspekty praktyczne dotyczące wykonania i uruchomienia interfejsów. W siódmym rozdziale opisano wykorzystywane procesy dynamiczne: stanowisko grzejąco-chłodzące, stanowisko świetlne oraz serwomechanizm. W ósmym rozdziale pracy przedstawiono i omówiono wyniki eksperymentów. W szczególności, zbadano wpływ rodzaju i parametrów algorytmów regulacji na jakość regulacji i czas obliczeń, koniecznych w każdej chwili próbkowania. Rozdział dziewiąty podsumowuje pracę, określono możliwe kierunki rozwoju.

Słowa kluczowe: regulacja zaawansowana, algorytmy regulacji predykcyjnej (MPC), bezpośrednio programowalne macierze bramek układów (FPGA)

Abstract

Model Predictive Control (MPC) algorithms are frequently used in industrial advanced control systems, e.g. in chemical and petrochemical engineering, in food industry and in paper industry. Chemical reactors and distillation columns are example controlled process. Such processes are characterised by slow dynamic properties, which means that sampling times of control algorithms' are of the order of seconds or even minutes. Programmable logic controllers or specialised industrial controllers are used for implementation. The development of electronics and the availability of fast and cheap microcontrollers allow the use of advanced control algorithms for fast processes, characterised by short sampling periods, which are of the order of fractions of a second, tens of milliseconds or even single milliseconds. Example applications include: control of electric and combustion engines, control of autonomous vehicles, control of unmanned aerial vehicles or vessels.

The subject of the dissertation is programming of control systems using model predictive control algorithms and Field Programmable Gate Arrays (FPGAs), which are far less popular as hardware platforms for implementation of advanced control methods than microcontrollers. A coherent methodology for designing control systems using MPC algorithms and FPGA structures is proposed that makes it possible to control fast dynamic systems characterised by short sampling periods. The proposed approach consists of a hardware layer (design of electronic circuits) and a software laey (implementation and testing), which are closely related.

This work is divided into nine chapters. The first chapter is an introduction, the research area, the objective and the thesis of the dissertation are defined. The second chapter briefly describes the MPC algorithms used: Dynamic Matrix Control (DMC) and Generalized Predictive Control (GPC), both in analytical and numerical versions. The third chapter defines the requirements for control algorithms used in embedded systems and then discusses the specific features of the three hardware platforms used to implement the control algorithms: programmable logic controllers, industrial controllers and microcontrollers. The fourth chapter presents FPGAs: their origin, structure and programming. The fifth chapter discusses the program layer of the developed system: its concept, its overall structure, as well as the implementation of layers: the NIOS II processor, VHDL and the C programming layer. The sixth chapter discusses the hardware layer of the developed system: the details of the designed interface board, interface platform design and practical aspects of the implementation and running of the interfaces are presented.

The seventh chapter describes the dynamic processes used: a heating and cooling process, a lighting process and a servomechanism. The eighth chapter presents and discusses the results of experiments. In particular, the influence of the type and parameters of the control algorithms on the control quality and the calculation time necessary at every sampling time is examined. Chapter nine summarises the dissertation and specifies possible development directions.

Keywords: advanced control, Model Predictive Control (MPC) algorithms, Field Programmable Gate Arrays (FPGA)

Rozdział 1

Wstęp

1.1. Cel, teza i zakres rozprawy

Algorytmy regulacji predykcyjnej (ang. Model Predictive Control, MPC) [21, 76, 107] są zaliczane do tzw. zaawansowanych technik regulacji. W przeciwieństwie do klasycznych, prostych, algorytmów proporcjonalno-całkująco-różniczkujących (ang. Proportional-Integrate-Derivative, PID), algorytmy regulacji predykcyjnej wyznaczają optymalne wartości zmiennych decyzyjnych (sygnałów sterujących) w wyniku cyklicznej optymalizacji pewnego wskaźnika jakości, który zazwyczaj uwzględnia prognozowane (predykcja) uchyby regulacji. Predykcja zmiennych procesowych wykonywana jest na bieżąco (ang. on-line) przy wykorzystaniu dynamicznego modelu sterowanego procesu. Dzięki takiemu sformułowaniu zasady działania, algorytmy regulacji predykcyjnej umożliwiają uzyskanie bardzo dobrej jakości regulacji. Model służący do predykcji umożliwia dobranie sekwencji sterującej, która uwzględnia wszystkie właściwości procesu. W szczególności, algorytmy regulacji predykcyjnej świetnie radzą sobie z regulacją procesów wielowymiarowych, o silnych sprzężeniach skrośnych, a także obiektów nieminimalnofazowych. Algorytmy omawianej klasy zapewniają jakość regulacji znacznie lepszą niż struktury PID, które przeznaczone są typowo do regulacji procesów jednowymiarowych, bez opóźnienia. Co więcej, algorytmy regulacji predykcyjnej umożliwiają w bardzo łatwy sposób uwzględnienie w zadaniu optymalizacji wszelkich ograniczeń sygnałów procesu. Spełnienie niektórych ograniczeń przez algorytm PID może być trudne lub wręcz niemożliwe.

Dzięki omówionym zaletom algorytmy regulacji predykcyjnej odniosły niekwestionowany sukces w praktyce [90]. Są powszechnie stosowane w przemyśle chemicznym [91], petrochemicznym [120], spożywczym [19] i papierniczym [52]. W wymienionych sytuacjach algorytmy regulacji predykcyjnej są zaimplementowane przy wykorzystaniu programowalnych sterowników logicznych (ang. Programmable Lo-

gic Controller, PLC) lub sterowników przemysłowych. Jest to rozwiązanie oczywiste, ponieważ wymienione procesy są dość wolne, okresy interwencji (próbkowania) algorytmu regulacji są rzędu sekund lub nawet minut.

W ostatnich latach można obserwować bardzo szybki rozwój układów mikroprocesorowych oraz bezpośrednio programowalnych macierzy bramek (ang. Field Programmable Gate Array, FPGA). Szybkość, możliwości obliczeniowe oraz zasoby (pamięć, porty wejścia-wyjścia, interfejsy komunikacyjne) tych platform sprzętowych są wystarczające do implementacji algorytmów regulacji predykcyjnej dla szybkich procesów. Jako przykłady aktualnych zastosowań tego typu systemów można wymienić:

a) sterowanie wtryskiem silnika spalinowego [8, 28],
b) sterowanie silnikami elektrycznymi [34, 38],
c) sterowanie samochodami autonomicznymi [93, 75],
d) sterowanie w celu omijania przeszkód pojazdów autonomicznych [6],
e) sterowanie w celu zachowania bezpiecznej odległości pojazdów [96],
f) sterowanie bezzałogowym statkiem powietrznym [13, 96],
g) sterowanie bezzałogowym statkiem pływającym [97],
h) sterowanie systemami aktywnej redukcji drgań w konstrukcjach mechanicznych [47].

Warto podkreślić, że wymienione procesy należą do zupełnie innej klasy niż klasyczne, procesy przemysłowe. W systemach wbudowanych zachodzi konieczność zapewnienia znacznie krótszego okresu interwencji regulatora niż w klasycznych procesach przemysłowych. W zależności od zastosowania, musi on być rzędu ułamków sekund, dziesiątek milisekund lub nawet pojedynczych milisekund. Zgodnie z aktualnymi prognozami liczba systemów wbudowanych będzie się gwałtownie zwiększała. Oznacza to, że zapotrzebowanie na skuteczne algorytmy regulacji takich systemów będzie również rosło.

Dopiero od niedawna podejmuje się próby implementacji zaawansowanych algorytmów regulacji, przede wszystkim algorytmów regulacji predykcyjnej, przy wykorzystaniu nowoczesnych mikrokontrolerów. Z reguły algorytmy regulacji implementowane niezależnie dla konkretnych procesów. Alternatywą jest wykorzystanie specjalizowanych, bardzo rozbudowanych, systemów informatycznych do automatycznej generacji kodu dla wybranej platformy sprzętowej. Jako przykłady rozwiązań tego typu można podać: ACADO [54], Multi-Parametric Toolbox [53] oraz AutoMATiC [23], który został opracowany w Instytucie Automatyki i Informatyki

Stosowanej Politechniki Warszawskiej. W systemach wbudowanych wykorzystujących zaawansowane algorytmy regulacji znacznie mniejszą popularnością niż mikrokontrolery cieszą się struktury FPGA. Powyższe spostrzeżenie zachęciło autora niniejszej rozprawy do podjęcia pracy badawczej.

Celem pracy jest zaproponowanie spójnej metodyki projektowania systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej i struktury FPGA, które umożliwiają sterowanie szybkich systemów dynamicznych, charakteryzujących się krótkimi okresami próbkowania (interwencji algorytmu).

Teza pracy jest następująca: **Przedstawiona metodyka umożliwia zaprojektowanie systemów sterujących szybkich procesów czasu rzeczywistego, wymagających bardzo krótkich okresów próbkowania, rzędu milisekund.** Proponowane podejście do projektowania systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej i struktury FPGA składa się z warstwy sprzętowej (projekt układów elektronicznych) oraz programowej (implementacja i testowanie), które są ze sobą ściśle związane.

W literaturze, np. [11, 66, 122, 74, 73], opisywane są przykłady zastosowania układów FPGA do implementacji algorytmów regulacji predykcyjnej, to jednak mają one wady, wymienić. Najważniejsze z nich są następujące:

1. Opisy dotyczą głównie na algorytmach, ale często brak jest szczegółowego opisu realizacji programowej i sprzętowej, w tym interfejsu jednostka sterująca (układ FPGA) – regulowany proces.
2. Projektowanie akceleratorów sprzętowych, których celem jest przyspieszenie obliczeń, jest zwykle przedstawiane dla konkretnych przykładów, brak jest większej ilości szczegółów dotyczących implementacji.
3. Brak standardowych interfejsów pozwalających podłączyć typowe urządzenia występujące w przemyśle.
4. Weryfikacja działania algorytmów odbywa się dla bardzo prostych przykładów, często dla procesów symulowanych.

Warto podkreślić, że autor rozprawy samodzielnie zaprojektował i wykonał wszystkie układy elektroniczne oraz opracował oryginalną implementację czterech algorytmów regulacji predykcyjnej, a mianowicie: algorytmu Dynamic Matrix Control (DMC) w wersji analitycznej (bez bieżącej optymalizacji), algorytmu DMC w wersji numerycznej (z bieżącą optymalizacją kwadratową), algorytmu Generalized Predictive Control (GPC) w wersji analitycznej oraz algorytmu GPC w wersji

numerycznej. Wszystkie algorytmy wykorzystują do predykcji liniowy model procesu: w algorytmie DMC jest to odpowiedź skokowa, w algorytmie GPC jest to równanie różnicowe (transmitancja dyskretna). Zastosowanie wymienionych modeli oznacza, że w oryginalnej wersji algorytm DMC może być zastosowany tylko do procesów stabilnych bez całkowania, algorytm GPC nie ma takich ograniczeń. Z drugiej jednak strony, ogromną zaletą algorytmu DMC jest bardzo prosta identyfikacja: do otrzymania modelu należy jedynie zarejestrować odpowiedź skokową procesu. Aby przedstawić uniwersalne rozwiązanie opracowany system oferuje:

1. Algorytmy DMC i GPC w klasycznej wersji analitycznej oszczędnej. W celu przyspieszenia obliczeń zaprojektowano specjalizowane akceleratory sprzętowe.
2. Bardziej złożone obliczeniowo wersje numeryczne tych algorytmów, które umożliwiają dokładne uwzględnienie istniejących ograniczeń sygnałów sterujących procesów i/lub ograniczeń prognozowanych sygnałów wyjściowych. Do implementacji algorytmów numerycznych wykorzystano procesor NIOS II.

Wszystkie algorytmy dostępne do w wersji wielowymiarowej (procesy o wielu wejściach i wielu wyjściach).

W trakcie prac zaprojektowano oraz zbudowano dwa oryginalne stanowiska laboratoryjne, które służą do testowania opracowanego systemu sprzętowo-programowego: stanowisko termiczne [116, 115] (razem z dr. inż. Patrykiem Chabrem) oraz stanowisko świetlne. Pierwsze z wymienionych stanowisk (zbudowano kilka jego kopii) wykorzystywane jest w trakcie ćwiczeń laboratoryjnych z kilku przedmiotów prowadzonych w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej (Diagnostyka procesów przemysłowych, Projektowanie układów sterowania (projekt grupowy), Systemy automatyki DCS i SCADA, Systemy mikroprocesorowe w sterowaniu).

1.2. Układ rozprawy

Praca składa się z dziewięciu rozdziałów oraz bibliografii.

W drugim rozdziale pracy przypomniano zasadę regulacji predykcyjnej, a następnie opisano algorytmy DMC oraz GPC w wersji analitycznej oraz numerycznej. W możliwie zwięzły sposób przedstawiono wszystkie wzory niezbędne do implementacji tych algorytmów, wykorzystywane w dalszej części pracy. Omówiono najważniejsze właściwości (w tym zalety) algorytmów regulacji predykcyjnej oraz podano wiele przykładów ich wykorzystania.

W trzecim rozdziale pracy zdefiniowano wymagania stawiane systemom wbudowanym, a następnie omówiono cechy szczególne trzech platform sprzętowych, służących do implementacji algorytmów regulacji, w tym algorytmów regulacji predykcyjnej, tzn.: programowalnych sterowników logicznych, sterowników przemysłowych oraz mikrokontrolerów. Dwie pierwsze platformy stosowane są w klasycznych systemach automatyki przemysłowej, trzecia z nich – w systemach wbudowanych. W szczególności, zwrócono uwagę na możliwość implementacji algorytmów regulacji procesów charakteryzujących się krótkimi okresami próbkowania, rzędu milisekund. Podano przykłady zastosowań wymienionych platform sprzętowych do regulacji różnych procesów.

W czwartym rozdziale pracy przedstawiono układy FPGA: genezę powstania, strukturę oraz programowanie. Podano przykłady zastosowań układów FPGA w różnych dziedzinach.

W piątym rozdziale pracy omówiono warstwę programową opracowanego systemu do regulacji predykcyjnej, wykorzystującego układy FPGA. Przedstawiono: koncepcję systemu i jego ogólną strukturę. Następnie opisano implementację: warstwy procesora NIOS II, warstwy VHDL, warstwy programowej C.

W szóstym rozdziale pracy omówiono warstwę sprzętową opracowanego systemu do regulacji predykcyjnej, wykorzystującego układy FPGA. Opisano wykorzystywaną płytkę rozwojową z układem FPGA. Następnie przedstawiono szczegóły dotyczące zaprojektowanej płytki interfejsowej, projekt platformy interfejsów oraz aspekty praktyczne dotyczące wykonania i uruchomienia platformy interfejsów.

W siódmym rozdziale pracy opisano wykorzystywane procesy dynamiczne: stanowisko termiczne, stanowisko świetlne oraz serwomechanizm. Jak już wspomniano, dwa pierwsze stanowiska laboratoryjne są oryginalną konstrukcją autora.

W ósmym rozdziale pracy przedstawiono i omówiono wyniki eksperymentów, polegających na testowaniu jakości regulacji i czasów obliczeń rozważanych algorytmów regulacji predykcyjnej w zastosowaniu do trzech stanowisk laboratoryjnych. W szczególności, zbadano wpływ rodzaju i parametrów algorytmów na czas obliczeń, koniecznych w każdej chwili próbkowania.

Rozdział dziewiąty podsumowuje pracę. Krótko wymieniono najważniejsze cechy opracowanego rozwiązania sprzętowo-programowego do implementacji algorytmów regulacji predykcyjnej przy wykorzystaniu układów FPGA. Omówiono zakres zastosowań systemu oraz możliwe kierunki rozwoju.

Rozdział 2

Algorytmy regulacji predykcyjnej

2.1. Klasyczna regulacja PID

Regulacja procesu dynamicznego to działanie, który może być wykonywane ręcznie lub automatycznie. Przykładem ręcznej regulacji temperatury w pomieszczeniu może być ustawianie zaworu dopływu ciepłej wody do kaloryfera. Dzięki temu w pokoju robi się cieplej lub chłodniej. W rozważanym przykładzie człowiek działa według bardzo prostego algorytmu: gdy w pokoju jest zbyt gorąco, należy nieco zakręcić zawór, natomiast gdy jest zbyt zimno, należy nieco odkręcić zawór. Po kilku modyfikacjach położenia zaworu udaje się osiągnąć oczekiwaną temperaturę powietrza w pomieszczeniu. Proces tej regulacji bardzo łatwo można zautomatyzować instalując regulator (element zastępujący człowieka i realizujący algorytm regulacji w sposób automatyczny). Należy zastosować element pomiarowy – czujnik temperatury, a także element wykonawczy – zawór sterowany. W przypadku zastosowania automatycznego układu regulacji człowiek zadaje wartość temperatury, która ma panować w pomieszczeniu, jako wartość wejściową dla regulatora. Druga wartość wejściowa regulatora to aktualna temperatura, mierzona za pomocą czujnika. Różnica tych dwóch wartości to uchyb regulacji, który wyznacza kierunek oraz wartość zmiany sygnału sterującego zaworem. W najprostszym podejściu sygnał sterujący jest proporcjonalny do aktualnej wartości uchybu, co prowadzi do zastosowania regulatora proporcjonalnego. Można go uzupełnić o człon całkujący i różniczkujący uchyb, co umożliwia zminimalizować przeregulowanie oraz czas osiągnięcia zadanej temperatury. Parametry algorytmu regulacji proporcjonalno-całkująco-inercyjnego (ang. Proportional-Integral-Derivative, PID) mogą być dobierane empirycznie, metodą prób i błędów, ale znacznie bardziej efektywne jest wykorzystanie metod doboru nastaw, np. metody Zieglera-Nicholsa. Warto zwrócić podkreślić, że klasyczny algorytm PID opisany jest pewnym wzorem

analitycznym, służącym do obliczania aktualnego sygnału sterującego na podstawie aktualnego uchybu oraz uchybów z przeszłości.

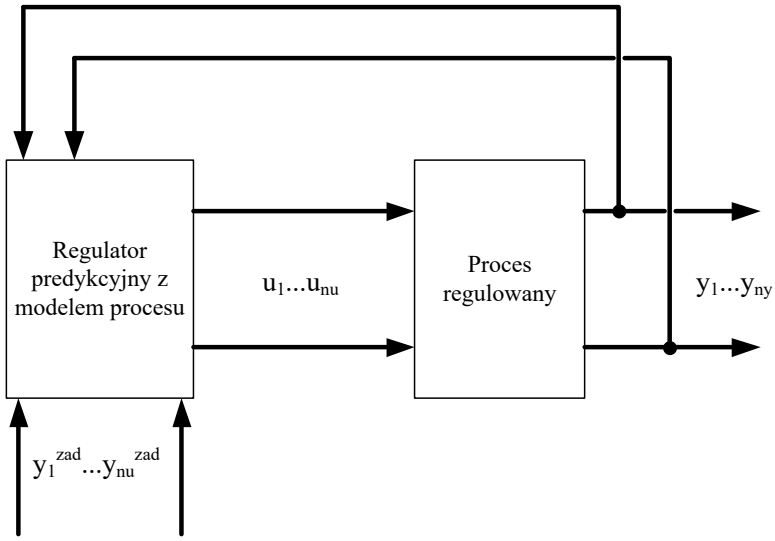
Regulatory PID świetnie pracują w przypadku procesów jednowymiarowych bez właściwości nieminimalnofazowych. Oczywiście, możliwe jest zastosowanie regulatorów PID w przypadku procesów wielowymiarowych, o wielu sygnałach wejściowych i wyjściowych. Klasyczne podejście to zastosowanie zbioru jednopętlowych regulatorów PID, co jest możliwe pod warunkiem, gdy liczba wejść procesu jest równa liczbie wyjść. Po drugie, poszczególne regulatory PID pracują niezależnie, często „przeszkadzając" sobie wzajemnie. Bardziej wyrafinowanym podejściem do regulacji procesu wielowymiarowego jest odsprzęganie, czyli próba eliminacji sprzężeń skrośnych, a następnie zastosowanie zestawu jednopętlowych regulatorów PID. Oczywiście, idealne odsprzęganie jest możliwe tylko w pewnych prostych przypadkach.

2.2. Zasada regulacji predykcyjnej

Algorytmy regulacji predykcyjnej [21, 76, 107] są interesującą alternatywą dla regulatorów klasycznych. Ich sposób działania jest inny. Wartość sygnału sterującego (w przypadku wielowymiarowym wartości sygnałów sterujących) są cyklicznie obliczane w każdej dyskretnej chwili próbkowania algorytmu w wyniku minimalizacji funkcji kosztów, która określa przewidywaną (prognozowaną) jakość regulacji. Do predykcji zachowania się regulowanego procesu w przyszłości stosuje się model matematyczny procesu. W porównaniu z klasycznymi regulatorami PID oczywista jest pierwsza zaleta regulacji predykcyjnej, a mianowicie możliwość skutecznej regulacji procesów wielowymiarowych. Warto podkreślić, że liczba wejść i wyjść regulowanego procesu może być różna, co dla zestawu klasycznych jednopętlowych regulatorów PID jest problemem.

Strukturę wielowymiarowego układu regulacji z algorytmem regulacji predykcyjnej przedstawiono na rys. 2.1. W ogólności rozważa się proces wielowymiarowy, który posiada n_{u} wejść (sygnałów sterujących), które oznaczamy przez $u = [u_1, \ldots, u_{n_{\mathrm{u}}}]^{\mathrm{T}}$, oraz n_{y} wyjść (sygnałów wyjściowych), które oznaczamy przez $y = [y_1, \ldots, y_{n_{\mathrm{y}}}]^{\mathrm{T}}$. Wektor sygnałów zadanych poszczególnych wyjść procesu ma postać $y^{\mathrm{zad}} = [y_1^{\mathrm{zad}}, \ldots, y_{n_{\mathrm{u}}}^{\mathrm{zad}}]^{\mathrm{T}}$.

W niniejszej pracy rozważa się dyskretne (cyfrowe) algorytmy regulacji. W każdej z dyskretnych chwil $k = 0, 1, 2, \ldots$ wykonywane są następujące czynności: po-



Rys. 2.1. Struktura wielowymiarowego układu regulacji z algorytmem regulacji predykcyjnej

miar wartości sygnałów wyjściowych procesu, obliczenie wartości sygnałów wejściowych i wysłanie ich do procesu, odczekanie na następną chwilę próbkowania. Czas między kolejnymi chwilami dyskretnymi jest nazywany czasem lub okresem próbkowania. Funkcja celu (kosztu), w wyniku minimalizacji której wyznacza się sygnały sterujące w dyskretnej chwili k, ma postać

$$\begin{aligned} J(k) = &\sum_{m=1}^{n_\mathrm{y}} \sum_{p=1}^{N} (y_m^{\mathrm{zad}}(k+p|k) - \hat{y}_m(k+p|k))^2 \psi_m \\ &+ \sum_{n=1}^{n_\mathrm{u}} \sum_{p=0}^{N_\mathrm{u}-1} (\triangle u_n(k+p|k))^2 \lambda_n \end{aligned} \tag{2.1}$$

gdzie $\hat{y}_m(k+p|k)$ oznacza przewidywaną wartość m-tego wyjścia procesu ($m = 1, \ldots, n_\mathrm{y}$) na przyszłą chwilę $k+p$ wyznaczoną w chwili k, $\triangle u_n(k+p|k) = u_n(k+p|k) - u_n(k+p-1|k)$ jest przyrostem sygnału n-tego wejścia procesu ($n = 1, \ldots, n_\mathrm{u}$). Sygnały $\hat{y}_m(k+p|k)$ obliczane są na bieżąco (ang. on-line) przy wykorzystaniu modelu matematycznego procesu. Współczynniki $\lambda_n > 0$ oraz $\psi_m \geqslant 0$ są skalarami

i służą do określania wpływu poszczególnych elementów funkcji celu. Pierwszy człon funkcji celu uwzględnia prognozowane uchyby regulacji na horyzoncie predykcji N. Drugi człon funkcji celu jest członem kary, ma on na celu ograniczyć zbyt duże zmiany sygnałów sterujących. Zakłada się, że tylko sygnały sterujące od chwili k do $k+N_{\mathrm{u}}-1$ są zmienne, gdzie N_{u} jest horyzontem sterowania (a więc $u_n(k+p|k) = u_n(k+N_{\mathrm{u}}-1|k)$ dla $p \geqslant N_{\mathrm{u}}$, $n = 1, \ldots, n_{\mathrm{u}}$). Zazwyczaj $N_{\mathrm{u}} < N$, co prowadzi do redukcji zmiennych decyzyjnych algorytmu optymalizacji. Ma on długość $n_{\mathrm{u}}N_{\mathrm{u}}$ i strukturę

$$\triangle \boldsymbol{U}(k) = \begin{bmatrix} \triangle u(k|k) \\ \vdots \\ \triangle u(k+N_{\mathrm{u}}-1|k) \end{bmatrix} \tag{2.2}$$

gdzie

$$\triangle u(k+p|k) = \begin{bmatrix} \triangle u_1(k+p|k) \ldots \triangle u_{n_{\mathrm{u}}}(k+p|k) \end{bmatrix}^{\mathrm{T}} \tag{2.3}$$

jest wektorem o długości n_{u}. Pomimo tego, że w każdej dyskretnej chwili próbkowania wyznacza się cały wektor przyrostów sygnałów sterujących na całym horyzoncie sterowania (2.2), do regulacji wykorzystuje się jedynie n_{u} pierwszych elementów tego wektora, czyli przyrosty dla aktualnej chwili k

$$u(k) = \triangle u(k|k) + u(k-1) \tag{2.4}$$

W kolejnej chwili dyskretnej, $k+1$, następuje powtórzenie całej procedury, czyli aktualizacja pomiarów sygnałów wyjściowych, optymalizacja przyrostów sygnałów sterujących itd. Konieczność wyznaczania całego wektora (2.2) będzie zrelaksowana w wersji analitycznej algorytmów regulacji predykcyjnej.

Kwadratowa funkcja kosztu (2.1) jest najczęściej stosowana w regulacji predykcyjnej. Znacznie mniej popularna jest suma wartości bezwzględnych prognozowanych uchybów regulacji $(y_m^{\mathrm{zad}}(k+p|k) - \hat{y}_m(k+p|k))$ na horyzoncie predykcji lub uwzględnienie jedynie największego z prognozowanych uchybów [76].

Drugą zaletą algorytmów regulacji predykcyjnej, oprócz skutecznej regulacji procesów wielowymiarowych, jest możliwość uwzględnienia sygnałów sterujących i prognozowanych sygnałów wyjściowych procesu. Ograniczenia są po prostu częścią

zadania optymalizacji, które rozwiązywane jest w każdej chwili próbkowania k

$$\min_{\triangle \boldsymbol{U}(k)} \left\{ J(k) = \sum_{m=1}^{n_\mathrm{y}} \sum_{p=1}^{N} (y_m^{\mathrm{zad}}(k+p|k) - \hat{y}_m(k+p|k))^2 \psi_m + \sum_{n=1}^{n_\mathrm{u}} \sum_{p=0}^{N_\mathrm{u}-1} (\triangle u_n(k+p|k))^2 \lambda_n \right\} \tag{2.5}$$

dla następujących ograniczeń

$$\begin{aligned} u_n^{\min} \leqslant \quad & u_n(k+p|k) \leqslant \quad u_n^{\max}, \text{ dla } p = 0, \ldots, N_\mathrm{u} - 1;\ n = 1, \ldots, n_\mathrm{u} \\ \triangle u_n^{\min} \leqslant & \triangle u_n(k+p|k) \leqslant \triangle u_n^{\max}, \text{ dla } p = 0, \ldots, N_\mathrm{u} - 1;\ n = 1, \ldots, n_\mathrm{u} \\ y_m^{\min} \leqslant \quad & \hat{y}_m(k+p|k) \leqslant \quad y_m^{\max}, \text{ dla } p = 1, \ldots, N; \qquad m = 1, \ldots, n_\mathrm{y} \end{aligned} \tag{2.6}$$

gdzie $u_n^{\min}$ i $u_n^{\max}$ definiują minimalne i maksymalne ograniczenia wartości n-tego sygnału sterującego, $\triangle u_n^{\min}$ i $\triangle u_n^{\max}$ oznaczają minimalne i maksymalne ograniczenia przyrostu (zmiany) n-tego sygnału sterującego, $y_m^{\min}$, $y_m^{\max}$ definiują minimalne i maksymalne ograniczenia wartości prognozowanej m-tego sygnału wyjściowego.

Jeżeli uwzględnione są ograniczenia wartości prognozowanych sygnałów wyjściowych, nieuniknione błędy modelu oraz zakłócenia prowadzą do powstania groźnego zjawiska braku zbioru dopuszczalnego rozwiązań zadania optymalizacji. Aby rozwiązać ten problem stosuje się tzw. miękkie ograniczenia wyjściowe, które mogą być chwilowo przekraczane, co prowadzi do powiększenia zbioru dopuszczalnego rozwiązań [108]. Wielkość przekroczeń ograniczeń twardych jest minimalizowana w każdej chwili próbkowania (są to dodatkowe zmienne problemu optymalizacji).

Możliwe jest zastosowanie bardziej zwięzłego opisu matematycznego zadania optymalizacji algorytmu regulacji predykcyjnej, a mianowicie wykorzystanie notacji macierzowo-wektorowej. Wektor wartości przyszłych sygnałów sterujących, o długość $n_\mathrm{u}N_\mathrm{u}$, ma strukturę odpowiadającą wektorowi zmiennych decyzyjnych (2.2)

$$\boldsymbol{U}(k) = \begin{bmatrix} u(k|k) \\ \vdots \\ u(k+N_\mathrm{u}-1|k) \end{bmatrix} \tag{2.7}$$

gdzie

$$u(k+p|k) = [u_1(k+p|k) \ldots u_{n_\mathrm{u}}(k+p|k)]^\mathrm{T} \tag{2.8}$$

jest wektorem o długości n_u.

Wektory trajektorii prognozowanej sygnałów wyjściowych oraz trajektorii zadanej, o długość $n_{\mathrm{y}}N$, mają postać

$$\hat{\boldsymbol{Y}}(k) = \begin{bmatrix} \hat{y}(k+1|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix}, \ \boldsymbol{Y}^{\mathrm{zad}}(k) = \begin{bmatrix} y^{\mathrm{zad}}(k+1|k) \\ \vdots \\ y^{\mathrm{zad}}(k+N|k) \end{bmatrix} \tag{2.9}$$

gdzie

$$\hat{y}(k+p|k) = \left[\hat{y}_1(k+p|k) \dots \hat{y}_{n_{\mathrm{y}}}(k+p|k)\right]^{\mathrm{T}} \tag{2.10}$$

$$y^{\mathrm{zad}}(k+p|k) = \left[y_1^{\mathrm{zad}}(k+p|k) \dots y_{n_{\mathrm{y}}}^{\mathrm{zad}}(k+p|k)\right]^{\mathrm{T}} \tag{2.11}$$

są wektorami o długości n_{y}. Parametry pierwszego członu funkcji kryterialnej zapisuje się w postaci macierzy diagonalnej o wymiarach $Nn_{\mathrm{y}} \times Nn_{\mathrm{y}}$

$$\boldsymbol{\Psi} = \mathrm{diag}(\Psi_1, \dots, \Psi_N) \tag{2.12}$$

gdzie

$$\Psi_p = \mathrm{diag}(\psi_1, \dots, \psi_{n_{\mathrm{y}}}) \tag{2.13}$$

jest macierzą o wymiarach $n_{\mathrm{y}} \times n_{\mathrm{y}}$, $p = 1, \dots, N$. Parametry drugiego członu funkcji kryterialnej zapisuje się w postaci macierzy diagonalnej o wymiarach $N_{\mathrm{u}}n_{\mathrm{u}} \times N_{\mathrm{u}}n_{\mathrm{u}}$

$$\boldsymbol{\Lambda} = \mathrm{diag}(\Lambda_0, \dots, \Lambda_{N_{\mathrm{u}}-1}) \tag{2.14}$$

gdzie

$$\Lambda_p = \mathrm{diag}(\lambda_1, \dots, \lambda_{n_{\mathrm{u}}}) \tag{2.15}$$

jest macierzą o wymiarach $n_{\mathrm{u}} \times n_{\mathrm{u}}$, $p = 0, \dots, N_{\mathrm{u}} - 1$.

Norma kwadratowa wektora x oznaczona jest przez $\|x\|^2_{\boldsymbol{\Theta}} = x^{\mathrm{T}}\Theta x$. Zadanie optymalizacji algorytmu regulacji predykcyjnej (2.5) z ograniczeniami (2.6) można wówczas zapisać w zwięzłej postaci

$$\min_{\triangle \boldsymbol{U}(k)} J(k) = \left\| \boldsymbol{Y}^{\mathrm{zad}}(k) - \hat{\boldsymbol{Y}}(k) \right\|^2_{\boldsymbol{\Psi}} + \left\| \triangle \boldsymbol{U}(k) \right\|^2_{\boldsymbol{\Lambda}} \tag{2.16}$$

z następującymi ograniczeniami

$$\begin{array}{rcccl} \triangle \boldsymbol{U}^{\min} & \leqslant & \triangle \boldsymbol{U}(k) & \leqslant & \triangle \boldsymbol{U}^{\max} \\ \boldsymbol{U}^{\min} & \leqslant & \boldsymbol{U}(k-1) + \boldsymbol{J}\triangle \boldsymbol{U}(k) & \leqslant & \boldsymbol{U}^{\max} \\ \boldsymbol{Y}^{\min} & \leqslant & \hat{\boldsymbol{Y}}(k) & \leqslant & \boldsymbol{Y}^{\max} \end{array} \tag{2.17}$$

Macierz $\boldsymbol{J}$, o wymiarach $N_{\mathrm{u}} n_{\mathrm{u}} \times N_{\mathrm{u}} n_{\mathrm{u}}$, zdefiniowana jest jako

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{I}_{n_{\mathrm{u}} \times n_{\mathrm{u}}} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ \boldsymbol{I}_{n_{\mathrm{u}} \times n_{\mathrm{u}}} & \boldsymbol{I}_{n_{\mathrm{u}} \times n_{\mathrm{u}}} & \dots & \boldsymbol{I}_{n_{\mathrm{u}} \times n_{\mathrm{u}}} \end{bmatrix} \tag{2.18}$$

gdzie $\boldsymbol{I}_{n_{\mathrm{u}} \times n_{\mathrm{u}}}$ jest macierzą jednostkową o wymiarach $n_{\mathrm{u}} \times n_{\mathrm{u}}$, natomiast wektor sygnałów sterujących wszystkich wyjść na horyzoncie sterowania, o długość $n_{\mathrm{u}} N_{\mathrm{u}}$, ma strukturę odpowiadającą wektorowi zmiennych decyzyjnych (2.2)

$$\boldsymbol{U}(k-1) = \begin{bmatrix} u(k-1) \\ \vdots \\ u(k-1) \end{bmatrix} \tag{2.19}$$

gdzie

$$u(k-1) = [u_1(k-1) \dots u_{n_{\mathrm{u}}}(k-1)]^{\mathrm{T}} \tag{2.20}$$

jest wektorem o długości n_{u}. Wektory definiujące ograniczenia sygnałów sterujących mają długość $n_{\mathrm{u}} N_{\mathrm{u}}$

$$\boldsymbol{U}^{\min} = \begin{bmatrix} u^{\min} \\ \vdots \\ u^{\min} \end{bmatrix}, \ \boldsymbol{U}^{\max} = \begin{bmatrix} u^{\max} \\ \vdots \\ u^{\max} \end{bmatrix} \tag{2.21}$$

$$\triangle \boldsymbol{U}^{\min} = \begin{bmatrix} \triangle u^{\min} \\ \vdots \\ \triangle u^{\min} \end{bmatrix}, \ \triangle \boldsymbol{U}^{\max} = \begin{bmatrix} \triangle u^{\max} \\ \dots \\ \triangle u^{\max} \end{bmatrix} \tag{2.22}$$

gdzie

$$\begin{array}{rlrl} u^{\min} & = \left[u_1^{\min} \dots u_{n_{\mathrm{u}}}^{\min}\right]^{\mathrm{T}}, & u^{\max} & = \left[u_1^{\max} \dots u_{n_{\mathrm{u}}}^{\max}\right]^{\mathrm{T}} \\ \triangle u^{\min} & = \left[\triangle u_1^{\min} \dots \triangle u_{n_{\mathrm{u}}}^{\min}\right]^{\mathrm{T}}, & \triangle u^{\max} & = \left[\triangle u_1^{\max} \dots \triangle u_{n_{\mathrm{u}}}^{\max}\right]^{\mathrm{T}} \end{array} \tag{2.23}$$

są wektorami o długości n_{u}. Wektory definiujące ograniczenia prognozowanych sygnałów wyjściowych mają długość $n_{\mathrm{y}}N$

$$\boldsymbol{Y}^{\min} = \begin{bmatrix} y^{\min} \\ \vdots \\ y^{\min} \end{bmatrix}, \ \boldsymbol{Y}^{\max} = \begin{bmatrix} y^{\max} \\ \vdots \\ y^{\max} \end{bmatrix} \tag{2.24}$$

gdzie

$$y^{\min} = \left[y_1^{\min} \dots y_{n_{\mathrm{y}}}^{\min}\right]^{\mathrm{T}}, \qquad y^{\max} = \left[y_1^{\max} \dots y_{n_{\mathrm{y}}}^{\max}\right]^{\mathrm{T}} \tag{2.25}$$

są wektorami o długości n_{y}.

Jeżeli do predykcji zostanie zastosowany model liniowy [108], to całą trajektorię prognozowaną zmiennych wyjściowych procesu można zapisać jako liniową funkcję obliczanych zmiennych decyzyjnych algorytmu regulacji predykcyjnej, czyli przyrostów sygnałów sterujących (2.2)

$$\hat{\boldsymbol{Y}}(k) = \boldsymbol{Y}^0(k) + \boldsymbol{M}\triangle\boldsymbol{U}(k) \tag{2.26}$$

gdzie macierz $\boldsymbol{M}$, o wymiarowości $n_{\mathrm{y}}N \times n_{\mathrm{u}}N_{\mathrm{u}}$, jest zwana macierzą dynamiczną i zawiera współczynniki odpowiedzi skokowej modelu procesu

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{S}_1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{S}_N & \boldsymbol{S}_{N-1} & \dots & \boldsymbol{S}_{N-N_{\mathrm{u}}+1} \end{bmatrix} \tag{2.27}$$

gdzie macierze współczynników odpowiedzi skokowej procesu wielowymiarowego dla dyskretnej chwili p mają postać

$$\boldsymbol{S}_p = \begin{bmatrix} s^{1,1} & \dots & s^{1,n_{\mathrm{u}}} \\ \vdots & \ddots & \vdots \\ s^{n_{\mathrm{y}},1} & \dots & s^{n_{\mathrm{y}},n_{\mathrm{u}}} \end{bmatrix} \tag{2.28}$$

Trajektoria prognozowana wyjść (2.26) zależy również od przeszłości (czyli wpływu sygnałów sterujących do chwili $k-1$ włącznie), którą opisuje trajektoria swobodna

o długości $n_{\mathrm{y}}N$

$$\boldsymbol{Y}^0(k) = \begin{bmatrix} y^0(k+1|k) \\ \vdots \\ y^0(k+N|k) \end{bmatrix} \tag{2.29}$$

gdzie

$$y^0(k+p|k) = \left[y_1^0(k+p|k) \dots y_{n_{\mathrm{y}}}^0(k+p|k)\right]^{\mathrm{T}} \tag{2.30}$$

jest wektorem o długości n_{y}. Stałą macierz współczynników odpowiedzi skokowych oraz wektor trajektorii swobodnej (zmienny w każdej chwili dyskretnej k) wyznacza się na podstawie modelu matematycznego regulowanego procesu. Szczegóły zależą od zastosowanego modelu i zostaną omówione w dalszej części rozdziału. Warto podkreślić, że dzięki zastosowaniu liniowego modelu, otrzymuje się liniowe równanie predykcji (2.26), co umożliwia sprowadzenie ogólnego zadania optymalizacji algorytmu regulacji predykcyjnej (2.5) z ograniczeniami (2.6) do łatwego do rozwiązania zadania optymalizacji kwadratowej (kwadratowa funkcja celu oraz liniowe ograniczenia). Zadanie takie ma tylko jedno rozwiązanie, które jest minimum globalnym.

2.3. Algorytm DMC

Algorytm Dynamic Matrix Control (DMC) [31] powstał w latach siedemdziesiątych XX w. Jego cechą szczególną jest wykorzystanie jako modelu zestawu odpowiedzi skokowych. Oznacza to, że należy jedynie zarejestrować odpowiedzi skokowe procesu, nie są potrzebne żadne wyszukane procedury identyfikacji modeli. Zaletą jest również łatwy sposób implementacji, znacznie łatwiejszy niż innych algorytmów tej klasy. Dlatego algorytm DMC , jest obecnie najczęściej spotykaną w przemyśle wersją algorytmu regulacji predykcyjnej. Jedynym ograniczeniem jest jednak wymaganie aby proces był w pętli otwartej (bez regulatora) stabilny i nie posiadał całkowania (drugie z wymienionych ograniczeń może nie być istotne po zastosowaniu rozszerzenia algorytmu DMC [108]).

2.3.1. Predykcja w algorytmie DMC

Model odpowiedzi skokowych stosowany w algorytmie DMC zdefiniowany jest zestawem macierzy (2.28) wielowymiarowej odpowiedzi skokowej. Aby pozyskać model obiektu należy wymusić skok sygnału pierwszego wejścia i rejestrować zmia-

ny wszystkich n_{y} wyjść obiektu aż do momentu ich stabilizacji, pozostałe wejścia procesu powinny być stałe (w punkcie równowago). Następnie należy podać skok sygnału drugiego, trzeciego itd. wejścia. Procedurę taką powtarza się dla wszystkich n_{u} sygnałów wejściowych. Liczbę współczynników odpowiedzi skokowej, która jest konieczna do opisania procesu, nazywa się horyzontem dynamiki D (zakłada się, że dla chwili $k > D$ odpowiedzi skokowe wszystkich torów wejście-wyjście stabilizują się, tzn. $s_k^{m,n} = s_D^{m,n}$ dla wszystkich $m = 1, \ldots, n_{\mathrm{y}}$, $n = 1, \ldots, n_{\mathrm{u}}$). Model odpowiedzi skokowej może posłużyć do wyznaczenia reakcji wyjść procesu w zależności od przyrostów sygnałów sterujących

$$y(k+p|k) = y(0) + \sum_{j=1}^{k+p} \boldsymbol{S}_j \triangle u_n(k+p-j) \tag{2.31}$$

Stosując model (2.31) można wyprowadzić ogólne równanie predykcji (2.26), co pokazano w pracy [108], gdzie macierz dynamiczna ma postać (2.27), natomiast wektor trajektorii swobodnej jest obliczony ze wzoru

$$\boldsymbol{Y}^0(k) = \boldsymbol{Y}(k) + \boldsymbol{M}^{\mathrm{P}} \triangle \boldsymbol{U}^{\mathrm{P}}(k) \tag{2.32}$$

Wektor pomiarów sygnałów wyjściowych procesu, o długości $n_{\mathrm{y}}N$, ma postać

$$\boldsymbol{Y}(k) = \begin{bmatrix} y(k) \\ \vdots \\ y(k) \end{bmatrix} \tag{2.33}$$

gdzie

$$y(k) = \begin{bmatrix} y_1(k) \ldots y_{n_{\mathrm{y}}}(k) \end{bmatrix}^{\mathrm{T}} \tag{2.34}$$

jest wektorem o długości n_{y}. Dodatkowa macierz współczynników odpowiedzi skokowej, o wymiarach $n_{\mathrm{y}}N \times n_{\mathrm{u}}(D-1)$, ma postać

$$\boldsymbol{M}^{\mathrm{P}} = \begin{bmatrix} \boldsymbol{S}_2 - \boldsymbol{S}_1 & \boldsymbol{S}_3 - \boldsymbol{S}_2 & \ldots & \boldsymbol{S}_D - \boldsymbol{S}_{D-1} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{S}_{N+1} - \boldsymbol{S}_1 & \boldsymbol{S}_{N+2} - \boldsymbol{S}_2 & \ldots & \boldsymbol{S}_{N+D-1} - \boldsymbol{S}_{D-1} \end{bmatrix} \tag{2.35}$$

Wektor przyrostów sygnałów sterujących z przeszłości, o długości $n_{\mathrm{u}}(D-1)$, ma postać

$$\triangle \boldsymbol{U}^{\mathrm{P}}(k) = \begin{bmatrix} \triangle u(k-1) \\ \vdots \\ \triangle u(k-(D-1)) \end{bmatrix} \tag{2.36}$$

2.3.2. Algorytm DMC w wersji numerycznej

Uwzględniając, że w algorytmach wykorzystujących do predykcji modele liniowe wektor prognozowany sygnałów wyjściowych jest liniową funkcją zmiennych decyzyjnych (równanie (2.26)), trajektorie swobodną wyznaczoną dla modelu odpowiedzi skokowej (równanie (2.32)), ogólny wektorowo-macierzowy problem optymalizacji algorytmu regulacji predykcyjnej (2.16) przyjmuje postać

$$\min_{\triangle \boldsymbol{U}(k)} \left\{ J(k) = \left\| \boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}(k) - \boldsymbol{M}^{\mathrm{P}} \triangle \boldsymbol{U}^{\mathrm{P}}(k) - \boldsymbol{M} \triangle \boldsymbol{U}(k) \right\|_{\boldsymbol{\Psi}}^{2} + \left\| \triangle \boldsymbol{U}(k) \right\|_{\boldsymbol{\Lambda}}^{2} \right\} \tag{2.37}$$

natomiast ogólne ograniczenia (2.17) sprowadza się do postaci

$$\begin{array}{rcccl} \triangle \boldsymbol{U}^{\min} & \leqslant & \triangle \boldsymbol{U}(k) & \leqslant & \triangle \boldsymbol{U}^{\max} \\ \boldsymbol{U}^{\min} & \leqslant & \boldsymbol{U}(k-1) + \boldsymbol{J} \triangle \boldsymbol{U}(k) & \leqslant & \boldsymbol{U}^{\max} \\ \boldsymbol{Y}^{\min} & \leqslant & \boldsymbol{Y}(k) + \boldsymbol{M}^{\mathrm{P}} \triangle \boldsymbol{U}^{\mathrm{P}}(k) + \boldsymbol{M} \triangle \boldsymbol{U}(k) & \leqslant & \boldsymbol{Y}^{\max} \end{array} \tag{2.38}$$

Ogólne zadanie optymalizacji algorytmu DMC, zdefiniowane równaniem (2.37), z ograniczeniami (2.38) można łatwo przekształcić do postaci standardowej, typowej dla zadań optymalizacji kwadratowej [108]

$$\begin{gathered} \min_{x} \left\{ J(k) = \tfrac{1}{2} x^{\mathrm{T}} \boldsymbol{H} x + f^{\mathrm{T}} x \right\} \\ x^{\min} \leqslant x \leqslant x^{\max} \\ \overline{\boldsymbol{A}} x \leqslant \overline{b} \end{gathered} \tag{2.39}$$

gdzie

$$x = \triangle \boldsymbol{U}(k),\ x^{\min} = \triangle \boldsymbol{U}^{\min},\ x^{\max} = \triangle \boldsymbol{U}^{\max} \tag{2.40}$$

macierz o wymiarach $N_{\mathrm{u}}n_{\mathrm{u}} \times N_{\mathrm{u}}n_{\mathrm{u}}$ ma postać

$$\boldsymbol{H} = 2\left(\boldsymbol{M}^{\mathrm{T}}\boldsymbol{\Psi}\boldsymbol{M} + \boldsymbol{\Lambda}\right) \tag{2.41}$$

natomiast wektor o długości $N_{\mathrm{u}}n_{\mathrm{u}}$ ma postać

$$f = -2\boldsymbol{M}^{\mathrm{T}}\boldsymbol{\Psi}(\boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}(k) - \boldsymbol{M}^{\mathrm{P}}\triangle\boldsymbol{U}^{\mathrm{P}}(k)) \tag{2.42}$$

Ograniczenia zdefiniowane są przez macierz $\overline{\boldsymbol{A}}$ o wymiarach $2(N_{\mathrm{u}}n_{\mathrm{u}}+Nn_{\mathrm{y}})\times N_{\mathrm{u}}n_{\mathrm{u}}$ oraz wektor $\overline{b}$ o długości $2(N_{\mathrm{u}}n_{\mathrm{u}} + Nn_{\mathrm{y}})$, które mają strukturę

$$\overline{\boldsymbol{A}} = \begin{bmatrix} -\boldsymbol{J} \\ \boldsymbol{J} \\ -\boldsymbol{M} \\ \boldsymbol{M} \end{bmatrix}, \quad \overline{b} = \begin{bmatrix} -\boldsymbol{U}^{\min} + \boldsymbol{U}(k-1) \\ \boldsymbol{U}^{\max} - \boldsymbol{U}(k-1) \\ -\boldsymbol{Y}^{\min} + \boldsymbol{Y}(k) + \boldsymbol{M}^{\mathrm{P}}\triangle\boldsymbol{U}^{\mathrm{P}}(k) \\ \boldsymbol{Y}^{\max} - \boldsymbol{Y}(k) - \boldsymbol{M}^{\mathrm{P}}\triangle\boldsymbol{U}^{\mathrm{P}}(k) \end{bmatrix} \tag{2.43}$$

Kwadratowe zadanie optymalizacji algorytmu DMC (2.39) można rozwiązać algorytmem zbioru ograniczeń aktywnych lub punktu wewnętrznego. Jak już wspomniano, otrzymane rozwiązanie jest zawsze minimum globalnym. Warto podkreślić, że w odróżnieniu od nieliniowych zadań optymalizacji, w zadaniach optymalizacji kwadratowej można określić przewidywany czas obliczeń. Ma to kluczowe znaczenie przy implementacji algorytmu dla szybkich procesów dynamicznych, przy wykorzystaniu jednostek obliczeniowych o ograniczonej szybkości (systemy wbudowane).

2.3.3. Algorytm DMC w wersji analitycznej

W każdej dyskretnej chwili próbkowania wersji numerycznej algorytmu DMC należy rozwiązać kwadratowe zadanie optymalizacji (2.37) z ograniczeniami (2.38) w celu wyznaczenia wektora zmiennych decyzyjnych (2.2). Gdy ograniczenia nie istnieją, lub są celowo pominięte, istnieje możliwość bardzo efektywnego wyznaczenia wektora zmiennych decyzyjnych, w sposób analityczny, bez potrzeby optymalizacji. Ponieważ minimalizowana funkcja celu algorytmu DMC jest kwadratowa względem wektora zmiennych decyzyjnych, z warunku zerowania się jej gradientu (wektor o długości $n_{\mathrm{u}}N_{\mathrm{u}}$), otrzymuje się rozwiązanie optymalne

$$\triangle\boldsymbol{U}(k) = \boldsymbol{K}(\boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}(k) - \boldsymbol{M}^{\mathrm{P}}\triangle\boldsymbol{U}^{\mathrm{P}}(k)) \tag{2.44}$$

gdzie

$$\boldsymbol{K} = (\boldsymbol{M}^{\mathrm{T}}\boldsymbol{\Psi}\boldsymbol{M} + \boldsymbol{\Lambda})^{-1}\boldsymbol{M}^{\mathrm{T}}\boldsymbol{\Psi} \tag{2.45}$$

jest macierzą o wymiarach $N_{\mathrm{u}}n_{\mathrm{u}} \times Nn_{\mathrm{y}}$.

Warto zauważyć, że wzór (2.44) wyznacza wektor przyrostów sygnałów sterowania określony na horyzoncie sterowania, choć w aktualnej chwili k do obiektu aplikowanych jest wyłącznie pierwszych n_{u} jego elementów (wzór (2.4)). Wystarczy więc wykonać tylko część obliczeń dla uzyskania wektora sterowania dla danej chwili k. W dalszej części pracy taki schemat obliczeniowy jest nazywany oszczędnym algorytmem analitycznym DMC. Macierz (2.45) można zapisać w postaci

$$\boldsymbol{K} = \begin{bmatrix} \overline{\boldsymbol{K}}_1 \\ \vdots \\ \overline{\boldsymbol{K}}_{N_{\mathrm{u}}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}_{1,1} & \dots & \boldsymbol{K}_{1,N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{K}_{N_{\mathrm{u}},1} & \dots & \boldsymbol{K}_{N_{\mathrm{u}},N} \end{bmatrix} \tag{2.46}$$

gdzie każda z podmacierzy $\overline{\boldsymbol{K}}_i$ ma wymiar $n_{\mathrm{u}} \times n_{\mathrm{y}}N$ $(i = 1, \dots, N_{\mathrm{u}})$, natomiast każda z podmacierzy $\boldsymbol{K}_{i,j}$ ma wymiar $n_{\mathrm{u}} \times n_{\mathrm{y}}$ i strukturę

$$\boldsymbol{K}_{i,j} = \begin{bmatrix} k_{i,j}^{1,1} & \dots & k_{i,j}^{1,n_{\mathrm{y}}} \\ \vdots & \ddots & \vdots \\ k_{i,j}^{n_{\mathrm{u}},1} & \dots & k_{i,j}^{n_{\mathrm{u}},n_{\mathrm{y}}} \end{bmatrix} \tag{2.47}$$

gdzie $i = 1, \dots, N_{\mathrm{u}}$, $j = 1, \dots, N$. Macierz $\boldsymbol{M}^{\mathrm{P}}$ określoną wzorem (2.35) przedstawia się jako $\boldsymbol{M}^{\mathrm{P}} = \left[\boldsymbol{M}^{\mathrm{P}} \dots \boldsymbol{M}^{\mathrm{P}}_{D-1}\right]$, gdzie każda z podmacierzy $\boldsymbol{M}^{\mathrm{P}}_i$ ma wymiar $n_{\mathrm{y}}N \times n_{\mathrm{u}}$. Szukany wektor przyrostów sygnałów sterujących dla aktualnej chwili k wyznacza się na podstawie wzoru

$$\triangle u(k|k) = \sum_{p=1}^{N} \boldsymbol{K}_{1,p}(y^{\mathrm{zad}}(k) - y(k)) - \sum_{i=1}^{D-1} \boldsymbol{K}_i^{\mathrm{u}} \triangle u(k-i) \tag{2.48}$$

gdzie

$$\boldsymbol{K}^{\mathrm{e}} = \sum_{p=1}^{N} \boldsymbol{K}_{1,p} \tag{2.49}$$

$$\boldsymbol{K}_i^{\mathrm{u}} = \overline{\boldsymbol{K}}_1 \boldsymbol{M}_i^{\mathrm{p}},\ i = 1, \dots, D-1 \tag{2.50}$$

są odpowiednio macierzami o wymiarach $n_{\mathrm{u}} \times n_{\mathrm{y}}$ i $n_{\mathrm{u}} \times (D-1)$. Zakładając, że wartości zadane sygnałów regulowanych są stałe na horyzoncie predykcji, czyli

$$y^{\mathrm{zad}}(k) = y^{\mathrm{zad}}(k+1|k) = \ldots = y^{\mathrm{zad}}(k+N|k) \tag{2.51}$$

szukany wektor przyrostów wyznacza się ze wzoru

$$\triangle u(k|k) = \boldsymbol{K}^{\mathrm{e}}(y^{\mathrm{zad}}(k) - y(k)) - \sum_{i=1}^{D-1} \boldsymbol{K}_i^{\mathrm{u}} \triangle u(k-i) \tag{2.52}$$

Oczywiście, zgodnie ze wzorem (2.4), aby wyznaczyć wartość sygnałów sterujących w aktualnej chwili próbkowania, należy uwzględnić wyznaczone przyrosty oraz ich wartość w poprzedniej chwili

$$u_n(k) = u_n(k-1) + \triangle u_n(k|k) \tag{2.53}$$

dla $n = 1, \ldots, n_{\mathrm{u}}$.

Pominięcie ograniczeń umożliwiło bardzo proste, analityczne wyznaczenie przyrostów sygnałów sterujących (wzory (2.44), (2.48) lub (2.52)). Ponieważ zazwyczaj nie można jednak zignorować ograniczeń, głównie ze względu na ograniczone możliwości elementów wykonawczych, w algorytmie analitycznym stosuje się rzutowanie wartości sterowania na zbiór rozwiązań dopuszczalnych stosując dla $n = 1, \ldots, n_{\mathrm{u}}$ następującą procedurę

$$\begin{aligned}
&\text{jeżeli } \triangle u_n(k|k) < \triangle u_n^{\min} \text{ to } \triangle u_n(k|k) = \triangle u_n^{\min},\\
&\text{jeżeli } \triangle u_n(k|k) > \triangle u_n^{\max} \text{ to } \triangle u_n(k|k) = \triangle u_n^{\max},\\
&u_n(k|k) = u_n(k-1) + \triangle u_n(k|k),\\
&\text{jeżeli } u_n(k|k) < u_n^{\min} \text{ to } u_n(k|k) = u_n^{\min},\\
&\text{jeżeli } u_n(k|k) > u_n^{\max} \text{ to } u_n(k|k) = u_n^{\max},\\
&u_n(k) = u_n(k|k)
\end{aligned} \tag{2.54}$$

Podsumowując wersję oszczędną algorytmu DMC, muszą być zrealizowane następujące obliczenia (off-line):

1. Dla danego modelu w postaci odpowiedzi skokowej określonego jako D podmacierzy odpowiedzi skokowych $\boldsymbol{S}_p$ i macierzy współczynników wag $\boldsymbol{M}$ i $\boldsymbol{\Lambda}$, należy obliczyć macierze $\boldsymbol{M}$ (równanie (2.27)) i $\boldsymbol{M}^{\mathrm{p}}$ (równanie (2.35)).

2. Z równania (2.46) oblicza się tylko pierwszy wiersz macierzy $\boldsymbol{K}$, czyli podmacierz $\overline{\boldsymbol{K}}_{1,1}$.
3. Macierze $\boldsymbol{K}^{\mathrm{e}}$ i $\boldsymbol{K}_i^{\mathrm{u}}$ są obliczane z równań (2.49) i (2.50).

W każdej chwili próbkowania obliczenia (on-line) optymalne wartości przyrostów sygnałów sterujących $\triangle u(k|k)$ wyznacza się ze wzoru (2.48) lub (2.52). Należy podkreślić, że w tym podejściu horyzonty predykcji i sterowania, N and N_{u}, nie wpływają na złożoność obliczeniową algorytmu. Złożoność obliczeniowa analitycznego algorytmu DMC zależy tylko od długości horyzontu dynamiki D i wymiarowości procesu (liczy wejść i wyjść). Mimo, że zakładanie małych wartości horyzontu dynamiki zmniejsza złożoność algorytmu należy pamiętać, że może to mieć negatywny wpływ na jakość regulacji z uwagi na niedostatecznie dokładny model regulowanego procesu. Uwzględnienie ograniczeń sygnałów sterujących polega na wykonaniu rzutowania wyznaczonego sterowania na zbiór dopuszczalnych rozwiązań (wzory (2.54)). Wykonywane jest to tuż przed wysłaniem wartości sterowań do regulowanego procesu.

2.4. Algorytm GPC

Algorytm Generalized Predictive Control (GPC) [27] powstał w latach osiemdziesiątych XX w. Eliminuje on zasadnicze ograniczenie algorytmu DMC, a mianowicie możliwość zastosowania wyłącznie do procesów stabilnych bez całkowania. W algorytmie GPC stosuje się model procesu w postaci równania różnicowego (transmitancji dyskretnej). Drugą, bardzo ważna zaletą algorytmu GPC w porównaniu z algorytmem DMC, jest mała zajętość pamięci. Modele stosowane w algorytmie GPC mają kilka, najwyżej kilkanaście parametrów (dla jednego toru wejście-wyjście), podczas gdy typowe modele odpowiedzi skokowej, oczywiście w zależności od okresu próbkowania, mają typowo kilkadziesiąt lub więcej współczynników, przy czym należy uwzględnić współczynniki dla wszystkich torów wejście-wyjście. Oczywiście, problem braku pamięci nie istnieje w klasycznych zastosowaniach przemysłowych, gdzie algorytm DMC zaimplementowany jest przy wykorzystaniu dedykowanego programowalnego sterownika logicznego lub sterownika przemysłowego o dużych zasobach. Problem braku pamięci może być natomiast istotny w systemach wbudowanych, o ograniczonych zasobach.

2.4.1. Predykcja w algorytmie GPC

Model ARX (AutoRegressive with eXogenous input) ma postać

$$\boldsymbol{A}(z^{-1})y(k) = \boldsymbol{B}(z^{-1})u(k) \tag{2.55}$$

gdzie elementy macierzy

$$\boldsymbol{A}(z^{-1}) = \begin{bmatrix} A_{1,1}(z^{-1}) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & A_{n_\mathrm{y},n_\mathrm{y}}(z^{-1}) \end{bmatrix} \tag{2.56}$$

oraz

$$\boldsymbol{B}(z^{-1}) = \begin{bmatrix} B_{1,1}(z^{-1}) & \dots & B_{1,n_\mathrm{u}}(z^{-1}) \\ \vdots & \ddots & \vdots \\ B_{n_\mathrm{y},1}(z^{-1}) & \dots & B_{n_\mathrm{y},n_\mathrm{u}}(z^{-1}) \end{bmatrix} \tag{2.57}$$

są wielomianami zmiennej z^{-1} (opóźnienia jednostkowego o okres próbkowania)

$$A_{m,m}(z^{-1}) = 1 + a_1^m z^{-1} + \ldots + a_{n_\mathrm{A}}^m z^{-n_\mathrm{A}} \tag{2.58}$$

dla $m = 1, \ldots, n_\mathrm{y}$ oraz

$$B_{m,n}(z^{-1}) = b_1^{m,n} z^{-1} + \ldots + b_{n_\mathrm{B}}^{m,n} z^{-n_\mathrm{B}} \tag{2.59}$$

dla $m = 1, \ldots, n_\mathrm{y}$, $n = 1, \ldots, n_\mathrm{u}$.

Jak już wspomniano, można wykazać, że jeżeli do predykcji zostanie zastosowany model liniowy, to wektor prognozowany sygnałów wyjściowych (na horyzoncie predykcji) będzie równy trajektorii swobodnej $\boldsymbol{Y}^0(k)$ oraz iloczynowi macierzy dynamicznej $\boldsymbol{M}$ oraz obliczanego wektora przyrostów sygnałów sterujących $\triangle\boldsymbol{U}(k)$ (wzór (2.26)) [108]. Warto podkreślić, że we wszystkich algorytmach wpływ przyszłości (drugi człon wzoru (2.26)) jest zawsze taki sam, inaczej jest natomiast modelowany wpływ przeszłości, zdefiniowany trajektorią swobodną. Algorytmy DMC i GPC różnią się rodzajem zastosowanego modelu. W algorytmie DMC, wykorzystującym model odpowiedzi skokowej, trajektoria swobodna obliczana jest na podstawie aktualnych pomiarów wyjść procesu (wektor $(\boldsymbol{Y}(k))$) oraz iloczynu macierzy współczynników odpowiedzi skokowej $\boldsymbol{M}^\mathrm{P}$ oraz wektora przyrostów sygnałów sterujących do chwili $k-1$ włącznie $\triangle\boldsymbol{U}^\mathrm{P}(k)$ (wzór (2.32)). W oryginalnej wersji algo-

rytmu GPC, wykorzystującym model ARX postaci (2.55), do obliczenia trajektorii swobodnej stosuje się dość złożone równania diofantyczne [27, 108]. W niniejszej pracy wykorzystuje się znacznie prostszą wersję algorytmu GPC, zwaną algorytmem GPC z modelem zakłóceń typu DMC [71, 108]. W tym podejściu stosując rekurencyjnie model ARX i uwzględniając jedynie wpływ przeszłości, oblicza się rekurencyjnie poszczególne składowe wektora trajektorii swobodnej

$$\begin{aligned} y_m^0(k+p|k) = &\sum_{n=1}^{n_\mathrm{u}} \sum_{l=1}^{I_\mathrm{uf}(p)} b_l^{m,n} u_n(k-1) + \sum_{n=1}^{n_\mathrm{u}} \sum_{l=I_\mathrm{uf}(p)+1}^{n_\mathrm{B}} b_l^{m,n} u_n(k-l+p) \\ &- \sum_{l=1}^{I_\mathrm{yf}(p)} a_l^m y_m^0(k-l+p|k) - \sum_{l=I_\mathrm{yf}(p)+1}^{n_\mathrm{A}} a_l^m y_m(k-l+p) \\ &+ d_m(k) \end{aligned} \tag{2.60}$$

gdzie

$$\begin{aligned} I_\mathrm{uf}(p) &= \max(\min(p, n_\mathrm{B}), 0) \\ I_\mathrm{yf}(p) &= \min(p-1, n_\mathrm{A}) \end{aligned} \tag{2.61}$$

dla $p = 1, \ldots, N$, natomiast $d_m(k)$ określa zakłócenie niemierzalne w chwili k, które można wyznaczyć jako różnicę między wartością zmierzoną m-tego sygnału wyjściowego a jego predykcją w chwili $k-1$ na chwilę k

$$d_m(k) = y_m(k) - \left(\sum_{n=1}^{n_\mathrm{u}} \sum_{i=1}^{n_\mathrm{B}} b_i^{m,n} u_n(k-i) - \sum_{i=1}^{n_\mathrm{A}} a_i^m y_m(k-i) \right) \tag{2.62}$$

Można wykazać [71], że trajektoria swobodna w algorytmie GPC może być zapisana w postaci

$$y_m^0(k+p|k) = \sum_{n=1}^{n_\mathrm{u}} \sum_{i=1}^{n_\mathrm{B}} e_i^{m,n}(p) u_n(k-i) + \sum_{i=0}^{n_\mathrm{A}} f_i^m(p) y_m(k-i) \tag{2.63}$$

gdzie

$$e_i^{m,n}(p) = \begin{cases} 0, & i = 1 \wedge n_\mathrm{B} = 1 \\ g_i^{m,n}(p), & i = 1 \wedge n_\mathrm{B} > 1 \\ g_i^{m,n}(p) - g_{i-1}^{m,n}(p), & i = 2, \ldots, n_\mathrm{B} - 1 \wedge i < n_\mathrm{B} \wedge n_\mathrm{B} > 1 \\ -g_{i-1}^{m,n}(p), & i = n_\mathrm{B} \wedge n_\mathrm{B} > 1 \end{cases} \tag{2.64}$$

oraz

$$g_i^{m,n}(p) = \begin{cases} g_{i+1}^{m,n}(p-1), & i = 1-p, \ldots, -1 \\ f_0^m(p-1) g_i^{m,n}(1) + g_{i+1}^{m,n}(p-1), & i = 0, \ldots, n_\mathrm{B} - 2 \\ f_0^m(p-1) g_i^{m,n}(1), & i = n_\mathrm{B} - 1 \end{cases} \tag{2.65}$$

dla $p = 2, \ldots, N$, natomiast dla $p = 1$, $g_i^{m,n}(1) = b_{i+1}^{m,n}$ dla każdego $i = 0, \ldots, n_\mathrm{B}-1$. Dla $p = 1$,

$$f_i^m(1) = \begin{cases} 1 - a_1^m, & i = 0 \\ a_i^m - a_{i+1}^m, & i = 1, \ldots, n_\mathrm{A} - 1 \\ a_i^m, & i = n_\mathrm{A} \end{cases} \tag{2.66}$$

natomiast dla $p = 2, \ldots, N$

$$f_i^m(p) = \begin{cases} f_0^m(p-1) f_i^m(1) + f_{i+1}^m(p-1), & i = 0, \ldots, n_\mathrm{A} - 1 \\ f_0^m(p-1) f_i^m(1), & i = n_\mathrm{A} \end{cases} \tag{2.67}$$

2.4.2. Algorytm GPC w wersji numerycznej

Uwzględniając, że w algorytmie GPC prognozowany sygnałów wyjściowych jest liniową funkcją zmiennych decyzyjnych (równanie (2.26)), ogólny wektorowo-macierzowy problem optymalizacji algorytmu regulacji predykcyjnej (2.16) przyjmuje postać

$$\min_{\triangle \boldsymbol{U}(k)} \left\{ J(k) = \left\| \boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}^0(k) - \boldsymbol{M} \triangle \boldsymbol{U}(k) \right\|_{\boldsymbol{\Psi}}^2 + \left\| \triangle \boldsymbol{U}(k) \right\|_{\boldsymbol{\Lambda}}^2 \right\} \tag{2.68}$$

natomiast ogólne ograniczenia (2.17) sprowadza się do postaci

$$\begin{array}{rcccl} \triangle \boldsymbol{U}^{\min} & \leqslant & \triangle \boldsymbol{U}(k) & \leqslant & \triangle \boldsymbol{U}^{\max} \\ \boldsymbol{U}^{\min} & \leqslant & \boldsymbol{U}(k-1) + \boldsymbol{J} \triangle \boldsymbol{U}(k) & \leqslant & \boldsymbol{U}^{\max} \\ \boldsymbol{Y}^{\min} & \leqslant & \boldsymbol{Y}^0(k) + \boldsymbol{M} \triangle \boldsymbol{U}(k) & \leqslant & \boldsymbol{Y}^{\max} \end{array} \tag{2.69}$$

Ogólne zadanie optymalizacji algorytmu DMC, zdefiniowane równaniem (2.68), z ograniczeniami (2.69) można łatwo przekształcić do postaci standardowej, typowej dla zadań optymalizacji kwadratowej określonej wzorem (2.39), gdzie macierz $\boldsymbol{H}$, analogicznie jak w algorytmie DMC, zdefiniowana jest wzorem (2.41), natomiast

wektor o długości $N_{\mathrm{u}} n_{\mathrm{u}}$ ma postać

$$f = -2\boldsymbol{M}^{\mathrm{T}}\boldsymbol{\Psi}(\boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}^{0}(k)) \tag{2.70}$$

Ograniczenia zdefiniowane są przez

$$\overline{\boldsymbol{A}} = \begin{bmatrix} -\boldsymbol{J} \\ \boldsymbol{J} \\ -\boldsymbol{M} \\ \boldsymbol{M} \end{bmatrix}, \quad \overline{b} = \begin{bmatrix} -\boldsymbol{U}^{\min} + \boldsymbol{U}(k-1) \\ \boldsymbol{U}^{\max} - \boldsymbol{U}(k-1) \\ -\boldsymbol{Y}^{\min} + \boldsymbol{Y}^{0}(k) \\ \boldsymbol{Y}^{\max} - \boldsymbol{Y}^{0}(k) \end{bmatrix} \tag{2.71}$$

Analogicznie jak w przypadku algorytmu DMC, wyznaczone rozwiązanie jest zawsze minimum globalnym.

2.4.3. Algorytm GPC w wersji analitycznej

Gdy ograniczenia nie istnieją, lub są celowo pominięte, również w algorytmie GPC istnieje możliwość bardzo efektywnego wyznaczenia wektora zmiennych decyzyjnych, w sposób analityczny, bez potrzeby optymalizacji. Analogicznie jak w algorytmie DMC minimalizowana funkcja celu algorytmu DMC jest kwadratowa względem wektora zmiennych decyzyjnych, z warunku zerowania się jej gradientu (wektor o długości $n_{\mathrm{u}} N_{\mathrm{u}}$), otrzymuje się rozwiązanie optymalne

$$\triangle\boldsymbol{U}(k) = \boldsymbol{K}(\boldsymbol{Y}^{\mathrm{zad}}(k) - \boldsymbol{Y}^{0}(k)) \tag{2.72}$$

Uzyskany wzór jest bardzo podobny do wzoru (2.44), który otrzymano dla algorytmu DMC, jedyną różnicą jest inny sposób wyznaczania trajektorii swobodnej. Uwzględniając, że jest ona obliczana ze wzoru (2.63) oraz zakładając, że wartości zadane sygnałów regulowanych są stałe na horyzoncie predykcji (wzór (2.51)), można wykazać, że analityczne prawo regulacji algorytmu GPC ma postać [25]

$$\begin{aligned} \triangle u_r(k|k) = &\sum_{m=1}^{n_{\mathrm{y}}} K_{r,m}^{\mathrm{y}^{\mathrm{zad}}}\, y_m^{\mathrm{zad}}(k) \\ &+ \sum_{n=1}^{n_{\mathrm{u}}} \sum_{j=1}^{n_{\mathrm{B}}} K_{r,n}^{\mathrm{u}}(j) u_n(k-j) \\ &+ \sum_{m=1}^{n_{\mathrm{y}}} \sum_{j=0}^{n_{\mathrm{A}}} K_{r,m}^{\mathrm{y}}(j) y_m(k-j) \end{aligned} \tag{2.73}$$

dla $r = 1, \ldots, n_{\mathrm{u}}$. W dalszej części pracy taki schemat obliczeniowy jest nazywany oszczędnym algorytmem analitycznym GPC. Poszczególne współczynniki mają następujące definicje

$$\begin{aligned} K_{r,m}^{\mathrm{y^{zad}}} &= \sum_{p=1}^{N} \boldsymbol{K}_{r,(p-1)n_{\mathrm{y}}+m} \quad \text{dla } r = 1, \ldots, n_{\mathrm{u}}; m = 1, \ldots, n_{\mathrm{y}} \\ K_{r,n}^{\mathrm{u}}(j) &= -\sum_{p=1}^{N} \sum_{m=1}^{n_{\mathrm{y}}} \boldsymbol{K}_{r,(p-1)n_{\mathrm{y}}+m} e_j^{m,n}(p) \quad \text{dla } r = 1, \ldots, n_{\mathrm{u}}; n = 1, \ldots, n_{\mathrm{u}} \\ K_{r,m}^{\mathrm{y}}(j) &= -\sum_{p=1}^{N} \boldsymbol{K}_{r,(p-1)n_{\mathrm{y}}+m} f_j^m(p) \quad \text{dla } r = 1, \ldots, n_{\mathrm{u}}, m = 1, \ldots, n_{\mathrm{y}} \end{aligned} \tag{2.74}$$

Współczynniki $\boldsymbol{K}_{r,(p-1)n_{\mathrm{y}}+m}$ są elementami macierzy $\boldsymbol{K}$ zdefiniowanej we wzorze (2.46).

Podsumowując wersję oszczędną algorytmu GPC, muszą być zrealizowane następujące obliczenia (off-line):

1. Dla danego modelu ARX (2.55) i macierzy współczynników wag $\boldsymbol{M}$ i $\boldsymbol{\Lambda}$, należy obliczyć macierz współczynników odpowiedzi skokowej o strukturze $\boldsymbol{M}$ (równanie (2.27)). Skalarne współczynniki dla poszczególnych torów wejść-wyjść (wzór (2.28)) wyznacza się ze wzoru [71]

$$s_p^{m,n}(k) = \sum_{i=1}^{\min(p,n_{\mathrm{B}})} b_i^{m,n}(k) - \sum_{i=1}^{\min(p-1,n_{\mathrm{A}})} a_i^m(k) s_{p-i}^{m,n}(k) \tag{2.75}$$

   dla wszystkich $m = 1, \ldots, n_{\mathrm{y}}$, $n = 1, \ldots, n_{\mathrm{u}}$ oraz $k = 1, \ldots, N$.
2. Współczynniki $e_i^{m,n}(p)$ oraz $f_i^m(p)$ trajektorii swobodnej należy wyznaczyć przy użyciu wzorów (2.64), (2.65), (2.66) i (2.67).
3. Współczynniki $K_{r,m}^{\mathrm{y^{zad}}}$, $K_{r,n}^{\mathrm{u}}(j)$ oraz $K_{r,m}^{\mathrm{y}}(j)$ analitycznego prawa regulacji należy wyznaczyć z równań (2.74).

W każdej chwili próbkowania obliczenia (on-line) optymalne wartości przyrostów sygnałów sterujących $\triangle u(k|k)$ wyznacza się ze wzoru (2.73). Należy podkreślić, że analogicznie jak w oszczędnym algorytmie analitycznym DMC, także w algorytmie GPC horyzonty predykcji i sterowania nie wpływają na złożoność obliczeniową algorytmu. Należy zauważyć, że złożoność obliczeniowa analitycznego algorytmu GPC zależy wyłącznie od rzędu dynamiki modelu, zdefiniowanego stałymi n_{A} oraz n_{B} i wymiarowości procesu (liczy wejść i wyjść). W przypadku algorytmu analitycznego DMC złożoność obliczeniowa zależy od długości horyzontu dynamiki.

Ponieważ typowo jest on znacznie dłuższy niż rząd modelu ARX, należy oczekiwać, że analityczny algorytm GPC jest mniej złożony obliczeniowo niż analityczny algorytm DMC, czyli obliczenia konieczne do wykonania w każdej chwili próbkowania trwają krócej [24, 25]. Analogicznie jak w analitycznym algorytmie DMC, jeżeli ograniczenia nie mogą być pominięte, należy rozwiązanie otrzymane analitycznie rzutować na zbiór rozwiązań dopuszczalnych (wzory (2.54)).

2.5. Zalety algorytmów regulacji predykcyjnej oraz przykładowe zastosowania

Główne zalety regulacji predykcyjnej są następujące:

1. Można bardzo efektywnie regulować procesy wielowymiarowe, nawet o różnej liczbie wejść i wyjść. Nie ma potrzeby odprzęgania jak w przypadku klasycznych jednopętlowych regulatorów PID. Procedura optymalizacji wyznacza wartości zmiennych decyzyjnych, które umożliwiają minimalizację prognozowanych uchybów regulacji wszystkich sygnałów wyjściowych.
2. Można łatwo uwzględnić ograniczenia sygnałów wejściowych i wyjściowych procesu. Są one po prostu częścią zadania optymalizacji.
3. Podstawowe sformułowanie algorytmu regulacji predykcyjnej jest bardzo uniwersalne. Można zaprojektować algorytmy dla różnych postaci modeli liniowych, np. w przestrzeni stanów, oraz nieliniowych [68, 108].
4. Jeżeli dostępny jest model mierzonych zakłóceń, można w algorytmie uwzględnić wpływ tych zakłóceń.

Należy również podkreślić, że algorytmy regulacji predykcyjnej działają tym lepiej, im lepszy jest model matematyczny regulowanego procesu. Jeżeli model nie jest dostępny lub jest ob bardzo niedokładny, należy zastosować „bezpieczne" regulatory PID.

Uważa się, że algorytmy omawianej klasy są jedyną zaawansowaną techniką regulacji, która odniosła istotny sukces praktyczny [108]. Od ponad 40 lat stosowane są w przemyśle (np. rafinerie, zakłady chemiczne, produkcja papieru, przetwórstwo żywności) [90]. Jako przykładowe procesy można podać: reaktory chemiczne [91], kolumny destylacyjne [120], wymienniki ciepła [99], piece [100], formowanie tworzyw sztucznych [35], maszyny papiernicze [52]. Jak podkreślono we wstępie, klasyczne, przemysłowe zastosowania algorytmów regulacji predykcyjnej charakteryzują się dość długimi okresami próbkowania, rzędu sekund, a nawet minut. Od-

rębną grupę zastosowań można znaleźć w systemach wbudowanych, których cechą charakterystyczną są krótkie (poniżej sekundy) lub bardzo krótkie (dziesiątki lub pojedyncze milisekundy) okresy próbkowania. Przykłady zastosowań algorytmów regulacji predykcyjnej w takich systemach (sterowanie silnikami, pojazdami autonomicznymi, bezzałogowymi statkami powietrznymi i pływającymi itd.) podano w pierwszym rozdziale pracy, s. 10.

Warto również wymienić inne, mniej typowe zastosowania (praktyczne zastosowania oraz symulacje) algorytmów regulacji predykcyjnej:

a) sterowanie robotów [81],
b) sterowanie szklarni [46],
c) sterowanie [10, 17],
d) sterowanie pieca w cementowni [101],
e) sterowanie chłodni w supermarkecie [95],
f) sterowanie oczyszczalni ścieków [20],
g) sterowanie ruchu drogowego [15],
h) sterowanie ruchu w sieci komputerowej [78],
i) sterowanie przepływu gazu w rurociągu [125],
j) sterowanie akceleratora cząstek [18]
h) sterowanie
k) dozowanie insuliny chorym na cukrzycę [70, 110],
l) dozowanie środka znieczulającego podczas operacji [77].
m) sterowanie
n) sterowanie

Algorytmy regulacji predykcyjnej są coraz częściej stosowane w energetyce. Przykładowe zastosowania są następujące:

a) pompy ciepła [9],
b) przekształtniki energii [104],
c) turbiny wiatrowe [51],
d) pojazdy elektryczne [62],
e) turbiny w elektrowni [69],
f) ogniwa paliwowe [45].

Ważnym obszarem zastosowania algorytmów regulacji predykcyjnej są systemy ogrzewania, wentylacji i klimatyzacji. W najprostszym ujęciu celem jest stabilizacja temperatury w budynku (pomieszczeniach) pomimo zmian temperatury zewnętrznej [44, 94, 102], w bardziej wyrafinowanych rozwiązaniach celem jest zapewnie-

nie i utrzymanie komfortu cieplnego [22, 123] (regulacja temperatury, wilgotności i innych czynników), również przy uwzględnieniu potrzeby minimalnego zużycia energii.

Rozdział 3

Implementacja algorytmów regulacji predykcyjnej w systemach sterujących: przegląd istniejących rozwiązań

3.1. Wymagania

Wymagania dotyczące platformy sprzętowej, na której zaimplementowany jest algorytm regulacji predykcyjnej zależą od konkretnego zastosowania. W ogólności, można wyróżnić następujące wymagania

— rodzaj platformy sprzętowej,
— czas obliczeń wymaganych w pojedynczej chwili próbkowania,
— sposób programowania (dostępne języki programowania),
— dostępna pamięć,
— rodzaje interfejsów wejść i wyjść,
— możliwości połączenia w sieć,
— bezpieczeństwo pracy oraz danych
— wymiary zewnętrzne,
— niezawodność,
— odporność na czynniki zewnętrze,
— cena,
— możliwość zasilania bateryjnego, czas pracy przy zasilaniu bateryjnym.

W dalszej części tego rozdziału omówiono 3 najważniejsze platformy sprzętowe, które są stosowane do implementacji zaawansowanych algorytmów obliczeniowych (nie tylko algorytmów regulacji), a mianowicie: programowalne sterowniki logiczne (PLC) lub sterowniki przemysłowe, mikrokontrolery.

3.2. Sterowniki programowalne

Programowalne sterowniki logiczne są powszechnie stosowanymi urządzeniami w branży automatyki przemysłowej. Ich wyspecjalizowana budowa jest dostosowa-

na do wymagań stawianych przez przemysł. Bardzo ważną cechą sterowników jest odporność na zakłócenia elektromagnetyczne występujące w strefie zakładów przemysłowych. Należy do nich zaliczyć skokowe włączanie i wyłączanie silników asynchronicznych klatkowych dużej mocy, sterowanie silników asynchronicznych dużej mocy przy pomocy techniki falownikowej, sterowanie silników typu serwo dużej mocy. Stosowanie odpowiednich filtrów elektromagnetycznych dla takich urządzeń jest oczywiście obligatoryjne, niemniej jednak często takie filtry są pomijane, szczególnie w urządzeniach starszej generacji.

Drugą cechą sterowników, jaką należy podkreślić, to dostosowanie do utartego od bardzo dawna standardu sygnałowego. Chodzi tutaj o sterowanie logiczne, gdzie referencją dla sygnałów zero jedynkowych jest napięcie stałe +24V. Jest to napięcie bezpieczne dla zdrowia człowieka, a jednocześnie odpowiednio wysokie, aby minimalizować wpływ zakłóceń elektromagnetycznych. Drugi standard dotyczy sygnałów analogowych, gdzie zakres napięć mieści się w zakresie ± 10V. Ten standard wymagany jest do regulacji ciągłej. Bardzo często ten sygnał wykorzystuje się do sterowania prędkości serwonapędu, sterowania momentu wyjściowego serwonapędu lub też sterowania częstotliwości wyjściowej falownika.

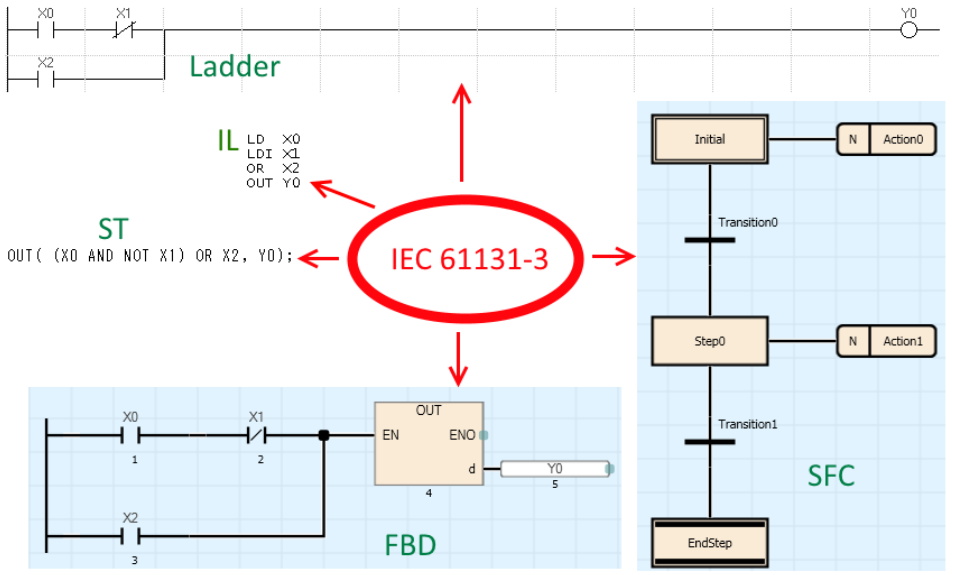
Z praktycznego punktu widzenia bardzo ważna jest budowa mechaniczna sterowników programowalnych. Standardem jest połączenie w postaci zacisku śrubowego lub sprężynowego. Jest to podejście dostosowane do łatwego łączenia wielu przewodów zbierających sygnału z maszyny. Wszystkie sterowniki programowalne mają obudowę zewnętrzną wykonaną z tworzywa sztucznego z otworami wentylacyjnymi. Obudowa uwzględnia praktycznie zawsze występujące diody świecące do celów diagnostycznych. Czasami można spotkać wyświetlacze ciekłokrystaliczne, które umożliwiają prezentację większej ilości danych do szybkiej diagnostyki sterownika w przypadku wystąpienia usterek.

Sterowniki mogą być programowane językami uwzględnionymi w normie IEC61131-3. W tabeli 3.1 przedstawiono wszystkie języki programowania sterowników wraz z krótkim opisem najważniejszych cech. Na rys. 3.1 przedstawiono przykłady wykorzystania tych języków.

Obecnie, dzięki rozwiniętej technologii sieciowej szerokie zastosowanie znajduje realizacja systemów rozproszonych. W takim podejściu sterownik PLC nie posiada już własnych wejść wyjść (lub też ich liczba jest zmniejszona do minimum). Wszystkie sygnały są doprowadzane do specjalnych modułów wejść i wyjść, które znajdują się w określonych częściach maszyny, pozycje te określane są mianem wysp. Ko-

Tabela 3.1. Języki programowania sterowników programowalnych

Język	Rodzaj	Opis
Ladder Diagram (LD), nazwa polska: Schemat Drabinkowy	Graficzny	Program w postaci przełożonego schematu elektrycznego ze stykami przekaźników
Function Block Diagram (FBD), nazwa polska: Schemat Bloków Funkcyjnych	Graficzny	Rozwinięta wersja schematu drabinkowego, uzupełniona o bardziej złożone bloki funkcyjne
Sequential Function Chart (SFC), nazwa polska: Sekwencyjna Lista Funkcji	Graficzny	Program opiera się o diagram stanów, warunki i akcje realizowane w języku drabinkowym
Structured Text (ST), nazwa polska: Tekst Strukturalny	Tekst	Język skryptowy podobny do języka Pascal
Instruction List (IL), nazwa polska: Lista Instrukcji	Tekst	Język niskiego poziomu, podobny do języka assembler



Rys. 3.1. Przykłady wykorzystania tych języków programowania sterowników programowalnych

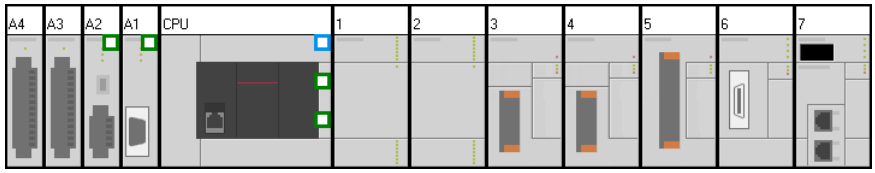
munikacja ze sterownikiem głównym odbywa się szybką magistralą np. Ethernet z wykorzystaniem dedykowanego protokołu, najczęściej autorstwa danego producenta komponentów. Trend ten pozwala zaoszczędzić wiele kilometrów przewodów, których nie trzeba już sprowadzać do jednej szafy zbiorczej. Ponadto, w przypadku sygnałów analogowych krótsze odległości wpływają pozytywnie na jakość ich przetwarzania. Istnieje również wiele specjalizowanych czujników np. czujnik odległości wykorzystuje promień lasera do pomiaru odległości od przedmiotu, obliczenie wartości odległości dzieje się w samym czujnika, a wysyłane w postaci cyfrowej są już gotowe wartości pomiarowe. Oznacza to, że komunikacja może być zrealizowana bezpośrednio przez sieć Ethernet z zachowaniem protokołu producenta sterownika głównego, do którego trafiają pomiary.

Sterowniki PLC można podzielić na dwie grupy: sterowniki kompaktowe i sterowniki modułowe. Sterowniki kompaktowe mają następujące cechy:

— stopień skomplikowania dedykowanych maszyn: proste i średnie,
— szybkość przetwarzania programu nie jest krytyczna,
— wejścia i wyjścia cyfrowe wbudowane (z możliwością rozbudowy o moduły dodatkowe),
— wejścia i wyjścia analogowe wbudowane (z możliwością rozbudowy o moduły dodatkowe),
— wyłącznie jeden procesor sterujący,
— ograniczone możliwości rozbudowy systemu,
— ograniczona wielkość pamięci programu,
— brak płyty bazowej dla modułów dodatkowych.

Przykładowy sterownik kompaktowy FX5U produkcji Mitsubishi Electric wraz z modułami dodatkowymi przedstawiono na rys. 3.2. Po lewej i prawej stronie sterownika można zobaczyć przykładową konfigurację modułów dodatkowych. Szczegółowa lista modułów jest następująca:

— sterownik PLC (CPU): FX5U-32MR/ES,
— moduł 1 wejść cyfrowych: FX5-8EX/ES,
— moduł 2 wyjść cyfrowych: FX5-8EYR/ES,
— moduł 3 wejść analogowych: FX5-4AD,
— moduł 4 wyjść analogowych: FX5-4DA,
— moduł 5 regulacji temperatury: FX5-4LC,
— moduł 6 sterowania serwonapędami: FX5-40SSC,
— moduł 7 komunikacji sieci CC-Link IE Field: FX5-CCLIEF,



Rys. 3.2. Przykładowy sterownik kompaktowy PLC wraz z modułami dodatkowymi

— moduł A1 komunikacji RS232: FX5-232ADP,
— moduł A2 komunikacji RS485ADP: FX5-485ADP,
— moduł A3 wejść analogowych: FX5-4AD-ADP,
— moduł A4 wejść typu termopara: FX5-4AD-TC-ADP.

Sterowniki kompaktowe znajdują coraz szersze zastosowanie w przemyśle. Można za ich pomocą zrealizować aplikacje wymagające podstawowego sterowania logicznego, regulacji temperatury, sterowania falownikami, zaawansowanej kontroli układów serwonapędowych łącznie z synchronizacją w czasie rzeczywistym, komunikacji sieciowej na płaszczyźnie wielu protokołów, logowania i archiwizacji danych. Ograniczeniami takich sterowników jest pamięć, która w przypadku sterownika FX5U wynosi 64k kroki programu. Przy bardziej rozbudowanej aplikacji ta wielkość może stanowić granicę nie do pokonania. Kolejna sprawa to tylko jedna jednostka obliczeniowa, a więc braj jest jakiejkolwiek możliwości prowadzenia obliczeń równolegle. Jeżeli chodzi o języki programowania to dostępne są: język drabinkowy, język skryptowy ST, język bloków funkcyjnych FBD. Istotną zaletą jest możliwość operowania na liczbach zmiennoprzecinkowych pojedynczej precyzji, dzięki czemu implementacja zaawansowanych algorytmów obliczeniowych jest możliwe.

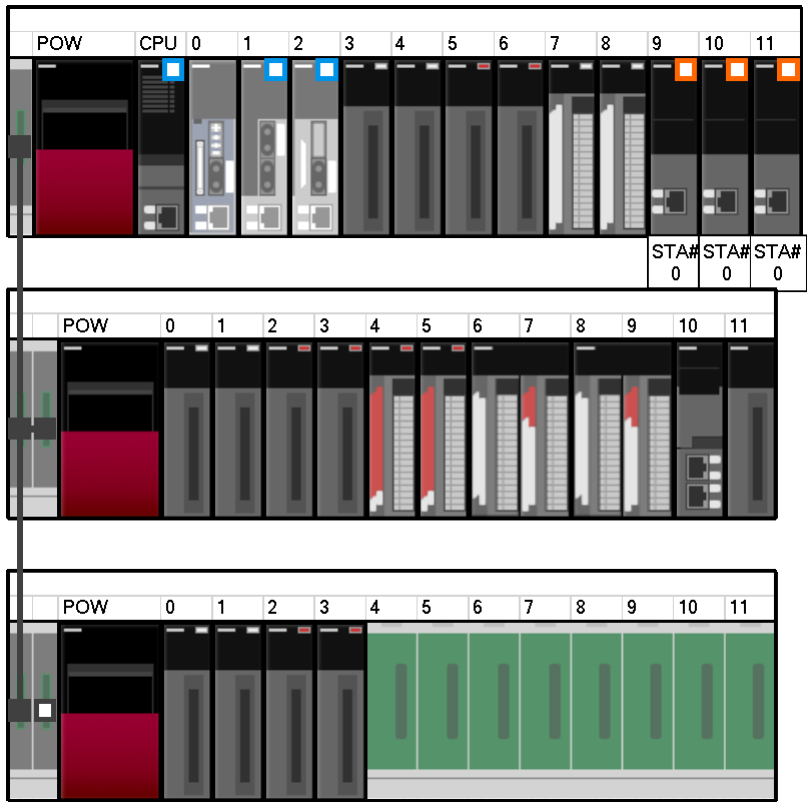
Sterowniki modułowe mają następujące cechy:
— stopień skomplikowania dedykowanych maszyn: średnie i zaawansowane,
— szybkość przetwarzania programu jest krytyczna,
— wejścia i wyjścia cyfrowe tylko jako moduły dodatkowe,
— wejścia i wyjścia analogowe tylko jako moduły dodatkowe,
— dostępna wieloprocesorowość (PLC, sterownik ruchu, sterownik robota, sterownik CNC, procesor C),
— wbudowane systemy bezpieczeństwa (Safety PLC),
— specjalizowane moduły: komunikacja, sterowanie ruchem, sterowanie temperaturą itp.,
— dedykowana płyta bazowa do montażu modułów.

Przykładowy sterownik modułowy iQ-R produkcji Mitsubishi Electric wraz z

modułami dodatkowymi przedstawiono na rys. 3.3. Należy podkreślić duże możliwości rozbudowy systemu przy wykorzystaniu rozszerzających płyt bazowych. Znaczącą przewagą w stosunku do sterowników kompaktowych jest również możliwość stosowania wielu różnych procesorów na płycie bazowej. W przedstawionym rozwiązaniu w jednym systemie dysponujemy procesorem PLC o 120k krokach programu, procesorem ruchu dedykowanym dla robota przemysłowego, procesorem ruchu pozwalającym synchronizować w czasie rzeczywistym do 64 napędów serwo oraz procesorem CNC pozwalającym sterować do 8 tokarek numerycznych. Szczegółowa lista modułów jest następująca:

— zasilacz,
— procesor CPU PLC: R120CPU,
— procesor 0 ruchu: R64MTCPU,
— procesor 1 robot: R16RTCPU,
— procesor 2 CNC: R16NCPCU,
— moduł 3 wejść cyfrowych: RX41C4,
— moduł 4 wejść cyfrowych: RX41C4,
— moduł 5 wyjść cyfrowych: RY41PT1P,
— moduł 6 wyjść cyfrowych: RY41PT1P,
— moduł 7 wejść analogowych: R60ADV8,
— moduł 8 wejść analogowych: R60ADV8,
— moduł 9 sterowania serwonapędami: RD77GF16,
— moduł 10 sterowania serwonapędami: RD77GF16,
— moduł 11 sterowania serwonapędami: RD77GF16.

Konieczność ość zastosowania sterownika modułowego najczęściej związane jest z maszyną o wygórowanych parametrach pracy lub też taką, która stanowi złożenie kilku maszyn. Ma to miejsce np. gdy rozważanym procesem jest cała linia produkcyjna. Bardzo duża szybkość wymiany informacji między procesorami, zapewniona przez dedykowaną płytę bazową, pozwala na realizację zadań w reżimie pojedynczych milisekund. Praca równoległa jest możliwa przez zastosowanie aż do 4 całkowicie niezależnych procesorów. Przykładowo, jeden sterownik PLC realizuje podstawą funkcję sterowania logicznego części maszyny, natomiast drugi sterownik PLC realizuje ściśle określone zadanie regulacji krytycznego procesu występujące w tej maszynie. Dzięki takiemu podziałowi można zapewnić najwyższy stopień niezawodności działania, a jednocześnie mieć szybką wymianę danych z procesorem logicznym. Ponadto, system może być rozbudowany o kolejny procesor ruchu (ste-



Rys. 3.3. przykładowy sterownik modułowy PLC wraz z modułami dodatkowymi

rowanie serwonapędami), pozwalający na całkowicie niezależne sterowanie aż do 64 napędów serwo. Dla takiego procesora tworzy się niezależny program realizujący komendy pozycjonowania silników serwo. Jego możliwości pozwalają także na już wspomniane uruchomienie pełnej synchronizacji wszystkich 64 napędów realizowanej w czasie rzeczywistym. Można sobie wyobrazić jeden sygnał wzorcowy podawany przez enkoder inkrementalny do procesora ruchu. Trafia on następnie jako komenda dowolnie wybranego silnika serwo. Aby zapewnić komfort kontroli synchronizacji między odpowiednimi silnikami są dostępne przekładnie elektroniczne, sprzęgła, mechanizmy różnicowe, krzywki dynamiczne. Aplikacja może być również wzbogacona o procesor robota przemysłowego, który może posiadać do 6 osi swobody. Warto podkreślić, że ruch robota może być synchronizowany do serwonapędu sterowanego z procesora ruchu, co zapewnia szybką wymianę danych przez płytę bazową oraz wbudowane mechanizmy synchronizacji. Aby zapewnić pełną integrację systemu sterowania możliwe jest również zastosowanie procesora CNC, który interpretuje instrukcje zapisane w języku GCODE. Proces obróbki jaki realizuje maszyna może być dowolnym z grupy: skrawania, cięcia plazmą, cięcia laserem, nakładania kleju.

Należy zwrócić uwagę na możliwości procesora, zastosowanego w sterownikach programowalnych. Pierwsza ważna cecha to możliwość realizacji operacji zmiennoprzecinkowych podwójnej precyzji, dzięki czemu możliwe jest wykonywanie obliczeń z dużą dokładnością. Druga cecha to możliwość implementacji ponad jednego miliona kroków programu. Zachowany jest przy tym czas realizacji jednej instrukcji typu LOAD (czytaj stan wejścia cyfrowego) rzędu nanosekund. Dostępne są dwa rodzaje pamięci dla danych: pamięć szybka (odpowiednik pamięci RAM) – dostępne kilkaset KB; pamięć plikowa (odpowiednik dysku twardego realizowanego jako karta SD) – dostępne rzędu kilkuset MB. W odniesieniu do mikrokontrolerów wielkość dostępnej pamięci można uznać za porównywalną. Natomiast w przypadku układów FPGA, gdzie można podłączyć dedykowaną magistralą pamięć RAM typu DDR3, jest to przeważająca zaleta.

Porównując wydajność sterowników wyróżnia się czas przetwarzania podstawowej instrukcji czytania wejścia LD (load - pobranie stanu wejścia cyfrowego). Dla sterownika kompaktowego FX5 czas przetwarzania tej instrukcji wynosi 35 ns, dla sterownika modułowego iQ-R wynosi 0.98 ns czyli 35 razy mniej. W pracy [33] przedstawiono wyniki czasowe obliczeń dla algorytmu DMC (proces świetlny), natomiast w pracy [33] wyniki dotyczące rozmytego algorytmu PID (proces

Tabela 3.2. Czasy obliczeń w każdej chwili próbkowania algorytmów regulacji zaimplementowanych na sterowniku programowalnym iQ-R: algorytm DMC (dla procesu świetlnego) i algorytm rozmyty PID (dla procesu lewitacji magnetycznej); D – horyzont dynamiki w algorytmie DMC

Algorytm	Okres próbkowania [ms]	D	Czas obliczeń [ms]
DMC 1x1	5	25	0.1
DMC 1x1	5	50	0.2
DMC 1x1	5	150	0.3
DMC 2x2	5	25	0.4
DMC 2x2	5	50	0.7
DMC 2x2	5	150	2.0
FPID	1	-	0.34

lewitacji magnetycznej), które zostały zaimplementowane na sterowniku iQ-R (badania zostały wykonane przez autora niniejszej rozprawy). Dla algorytmu DMC przedstawiono przykładowe horyzonty dynamiki i czasy obliczeń wykonywanych w pojedynczej chwili próbkowania algorytmów, odpowiednio dla wersji jednowymiarowej i wielowymiarowej. Okres próbkowania wynosił 5 ms. W przypadku algorytmu DMC dla obiektu z dwoma wejściami i dwoma wyjściami, przy horyzoncie dynamiki $D = 150$, czas obliczeń dla pojedynczej chwili próbkowania wynosił aż 40% okresu próbkowania. W przypadku regulatora rozmytego PID wymagania czasowe były bardzo rygorystyczne dla otrzymania dobrej jakości regulacji. Stąd też okres próbkowania ustawiony został na 1 ms, a czas obliczenia w każdej chwili próbkowania wynosił 0.34ms.

W pracy [41] przedstawiono implementację regulatora predykcyjnego typu explicit MPC [16] (algorytm bazuje na modelu liniowym, w celu eliminacji konieczności optymalizacji uwzględnia się zbiór algorytmów analitycznych, pracujących lokalnie), wykonaną dla sterownika programowalnego. Zastosowano sterownik Siemens Simatic S7-200. Pokazano, że sterownik jest w stanie realizować algorytm regulacji predykcyjnej. W wyniku zastosowania specyficznego, dość rozbudowanego, algorytmu, zostało bardzo mało pamięci dla użytkownika do implementacji innych operacji realizowanych przez sterownik. Czas próbkowania wynosił 250 ms. W pracy [56] przedstawiono porównanie dwóch systemów sterowników programowalnych z PAC (Programmable Automation Controllers). Zastosowano sterowniki PAC Compact RIO od National Instruments oraz PLC S7-319 od Siemensa. Testowano implementację algorytmu regulacji predykcyjnej z modelem w przestrzeni stanu w wersji numerycznej. Testowano trzy biblioteki programowania kwadratowego: qpOASES,

Hildreth, CVXGEN. W obu przypadkach rozważano proces regulacji temperatury powietrza.

Wykorzystanie sterownika programowalnego do implementacji regulatora predykcyjnego temperatury w wersji numerycznej opisano w pracy [111]. Głównym celem było wykorzystanie minimalnej informacji o procesie oraz zastosowanie automatycznego dostrajania algorytmu. Można również znaleźć przykład implementacji algorytmów regulacji predykcyjnej przy wykorzystania sterownika programowalnego, przy użyciu własnych funkcji operacji na macierzach (język ST) [106]. Istnieje również przykład implementacji złożonego obliczeniowo nieliniowego algorytmu regulacji predykcyjnej ma sterowniku programowalnym [61]. Udało się zastosować czas próbkowania rzędu milisekund, rozważano regulację procesu odwróconego wahadła. W pracy [55] opisano zastosowanie sterownika PLC do regulacji przemysłowej kolumny destylacyjnej, zastosowano numeryczny algorytm regulacji predykcyjnej wykorzystujący do predykcji modele liniowe w przestrzeni stanu.

W odniesieniu do założeń zdefiniowanych w podrozdziale 3.1, cechy sterowników programowalnych można podsumować następująco:

— czas obliczeń wykonywanych w pojedynczej chwili próbkowania: możliwe do uzyskania pojedyncze milisekundy,
— sposób programowania: kilka dostępnych języków programowania,
— dostępna pamięć: kilka MB,
— rodzaje interfejsów wejść i wyjść: duża różnorodność,
— możliwości połączenia w sieć: rozbudowane możliwości,
— bezpieczeństwo pracy oraz danych: na wysokim poziomie,
— wymiary zewnętrzne, stosunkowo małe, ale nie bardzo małe,
— niezawodność: wysoka,
— odporność na czynniki zewnętrze: duża,
— cena: wysoka,
— zasilanie bateryjne: praktycznie nie stosowane.

Podsumowując, należy podkreślić, że sterowniki programowalne spełniają określone wymagania, dzięki czemu można je stosować w niekorzystnych warunkach zakładów przemysłowych. Grupą docelową obiektów sterowania są różnego rodzaju maszyny ale również duże linie technologiczne. Ich zastosowanie w warstwie sterowania logicznego nie wymaga dużych szybkości przetwarzania informacji. Zaawansowane procesy potrzebujące krótkich czasów regulacji mogą być również sterowane przez PLC, dzięki nowoczesnym rozwiązaniom technologicznym. Rozległe

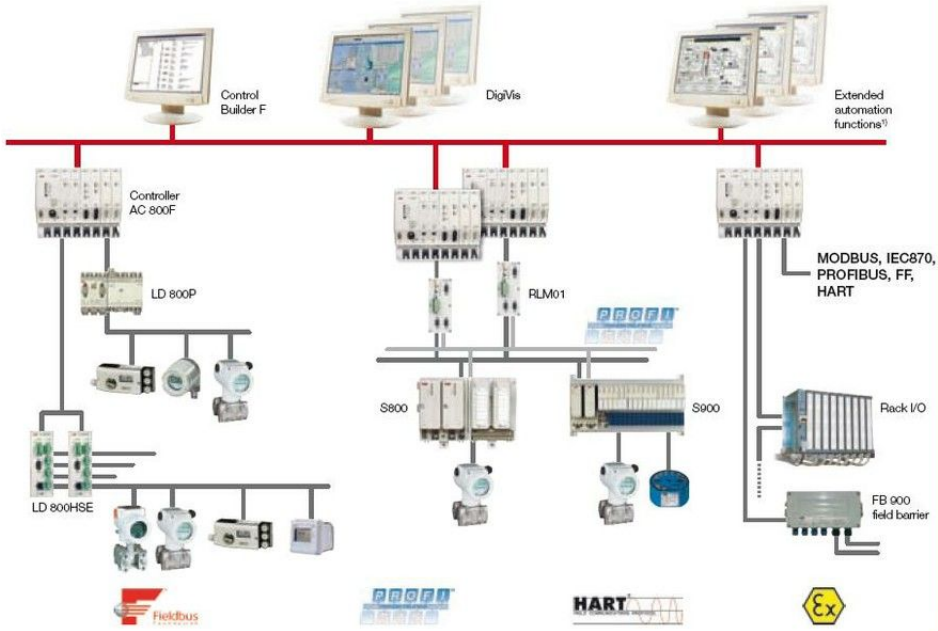
w przestrzeni systemy mogą być sterowane i diagnozowane bardzo szybko dzięki zastosowaniu deterministycznych sieci opartych o Ethernet (czas wymiany setek rejestrów w takiej sieci zajmuje niecałą milisekundę). Wadą profesjonalnych sterowników programowalnych są dość wysokie ceny. Dlatego nie są one stosowane w zastosowaniach amatorskich, gdzie typowo stosuje się mikrokontrolery (gotowe płytki rozwojowe). Zastosowanie sterownika programowalnego w systemach wbudowanych, np. w pojazdach autonomicznych również nie jest akceptowalne z uwagi na wygórowane wymagania szczelności i odporności na wilgoć (sterownik programowalny jest urządzeniem otwartym o niskim stopniu odporności IP). Jego wielkość również zdecydowanie przewyższa wymiary komputerów pokładowych stosowanych w samochodach.

3.3. Sterowniki przemysłowe

Sterowniki przemysłowe pracujące w rozproszonych systemach sterowania (ang. Distribured Control System, DCS) charakteryzują się przede wszystkim rozproszoną budową. Są stosowane w złożonych systemach (tysięcy wejść i wyjść). Dla wygody pracy operator korzysta z aplikacji przygotowanej na duże monitory przemysłowe, które pozwalają zawrzeć wiele informacji niezbędnych do monitorowania pracy bardzo złożonego procesu, np. linii destylacji ropy naftowej lub całej fabryki. Z uwagi na bardzo wygórowane potrzeby sterowniki takie muszą być przede wszystkim działać niezawodne. Typowo, sterowniki przemysłowe wyposażone są w możliwość pracy redundantnej, aby w razie uszkodzenia jednego kontrolera krytyczny proces mógł być dalej sterowany i nadzorowany [87]. Do podstawowych kryteriów wyboru tych sterowników należy zaliczyć:

— bezpieczeństwo ludzi,
— bezpieczeństwo środowiska,
— bezpieczeństwo sprzętu,
— stabilność w działaniu,
— najwyższa jakość produktu,
— optymalizacja zysków,
— monitorowanie,
— diagnostyka.

Przykładowa struktura systemu DCS przedstawiona została na rys. 3.4. Można w niej wyróżnić 3 stacje komputerowe: operatora, technologa oraz dewelopera. Trzy



Rys. 3.4. Przykładowa struktura sieci sterownika przemysłowego produkcji ABB

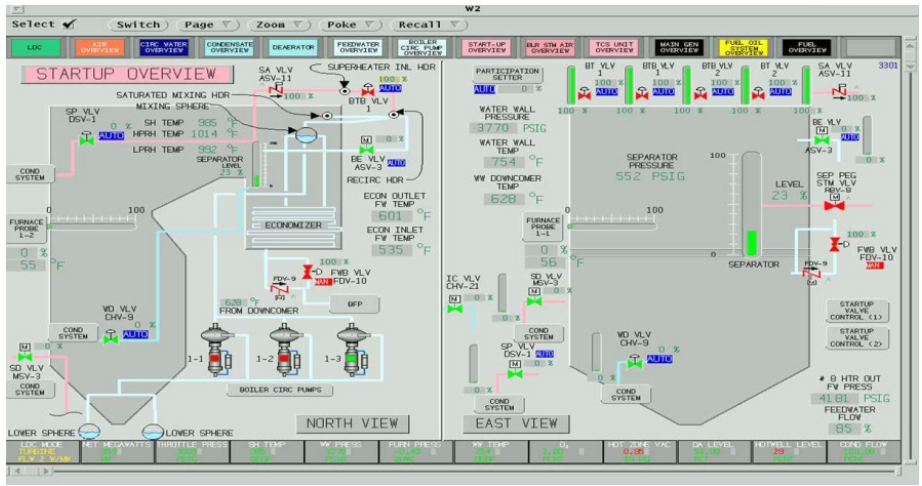
sterowniki przemysłowe połączone są dedykowaną siecią przemysłową. Do każdego sterownika podłączone są dodatkowe sterowniki z różnych fragmentów obiektu regulowanego. Ponadto, każde urządzenie może komunikować się różnymi sieciami jak na przykład Fieldbus, Profinet czy HART. Przykładowe użycie algorytmu regulacji predykcyjnej w energetyce na bazie sterownika firmy ABB można znaleźć w pracach [43, 57].

Rys. 3.4: wstawić plik o wyższej rozdzielczości.

Bardzo ważnym elementem systemu, w skład którego wchodzą sterowniki przemysłowe, są systemy informatyczne nadzoru oraz gromadzenia danych (ang. Supervisory Control And Data Acquisition, SCADA). Przykładowy panel operatora przedstawiono na rys. 3.5. Stosuje się typowe rozwiązania, np. grafiki w odcieniach szarości, elementy aktywne w kolorach czerwonym lub zielonym, wartości numeryczne i wskaźniki poziomu koloru ciemno zielonego, przyciski do przejścia na inne ekrany w różnych kolorach [121].

Każdy sterowniki przemysłowy powinien posiadać następujące własności:

— zdefiniowanie bloki funkcjonalne dla programistów,



Rys. 3.5. Przykładowy panel operatora systemu DCS

— dostęp do najnowszej dokumentacji całego systemu z poziomu stacji inżynierskiej,
— rejestracja wszystkich punktów i zdarzeń w dużym przedziale czasu,
— redundancja kontrolerów, wejść, wyjść, stacji operatorskich,
— możliwość zmiany programu bez zatrzymania pracy systemu,
— obsługa dużych obiektów, gdzie liczba sygnałów wejściowych i wyjściowych może sięgać dziesiątek tysięcy,
— jednoczesny zapis programu z kilku stacji inżynierskich,
— możliwość podłączenia urządzeń w dowolnym standardzie komunikacyjnym.

Przykładowy sterownik przemysłowy firmy Emerson [86] wyposażony jest w procesor Intel Celeron z częstotliwością taktowania zegara 400MHz. Wykorzystany został system operacyjny czasu rzeczywistego VxWorks. Pamięć programu wynosi 128MB, pamięć RAM wynosi do 128MB. Możliwe jest skorzystanie z 5 procesów o okresach wywołania 100ms, 1000ms, i trzech konfigurowalnych od 10ms do 30s. System jest w pełni redundantny, z funkcją Watchdog. Dostępne są interfejsy do komunikacji, między innymi, ze sprzętem takich producentów jak Allen-Bradley, Toshiba czy GE.

W pracy [88] przedstawiono historyczne początki regulacji predykcyjnej w systemach DCS, podkreślono dynamiczny rozwój algorytmów u każdego producenta systemu. W pracy [92] przedstawiono realizację algorytmu predykcyjnego kontrolującego poziom cieczy w zbiorniku. Badano wpływ częstotliwości próbkowania oraz ilości zmian sygnału zadanego na ostateczną jakość regulacji. Z kolei praca [12] pokazuje jakie ograniczenia posiada sterownik DCS firmy Siemens PCS7 w zakresie biblioteki gotowych rozwiązań. Na tej podstawie zaproponowano nowe bloki funkcyjne rozszerzające możliwość stosowania algorytmów MPC w wymagających procesach - zrealizowano tylko testy symulacyjne.

Systemy DCS są dedykowane konkretnym aplikacjom, a przez to są w pewien sposób ograniczone. Użytkownik końcowy czy programista nie ma dostępu do wszystkich funkcji systemu. Może wykorzystywać tylko gotowe szablony (kreatory) przygotowane przez producenta. Z tego punktu widzenia sterownifki przemysłowe DCS są porównywalne do sterowników programowalnych. Obie te grupy natomiast są zdecydowanie odmienne od mikrokontrolerów, gdzie użytkowników ma dostęp do warstwy bardzo niskiego poziomu. Z punktu widzenia inżyniera procesu istnienie i możliwość łatwego wykorzystania szablonów różnych regulatorów są zdecydowaną zaletą. Umożliwia to szybką implementację regulatorów złożonych procesów,

dla których gdzie liczba regulatorów może wynosić kilkaset. Dla takiego procesu implementacja ręczna lub półautomatyczna algorytmów regulacji będzie problematyczna.

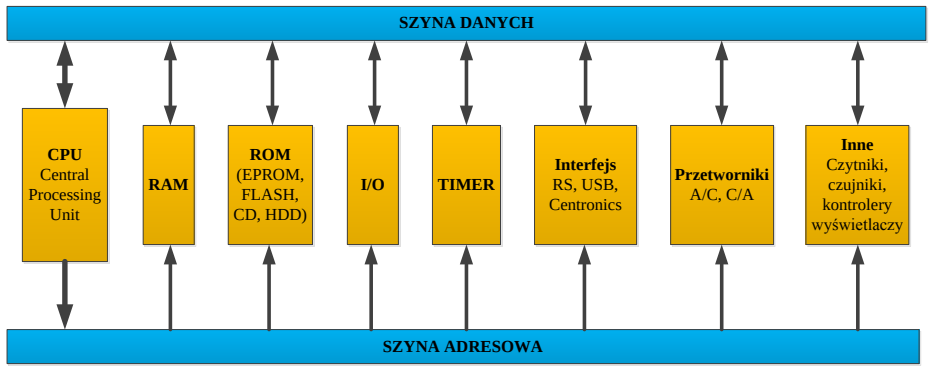
W odniesieniu do założeń zdefiniowanych w podrozdziale 3.1, cechy sterowników przemysłowych można podsumować następująco:

— czas obliczeń wykonywanych w pojedynczej chwili próbkowania: możliwe do uzyskania pojedyncze milisekundy,
— dostępna pamięć: kilkadziesiąt MB,
— rodzaje interfejsów wejść i wyjść: duża różnorodność.
— możliwości połączenia w sieć: rozbudowane,
— bezpieczeństwo pracy oraz danych: na bardzo wysokim poziomie,
— wymiary zewnętrzne: duże,
— niezawodność: bardzo wysoka z zachowaniem redundancji,
— odporność na czynniki zewnętrze: duża,
— cena: bardzo wysoka,
— zasilanie bateryjne: praktycznie nie stosowane.

Podsumowując, należy podkreślić, że sterowniki przemysłowe stosuje się w bardzo złożonych systemach sterujących procesów dużej skali, podczas gdy sterowniki programowalne stosuje się w procesach mniejszej skali. Sterowniki programowalne są częścią złożonych rozproszonych, sieciowych systemów sterowania. Ich elementem bardzo często jest system SCADA. Złożone systemy informatyczne oferują możliwość łatwej implementacji setek algorytmów. Należy podkreślić, że sterowniki przemysłowe są bardzo drogie, ich zakup przez osobę prywatną jest praktycznie niemożliwe. Zastosowanie sterowników przemysłowych w systemach wbudowanych nie jest stosowane.

3.4. Mikrokontrolery

Mikrokontrolery, zwane również mikrokomputerami jednoukładowymi, są powszechnie stosowane w systemach wbudowanych. Jest to konsekwencją możliwości technicznych, atrakcyjnej ceny oraz możliwości opracowania dedykowanego rozwiązania. Mikrokontrolery charakteryzują się bardzo zwartą budową i małymi wymiarami. Są to pojedyncze układy scalone o bardzo dużej skali integracji. Każdy współcześnie dostępny mikrokontroler obejmuje następujące jednostki:



Rys. 3.6. Podstawowe moduły mikrokontrolera

— główną jednostkę obliczeniową i przetwarzająca program (ang. Central Processing Unit, CPU),
— pamięć danych o szybkim dostępie (ang. Random Access Memory, RAM),
— pamięć programu (ang. Read Only Memory, ROM), najczęściej jest to pamięć typu EPROM, FLASH,
— fizyczne porty wejść i wyjść, obecnie każdy pin fizyczny jest konfigurowalny, tzn. może być wejściem lub wyjściem,
— zegar (ang. timer): precyzyjne liczniki czasu pozwalające odliczać zadane czasy i taktować inne podzespoły, jak na przykład przetworniki analogowo-cyfrowe,
— interfejsy komunikacyjne: są to dedykowane urządzenia znajdujące się w strukturze scalonej służące do komunikacji, np. w standardzie UART, USB lub Ethernet,
— inne wbudowane podzespoły: np. przetworniki analogowo-cyfrowe i cyfrowo-analogowo, czujniki różnych wielkości fizycznych, kontrolery wyświetlaczy itp.,
— kontroler przerwań, który zarządza przerwaniami z wejść i liczników czasu.

Programowanie mikrokontrolera jest zazwyczaj realizowane przy użyciu programatora szeregowego ze złączem SPI lub programatora/debuggera ze złączem JTAG. Na rys. 3.6 przedstawiono schemat budowy typowego mikrokontrolera z uwzględnieniem przedstawionych modułów, zaznaczono szynę danych i adresową. Szczegóły budowy wewnętrznej mikroprocesorów są różne dla każdego producenta oraz poszczególnych modeli danego producenta. Przykładowo, popularny mikrokontroler serii 8051 występuje w ponad 1100 odmianach.

Głównym kryterium, uwzględnianym podczas wyboru mikrokontrolera przeznaczonego do złożonych obliczeń, jest liczba przetwarzanych instrukcji w ciągu jednej sekundy (ang. Million Instructions Per Second, MIPS). Jest ona bezpośrednio zależna od częstotliwości taktowania kwarcu podłączonego do układów zegarowych mikrokontrolera. Kolejnymi kryteriami są wielkości pamięci programu i pamięci operacyjnej, a także dostępne interfejsy, przetworniki itd. Niewątpliwą cechą wspólną dla mikrokontrolerów (niezależnie od producenta) jest uniwersalność posiadanych interfejsów. Jest to niewątpliwa zaleta w stosunku do sterowników PLC i kontrolerów przemysłowych, gdzie każdy producent oferuje własne rozwiązanie sieciowe w większości przypadków zamknięte. Przykładowe rodzaje interfejsów wykorzystywanych w mikrokontrolerach to:

— USART: synchroniczny lub asynchroniczny interfejs szeregowy,
— I2C: protokół szeregowy komunikacji między mikrokontrolerami i urządzeniami występującymi w obrębie płytki drukowanej systemu, występuje w nim szyna zegarowa i jednobitowa magistrala danych,
— SPI: protokół szeregowy komunikacji między mikrokontrolerem i urządzeniami występującymi w obrębie płytki drukowanej systemu, występuje w nim szyna zegarowa i dwukierunkowa jednobitowa magistrala danych. Dodatkowo występuje selekcja urządzenia podrzędnego przez dodatkowe wyjście.
— One Wire: magistrala wykorzystywana do komunikacji z czujnikami lub pamięcią EEPROM, występuje tylko jedna linia danych, urządzenia podrzędne są wywoływane przez unikalny adres.

Mikrokontrolery są bardzo często stosowane, nie tylko przez przez hobbystów, ale są powszechnie wykorzystywane w rozwiązaniach komercyjnych. Stosunkowo niskim kosztem można zbudować sterowniki najróżniejszych urządzeń. Obecnie bardzo popularne są moduły sterowania inteligentnych budynków, czy też jako jednostki centralno-sterujące różnych urządzeń, np. wzmacniacza lampowego. Mikrokontrolery, nie tylko ze względu na niski koszt i duże możliwości, ale również z uwagi na małe wymiary, znajdują zastosowanie jako sterowniki dronów oraz robotów mobilnymi o niewielkich wymiarach. Aplikacja sterowania całego roju dronów przedstawiona została w pracy [124]. Każdy dron niezależnie posiada mikrokontroler do odbierania i utrzymywania zadanych pozycji. Wszystkie drony są natomiast sterowane z centralnego komputera, który śledził obiekty kamerą i wysyłał odpowiednie wartości zadane. Mikrokontrolery świetnie też sprawdzają się w funkcji sterującej innych procesów, np. zastosowanie mikrokontrolerów w konstrukcji sta-

nowiska laboratoryjnego z szerokim wachlarzem możliwości przedstawiono w pracy [114] (jest to oryginalna konstrukcja autora rozprawy). W omawianym rozwiązaniu mikrokontroler idealnie spełnił zadanie sterowania układów wykonawczych oraz zbierania informacji pomiarowych. Zastosowane zostały różne możliwości komunikacji, takie jak USB, sygnały analogowe napięciowe, jak również komunikacja szeregowa RS482 z protokołem MODBUS RTU. Bardzo ciekawe praktyczne wyniki implementacji algorytmu DMC na mikrokontrolerach dedykowanych do pracy w warunkach czasu rzeczywistego przedstawiono w artykule [32]. Otrzymane wyniki pokazują dwie wersje analitycznego algorytmu DMC: klasyczną i oszczędną. Dzięki zastosowaniu algorytmu oszczędnego czas obliczeń algorytmu jest ok. 100 krótszy w porównaniu z wersją klasyczną. W algorytmie oszczędnym możliwe jest zastosowanie bardzo krótkich okresów próbkowania, rzędu pojedynczych mikrosekund.

W pracy [75] omówiono zastosowania mikrokontrolerów do implementacji algorytmu regulacji predykcyjnej sterującym miniaturowym pojazdem. Nieliniowy model pojazdu w przestrzeni stanu jest cyklicznie linearyzowany, co prowadzi do kwadratowego zadania optymalizacji. W pracy [25] omówiono implementację algorytmów analitycznych DMC i GPC, podano zastosowanie do sterowania serwomechanizmu laboratoryjnego, przedstawiono szczegółowe badania dotyczące wpływu parametrów (horyzontów) na czas obliczeń. W pracy [126] omówiono implementację numerycznego algorytmu regulacji predykcyjnej, w którym do predykcji użyto liniowy model w przestrzeni stanu, podano wyniki dotyczące sterowania niestabilnego dwukołowego robota mobilnego. Praca [64] omawia zastosowanie numerycznego algorytmu regulacji predykcyjnej do sterowania miniaturowego helikoptera. Zastosowano nieliniowy model, ale jest on cyklicznie linearyzowany, co umożliwia sformułowanie zadania optymalizacji kwadratowej.

Dostępne są otwarte środowiska programistyczny jak również komercyjne rozbudowane programy wspomagające proces programowania mikrokontrolerów. Głównie wykorzystuje się język C lub, w szczególnych przypadkach, język maszynowy (asembler). Istnieje również możliwość automatycznego generowania kodu dla mikrokontrolerów. Ma to szczególne znaczenie w przypadku złożonych algorytmów obliczeniowych, np. zaawansowanych algorytmów regulacji. Jako przykłady pakietów do automatycznej generacji kodu algorytmów regulacji predykcyjnej można wymienić: AutoMATiC [23] (algorytmy regulacji predykcyjnej DMC i GPC), ACADO [40] (algorytmy regulacji predykcyjnej bazujące na liniowych modelach w przestrzeni stanu), CVXGEN [79] (automatyczna generacja program w języku C dla zadań

optymalizacji kwadratowej), Multi-Parametric Toolbox [53] (algorytmy regulacji predykcyjnej typu explicit). Ciekawe zastosowania pakietu ACADO do generacji kodu algorytmu służącego do omijania przeszkód pojazdu przedstawiono w pracy [42]. Zastosowano nieliniowe algorytmy regulacji predykcyjnej, wymagające nieliniowej cyklicznej optymalizacji. Aby ograniczyć ilość obliczeń, koniecznych do wykonania w każdej chwili dyskretnej algorytmu regulacji, powodzeniem cieszy się tzw. algorytm explicit MPC [16]. Algorytm ten bazuje na modelu liniowym, w celu eliminacji konieczności optymalizacji uwzględnia się zbiór algorytmów analitycznych, pracujących lokalnie. Choć jego wadą może być dość duża liczba regulatorów lokalnych, w praktyce udaje się zaprojektować skutecznie i szybko działające algorytmy regulaji, złożone z ograniczonej liczby regulatorów lokalnych (kilkanaście lub kilkadziesiąt). [65]. W pracy [105] przedstawiono zastosowanie tego algorytmu do sterowania systemami aktywnej redukcji drgań w konstrukcjach mechanicznych.

Warto podkreślić fakt, że powszechnie stosowane mikrokontrolery posiadają wyłącznie jeden rdzeń obliczeniowy. Oznacza to, że przetwarzanie informacji jest możliwe tylko w sposób sekwencyjny. Dostępne mechanizmy przerwań i bezpośredniego zapisu do pamięci DMA zwiększają efektywność pracy mikrokontrolera, jednak nadal praca pdbywa się przy wykorzystaniu tylko jednego rdzenia obliczeniowego. Bardziej rozbudowane mikrokontrolery mogą posiadać więcej rdzeni obliczeniowych, np. rodzina mikrokontrolerów NXP MAC57D5xx, które posiadają w jednym układzie scalonym aż trzy rdzenie. Co ciekawe, każdy z tych rdzeni jest dedykowany pod konkretny segment pracy jak aplikacja właściwa użytkownika, rdzeń dedykowany dla branży motoryzacyjnej oraz rdzeń obsługujący wejścia i wyjścia.

W odniesieniu do założeń zdefiniowanych w podrozdziale 3.1, cechy mikrokontrolerów można podsumować następująco:

— czas obliczeń wykonywanych w pojedynczej chwili próbkowania: poniżej jednej milisekundy,
— dostępna pamięć: pojedyncze MB,
— rodzaje interfejsów wejść i wyjść: standard 3.3V,
— możliwości połączenia w sieć: rozbudowane,
— bezpieczeństwo pracy oraz danych: do realizacji we własnym zakresie,
— wymiary zewnętrzne: małe,
— niezawodność: stosunkowo duża,
— odporność na czynniki zewnętrze: mała,

— cena: niska,

— zasilanie bateryjne: dedykowane.

Podsumowując, należy podkreślić, że mikrokontrolery posiadają bardzo duże możliwości. Mogą być zastosowanie w prostych zastosowaniach hobbystycznych, zaawansowanych aplikacjach algorytmów regulacji predykcyjnej, kończąc na specjalizowanych układach czasu rzeczywistego. Wszyscy producenci mikrokontrolerów wyposażają swoje układu w liczne interfejsy komunikacyjne ,dzięki czemu nie ma problemów łączenia układów ze sobą i wymianę informacji. Szybkości przetwarzania informacji i możliwości wykorzystania jednostek zmiennoprzecinkowych pozwala na implementację zaawansowanych algorytmów obliczeniowych, w tym algorytmów regulacji zaawansowanej, z zachowaniem bardzo krótkich czasów obliczeń. Bardzo ważnym czynnikiem sprzyjającym popularności mikrokontrolerów jest niska cena i powszechna dostępność.

Rozdział 4

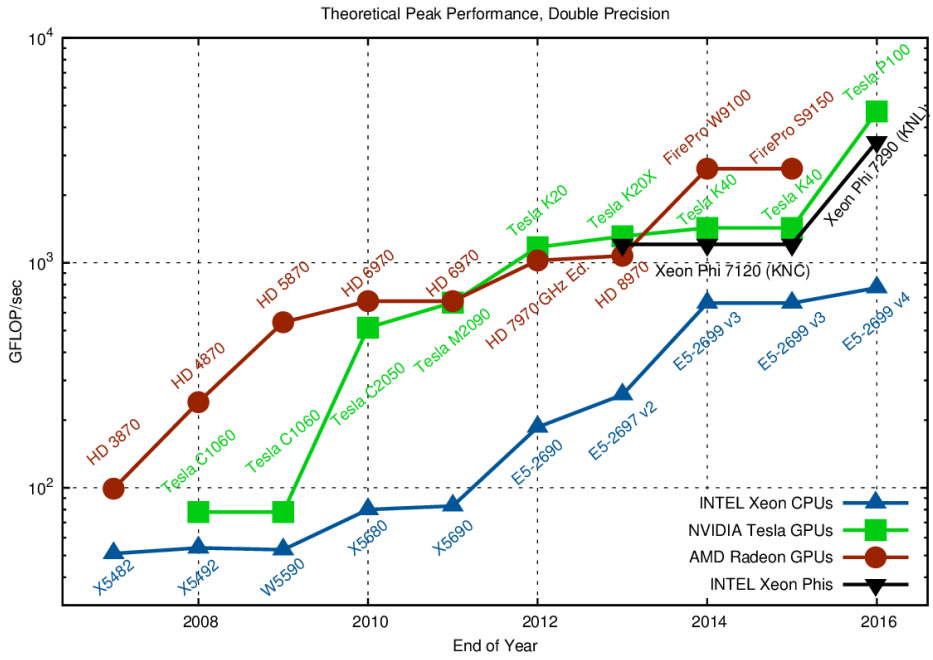
Układy FPGA

Możliwość wzrostu częstotliwości zegarów procesorów została ograniczona z powodów termicznych już w roku 2003. Prawo Moore'a głosi, że wzrost mocy obliczeniowej komputerów będzie się podwajał co 2 lata. Udało się utrzymać ten stan przez włączanie dodatkowych rdzeni procesorów. Jednak samo włączenie dodatkowego rdzenia nie spowoduje przyśpieszenia jeśli program będzie napisany sekwencyjnie zgodnie z podejściem von Neumanna z roku 1945. Wymagane jest nowe podejście do programowania w sposób równoległy. Można rozważyć dwa podejścia: pierwsze to procesory kilkurdzeniowe, przy czym każdy rdzeń zajmuje się kilkoma wątkami; drugie podejście to procesory wielordzeniowe (kilkaset do tysięcy) w czym każdy rdzeń zajmuje się pojedynczym wątkiem. Na rys. 4.1 przedstawiono wykres przyrostu mocy obliczeniowej procesorów w ostatnich latach. Na rok 2016 na prowadzeniu plasuje się procesor graficzny NVIDIA Tesla, który posiada drugą z wymienionych struktur rdzeni obliczeniowych. W bieżącym rozdziale omówiono jak można w tym wyścigu umieścić układy FPGA, które posiadają bardzo rozbudowane możliwości, konieczne w systemów szybko przetwarzających informacje.

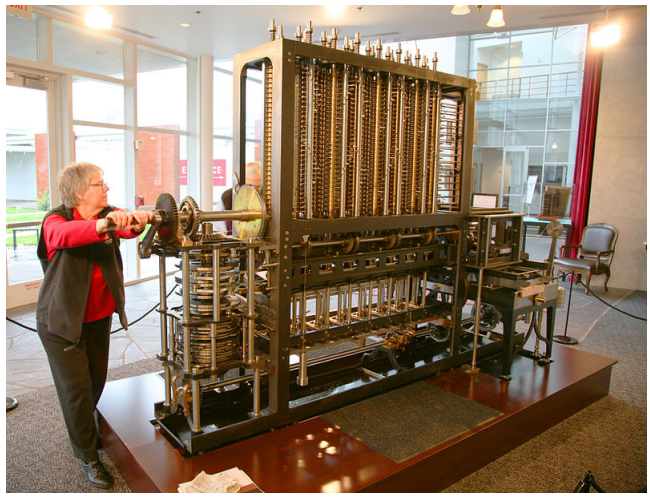
4.1. Geneza układów FPGA

W roku 1832 Charles Babbage opracowuje maszynę liczącą, która była rodzajem ulepszonego kalkulatora wykorzystującego metodę różnicową. Muzealny eksponat maszyny przedstawiono na rys. 4.2.

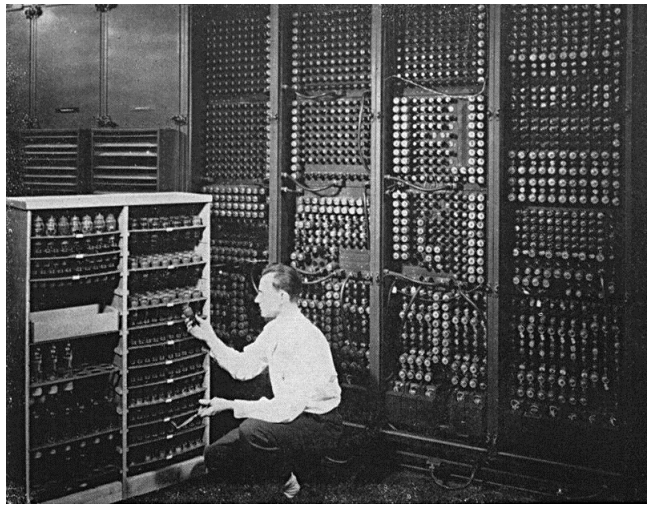
W roku 1850 powstaje algebra Boole'a stworzona przez George'a Boole'a. Autor przedstawił wyrażenia logiczne za pomocą symboli, co pozwoliło na operowanie wyrażeniami logicznymi w świecie matematyki. Zbiór 0, 1 to najprostsza algebra Boole'a, mająca tylko dwa elementy stanowiąca podstawowe narzędzie matematyczne do opisu układów cyfrowych. W roku 1925 wiadome stało się jak zrobić tranzy-



Rys. 4.1. Moc obliczeniowa procesorów, źródło:
https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/



Rys. 4.2. Maszyna różnicowa autorstwa Charles'a Babbage'a, źródło:
https://commons.wikimedia.org/w/index.php?curid=9084093

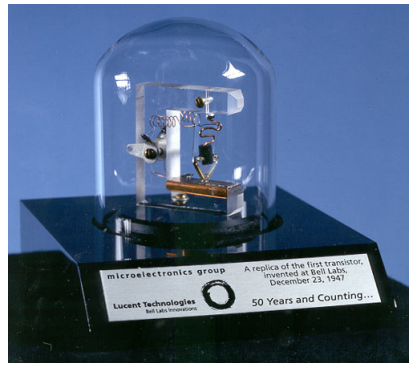


Rys. 4.3. Pierwszy komputer lampowy ENIAC, źródło: https://commons.wikimedia.org/w/index.php?curid=8813812

stor FET, jednak technologia wytwarzania układów scalonych była jeszcze w fazie raczkowania, a świat pracował nad technologią lamp elektronowych. Autorem pierwszego patentu tranzystora FET został Julius Edgar Lilenfeld. W roku 1938 Claude Shannon kończy swoją pracę magisterską w której połączył algebrę Boole'a z układami przełączającymi, co dużym echem odbiło się w najbliższej przyszłości. W roku 1945 lampy elektronowe zostały wykorzystane jako elementy przełączające do pierwszego komputera z pamięcią programu opracowanego przez John'a von Neumanna. W roku 1946 przedstawiono pierwszy komputer elektroniczny o nazwie ENIAC, zbudowany z 18000 lamp elektronowych. Fragment komputera wraz z inżynierem szukającym uszkodzonej lampy wśród tysięcy możliwych kombinacji przedstawiono na rys. 4.3. Komputer był w stanie przetwarzań kilka tysięcy mnożeń na minutę.

W roku 1947 następuje przełom technologiczny, kiedy Brittain oraz Bardeen opracowują pierwszy tranzystor ostrzowy. Replikę tranzystora przedstawiono na rys. 4.4.

W roku 1949 Schockley opracował tranzystor bipolarny, co pozwoliło na zastąpienie bardzo awaryjnych lamp elektronowych. Dodatkowo, możliwe stało się integrowanie elementów w jednej obudowie. W 1956 roku Harris opracował pierwszą bipolarną bramkę logiczną. Trzy lata później Jack Kilby z Texas Instruments zbudował pierwszy monolityczny układ scalony (ang. Integrated Circuit, IC). Skła-



Rys. 4.4. Pierwszy tranzystor ostrzowy Bell Laboratories, źródło: https://pl.wikipedia.org/wiki/Plik:Replica-of-first-transistor.jpg

dał się tylko z jednego tranzystora, trzech rezystorów i jednego kondensatora, ale było to tak przełomowe, że autor w 2000 roku otrzymał nagrodę Nobla. W roku 1960 firma Fairchild wprowadziła pierwszy komercyjny układ scalony w technologii TTL (ang. Transistor Transistor Logic), co otworzyło bardzo długą historię tych układów, aż do roku 1990. Równolegle pracowano nad technologią FET oraz MOS, które nie były dostatecznie doceniane. W 1960 opracowano pierwszą technologię CMOS, pojawiły się kalkulatory z tranzystorami pMOS, w 1970 zastosowano tranzystory nMOS do mikroprocesorów 4004 oraz 8080 firmy Intel. Od roku 1980 nastąpił wielki przełom technologii MOS i zaczęła ona dominować na całym rynku układów scalonych. W roku 1980 świętowano pierwszy milion tranzystorów w układzie scalonym, obecnie w jednej kości scalonej mieczczą się setki milionów tranzystorów w technologii MOS.

Zastosowanie nowych technologii doprowadziło do dynamicznego rozwoju świata cyfrowego. Wyzwania stawiane inżynierom stawały się coraz bardziej złożone. Początkowo używano układów scalonych średniej skali integracji (ang. Medium Scale Integrated, MSI), montowanych na płytach drukowanych z tysiącami połączeń między nimi. Bardzo popularna była seria układów scalonych TTL 74XXX, która później została zastąpiona przez rodzinę opartą o tranzystory MOSFET. Połączenie kilku elementów logicznych pozwalało na realizację bardziej skomplikowanego zadania logicznego. Druga technologia, która pozwoliła optymalizować zajmowaną przestrzeń to Full-Custom, gdzie należy samodzielnie zaprojektować strukturę połączeń układów logicznych, następnie przenieść to na rozmieszczenie i połączenia poszczególnych tranzystorów w krzemie, a następnie zlecić do produkcji. Ta me-

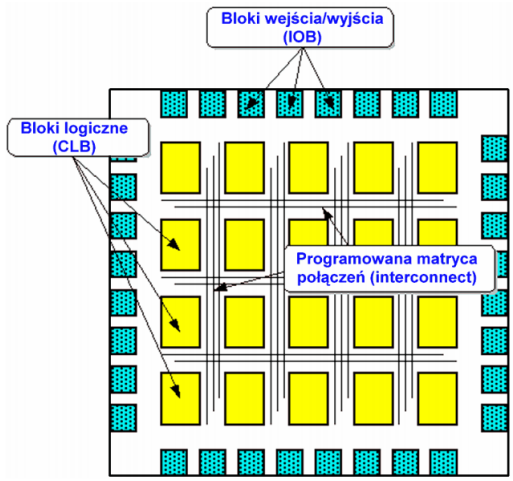
toda ma uzasadnienie w bardzo wymagających aplikacjach, gdzie należy uzyskać najbardziej wygórowane parametry. Istnieje jeszcze pośrednia technologia zwana Semi-Custom, gdzie zamiast rozmieszczania poszczególnych tranzystorów w strukturze scalonej można posłużyć się pewnymi blokami bibliotecznymi tworząc gotowy schemat rozmieszczenia i połączeń. Następnie producent układu scalonego realizuje projekt według schematu dostarczonego przez projektanta.

Przełomem w tej dziedzinie okazało się skonstruowanie reprogramowalnego układu scalonego. W roku 1984 firma Altera po raz pierwszy dostarczyła na rynek reprogramowalny logiczny układ scalony o symbolu EP300. Programowanie logiki realizowane było za pomocą pamięci EPROM. Obecnie na rynku jest kilka wiodących firm, które oferują układy FPGA: Intel (dawna Altera), Xilinx, Lattice, Microsemi. Główna zasada działania jest dokładnie taka sama u wszystkich producentów. Użycie języków opisu sprzętu VHDL lub Verilog pozwala na przenoszenie projektu między różnymi producentami, niestety użycie bloków funkcyjnych danego producenta nie pozwala w pełni przenosić kodu ponieważ trzeba dostosować te bloki do nowego producenta lub wręcz użyć jego dedykowanych funkcji. Różnice pojawiają się również w osprzęcie peryferyjnym i możliwościach sprzętowych (dodatkowe układy PLL czy chociażby dodatkowe rdzenie wbudowanych mikrokontrolerów).

4.2. Struktura układów FPGA

Zaawansowane układy scalone FPGA zawierają w swojej strukturze bardzo dużą liczbę programowalnych komórek logicznych, typowo od 3 tysięcy do nawet 9 milionów. Wyróżniającą cechą układów FPGA jest możliwość wielokrotnej konfiguracji bezpośrednio przez użytkownika. Idealnie nadają się do testowania koncepcji w fazach testów. Następnie, na potrzeby uruchomienia produkcji seryjnej, wybiera się układy typu ASIC (ang. Application Specific Integrated Circuit), które konfigurację mają zapisaną na stałe w swojej strukturze.

Trzy najważniejsze elementy układów FPGA są następujące:
— bloki logiczne,
— bloki wejść i wyjść,
— matryca połączeń wewnętrznych.
które również przedstawiono na rys. 4.5. Bloki logiczne są konfigurowalne przez użytkownika. Podobnie, bloki wejść i wyjść są w pełni konfigurowalne. Aby cały system mógł działać prawidłowo, potrzebne jest jeszcze skonfigurowanie matrycy



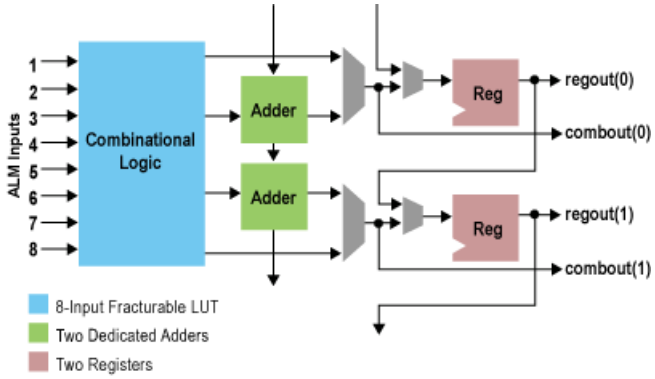
Rys. 4.5. Schematyczna struktura wewnętrzna układu programowalnego FPGA

połączeń wewnętrznych, gdzie można ustawić odpowiednie połączenia pomiędzy blokami logicznymi oraz blokami wejść i wyjść.

Każda komórka logiczna składa się z tablic typu LUT (ang. look up table), multiplekserów, logiki arytmetycznej oraz rejestrów. Bloki te umożliwiają optymalne odwzorowanie wymaganych przez użytkownika funkcji logicznych. Przykładowa struktura wewnętrzna bloku logicznego przedstawiona została na rys. 4.6. Należy podkreślić, że bloki logiczne pracują jednocześnie i mogą realizować niezależne operacje. Mogą one również wykonywać operacje matematyczne. Jeżeli połączymy obie te właściwości to układy programowalne pozwalają na realizacje różnych zadań w tym samym czasie w wielu jednostkach arytmetycznych. Ograniczeniem jest oczywiście liczba dostępnych układów logicznych, ale w przypadku obecnej technologii to ograniczenie jest mało znaczące.

Bloki wejść i wyjść pozwalają wykorzystać każdy pin jako wejście lub jako wyjście. Jeżeli wybrana jest opcję wejścia, istnieje możliwość wykorzystania rezystorów dociągających linię do napięcia zasilania lub masy. Wejścia są również zabezpieczone przed wyładowaniami ładunku elektrycznego. W konfiguracji wyjściowej można wykorzystać wbudowany przerzutnik D, bufor trójstanowy, możliwość pracy synchronicznej lub asynchronicznej.

Matryca połączeń wewnętrznych zapewnia optymalne połączenia między wewnętrznymi blokami dzięki trzem rodzajom połączeń. Dostępne są: połączenia



Rys. 4.6. Przykładowa struktura wewnętrzna bloku logicznego, źródło: https://www.intel.com/content/www/us/en/programmable/products/fpga/features/stxiv-alm-logic-structure.html

ogólnego przeznaczenia, linie długie, połączenia bezpośrednie. Połączenia ogólnego przeznaczenia wykorzystując ścieżki metalizacji pogrupowane w segmenty. Na skrzyżowaniach występują matryce przełączników, a co pewien dystans występują również wbudowane bufory do regencji sygnałów. Linie długie to dedykowane trasy, który omijają matryce przełączników i są dedykowane do łączenia odległych bloków. Uzyskuje się w ten sposób małe przesunięcie fazowe sygnału zegarowego, co ma szczególne znaczenie w pracy synchronicznej układów logicznych. Najczęściej te połączenia służą do dystrybucji sygnału zegarowego wewnątrz struktury układu logicznego. Połączenia bezpośrednie wykorzystywane są do łączenia sąsiadujących bloków, dzięki czemu uzyskuje się minimalne opóźnienia w propagacji sygnałów pomijając matryce przełączników.

Cała konfiguracja układu FPGA przechowywana jest w pamięci konfiguracyjnej. Najczęściej jest to pamięć statyczna RAM. Po włączeniu zasilania konfiguracja jest ładowana z pamięci FLASH i dopiero później wykorzystywana jest pamięć RAM.

Układy PLD (ang. Programmable Logic Device) są bardzo prostymi matrycami programowalnych układów logicznych, których koszty są bardzo niskie. Po drugiej stronie horyzontu możemy skorzystać z bardzo drogich rozwiązań typu ASIC (ang. Application Specyfic Integrated Circuit), gdzie użytkownik projektuje strukturę całego systemu w krzemie. Z uwagi na powyższe można stwierdzić, że technologia FPGA plasuje się między PLD i ASIC.

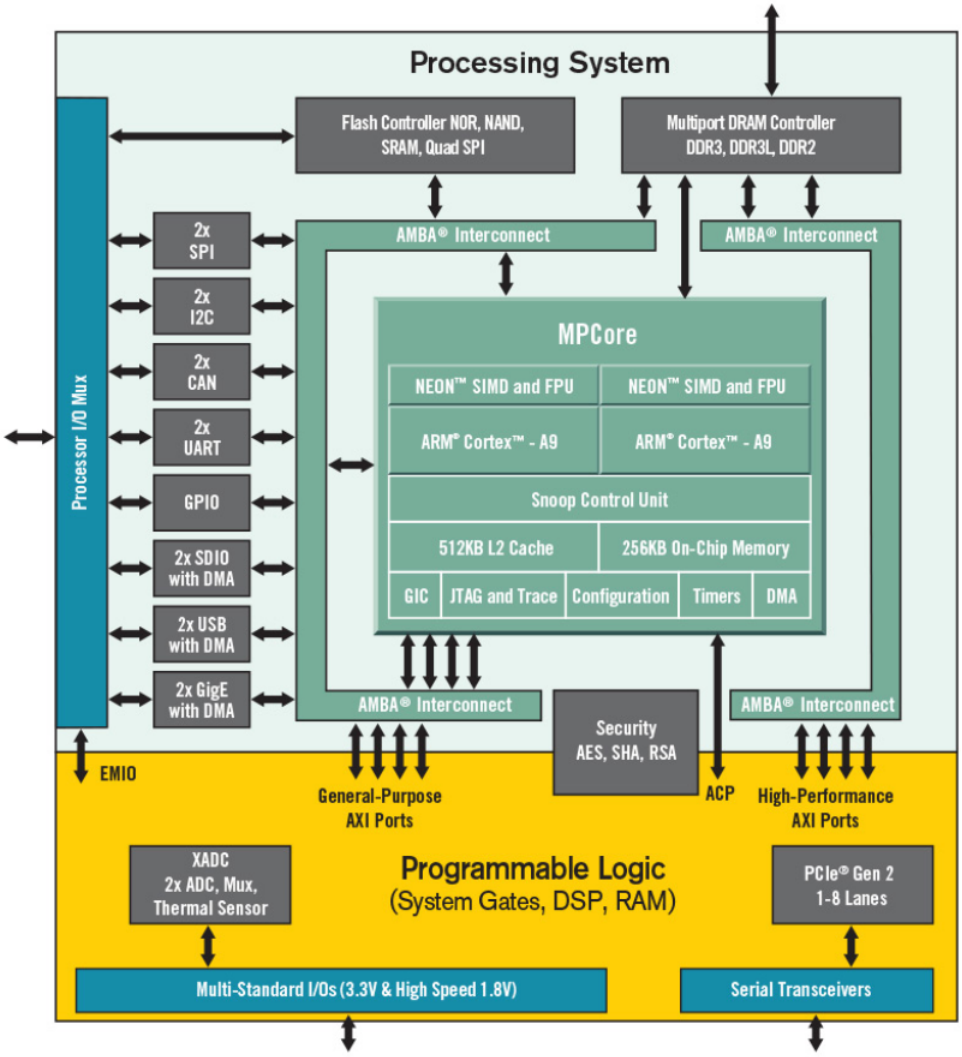
Bardzo ważną cechą układów FPGA jest możliwość ich pełnej konfiguracji już po produkcji i zastosowaniu w projektowanym systemie. Koszty związane z zaku-

pem takiego rozwiązania są również niższe w stosunku do układów ASIC a, czas implementacji projektu jest krótszy. Nieocenioną zaletą jest możliwość wprowadzania poprawek, dzięki czemu rozwiązanie jest idealne do celów prototypowania. Dzięki wsparciu producentów układów FPGA można korzystać z bardzo wielu bibliotek i funkcji specjalnych, które pozwalają na realizacje równie zaawansowanych operacji jak w przypadku układów ASIC. Jeżeli chodzi o obliczenia arytmetyczne, to bardzo dobrze opracowane są funkcje dla liczb stałoprzecinkowych. Nieco gorzej sytuacja wygląda dla operacji zmiennoprzecinkowych, aczkolwiek jak wspomniano producenci udostępniają pakiet bibliotek, gdzie można między innymi znaleźć funkcję do realizacji obliczeń zmiennoprzecinkowych. Czas rekonfiguracji układu FPGA jest rzędu kilkudziesięciu milisekund. Niestety, z uwagi na uniwersalność należy liczyć się z większymi opóźnieniami wynikającymi z matrycy połączeń wewnętrznych. Bardzo istotne znaczenie mają tutaj algorytmy syntezy logicznej oraz rozmieszczenia logiki w strukturze wewnętrznej układu.

Na rynku występują układy hybrydowe, posiadające zamknięte w jednej kości układ FPGA i mikroprocesor ogólnego zastosowania. Pozwala to na bardzo korzystne rozwiązanie kwestii przesyłania danych, które może być wykonywane przez procesor. Z kolei obliczenia są już wykorzystywane w układach cyfrowych matrycy FPGA. Połączenie między dwoma układami wewnątrz jednej kości są realizowane dedykowanymi magistralami danych. Takie rozwiązanie przedstawiono na rys. 4.7

4.3. Programowanie układów FPGA

Programowanie układów logicznych rozpoczęło się od metody równań wykorzystujących algebrę Boole'a, na bazie których można było realizować schematy z bramkami logicznymi. Kolejnym etapem było wprowadzenie systemów synchronicznych i zastosowanie elementów pamięci typu przerzutnik, gdzie system był odwzorowany w strukturze logicznej dokładnie tak jak go opisano. Następnym krokiem rozwoju było stworzenie języków VHDL i Verilog do pełnego opisu zachowania układu na bazie, którego kompilatory tworzyły odpowiednie sieci połączeń, strukturę RTL i ostateczne schematy połączeń między bramkami logicznymi w układzie programowalnym. Jeszcze wyższą warstwą abstrakcji są tak zwane biblioteki Intellectual Property, które są oferowane przez producentów układów logicznych w celu szybszego opracowania aplikacji przez inżynierów. Dzięki tym blokom można skrócić czas projektu przez użycie gotowych funkcji, na przykład jednostki obliczeń



Rys. 4.7. System hybrydowy: układ FPGA oraz mikroprocesor, źródło: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

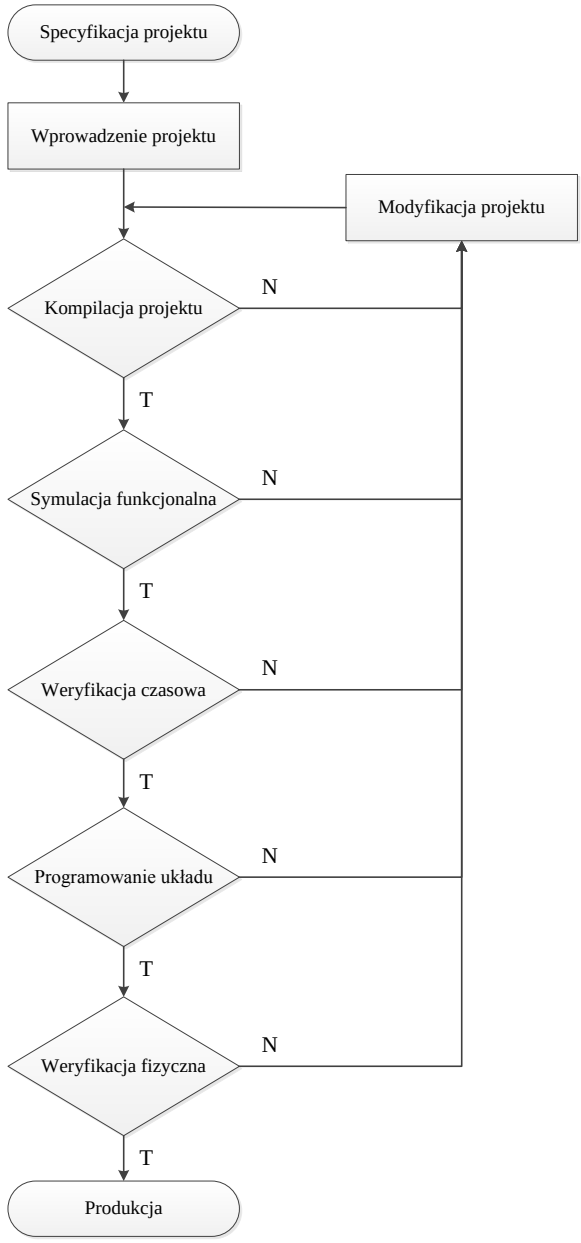
zmiennoprzecinkowych czy filtrów cyfrowych. Ponad tymi wszystkimi metodami stoi nadrzędne narzędzie System C, gdzie możliwe jest bezpośrednie implementowanie w języku C. Metoda ta pozwala na projektowanie na poziomie całego systemu. Jednak jej użycie nie zostało dobrze przyjęte i nie widać specjalnego rozwoju.

Proces projektowania systemu w oprogramowaniu narzędziowym producenta przedstawiono na rys. 4.8. Najważniejszy etap to oczywiście implementacja pro-

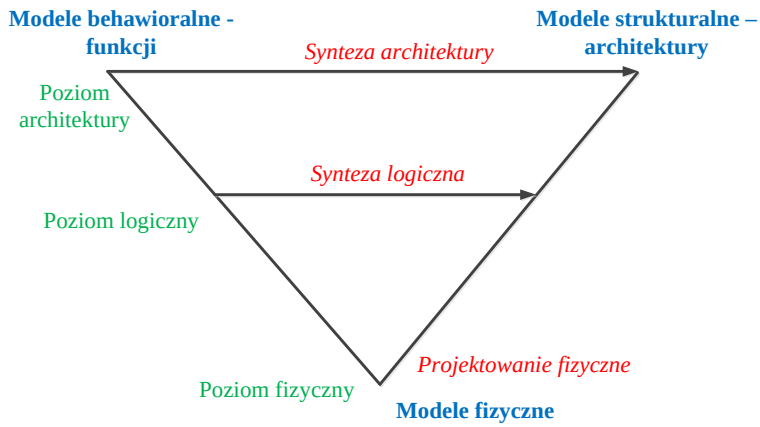
jektu w wybranym języku opisu sprzętu. Następnie, tak jak w klasycznym programowaniu, występuje etap kompilacji, gdzie sprawdzana jest składnia języka. Symulacja funkcjonalna może zostać porównana do typowego etapu wykrywania usterek działania układu (debugowania). W odróżnieniu od typowego debugowania w symulacji funkcjonalnej analizuje się przebiegi czasowe sygnałów, co pozwala na sprawdzenie czy odpowiednie sygnały pojawiają w odpowiednich chwilach czasowych. Weryfikacja czasowa jest poniekąd w pewien sposób powiązana z symulacją funkcjonalną, ale dodatkowo zostały uwzględnione wszystkie opóźnienia czasowe wynikające z połączeń między bramkami logicznymi. Dopiero po tej weryfikacji można mieć pewność, że sygnały pojawiają się w odpowiednich momentach. Ponadto, weryfikacja czasowa wyznacza ścieżkę krytyczną, która jest najdłuższym czasem potrzebnym do realizacji zadanej funkcji. Odwrotność tego czasu pozwala na określenie maksymalnej częstotliwości taktowania układu logicznego. Kolejny etap to programowanie układu logicznego. Konfiguracja układu FPGA jest wgrywana do zewnętrznego układu programowalnego. Po restarcie konfiguracja zostaje automatycznie wgrana do FPGA i układ jest w pełni gotowy do działania. Weryfikacja fizyczna jest ostatnim etapem, który finalnie potwierdza, że dana aplikacja może pracować prawidłowo. Szczególne znaczenie ma tutaj weryfikacja pracy układów podrzędnych takich jak zewnętrzna pamięć czy dodatkowe warstwy fizyczne portów interfejsowych.

W modelu projektowania układu logicznego można wyróżnić trzy poziomy abstrakcji. Przedstawiono to schematycznie na rys. 4.9. Poziom architektury skupia się głównie na specyficznych operacjach. Możnan do nich zaliczyć transfer danych, obliczenia, akumulację danych. Na tym poziomie projektant posługuje się językami opisu sprzętu, a także schematami blokowymi. Drugi poziom to poziom logiczny, który charakteryzuje się użyciem zestawu funkcji logicznych opartych na grafach stanów, tablicach prawdy, schematach logicznych. Ostatni poziom to najniższy poziom określany jako geometryczny, gdzie projektant decyduje samodzielnie o rozłożeniu elementów i finalnej topografii układu. Ta opcja jest naturalnie zarezerwowana dla układów ściśle dedykowanych np. typu ASIC.

Następnym etapem projektowania jest proces syntezy. Na poziomie architektury chodzi głównie o organizację ścieżki danych i logiki sterującej, operatory przekładają się na funkcje układu (zasoby), wprowadzane są odpowiednie powiązania oraz kolejność i czas wykonywania. Synteza logiczna to opracowanie mikroskopowej struktury układu, w skład której wchodzi automat sterujący, schemat logiczny we-



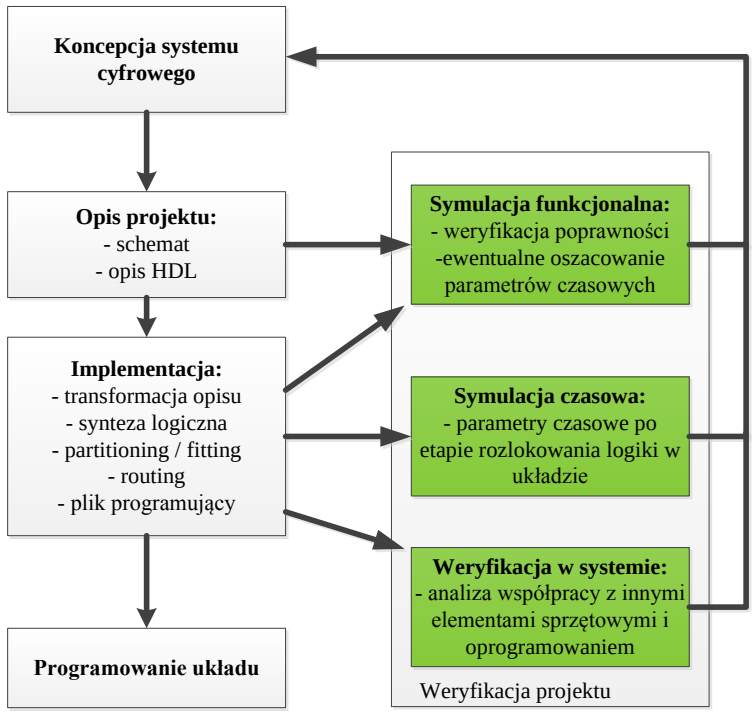
Rys. 4.8. Proces projektowania i weryfikacji w układach FPGA



Rys. 4.9. Modele i zadania syntezy w układach FPGA: poziomy abstrakcji

dług opisu w języku HDL. Projektowanie fizyczne to opracowanie topografii układu scalonego oraz jednoczesna synteza i optymalizacja geometrii układu, następnie generowanie komórek i ostatecznie rozmieszczanie elementów i połączeń. Kryteria optymalizacji składają są następujące: kryteria ogólne, takie jak zajmowana powierzchnia, kryteria szczegółowe, takie jak szybkość działania (czas propagacji, cyklu, zwłoki), szybkość przetwarzania danych (przepustowość).

Układy FPGA posiadają określoną ścieżkę etapów projektowania aplikacji. Niektóre etapy są realizowane przez użytkownika, a inne przez wykorzystywane narzędzie programowania. Pełny schemat działania przedstawiono na rys. 4.10. Wszystko zaczyna się od pewnej koncepcji układu w znaczeniu ogólnym, określonej przez zarys wymaganych funkcji i możliwości bez podania szczegółów realizacji. Kiedy już etap koncepcji zostaje zaakceptowany, należy wykonać opis realizowanego algorytmu np. w formie schematu graficznego. Schemat graficzny pozwala zrozumieć działanie układu użytkownikowi czy projektantowi, jednak narzędzie programistyczne musi otrzymać plik tekstowy zawierający zapis algorytmu w jednym z języków opisu sprzętu HDL. Tutaj projektanci dysponują co najmniej trzema metodami realizacji zadania. Pierwsze podejście to modelowanie układu zarówno na poziomie strukturalnym (sprzętowym) jak i behawioralnym (funkcjonalnym). Druga metoda opiera się o wykorzystanie tak zwanych funkcji złożonych, które są dostępne w narzędziu programowania. Bardzo często są to odzwierciedlone układy cyfrowe np. serii 74XXX. Trzecia metoda pozwala użyć uniwersalnych i sparametryzowanych



Rys. 4.10. Pełna ścieżka projektowania aplikacji dla FPGA

bloków konstrukcyjnych służących do tworzenia zaawansowanych układów cyfrowych. Bloki te określane są jako IP (ang. Intelctual Property).

Bardzo ważny etap implementacji projektu obejmuje kilka punktów, które opisano poniżej:

— translacja języka opisu sprzętu (HDL) na opis niskiego poziomu (równania boolowskie), co jest wymagane do użycia narzędzi optymalizujących liczbę układów logicznych,

— synteza logiczna realizuje optymalizację projektu pod względem zajętości bramek, szybkości działania, pobieranej mocy,

— odwzorowanie technologiczne przekłada optymalny projekt na bramki logiczne dostępne w FPGA z uwzględnieniem budowy pojedynczego bloku logicznego,

— podział (ang. partitioning) wykonywane jest w przypadku wymaganego podzielenia projektu na części,

— rozmieszczenie (ang. fitting) polega na optymalnym rozmieszczeniu podstawowych bloków logicznych z punktu widzenia odległości i dostępnych zasobów,
— generacja pliku programującego.

Na poszczególnych etapach projektu wykonuje się weryfikację. Przed etapem podziału i rozmieszczenia wykonywane są testy funkcjonalne działania układu logicznego, czyli sprawdzenie czy logika działania jest prawidłowa. Kiedy wynik jest prawidłowy wykonuje się testy czasowe, który już uwzględniają podział i rozmieszczenie w fizycznej strukturze układu FPGA. Uzupełnia się informacje o dane na temat zależności czasowych, co umożliwia dokładne przeanalizowanie projektu pod względem niepożądanych efektów związanych z opóźnieniami wprowadzanymi przez rzeczywiste układy cyfrowe. Ostatni i najważniejszy test to weryfikacja w rzeczywistym systemie, czyli sprawdzenie działania układu w warunkach w jakich rzeczywiście będzie pracował.

Etap programowania fizycznego układu FPGA jest realizowany w momencie, kiedy projektant posiada plik programujący, dzięki niemu układ jest konfigurowany i natychmiast gotowy do pracy. Większość technologii programowalnych wymaga odpowiedniego urządzenia programującego. Jednak układy programowalne w systemach nie muszą wykorzystywać fizycznego urządzenia programującego, a wymagają jedynie pewnych inteligentnych mechanizmów umożliwiający załadowanie konfiguracji do układu bez konieczności wyjmowania go z systemu, a nawet bez konieczności wyłączania systemu. Tego typu konfiguracja możliwa jest przy wykorzystaniu mikroprocesora lub złącza JTAG.

Znając już precyzyjną procedurę programowania układów FPGA należy zwrócić uwagę na języki opisu sprzętu HDL i pokrótce opisać ich możliwości. Można wymienić najważniejsze cechy każdego języka HDL:
— semantyka zawierająca koncepcję jednostki projektowej, połączenia, współbieżności i zależności czasowych,
— możliwość opisu opóźnień w propagacji sygnałów i zależności czasowych,
— konstrukcje umożliwiające opis realizacji strukturalnej,
— udostępnianie instrukcji umożliwiających opis funkcjonalny układu w sposób znany z tradycyjnych języków programowania,
— możliwość efektywnego opisu operacji i struktur danych na poziomie bramek i przesłań między rejestrowych,
— dostępność konstrukcji umożliwiających projektowania w sposób hierarchiczny.

Pierwszy opisywany język HDL to VHDL (ang. Very high speed integrated

Hardware Description Language), który został przyjęty jako przemysłowy standard języka HDL (IEEE Std. 1076). Był on modyfikowany w latach 1987, 1993 i 2001, kiedy powstawały standardy VHDL-87, -93, -2001. Wzbogacano również język o dodatkowe pakiety takie jak:

— IEEE Std 1076.2-1996, VHDL Mathematical Packages: definiuje dodatkowe funkcje matematyczne dla liczb rzeczywistych i zespolonych,
— IEEE Std 1076.3-1997, Synthesis Packages: definiuje arytmetyczne operacje na zestawach bitów,
— IEEE Std 1076.6-1999, VHDL Register Transfer Level (RTL) Synthesis: definiuje podzbiór języka odpowiedni dla procesów syntezy,
— IEEE Std 1164-1993 Multivalue Logic System for VHDL Modeling Interoperability: definiuje nowe typy danych dla celów modelowania logiki wielowartościowej,
— IEEE Std 1029.1-1998 VHDL Waveform and Vector Exchange to Support Design and Test Verification (WAVES): definiuje sposób wykorzystania języka VHDL do wymiany danych w środowisku symulacyjnym.

Bardzo ważną cechą języka VHDL jest jego przenośność, dzięki czemu może być stosowany przez systemy różnych firm produkujących układy FPGA. Ponadto, zawiera użyteczne konstrukcje semantyczne umożliwiające zwięzłą specyfikację złożonych układów cyfrowych. Projekt może być opisany hierarchicznie (na wielu poziomach) oraz możliwe jest korzystanie z biblioteki gotowych elementów i tworzenie podukładów (tzw. komponentów). Jest również powszechnie stosowany w specyfikacjach układów cyfrowego przetwarzania sygnałów [80, 83].

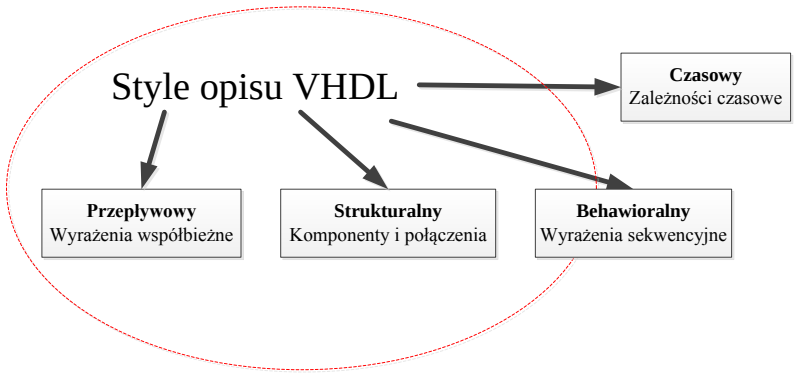
Generalna składnia języka VHDL składa się z trzech sekcji:

— deklaracje bibliotek (LIBRARY),
— deklaracja jednostki (ENTITY),
— opis architektury (ARCHITECTURE).

Na przykładzie bramki logicznej NAND przedstawiono trzy najważniejsze sekcje w listingu poniżej:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bramka_nand IS
PORT(
a : IN STD_LOGIC;
b : IN STD_LOGIC;
z : OUT STD_LOGIC
);
END bramka_nand;
```

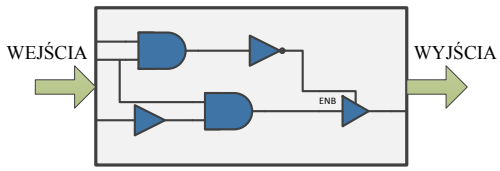


Rys. 4.11. Style opisu projektu w języku VHDL

```
ARCHITECTURE model OF bramka_nand IS
BEGIN
z <= a NAND b;
END model;
```

W języku VHDL można określić różne style opisu działania docelowego projektu. Przedstawiono każdy styl wraz z krótkim przykładem użycia. Rys. 4.11 pokazuje style opisów razem z uwzględnieniem czy dany opis może być zrealizowany w rzeczywistym układzie. Opis czasowy polega na wprowadzaniu sztucznych opóźnień w sygnałach wyrażanych w jednostkach czasu, przy czym nie może być odzwierciedlony w rzeczywistym układzie ponieważ opóźnienia czasowe sygnału wynikają bezpośrednio z rozmieszczenia bramek w strukturze FPGA. Styl opisu behawioralnego jest na pograniczu możliwości syntezy w rzeczywistym układzie, ponieważ czasami programista nieświadomie zrealizuje operacje, które mogą się skompilować ale nie mogą być zrealizowane w postaci układu logicznego.

Styl przepływowy (współbieżny, funkcjonalny) charakteryzuje się tym, że operacje wykonywane na danych są współbieżne, dokładnie w tym samym momencie. Jest to w zasadzie najlepszy sposób kodowania równań boolowskich. Bardzo ważne jest, że zapis będzie bezpośrednio odzwierciedlony w odpowiednich układach logicznych i połączeniach między nimi. Ponadto, kolejność zapisanych instrukcji nie ma w tym stylu żadnego znaczenia. Wykorzystuje się układy cyfrowe, takie jak najprostsza bramka typu suma logiczna czy iloczyn logiczny. Przykład takiego programowania przedstawiono na rys. 4.12 oraz na listingu poniżej:



Rys. 4.12. Przykład opisu współbieżnego projektu w języku VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bramka_xor3 IS
PORT(
A : IN STD_LOGIC;
B : IN STD_LOGIC;
C : IN STD_LOGIC;
Z : OUT STD_LOGIC
);
end bramka_xor3;

ARCHITECTURE dataflow OF bramka_xor3 IS
SIGNAL x: STD_LOGIC;
BEGIN
x <= A XOR B;
Z <= x XOR C;
END dataflow;
```

Styl opisu strukturalny jest bardzo podobny do opisu przepływowego z tą zasadniczą różnicą, że obejmuje on większe formy układów a nie tylko pojedyncze bramki logiczne. Jest to metoda, która pozwala odwzorować schemat logiczny. Można łączyć mniejsze elementy do tworzenia bardziej złożonych funkcji logicznych, następnie wykorzystać funkcje do tworzenia hierarchicznych struktur projektu. Jest to naturalny opis projektu złożonego z mniejszych bloków, czego przykładem może być system mikroprocesorowy, w skład którego muszą wejść bloki typu: procesor, pamięć, koder, dekoder, blok obliczeniowy. Przykład listingu poniżej przedstawia projekt pojedynczej bramki XOR, jest to typowe podejście realizacji stylu współbieżnego:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY xor2 IS
PORT(
I1 : IN STD_LOGIC;
I2 : IN STD_LOGIC;
Y : OUT STD_LOGIC
);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
Y <= I1 xor I2;
END dataflow;
```

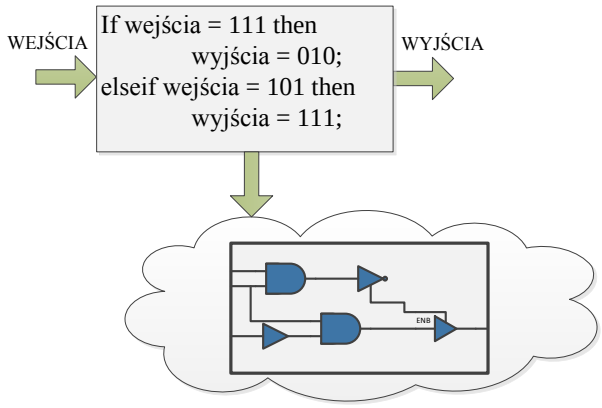
Kolejny listing powołuje komponent już zaprojektowany wykorzystuje go dwukrotnie do realizacji większego zadania, a mianowicie bramki XOR z trzema wejściami (podejście strukturalne):

```
ARCHITECTURE struct1 OF bramka_xor3 IS
COMPONENT xor2 IS PORT(
I1 : IN STD_LOGIC;
I2 : IN STD_LOGIC;
Y : OUT STD_LOGIC );
END COMPONENT;

SIGNAL t: STD_LOGIC;

BEGIN
U1: xor2 PORT MAP (
I1 => A,
I2 => B,
Y => t);
U2: xor2 PORT MAP (
I1 => t,
I2 => C,
Y => Z);
END struct1;
```

Pierwszym najbardziej intuicyjnym podejściem jest metoda języka funkcjonalnego – styl behawioralny. Dzięki takiemu podejściu można w prosty sposób opisać, co dany algorytm ma wykonywać w kolejnych krokach. Jak będzie to szczegółowo zrealizowane w blokach sprzętowych, zależy od kompilatora, który zapewnia odpowiednią strukturę sprzętową. Można przyjąć, że takie rozwiązanie jest typu „czarna skrzynka". Inaczej mówiąc, jest to pewnego rodzaju język wysokiego poziomu dla układów FPGA. Sam opis jest abstrakcyjny: pomija bramki i elementy logiczne. Może opisywać elementy sekwencyjne takie jak: rejestry, liczniki, automaty stanu. Istnieje ryzyko, że dany opis wykracza poza możliwości realizacji sprzętowej. Rozwiązanie tego typu schematycznie przedstawia rys. 4.13.



Rys. 4.13. Przykład programowania funkcjonalnego

Styl behawioralny opiera się o mechanizm tak zwanego procesu. Proces można określić mianem zdarzenia, które się wykona w momencie, gdy sygnał umieszczony na liście czułości ulegnie zmianie. W ciele mechanizmu instrukcje wykonują się jedna po drugiej, czyli sekwencyjnie. W procesie mogą być definiowane zmienne lokalne. Lista czułości wskazuje sygnały, których zmiana powoduje uruchomienie procesu. Szczególnie istotna w procesach asynchronicznych: należy podać wszystkie sygnały wejściowe; w procesach synchronicznych: sygnały zegara CLK oraz sygnał RESET. Ograniczenia opisu procesów jakie mogą spotkać projektanta to w pierwszej kolejności możliwość opisywania dowolnego zachowania układu. Język VHDL pozwala na opis sytuacji niemożliwych do fizycznej realizacji (np. dwa sygnały zegarowe, licznik na dwóch zboczach zegara). Może to mieć niepożądany skutek, a mianowicie nie każdy proces jest syntezowalny. W opisie należy uwzględniać ograniczenia fizyczne. Przykładowy opis behawioralny przedstawiono na poniższym listingu:

```
ARCHITECTURE behavioral OF bramka_xor3 IS
BEGIN

opis_xor3: PROCESS (A, B, C)
BEGIN
IF ((A XOR B XOR C) = '1') THEN
Z <= '1';
ELSE
Z <= '0';
```

```
END IF;
END PROCESS opis_xor3;
END behavioral;
```

Przykładowy fragment kodu w języku VERILOG przedstawiono na listingu poniżej:

```
module dff_sync_reset (
data    , // Data Input
clk     , // Clock Input
reset   , // Reset input
q         // Q output
);

//-----------Input Ports---------------
input data, clk, reset ;

//-----------Output Ports---------------
output q;

//------------Internal Variables--------
reg q;

//-------------Code Starts Here---------
always @ ( posedge clk)
if (~reset) begin
        q <= 1'b0;
end
else begin
        q <= data;
end
endmodule
```

Jednym z elementów projektowania systemu jest testowanie działania układu logicznego. W ramach symulacji stosuje się tak zwane pojęcie Testbench. Jest to program napisany np. w języku VHDL, który testuje program właściwy. W przypadku wystąpienia błędu system podaje informację zwrotną w jakim momencie wystąpił błąd. Symulację można porównać do badania fizycznego układu, którego wejścia podłącza się do stymulatora (generatora testów) i obserwuje wyjścia. Symulacja kodu VHDL to wykonywanie wirtualnego eksperymentu, w którym fizyczny układ zastąpiono opisem VHDL. Dodatkowo, można wykorzystać język VHDL do opisu generatora testów i modułu kolekcjonującego odpowiedzi układu i porównującego je ze wzorcem.

Bardzo ważnym aspektem układów programowalnych FPGA jest fakt możliwości realizacji struktury zwanej System-on-chip. Jest to filozofia połączenia sprzętowych bloków logicznych z programem sekwencyjnym realizowanym w mikroprocesorze, który jest częścią sprzętu danego systemu. Inaczej mówiąc jest to połączenia

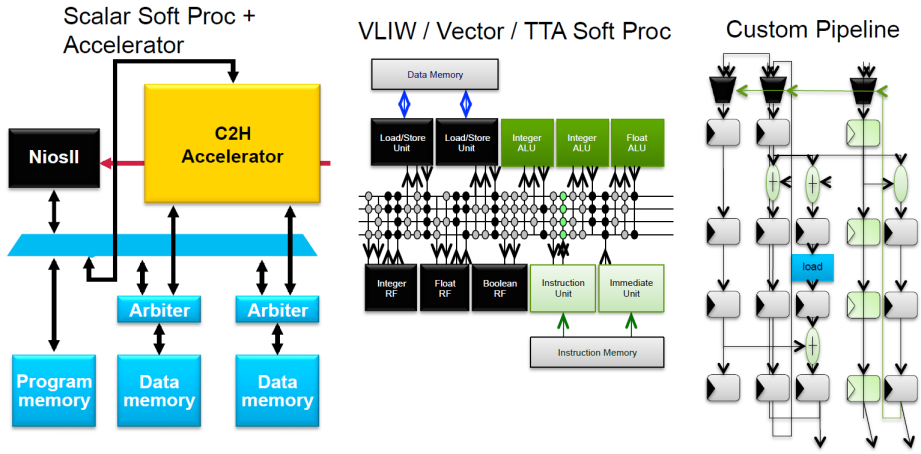
części sprzętowej i programowej w jednym układzie scalonym z możliwością dowolnej konfiguracji. Z uwagi na coraz większe pojemności układów logicznych taki system może zawierać procesory, koprocesory (FPU), pamięci programów, pamięci danych, sprzętowe akceleratory (FFT, AES), układy peryferyjne i komunikacyjne oraz inne specjalizowane układy logiczne.

Rozważając powyższe podejście do systemu na przykładzie producenta Intel (dawna Altera), można wykorzystać procesor NIOS II. Dostępne są dwie wersje procesora: NIOS II Fast oraz NIOS II Economy. Różnica w głównej mierze polega na wydajności operacji. Przykładowo, dla układu FPGA Cyclone IV GX procesor w wersji Economy posiada wydajność obliczeniową 30 DMIPS przy częstotliwości taktowania 175 MHz, a wersja Fast osiągnęła 190 DMIPS przy tych samych parametrach. Należy podkreślić, że możliwa jest implementacja kilku procesorów programowych w jednym układzie FPGA. Dzięki temu zyskuje się bardzo dużą elastyczność budowania całego systemu i możliwe jest precyzyjne połączenie z dodatkowymi układami takimi jak pamięć, peryferia, dedykowane bloki wspomagające. Pozwala to również na oszczędność objętości płytki drukowanej systemu, a także obniża koszty dodatkowych elementów, które należałoby stosować.

Architektura procesora NIOS II to RISC (ang. Reduced Instruction Set Computing). Dostępne są rejestry 32 bitowe i zmiennoprzecinkowa jednostka obliczeniowa. Dodatkowo można wykorzystać układy zegarowe, porty szeregowe, porty równoległe, a także kontrolery pamięci SRAM. Do programowania wbudowanego procesora można wykorzystać interfejs JTAG z szerokimi możliwościami śledzenia błędów programu wykonawczego. Do lepszej integracji z częścią sprzętową użytkownik może wykonać własne funkcje w celu przyspieszenia obliczeń. Procesory NIOS II są wspierane przez wielu producentów systemów operacyjnych czasu rzeczywistego.

Oprócz bezpośredniego programowania układów logicznych wykorzystując opisane wyżej metody, można zastosować pewien język wyższego poziomu, który pozwoli zaimplementować dany system w strukturze FPGA. Takie mechanizmy można znaleźć chociażby w środowisku MATLAB, jednak w tym przypadku nie ma mowy o projekcie całego systemu, a jedynie o pewnych nieskomplikowanych fragmentach. Ciekawą alternatywą jest otwarty standard oprogramowania OpenCL (ang. Open Computing Language) wykonywanego na platformach heterogenicznych: CPU, GPU, DSP. Zdefiniowany jest język bazujący na C do kodowania funkcji jądra wykonywanych na odpowiednich urządzeniach. Dodatkowo dostarczone jest API do definiowania i kontrolowania platform obliczeniowych. Istotne

jest wsparcie obliczeń równoległych. Jako przykład można podać układ FPGA jako wielordzeniowy procesor, gdzie każdy z rdzeni jest odpowiedzialny za swoje specyficzne operacje. Możemy wyróżnić podstawowy procesor skalarny (CPU) do podstawowych operacji, dedykowany procesor wektorowy (VLIW) do potokowego przetwarzania danych. Każdy z procesorów ma własną pamięć lokalną. Tworzy to cały system skalowalny na potrzeby aplikacji. Reprezentację graficzną rozważanego systemu przedstawiono na rys. 4.14.



Rys. 4.14. OpenCL: przykład systemu heterogenicznego z wykorzystaniem FPGA, źródło: https://www.intel.com/literature/wp-01173-opencl

Zaletą realizacji obliczeń w FPGA przy użyciu OpenCL to brak potrzeby znajomości zaawansowanej budowy układu FPGA, bardzo szybka implementacja (programowanie w języku C), bazowanie na sprawdzonych bibliotekach i blokach obliczeniowych, błyskawiczny rozwój i weryfikacja projektu, potok przetwarzania danych zajmuje minimalne zasoby. W porównaniu z procesorem programowym, optymalne wykorzystanie logiki FPGA oraz znacznie mniejszy pobór energii, niż w przypadku GPU.

Reasumując, jesteśmy świadkami przełomu w dziedzinie obliczeń realizowanych w układach FPGA, które przeszły długą drogę od "logiki klejącej" do zaawansowanych platform obliczeń równoległych. Możliwe obecnie są implementacje wielordzeniowych systemów System on Chip (SoC), gdzie stosuje się podejście projektowania systemowego, niemożliwe jest zajmowanie się każdą pojedynczą bramką logiczną. Algorytmy, które niegdyś były obecnie jedynie w teorii, obecnie są w podstawo-

wych bibliotekach wielu producentów. Układy programowalne oferują tysiące bramek logicznych, dostępne są wbudowane układy mnożące DSP, bloki arytmetyki zmiennoprzecinkowej, a także możliwości opisu w języku niezależnym od technologii sprzętowej (OpenCL).

4.4. Typowe zastosowania układów FPGA

Zastosowanie układów FPGA jest bardzo szerokie. Ogólne możliwości obliczeniowe przedstawiono w artykule [113]. Na bazie licznych przykładów można stwierdzić, że układy FPGA pozwalają uzyskać dużo lepsze parametry niż bardzo szybkie klasyczne procesory. W wielu przypadkach można uzyskać ponad 100-krotne przyśpieszenie algorytmu dzięki wykorzystaniu dekompozycji procesu i zrównolegleniu obliczeń [30, 36]. Wykorzystanie FPGA w mnożeniu macierzy w notacji całkowitoliczbowej przedstawiono w pracy [98], gdzie optymalne wykorzystanie układów mnożących pozwoliło na zmniejszenie wykorzystania zasobów układów logicznych oraz zwiększenie szybkości obliczeń. W dziedzinie generatorów liczb pseudolosowych można zwrócić uwagę na rozprawę doktorską [58], w której dokładnie opisano możliwości stosowalności układów FPGA. Dziedzina kryptografii jest chyba najchętniej poruszaną tematyką jeżeli chodzi o implementację w strukturach FPGA i możliwości przyśpieszenia procesów szyfrowania [48]. Również zastosowanie w pojazdach kosmicznych znajduje wiele możliwości z punktu widzenia obliczeń i ich szybkości przy wykorzystaniu niskich częstotliwości zegara [39, 49, 112]. Rozwiązania dla przemysłu wojskowego [5]. Zastosowanie FPGA w automatyzacji procesów przemysłowych jest powszechne. Przykładowe zastosowania przedstawiaja artykuł opisujący sterowanie CNC [82] lub praca [89], w której opisano autorski sterownik PLC zrealizowany przy wykorzystaniu układu FPGA. Sterowanie silnikiem DC algorytmami regulacji PID oraz IMC, zrealizowanymi sprzętowo w układzie FPGA, przedstawiono w pracy [26]. Przykładowe użycie do zastosowań w robotach przemysłowych można znaleźć w pracy [84]. Istnieją zastosowania FPGA w medycynie, np. w zadaniach obrazowania wyników tomografii komputerowej [29, 59]. Układy FPGA posiadają dedykowane wersje do zastosowań militarnych, gdzie na przykładzie firmy Xilinx można zastosować szyfrowane połączenia radiowe [5]. Nowoczesne bezzałogowe statki powietrzne wykorzystują technologię FPGA do planowania trajektorii lotu [7]. Układy FPGA można również wykorzystać do sterowania elektronicznymi układami kluczowania baterii kondensatorów [109].

W literaturze można znaleźć raporty wykorzystania układów FPGA do implementacji różnych wersji algorytmów regulacji predykcyjnej, zarówno w wersji analitycznej, jak i numerycznej.

Praca [11] przedstawia implementację analitycznego algorytmu GPC, który zastosowano do symulowanego procesu drugiego rzędu. Zastosowano arytmetykę stałopozycyjną 12 bitową. Niestety, nie przedstawiono podsumowania ostatecznej wydajności algorytmu. Zastosowany horyzont predykcji równy 5, horyzont sterowania 1. Porównano wyniki z wynikami uzyskanymi w środowisku MATLAB. Widoczne są duże różnice między dwoma wersjami rozwiązania. Nie wiadomo czy aż tak duży wpływ miała zastosowana arytmetyka. Wydaje się, że w wersji VHDL źle dobrano skalowanie wartości w wybranej arytmetyce i otrzymywano niedokładne wyniki. Implementację analitycznego algorytmu regulacji predykcyjnej typu explicit MPC dla struktury FPGA omówiono w pracy [60]. Zbadano wpływ parametrów, w tym horyzontów, na czas obliczeń oraz zajętość pamięci.

Ciekawym i dającym bardzo obrazowe porównanie jest praca [66], w której opisano implementację numerycznego algorytmu DMC w układzie FPGA. Autorzy motywują swoje badania poszukiwaniem alternatywnych rozwiązań dla drogich realizacji opartych o komputery przemysłowe. Zwracają uwagę na duży potencjał obliczeń równoległych. Wykorzystano realizację sprzętową w operacjach mnożenia i odwracania macierzy. Procesor NIOS, jako główna jednostka sterująca przepływem danych między dedykowanymi blokami, służy do szybkich obliczeń. Proponowana struktura systemu wydaje się być dość klasyczna w tego typu rozwiązaniu. Wyniki pokazują, że implementacja w układzie FPGA pozwala osiągnąć jakość regulacji praktycznie taką samą jak przy realizacji w języku MATLAB. Przedstawiono również porównanie efektywności bloku sprzętowego obliczeń macierzy i realizacji programowej w NIOS. Wyniki przy realizacji sprzętowej są dwukrotnie szybsze. Implementacja numerycznego algorytmu DMC i jego zastosowanie do procesu odsalania wody morskiej omówiono w pracy [72]. Proces jest wolny, stałe czasowe wynoszą 5 s, w każdej chwili próbkowania czas obliczeń nie przekracza 10 ms.

Implementację numerycznego algorytmu regulacji predykcyjnej, bazującego na liniowym modelu w przestrzeni stanu, omówiono w pracy [122]. Zastosowano sprzętowe bloki rozwiązujących zadanie programowania kwadratowego (metoda ograniczeń aktywnych). Algorytm wykorzystano do problemu synchronizacji między dwoma silnikami.

Ciekawa pozycja [103] przedstawia realizację ograniczeń w sterowanych pro-

cesach kosztem zmniejszenie dokładności obliczeń. W tym celu wykorzystano dedykowane bloki obliczeniowe w FPGA. Inny przykład realizacji algorytmu MPC w wersji numerycznej z wykorzystaniem notacji stałoprzecinkowej podano w pracy [14].

W pracy [74] omówiono integrację środowiska MATLAB z FPGA i językiem Handel C, wspomagającym implementację algorytmów numerycznych, bazujących na modelach liniowych w przestrzeni stanu, na platformach FPGA. Obliczenia wykonano dla symulowanego procesu samolotu Cessna. Wyniki pokazują, że zastosowanie układów FGPA w tego typu aplikacjach jest interesującą alternatywą i pozwala uzyskać porównywalne czasy obliczeń jak w przypadku komputerów klasy PC. Podkreślona została możliwość zastosowania do sterowania procesów wymagających krótkich okresów próbkowania, np. statków, pojazdów kosmicznych, samolotów czy samochodów.

Ciekawa pozycja [73] porusza tematykę zastosowania numerycznego algorytmu regulacji predykcyjnej, bazującego na modelach liniowych w przestrzeni stanu, w systemie wbudowanym (dla symulowanego procesu samolotu Cessna). Aby skrócić czas obliczeń zastosowano operacje równoległe. Wykonano optymalizację operacji przy rozwiązywaniu zadania programowania kwadratowego. Przedstawiono wyniki pokazujące, że realizacja równoległa pozwalała na praktycznie dwukrotne zmniejszenie czasu obliczeń.

Praca [85] przedstawia nowatorskie podejście do automatycznego generowania implementacji programowo-sprzętowej algorytmów sterowania przy wykorzystaniu środowiska MATLAB/Simulink. Nie podano czy generacja jest wspomagana wbudowanymi narzędziami MATLAB. Struktura działania algorytmów jest klasyczna, tzn. łącząca procesor NIOS i dedykowane bloki do obliczeń. Niestety, rozważania dotyczą tylko algorytmów regulacji PID. Nie podano możliwości realizacji regulatorów predykcyjnych. Przedstawiono schemat blokowy ścieżki konwersji od schematu w Simulinku do systemu w układzie FPGA.

Porównanie różnych metod rozwiązywania zadania kwadratowego problemu optymalizacji kwadratowej numerycznego algorytmu regulacji predykcyjnej (zastosowano model w przestrzeni stanu) oraz szczegóły implementacji przy wykorzystaniu układu FPGA podano w pracy [50]. Rozważono różne metody implementacji (jako bloki sprzętowe jak również rozwiązania programowe w soft-procesorze), pokazano wpływ liczby bitów na dokładność obliczeń. Podobna praca [37] pokazuje

wykorzystanie numerycznej wersji algorytmu predykcyjnego i porównanie wyników wydajności tego podejścia na platformie FPGA.

Ciekawe porównanie złożoności obliczeniowej dwóch metod rozwiązania zadań optymalizacji kwadratowej: metody punktu wewnętrznego i zbioru ograniczeń aktywnych, zaprezentowano w pracy [67]. W regulacji predykcyjnej rozważono liniowy model w przestrzeni stanu. Podano relację między liczbą zmiennych decyzyjnych oraz ograniczeń a złożonością obliczeniową.

Układy FPGA znalazły również zastosowanie do implementacji nieliniowych algorytmów regulacji predykcyjnej. Ciekawa praca [63] dotyczy algorytmu regulacji predykcyjnej dla procesów opisanych nieliniowymi równaniami w przestrzeni stanu. Do optymalizacji zastosowano algorytm sekwencyjnego programowania kwadratowego (ang. Sequential Quadratic Programming, SQP), w którym wielokrotnie rozwiązuje się klasyczne zadania optymalizacji kwadratowej. W trakcie optymalizacji kwadratowej używa się rozkładu macierzy QR, rozważono zrównoleglenie obliczeń podczas tej operacji. W pracy [119] również rozważa się nieliniowy algorytm regulacji predykcyjnej (aplikacja dotyczy sterowania zapłonem silnika spalinowego), ale do optymalizacji zastosowano heurystyczny algorytm roju cząstek (ang. Particle Swarm Optimisation, PSO).

Analizując cytowaną literaturę, można stwierdzić, że struktura „procesor NIOS + bloki dedykowane" uważana jest jako najwygodniejsze rozwiązanie. Zrównoleglenie obliczeń pozwala zwiększyć efektywność obliczeń. Taki sposób implementacji jest bardzo korzystny, nawet w przypadku realizacji klasycznych jednopętlowych regulatorów PID. Oczywiście, zrównoleglenie ma sens, jeżeli występuje wiele takich regulatorów, co ma miejsce w przypadku procesów wielowymiarowych.

Wszystkie omawiane rozwiązania pokazują bardzo ogólnikowe podejście do implementacji regulatorów predykcyjnych w układach logicznych. Brakuje wskazania pewnych procedur pozwalających zbudować własny system. Dlatego też w pracy tej postawiono duży nacisk na opracowanie spójnej metodyki projektowania takich systemów wychodząc od analizy procesu regulowanego, przechodząc do warstwy sprzętowo komunikacyjnej i kończąc na sposobach implementacji algorytmów regulacji predykcyjnej w warstwie logicznej. Wszystkie etapy zebrane w postaci szczegółowych opisów technicznych pozwalać będą na zaprojektowanie systemu wbudowanego opartego o struktury logiczne FPGA.

Rozdział 5

System do regulacji predykcyjnej oparty na strukturze FPGA: warstwa programowa

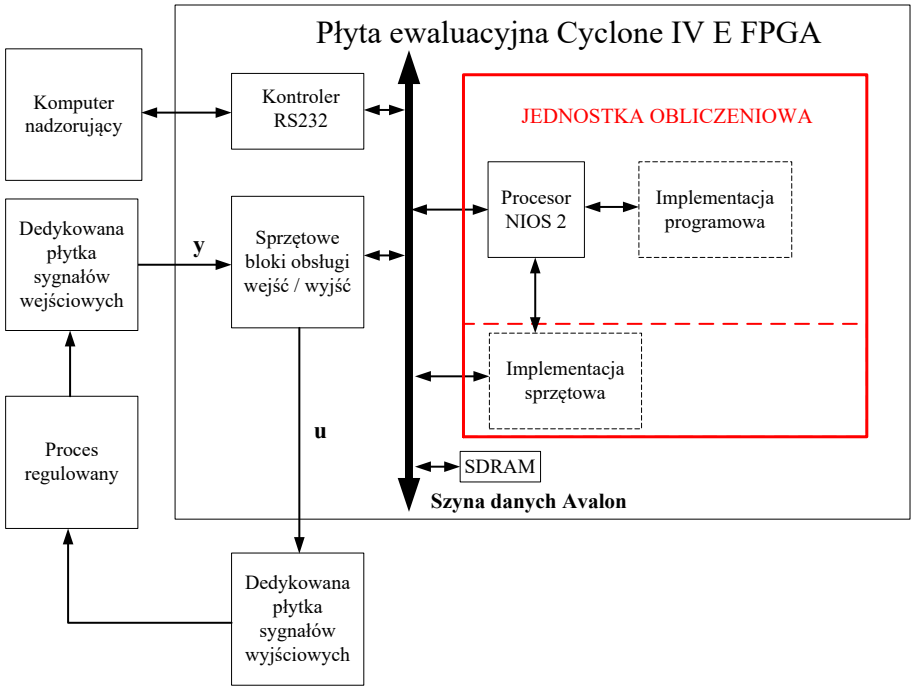
Opracowany system do regulacji predykcyjnej oparty na strukturze FPGA składa się z dwóch warstw: programowej oraz sprzętowej, wzajemnie od siebie zależnych. W niniejszym rozdziale omówiono spójne, uniwersalne podejście do implementacji algorytmów regulacji predykcyjnej przy wykorzystaniu układu FPGA. Oprócz klasycznego podejścia, gdzie wykorzystany jest procesor NIOS II jako jednostka obliczeniowa, pokazano również użycie dedykowanego akceleratora sprzętowego realizowanego w logice programowalnej. Jego zadaniem jest przyśpieszenie obliczeń algorytmów analitycznych w oszczędnej wersji obliczeniowej dzięki wykorzystaniu możliwości równoległej pracy układów logicznych.

5.1. Ogólna struktura systemu

Ogólna struktura systemu sterującego z układem FPGA została przedstawiona na rys. 5.1. Jej najważniejsze składniki są następujące:
— Płyta ewaluacyjna z układem FPGA Cyclone IV
— Dedykowane płyty interfejsowe sygnałów wejściowych i wyjściowych
— Proces regulowany
— Komputer nadzorujący - opcjonalnie stosowany do wizualizacji

Poniżej przedstawione zostały poszczególne etapy projektowania systemu sterowania opartego o układ FPGA:

1. Projekt i analiza procesu regulowanego
2. Dobór i projekt regulatora
3. Dobór i projekt warstwy sprzętowej interfejsów sterowania
4. Dobór i projekt warstwy sprzętowej układu sterowania
5. Implementacja VHDL interfejsów sterowania
6. Implementacja VHDL układu sterowania - procesor NIOS II [2], akceleratory



Rys. 5.1. Ogólna struktura systemu sterującego z układem FPGA

7. Implementacja C procesora sterującego - logika działania, algorytmy regulacji

Projekt i analiza procesu regulowanego jest najczęściej faktem dokonanym z którym projektant musi się zmierzyć. Dany jest proces, którego zachowanie oraz struktura musi zostać dokładnie przeanalizowana pod kątem dalszego doboru systemu. W przypadku tej pracy zostały zaprojektowane trzy procesy rzeczywiste, które różnią się stałymi czasowymi, liczbą wejść i wyjść, możliwością modyfikacji oraz stopniem skomplikowania.

Dobór i projekt regulatora możliwy jest dopiero po analizie procesu. Znając zachowanie oraz strukturę procesu można dokonać wyboru. Determinuje to również sposób reprezentacji modelu procesu w systemie - przykładowo dla regulatora DMC będzie to odpowiedz skokowa, dla regulatora GPC będzie to równanie różnicowe. Należy wziąć pod uwagę wersję algorytmu sterowania: analityczny, analityczny oszczędny, numeryczny. W przypadku wersji analitycznej oszczędnej należy także zdecydować czy będzie wykorzystany sprzętowy akcelerator obliczeń.

Dobór i projekt warstwy sprzętowej interfejsów sterowania to bardzo ważny

punkt od którego może zależeć prawidłowe próbkowanie sygnałów z procesu i sterowanie sygnałami wykonawczymi. Między innymi należy zapewnić odpowiednie kondycjonowanie poziomów sygnałów i izolację galwaniczną.

Dobór i projekt warstwy sprzętowej układu sterowania jest tutaj w pewnym sensie narzucony przez tematykę badawczą niniejszej pracy. Zakładamy wykorzystanie wydajnego układu FPGA. Zastosowano płytę ewaluacyjną z układem FPGA Intel Cyclone IV [1] do oprogramowania którego użyto pakietu Intel Quartus Prime [4]. Na płycie ewaluacyjnej dostępnych jest wiele form interfejsów komunikacyjnych, komputer wbudowany, różne możliwości diagnostycze oraz złącze do dedykowanych urządzeń użytkownika co pozwala w maksymalny sposób wykorzystać zasoby układu FPGA.

Implementacja VHDL interfejsów sterowania jest odpowiedzialna za prawidłową obsługę szybkich przetworników analogowo cyfrowych, bloków PWM o wysokiej rozdzielczości, warstwy komunikacyjnej czy sprzętowych liczników czasu. Istotnym faktem jest tutaj całkowicie niezależne działanie wszystkich bloków przetwarzania dzięki możliwości równoległej pracy układów logicznych.

Implementacja VHDL układu sterowania to po pierwsze stworzenie procesora NIOS II [2], który pełni rolę centralnego sterowania. Przygotowanie struktury sprzętowej procesora programowego NIOS II musi obejmować ilość potrzebnej pamięci i jej rodzaj, porty interfejsowe czy też dedykowane układy liczników czasu. Po drugie to dedykowane bloki sprzętowych akceleratorów obliczeń pozwalających przyśpieszyć obliczenia przez równoległe wykonywanie operacji - całkowicie niezależnie od procesora sterującego, który wykonuje swoje operacje sekwencyjnie. Po trzecie należy wykonać implementację sprzętową całego systemu w części logicznej. Oprócz procesora NIOS II, należy przewidzieć układy kondycjonowania zegara, dedykowane bloki do obsługi interfejsów pomiarowych oraz sterujących, komunikację, dostęp do pamięci czy też możliwość sterowania wyświetlaczem LCD.

Implementacja C procesora sterującego obejmuje część inicjalizacji interfejsów, przygotowania pamięci, właściwego automatu sterującego algorytmem regulacji, a ostatecznie sam algorytm regulacji. Główne sterowanie realizowane jest przez procesor NIOS II, który jest między innymi wyznacznikiem odpowiednich chwil, w których następuje uruchomienie odpowiedniego algorytmu programowego lub bloku obliczeń sprzętowych. W konfiguracji sprzętowej procesora NIOS II możliwe jest stworzenie własnej struktury z wymaganymi interfejsami i jednostkami operacyjnymi. W dalszej części rozdziału przedstawiona zostanie również realizacja

analitycznego algorytmu oszczędnego DMC oraz GPC w postaci całkowicie sprzętowej, przy wykorzystaniu dedykowanego bloku logicznego, wykonującego obliczenia. Procesor nadzoruje pracę całego systemu, aby zachować prawidłowe przetwarzanie sygnałów. W rozważanym systemie jeden procesor NIOS II może sterować kilkoma dedykowanymi blokami akceleracji, dzięki czemu można wykonywać kilka operacji równolegle. Może to również służyć do redundancji obliczeń numerycznych. Ma on również dodatkową funkcję, polegającą na komunikacji z komputerem nadrzędnym w celu wymiany danych pomiarowych oraz przekazywania nowych wartości zadanych. Środowiskiem nadrzędnym jest w tym przypadku MATLAB z odpowiednią aplikacją do wizualizacji pracy procesu i algorytmu regulacji.

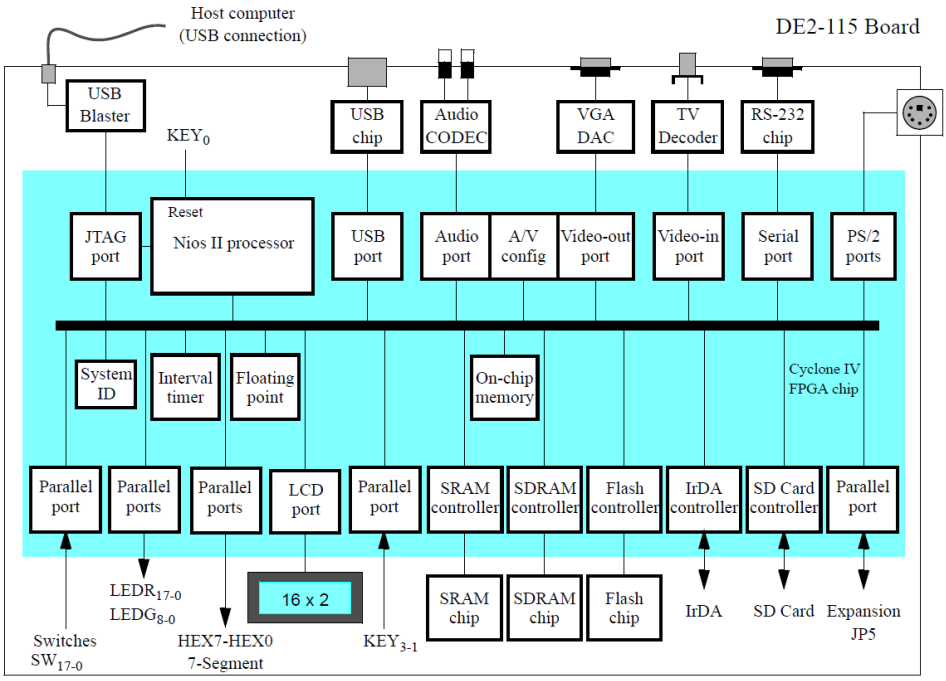
W kolejnych rozdziałach zostaną opisane szczegóły realizacji poszczególnych etapów projektowania systemu.

5.2. Przykładowa implementacja procesora NIOS II

Projekt części sprzętowej mikroprocesora NIOS II wykonuje się w dedykowanym środowisku QSYS [3] firmy Intel. Jest to program zintegrowany z produktem Quartus Prime. Oprogramowanie Qsys oferuje bibliotekę dostępnych komponentów procesora, dostarczoną przez producenta. Znajdują się tam między innymi główny rdzeń procesora, porty wejść i wyjść, układy zegarowe, moduł programowania JTAG, jednostkę obliczeń zmiennoprzecinkowych i wiele innych. Ponadto, istnieje możliwość dołożenia własnego dedykowanego modułu, który następnie można wywoływać w programie odpowiednią instrukcją. Oprogramowanie QSYS w wygodny sposób prezentuje elementy składowe mikroprocesora wraz z połączeniami między nimi, parametry poszczególnych elementów, a także dostarcza informacje o błędach.

Procesora NIOS II ma architekturę RISC (ang. Reduced Instruction Set Computing). Dostępne są rejestry 32 bitowe rejestry i zmiennoprzecinkowa jednostka obliczeniowa. Dodatkowo, można wykorzystać układy zegarowe, porty szeregowe, porty równoległe, a także kontrolery pamięci SRAM. Do diagnostyki działania programu można użyć modułu JTAG. Przykładowa struktura wewnętrzna mikroprocesora została przedstawiona na rys. 5.2.

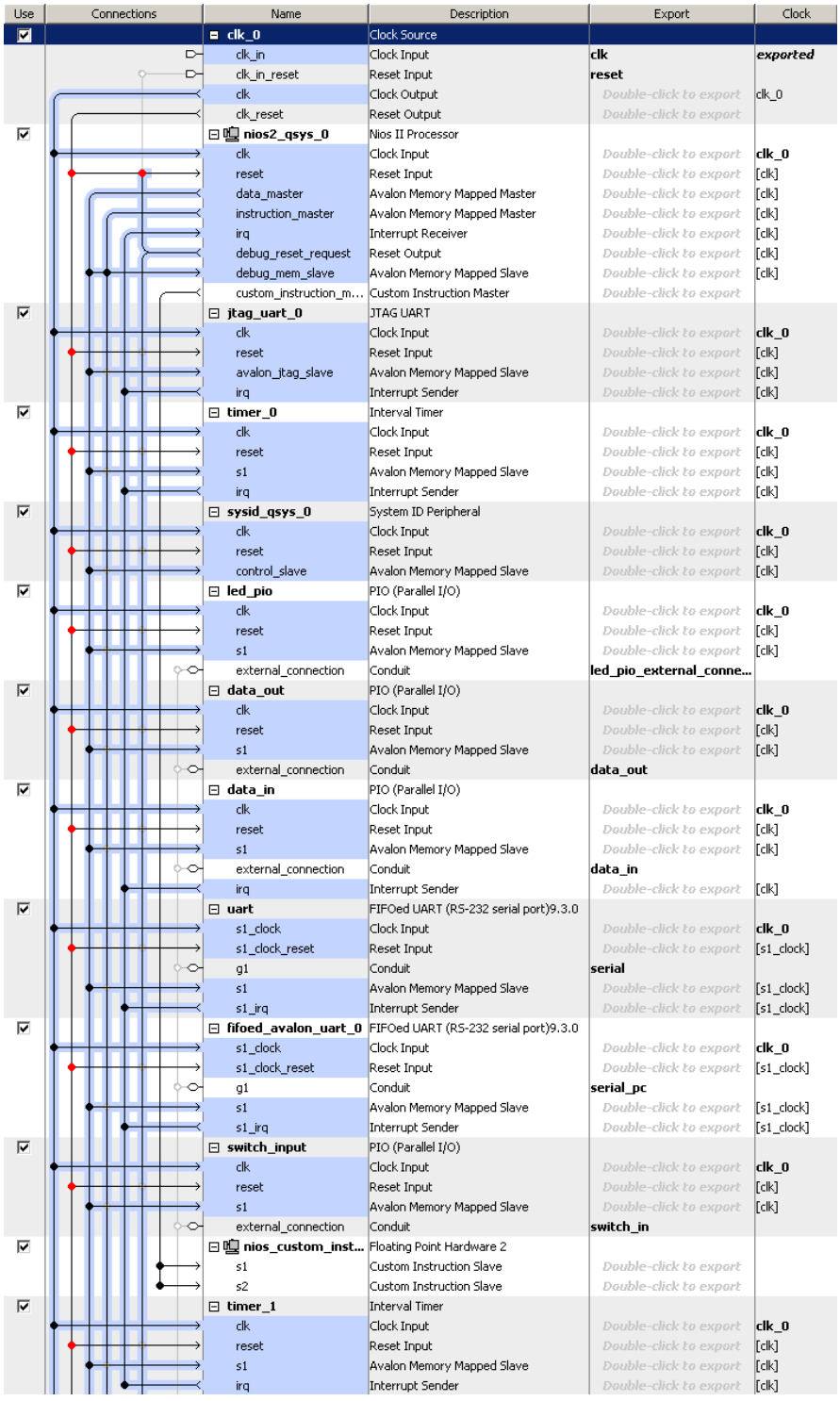
Wszystkie układy są podłączone do dedykowanej magistrali. Rdzeń procesora ma dostęp do wszystkich modułów przez odpowiednią adresację, którą ustala się w momencie projektowania mikroprocesora. Programowanie i komunikacja przez JTAG są realizowane przy pomocy modułu JTAG port, jest on również podłączony



Rys. 5.2. Przykładowa budowa strukturalna procesora NIOS II, źródło: ftp.intel.com/DE2-115/DE2-115-Media-Computer

bezpośrednio do rdzenia procesora do sprawowania operacji debugowania. Każdy procesor w systemie może być identyfikowany przez specjalny numer, który jest zakodowany w module System ID. Użycie licznika sprzętowego wymaga dołożenie modułu Interval Timer, który może zwracać do rdzenia mikroprocesora aktualną wartość, a także wysyłać przerwanie przy przepełnieniu licznika. Możliwe jest wykorzystanie dwóch rodzajów sterownika pamięci: On-chip memory, która jest pamięcią wykorzystującą zasoby układu FPGA (szybki dostęp), SDRAM Controller, która jest pamięcią zewnętrzną w postaci odrębnego układu scalonego, do którego przygotowany jest dedykowany interfejs wejść i wyjść. Moduły wejść i wyjść oraz komunikacyjne posiadają porty zewnętrzne, które muszą być podłączone do przygotowanych układów logicznych w pozostałej części FPGA lub do fizycznych pinów. Na rys. 5.3 przedstawiono listę modułów składających się na finalną architekturę mikroprocesora.

Poniżej wymienione zostały odpowiednie moduły z krótkim opisem celu ich zastosowania:



Rys. 5.3. Przykładowa implementacja procesora NIOS II

1. kondycjonowanie sygnału zegarowego: dedykowany sygnał zegarowy do wszystkim modułów procesora,
2. rdzeń procesora NIOS II w wersji e,
3. interfejs JTAG do komunikacji, programowania i debugowania,
4. jednostka obliczeń zmiennoprzecinkowych pojedynczej precyzji,
5. licznik czasu do realizacji opóźnienia i chwil taktowania,
6. identyfikator systemu,
7. 16-bitowe porty wejściowe: komunikacja z innymi blokami logicznymi,
8. 16-bitowe porty wyjściowe: komunikacja z innymi blokami logicznymi,
9. port komunikacyjny UART do komunikacji z komputerem PC,
10. szybka pamięć typu On-Chip Memory,
11. port do sterowania bloków VHDL dla przetworników C/A,
12. port do sterowania bloków VHDL dla przetworników A/C.

Istnieje możliwość wyboru wersji rdzenia procesora: wersja e – darmowa z podstawowymi możliwościami, wersja f – płatna z rozbudowanymi funkcjami, takimi jak sprzętowe mnożenie i dzielenie, pamięć podręczna (cache) instrukcji oraz danych, dodatkowy kontroler przerwań. W pracy wybrana została wersja darmowa, aby rozwiązanie było bardziej uniwersalne i nie zmuszało do zakupu płatnej wersji licencji. Istnieje również możliwość sprawdzenia opis jakie możliwości daje zastosowanie dedykowanego modułu obliczeń zmiennoprzecinkowych. Można sprawdzić ile cykli zegarowych wymagają poszczególne instrukcje arytmetyczne. Ponadto mamy możliwość zdecydowania, które instrukcje mają być realizowane przy pomocy dedykowanego modułu, a które będą realizowane w sposób programowy przez rdzeń mikroprocesora.

Po zakończeniu projektowania struktury mikroprocesora NIOS II należy wykonać kompilację projektu i generację odpowiednich plików w języku opisu sprzętu. Wygenerowane pliki są bezpośrednio importowane do projektu w środowisku Quartus i zostają wbudowane w cały projekt układu logicznego. Aby było to możliwe, wykorzystuje się przykładową instancję, która może zostać wygenerowana w programie QSYS. Należy wybrać w jakim języku opisu sprzętu ma zostać stworzona instancja i otrzymuje się gotowy kod do projektu. Dla rozważanego mikroprocesora definicja instancji została przedstawiona na rys. 5.4.

W pierwszej części kodu jest zdefiniowana lista portów, jakie są dostępne w mikroprocesorze. W szczególności, można sprawdzić które są wejściami lub wyjściami oraz jaka jest liczba bitów w słowie danego portu. W drugiej części kodu moż-

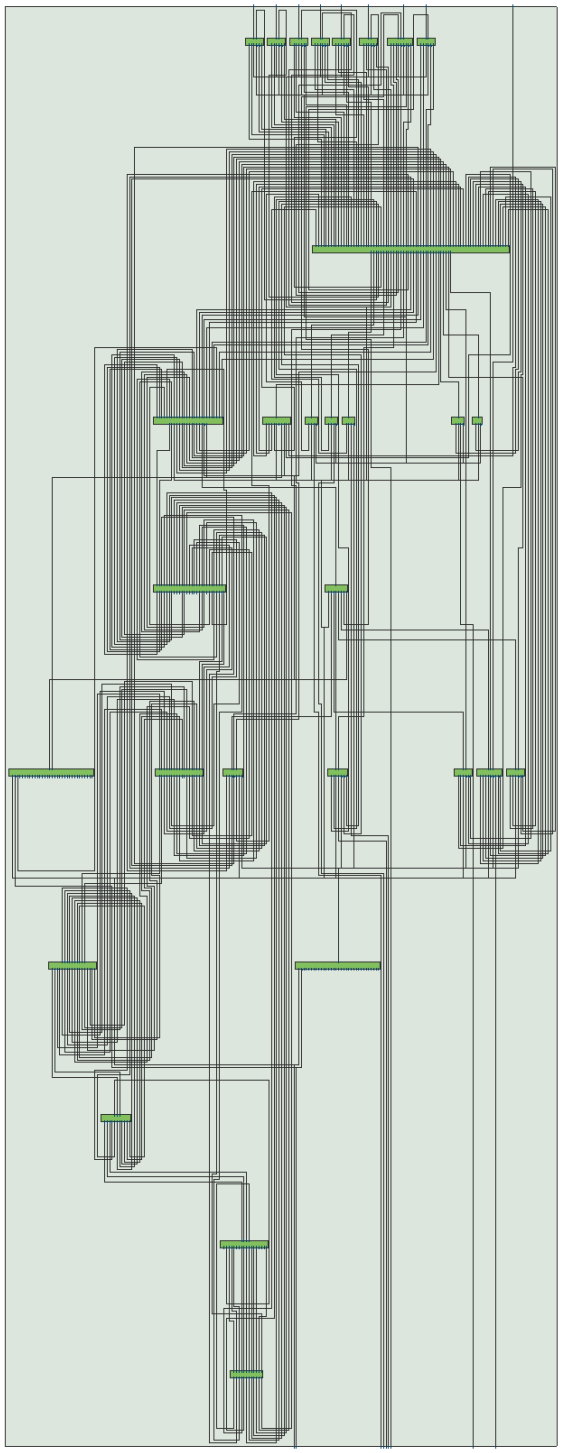
```
component Demo_test_NIOS is
    port (
        ab_analog_in_nios_external_export        : in  std_logic_vector(31 downto 0) := (others => 'X'); --
        ab_analog_out_external_connection_export : out std_logic_vector(31 downto 0);                     --
        cd_analog_in_nios_external_export        : in  std_logic_vector(31 downto 0) := (others => 'X'); --
        cd_analog_out_external_connection_export : out std_logic_vector(31 downto 0);                     --
        clk_clk                                  : in  std_logic                     := 'X';             --
        data_in_export                           : in  std_logic_vector(31 downto 0) := (others => 'X'); --
        data_out_export                          : out std_logic_vector(31 downto 0);                     --
        led_pio_external_connection_export       : out std_logic_vector(7 downto 0);                      --
        pwm_0_out_nios_external_export           : out std_logic_vector(31 downto 0);                     --
        pwm_1_out_nios_external_export           : out std_logic_vector(31 downto 0);                     --
        reset_reset_n                            : in  std_logic                     := 'X';             --
        serial_txd                               : out std_logic;                                        --
        serial_rxd                               : in  std_logic                     := 'X';             --
        serial_pc_txd                            : out std_logic;                                        --
        serial_pc_rxd                            : in  std_logic                     := 'X';             --
        switch_in_export                         : in  std_logic_vector(3 downto 0)  := (others => 'X'); --
        time_input_export                        : in  std_logic_vector(31 downto 0) := (others => 'X'); --
        time_pio_control_export                  : out std_logic_vector(7 downto 0)                      --
    );
end component Demo_test_NIOS;

u0 : component Demo_test_NIOS
    port map (
        ab_analog_in_nios_external_export        => CONNECTED_TO_ab_analog_in_nios_external_export,
        ab_analog_out_external_connection_export => CONNECTED_TO_ab_analog_out_external_connection_export,
        cd_analog_in_nios_external_export        => CONNECTED_TO_cd_analog_in_nios_external_export,
        cd_analog_out_external_connection_export => CONNECTED_TO_cd_analog_out_external_connection_export,
        clk_clk                                  => CONNECTED_TO_clk_clk,
        data_in_export                           => CONNECTED_TO_data_in_export,
        data_out_export                          => CONNECTED_TO_data_out_export,
        led_pio_external_connection_export       => CONNECTED_TO_led_pio_external_connection_export,
        pwm_0_out_nios_external_export           => CONNECTED_TO_pwm_0_out_nios_external_export,
        pwm_1_out_nios_external_export           => CONNECTED_TO_pwm_1_out_nios_external_export,
        reset_reset_n                            => CONNECTED_TO_reset_reset_n,
        serial_txd                               => CONNECTED_TO_serial_txd,
        serial_rxd                               => CONNECTED_TO_serial_rxd,
        serial_pc_txd                            => CONNECTED_TO_serial_pc_txd,
        serial_pc_rxd                            => CONNECTED_TO_serial_pc_rxd,
        switch_in_export                         => CONNECTED_TO_switch_in_export,
        time_input_export                        => CONNECTED_TO_time_input_export,
        time_pio_control_export                  => CONNECTED_TO_time_pio_control_export
    );
```

Rys. 5.4. Instancja procesora NIOS II w języku VHDL

na zobaczyć deklarację odpowiednich portów i fizyczne przypisania do sygnałów występujących w pozostałej części logicznej układu FPGA.

Po wykonaniu kompilacji i syntezy logicznej całego systemu w środowisku Quartus można wygenerować strukturę połączeń logicznych mikroprocesora. Przy tym skomplikowaniu układu struktura jest bardzo rozbudowana, a śledzenie ewentualnych błędów w takim schemacie może być bardzo trudne, niemniej jednak w szczególnych przypadkach można się posłużyć takim schematem do weryfikacji czy wyszukania błędów w realizacji mikroprocesora, np. przy stosowaniu własnych dedykowanych bloków obliczeń. Strukturę logiczną dla rozważanego mikroprocesora przedstawiono na rys. 5.5.



Rys. 5.5. Poglądowy schemat logiczny gotowej implementacji procesora NIOS II

Podsumowując implementację struktury mikroprocesora NIOS II, należy podkreślić uniwersalność rozwiązania. Dzięki zastosowaniu odpowiednich bloków można realizować własne operacje dedykowane. Ważny jest również fakt, że dany mikroprocesor można w dowolnym momencie przeprojektować, wygenerować ponownie opis instancji, a następnie zmodyfikować ją w finalnym projekcie. Dzięki temu, jeżeli w czasie realizacji dalszej części logicznej systemu lub też implementacji oprogramowania dla mikroprocesora okaże się, że wymagana jest jego modyfikacji, może to być wykonane w łatwy sposób.

5.3. Implementacja warstwy sprzętowej VHDL

Projekt części sprzętowej realizowany jest w dedykowanym środowisku Quartus Prime w wersji 17.0 obecnie firmy Intel (producentem była firma Altera). Jest to kompleksowe narzędzie do realizacji projektów dla układów FPGA (Field Programmable Gate Array), SoC (System on Chip) oraz CPLD (Comple Programmable Logic Devives). W kolejności wykonuje się implementację w języku opisu sprzętu, następnie uruchania się proces weryfikacji, syntezy logicznej oraz optymalizacji struktury ułożenia połączeń. Oprogramowanie posiada bogatą bazę podstawowych bloków logicznych. Możliwe jest również wykonanie symulacji w powiązanym pakiecie ModelSim.

W pracy wykonano podstawową konfigurację układu FPGA opierając się na języku VHDL. Główna jednostka zawiera, między innymi, instancje procesora NIOS II, modułów PLL, liczników sprzętowych, interfejsów analogowych oraz interfejsów PWM. Zarówno bloki PWM, jak i sygnały analogowe są przygotowane do pracy z częstotliwością próbkowania rzędu 1 kHz, aby zapewnić próbkowanie sygnału na poziomie 1 ms. Nie wykonuje się żadnych operacji filtracji cyfrowej sygnałów pochodzących z wejść analogowych. Definicja instancji oraz deklaracja interfejsów analogowych wejściowych przedstawiona została na poniższym listingu:

```
...

COMPONENT analog_in IS

PORT(
CLK     :       IN STD_LOGIC;
RST     :       IN STD_LOGIC;
DOUT    :       IN STD_LOGIC;
BUSY    :       IN STD_LOGIC;
DIN     :       OUT STD_LOGIC;
```

```
A       :       OUT STD_LOGIC_VECTOR(15 downto 0);
B       :       OUT STD_LOGIC_VECTOR(15 downto 0);
C       :       OUT STD_LOGIC_VECTOR(15 downto 0);
D       :       OUT STD_LOGIC_VECTOR(15 downto 0)
);

...

analog_in_inst : analog_in
PORT MAP(
CLK => clk_pll_analog_in_sig,
RST => NOT SW(16),
DOUT => GPIO(5),
BUSY => GPIO(7),
DIN => GPIO(1),

A => A_analog_sig,
B => B_analog_sig,
C => C_analog_sig,
D => D_analog_sig
);
```

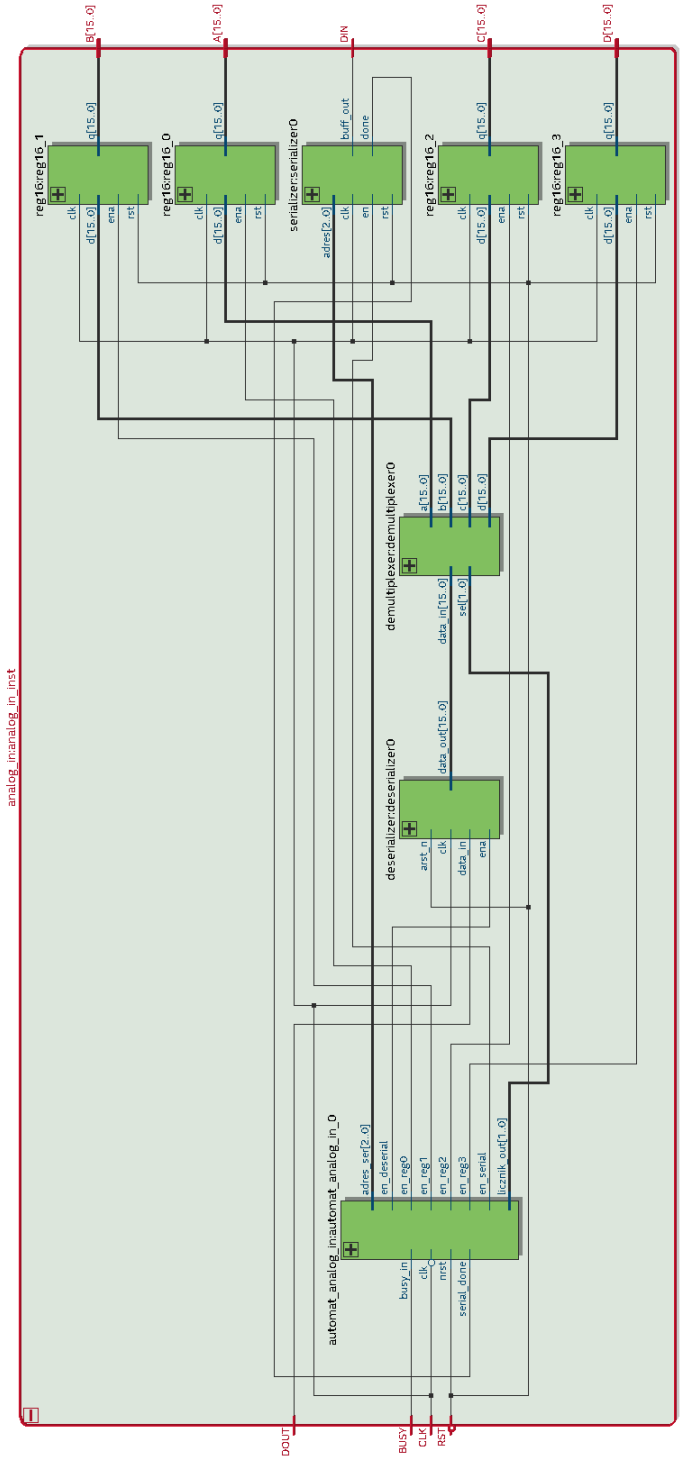
Na rys. 5.6 przedstawiono schemat logiczny modułu sterowania wejść analogowych wygenerowany podczas syntezy. Układ został zaprojektowany w ten sposób, aby w ciągu 1 ms zostały pobrane i umieszczone w rejestrach wartości z czterech przetworników analogowo cyfrowych (A, B, C, D). Dane pochodzące z przetwornika są podawane szeregowo jednym pinem, dlatego też później muszą zostać przetworzone w 16-bitowe rejestry i umieszczone w odpowiednich przerzutnikach wyjściowych. Całością pracy steruje całkowicie niezależny automat stanów zrealizowany tylko przy pomocy układów logicznych wykorzystując mechanizm procesu w języku VHDL.

Definicja instancji oraz deklaracja interfejsów analogowych wyjściowych przedstawiona została na poniższym listingu:

```
...

COMPONENT analog_out IS
PORT(
CLK     :       IN STD_LOGIC;
NRST    :       IN STD_LOGIC;
DIN     :       OUT STD_LOGIC;
SYNCA   :       OUT STD_LOGIC;
SYNCB   :       OUT STD_LOGIC;
SYNCC   :       OUT STD_LOGIC;
SYNCD   :       OUT STD_LOGIC;

A       :       IN STD_LOGIC_VECTOR(15 downto 0);
```



Rys. 5.6. Schemat logiczny modułu sterowania wejść analogowych

```
B       :       IN STD_LOGIC_VECTOR(15 downto 0);
C       :       IN STD_LOGIC_VECTOR(15 downto 0);
D       :       IN STD_LOGIC_VECTOR(15 downto 0)
);
END COMPONENT;

...

analog_out_inst : analog_out
PORT MAP(
CLK     =>      clk_PWM_sig,
NRST    =>      NOT SW(16),

DIN     =>      GPIO(10),
SYNCA   =>      GPIO(6),
SYNCB   =>      GPIO(4),
SYNCC   =>      GPIO(2),
SYNCD   =>      GPIO(0),

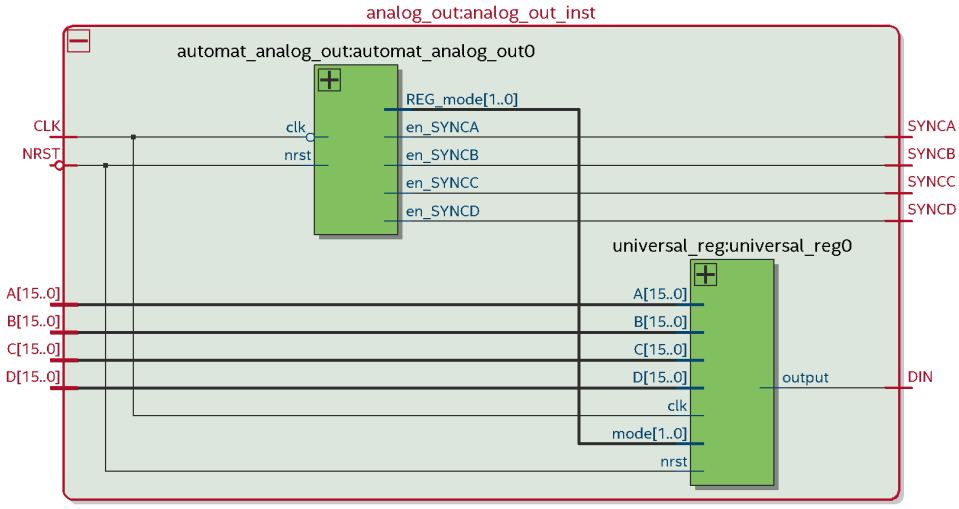
A       =>      ab_analog_out_nios_external_export_sig(31 downto 16),
B       =>      ab_analog_out_nios_external_export_sig(15 downto 0),
C       =>      cd_analog_out_nios_external_export_sig(31 downto 16),
D       =>      cd_analog_out_nios_external_export_sig(15 downto 0)
);
```

Na rys. 5.7 przedstawiono schemat logiczny modułu sterowania wyjść analogowych wygenerowany podczas syntezy. Układ został zaprojektowany w ten sposób, aby w ciągu 1 ms zostały ustawione na każdym wyjściu analogowym wartości z czterech rejestrów cyfrowych (A, B, C, D). Dane pochodzące z rejestrów są podawane równolegle i muszą zostać przetworzone w jeden sygnał podany do przetwornika. Całością pracy steruje całkowicie niezależny automat stanów zrealizowany tylko przy pomocy układów logicznych wykorzystując mechanizm procesu w języku VHDL. Automat w odpowiednich momentach ustawia wyjścia synchronizacji w korelacji do wybranego rejestru cyfrowego.

Definicja instancji oraz deklaracja interfejsów sygnałów PWM przedstawiona została na poniższym listingu:

```
...
COMPONENT PWM_output is
port(
clk     :       in std_logic;
data    :       in std_logic_vector(9 downto 0);
PWM     :       out std_logic
);
end COMPONENT;

...
```



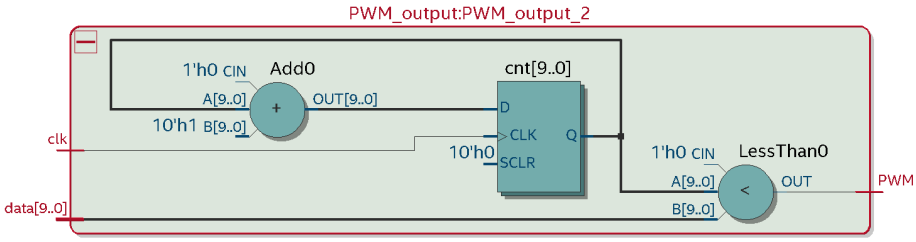
Rys. 5.7. Schemat logiczny modułu sterowania wyjść analogowych

```
PWM_output_0 : PWM_output
PORT MAP(
clk     =>      clk_PWM_sig,
data    =>      data_PWM_0_sig(9 downto 0),
PWM     =>      GPIO(13)
);

PWM_output_1 : PWM_output
PORT MAP(
clk     =>      clk_PWM_sig,
data    =>      data_PWM_1_sig(9 downto 0),
PWM     =>      GPIO(11)
);
```

Na rys. 5.8 przedstawiono schemat logiczny modułu sterowania wyjść PWM wygenerowany podczas syntezy. Układ został zaprojektowany w ten sposób, aby w ciągu 1 ms zaktualizować wartość wypełnienia sygnału. Częstotliwość nośna sygnału PWM wynosi 1 kHz. Sterowanie wypełnieniem jest realizowane przez licznik 10-bitowy. Najmniejszy kwant możliwy do ustawienia wynosi 1 ms/1024. Fizyczna realizacja tego układu została zrealizowana przy pomocy sumatora, licznika oraz układu porównania.

Należy podkreślić, że dzięki stosowaniu układów cyfrowych różne operacje są wykonywane równolegle i dzięki temu można w każdej chwili próbkowania (nawet co 1 ms) uzyskać aktualne pomiary z każdego wejścia analogowego, wysłać nowe



Rys. 5.8. Schemat logiczny modułu sterowania wyjść PWM

wartości sygnałów sterujących do każdego wyjścia analogowego oraz nowe wartości sygnałów do wyjść PWM.

Implementacja i złożenie całego systemu zostało wykonane w jednym pliku VHDL. Znajdują się w nim wszystkie instancje potrzebnych bloków logicznych oraz sygnały komunikacyjne między nimi. W wyniku kompilacji i syntezy całego projektu uzyskany został schemat logiczny systemu przedstawiony na rys. 5.9.

Szukanie błędu w powyższym schemacie logicznym jest dużym wyzwaniem. Kompilacja naturalnie nie podpowie o błędzie funkcjonalnym układów logicznych. Przy nieprawidłowej implementacji kodu opisu sprzętu (szczególnie w procesie) może się zdarzyć, że nie będzie możliwości syntezy danego algorytmu. Wówczas kompilacja i synteza zakończą się pomyślnie, ale wygenerowana struktura bramek logicznych w ogólnie nie będą przypominać projektowanego układu, uzyskany wynik nie będzie spełniał założonych wymagań funkcjonalnych. Aby skutecznie weryfikować takie sytuacje należy posłużyć się narzędziem do symulacji układów logicznych. Dla firmy Intel wbudowanym narzędziem jest program ModelSim. Jest to rozbudowane środowisko pozwalające w pełni symulować działanie kodu VHDL. Możliwe jest symulowanie oryginalnego kodu napisanego przez użytkownika oraz kodu, który jest dostępny po kompilacji i syntezie w systemie Quartus. Taka wersja kodu jest dokładnym odzwierciedleniem układu logicznego jaki będzie w układzie scalonym. Uwzględnia on opóźnienia przesyłania sygnałów i działania poszczególnych bramek. Dlatego też ważne jest aby weryfikować własny kod oraz kod wygenerowany przez program Quartus, aby mieć całkowitą pewność, że układ finalnie zadziała.

Dla ułatwienia weryfikacji układów w ModelSim można posłużyć się skryptami automatyzującymi testy. W skryptach takich można wykonać automatyczną kompilację kodu, dodać sygnały do rysowania, podać odpowiednia wymuszenia na

sygnałach zadanych, a ostatecznie uruchomić symulację na określony czas. Przykładowy fragment takie skryptu przedstawiono na poniższym listingu:

```
vlib work

vcom -reportprogress 300 -work work .../universal_reg.vhd
vcom -reportprogress 300 -work work .../automat_analog_out.vhd
vcom -reportprogress 300 -work work .../analog_out.vhd

vsim work.analog_out

force -freeze sim:/analog_out/clk 1 0, 0 {50 ps} -r 100
force -freeze sim:/analog_out/nrst 0 0
force -freeze sim:/analog_out/A 10#245 0
force -freeze sim:/analog_out/B 10#213 0
force -freeze sim:/analog_out/C 10#181 0
force -freeze sim:/analog_out/D 10#191 0

add wave sim:/analog_out/*

run
```

5.3.1. Sprzętowy akcelerator analitycznego algorytmu DMC

Warto bardziej szczegółowo omówić implementację sprzętową akceleratora, który służy do obliczenia częściowego wyniku algorytmu regulacji DMC w analitycznej wersji oszczędnej, wykorzystującej prawo regulacji określone wzorem (2.52). Opis dotyczy procesu z jednym wejściem i jednym wyjściem. Podano sposób obliczenia drugiego członu wzoru (2.52), czyli sumy zależnej od historii zmian wartości sterowania na horyzoncie dynamiki. Został on wybrany z uwagi na ilość wymaganych mnożeń będących następnie sumowanych. W realizacji takiego programu na mikrokontrolerze w języku C użyta zostanie pętla for, w której najpierw będzie wykonywane mnożenie, a następnie sumowanie do zmiennej tymczasowej. W przypadku rozwiązania sprzętowego możliwe jest wykonanie wszystkich mnożeń równolegle, a następnie zsumowanie wyników pośrednich. Do realizacji tego zadania w wersji sprzętowej przygotowano dedykowany blok Akcelerator, który przedstawiono na rys. 5.10. Wejście układu przyjmuje rejestr $\triangle u(k-i)$, który dostarczony jest z procesora NIOS II. Nowa wartość zostaje pobrana do bufora w momencie zmiany sygnału Shift, który uruchamia przerzutniki typu D. Kolejne wartości zmian sygnału sterowania przesuwają się do następnych przerzutników. W dalszej kolejności wartości te trafiają do układów mnożących je przez odpowiadające im współczynniki $\boldsymbol{K}_i^{\mathrm{u}}$. W ostatnim etapie wszystkie wartości są sumowane i zwracany jest wynik,

który trafi z powrotem do procesora NIOS II. Maksymalny czas wykonania wszystkich obliczeń będzie wynosił $1 + M + D * S$ taktów zegara, gdzie w kolejności można wyróżnić: 1 - ładowanie nowej wartości $\triangle u(k-i)$, M - wykonanie mnożeń, S - wykonanie pojedynczego sumowania.

Blok realizujący obliczenia został opisany przy pomocy reprezentacji funkcjonalnej. Poniżej przedstawiono fragment kodu źródłowego opisującego odpowiednie elementy składowe akceleratora.

```
BEGIN

D_P_1   :       D_P PORT MAP(
D_in    =>      DUK_1_IN,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      DUK_1);

D_P_2   :       D_P PORT MAP(
D_in    =>      DUK_1,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      DUK_2);

D_P_3   :       D_P PORT MAP(
D_in    =>      DUK_2,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      DUK_3);

ALTFP_MUL_1 : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => DUK_1,
datab    => K1u,
result => MUL_K1);

ALTFP_MUL_2 : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => DUK_2,
datab    => K1u,
result => MUL_K2);

ALTFP_MUL_3 : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => DUK_3,
datab    => K1u,
result => MUL_K3);

ALTFP_ADDER_1 : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => MUL_K1,
datab    => MUL_K2,
```

```
overflow         => overflow_sig(1),
result   => PSUM_1);

ALTFP_ADDER_2 : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => PSUM_1,
datab    => MUL_K3,
overflow         => overflow_sig(0),
result   => Wynik);
```

W pierwszej kolejności znajdują się definicje portów wejściowych i wyjściowych dla bloków przerzutników, następnie dla układów mnożących i ostatecznie dla sumatorów. Układy logiczne przerzutników napisano samodzielnie, natomiast układy logiczne mnożące i sumujące wykorzystano dedykowane bloki z biblioteki IP Intel - realizują one operacje na liczbach zmiennoprzecinkowych pojedynczej precyzji - dlatego też wszystkie magistrale danych są długości 32 bitów.

Opis funkcjonalny nie wymaga od użytkownika implementacji struktury układów logicznych. Zadanie to realizuje kompilator. W wyniku operacji kompilacji, a następnie syntezy (uzyskania prawidłowej struktury układów logicznych), otrzymuje się gotowy schemat połączeń układu. Dla realizowanego bloku akceleratora otrzymany został schemat logiczny, przedstawiony na rys. 5.11. Jest to fragment implementacji, gdzie widać układy przerzutników, efektywnych układów mnożących oraz sumujących. Procesor NIOS eksportuje do części logicznej obliczenia, które mogą zostać zrealizowane szybciej i efektywniej. W podobnej konwencji można przenosić fragmenty obliczeń dla pozostałych algorytmów predykcyjnych i dzięki temu jeszcze bardziej zwiększać szybkość obliczeń.

Warto w tym miejscu zauważyć, że struktura układów logicznych ma kluczowe znaczenie na wydajność obliczeń. Układy sumujące wykonują się w dalszym ciągu sekwencyjnie. Można to zmodyfikować i wykonać równolegle obliczenia dla odpowiednich par wyników z układów mnożących. Tworzy się dzięki temu pełne drzewo binarne, które złożoność obliczeniowa jest korzystniejsza. Przykładowe rozwiązanie w postaci schematu logicznego przedstawiono na rys. 5.12.

5.3.2. Sprzętowy akcelerator analitycznego algorytmu GPC

Zbliżone rozwiązanie można zaproponować dla obliczenia częściowego wyniku algorytmu regulacji GPC w analitycznej wersji oszczędnej, wykorzystującej prawo regulacji określone wzorem (2.73). Opis dotyczy procesu z jednym wejściem i jednym wyjściem. Podano sposób obliczenia sumy zależnej od historii wartości

sterowania oraz sumy zależnej od historii sygnałów z procesu. Został on wybrany z uwagi na ilość wymaganych mnożeń będących następnie sumowanych, które mogą być zrealizowane całkowicie niezależnie i równolegle. W realizacji takiego programu na mikrokontrolerze w języku C użyta zostanie pętla for, w której najpierw będzie wykonywane mnożenie, a następnie sumowanie do zmiennej tymczasowej. W przypadku rozwiązania sprzętowego możliwe jest wykonanie wszystkich mnożeń równolegle, a następnie zsumowanie wyników pośrednich. Do realizacji tego zadania w wersji sprzętowej przygotowano dedykowany blok Akcelerator, który przedstawiono na rys. 5.13. Wejście układu przyjmuje rejestry $u(k-i)$ oraz $y(k)$, które dostarczone są z procesora NIOS II. Nowe wartości zostają pobrane do buforów w momencie zmiany sygnału Shift, który uruchamia przerzutniki typu D. Kolejne wartości sygnału sterowania oraz wartości procesu przesuwają się do następnych przerzutników. W dalszej kolejności wartości te odpowiednio trafiają do układów mnożących je przez odpowiadające im współczynniki $\boldsymbol{K}_i^{\mathrm{u}}$ oraz $\boldsymbol{K}_i^{\mathrm{y}}$. W ostatnim etapie wszystkie wartości są sumowane i zwracany jest wynik, który trafi z powrotem do procesora NIOS II. Maksymalny czas wykonania wszystkich obliczeń będzie wynosił $1 + M + max(\lceil \log_2(n_b) \rceil, \lceil \log_2(n_a + 1) \rceil) * S + S$ taktów zegara, gdzie w kolejności można wyróżnić: 1 - ładowanie nowych wartości $u(k-i), y(k)$, M - wykonanie mnożeń, S - wykonanie pojedynczego sumowania.

Blok realizujący obliczenia został opisany przy pomocy reprezentacji funkcjonalnej. Poniżej przedstawiono fragment kodu źródłowego opisującego odpowiednie elementy składowe akceleratora.

```
BEGIN

D_P_1_U :       D_P PORT MAP(
D_in    =>      UK_1_IN,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      UK_1);

D_P_2_U :       D_P PORT MAP(
D_in    =>      UK_1,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      UK_2);

D_P_3_U :       D_P PORT MAP(
D_in    =>      UK_2,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      UK_3);
```

```
D_P_4_U :       D_P PORT MAP(
D_in    =>      UK_3,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      UK_4);

ALTFP_MUL_1_U : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => UK_1,
datab    => K0u,
result  => UMUL_K0);

ALTFP_MUL_2_U : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => UK_2,
datab    => K1u,
result => UMUL_K1);

ALTFP_MUL_3_U : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => UK_3,
datab    => K2u,
result => UMUL_K2);

ALTFP_MUL_4_U : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => UK_4,
datab    => K3u,
result => UMUL_K3);

ALTFP_ADDER_1_U : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => UMUL_K0,
datab    => UMUL_K1,
overflow         => overflow_sig(0),
result   => USUM_1);

ALTFP_ADDER_2_U : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => UMUL_K2,
datab    => UMUL_K3,
overflow         => overflow_sig(1),
result   => USUM_2);

ALTFP_ADDER_3_U : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => USUM_1,
datab    => USUM_2,
overflow         => overflow_sig(2),
result   => USUM_3);

-- wspolczynniki y
D_P_1_Y :       D_P PORT MAP(
```

```
D_in    =>      YK_IN,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      YK);

D_P_2_Y :       D_P PORT MAP(
D_in    =>      YK,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      YK_1);

D_P_3_Y :       D_P PORT MAP(
D_in    =>      YK_1,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      YK_2);

D_P_4_Y :       D_P PORT MAP(
D_in    =>      YK_2,
Clk     =>      Shift_IN,
Rst     =>      Rst,
Q_out   =>      YK_3);

ALTFP_MUL_1_Y : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => YK,
datab    => K0y,
result => YMUL_K0);

ALTFP_MUL_2_Y : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => YK_1,
datab    => K1y,
result => YMUL_K1);

ALTFP_MUL_3_Y : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => YK_2,
datab    => K2y,
result => YMUL_K2);

ALTFP_MUL_4_Y : ALTFP_MUL PORT MAP(
clock    => Clk,
dataa    => YK_3,
datab    => K3y,
result => YMUL_K3);

ALTFP_ADDER_1_Y : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => YMUL_K0,
datab    => YMUL_K1,
overflow         => overflow_sig(3),
result   => YSUM_1);
```

```
ALTFP_ADDER_2_Y : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => YMUL_K2,
datab    => YMUL_K3,
overflow         => overflow_sig(4),
result   => YSUM_2);

ALTFP_ADDER_3_Y : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => YSUM_1,
datab    => YSUM_2,
overflow         => overflow_sig(5),
result   => YSUM_3);

--wynik
ALTFP_ADDER_W : ALTFP_ADDER PORT MAP(
clock    => Clk,
dataa    => USUM_3,
datab    => YSUM_3,
overflow         => overflow_sig(6),
result   => wynik);
```

W pierwszej kolejności znajdują się definicje portów wejściowych i wyjściowych dla bloków przerzutników, następnie dla układów mnożących i ostatecznie dla sumatorów. Układy logiczne przerzutników napisano samodzielnie, natomiast układy logiczne mnożące i sumujące wykorzystano dedykowane bloki z biblioteki IP Intel - realizują one operacje na liczbach zmiennoprzecinkowych pojedynczej precyzji - dlatego też wszystkie magistrale danych są długości 32 bitów.

Opis funkcjonalny nie wymaga od użytkownika implementacji struktury układów logicznych. Zadanie to realizuje kompilator. W wyniku operacji kompilacji, a następnie syntezy (uzyskania prawidłowej struktury układów logicznych), otrzymuje się gotowy schemat połączeń układu. Dla realizowanego bloku akceleratora otrzymany został schemat logiczny, przedstawiony na rys. 5.14. Jest to fragment implementacji, gdzie widać układy przerzutników, efektywnych układów mnożących oraz sumujących. Procesor NIOS eksportuje do części logicznej obliczenia, które mogą zostać zrealizowane szybciej i efektywniej. W podobnej konwencji można przenosić fragmenty obliczeń dla pozostałych algorytmów predykcyjnych i dzięki temu jeszcze bardziej zwiększać szybkość obliczeń.

5.4. Implementacja warstwy programowej C

Rozdział ten obejmuje etap implementacji programowej w języku C dla procesora NIOS II. Zrealizowane zostaną funkcje inicjalizacji, główny automat stanów oraz odpowiednie algorytmy regulacji. W celach wizualizacji i diagnostyki wykorzystana zostanie komunikacja z komputerem PC. Jako środowisko programistyczne dla procesora NIOS II zastosowano Eclipse MARS.2 z dedykowaną wtyczką do obsługi FPGA. W oprogramowaniu dostępny jest debugger, dzięki któremu można śledzić każdy krok pracy procesora NIOS II. Ponadto, możliwe jest wyświetlenie konsoli diagnostycznej, na którą można wysyłać komunikaty z procesora NIOS II. Po wykonaniu implementacji sprzętowej mikroprocesora otrzymywany jest specjalny plik BSD, który zawiera informacje o budowie mikroprocesora użyteczne dla środowiska Eclipse. Zawarte informacje posiadają nagłówki z adresami odpowiednich interfejsów, dzięki czemu część programowa dysponuję informację określającą dokładne miejsca odniesienia w pamięci urządzenia.

5.4.1. Inicjalizacja funkcji sprzętowych

Część sprzętowa musi zostać na początku odpowiednio skonfigurowana: moduły muszą otrzymać odpowiednie parametry, a następnie muszą zostać uruchomione. Podano kilka najważniejszych konfiguracji. Poniższy listing przedstawia implementację inicjalizacji licznika czasu wbudowanego w mikroprocesor:

```
void init_timer0_interrupt(void)
{
 int* ptr;
 IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
                (1<<3) | (1 << 1) |(1 << 0));
 IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0);
        // Clear TO Bit(Reaching 0)
 IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_0_BASE,(alt_u16)(50000));
        //1ms //1ms - 50000//0,1ms - 5000
 IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_0_BASE,(alt_u16)(0));
 alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID, TIMER_0_IRQ,
                handle_timer0_interrupts, ptr, 0x0);
        //Register Interrupt
 IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
                (1<<2) | (1 << 1) | (1 << 0) );
 //Start Timer, IRQ enable, Continuous enable
}
```

Ustawiany jest okres odliczania czasu oraz powiązanie do funkcji wywoływanej w czasie przerwania od przepełnienia licznika. Po konfiguracji licznik czasu zostaje uruchomiony.

Poniższy listing przedstawia implementację funkcji przerwania dla licznika czasu wbudowanego w mikroprocesor

```
void handle_timer0_interrupts(void* context)
{
 IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0);

 czas_pomiar += 1;
}
```

Każde pojawienie się przerwania od odmierzenia czasu powoduje wyzerowanie odpowiedniego bitu przerwania. Następnie dodana zostaje wartość do zmiennej globalnej czas_pomiar. Zmienna ta pozwala na pomiar czasu z dokładnością do 1 ms przy wykorzystaniu dodatkowych funkcji operujących na zmiennej czas_pomiar.

Poniższy listing przedstawia implementację funkcji uruchamiającej odmierzanie czasu:

```
void mierz_start(void)
{
 czas_start = czas_pomiar;
}

long mierz_stop(void)
{
 czas_stop = czas_pomiar;
 czas_delta = czas_stop - czas_start;
 return czas_delta;
}
```

Aktualna wartość zmiennej czas_pomiar jest zapamiętywana w zmiennej czas_start. Kolejny listing pokazuje implementację funkcji zatrzymującej pomiar czasu i zwracającej wynik. Do zmiennej czas_stop jest przepisywana zmienna *czas_start* i obliczana jest różnica między tymi wartościami. Wynikiem jest czas w milisekundach.

Poniższy listing przedstawia implementację funkcji uruchamiającej odmierzanie czasu dla licznika zewnętrznego zaimplementowanego w warstwie sprzętowej:

```
void mierz_start_EXT(void)
{
 IOWR_ALTERA_AVALON_PIO_DATA(TIME_PIO_CONTROL_BASE,0);
 IOWR_ALTERA_AVALON_PIO_DATA(TIME_PIO_CONTROL_BASE,3);
}

long mierz_stop_EXT(void)
```

```
{
 IOWR_ALTERA_AVALON_PIO_DATA(TIME_PIO_CONTROL_BASE,1);
 czas_delta = IORD_ALTERA_AVALON_PIO_DATA(TIME_INPUT_BASE);
 IOWR_ALTERA_AVALON_PIO_DATA(TIME_PIO_CONTROL_BASE,0);
 return czas_delta;
}
```

Funkcja *mierz_start_EXT* uruchamia odliczanie czasu. Funkcja *mierz_stop_EXT* zatrzymuje odliczanie, odczytuje wynik z licznika i zwraca jego wartość. Dokładność pomiaru w tym przypadku jest rzędu 0.1 ms.

Poniższy listing przedstawia implementację funkcji inicjalizacji komunikacji przez port RS232:

```
void rs_pc_init()
{
 volatile int uart_context;

 alt_irq_register(FIFOED_AVALON_UART_0_IRQ, uart_context,
                  handle_uart_pc_interrupt);
 uart_pc_adres[3] = uart_pc_adres[3] | (1<<7);
        //przerwanie tylko od IRRDY
}
```

Należy przypisać funkcję obsługującą wystąpienie przerwania od portu odbierania danych. Kolejny listing jest już fragmentem automatu stanów, który realizuje odpowiednie dekodowanie odbieranych danych w zależności od znaków sterujących. Automat stanów został zrealizowany przy pomocy funkcji switch-case:

```
void handle_uart_pc_interrupt(void* context,alt_u32 id)
        //przerwanie od danych z RS
{
...

switch(rcv_pc_state){
        case READY_PC:
        if(rcv == 0xF0){
                rcv_pc_state = GET_PC_ID;
        }
        break;
        case GET_PC_ID:
        j = 0;
        if (rcv == 'Y')
        {
                rcv_pc_state = GET_YZAD;
                break;
        }
        rcv_pc_state = READY_PC;
        break;

        case GET_YZAD:
```

```
        j = 0;
        if (rcv == '1')
        {
                rcv_pc_state = GET_YZAD1;
                yzad_temp[1] = 0;
                wartosc_ujemna = 0;
                break;
        }
        if (rcv == '2')
        {
                rcv_pc_state = GET_YZAD2;
                yzad_temp[2] = 0;
                wartosc_ujemna = 0;
                break;
        }
        rcv_pc_state = READY_PC;
        break;

        ...
```

5.4.2. Ogólna organizacja pracy systemu

Program zawiera automat stanów, który steruje pracą całego systemu. Graficzna reprezentacja automatu stanów została przedstawiona na rys. 5.15. Po uruchomieniu mikroprocesora program przygotowuje parametry dla interfejsów sprzętowych wewnętrznych oraz zewnętrznych. Następnie, zgodnie z ustawieniami pracy zostają zadeklarowane odpowiednie zmienne dla algorytmu regulacji. W tym momencie algorytm rozpoczyna aktywne oczekiwanie na informację z pakietu MATLAB o trybie pracy. W zależności od wybranego trybu pracy uruchomiona zostaje procedura regulacji procesu lub akwizycji danych pomiarowych z procesu, np. tutaj odpowiedz skokowa. Przygotowana zostaje trajektoria wartości zadanych dla eksperymentu. W tym momencie algorytm jest już w pełni gotowy i oczekuje chwili próbkowania, która wyznaczana jest przez wewnętrzny zegar mikroprocesora. Procedura algorytmu regulacji rozpoczyna się od pobrania aktualnej wartości procesu. Następnie, pobrana zostaje wartość zadana sygnału i zgodnie z wybranym algorytmem zaczyna się proces obliczenia nowej wartości zadanej. Gdy nowa wartość będzie obliczona, zostanie zaaplikowana do regulowanego procesu. Zwiększa się licznik próbek i wykonywany jest test końca eksperymentu, którego warunkiem jest obliczenie zadanej liczby iteracji. W przypadku dalszej pracy algorytm będzie czekał na kolejną iterację obliczeń. W przeciwnym razie zebrane wyniki zostają przesłane do środowiska MATLAB i cały algorytm wraca do stanu początkowego.

Wybór parametrów regulacji dokonywany jest przy pomocy definicji stałych przedstawionych na poniższym listingu (przykład dotyczy procesu o dwóch wejściach i dwóch wyjściach):

```
#define nu 2    //liczba wejść
#define ny 2    //liczba wyjść
#define H_D 30  //horyzont dynamiki
#define H_P 10  //horyzont predykcji
#define H_S 3   //horyzont sterowania
#define KP 6    //początek sterowania
#define KK 600  //koniec sterowania
#define K_STEP 50       //skok jednostkowy
#define okres_probkowania 50    //okres próbkowania
#define LAMBDA_1 1.0f   //kara za sterowanie U1
#define LAMBDA_2 1.0f   //kara za sterowanie U2
#define PIS_1 1.0f      //współczynnik prognozowanych uchybow Y1
#define PSI_2 1.0f      //współczynnik prognozowanych uchybow Y2
#define MI 1.0f //kara za predykcje
#define DUMAX 10.0f     //maksymalny przyrost sterowania
#define UMIN -1.0f      //minimalna wartość sterowania
#define UMAX 1.0f       //maksymalna wartość sterowania
#define REG_TYP 2
//1 - DMC
//2 - GPC
//3 - DMC LC
//4 - GPC LC
#define REG_OPT 1
//0 - reg. analityczny
//1 - reg. numeryczny
#define CHS_CZAS_STAB 50 //czas stabilizacji
```

Możliwe są do ustawienia liczba wejść i wyjść procesu, horyzonty dynamiki, predykcji oraz sterowania, chwila początku i końca regulacji, chwila zmiany sygnału sterowania przy pobieraniu odpowiedzi skokowej, okres próbkowania, współczynniki funkcji celu, ograniczenie zmiany sygnału sterującego w pojedynczej iteracji, wartość minimalna sterowania, wartość maksymalna sterowania, rodzaj algorytmu, obecność optymalizacji oraz czas stabilizacji sygnału przy zbieraniu charakterystyki statycznej. Szczególną uwagę należy zwrócić na wybór algorytmu stałą REG_TYP. Możliwe jest wybranie algorytmu DMC analitycznego lub numerycznego, DMC analitycznego w wersji oszczędnej, GPC analitycznego lub numerycznego, GPC analitycznego w wersji oszczędnej. Wybór wersji regulacji z optymalizacją(numeryczny) lub bez optymalizacji(analityczny) możliwy jest dzięki definicji stałej REG_OPT.

5.4.3. Algorytmy analityczne w wersji oszczędnej

Rozważając algorytmy predykcyjne w oszczędnej wersji analitycznej, w pierwszym kroku należy zebrać informacje o procesie. Na rys. 5.16 przedstawiono ścieżkę postępowania, która jest analogiczna dla algorytmu DMC jak i GPC. Na początku należy pobrać odpowiedz skokową i wysłać ją do środowiska MATLAB. W kolejnym kroku wykonywana jest wstępna analizę, można zastosować filtrację odpowiedzi w celu wygładzenia jej charakteru. Kolejnym etapem jest dopasowanie modelu do pozyskanej odpowiedzi skokowej, stosuje się model inercji odpowiedniego rzędu i opóźnieniem. Ostatnim etapem jest generacji odpowiednich parametrów gotowych do wklejenia do kodu C regulatora właściwego.

Analityczny algorytm DMC może być zaimplementowany w wersji oszczędnej. Jest to rozwiązanie dedykowane dla systemów o mniejszych zasobach obliczeniowych. Implementacja taka jest reprezentowana na poniższym listingu (parametry podano dla procesu o dwóch wejściach i dwóch wyjściach, główna część programu jest uniwersalna):

```
//Definicja parametrów
#define lambda_u1 0.030000
#define lambda_u2 0.100000
#define nu 2
#define ny 2
#define D 60
float Ku[nu*(D-1)*nu] = {+9.862296e-01f,...
float Ke[nu*ny] = {+3.237607e+00f,+1.152858e-01f,
        -3.203784e-01f,+2.057038e+00f};

//Pomiar wartości z procesu
data_analog_ab = IORD_ALTERA_AVALON_PIO_DATA(AB_ANALOG_IN_NIOS_BASE);
data_analog_b =  data_analog_ab & 0x0000FFFF;
data_analog_a =  (data_analog_ab >> 16) & 0x0000FFFF;

//przetwornik 16 bitów - 65535 maksymalna wartość

sygn_y[0][k] = ((float)data_analog_a-32767.5f)/32767.5f;
        // przejscie z 0 - 65535 do -1.0 - 1.0
sygn_y[1][k] = ((float)data_analog_b-32767.5f)/32767.5f;
        // przejscie z 0 - 65535 do -1.0 - 1.0

//Wartosc zadana YZ1
if      (k< 100) sygn_yzad[0] =  .1f;
else if(k< 200) sygn_yzad[0] = .5f;
else if(k< 300) sygn_yzad[0] = .6f;
else if(k< 400) sygn_yzad[0] = .5f;
else            sygn_yzad[0] = .5f;

// Wartosc zadana YZ2
```

```
if      (k< 150) sygn_yzad[1] =  .15f;
else if(k< 250) sygn_yzad[1] = .2f;
else if(k< 350) sygn_yzad[1] = .4f;
else if(k< 450) sygn_yzad[1] = .1f;
else            sygn_yzad[1] = .1f;

//Algorytm
for(m=0; m<ny; ++m)
{
        Ydiff[m] = sygn_yzad[m] - sygn_y[m][k];
}

i=0;
for(p=0; p<(D-1); ++p)
{
        for(n=0; n<nu; ++n)
        {
                dUp[i] = 0.0f;
                if(k >= p+1)
                dUp[i] += sygn_u[n][k-p-1];
                if(k >= p+2)
                dUp[i] -= sygn_u[n][k-p-2];
                ++i;
        }
}

mat_mul(Ke,nu,ny, Ydiff,ny,1, sygn_du);
mat_mul(Ku,nu,(D-1)*nu, dUp,(D-1)*nu,1, dutmp);

for(n=0; n<nu; ++n)
{
        sygn_u[n][k] = (sygn_du[n]-dutmp[n]) + sygn_u[n][k-1];
}

//Rzutowanie na zbiór ograniczeń
for(n=0; n<nu; ++n)
{
        if(sygn_u[n][k] > umax)
        sygn_u[n][k] = umax;
        if(sygn_u[n][k] < umin)
        sygn_u[n][k] = umin;
}

//Wysłanie wartości sterowania do procesu
pwm_0 = (long)((sygn_u[0][k]*511.5f)+511.5f);
        // przejście z -1 - 1 do 0 - 1023
IOWR_ALTERA_AVALON_PIO_DATA(PWM_0_OUT_NIOS_BASE,pwm_0);
pwm_1 = (long)((sygn_u[1][k]*511.5f)+511.5f);
        // przejście z -1 - 1 do 0 - 1023
IOWR_ALTERA_AVALON_PIO_DATA(PWM_1_OUT_NIOS_BASE,pwm_1);
```

W pierwszej kolejności należy zwrócić uwagę na parametry wejściowe algorytmu.

Wykorzystywany jest model odpowiedzi skokowej, współczynniki funkcji celu, na bazie których obliczone zostają parametry regulatora Ke oraz Ku (wzory (2.49) oraz (2.50)). Parametry te mogą być obliczone w środowisku MATLAB, a następnie przeniesione jako stałe do algorytmu wykonawczego w języku C. W trakcie regulacji pierwszą czynnością jest pobranie wartości sygnału z procesu i przeskalowanie jej do wartości znormalizowanej, używanej w algorytmie. Pojedyncza iteracja algorytmu oblicza optymalne wartości przyrostów sygnałów sterujących dla aktualnej chwili czasowej bazując na historii sygnałów sterujących oraz współczynnikach Ku, a także na aktualnym uchybie i wartości współczynnika Ke. Obliczone przyrosty są sumowane z ostatnią wartością sterowania i następnie rzutowane na zbiór ograniczeń (2.54). Obliczone wartości sygnałów sterujących są gotowe do wysłania do obiektu regulacji dopiero do odpowiednim przeskalowaniu na wartości akceptowane. Wysłanie sygnałów sterujących do obiektu kończy iterację algorytmu.

Dla algorytmu DMC w wersji oszczędnej z wykorzystaniem akceleratora obliczeń zadania realizowane przez procesor NIOS II będą bardzo podobne do powyższych wykonanych wyłącznie w języku C. Kolejne kroki algorytmu przedstawiono na rys. 5.17.

Najważniejsza różnica polega na tym, że w wersji z akceleratorem, po wykonaniu obliczenia nowych wartości sygnałów sterujących do akceleratora wysyłane są nowe wartości różnic sygnałów sterujących w stosunku do poprzedniej iteracji. Dzięki temu, obliczenia rozpoczynają się w części sprzętowej FGPA, a program w języku C kończy pętlę regulacji i może wykonywać inne zadania. Gdy nastąpi kolejna chwila dyskretna (aktualizacji sygnałów sterujących), obliczenia sprzętowe są już zakończone i można pobrać nowe wyniki. Wówczas w procesorze NIOS jest realizowanych bardzo mało obliczeń i nowe wartości sygnałów sterujących mogą zostać wysłane do procesu.

Również analityczny algorytm GPC może być zaimplementowany w wersji oszczędnej. Jest to rozwiązanie dedykowane dla systemów o mniejszych zasobach obliczeniowych. Implementacja taka jest reprezentowana na poniższym listingu (parametry podano dla procesu o dwóch wejściach i dwóch wyjściach, główna część programu jest uniwersalna):

```
//Definicja parametrów
#define lambda_u1 5.000000
#define lambda_u2 30.000000
#define na 2
#define nb 2
#define nu 2
```

```
#define ny 2
const float b[ny][nu][nb] = {{{+1.219184e-01f,...;
const float a[ny][na] = {{-1.531794e+00f,...;
float Kyzad[nu][ny] = {{+3.947612e-01f,...;
float Ky[nu][ny][na+1] = {{{-4.287331e+00f,...;
float Ku[nu][nu][nb] = {{{+3.708263e-01f,...;

//Pomiar wartości z procesu
data_analog_ab = IORD_ALTERA_AVALON_PIO_DATA(AB_ANALOG_IN_NIOS_BASE);
data_analog_b =  data_analog_ab & 0x0000FFFF;
data_analog_a =  (data_analog_ab >> 16) & 0x0000FFFF;
//przetwornik 16 bitow - 65535 maksymalna wartosc
sygn_y[0][k] = ((float)data_analog_a-32767.5f)/32767.5f;
        // przejscie z 0 - 65535 do -1.0 - 1.0
sygn_y[1][k] = ((float)data_analog_b-32767.5f)/32767.5f;
        // przejscie z 0 - 65535 do -1.0 - 1.0

//Ustawienie wartości zadanej
if      (k< 100) sygn_yzad[0] =  .1f;
else if(k< 200) sygn_yzad[0] = .5f;
else if(k< 300) sygn_yzad[0] = .6f;
else if(k< 400) sygn_yzad[0] = .5f;
else            sygn_yzad[0] = .5f;

if      (k< 150) sygn_yzad[1] =  .15f;
else if(k< 250) sygn_yzad[1] = .2f;
else if(k< 350) sygn_yzad[1] = .4f;
else if(k< 450) sygn_yzad[1] = .1f;
else            sygn_yzad[1] = .1f;

//Algorytm
for(r=0;r<nu;++r)
{
        du[r] = 0.0f;
        for(m=0;m<ny;++m)
        {
                du[r] += Kyzad[r][m]*sygn_yzad[m];
        }
        for(n=0;n<nu;++n)
        {
                for(j=0;j<nb;++j)
                {
                        du[r] += Ku[r][n][j]*sygn_u[n][k-j-1];
                        // dla j=0 -> u[n][k-1]
                }
        }
        for(m=0;m<ny;++m)
        {
                for(j=0;j<(na+1);++j)
                {
                        du[r] += Ky[r][m][j]*sygn_y[m][k-j];
                        // dla j=0; -> y[m][k]
                }
```

```
        }
}

for(n=0;n<nu;++n)
{
        sygn_u[n][k] = du[n]+sygn_u[n][k-1];
}

//Rzutowanie na zbiór ograniczeń
for(n=0; n<nu; ++n)
{
        if(sygn_u[n][k] > umax)
                sygn_u[n][k] = umax;
        if(sygn_u[n][k] < umin)
                sygn_u[n][k] = umin;
}

//Wysłanie wartości sterowania do procesu
pwm_0 = (long)((sygn_u[0][k]*511.5f)+511.5f);
        // przejście z -1 - 1 do 0 - 1023
IOWR_ALTERA_AVALON_PIO_DATA(PWM_0_OUT_NIOS_BASE,pwm_0);
pwm_1 = (long)((sygn_u[1][k]*511.5f)+511.5f);
        // przejście z -1 - 1 do 0 - 1023
IOWR_ALTERA_AVALON_PIO_DATA(PWM_1_OUT_NIOS_BASE,pwm_1);
```

Na podstawie współczynników modelu generowane są wektory Kyzad, Ku oraz Ky (ze wzorów (2.74)). Odpowiednie obliczenia mogą zostać wykonane w skrypcie programu MATLAB i następnie wstawione do kodu w języku C. Należy podkreślić, że liczba parametrów jest znacząco niższa niż w przypadku algorytmu DMC. W trakcie regulacji pierwszą czynnością jest pobranie wartości sygnałów wyjściowych z procesu i przeskalowanie ich do wartości znormalizowanych, używanych w algorytmie. Pojedyncza iteracja algorytmu oblicza optymalne wartości przyrostów sygnałów sterujących dla aktualnej chwili dyskretnej, bazując na historii sygnałów sterujących oraz współczynnikach Ku, a także na aktualnych uchybach i współczynnikach Ke. Obliczone przyrosty są sumowane z ostatnią wartością sterowania i następnie rzutowane na zbiór ograniczeń (2.54). Obliczone wartości sygnałów sterujących są gotowe do wysłania do obiektu regulacji dopiero do odpowiednim przeskalowaniu na wartości akceptowane. Wysłanie sygnałów sterujących do obiektu kończy iterację algorytmu.

5.4.4. Algorytmy analityczne (w wersji pełnej) i numeryczne

Dla pełnych wersji algorytmów analitycznych oraz numerycznych wszystkie obliczenia są wykonywane w mikroprocesorze sprzętowym. Zaletą tego rozwiązania jest

brak potrzeby stosowanie dodatkowego narzędzia w postaci środowiska MATLAB. Ponadto, możliwe jest dostrajanie parametrów regulatora bez potrzeby ponownego przetwarzania danych przez narzędzia zewnętrzne. Wadą tego rozwiązania jest o wiele większa zajętość pamięci programu i pamięci operacyjnej mikroprocesora. Ponadto, czas obliczenia pojedynczej iteracji algorytmu jest również większy. Do pracy algorytmu potrzebny jest model procesu. Na bazie modelu wykonywane są wszystkie niezbędne obliczenia dla wybranego algorytmu ze zbioru: DMC analityczny, DMC numeryczny, GPC analityczny, GPC numeryczny.

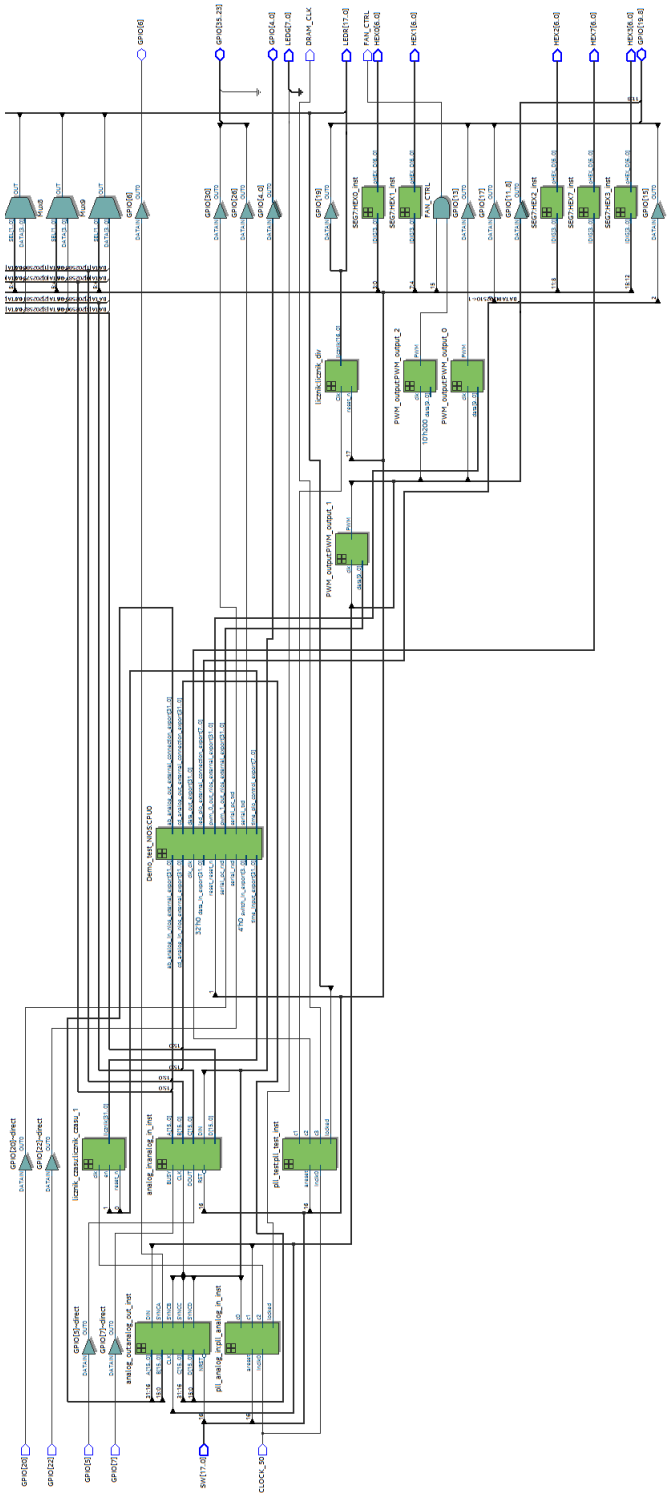
Podczas inicjalizacji dowolnego algorytmu zawsze wykonywana jest procedura przedstawiona na rys. 5.18 zaimplementowana jako funkcja inicjalizacjamodelu(void). W pierwszej kolejności przydziela ona pamięć na dane współczynników, a następnie ustawia wartości współczynników modelu. Liczba oraz wartości współczynników wynikają z dobranego w środowisku MATLAB modelu.

Drugi etap inicjalizacji to przygotowanie odpowiednich obszarów pamięci na dane wykorzystywane podczas regulacji. Zajmuje się tym funkcja inicjalizacjasygnalow(void) przedstawiona na rys. 5.19, której głównym zadaniem jest przydzielenie pamięci na tablice sygnałów sterujących, wyjściowych i zadanych.

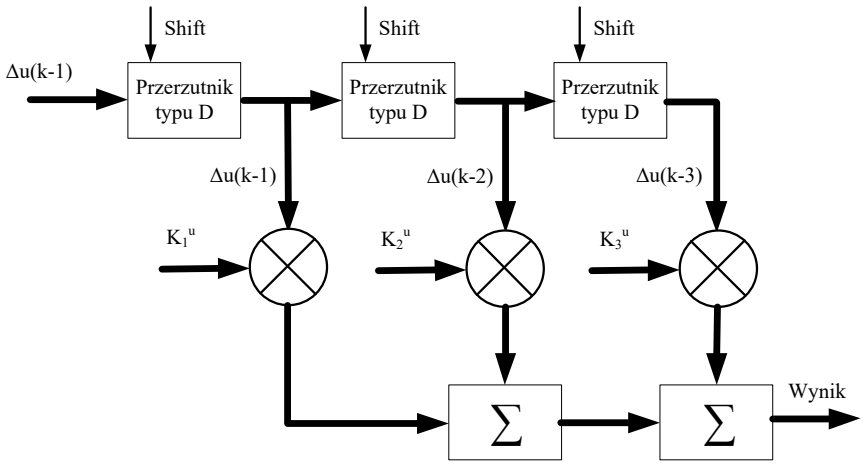
Ostatnią funkcją na etapie inicjalizacji jest inicjalizacjaregulatora(void) przedstawiona na rys. 5.20, która przydziela pamięć i ustala wartości głównych zmiennych wykorzystywanych podczas procesu regulacji. Kolejne operacje wykonują przydzielenie pamięci dla współczynników kar $\psi_1, \ldots, \psi_{n_y}$ oraz $\lambda_1, \ldots, \lambda_{n_u}$, macierzy współczynników odpowiedzi skokowej, macierzy $\boldsymbol{M}$ (2.27), macierzy $\boldsymbol{M}^{\mathrm{P}}$ (2.35) dla regulatora DMC, w wersji bez optymalizacji przygotowanie macierzy $\boldsymbol{K}$ (2.46), dla wersji z optymalizacją przygotowanie macierzy $\boldsymbol{H}$ (2.41), macierzy $\overline{\boldsymbol{A}}$ i wektora $\overline{b}$, definiujących ograniczenia (2.43) zadania optymalizacji.

W każdej iteracji algorytmu wywoływana jest funkcja regulator(), która realizuje właściwe obliczenie nowej wartości sygnałów sterujących. Jest ona przedstawiona na rys. 5.21. Implementacja tej funkcji jest zrealizowana dla algorytmów analitycznych i numerycznych. W zależności od wybranego rodzaju algorytmu wykonują się odpowiednie fragmenty implementacji. Dla algorytmu DMC, na początku wyznaczane są wartości wektora $\triangle\boldsymbol{U}^{\mathrm{P}}$ (2.36) oraz $\boldsymbol{Y}$, a następnie obliczony zostaje wektor odpowiedzi swobodnej $\boldsymbol{Y}^0$ (2.32). Dla algorytmu GPC wyznaczana jest odpowiedź swobodna ze wzoru (2.60). Dla wersji analitycznej algorytmu obliczany zostaje optymalna wartość przyrostu sterowania na horyzoncie sterowania. Dla wersji numerycznej algorytmu najpierw obliczane są wektory ograniczeń $\overline{\boldsymbol{A}}$ i $\overline{b}$,

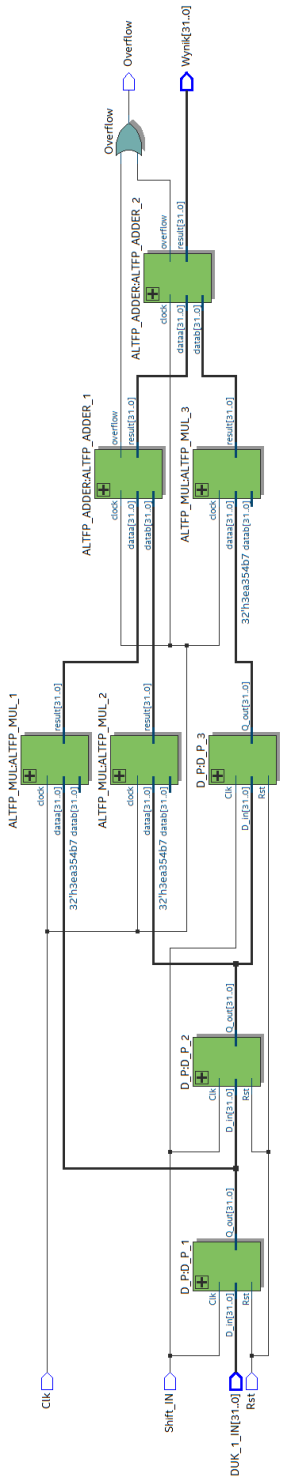
następnie obliczana jest funkcja kryterialna, która będzie podlegać optymalizacji i ostatecznie wywoływana jest właściwa procedura optymalizacji pk(). Otrzymane wartości przyrostów sygnałów sterujących zostają uwzględnione dla wartości aktualnego sterowania (2.53). Przed ostateczną aplikacją tych wartości do obiektu regulacji zostają rzutowane na zbiór ograniczeń (2.54).



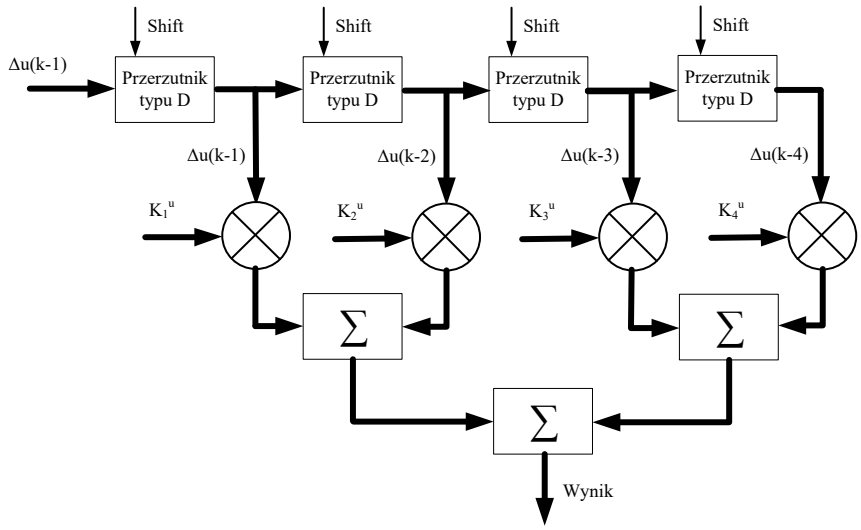
Rys. 5.9. Poglądowy schemat logiczny systemu w FPGA



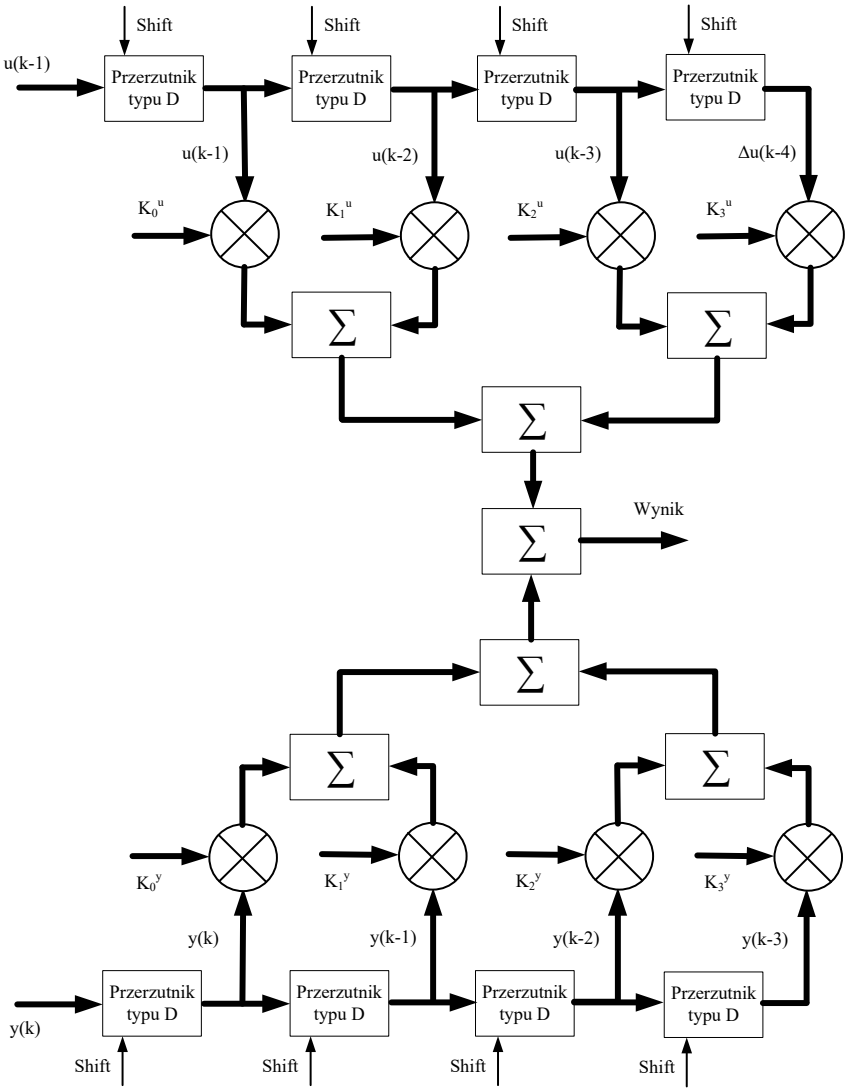
Rys. 5.10. Blok akceleratora analitycznego algorytmu DMC w strukturze FPGA



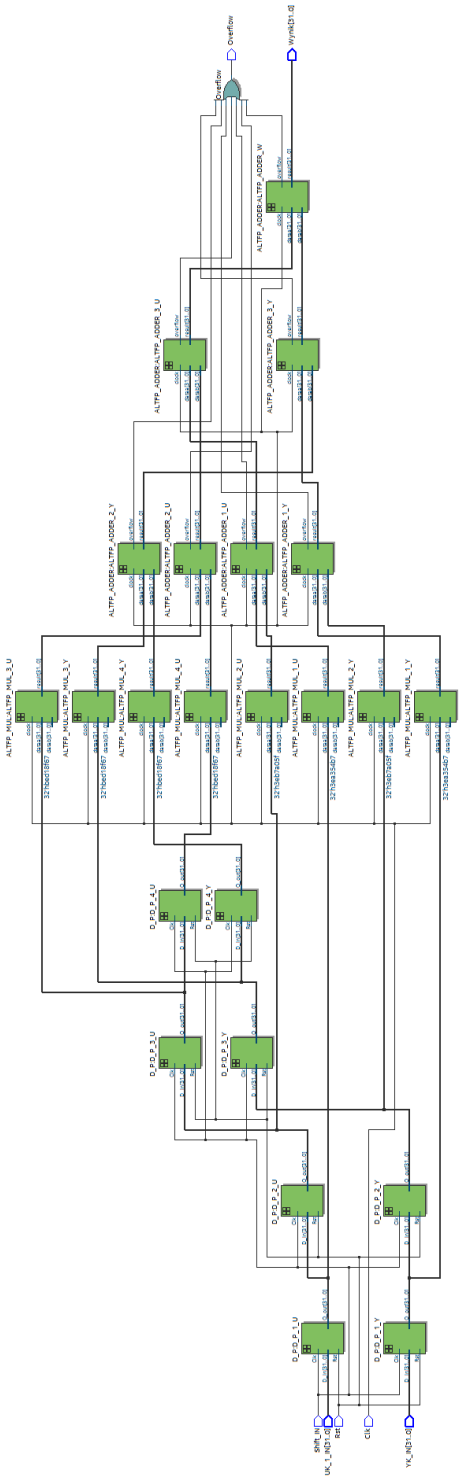
Rys. 5.11. Logika realizująca blok akceleratora analitycznego algorytmu DMC



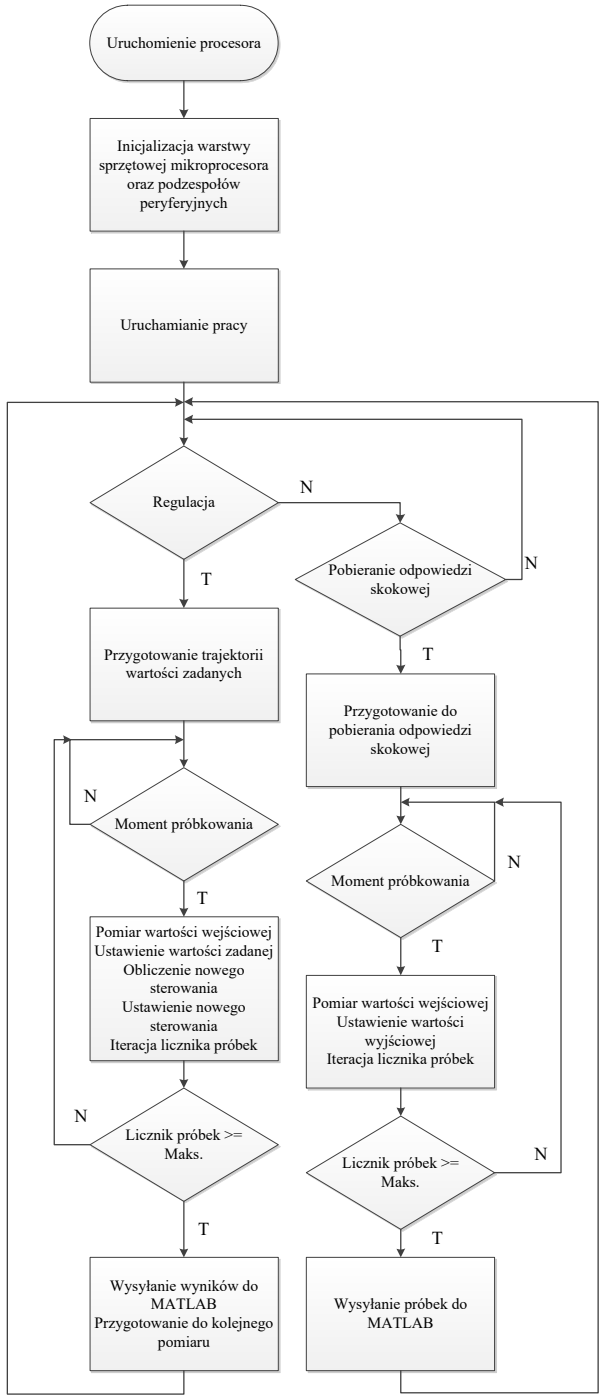
Rys. 5.12. Blok akceleratora analitycznego algorytmu DMC w strukturze FPGA - wersja z równoległym sumowaniem



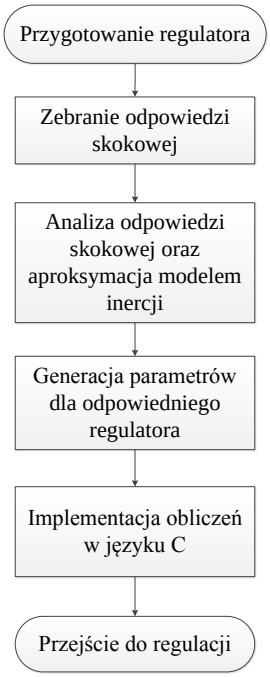
Rys. 5.13. Blok akceleratora analitycznego algorytmu GPC w strukturze FPGA



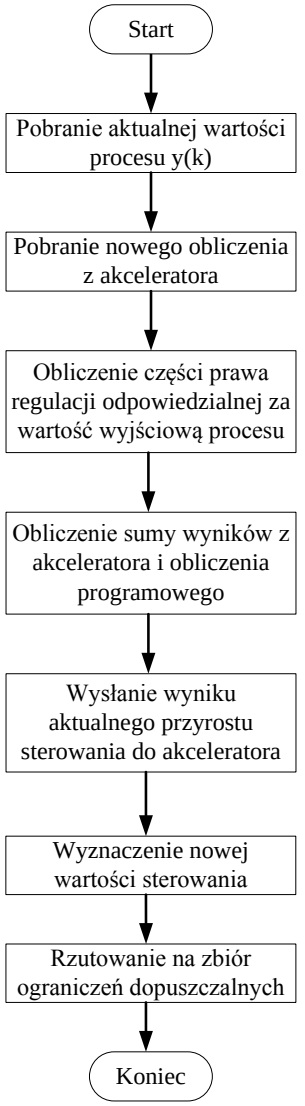
Rys. 5.14. Logika realizująca blok akceleratora analitycznego algorytmu GPC



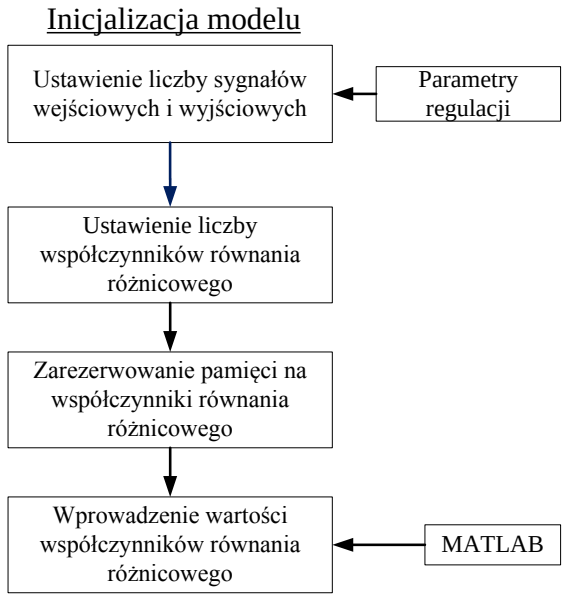
Rys. 5.15. Algorytm działania warstwy programowej w NIOS II



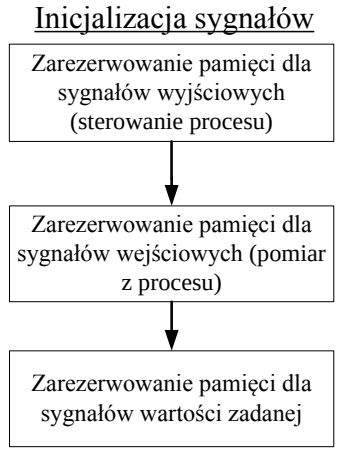
Rys. 5.16. Procedura pozyskania parametrów dla algorytmów analitycznych w wersji oszczędnej



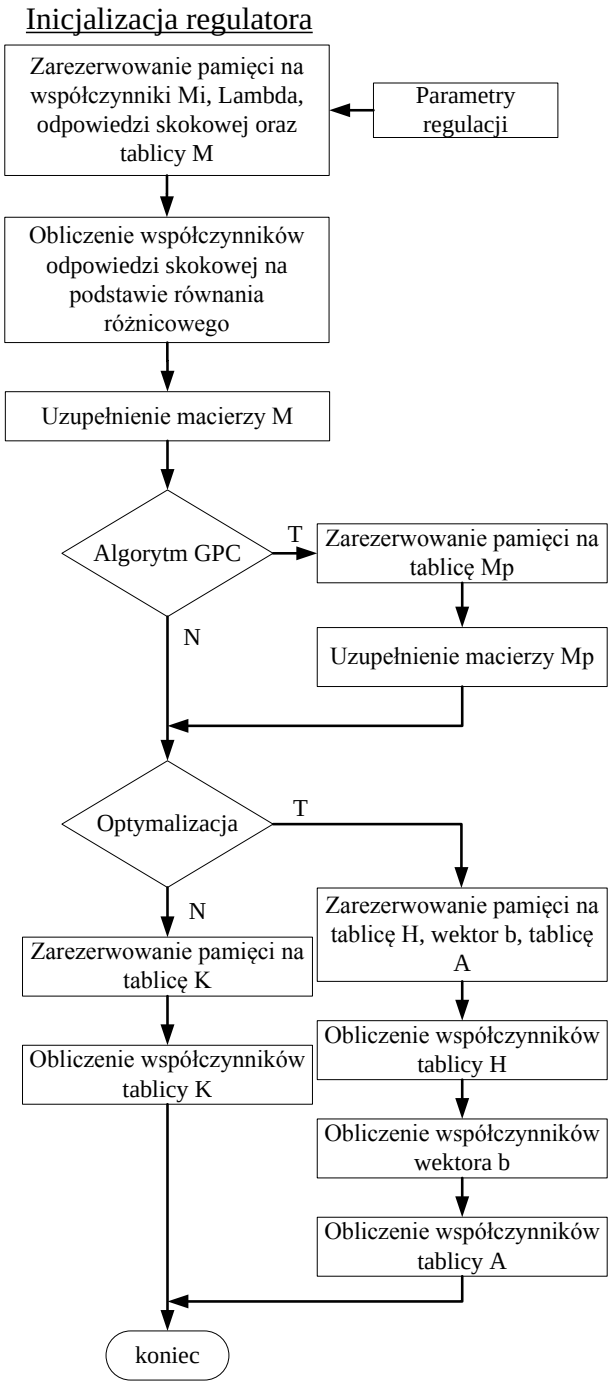
Rys. 5.17. Zadania realizowane przez procesor NIOS II w analitycznym algorytmie DMC w wersji oszczędnej



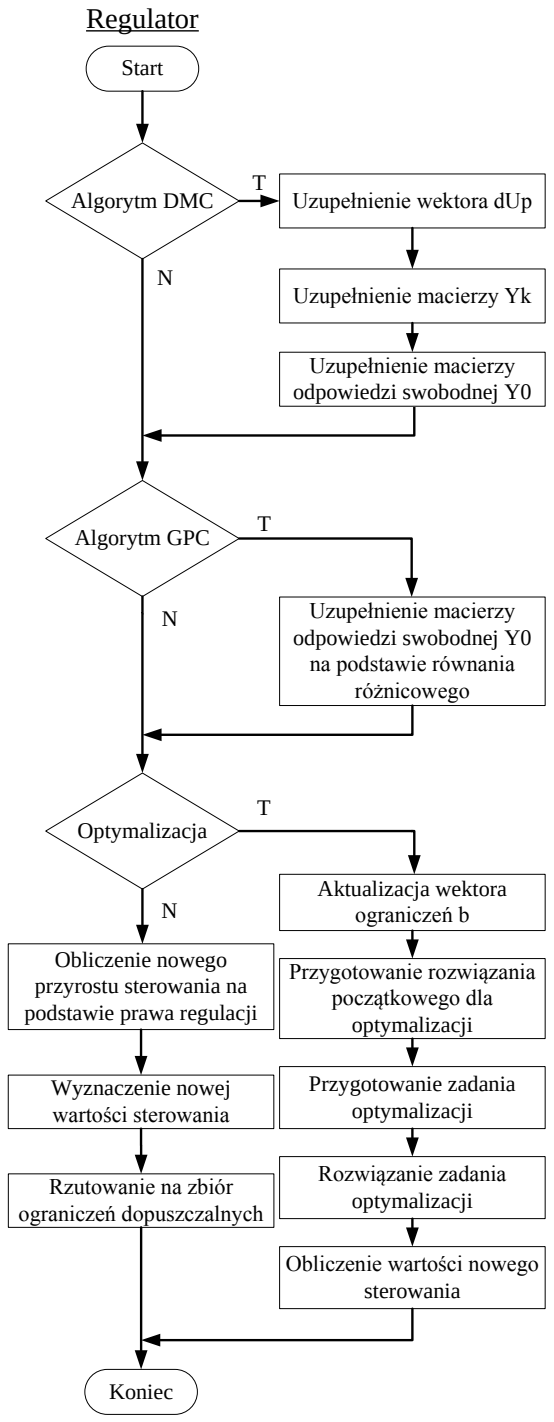
Rys. 5.18. Funkcja inicjalizacji modelu w algorytmach analitycznych (w wersji pełnej) i numerycznych



Rys. 5.19. Funkcja inicjalizacji sygnałów w algorytmach analitycznych (w wersji pełnej) i numerycznych



Rys. 5.20. Funkcja inicjalizacji regulatora w algorytmach analitycznych (w wersji pełnej) i numerycznych



Rys. 5.21. Funkcja regulatora w algorytmach analitycznych (w wersji pełnej) i numerycznych

Rozdział 6

System do regulacji predykcyjnej oparty na strukturze FPGA: warstwa sprzętowa

Bardzo ważnym etapem syntezy całego systemu sterującego jest opracowanie platformy sprzętowej, umożliwiającej implementację programową algorytmów regulacji predykcyjnej. W niniejszym rozdziale opisano wybraną płytę testową z układem FPGA, a także metodykę projektowania oraz szczegóły dotyczące wykonania i uruchomienia płyty interfejsów w standardzie przemysłowym, umożliwiającą podłączenie jednostki obliczeniowej FPGA do procesu.

6.1. Płyta FPGA

W ramach Grantu Dziekańskiego realizowanego przez autora w 2017 r. zakupiono płytę testową z układem FPGA firmy Intel serii Cyclone IV, która została pokazana na rys. 6.1. Układ posiada 150 tysięcy komórek logicznych. Dzięki dużej skali integracji i szybkości układ bardzo dobrze spełnia swoją rolę w obsłudze równoległych procesów. Ma to istotne znaczenie w sytuacji, kiedy wymagane są czasy próbkowania rzędu pojedynczych milisekund, i/lub pojedyncza struktura FPGA jest wykorzystana do implementacji wielu algorytmów regulacji. Płyta wyposażona jest w wiele interfejsów, dzięki czemu można realizować różne rodzaje połączeń i wykorzystywać różne urządzenia zewnętrzne. W celu szybkiej diagnostyki można użyć wbudowanego wyświetlacza LCD, wyświetlaczy siedmiosegmentowych lub standardowych diod LED. Interakcja z użytkownikem możliwa jest dzięki przyciskom i przełącznikom. Dostępny jest również port Ethernet, np. do połączenia z komputerem PC. Na płycie ewaluacyjnej znajduje się również minikomputer przemysłowy oparty na procesorze Intel Atom. Dostępne są wszystkie podstawowe interfejsy potrzebne przy tego rodzaju komputerach. Istotną zaletą jest również połączenie wewnętrzne między magistralą PCI Express minikomputera a portami układu FPGA. Dzięki temu możliwa jest szybka wymiana danych między dwoma

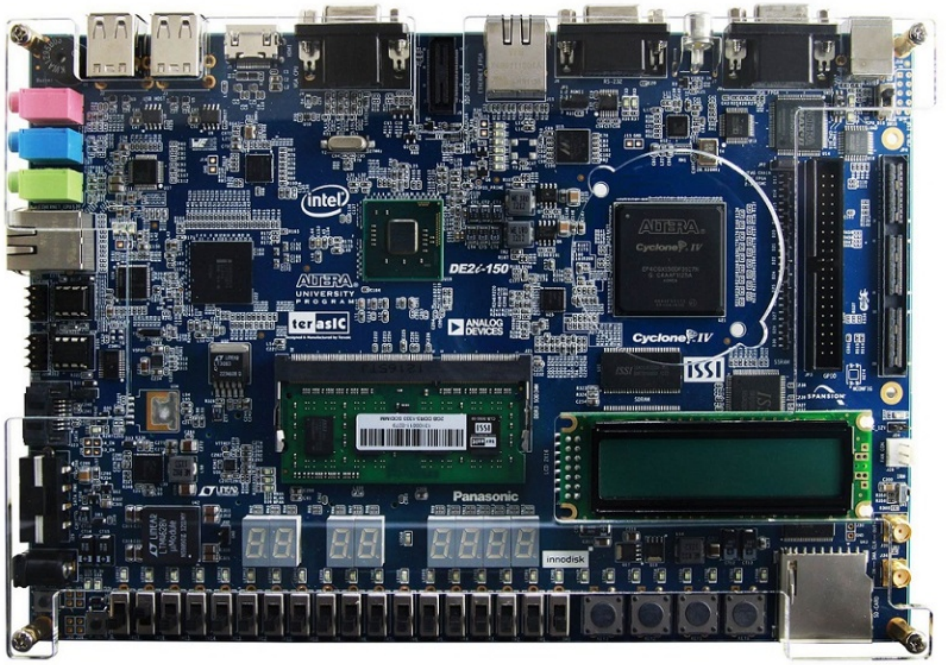
urządzeniami. Producent przygotował zestaw odpowiednich funkcji, aby zrealizować powyższą komunikację. Bardzo istotne z punktu widzenia wyboru docelowej platformy była możliwość podłączenia dodatkowych wejść i wyjść do platformy interfejsów sprzętowych. Płyta FPGA posiada dwa złącza: komputerowe złącze 40 pinowe oraz dedykowane złącze sygnałów różnicowych. Ponadto płyta ewaulacyjna wyposażona jest w następujące interfejsy podłączone do FPGA:

— wyjście VGA
— wyjście TV S-Video
— gigabitowy port Ethernet
— RS-232
— port odbiorczy podczerwieni
— dwa wyjścia SMA

Wybór płyty był również podyktowany czynnikami ekonomicznymi. Dostępne możliwości samego układu FPGA jak i interfejsów w stosunku do ceny są bardzo korzystne. Dostępne są oczywiście układy FPGA o dużo większej liczbie bramek logicznych, ale idea działania jest identyczna. Ponadto dzięki koncepcji języka HDL projekt zrealizowany dla jednego konkretnego układu FPGA bez problemu można przenieść na układ bardziej rozbudowany.

6.2. Projekt platformy interfejsów

Wybrana płyta FPGA udostępnia uniwersalne złącza, do których można podłączyć własne urządzenia. Dopuszczalne poziomy napięć sygnałów mieszczą się między 0 a 3,3 V. Taki standard jest oczywiście całkowicie niepraktyczny z puntu widzenia wykorzystania w sterowaniu procesów przemysłowych. Może to być bardzo często dużym problemem, podczas gdy np. w sterownikach PLC interfejsy są gotowe i spełniają standardy przemysłowe. Przygotowane opracowanie może służyć jako inspiracja dla projektantów do zwiększenia implementacji układów FPGA w przemyśle. Aby rozwiązać ten problem zaprojektowano i wykonano płytę interfejsów dedykowaną dla rozważanego układu FPGA. Każde wejście i wyjście jest izolowane galwanicznie przy pomocy układów ADUM, zapewniających izolację magnetyczną. Zabezpiecza to porty układu FPGA przed uszkodzeniem nieprawidłowym napięciem. Przygotowane zostały następujące interfejsy, przy czym główny nacisk został położony na standardy przemysłowe:



Rys. 6.1. Płyta testowa FPGA

1. analogowy wejściowy ±10 V: sygnały z czujników, pomiar prędkości obrotowej serwo lub falownika,
2. analogowy wyjściowy ±10 V: sterowanie prędkością serwo lub falownika, zawory proporcjonalne,
3. Cyfrowy wejściowy +24 V: sygnały binarne, czujniki, enkodery inkrementalne,
4. Cyfrowy wejściowy różnicowy A, /A, B, /B: enkodery inkrementalne z serwo,
5. Cyfrowy wejściowy +3,3 V: inne zestawy ewaluacyjne,
6. Cyfrowy wyjściowy +24 V: PWM, wyjścia binarne, sterowanie silników krokowych lub serwo.
7. Cyfrowy wyjściowy +3,3 V: inne zestawy ewaluacyjne.

Na rys. 6.2 przedstawiono schemat ideowy całej płyty interfejsów. Widoczne jest złącze IDC z numerami pinów układu FPGA oraz poszczególne układy wraz z odpowiednimi sygnałami. Od strony płyty FPGA doprowadzone jest również zasilanie +5 V oraz +3,3 V, które służy do zasilenia układów izolacji galwanicznej.

Na rys. 6.3 przedstawiono schemat zasilacza, którego głównym zadaniem jest dostarczenie wszystkich wymaganych napięć po stronie izolowanej. Główne wejście za-

silania wymaga napięcia 24 V. Dostępny jest stabilizator 10 V, stabilizator 5 V, stabilizator 3,3 V dla układów analogowych, stabilizator 3,3 V dla układów cyfrowych oraz stabilizator 2.5 V jako referencja dla przetworników analogowo-cyfrowych. Dodatkowo dostępny jest moduł przetwornicy napięcia wytwarzający napięcia +10 V i −10 V dla wyjść sygnałów analogowych. Każdy ze stabilizatorów jest obudowany baterią kondensatorów, zapewniających odpowiednie tłumienie tętnień zasilania. Zasilacz wyposażony jest w diody świecące potwierdzające jego działanie.

Na rys. 6.4 przedstawiono schemat pojedynczego wejścia analogowego ±10 V, które zrealizowano jako dzielnik rezystancyjny z możliwością korekcji. Napięcie na wyjściu może zmieniać się w zakresie od 0 do 2.5 V, które jest wymagane przez przetwornik analogowo-cyfrowy. Nad zabezpieczeniem dopuszczalnego zakresu napięcia czuwają diody Schottky'ego. Sygnał wejściowy można podać na złącze śrubowe ARK lub dwa goldpiny. Wyjście tego prostego układu podane zostało na wtórnik zbudowany na wzmacniaczu operacyjnym MCP6004. Stąd też dioda zabezpieczająca jest doprowadzona do zasilania +3,3V_EA.

Na rys. 6.5 przedstawiono dalszą część toru przetwarzania analogowo cyfrowego. Każdy dzielnik napięcia wejściowego jest podłączony do wzmacniacza operacyjnego, którego zadaniem jest realizacja wysokiej impedancji wejścia, a także dodatkowa separacja wejścia fizycznego od przetwornika AC. Przetwornik jest wyposażony w 8 kanałów, z czego 4 są podłączone na stałe do wyjść wzmacniaczy operacyjnych. Cztery pozostałe wejścia można połączyć przewodami zgodnie z potrzebami użytkownika, np. mogą to być pozostałe wzmacniacze operacyjne lub inne zewnętrzne układy dedykowane do aplikacji. Przetwornik AC ADS8344EB posiada wewnętrzny multiplekser, który wybiera wejście do przetwarzania sygnału metodą sukcesywnej aproksymacji SAR. Wynik dostępny jest na magistrali cyfrowej, która jest sterowana z układu FPGA. Do przekazywania sygnałów wykorzystano układ izolacji galwanicznej ADUM163N0BRZ. Układ ten posiada wewnętrzne układy kodująco-dekodujące, dzięki czemu jest w stanie przenosić zarówno składową zmienną jak i składową stałą.

Na rys. 6.6 przedstawiono fragment toru przetwarzania cyfrowo analogowego. Widoczna jest separacja galwaniczna w postaci układu ADUM160N0, który przekazuje sygnały do 4 niezależnych przetworników CA. Przetworniki wymagają tylko sygnału zegarowego SCLK, sygnału danych DIN oraz sygnału wyboru SYNC. Sterowaniem odpowiednimi stanami sygnałów zajmuje się dedykowany temu celowi układ logiczny zaimplementowany w strukturze FPGA.

Na rys. 6.7 przedstawiono dalszy tor przetwarzania cyfrowo analogowego, gdzie widoczny jest sam przetwornik sygnału DAC8560 oraz wzmacniacz operacyjny OPA277U, który konwertuje napięcie analogowe z zakresu 0/3,3 V do $-10/+10$ V.

Na rys. 6.8 przedstawiono schemat pojedynczego wejścia binarnego dla sygnału 24 V. Układ składa się z prostego dzielnika napięcia. Dodatkowo uwzględniona została dioda świecąca dla szybkiej weryfikacji aktualnego stanu wejścia. Po dzielniku występuje również zabezpieczenie zakresu napięcia diodami Schottky'iego.

Na rys. 6.9 przedstawiono dalszy schemat toru sygnału binarnego 24V. Odpowiednio przygotowany sygnał trafia do układu izolacji galwanicznej ADUM160N. Analogicznie zostało rozwiązane wejście binarne 3,3V.

Rys. 6.9: wstawić plik o większej rozdzielczości.

Na rys. 6.10 przedstawiono schemat wyjścia binarnego 24 V. Zapewniono w tym rozwiązaniu możliwość ręcznego sterowania wyjścia przyciskiem PB1. Jest ono wyjście, ponieważ istnieje możliwość pracy wyjścia w dwóch wariantach logiki. Tranzystor Q2 pozwala na sterowanie sygnałem przez zwieranie do masy układu, tranzystor Q1 pozwala na sterowanie sygnałem przez zwieranie do zasilania układu. Rodzaj sterowania można wybrać przez zworki goldpin. Ponadto, wyjście jest wyposażone w diodę świecącą, która sygnalizuje jego aktualny stan. Występują również diody zabezpieczające przeciw przepięciom, które mogą wystąpić przy sterowaniu elementów wykonawczych o dużej indukcyjności np. cewki zaworów.

Na rys. 6.11 przedstawiono schemat układu wejść różnicowych. Dostępne są 4 kanały różnicowe, które przetwarzane są na sygnały cyfrowe przez układ AM26LS32ACD. Następnie te sygnały trafiają do układu izolacji galwanicznej ADUM160N0. Dodatkowe diody świecące pozwalają na szybką weryfikację poprawności podłączenia. Dla rozważanego przypadku możliwe jest podłączenie dwóch silników serwo z sygnałami różnicowymi z wyjściami A oraz B. Dzięki temu można zliczać położenie aktualne silnika lub niezależnego enkodera inkrementalnego.

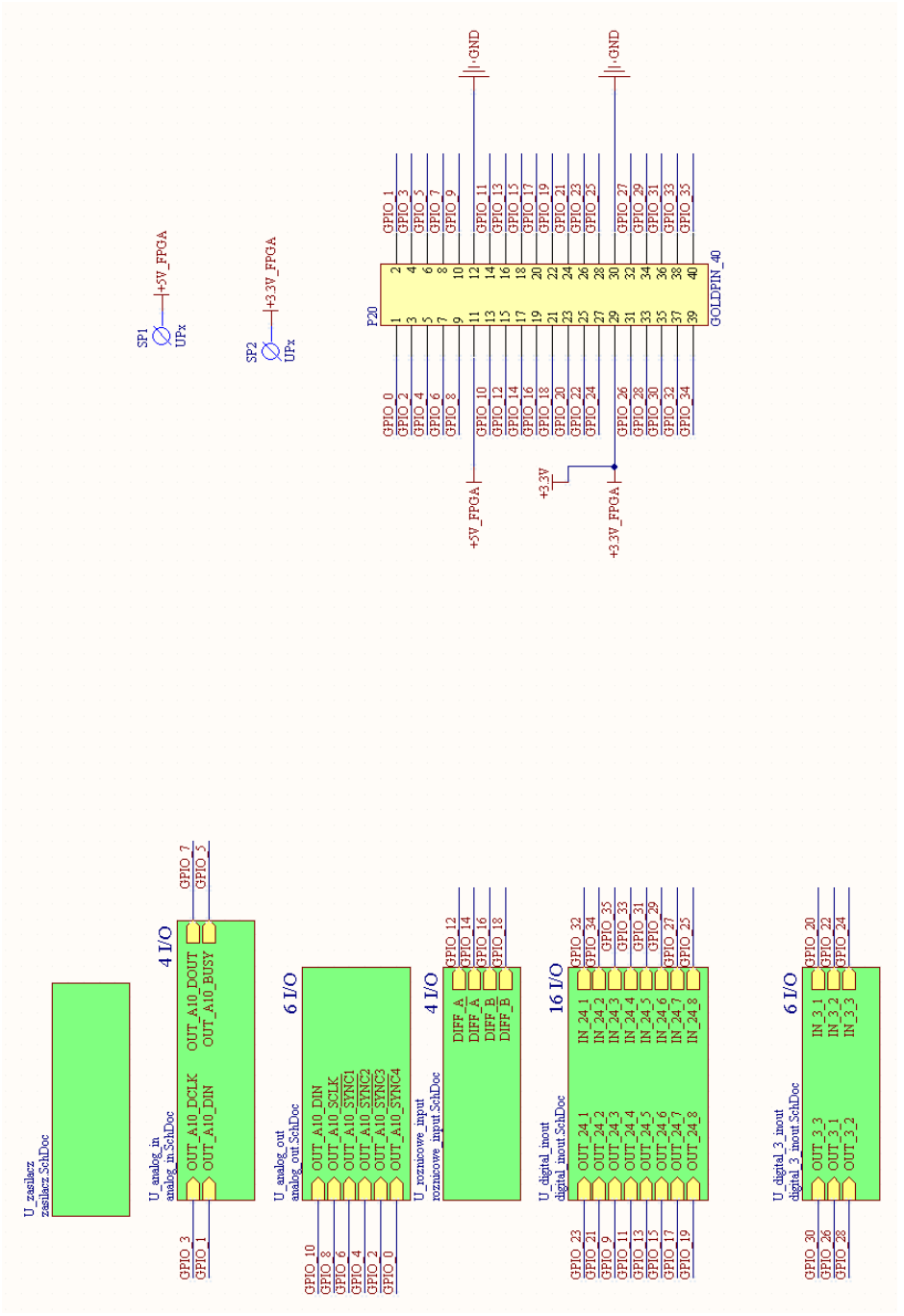
Na rys. 6.12 przedstawiono model zaprojektowanej płyty PCB. W centralnej części znajduje się złącze IDC do połączenia z główną płytą FPGA. Wokół złącza znajdują się układy ADUM izolacji galwanicznej. W dolnej części znajduje się zasilacz dla sygnałów zewnętrznych. Po lewej stronie znajdują się układy wyjściowe 3,3 V, powyżej na prawo znajdują się wejścia 3,3 V, w dalszej kolejności wejścia różnicowe, następnie układy wyjść analogowych ± 10 V, po prawej stronie od góry znajdują się wejścia binarne 24 V, poniżej wejścia analogowe ± 10 V. Pomiędzy złączem IDC a zasilaczem znajdują się wyjścia binarne 24 V. Zaprojektowana płyta

jest dwustronna, gdzie dolna warstwa w głównej mierze stanowi płaszczyznę masy. Masa analogowa jest separowana od masy cyfrowej, aby zminimalizować zakłócenia.

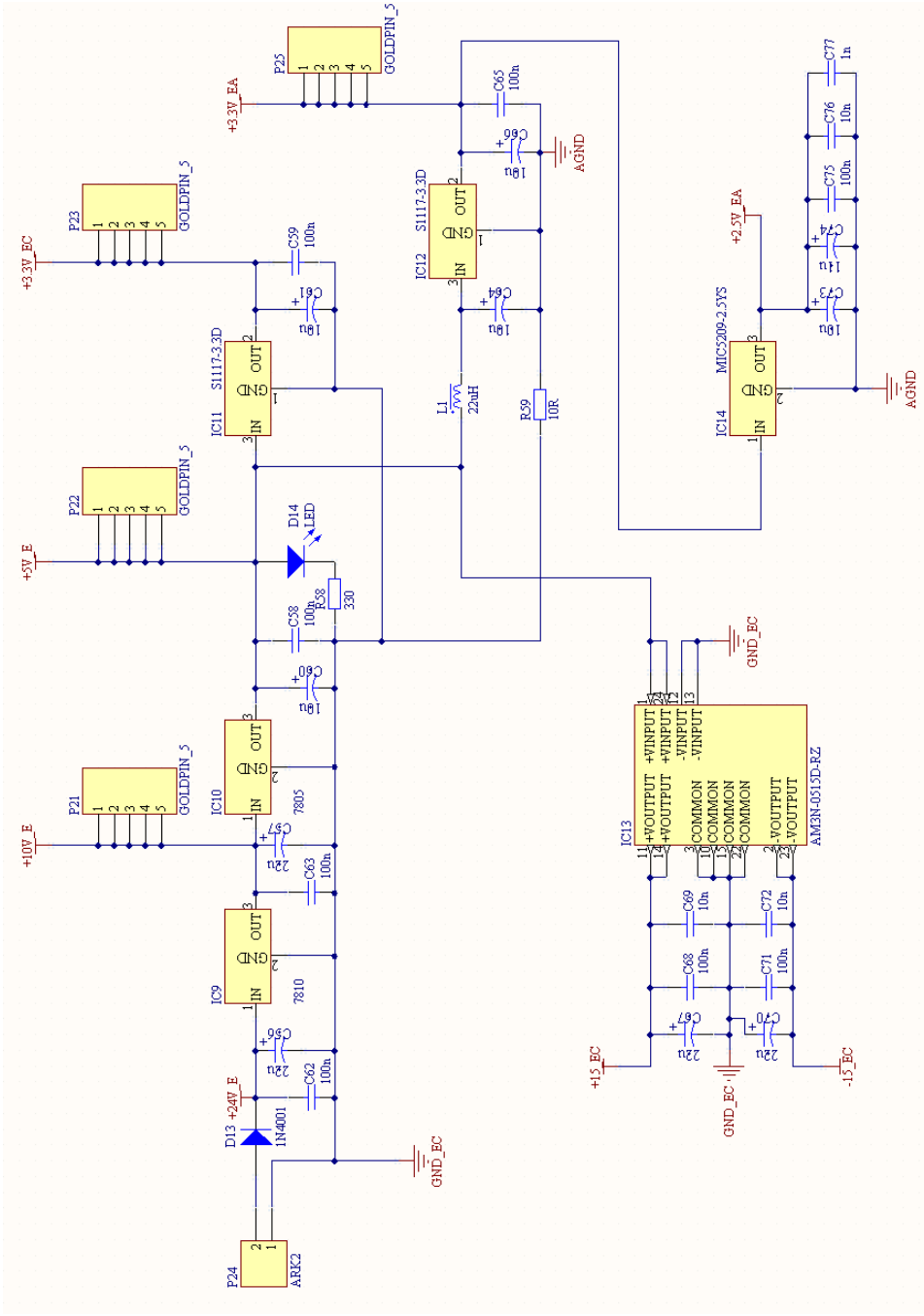
6.3. Wykonanie i uruchomienie platformy interfejsów

Z uwagi na wielkość zaprojektowanej płyty, gęstość upakowania elementów oraz bardzo małe elementy scalone wymagające do lutowania soldermaski, została ona zlecona do wykonania. Zakupiono wszystkie podzespoły i rozpoczęto proces montażu. W pierwszej kolejności zlutowany został zasilacz i zweryfikowane zostały wszystkie potrzebne napięcia. Gdy stwierdzono ich prawidłowe działanie, zlutowano część połączeniową z płytą FPGA: złącze, kondensatory, rezystory oraz układy ADUM. Przetestowano działanie wejść i wyjść układów ADUM, w szczególności zweryfikowano poprawność przenoszenia składowej stałej. Maksymalna częstotliwość przenoszona wynosi około 100 kHz. Kolejnym etapem było zlutowanie wejść i wyjść w standardzie 3,3 V oraz ich przetestowanie. Możliwe już było podłączenie płyty do komputera PC przez konwerter USB-RS232 i przetestowanie możliwości komunikacji w obu kierunkach. Dodatkowo, przetestowano komunikację ze stanowiskiem termicznym, w którym komunikacja odbywa się zgodnie z protokołem RS485. Ostatnimi do uruchomienia sygnałami binarnymi zostały wejścia i wyjścia 24 V. Zlutowano odpowiednie obwody i podłączono kilka sygnałów testowych wejściowych. Do wyjść z kolei podłączono stanowisko świetlne, gdzie możliwe było przetestowanie świecenia diod przy sterowaniu sygnałami PWM. Kolejno zlutowano obwody sygnałów analogowych wejściowych i wyjściowych. Podłączono sygnał pochodzący ze stanowiska świetlnego i przetestowano działanie przetworników wejściowych. Przetworniki wyjściowe zostały zweryfikowane multimetrem. Sygnały różnicowe nie były w obecnej chwili wymagane więc pominięto ich montaż i test.

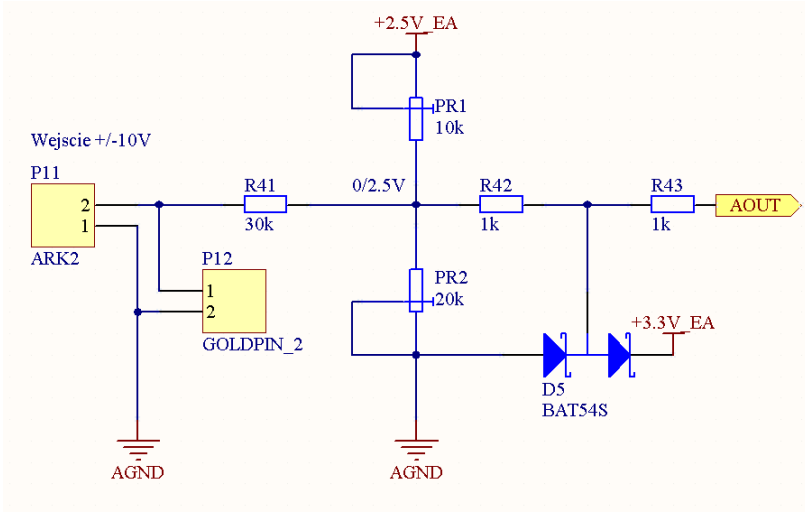
Na rys. 6.13 przedstawiono finalnie wykonaną płytę interfejsów przemysłowych podłączoną do płyty FPGA. Wykorzystano w tym celu kabel do dysków IDE. Pozostałe połączenia z płyty interfejsów do stanowisk wykonane były przewodami do płytek stykowych. Po zakończonych testach funkcjonalnych płyty interfejsów można było rozpocząć prace nad docelowymi eksperymentami z algorytmami regulacji predykcyjnej.



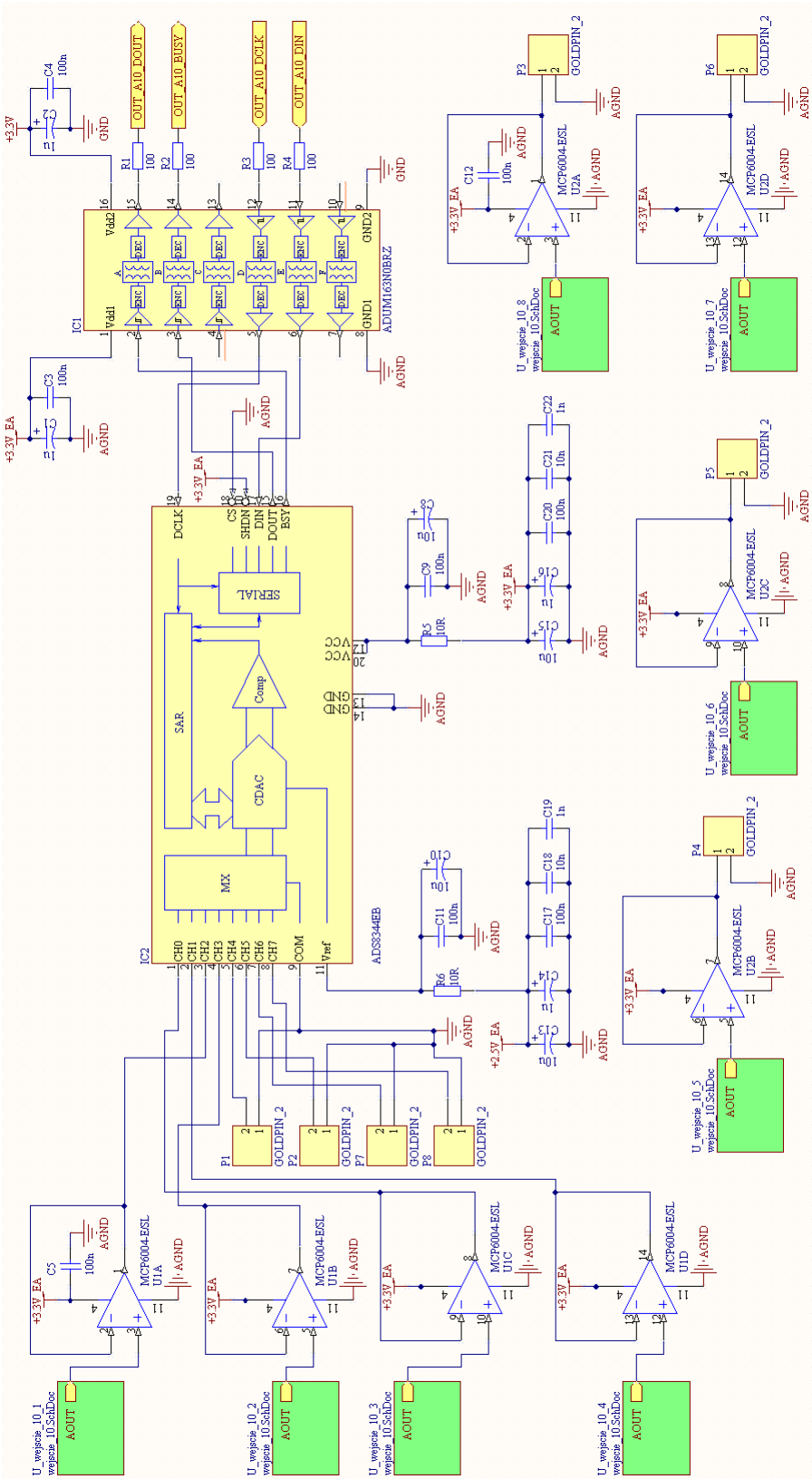
Rys. 6.2. Schemat ideowy wszystkich układów i złącza IDC



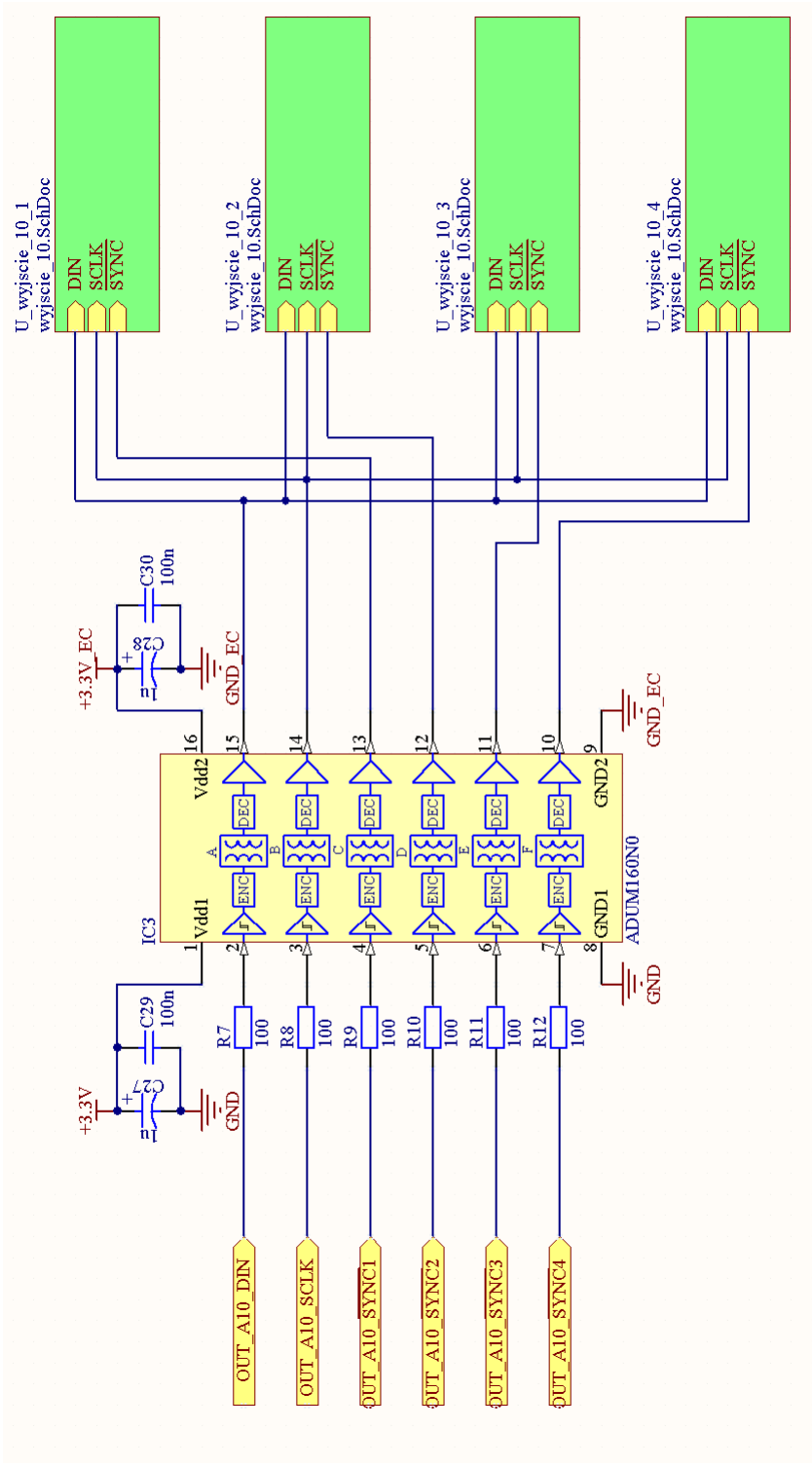
Rys. 6.3. Schemat zasilacza



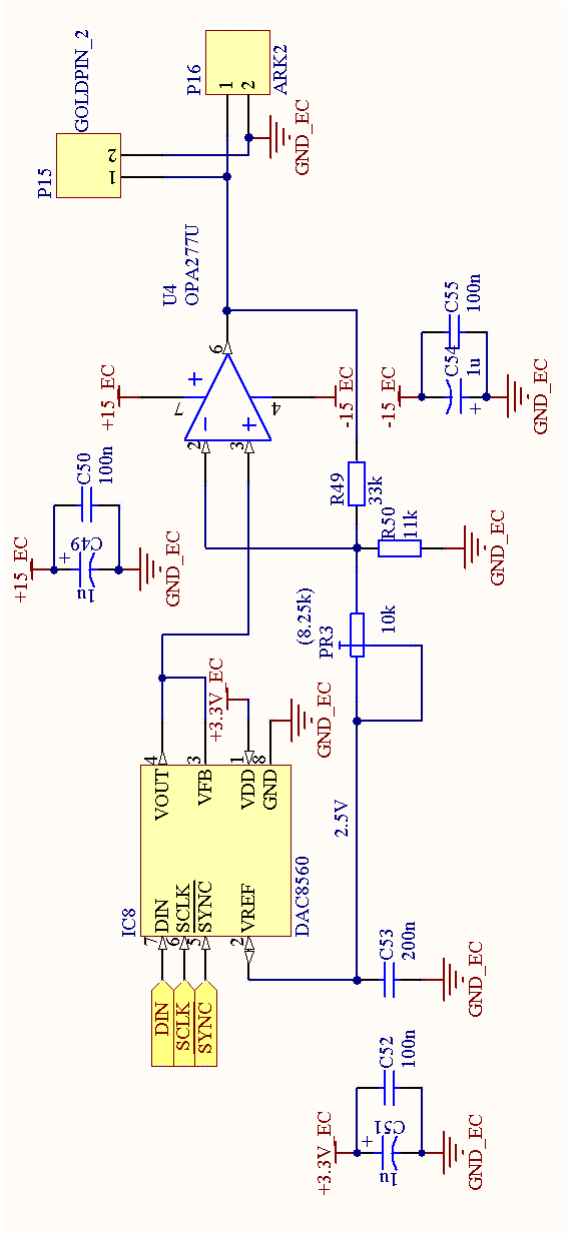
Rys. 6.4. Schemat pojedynczego wejścia analogowego ±10 V



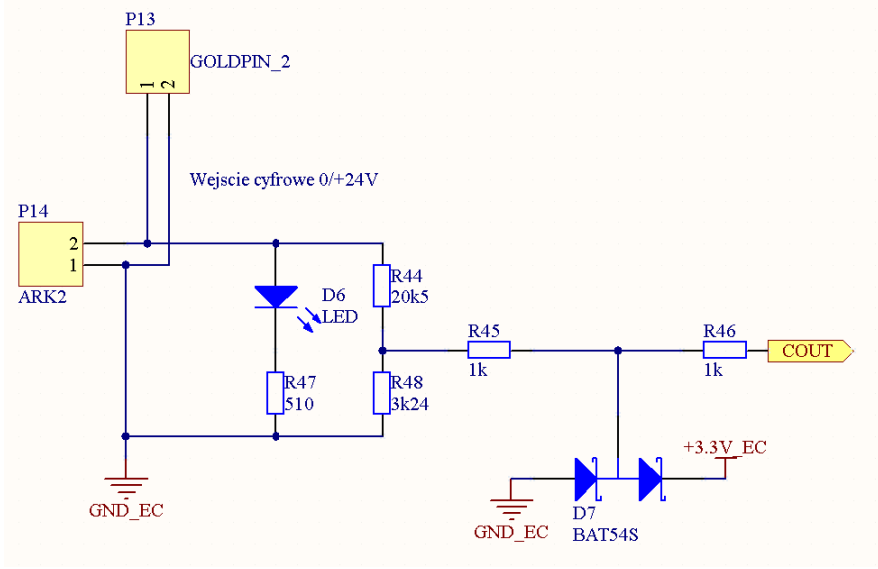
Rys. 6.5. Schemat układów wejść analogowych



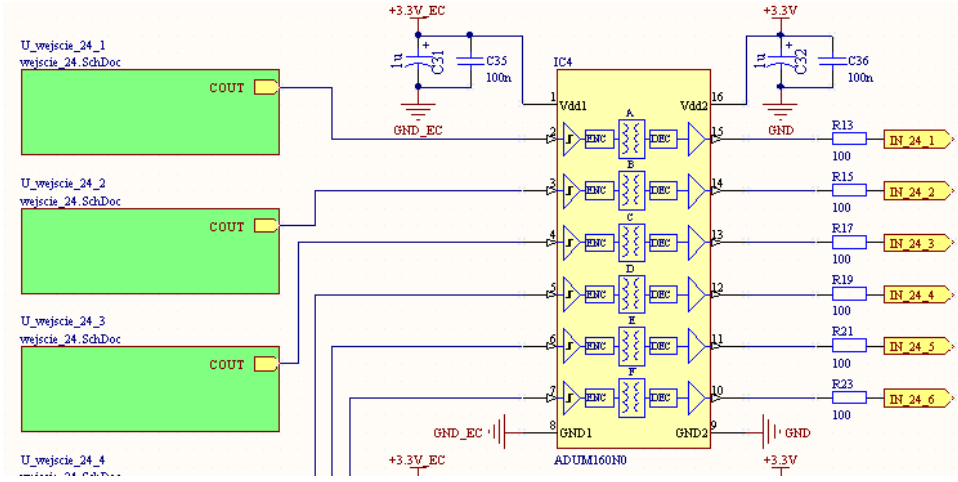
Rys. 6.6. Schemat układów wyjść analogowych



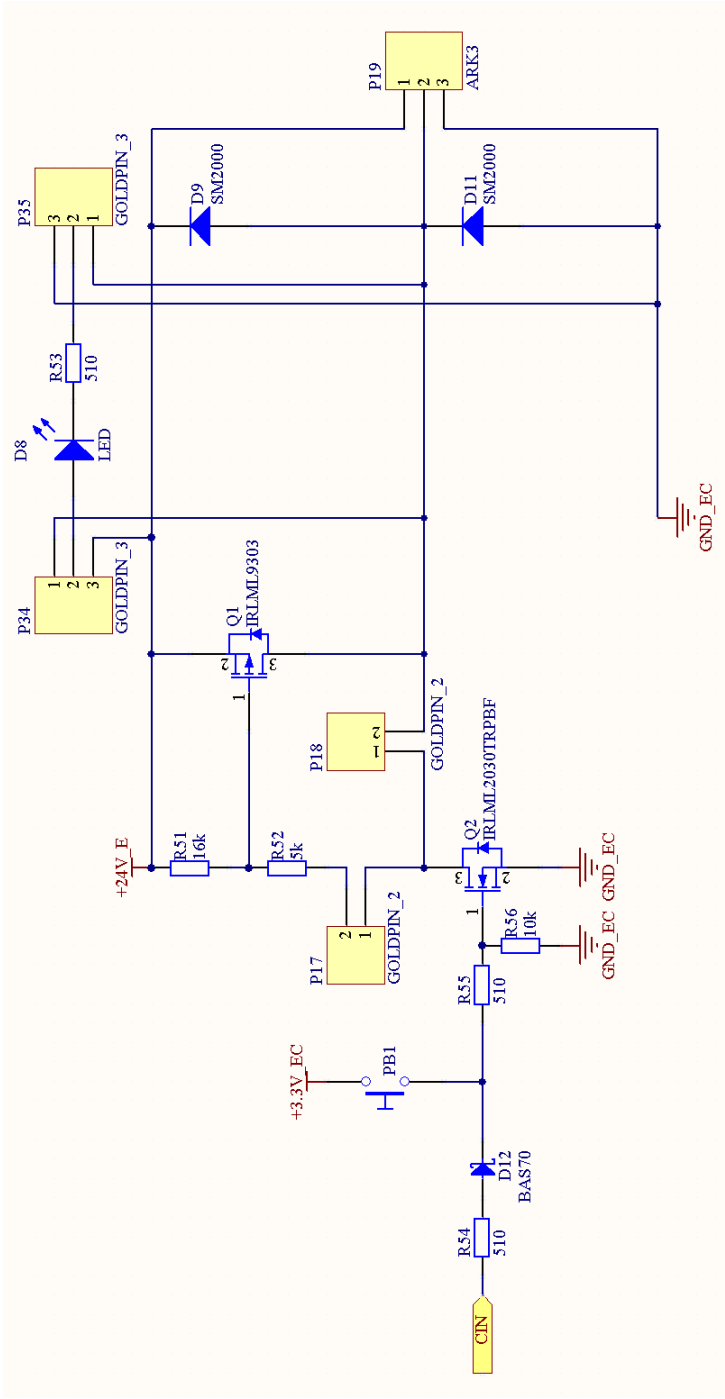
Rys. 6.7. Schemat pojedynczego wyjścia analogowego



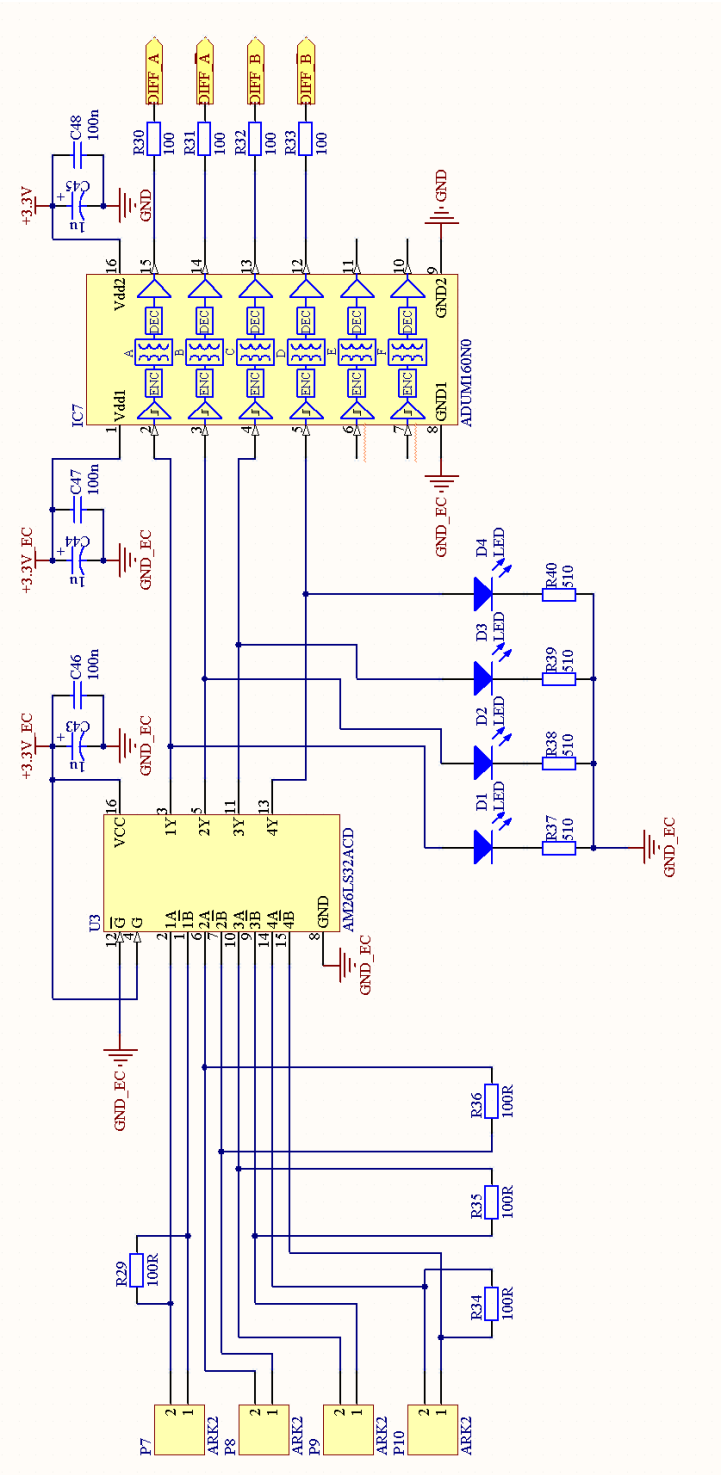
Rys. 6.8. Schemat układu wejścia binarnego 24 V



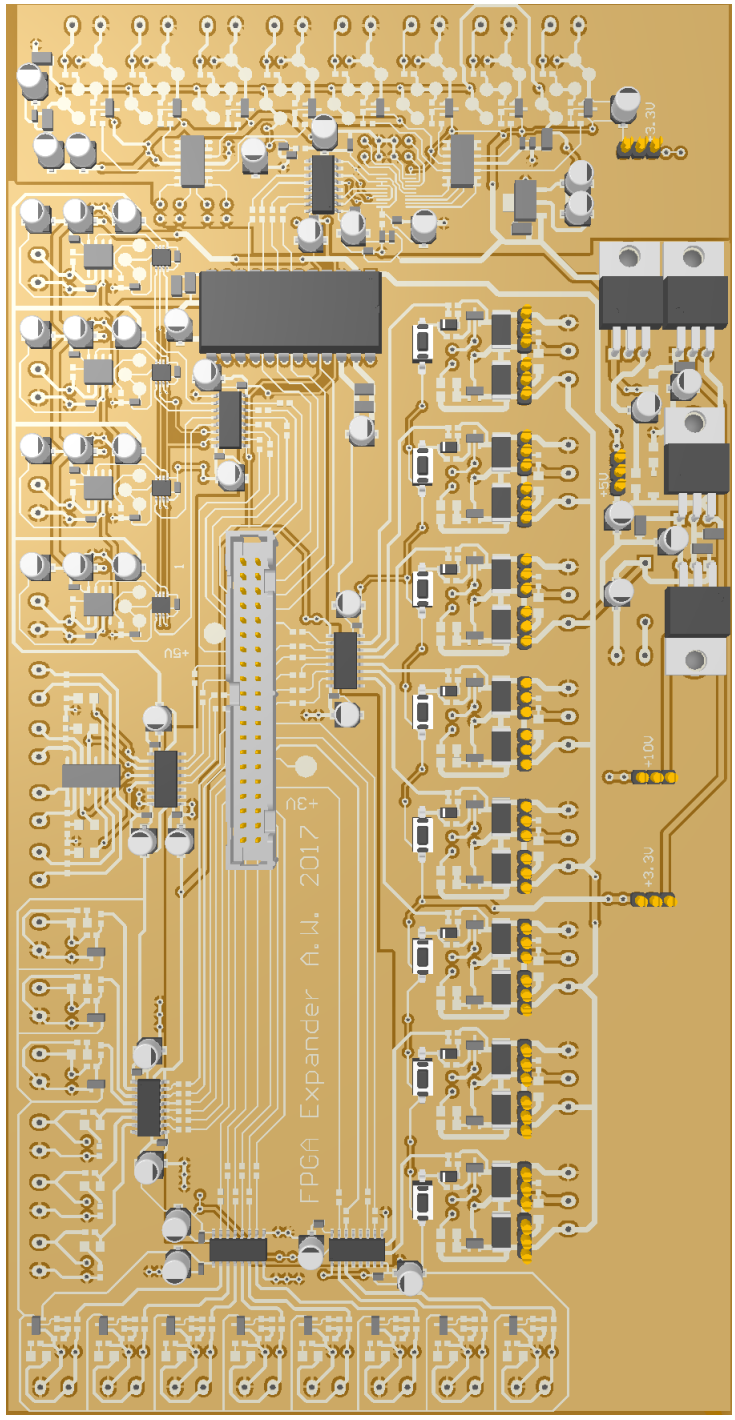
Rys. 6.9. Schemat układu wejścia binarnego 3,3 V



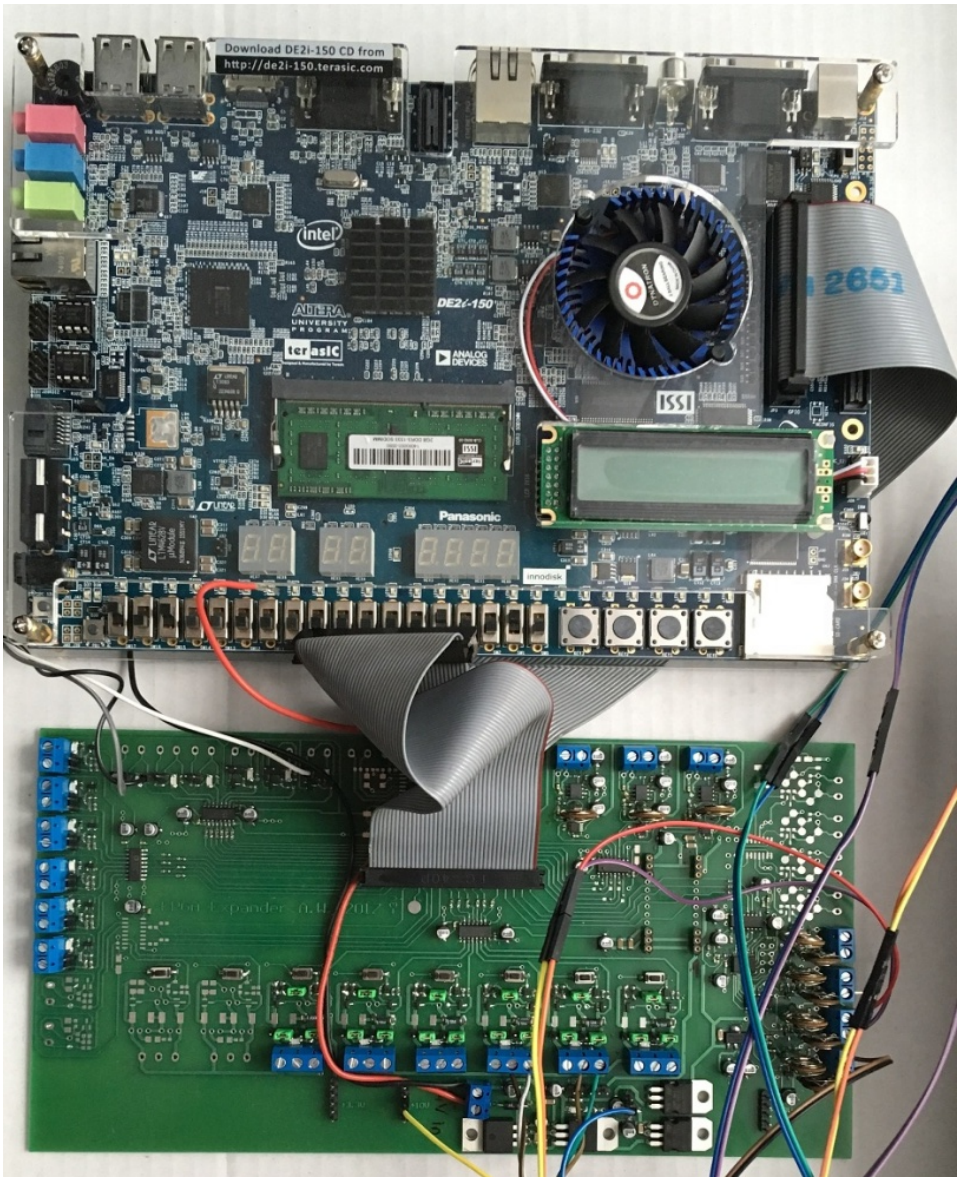
Rys. 6.10. Schemat układu wyjścia binarnego 24 V



Rys. 6.11. Schemat układu wejścia różnicowego



Rys. 6.12. Model 3D płyty drukowanej



Rys. 6.13. Płyta FPGA (na górze) oraz płyta interfejsów przemysłowych (na dole)

Rozdział 7

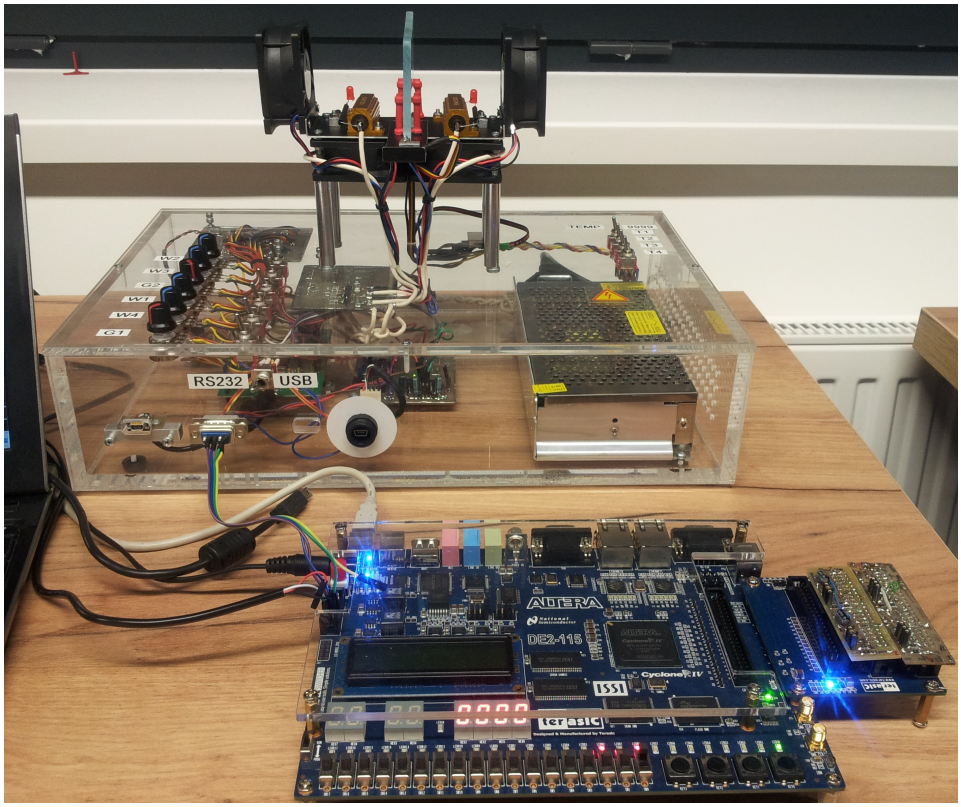
Przygotowanie stanowisk testowych

W celu weryfikacji działania całego systemu sprzętowo-programowego do regulacji predykcyjnej, wykonano szereg badań eksperymentalnych. W badaniach wykorzystano trzy stanowiska laboratoryjne: stanowisko grzejąco-chłodzące, stanowisko świetlne oraz serwomechanizm, przy czym dwa pierwsze z nich zostały zaprojektowane i skonstruowane przez autora rozprawy. Każde z rozważanych stanowisk posiada charakterystyczne cechy, wyróżniające je na tle innych. Możliwa jest regulacja procesów jednowymiarowych, wielowymiarowych, wolnych (wymagających regulacji o długich okresach próbkowania), szybkie (wymagających krótkich okresów próbkowania). Istnieje możliwość konfiguracji parametrów procesów, badania pracy przy zmianach wartości zadanej (zadanych) oraz reakcji na zakłócenie (zakłócenia). Dzięki temu wymienionym cechom procesów, istnieje możliwość badania efektywności pracy algorytmów regulacji predykcyjnej w różnych warunkach. W dalszej części rozdziału opisano omawiane procesy.

7.1. Stanowisko grzejąco-chłodzące

Stanowisko grzejąco-chłodzące zostało opracowane i wykonane w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej [115, 116]. Jest wykorzystywane w zajęciach dydaktycznych, a także w pracach badawczych. Charakteryzuje się ono dość wolną dynamiką, co umożliwia zastosowanie algorytmów regulacji o dość długim czasie próbkowania (rzędu kilku sekund). Bardzo ważną cechą procesu jest możliwość konfiguracji: możliwe jest rozważenie procesu jednowymiarowego oraz wielu wersji procesu wielowymiarowego, maksymalnie sześć wejść i siedem wyjść, oczywiście możliwe są kombinacje pośrednie. Możliwe jest sterowanie dwoma grzałkami i czterema wentylatorami. Możliwy jest pomiar pięciu temperatur, pomiar prądu i napięcia pobieranego z zasilacza. Możliwe scenariu-

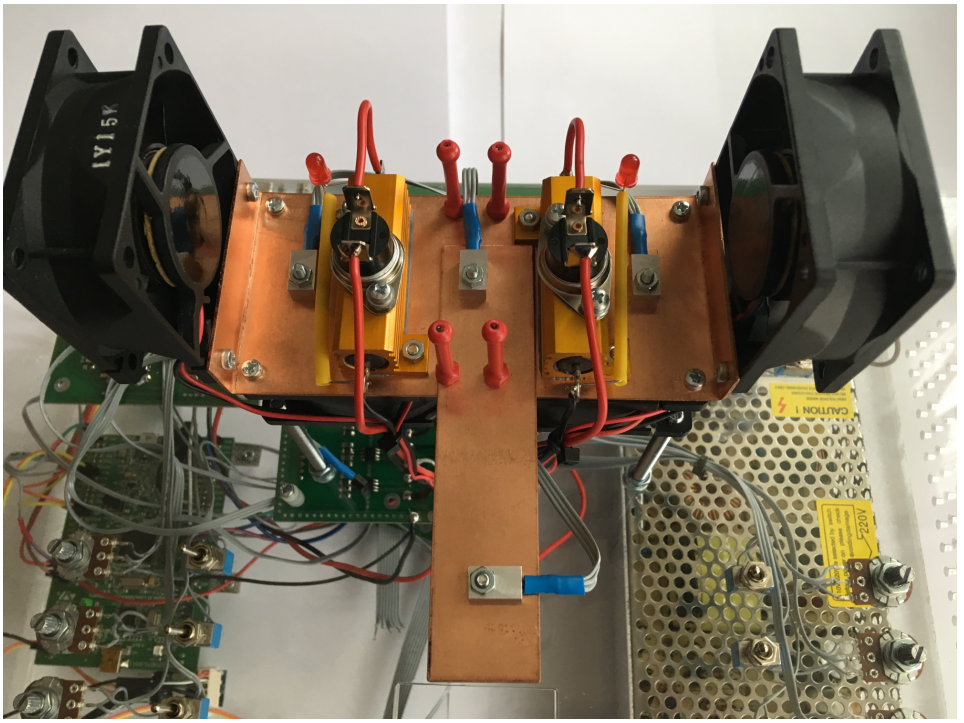
sze regulacji mogą obejmować, między innymi, optymalne sterowanie z punktu widzenia zużycia energii. Ponadto, możliwe jest testowanie działania algorytmów regulacji w sytuacji uszkodzeń elementów wykonawczych i pomiarowych (można symulować wystąpienie takich uszkodzeń). Testowa realizacja sprzętowa została przedstawiona na rys. 7.1, gdzie podłączone stanowisko grzejąco-chłodzące jest sterowane przy wykorzystaniu układu FPGA.



Rys. 7.1. Stanowisko grzejąco-chłodzące podłączone do płyty FPGA

Elementem przekazującym ciepło w obiekcie właściwym jest odpowiednio wyprofilowana blacha aluminiowa o grubości 1 mm, do której dołączone są rezystory mocy, wentylatory oraz czujniki temperatury. Dodatkowy pomost został wykonany z tego samego materiału i również jest grubości równej 1 mm. Na pomoście znajduje się czujnik temperatury TF. Takie jego umiejscowienie powoduje, że zmiana wartości temperatury pojawia się na tym czujniku z opóźnieniem względem pozostałych. Na rys. 7.2 przedstawiono zbliżenie głównej części obiektu. Jest ona

wykonana z miedzi. Widoczne są szczegóły konstrukcji takie jak rozmieszczenie czujników temperatur, rezystorów grzejnych oraz wentylatorów. W tle widać zamkniętą w obudowie elektronikę sterującą.



Rys. 7.2. Stanowisko grzejąco-chłodzące: główna część wykonana z miedzi

Na rys. 7.3 przedstawiony jest schemat omawianego obiektu. Stanowisko posiada 6 wejść (sygnałów sterujących), możliwy zakres sygnałów od 0 (0% mocy) do 1000 (100% mocy):

— wentylator FLU (lewy górny),
— wentylator FLB (lewy dolny),
— wentylator FRU (prawy górny),
— wentylator FRB (prawy dolny),
— grzałka HL (lewa),
— grzałka HR (prawa).

Sygnały wyjściowe są następujące:

— pomiary temperatury TL, TM, TR, TF, możliwe wartości od $-55.0°C$ do $+125°C$,

— pomiar temperatury otoczenia TA, możliwe wartości od $-55°C$ do $+125°C$,
— pomiar prądu C,
— pomiar napięcia V.

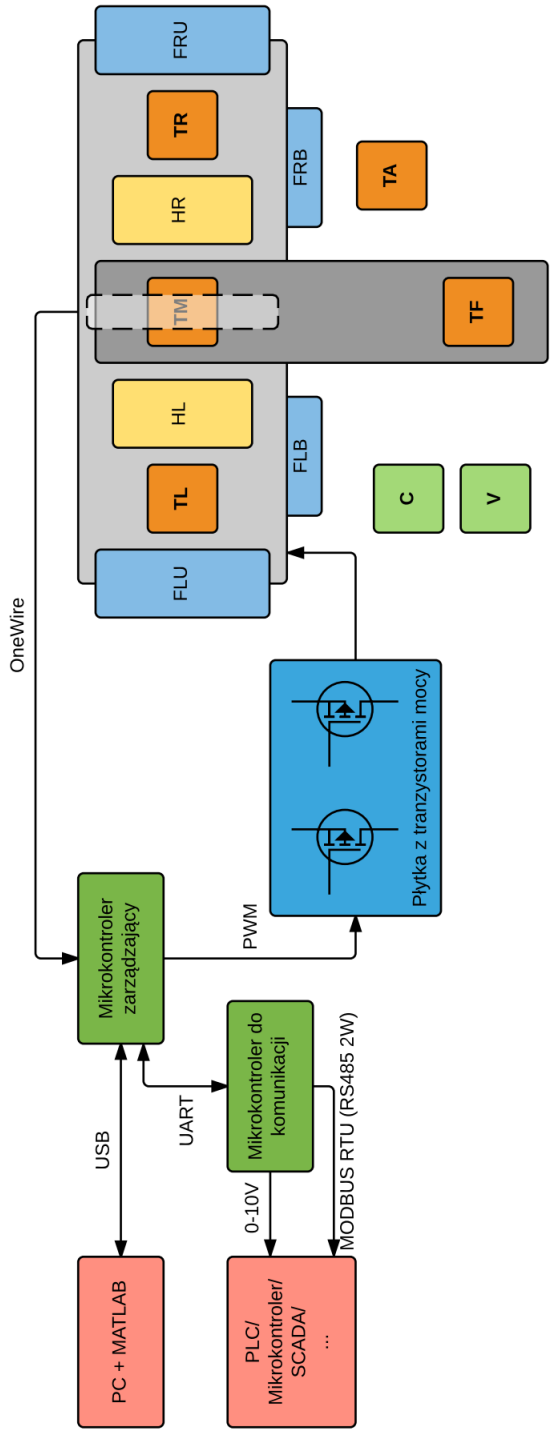
Elementy wykonawcze sterowane są sygnałem PWM, czujniki temperatury komunikują się z użyciem magistrali OneWire, natomiast pomiary prądu i napięcia realizowane są poprzez zastosowanie dedykowanych układów elektronicznych. Wszystkie wartości sygnałów sterujących wyrażone są w promilach (0 reprezentuje 0.0%, 1 reprezentuje 0.1%, 1000 reprezentuje 100.0%). Temperatury wyrażone są w setnych stopnia Celsjusza (0 reprezentuje $0.00°C$, 1 reprezentuje $0.01°C$, 9999 reprezentuje $99.99°C$. Prąd wyrażony jest w setnych ampera (0 reprezentuje 0.00 A, 1 reprezentuje 0.01 A, 1000 reprezentuje 10.00 A). Napięcie wyrażone jest w setnych wolta (0 reprezentuje 0.00 V, 1 reprezentuje 0.01 V, 1000 reprezentuje 10.00 V). Przekazywanie danych może następować również przy użyciu standardu napięciowego 0-10 V. Umiejscowienie poszczególnych sygnałów na złączach śrubowych opisane jest na naklejce, pokazanej na rys. 7.7. Warto zauważyć, że dla wartości sygnału sterowania 0 V element wykonawczy jest wyłączony, natomiast dla sygnału 10 V jest on włączony z pełną mocą. Temperatury należy przeskalować następująco: $t = \frac{v}{10V} 80°C + 10°C$, gdzie v jest to pomiar napięcia na odpowiednim złączu śrubowym, natomiast t jest odpowiadającą temu napięciu wartością temperatury. Dla pomiarów prądu 0 V oznacza 0 A, natomiast 10 V oznacza 12 A. Pomiar wartości napięcia z wykorzystaniem złączy śrubowych jest równoznaczny z pomiarem wartości napięcia bezpośrednio przy złączach zasilacza.

W stanowisku znajdują się jeszcze elementy odpowiedzialne za sterowanie oraz elementy służące do dostarczenia odpowiedniej ilości energii do elementów wykonawczych. Do elementów odpowiedzialnych za sterowanie zalicza się

— płytkę z tranzystorami mocy (sterowanie mocą dostarczaną do grzałek i wentylatorów),
— ramę procesu (z aluminiowej lub miedzianej blachy) na której zamontowane są rezystory mocy, wentylatory, czujniki temperatury i przegroda mechaniczna (wszystko to razem nazywane obiektem właściwym)
— płytkę zasilacza z pomiarem prądu oraz napięcia,
— zasilacz 24V.

Do elementów odpowiedzialne za sterowanie zalicza się:

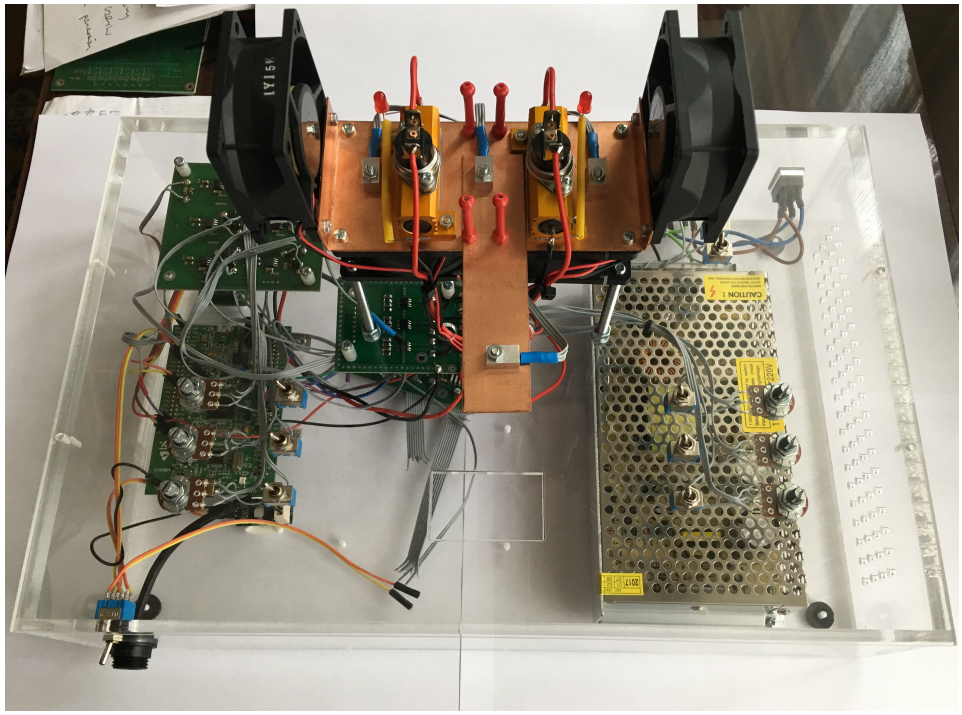
— płytki z mikrokontrolerem, którego zadaniem jest utrzymywanie komunikacji ze sterownikami (PLC, komputerem, FPGA, innym mikrokontrolerem)



Rys. 7.3. Stanowisko grzejąco-chłodzące: schemat ogólny

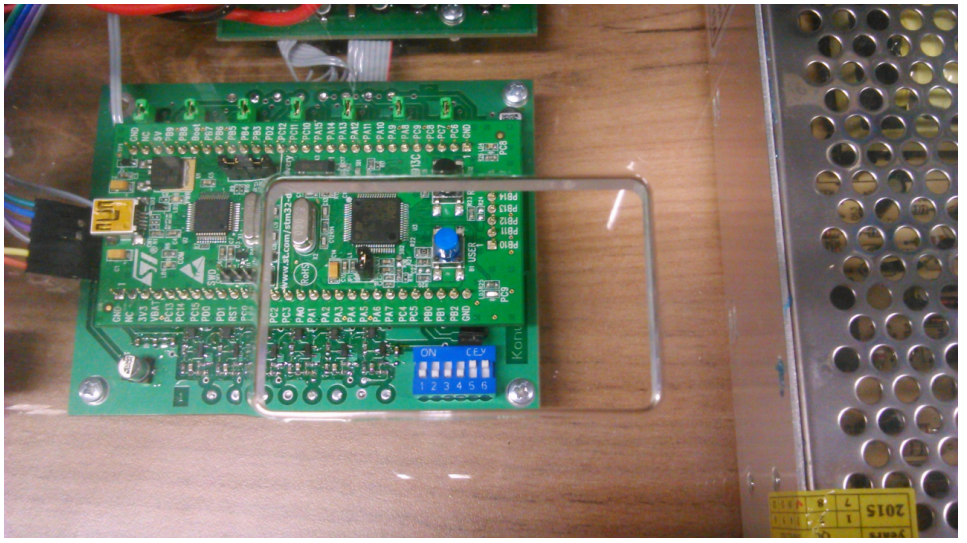
— komputer z dedykowanym programem, napisanym w języku MATLAB, do regulacji procesu. Poza komputerem, proces ten może być regulowany z użyciem sterownika PLC, mikrokontrolera, FPGA oraz systemu SCADA.

Jeżeli spojrzymy na stanowisko od frontu (rys. 7.4), to na jego górnej części znajdują się (wymieniając od lewej strony): pierwsza część panelu do sterowania ręcznego, obiekt właściwy, okienko dające dostęp do przełączników konfigurujących (rys. 7.5), druga część panelu do sterowania ręcznego, przełączniki do symulacji uszkodzeń czujników. Nad czujnikiem temperatury TM znajduje się bariera mechaniczna. Jest ona wykorzystana do oddzielenia od siebie dwóch strumieni powietrza od przeciwległych wentylatorów.



Rys. 7.4. Stanowisko grzejąco-chłodzące: widok z frontu

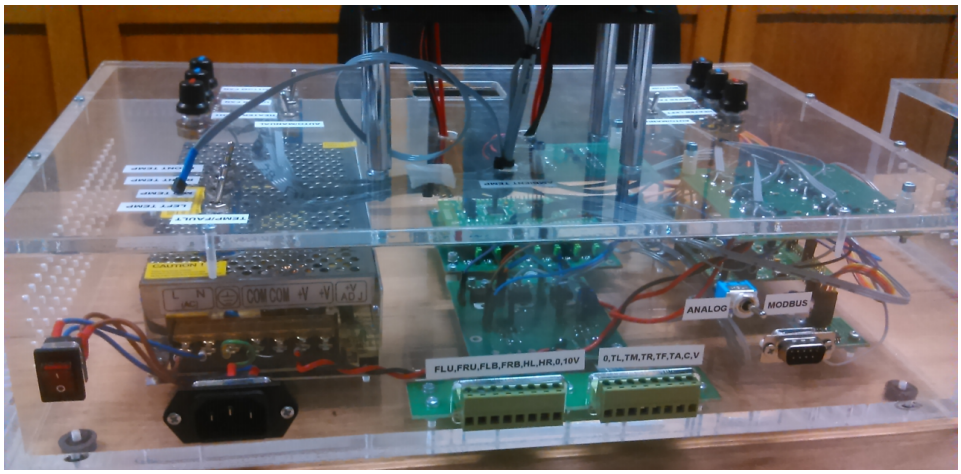
Przedni panel stanowiska (rys. 7.4) zawiera złącze USB mini-B, które jest wykorzystane do komunikacji z pakietem MATLAB za pomocą przygotowanych do tego celu bibliotek. Nad nim znajduje się przełącznik do wyboru kanału komunikacji: USB lub standard MODBUS/sygnał analogowy 0-10V. Na tej ściance znajduje się również naklejka (rys. 7.7), określająca znaczenie poszczególnych opcji konfigura-



Rys. 7.5. Stanowisko grzejąco-chłodzące: okno dostępu do przełączników konfiguracji MODBUS

cji protokołu MODBUS oraz sygnałów analogowych występujących na złączach śrubowych.

Na tylnej ścianie obudowy obiektu (rys. 7.6) znajdują się złącza służące do komunikacji z wykorzystaniem standardu analogowego 0-10 V oraz złącze DB9, przy użyciu którego przekazywana jest transmisja wiadomości MODBUS RTU (standard napięciowy RS485 2W). Przełącznik służący do wyboru między tymi dwoma trybami znajduje się nad nimi. Gniazdo kabla zasilającego (230V) i wyłącznik zasilania znajdują się po lewej stronie na tym panelu.

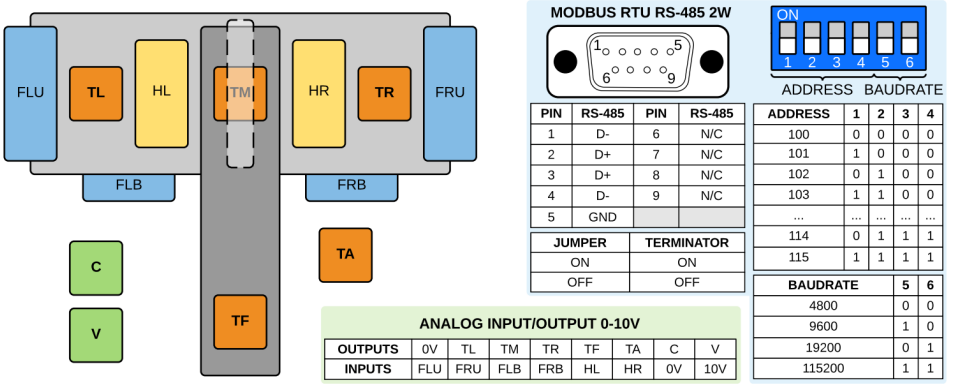


Rys. 7.6. Stanowisko grzejąco-chłodzące: tylny panel ze złączami

Struktura stanowiska zapewnia wpływ działania wielu elementów wykonawczych na te same wielkości wyjściowe, co prowadzi do powstania w obiekcie wielowymiarowym sprzężeń skrośnych i zmusza do implementacji bardziej skomplikowanych algorytmów regulacji niż proste jednopętlowe regulatory PID. Przykładowo, ogrzewanie grzałką HL bezpośrednio podnosi jednocześnie temperaturę TL i TM, natomiast wentylator FRU (zakładając, że przegroda jest założona) obniża temperaturę TM i TR. Warto pamiętać, że jest to pewne uproszczenie – w rzeczywistości grzałki wpływają na temperaturę całego stanowiska, ponieważ temperatura ta propaguje się poprzez szkielet obiektu. Nie mniej, czas propagacji temperatury z jednego końca obiektu na drugi jest na tyle duży, że można go w wielu przypadkach pominąć.

Istnieją trzy sposoby komunikacji z obiektem. Obejmują one: wykorzystanie standardu USB i implementacji dedykowanego protokołu, użycie standardu napię-

ciowego RS485 wraz z protokołem MODBUS RTU oraz użycie złączy śrubowych, gdzie poszczególne sygnały reprezentowane są przez wartości napięcia od 0 do 10 V. Połączenie z wykorzystaniem USB wykorzystuje konwerter UART-USB, który pozwala na przejście z informacji w standardzie USB do klasycznej transmisji szeregowej z wykorzystaniem linii Rx oraz Tx po stronie mikrokontrolera zarządzającego. Najwygodniejszą metodą na podłączenie się z użyciem kabla USB jest wykorzystanie środowiska MATLAB. W tym celu przygotowany został zestaw funkcji w tym właśnie języku. Standard komunikacyjny MODBUS RTU zaimplementowany został na mikrokontrolerze STM32F100RB, który z jednej strony komunikuje się z mikrokontrolerem zarządzającym, korzystając z dedykowanego protokołu, a z drugiej przy użyciu protokołu MODBUS RTU i przejściówki UARTRS485, ze światem zewnętrznym. Implementacja MODBUS RTU oparta jest na implementacji FreeMODBUS, której port został zrealizowany tak, aby pasował do mikrokontrolerów z rodziny STM32. Wykorzystanie standardu napięciowego RS485 pozwala na zastosowanie hierarchii master-slave, gdzie urządzeń drugiego typu może być aż 247. Zmiana parametrów konfiguracji komunikacji z wykorzystaniem MODBUS RTU, odpowiednie przełączniki umieszczone tuż obiektem właściwym, muszą zostać poprawnie poustawiane. Zmienione mogą zostać dwa parametry: prędkość transmisji (może przyjąć wartość: 4800, 9600, 19200, 115200) oraz adres urządzenia (od 100 do 115). Dodatkowo można założyć zworę, która dołącza terminator do linii komunikacyjnej MODBUS. Instrukcja dotycząca znaczenia poszczególnych konfiguracji widoczna jest na naklejce rys. 7.7 naklejonej na frontowej ścianie obiektu.



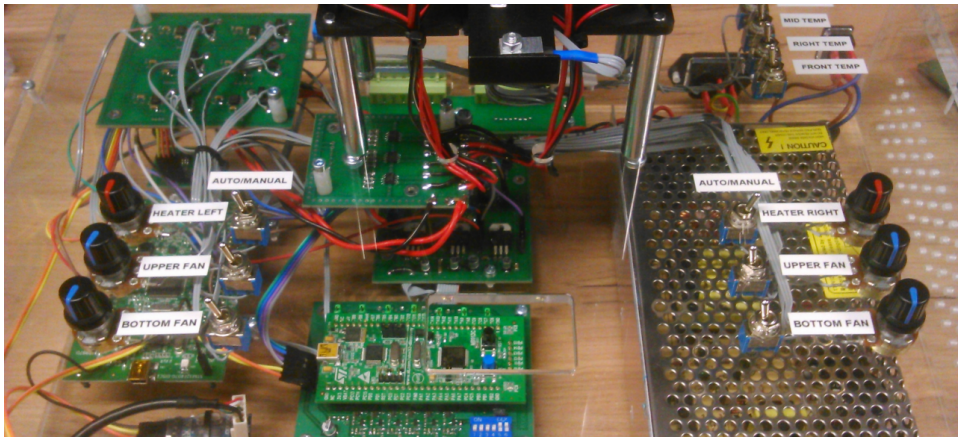
Rys. 7.7. Stanowisko grzejąco-chłodzące: naklejka przedstawiająca najważniejsze parametry komunikacji

Pozostałe parametry, które nie mogą zostać zmodyfikowane to test parzystości (domyślnie jest to test na parzystość), 8 bitów danych oraz 1 bit stopu. Linie D+ oraz D- podłączone są do pary wyprowadzeń z gniazda DB9, zostały zaimplementowane dwie najpopularniejsze konwencje (szczegóły na naklejce pokazanej na rys. 7.7). Aby korzystać z protokołu MODBUS RTU, musi zostać wybrany odpowiedni kanał komunikacyjny na obiekcie regulowanym, tj. frontowy przełącznik w pozycji MODBUS ANALOG oraz tylni w pozycji MODBUS. Następujące funkcje protokołu MODBUS zostały zaimplementowane:

1. Read Holding Registers (0x03),
2. Read Input Register (0x04),
3. Write Single Register (0x06),
4. Write Multiple Registers (0x10),
5. Read/Write Multiple Registers (0x17).

Do sterowania ręcznego wykorzystuje się przełącznik zmiany trybu pracy oraz potencjometr regulujący poziom mocy dostarczanej do elementu wykonawczego. Przy każdej parze znajduje się opis konkretnego elementu wykonawczego (rys. 7.8). Potencjometrem można ustawić poziom mocy dostarczanej do elementu wykonawczego, która jest proporcjonalna do wypełnienia sygnału PWM generowanego na płytce PWM. Częstotliwość sygnału wynosi około 1,4 kHz. Zakres regulacji wynosi $5 - 100\%$. Jeżeli przełącznik jest w lewym położeniu (etykieta „AUTO"), sygnał PWM pochodzi z płytki sterowania, jeżeli natomiast w prawym położeniu („MANUAL"), sygnał PWM pochodzi z płytki PWM. Zastosowanie takiego rozwiązania pozwala na symulację uszkodzeń elementów wykonawczych. W szczególności, można symulować odłączenie elementu wykonawczego przechodząc w tryb sterowania ręcznego z minimalnym poziomem sterowania.

Na 7.9 przedstawiono panel symulacji uszkodzeń czujników temperatury TL, TM, TR, TF. Umożliwia on sprzętowe odłączenie danego czujnika od magistrali OneWire, przez co wartość odczytywanej temperatury będzie równy wartości domyślnej $327.67°C$. Warto zauważyć, że wykorzystane czujniki są w stanie zmierzyć temperaturę od $-55°C$ do $125°C$, co oznacza, że wartość otrzymana w sytuacji odłączenia czujnika jest niemożliwa do zmierzenia w normalnych warunkach pracy, co pozwala jednocześnie na precyzyjne zidentyfikowanie wadliwego połączenia z czujnikiem. Dla czujnika temperatury otoczenia TA funkcja ta nie jest przewidziana i jego temperatura jest zawsze odczytywana prawidłowo. Tak samo wartości pomiarów prądu oraz napięcia są zawsze poprawne. Aby symulować odłączenie jed-



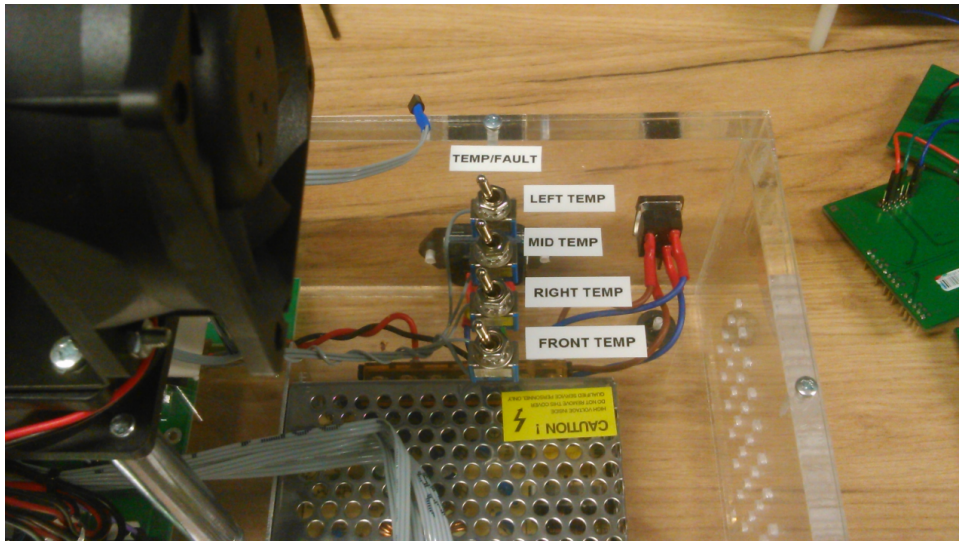
Rys. 7.8. Stanowisko grzejąco-chłodzące: zdjęcie panelu sterowania ręcznego

nego z czujników TL, TM, TR, TF, przełącznik przy jego nazwie należy ustawić w położeniu „FAULT". Aby symulować zakłócenie, można przełączyć jeden z elementów wykonawczych na sterowanie ręczne na krótki okres czasu. To podejście nie jest jednak zalecane, ponieważ generowane zakłócenie jest niezależne od obecnej wartości sterowania algorytmu regulacji. Inną (lepszą) metodą jest wykorzystanie nieużywanych elementów wykonawczych. Przykładowo, jeśli wykorzystane do regulacji zostaną wejścia HL i FLU, a regulowaną temperaturą jest temperatura TL, to do zakłóceń można wykorzystać grzałkę HR lub wentylatory TRU, TLB (po zdjęciu przesłony). Takie podejście pozwala na wymuszenie na algorytmie regulacji zmianę wartości sygnału sterującego zgodnie ze zmienionym środowiskiem.

7.2. Stanowisko świetlne

Stanowisko świetlne również zostało opracowane i wykonane w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej. W odróżnienia od stanowiska grzejąco-chłodzącego, charakteryzuje się bardzo szybką dynamiką i jest wykorzystywane w celu testowania szybkich algorytmów regulacji, o możliwie krótkim czasie obliczeń. Ciekawe wykorzystanie stanowiska wraz z wynikami podano w pracach [117, 118].

Ogólna struktura stanowiska świetnego została pokazana na rys. 7.10. Główna idea działania polega na umieszczeniu w tunelu świetlnym nadajnika (dioda LED) po jednej stronie i odbiornika (detektor światła) po drugiej stronie. Wykorzystano



Rys. 7.9. Stanowisko grzejąco-chłodzące: zdjęcie panelu symulacji uszkodzeń czujników temperatury

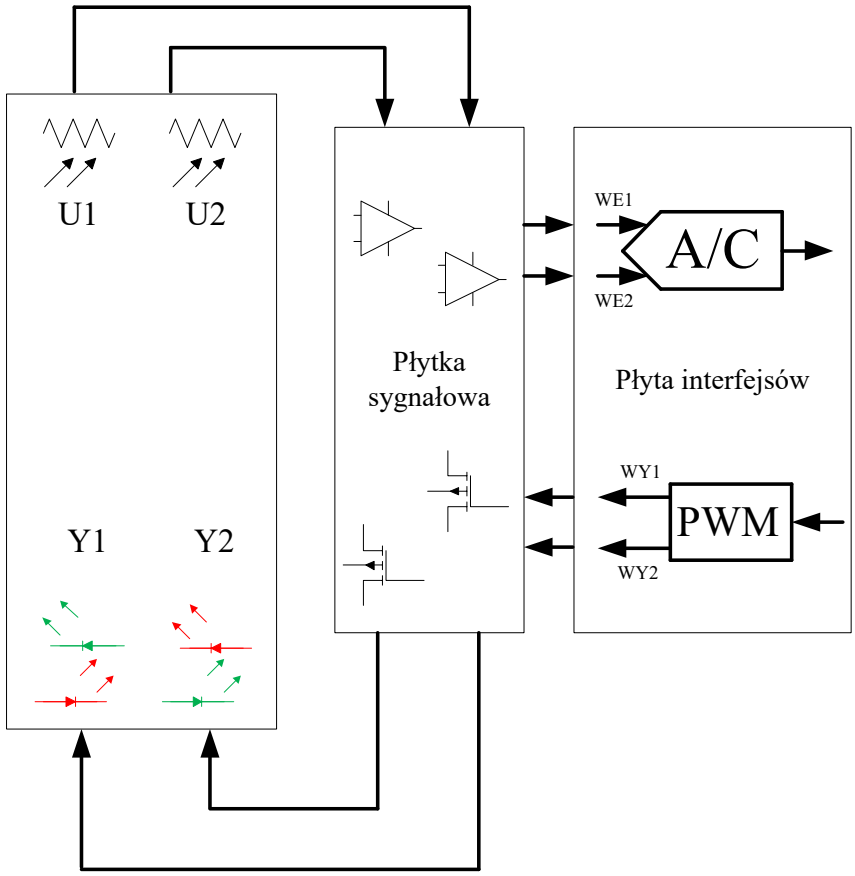
dwa takie tunele świetlne, aby umożliwić testowanie obiektów o dwóch wejściach i dwóch wyjściach. Ponadto, zastosowano dodatkowe diody, aby uzyskać efekt sprzężenia skrośnego, również można konfigurować charakterystykę sprzężenia skrośnego. Ponieważ informacja niesiona jest przez światło, można uzyskać bardzo szybkie odpowiedzi układu. Dlatego też obiekt został wyposażony w układy opóźnienia czasowego, oparte na klasycznej strukturze RC. Dzięki temu można modelować odpowiedź skokową układu i dopasować do potrzeb.

Opisywany proces służy do testowania szybkich algorytmów regulacji, o czasie próbkowania rzędu pojedynczych milisekund. Możliwe konfiguracje procesu są następujące: jedno wejście i jedno wyjście, jedno wejście i jedno wyjście z zakłóceniami, dwa wejścia i dwa wyjścia ze sprzężeniami skrośnymi.

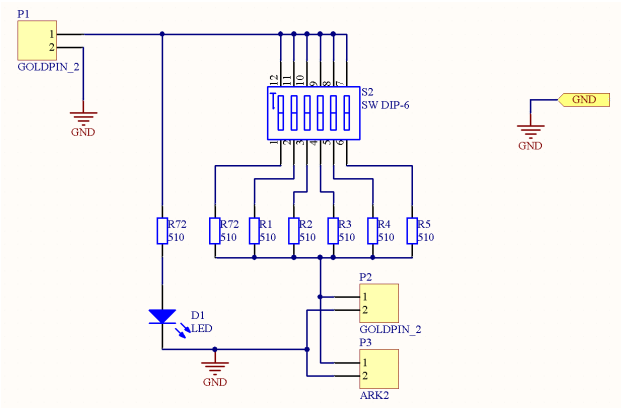
Sygnałami sterującymi są sygnały PWM doprowadzone do diód świecących LED. Schemat wejścia przedstawiono na rys. 7.11. Dzięki drabince rezystorów oraz przełącznikowi segmentowemu można ustawić rezystancją ograniczającą prąd diody, a tym samym dobrać wymaganą jasność diód. Dodatkowo, na płytce jest zamontowana dioda sygnalizująca prawidłową pracę danego wejścia.

Sygnałami wyjściowymi są sygnały analogowe napięciowy pochodzące z detektora światła, który są oświetlane diodami LED. Schemat takiego układu przedstawiono na rys. 7.12. Podobnie jak w przypadku sygnałów sterujących, również w przypadku sygnałów wyjściowych zastosowano drabinki rezystorów do ustawienia wstępnego prądu polaryzacji detektora oświetlenia. Wykorzystane zostały dwa rodzaje detektorów: TEPT4400 oraz GL5537. Sygnały wyjściowy trafiają do wtórnika zbudowanego na wzmacniaczu operacyjnym MCP6H01T. Jego zadaniem jest zapewnienie odpowiednio dużej rezystancji wejściowej, aby nie obciążać prądowo układu samego detektora. Za układem wtórnika znajduje się już wspomniany układ RC, dzięki któremu można wybrać odpowiednie stałe czasowe a tym samym wpływać na charakterystykę dynamiczną układu pomiarowego. Podobnie jak poprzednio zrealizowano ten układ przy pomocy drabinki rezystorów i kondensatorów z odpowiadającymi im przełącznikami segmentowymi. Na wyjściu układu również zastosowano wtórnik, którego zadanie jest identyczne jak poprzedniego. Dodatkowo wtórnik ten ma zapewniać odpowiednią wydajność prądową wyjścia, które może zostać podłączone do urządzenia pomiarowego.

Wszystkie układy elektroniczne, które są wymagane do stanowiska świetlnego zostały umieszczone na płytce drukowanej, której model przedstawiono na rys. 7.13.



Rys. 7.10. Stanowisko świetlne: struktura ogólna

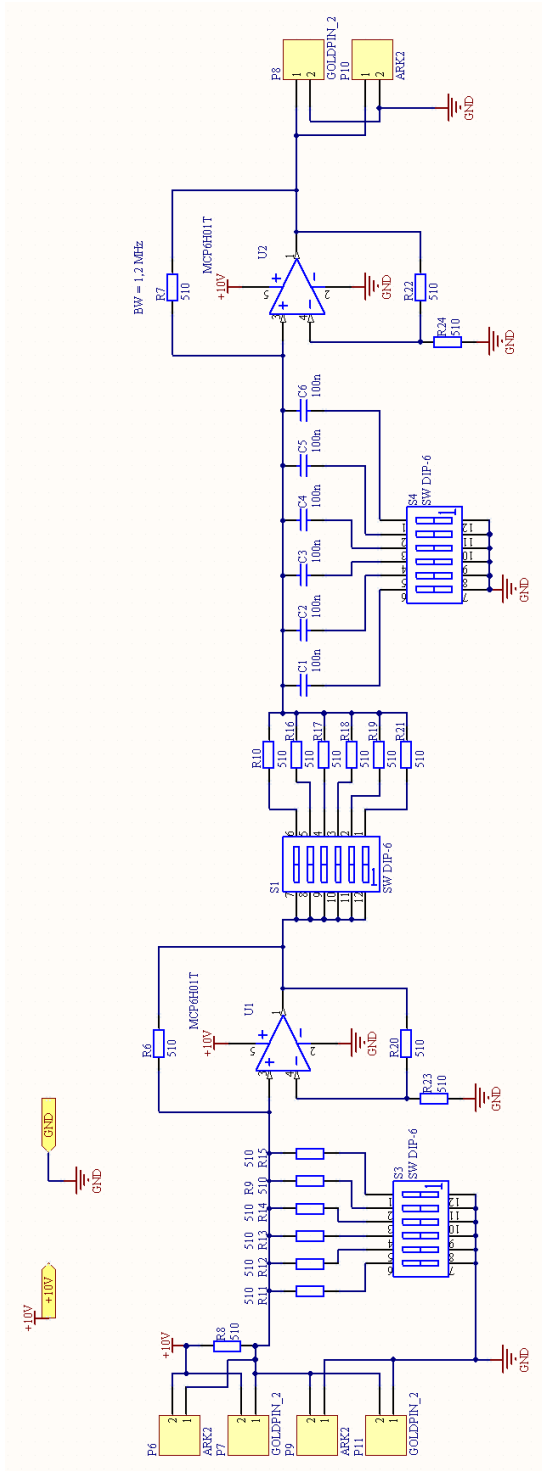


Rys. 7.11. Stanowisko świetlne: schemat obwodu wejściowego

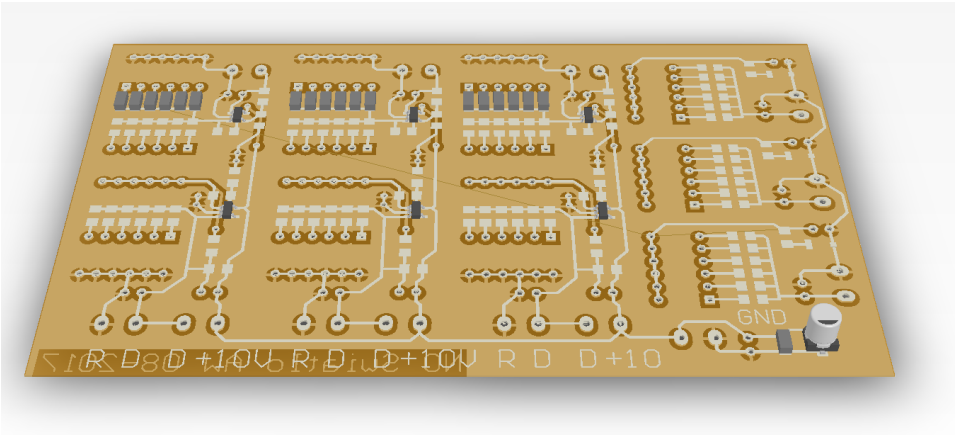
Całe stanowisko świetlne podłączone do systemu pomiarowego FPGA zostało przedstawione na rys. 7.14. Główny element stanowiska czyli tor świetlny został zrealizowany przy pomocy dwóch plastikowych korytek kablowych umieszczonych obok siebie. Oprócz sprzężenia skrośnego realizowanego dodatkowymi diodami LED, możliwe jest przesuwanie pokrywek korytek względem siebie. Zostały w nich wyfrezowane specjalne otwory, aby światło mogło przenikać z jednego toru świetlnego do drugiego. Dzięki temu można użyć dodatkowego sprzężenia skrośnego.

7.3. Serwomechanizm

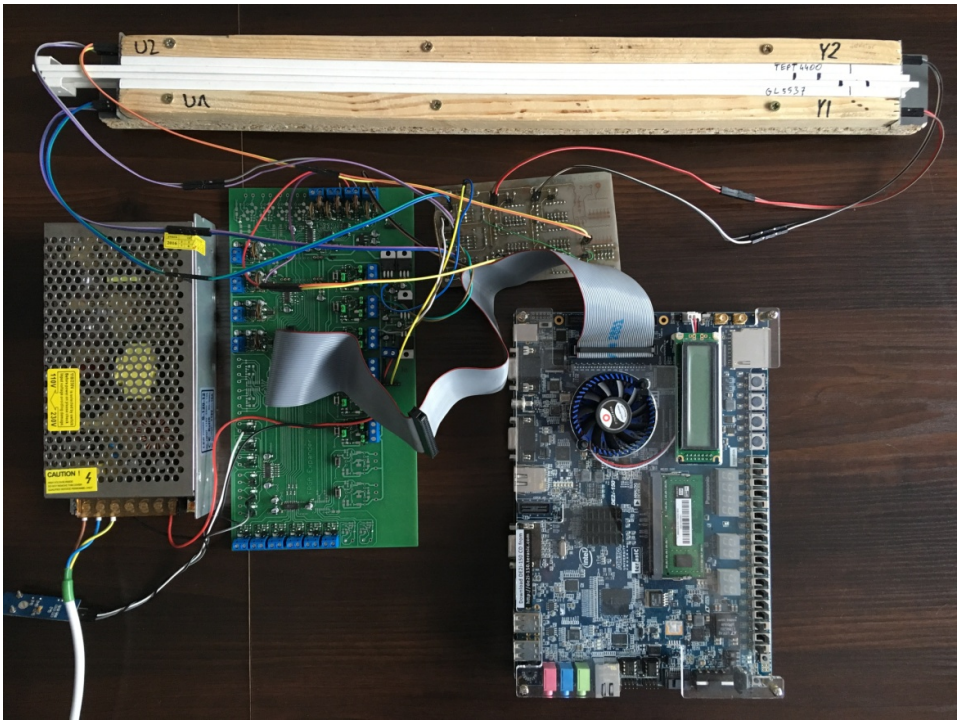
Podobnie jak proces świetlny, również stanowisko serwomechanizm zostało skonstruowane w celu testowania szybkich algorytmów regulacji, charakteryzujących się krótkim czasem próbkowania. Wykorzystano w tym celu przemysłowy zestaw napędowy składający się ze wzmacniacza oraz silnika serwo małej mocy i niskiej bezwładności własnej. Producentem sprzętu jest japońska firma Mitsubishi Electric. Zastosowany wzmacniacz i silnik jest produktem wysokiej klasy, spełniającym wygórowane wymagania przemysłowe dotyczące kompatybilności elektromagnetycznej. Wzmacniacz jest zbudowany z dedykowanych układów elektronicznych, opartych na najnowszych technologiach półprzewodnikowych. Steruje on silnik synchroniczny z magnesami trwałymi o bardzo dużej dynamice i wysokiej precyzji kontroli położenia, co jest zrealizowane przy pomocy przetwornika położenia kątowego o rozdzielczości 131072 impulsów na jeden obrót wału. Wzmacniacz serwo i silnik przedstawiono na rys. 7.15.



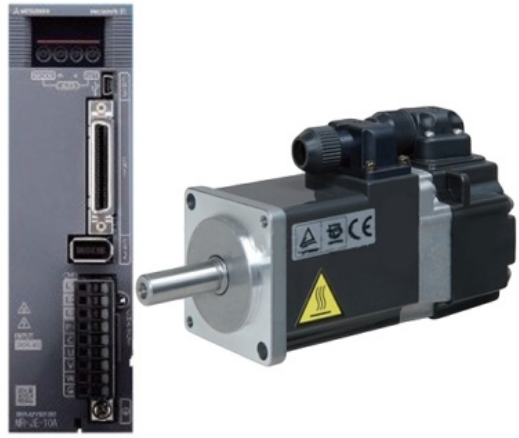
Rys. 7.12. Stanowisko świetlne: schemat obwodu wyjściowego



Rys. 7.13. Stanowisko świetlne: model płytki drukowanej



Rys. 7.14. Stanowisko świetlne w systemie regulacji

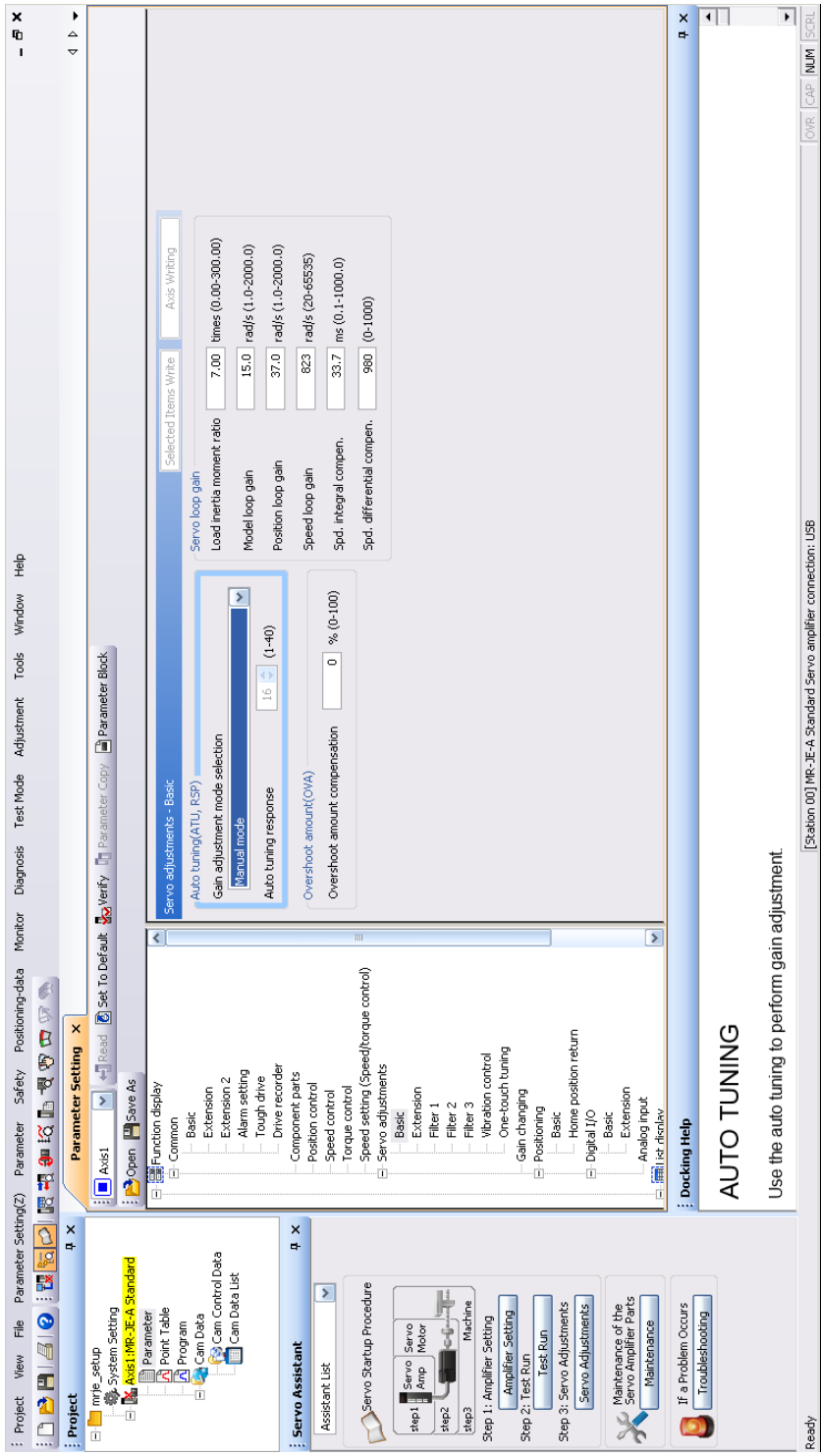


Rys. 7.15. Stanowisko serwomechanizm: silnik serwo i wzmacniacz

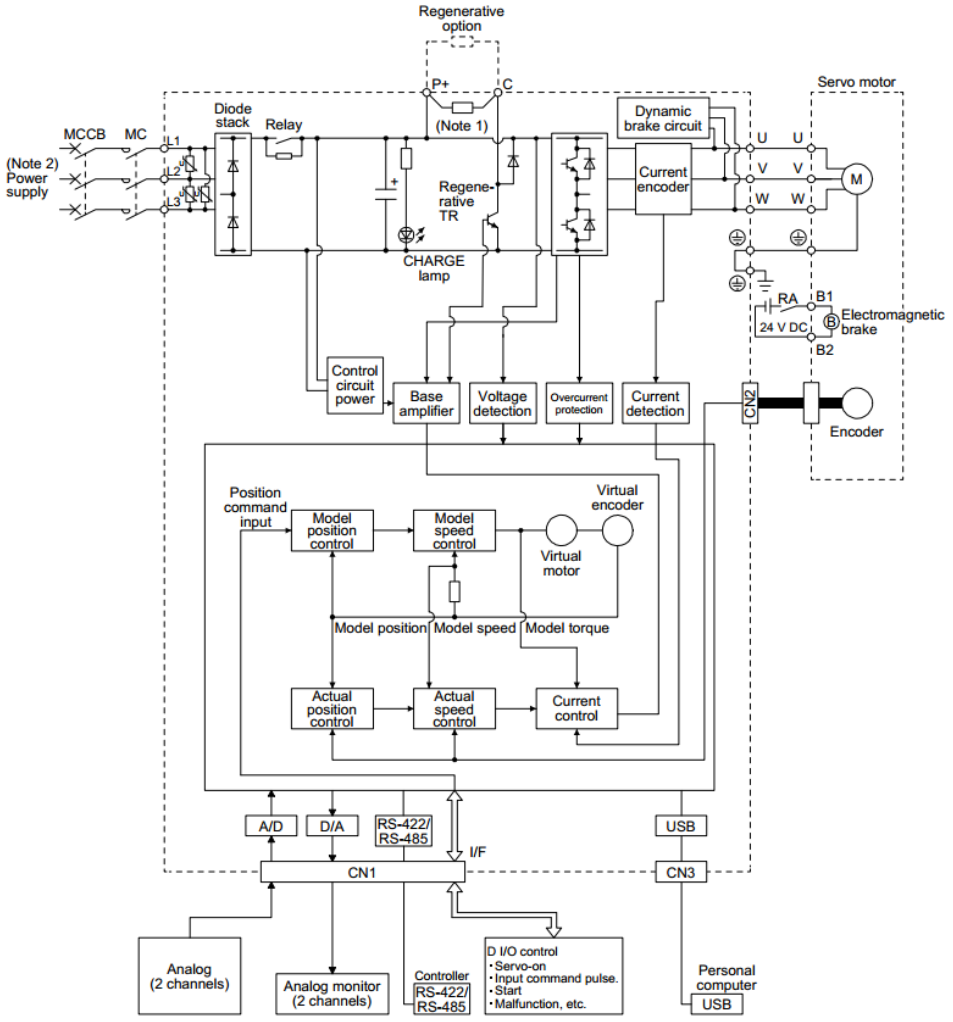
Do zarządzania układem wzmacniacza dołączona jest przez producenta aplikacja na komputer PC. Połączenie do wzmacniacza zrealizowano przy pomocy gniazda USB. Przy pomocy oprogramowania możliwa jest pełna parametryzacja urządzenia, możliwość diagnostyki, funkcja sterowania ręcznego oraz strojenia wbudowanego regulatora prędkości i pozycji. Przykładowy zrzut ekranu przedstawiono na rys. 7.16, gdzie pokazany jest moduł strojenia wbudowanego regulatora prędkości i pozycji.

Układ regulacji wzmacniacza posiada skomplikowaną strukturę opartą na wewnętrznym modelu i strukturze PID. Na rys. 7.17 przedstawiono modułowy schemat wewnętrzny elektryczny oraz układu regulatora. W układzie możliwe są trzy zasadnicze metody sterowania silnikiem. Pierwsza to z nich to metoda sterowania momentu wyjściowego, realizowana przez pętle regulacji prądu wyjściowego na silnik. Zależność między prądem a momentem jest w przybliżeniu proporcjonalna, a więc podawany sygnał analogowy jest wartością sterującą. Druga metoda to sterowania prędkościowe, w której zadawany sygnał analogowy przekłada się bezpośrednio na prędkość obrotową silnika. W tym podejściu wyjściowy moment elektryczny jest dobierany automatycznie przez pętle regulacji prądu. Trzecia metoda to sterowanie pozycyjne, gdzie sterowaniem są impulsy cyfrowe podawane na wejście wzmacniacza i zliczane przez szybki licznik. Każda z tych metod posiada odpowiednie nastawy poszczególnych regulatorów, parametry te są pokazane na rys. 7.16. Każdy z regulatorów ma swój odpowiednik w świecie wirtualnym wzmacniacza, którego parametry również można zmieniać. Dzięki temu można zrealizować układ o bardzo wysokiej dynamice i dużej sztywności oraz układ o bardzo małej dynamice i całkowicie miękki. Podobnie jak ma to miejsce w stanowisku stanowiska świetlnego, również i w stanowisku serwomechanizm możliwe jest dobranie charakterystyki obiektu przez parametrów. Na przykład, można bardzo łatwo odtworzyć parametry powtórzyć eksperymenty.

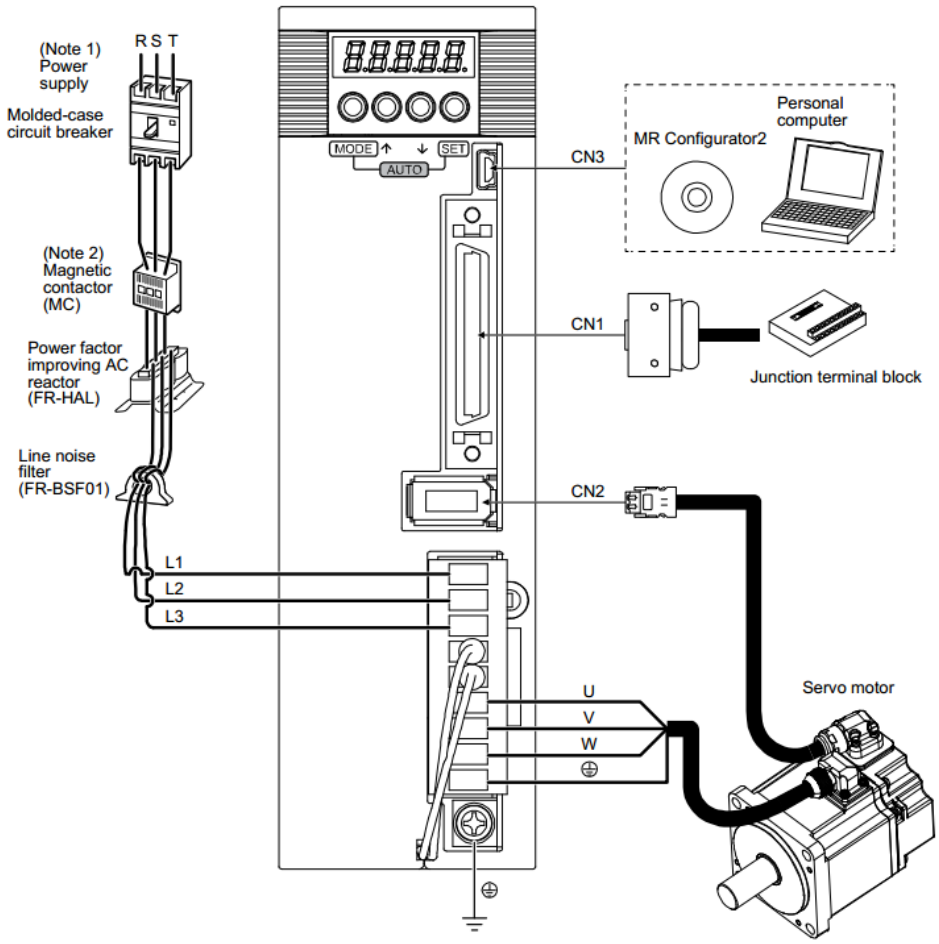
Na rys. 7.18 przedstawiono schemat podłączenia wzmacniacza i silnika do zasilania. Zasilanie wzmacniacza zrealizowane jest za pomocą trzech faz 200 V. Wewnątrz wzmacniacza napięcie to jest stabilizowane i sięga około 400 V. Buforem prądowym jest bateria kondensatorów, która w chwilach dynamicznych ruchów silnika jest w stanie dostarczyć odpowiedni prąd. Dany wzmacniacz jest w stanie przekroczyć swój nominalny moment wyjściowy o 300% (oczywiście tylko przez krótki czas). Silnik z kolei jest zasilony z trzech faz U, V oraz W, generowanych przez tranzystory wyjściowe odpowiednio kluczując wspomnianą szynę DC.



Rys. 7.16. Stanowisko serwomechanizm: konfiguracja parametrów



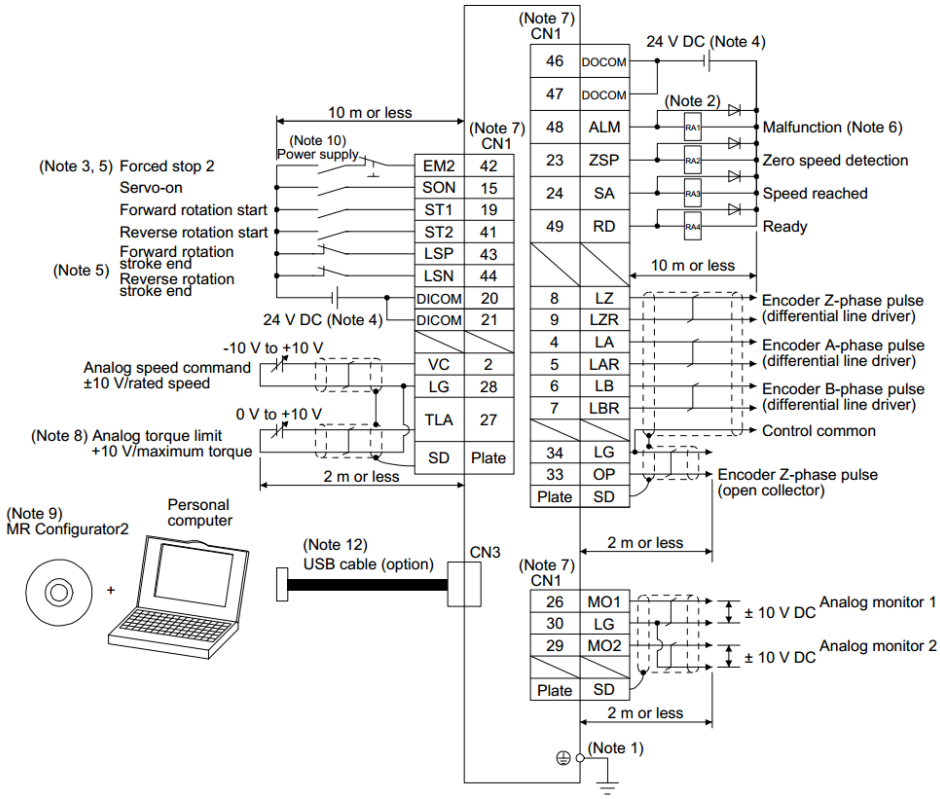
Rys. 7.17. Stanowisko serwomechanizm: schemat wewnętrzny regulatora



Rys. 7.18. Stanowisko serwomechanizm: schemat podłączeń elektrycznych

W zrealizowanym stanowisku wykorzystano metodę sterowania prędkością silnika. W tym celu należy do wzmacniacza dostarczyć sygnał analogowy z zakresu ±10 V. Na rys. 7.19 będą to zaciski VC(2) i LG(28). Maksymalne napięcie +10 V będzie oznaczało zadaną prędkość silnika 6000 obrotów na minutę. Jest to zatem 100 obrotów na sekundę. Jak wspomniano wyżej przetwornik położenia liczy 131072 impulsów na obrót silnika. Zatem częstotliwość sygnału enkodera może sięgać $131072 \cdot 100 = 13107200$ Hz. Warto podkreślić, że układy regulacji przy takich częstotliwościach wymagają bardzo szybkiej reakcji, stąd zasadne jest wybranie dostatecznie szybkiej platformy sprzętowej. Sygnałem wyjściowym do regulatora zewnętrznego jest również sygnał analogowy pochodzący z zacisków MO1 (26) i LG(30), który podaje napięcie proporcjonalne do aktualnej prędkości silnika. Dodatkowo możliwe jest ograniczenie momentu silnika sygnałem analogowym oraz pomiar aktualnego momentu na wyjściu silnika. Można w ten sposób zrealizować np. regulację momentu docisku prasy elektrycznej. Pozostałe sygnały cyfrowe służą do aktywacji pracy układu, sterowania dodatkowych funkcji czy wyłączenia awaryjnego.

Podsumowując, stanowisko serwomechanizm umożliwia parametryzację dynamiki układu oraz szeroki zakres regulacji prędkości. Dzięki temu można dowolnie profilować obiekt i dostosować do potrzeb testowanego regulatora. Cechy te pozwalają testowanie szybkich algorytmów przy czasie próbkowania rzędu pojedynczych milisekund.



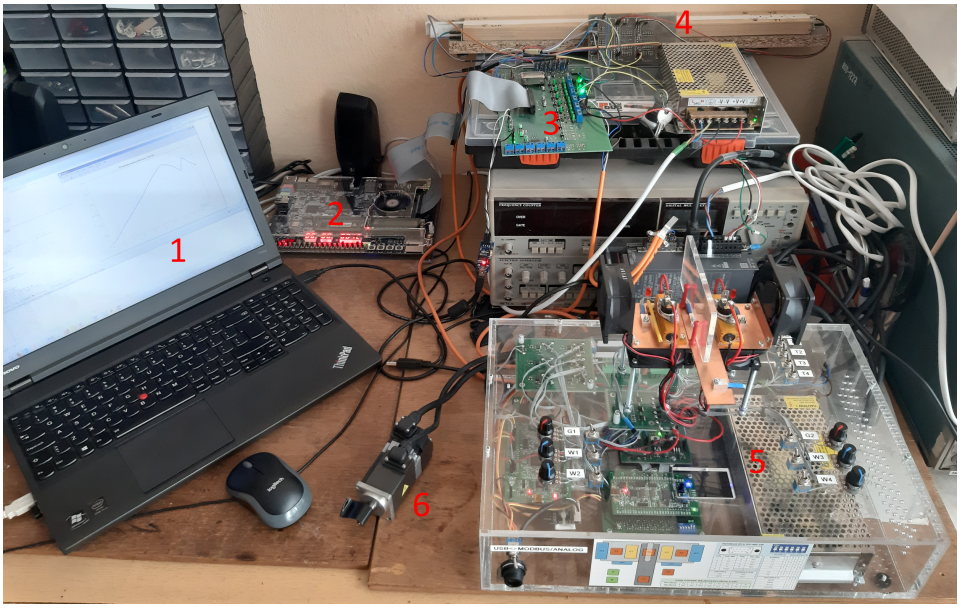
Rys. 7.19. Stanowisko serwomechanizm: podłączenie sygnałów sterujących

Rozdział 8

Wyniki eksperymentów

Na rys. 8.1 przedstawiono całe stanowisko badawcze. Wszystkie testowane obiekty mogą być podłączone jednocześnie do płyty interfejsów sprzętowych. Poniżej znajduje się lista elementów składowych:

1. komputer ze środowiskiem Quartus Prime oraz MATLAB,
2. płytka uruchomieniowa FPGA DE2i-150,
3. dedykowana płyta interfejsów sprzętowych,
4. stanowisko świetlne,
5. stanowisko grzejąco-chłodzące,
6. silnik serwo ze stanowiska serwomechanizm.



Rys. 8.1. Zdjęcie całego systemu pomiarowego

Na rys. 8.2 przedstawiono przykładową konfigurację całego układu regulacji obiektu świetlnego, który obejmuje:

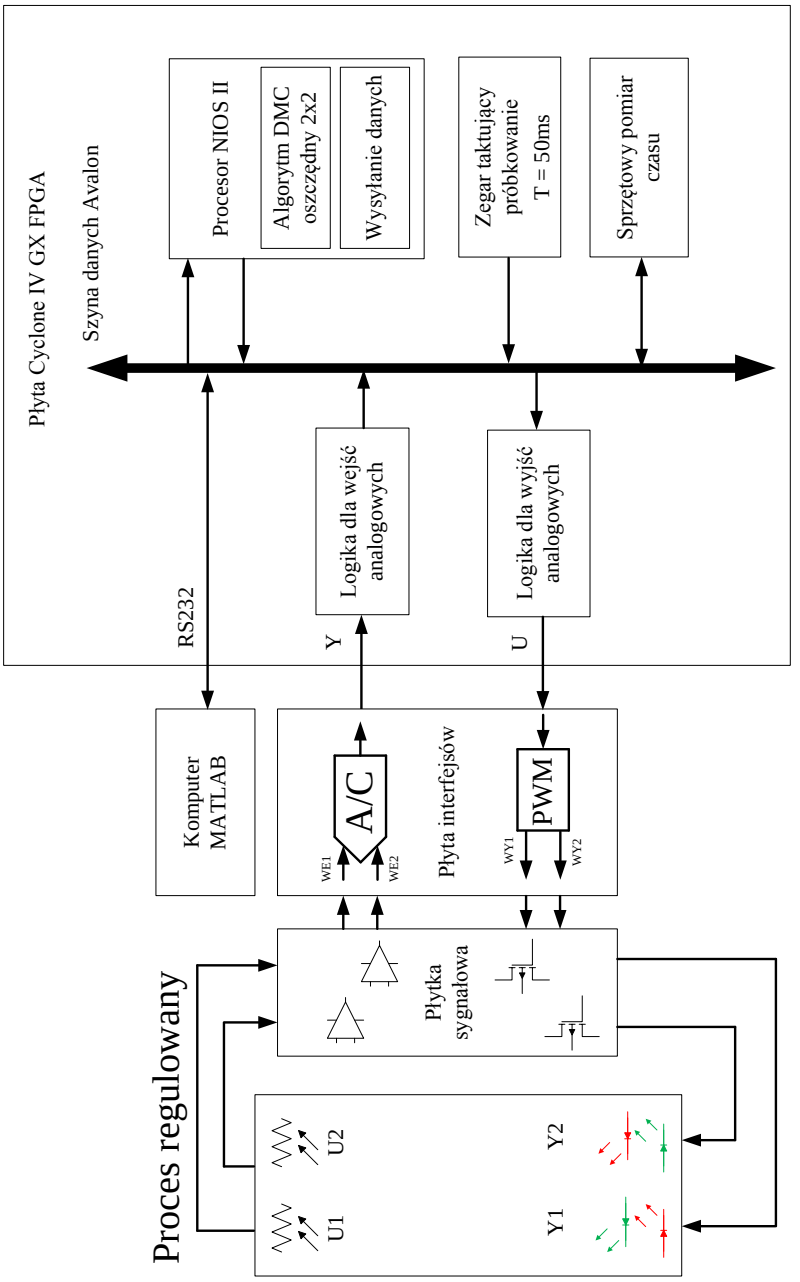
— płytę FPGA, na której zaimplementowano regulator używając języka VHDL oraz C,
— komputer z pakietem MATLAB do akwizycji i prezentacji wyników regulacji,
— płytę interfejsów dopasowującą sygnały przemysłowe do +3.3 V,
— obiekt regulacji – w tym przypadku jest to stanowisko świetlne.

Sercem systemu sterującego jest procesor NIOS II z zaimplementowanymi algorytmami regulacji predykcyjnej. Część sprzętowa musi zapewnić prawidłowe opisanie zasobów procesora NIOS II oraz bloki pomiarowo-sterujące. Taka struktura nie pozwoli bezpośrednio zrealizować operacji równoległych (z uwagi na sekwencyjną naturę działania procesora NIOS II) czy redundancji. Możliwe jest natomiast zaimplementowanie kilku procesorów NIOS II w jednej strukturze FPGA i dzięki takiemu podejściu można realizować operacje równolegle, a także zapewnić redundancję systemu. Ponadto, w takiej strukturze można zastosować następujący scenariusz: pierwszy procesor pracuje jako interfejs komunikacyjny, drugi procesor pracuje jako zarządca danych z urządzeń kontrolowanych, trzeci procesor pracuje jako regulator procesu krytycznego, o szybkim czasie reakcji rzędu milisekund, czwarty procesor pracuje jako regulator kilku procesów podrzędnych, o czasie reakcji rzędu sekund.

Dla każdego stanowiska przeprowadzono szereg testów algorytmów regulacji predykcyjnej. W dalszej części niniejszego rozdziału kolejno przedstawione są wyniki dla następujących algorytmów: DMC w wersji analitycznej, DMC w wersji analitycznej oszczędnej, DMC w wersji numerycznej, GPC w wersji analitycznej, GPC w wersji analitycznej oszczędnej, GPC w wersji numerycznej.

Dla każdego obiektu wykonywano procedurę strojenia regulatora predykcyjnego [120], która obejmowała przedstawione poniżej etapy:

1. dobór horyzontu dynamiki obiektu (tylko dla algorytmu DMC),
2. ustawienie jednostkowych współczynników λ_n oraz ψ_m,
3. ustawienie długich horyzontów predykcji i sterowania (dla algorytmu DMC równych horyzontowi dynamiki),
4. stopniowe zmniejszanie długości horyzontów predykcji i sterowania, obserwacja jakości regulacji,
5. po ustaleniu długości horyzontu predykcji stopniowe zmniejszanie długości horyzontu sterowania, obserwacja jakości regulacji,
6. po ustaleniu długości horyzontu sterowania ewentualna modyfikacja współczyn-

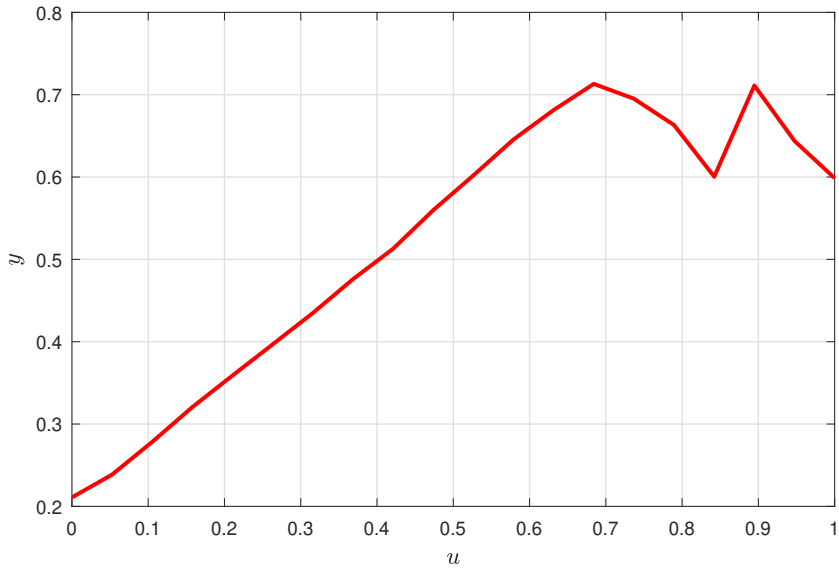


Rys. 8.2. Przykładowy schemat pomiarowy dla obiektu świetlnego

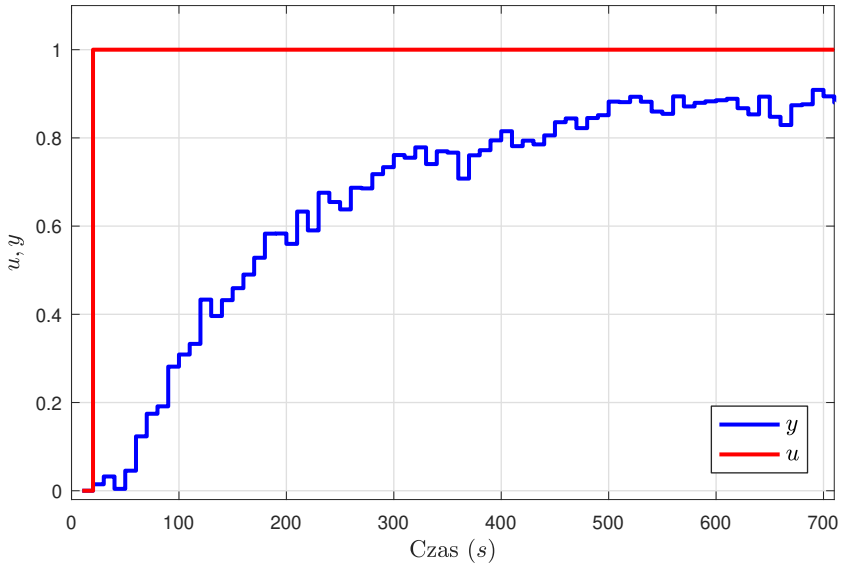
ników λ_n oraz ψ_m w celu ograniczenia dynamiki zmiany sygnałów sterujących oraz poprawy jakości regulacji.

8.1. Stanowisko grzejąco-chłodzące

Stanowisko grzejąco-chłodzące zostało skonfigurowane jako obiekt o jednym wejściu i jednym wyjściu. Elementem sterowanym jest grzałka G1, równolegle uruchomiony jest wentylator W1 ze stałą wartością sterowania. Pomiar aktualnej wartości temperatury realizowany jest przez czujnik T1. Okres próbkowania wynosi 10 sekund. Przyjęto ograniczenia sygnału sterującego: $u =< 0,1 >$. Na rys. 8.3 przedstawiono charakterystykę statyczną układu. W zakresie wartości sterowania od 0 do około 0,7 widać liniową zależność wyjście-wejście. Od wartości 0,7 widać charakterystyczne skoki wartości wyjściowej, które są spowodowane przez sprzętowe zabezpieczenie termiczne stanowiska. Na rys. 8.4 przedstawiono odpowiedź skokową układu. Okres próbkowania sygnałów wynosi 10 sekund.



Rys. 8.3. Stanowisko grzejąco-chłodzące: charakterystyka statyczna



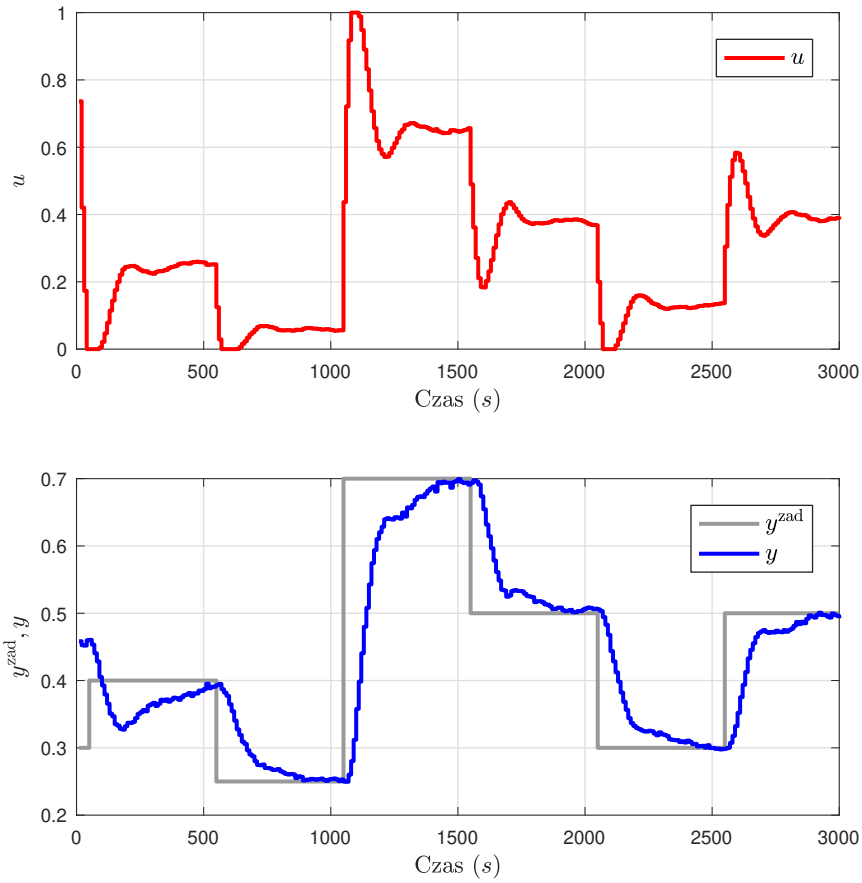
Rys. 8.4. Stanowisko grzejąco-chłodzące: znormalizowana odpowiedz skokowa

Model obiektu można przedstawić w postaci równania różnicowego (2.55), gdzie

$$A_{1,1}(z^{-1}) = 1 - 1.7805z^{-1} + 0.7925z^{-2} \tag{8.1}$$

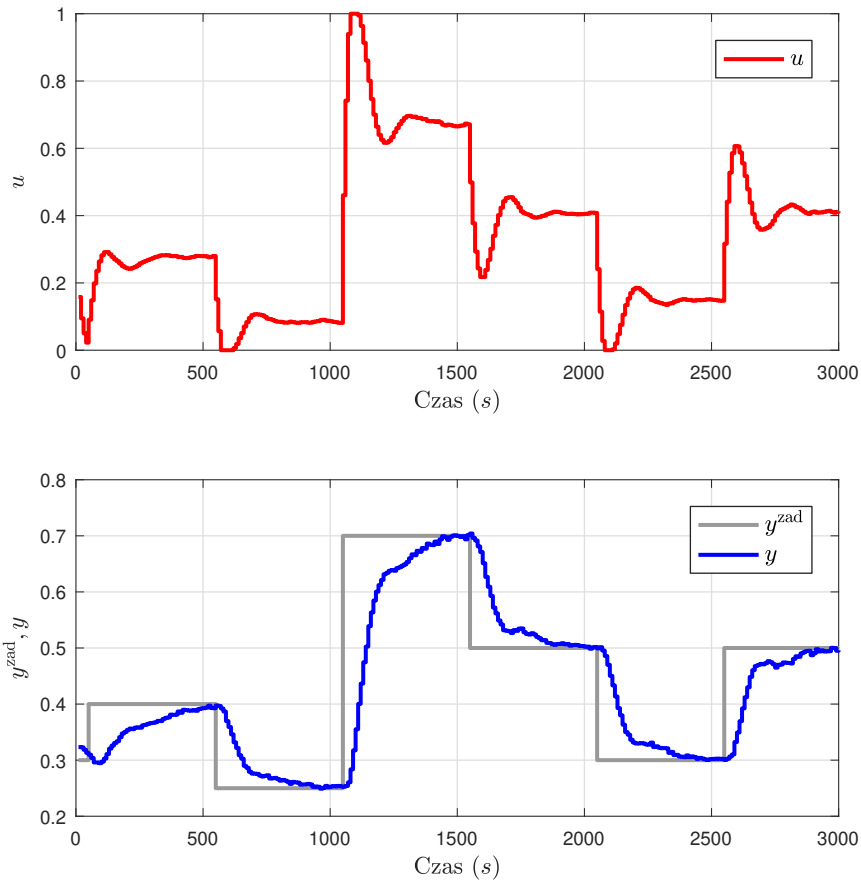
$$B_{1,1}(z^{-1}) = 0.0055z^{-1} + 0.0051z^{-2} \tag{8.2}$$

Na rys. 8.5 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC. Zestaw parametrów jest następujący: horyzont dynamiki $D = 65$, horyzont predykcji $N = 20$, horyzont sterowania $N_{\mathrm{u}} = 8$, $\lambda = \psi = 1$. Ich dobór został wykonany doświadczalnie w celu uzyskania dobrej jakości regulacji. Są one stosowane we wszystkich sześciu algorytmach regulacji procesu grzejąco-chłodzącego (horyzont dynamiki tylko w algorytmie DMC). Czas potrzebny do wyznaczenia sterowania w jednej chwili próbkowania algorytmu wynosi 4,4 ms. W algorytmach analitycznych czas ten jest zawsze stały.



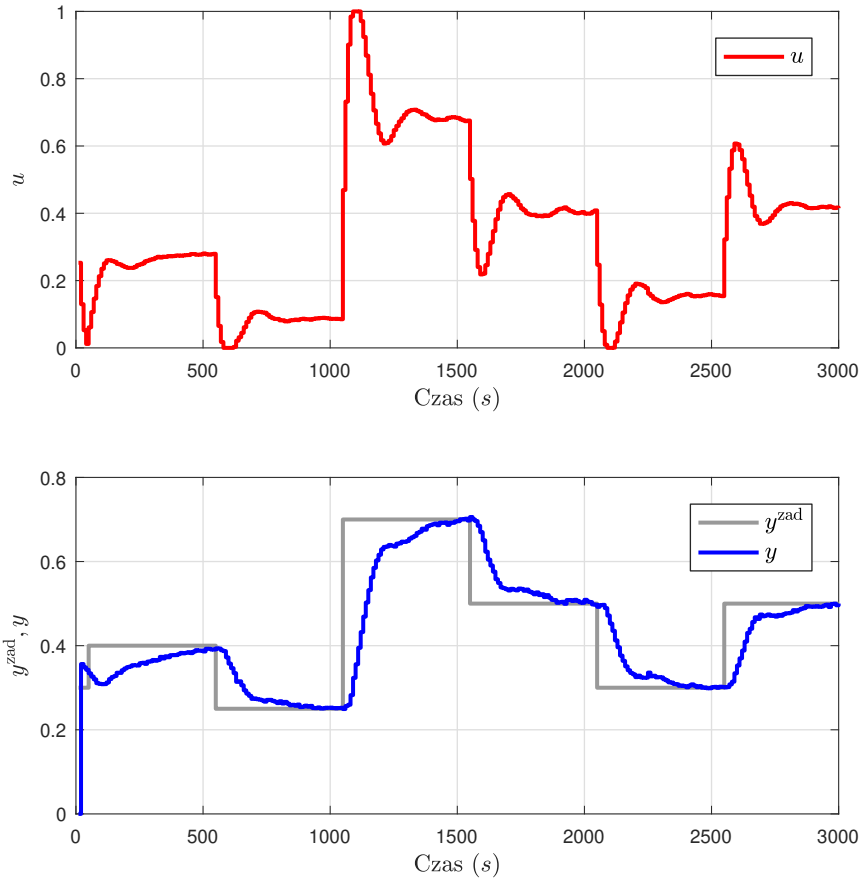
Rys. 8.5. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC

Na rys. 8.6 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej. Przebiegi są praktycznie takie same w obu wersjach (niewielkie różnice spowodowane są nieco innymi warunkami początkowymi, szumem pomiarowym i zmianami temperatury zewnętrznej). W porównaniu z klasyczną wersją analityczną, czas obliczeń został znacznie znacznie zredukowany, do wartości 0,4 ms.



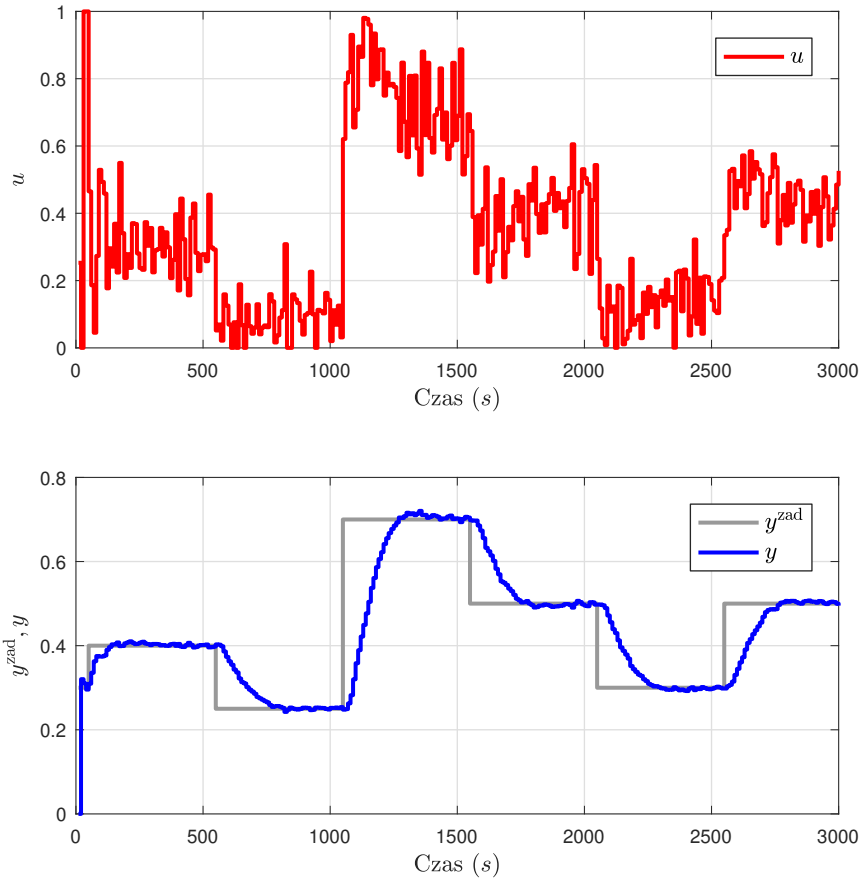
Rys. 8.6. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej

Na rys. 8.7 przedstawiono trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu DMC, które są bardzo podobne do trajektorii uzyskanych przy użyciu algorytmów analitycznych. Zjawisko to jest opisywane w literaturze [107, 108], różnice między algorytmami analitycznymi i numerycznymi mogą pojawić się w trudnych przypadkach wielowymiarowych oraz przy uwzględnieniu ograniczeń prognozowanych sygnałów wyjściowych. W porównaniu z algorytmami analitycznymi znacznie zwiększył się czas obliczeń, maksymalna jego wartość wynosi 33,8 ms, ale jest on i tak bardzo krótki w porównaniu z okresem próbkowania (10 sekund). Warto podkreślić, że algorytmie numerycznym czas obliczeń w poszczególnych chwilach dyskretnych nie jest taki sam, ponieważ czas rozwiązania zadania optymalizacji kwadratowej zależy od aktualnego punktu początkowego.



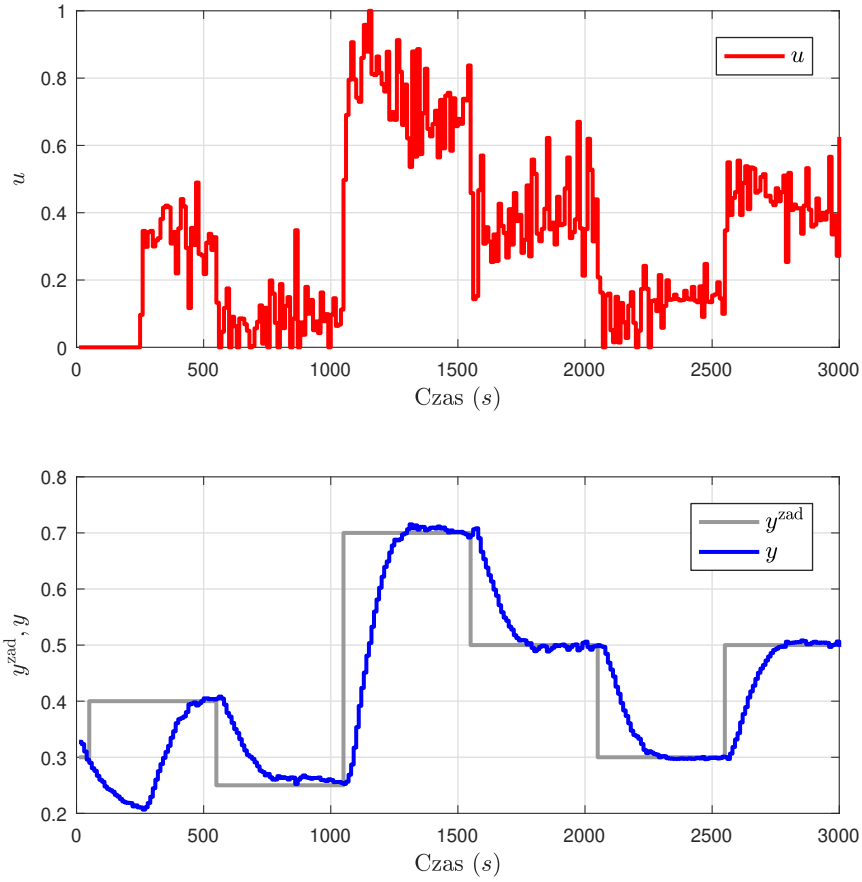
Rys. 8.7. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu DMC

Na rys. 8.8 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu GPC. Uzyskany przebieg sygnału wyjściowego jest praktycznie taki sam jak w algorytmach DMC, ale trajektoria sygnału sterującego charakteryzuje się dużą zmiennością. Jest to spowodowane tym, że w algorytmie DMC na obliczany sygnał sterujący wpływa jedynie sygnał wyjściowy procesu z chwili k (wzór (2.52)), natomiast w algorytmie GPC – sygnał wyjściowy z chwil $k-n_{\mathrm{A}}, \ldots, 0$ (wzór (2.73)). Ponieważ w algorytmie GPC stosuje się prosty model ARX o niewielkiej liczbie parametrów, a nie model odpowiedzi skokowej o dużej liczbie parametrów, czas obliczeń w każdej chwili próbkowania wynosi 1,5 ms, podczas gdy w analitycznym algorytmie DMC wynosi on 4,4 ms.



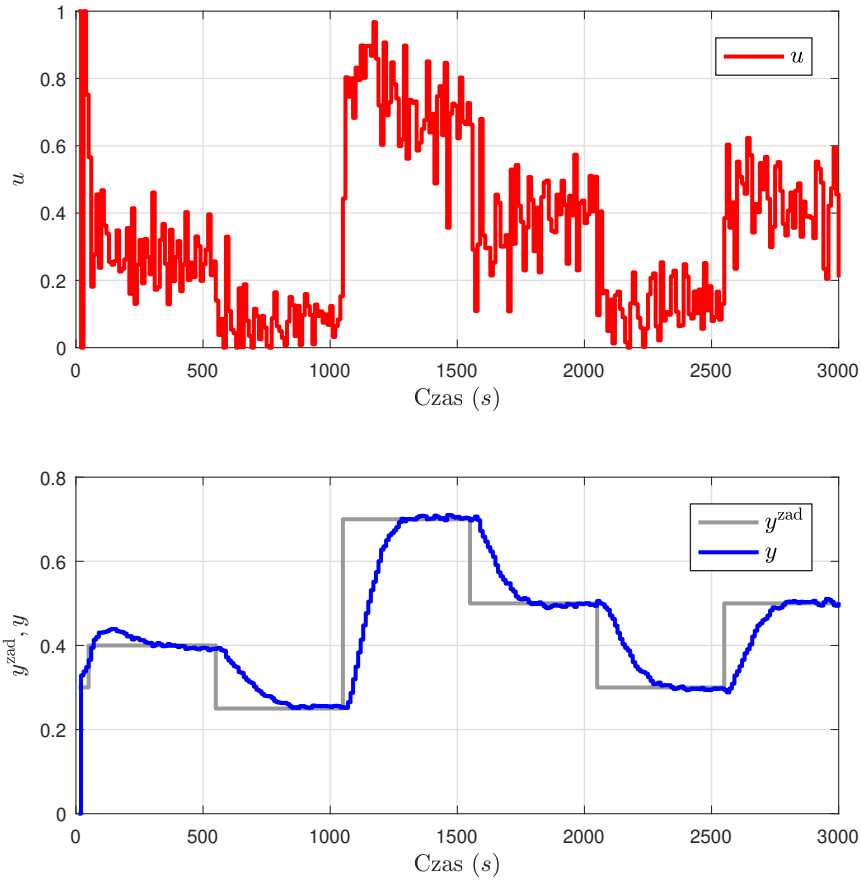
Rys. 8.8. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu GPC

Na rys. 8.9 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu GPC w wersji oszczędnej. Czas obliczeń w każdej chwili próbkowania wynosi poniżej 0,1 ms, podczas gdy w analitycznym algorytmie DMC w wersji oszczędnej wynosi on 0,4 ms, a w klasycznym analitycznym algorytmie GPC 1,5 ms.



Rys. 8.9. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu GPC w wersji oszczędnej

Na rys. 8.10 przedstawiono trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu GPC. Maksymalny czas obliczeń wynosi 30,6 ms. Jest on nieznacznie krótszy od czasu obliczeń algorytmu numerycznego DMC (33,8 ms). Spowodowane jest to tym, że zależy on głównie od czasu rozwiązania zadania optymalizacji kwadratowej, które w obu algorytmach (DMC i GPC) jest praktycznie takie samo. Oczywiście, czas obliczeń jest znacznie dłuższy niż w przypadku algorytmów analitycznych, zarówno DMC, jak i GPC.



Rys. 8.10. Stanowisko grzejąco-chłodzące: trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu GPC

W tab. 8.1 porównano czasy obliczeń wszystkich sześciu rozważanych algorytmów regulacji predykcyjnej stanowiska grzejąco-chłodzącego.

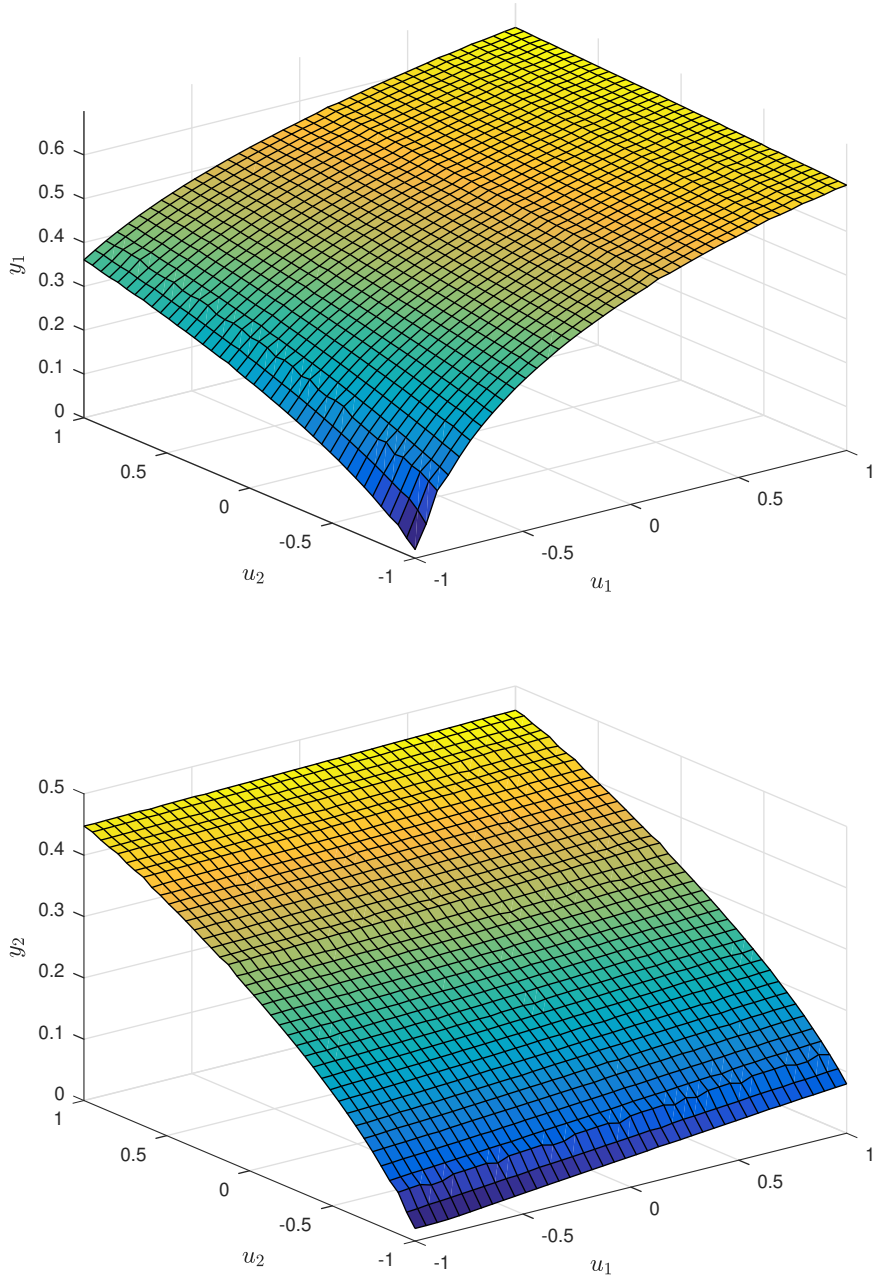
Tabela 8.1. Stanowisko grzejąco-chłodzące: porównanie czasów obliczeń (T_{obl} (ms)) pojedynczej iteracji rozważanych algorytmów regulacji predykcyjnej

Algorytm	D	N	N_{u}	T_{obl} (ms)
DMC analityczny	65	20	8	4,4
DMC analityczny oszczędny	65	20	8	0,4
DMC numeryczny	65	20	8	33,8
GPC analityczny	–	20	8	1,5
GPC analityczny oszczędny	–	20	8	0,1
GPC numeryczny	–	20	8	30,6

8.2. Stanowisko świetlne

Stanowisko świetlne zostało skonfigurowane jako obiekt o dwóch wejściach i dwóch wyjściach. Sygnały sterujące dwoma torami świetlnymi to u_1 oraz u_2. Przyjęto ograniczenia sygnału sterującego: $u_1 =< -1, 1 >, u_2 =< -1, 1 >$. Pomiar aktualnych wartości światła realizowany jest przez czujniki y_1 oraz y_2. Okres próbkowania wynosił 50 ms. Na rys. 8.11 przedstawiono charakterystyki statyczne zmierzone i zapisane przez system pomiarowy FPGA. Z charakterystyk można odczytać, że tory główne u_1-y_1 oraz u_2-y_2 są dominujące, ale w procesie występuje sprzężenie skrośne obu torów wejście-wyjście. Można zauważyć, że przy małych wartościach sterowania właściwości statyczne obiektu są silnie nieliniowe. Przy większych wartościach sygnałów sterujących obiekt przyjmuje charakter zbliżony do liniowego.

Na rys. 8.12 przedstawiono rzeczywiste (zmierzone) odpowiedzi skokowe (oznaczenie P) stanowiska świetlnego. Rysunek przedstawia dodatkowo charakterystyki dobranego modelu (oznaczenie M), zastosowano modele o pierwszym rzędzie dynamiki. Model obiektu można przedstawić w postaci równania różnicowego (2.55),



Rys. 8.11. Stanowisko świetlne: charakterystyka statyczna

gdzie

$$A_{1,1}(z^{-1}) = 1 - 1.2085z^{-1} + 0.3642z^{-2} \tag{8.3}$$

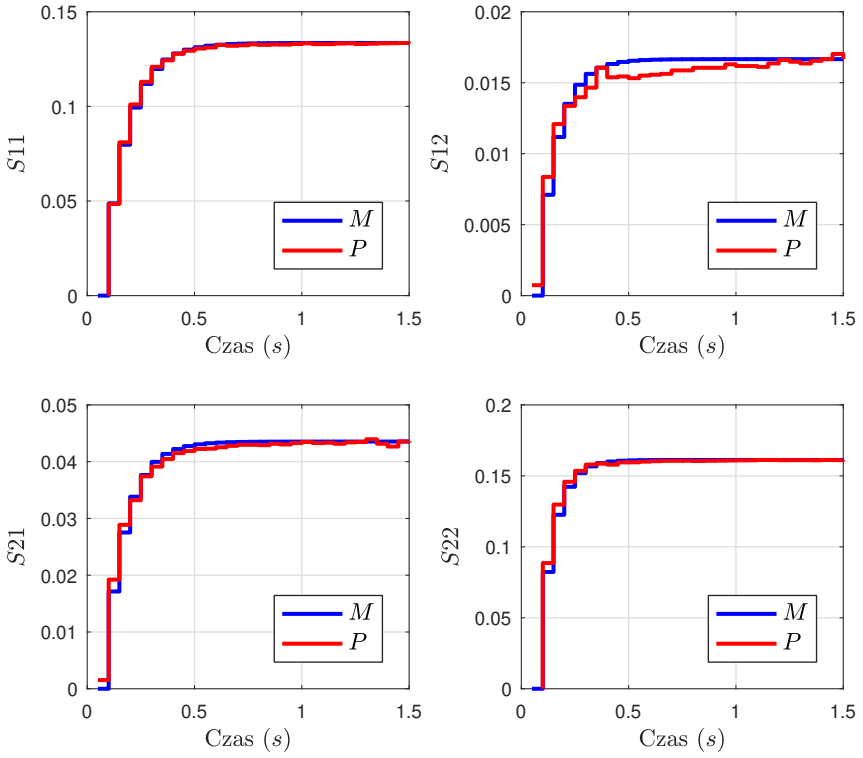
$$A_{2,2}(z^{-1}) = 1 - 1.0961z^{-1} + 0.2969z^{-2} \tag{8.4}$$

$$B_{1,1}(z^{-1}) = 0.0488z^{-1} - 0.0280z^{-2} \tag{8.5}$$

$$B_{1,2}(z^{-1}) = 0.0071z^{-1} - 0.00045z^{-2} \tag{8.6}$$

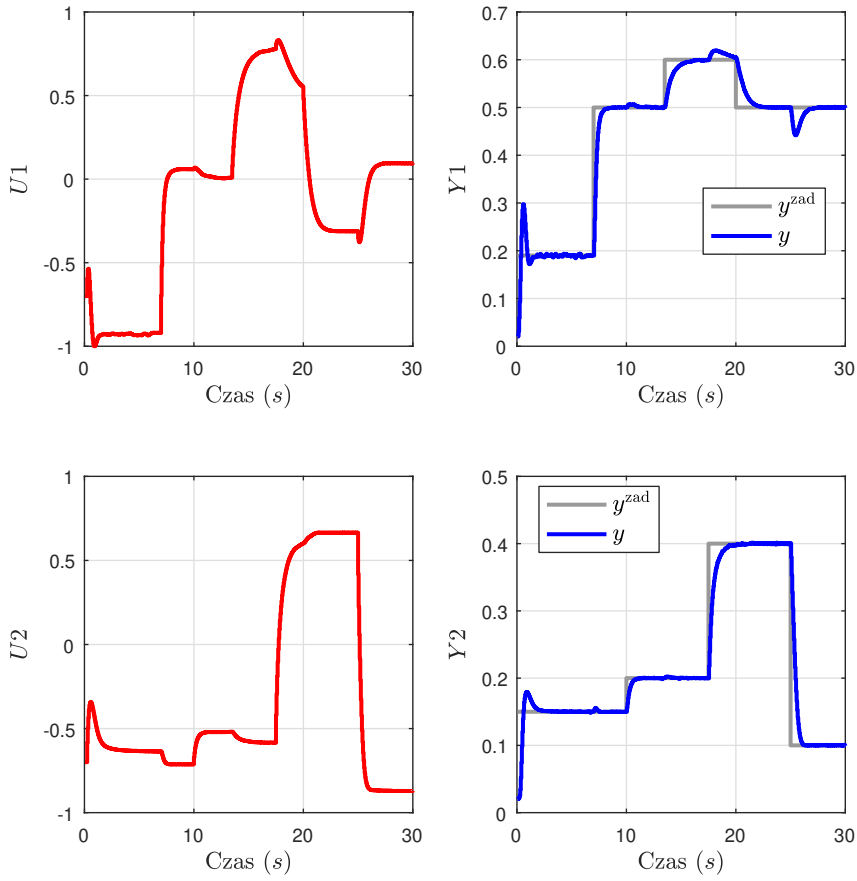
$$B_{2,1}(z^{-1}) = 0.0171z^{-1} - 0.0084z^{-2} \tag{8.7}$$

$$B_{2,2}(z^{-1}) = 0.0823z^{-1} - 0.0499z^{-2} \tag{8.8}$$



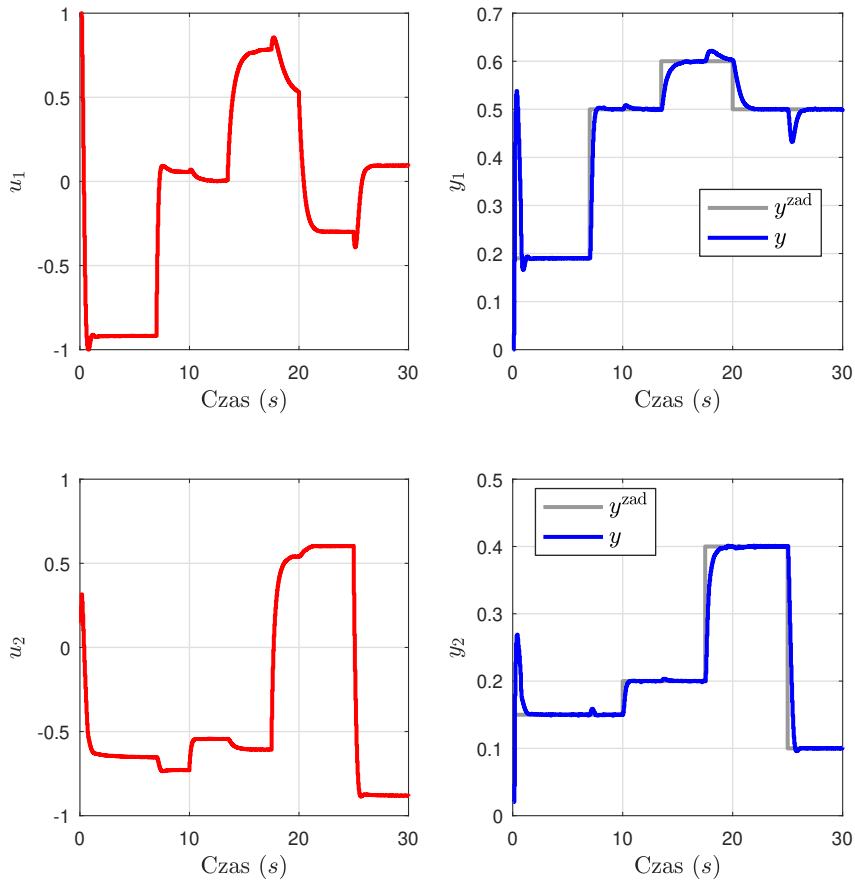
Rys. 8.12. Stanowisko świetlne: znormalizowane odpowiedzi skokowe; P – rzeczywiste odpowiedzi skokowe, M – aproksymacja

Na rys. 8.13 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej (trajektorie uzyskane w klasycznym algorytmie analitycznym są oczywiście praktycznie takie same). Zestaw parametrów jest następujący: horyzont dynamiki $D = 30$, horyzont predykcji $N = 10$, horyzont sterowania $N_{\mathrm{u}} = 5$, $\lambda_1 = \lambda_2 = \psi_1 = \psi_2 = 1$. Ich dobór został wykonany doświadczalnie w celu uzyskania dobrej jakości regulacji. Są one stosowane we wszystkich sześciu algorytmach regulacji procesu świetlnego (horyzont dynamiki tylko w algorytmie DMC). Czas potrzebny do wyznaczenia sterowania w jednej chwili próbkowania algorytmu wynosi 2,2 ms, podczas gdy w klasycznej wersji analitycznej 4,1 ms.



Rys. 8.13. Stanowisko świetlne: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej

Na rys. 8.14 przedstawiono wyniki trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu DMC. W porównaniu z algorytmami analitycznymi znacznie zwiększył się czas obliczeń, maksymalna jego wartość wynosi 20,7 ms, ale jest krótszy niż okres próbkowania (50 ms). Analogicznie jak w przypadku stanowiska grzejąco-chłodzącego, jest on znacznie dłuższy od czasu obliczeń algorytmów analitycznych, ponieważ zależy on głównie od czasu rozwiązania zadania optymalizacji kwadratowej.



Rys. 8.14. Stanowisko świetlne: trajektorie uzyskane przy wykorzystaniu numerycznego algorytmu DMC

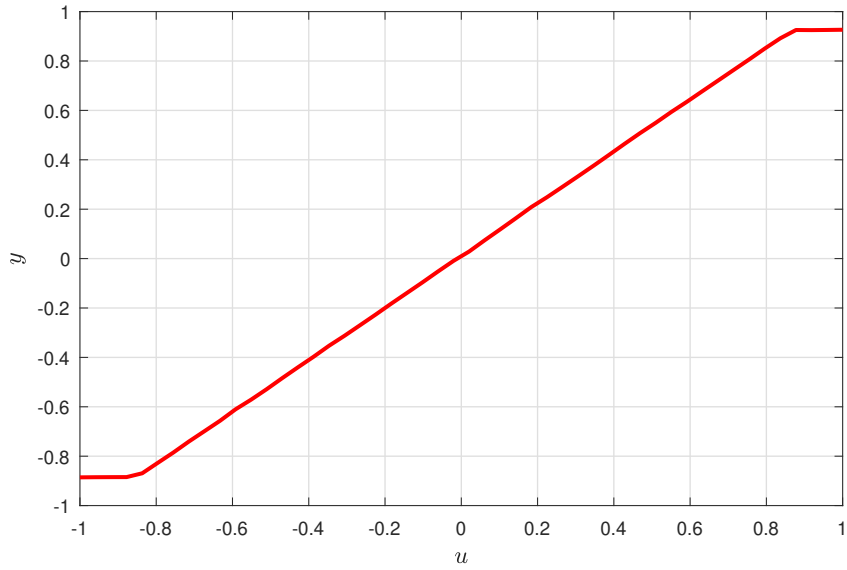
Trajektorie algorytmów GPC są bardzo podobne do trajektorii algorytmów DMC, ale dzięki zastosowaniu modelu o mniejszej liczbie parametrów, umożliwia skrócenie czasu obliczeń. W tab. 8.2 porównano czasy obliczeń wszystkich sześciu rozważanych algorytmów regulacji predykcyjnej stanowiska świetlnego. Analogicznie jak w przypadku procesu grzejąco-chłodzącego, najmniej złożone obliczeniowo są algorytmy analityczne, szczególnie w wersji oszczędnej, algorytmy numeryczne wymagają kilkukrotnie dłuższego czasu obliczeń. Dzięki małej liczbie parametrów modelu, algorytm GPC umożliwia osiągnięcie krótszych czasów obliczeń niż algorytm DMC, co jest szczególnie widoczne w algorytmach analitycznych. Czas obliczeń algorytmów numerycznych głównie zależy od czasu rozwiązania zadania optymalizacji kwadratowej, w algorytmie GPC jest on nieco krótszy niż w algorytmie DMC.

Tabela 8.2. Stanowisko świetlne: porównanie czasów obliczeń (T_{obl} (ms)) pojedynczej iteracji rozważanych algorytmów regulacji predykcyjnej

Algorytm	D	N	N_{u}	T_{obl} (ms)
DMC analityczny	30	10	5	4,1
DMC analityczny oszczędny	30	10	5	2,2
DMC numeryczny	30	10	5	20,7
GPC analityczny	–	10	5	1,5
GPC analityczny oszczędny	-	10	5	1,2
GPC numeryczny	–	10	5	18,1

8.3. Serwomechanizm

Stanowisko serwomechanizm zostało skonfigurowane jako obiekt o jednym wejściu i jednym wyjściu. Sygnałem sterującym jest wartość zadana prędkości silnika. Pomiar aktualnej wartości prędkości jest wyjściem procesu. Okres próbkowania wynosi 50 ms. Przyjęto ograniczenia sygnału sterującego: $u =< -1, 1 >$. Charakterystykę statyczną obiektu przedstawiono na rys. 8.15. Jest to charakterystyka idealnie liniowa, załamania sygnału świadczą o ograniczeniach maksymalnej prędkości w obu kierunkach obrotu. Widoczne przesunięcie w zerze wartości sterowania wynika z występowaniu miliwoltowego sygnału na wejściu wzmacniacza, który powoduje powolne obracanie się (ten sygnał można skompensować odpowiednimi parametrami w konfiguracji wzmacniacza lub korygując sygnał sterowania z systemu regulacji.

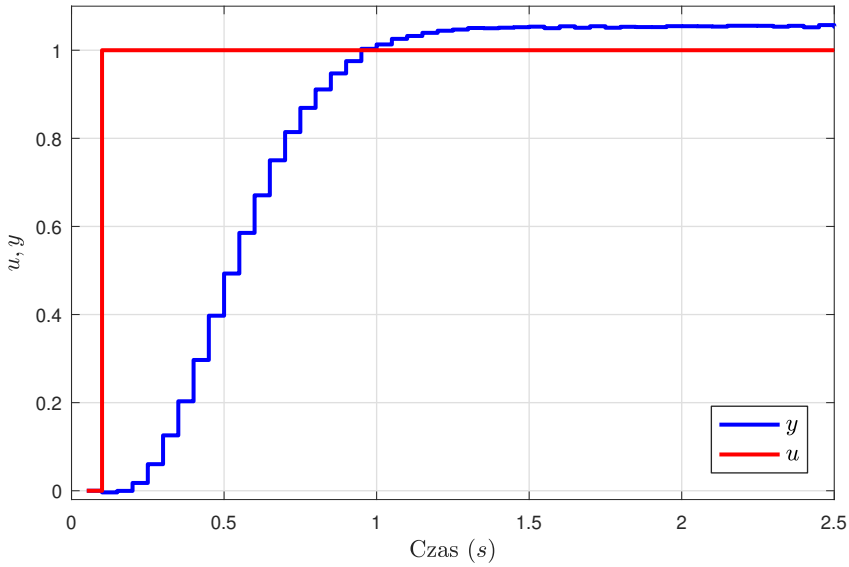


Rys. 8.15. Serwomechanizm: charakterystyka statyczna

Na rys. 8.16 przedstawiono znormalizowaną odpowiedź skokową. Widoczne jest, że horyzont dynamiki dla tej charakterystyki wynosi około 1400 ms. Wynik ten osiągnięto dla fabrycznych nastaw wbudowanych regulatorów wzmacniacza. Model obiektu można przedstawić w postaci równania różnicowego (2.55), gdzie

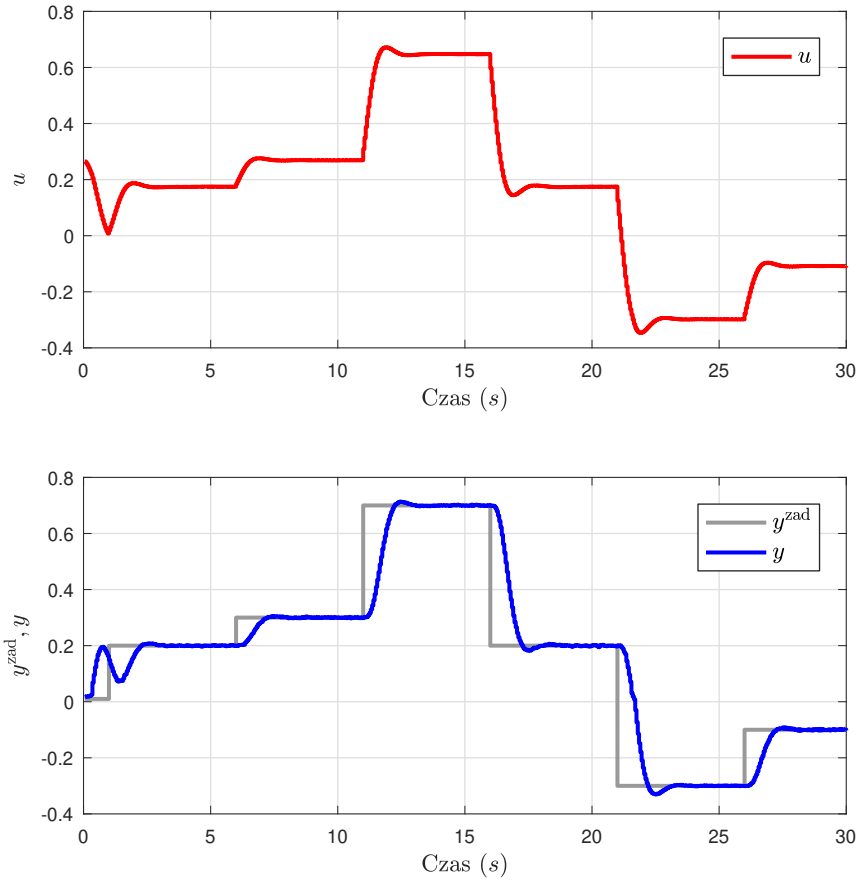
$$A_{1,1}(z^{-1}) = 1 - 3.1737z^{-1} + 4.0289z^{-2} - 2.5573z^{-3} + 0.8116z^{-4} - 0.1030z^{-5} \tag{8.9}$$

$$B_{1,1}(z^{-1}) = 0.0001z^{-1} + 0.0021z^{-2} + 0.0037z^{-3} + 0.0010z^{-4} + 0.00002z^{-5} \tag{8.10}$$



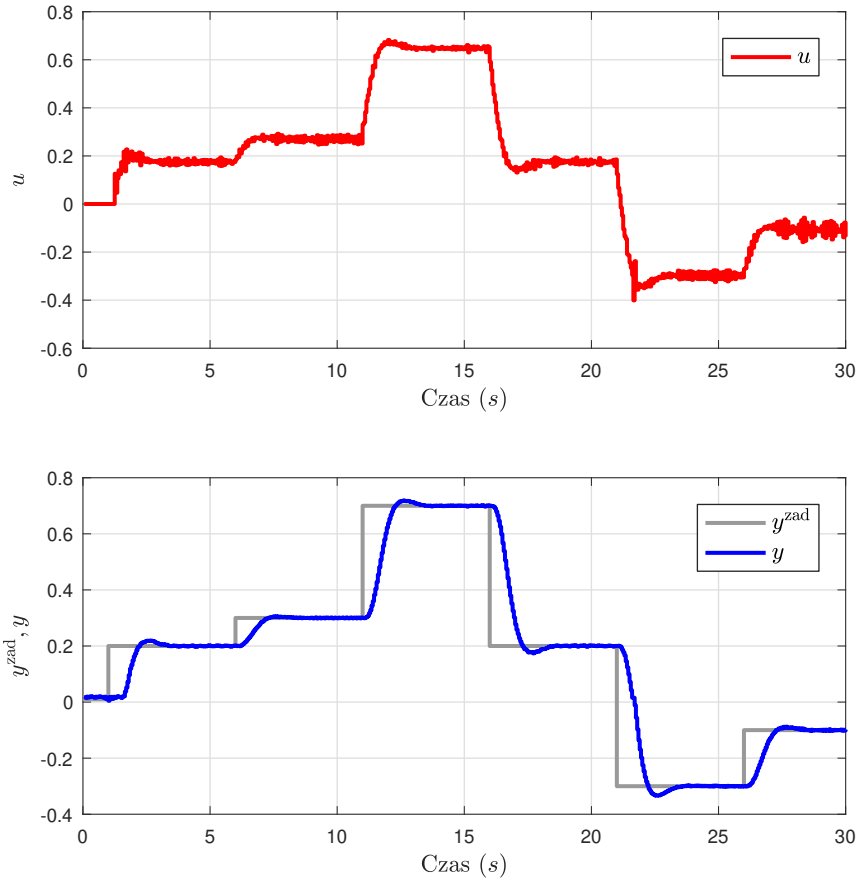
Rys. 8.16. Serwomechanizm: znormalizowana odpowiedz skokowa

Na rys. 8.17 przedstawiono trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej (trajektorie uzyskane w klasycznym algorytmie analitycznym są oczywiście praktycznie takie same). Zestaw parametrów jest następujący: horyzont dynamiki $D = 30$, horyzont predykcji $N = 5$, horyzont sterowania $N_{\mathrm{u}} = 4$, $\lambda = 1.4$ $\psi = 1$. Ich dobór został wykonany doświadczalnie w celu uzyskania dobrej jakości regulacji. Są one stosowane we wszystkich sześciu algorytmach regulacji serwomechanizmu (horyzont dynamiki tylko w algorytmie DMC). Czas obliczeń wynosi 0,2 ms, podczas gdy w klasycznym algorytmie analitycznym 0,8 ms.



Rys. 8.17. Serwomechanizm: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu DMC w wersji oszczędnej

Na rys. 8.18 przedstawiono wyniki regulacji obiektu serwomechanizm algorytmem GPC w wersji analitycznej oszczędnej. Analogicznie jak w przypadku procesu grzejąco-chłodzącego, uzyskany przebieg sygnału wyjściowego jest praktycznie taki sam jak w algorytmach DMC, ale trajektoria sygnału sterującego charakteryzuje się dużą zmiennością. Czas obliczeń algorytmu wynosi 0,1 ms, czas obliczeń klasycznej wersji analitycznej 0,4 ms.



Rys. 8.18. Serwomechanizm: trajektorie uzyskane przy wykorzystaniu analitycznego algorytmu GPC w wersji oszczędnej

W tab. 8.3 porównano czasy obliczeń wszystkich sześciu rozważanych algorytmów regulacji predykcyjnej serwomechanizmu.

Tabela 8.3. Serwomechanizm: porównanie czasów obliczeń (T_{obl} (ms)) pojedynczej iteracji rozważanych algorytmów regulacji predykcyjnej

Algorytm	D	N	N_{u}	T_{obl} (ms)
DMC analityczny	30	5	4	0,8
DMC analityczny oszczędny	30	5	4	0,2
DMC numeryczny	30	5	4	4,7
GPC analityczny	–	5	4	0,4
GPC analityczny oszczędny	–	5	4	0,1
GPC numeryczny	–	5	4	4,4

Na podstawie przeprowadzonych eksperymentów można sformułować następujące wnioski dotyczące czasu obliczeń, wykonywanych w każdej chwili próbkowania:

1. Najkrótszym czasem obliczeń charakteryzują się algorytmy regulacji predykcyjnej w wersji oszczędnej, kilkukrotnie krótszym niż klasyczne algorytmy analityczne.
2. Stosowany w algorytmach GPC model ARX ma typowo kilka razy mniej współczynników niż model odpowiedzi skokowej używany w algorytmach DMC. Dzięki temu algorytmy analityczne (klasyczny i oszczędny) GPC charakteryzują się kilkukrotnie krótszym czasem obliczeń niż odpowiadające im algorytmy DMC.
3. Algorytm analityczny GPC w wersji oszczędnej umożliwia osiągnięcie najkrótszego czasu obliczeń.
4. Algorytmy numeryczne wymagają kilkukrotnie dłuższego czasu obliczeń. Choć czas obliczeń algorytmu GPC jest krótszy niż algorytmu DMC, nie jest to różnica duża (ok. 10%). Jest to spowodowane tym, że czas obliczeń w algorytmach numerycznych głównie zależy od czasu rozwiązania zadania optymalizacji kwadratowej.
5. Czas obliczeń algorytmów analitycznych jest stały, niezależny od punktu pracy, wartości zadanych, zakłóceń itd. W algorytmach numerycznych może być jedynie oszacowany maksymalny czas obliczeń, ponieważ procedura optymalizacji kwadratowej, w zależności od punktu początkowego, charakteryzuje się zmiennym czasem wykonania.

Dla wszystkich trzech rozważanych procesów trajektorie wyjściowe, były bardzo podobne, zarówno w algorytmach DMC oraz GPC. Istotne różnice występują w obliczanych trajektoriach sygnałów sterujących: w algorytmie DMC są one łagodne, w algorytmie GPC charakteryzują się dużymi i dość gwałtownymi zmianami (któ-

re mogą być ograniczone po zwiększeniu współczynników minimalizowanej funkcji celu). Jest to spowodowane innną zależnością sygnałów sterujących od sygnałów wyjściowych procesu (wzory (2.52) i (2.73)).

Dla rozważanych procesów trajektorie uzyskane w algorytmach numeryczny i analitycznych są podobne. Istotne różnice mogą pojawić się w trudnych przypadkach wielowymiarowych oraz przy uwzględnieniu ograniczeń prognozowanych sygnałów wyjściowych.

Rozdział 9

Podsumowanie i wnioski

Niniejsza rozprawa dotyczy projektowania systemów sterujących wykorzystujących algorytmy regulacji predykcyjnej i struktury FPGA. Opracowana metodyka projektowania umożliwia sterowanie szybkich systemów dynamicznych, charakteryzujących się krótkimi okresami próbkowania. Proponowane podejście składa się z warstwy sprzętowej (projekt układów elektronicznych) oraz programowej (implementacja i testowanie), które są ze sobą ściśle związane.

W pracy rozważa się dwa najpopularniejsze algorytmy regulacji predykcyjnej: DMC oraz GPC, oba w trzech wersjach (analitycznej, analitycznej oszczędnej i numerycznej). W celu przyspieszenia obliczeń algorytmów numerycznych oszczędnych zaprojektowano specjalizowane akceleratory sprzętowe, wykorzystujące strukturę FPGA (jest to również możliwe dla klasycznych wersji analitycznych). Sygnały sterujące obliczane są szybko, w sposób całkowicie analityczny, istniejące ograniczenia sygnałów sterujących uwzględnia się rzutując uzyskane rozwiązanie na zbiór rozwiązań dopuszczalnych. Dokładne uwzględnienie różnych rodzajów ograniczeń, w tym ograniczeń prognozowanych sygnałów wyjściowych procesu, możliwe jest w wersjach numerycznych rozważanych algorytmów, do implementacji których wykorzystano procesor NIOS II.

W oryginalnej wersji algorytm DMC może być zastosowany tylko do procesów stabilnych bez całkowania, algorytm GPC nie ma takich ograniczeń i może być zastosowany np. do regulacji procesów niestabilnych. Podstawową zaletą algorytmu DMC jest bardzo prosta identyfikacja: do otrzymania modelu należy jedynie zarejestrować odpowiedź skokową procesu. Stosowany w algorytmie GPC model ARX ma typowo kilka razy mniej współczynników niż model odpowiedzi skokowej używany w algorytmie DMC. Dzięki temu algorytmy analityczne (klasyczny i oszczędny) GPC charakteryzują się kilkukrotnie krótszym czasem obliczeń niż odpowiadające im algorytmy DMC. Ze wszystkich rozważanych algorytmów, najkrótszym czasem

obliczeń charakteryzują się algorytmy regulacji predykcyjnej w wersji oszczędnej, kilkukrotnie krótszym niż klasyczne algorytmy analityczne. Algorytm analityczny GPC w wersji oszczędnej umożliwia osiągnięcie najkrótszego czasu obliczeń. Czas obliczeń algorytmów analitycznych jest stały, niezależny od punktu pracy, wartości zadanych, zakłóceń itd. W algorytmach numerycznych może być jedynie oszacowany maksymalny czas obliczeń, ponieważ procedura optymalizacji kwadratowej, w zależności od punktu początkowego, charakteryzuje się zmiennym czasem wykonania. Algorytmy numeryczne wymagają kilkukrotnie dłuższego czasu obliczeń. Choć czas obliczeń algorytmu GPC jest krótszy niż algorytmu DMC, nie jest to różnica duża (ok. 10%). Jest to spowodowane tym, że czas obliczeń w algorytmach numerycznych głównie zależy od czasu rozwiązania zadania optymalizacji kwadratowej.

Zalety opracowanego systemu sprzętowo-programowego do regulacji predykcyjnej są następujące:

1. Umożliwia regulację predykcyjną szybkich procesów dynamicznych, o okresach próbkowania rzędu milisekund.
2. Oferuje 3 wersje algorytmu DMC i 3 wersje algorytmu GPC.
3. Akceleracja sprzętowa obliczeń algorytmów analitycznych.
4. System jest elastyczny pod kątem konfiguracji sprzętowej jak i programowej.
5. Możliwa jest redundancja pracy algorytmów regulacji.
6. Projektowanie systemu zostało podzielone na etapy.
7. Warstwa sprzętowa interfejsów pozwala podłączyć typowe urządzenia występujące w przemyśle: czujniki cyfrowe, analogowe, enkodery położenia, moduły sterowania ruchem, moduły PWM.

Warto podkreślić, że autor rozprawy nie tylko zaprogramował układ FPGA, ale również samodzielnie zaprojektował i wykonał wszystkie układy elektroniczne, będące częścią opisywanego systemu do regulacji predykcyjnej. Co więcej, autor zaprojektował oraz zbudował dwa oryginalne stanowiska laboratoryjne, które służą do testowania opracowanego systemu sprzętowo-programowego: stanowisko termiczne [116, 115] (razem z dr. inż. Patrykiem Chabrem) oraz stanowisko świetlne. Pierwsze z wymienionych stanowisk (zbudowano kilka jego kopii) wykorzystywane jest w trakcie ćwiczeń laboratoryjnych z kilku przedmiotów prowadzonych w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej, kilka stanowisk zostało też sprzedanych.

Naturalnym rozszerzeniem zbudowanego systemu jest uwzględnienie innych al-

gorytmów regulacji predykcyjnej. Jako najważniejsze przykłady można podać: algorytmy bazujące na liniowych modelach w przestrzeni stanu [108], zazwyczaj wymagające obserwatorów stanu, a także algorytmy nieliniowe, bazujące na liniowych modelach procesów [68].

W trakcie prac opublikowano 6 prac naukowych: jeden artykuł w czasopiśmie *International Journal of Control Automation and Systems* (impact factor równy 2,181 (2018 r.)) [33], jeden artykuł w czasopiśmie *Pomiary Automatyka Robotyka* [115], jeden rozdział w książce [114] oraz 3 prace konferencyjne [116, 117, 118].

Bibliografia

[1] Intel cyclone iv. https://www.intel.com/content/www/us/en/programmable/products/fpga/cyclone-series/cyclone-iv/overview.tablet.html.

[2] Intel nios ii. https://www.intel.pl/content/www/pl/pl/products/programmable/processor/nios-ii.html.

[3] Intel qsys. https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-software/qsys.html.

[4] Intel quartus prime. https://www.intel.pl/content/www/pl/pl/software/programmable/quartus-prime/overview.html.

[5] Xilinx aerospace and defence. https://www.xilinx.com/applications/aerospace-and-defense/milcom.html.

[6] M. A. Abbas, R. Milman, J. M. Eklund. Obstacle avoidance in real time with nonlinear model predictive control of autonomous vehicles. *Canadian Journal of Electrical and Computer Engineering*, 40:12–22, 2017.

[7] F. C. J. Allaire, M. Tarbouchi, G. Labonte, G. Fusina. Fpga implementation of genetic algorithm for uav real-time path planning. *Journal of Intelligent and Robotic Systems*, 54:495–510, 2008.

[8] R. Amari, M. Alamir, P. Tona. Unified mpc strategy for idle-speed control, vehicle start-up and gearing applied to an automated manual transmission. 41:7079–7085, 2008.

[9] S. Antonov, L. Helsen. Robustness analysis of a hybrid ground coupled heat pump system with model predictive control. *Journal of Process Control*, 47:191–200, 2016.

[10] M. R. Arahal, M. Berenguel, E. F. Camacho. Neural identification applied to predictive control of a solar plant. *Control Engineering Practice*, 3:333–344, 1998.

[11] E. Aridhi, M. Abbes, A. Mami. Fpga implementation of predictive control. *2012 16th IEEE Mediterranean Electrotechnical Conference*, strony 191–196, Yasmine Hammamet, Tunisia, 2012.

[12] H. Asadipooya, A. A. Safavi. Enhancement of model predictive control implementation on a dcs pcs7. *2016 24th Iranian Conference on Electrical Engineering (ICEE)*, strony 1272–1277, 2016.

[13] T. Baca, G. Loianno, M. Saska. Embedded model predictive control of unmanned micro aerial vehicles. *The 21st International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje*, strony 992–997, 2016.

[14] K. Basterretxea, K. Benkrid. Embedded high-speed model predictive controller on a fpga. *NASA/ESA Conference on Adaptive Hardware and Systems*, strony 327–335, 2011.

[15] T. Bellemans, B. Schutter, B. Moor. Model predictive control for ramp metering of motorway traffic: a case study. *Control Engineering Practice*, 7:757–767, 2006.

[16] A. Bemporad, M. Morari, V. Dua, E. N. Pistikopoulos. The explicit linear quadratic regulator forconstrained systems. *Automatica*, 38:3–20, 2002.

[17] M. Berenguel, M. R. Arahal, E. F. Camacho. Modelling the free response of a solar plant for predictive control. *Control Engineering Practice*, 10:1257–1266, 1998.

[18] E. Blanco, C. Prada, S. Cristea, J. Casas. Nonlinear predictive control in the lhc accelerator. *Control Engineering Practice*, 10:1136–1147, 2009.

[19] C. Bordons, E. F. Camacho. Applications of model predictive controllers in a sugar factory. *IFAC Proceedings Volumes*, 33:329–334, 2000.

[20] M. A. Brdys, M. Grochowski, T. Gminski, K. Konarczak, M. Drewa. Model predictive control of a rotary cement kiln. *Control Engineering Practice*, 6:751–767, 2008.

[21] E. F. Camacho, C. Bordons. *Model Predictive Control*. Springer, London, 1999.

[22] M. d. M. Castilla, J. Alvarez, F. Rodriguez, M. Berenguel. *Comfort control in buildings*. Springer, Cham, 2014.

[23] P. Chaber, M. Ławryńczuk. Automatic: Code generation of model predictive control algorithms for microcontrollers. *IEEE Transactions on Industrial Informatics*. Przyję do publikacji.

[24] P. Chaber, M. Ławryńczuk. Implementation of analytical generalized predictive controller for very fast applications using microcontrollers: Preliminary results. W. Mitkowski, J. Kacprzyk, K. Oprzędkiewicz, P. Skruch, redaktorzy, *Trends in Advanced Intelligent Control, Optimization and Automation*, strony 378–387, Cham, 2017. Springer International Publishing.

[25] P. Chaber, M. Ławryńczuk. Fast analytical model predictive controllers and their implementation for STM32 ARM microcontroller. *IEEE Transactions on Industrial Informatics*, 15:4580–45900, 2019.

[26] J. Ciganek, M. Kocur, S. Kozak. Hardware realization of advanced controller design methods using fpga. *IFAC-PapersOnLine*, 49:163–168, 2016.

[27] D. W. Clarke, C. Mohtadi, P. S. Tuffs. Generalized predictive control. *Automatica*, 23:137–160, 1987.

[28] G. Colin, Y. Chamaillard, G. Bloch, G. Corde. Neural control of fast nonlinear

systems–application to a turbocharged si engine with vct. *IEEE Transactions on Neural Networks*, 4:1101–1114, 2007.

[29] S. Coric, M. Leeser, E. Miller, M. Trepanier. Parallel-beam backprojection: An fpga implementation optimized for medical imaging. *Journal of VLSI signal processing systems for signal, image and video technology*, 39:295–311, 2002.

[30] S. Craven, P. Athanas. Examining the viability of fpga supercomputing. *EURASIP Journal on Embedded Systems*, 2007:093652, 2006.

[31] C. R. Cutler, B. L. Ramaker. Dynamic matrix control – a computer control algorithm. *Joint Automatic Control Conference*, 17:72, 1980.

[32] K. Czerwiński, M. Ławryńczuk. Dynamic matrix control algorithm implementation on arm cortex-r5 mcu: Performance analysis. *IFAC-PapersOnLine*, 51:330–335, 2018.

[33] K. Czerwiński, A. Wojtulewicz, M. Ławryńczuk. Fuzzy controller for laboratory levitation system: Real-time experiments using programmable logic controller. *International Journal of Control Automation and Systems*, 17:1507–1514, 2019.

[34] P. Dhinakaran, D. Manamalli. Novel strategies in the model-based optimization and control of permanent magnet dc motors. *Computers and Electrical Engineering*, 44:34–41, 2015.

[35] R. Dubay. Optimizing mpc of melt temperature in injection moulding. *ISA Transactions*, 41:81–94, 2002.

[36] M. D. Edwards, J. Forrest, A. E. Whelan. Acceleration of software algorithms using hard-ware/software co-design techniques. *J. of Systems and Architecture*, 42:697–707, 1996.

[37] C. Ekaputri, A. Syaichu-Rohman. Model predictive control (mpc) design and implementation using algorithm-3 on board spartan 6 fpga sp605 evaluation kit. strony 115–120, 2013.

[38] T. Englert, K. Graichen. Nonlinear model predictive torque control of pmsms for high performance applications. *Control Engineering Practice*, 81:43–54, 2018.

[39] R. C. Ferguson, R. Tate. Use of field programmable gate array technology in future space avionics. *24th Digital Avionics Systems Conference*, Washington, DC, USA, 2005.

[40] H. J. Ferreau, T. Kraus, M. Vukov, W. Saeys, M. Diehl. High-speed moving horizon estimation based on automatic code generation. *The 51th IEEE Conference on Decision and Control (CDC 2012)*, Maui, HI, USA, 2012.

[41] M. Fikar, M. Kvasnica. Real-time model predictive control of a fan heater via plc. *The 18th International Conference on Process Control*, 2011.

[42] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, M. Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. 2013.

[43] E. Gallestey, M. Stalder. Advancing system 800xa. *ABB review*, 3:52–59, 2014.

[44] D. Gorni, M. del Mar Castilla, A. Visioli. Model predictive control of a rotary cement kiln. *Building and Environment*, 103:86–98, 2016.

[45] J. K. Gruber, M. Doll, C. Bordons. Design and experimental validation of a constrained mpc for the air feed of a fuel cell. *Control Engineering Practice*, 8:874–885, 2009.

[46] J. K. Gruber, J. L. Guzman, F. Rodriguez, C. Bordons, M. Berenguel, J. A. Sanchez. Nonlinear mpc based on a volterra series model for greenhouse temperature control using natural ventilation. *Control Engineering Practice*, 4:354–366, 2011.

[47] M. Gulan, G. Takács, N. A. Nguyen, S. Olaru, P. Rodriguez-Ayerbe, B. Rohal'-Ilkiv. Efficient embedded model predictive vibration control via convex lifting. *IEEE Transactions on Control Systems Technology*, 27:1–15, 2017.

[48] T. Guneysu, T. Kasper, M. Novotny, C. Paar, A. Rupp. Cryptanalysis with copacobana. *IEEE Transactions on Computers*, 57:498–1513, 2008.

[49] S. Habinc. Suitability of reprogrammable fpgas in space applications. Raport instytutowy, Gaisler Research, 2002.

[50] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, G. E. Constantinides. Predictive control using an fpga with application to aircraft control. *IEEE Transactions on Control Systems Technology*, 22:1006–1017, 2014.

[51] I. M.-B. Hassine, M. W. Naouar, N. Mrabet-Bellaaj. Predictive control strategies for wind turbine system based on permanent magnet synchronous generator. *ISA Transactions*, 62:73–80, 2016.

[52] T. A. Hauge, R. Slora, B. Lie. Application and roll-out of infinite horizon mpc employing a nonlinear mechanistic model to paper machines. *Journal of Process Control*, 15:201–213, 2005.

[53] M. Herceg, M. Kvasnica, C. N. Jones, M. Morari. Multi-parametric toolbox 3.0. *European Control Conference (ECC), Zürich, Szwajcaria*, strony 502–510, 2013.

[54] B. Houska, H. J. Ferreau, M. Diehl. ACADO toolkit – an open source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32:298–312, 2011.

[55] B. Huyck, J. Brabanter, B. Moor, J. Impe, F. Logist. Online model predictive control of industrial processes using low levelcontrol hardware: A pilot-scale distillation column case study. *Control Engineering Practice*, 28:34–48, 2013.

[56] B. Huyck, H. J. Ferreau, M. Diehl, J. De Brabatner, J. F. M. Van Impe, B. De Moor, F. Logist. Towards online model predictive control on a programmable logic controller: Practical considerations. *Mathematical Problems in Engineering*, 2012.

[57] G. Jarmoszewicz, S. Plamowski, K. Świrski. Zaawansowane algorytmy optymalizacji i regulacji jako integralna czesc cyfrowego systemu automatyki (DCS) - stan obecny

i przyklad implementacji. *Kontrola, Sterowanie i Automatyzacja Procesu Spalania w Kotłach Energetycznych*, strony 43–61, 2005.

[58] M. Jaworski. *Rozprawa doktorska: Wykorzystanie zasobów FPGA do budowy szybkich generatorów pseudolosowych*. Politechnika Poznańska, 2016.

[59] R. M. Jiang, D. Crookes. Fpga implementation of 3d discrete wavelet transform for real-time medical imaging. *2007 18th European Conference on Circuit Theory and Design*, strony 519–522, Seville, Spain, 2007.

[60] T. A. Johansen, W. Jackson, R. Schreiber, P. Tondel. Hardware synthesis of explicit model predictive controllers. *IEEE Transactions on Control Systems Technology*, 15:191–197, 2007.

[61] B. Kapernick, K. Graichen. Plc implementation of a nonlinear model predictive controller. strony 1892–1897, 2014.

[62] M. H. Khooban, N. Vafamand, T. Niknam. T–s fuzzy model predictive speed control of electrical vehicles. *ISA Transactions*, 64:231–240, 2016.

[63] G. Knagge, A. Wills, A. Mills, B. Ninness. Asic and fpga implementation strategies for model predictive control. *2009 European Control Conference (ECC)*, strony 144–149, Budapest, Hungary, 2009.

[64] K. Kunz, S. M. Huck, T. H. Summers. Fast model predictive control of miniature helicopters. *European Control Conference (ECC), Zürich, Szwajcaria*, strony 1205–1210, 2012.

[65] M. Kvasnica, I. Raunova, M. Fikar. Automatic code generation for real-time implementation of model predictive control. *2010 IEEE International Symposium on Computer-Aided Control System Design, Jokohama, Japonia*, strony 993–998, 2010.

[66] J. Lan, D. Li, Y. Xi. Implementation of dynamic matrix control on fpga. *Proceedings of the 29th Chinese Control Conference*, strony 5970–5974, Beijing, China, 2010.

[67] M. K. S. Lau, S. P. Yue, K. V. Ling, J. M. Maciejowski. A comparison of interior point and active set methods for FPGA implementation of model predictive control. *European Control Conference (ECC), Budapeszt, Węgry*, strony 156–161, 200.

[68] M. Ławryńczuk. *Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach*, wolumen 3 serii *Studies in Systems, Decision and Control*. Springer, Heidelberg, 2014.

[69] M. Ławryńczuk. Nonlinear predictive control of a boiler-turbine unit: A state-space approach with successive on-line model linearisation and quadratic optimisation. *ISA Transactions*, 67:476–495, 2017.

[70] M. Ławryńczuk, P. Marusak, P. Tatjewski. Efficient predictive control algorithms based on soft computing approaches: application to glucose concentration stabilization. *Novel Algorithms and Techniques in Telecommunications, Automation and Industrial Electronics*, strony 425–430, 2010.

[71] M. Ławryńczuk, P. Tatjewski. Efektywny obliczeniowo algorytm wielowymiaro-

wej regulacji predykcyjnej dla modeli typu wejscie-wyjscie. *Pomiary Automatyka Kontrola*, 49:10–14, 2003.

[72] D. Li, N. Yang, R. Niu, H. Qiu, Y. Xi. Fpga based qdmc control for reverse-osmosis water desalination system. *Desalination*, 285:83–90, 2012.

[73] K. V. Ling, B. F. Wu, J. M. Maciejowski. Embedded model predictive control (mpc) using a fpga. *IFAC Proceedings Volumes*, wolumen 41, strony 15250–15255, 2008.

[74] K. V. Ling, S. P. Yue, J. M. Maciejowski. A fpga implementation of model predictive control. *2006 American Control Conference*, strony 6 pp.–, Minneapolis, MN, USA, 2006.

[75] L. Liniger, A. Domahidi, M. Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36:628–647, 2015.

[76] Maciejowski J. M. *Predictive control with constraints*. Prentice Hall, Harlow, 2009.

[77] M. Mahfouf, D. A. Linkens. Non-linear generalized predictive control (nlgpc) applied to muscle relaxant anaesthesia. *International Journal of Control*, 2:239–257, 1998.

[78] B. Marami, M. Haeri. Implementation of mpc as an aqm controller. *Implementation of MPC as an AQM controller*, 2:227–239, 2010.

[79] J. Mattingley, S. Boyd. Cvxgen: a code generator for embedded convex optimization. 2012.

[80] U. Meyer Baese. *DSP with FPGAs*. Springer, Berlin, 2001.

[81] J. G. Ortega, E. F. Camacho. Mobile robot navigation in a partially structured static environment, using neural predictive control. *Control Engineering Practice*, 12:1669–1679, 1996.

[82] R. A. Osornio-Rios, R. Romero-Troncoso, G. Herrera-Ruiz, R. Castaneda-Miranda. The application of reconfigurable logic to high speed cnc milling machines controllers. *Control Engneering Practice*, 16:674–684, 2007.

[83] K. K. Parchi, T. Nishitani. *Digital Signal Processing for Multimedia Systems*. Marcel Dekker, New York, 1999.

[84] M. Petko, G. Karpiel, T. Uhl. Implementacja algorytmów sterowania w układach fpga na przykładzie robota równoleglego. *Pomiary Automatyka Kontrola*, 5:27–30, 2006.

[85] M. Petko, M. Staworko, M. Lubieniecki. Automatyczna implementacja programowo-sprzętowa algorytmów sterowania w układach fpga. *Pomiary, Automatyka, Kontrola*, 55:297–300, 2009.

[86] S. Plamowski. Implementation of dmc algorithm in embedded controller - resources, memory and numerical modifications. *Advances in Intelligent Systems and Computing*, 577:335–343, 2017.

[87] S. Plamowski. Perspectives of dcs and scada systems in high-energy physics experiments. *Acta Physica Polonica B Proceedings Supplement*, 11:681–684, 2018.

[88] Aang Pml. *Distributed Control Systems*. Process Control in the Chemical Industries, Chemical Engineering Department, King Saud University, 2002.

[89] P. Puton. Opracowanie systemu cyfrowego realizującego sterownik PLC w oparciu o układy FPGA. implementacja oprogramowania narzędziowego do realizacji programów wykonawczych. Praca magisterska, 2018.

[90] S. J. Qin, T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[91] Z. Razavinasab, M. Farsangi, M. Barkhordari. Robust output feedback distributed model predictive control of networked systems with communication delays in the presence of disturbance. *ISA Transactions*, 80:12–21, 2018.

[92] C. A. Rietz, D. K. Rollins. Implementation of a predictive modeling technique on a dcs. *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, wolumen 5, strony 2951–2955 vol.5, 1998.

[93] M. Romero, A. P. Madrid, C. Manoso, V. Milanes. Low speed hybrid generalized predictive control. *ISA Transactions*, 57:373–381, 2015.

[94] T. Salsbury, P. Mhaskar, S. J. Qin. Predictive control methods to improve energy efficiency and reduce demand in buildings. *Computers and Chemical Engineering*, 51:77–85, 2013.

[95] D. Sarabia, F. Capraro, L. F. S. Larsen, C. Prada. Hybrid nmpc of supermarket display cases. *Journal of Process Control*, 4:428–441, 2009.

[96] P. Shakouri, A. Ordys. Nonlinear model predictive control approach in design of adaptive cruise control with automated switching to cruise control. *Control Engineering Practice*, 26:160–177, 2014.

[97] R. Śmierzchalski, M. Kwiesielewicz, M. Szymański, R. Sutton. Predictive control and dynamic planning of an autonomous underwater vehicle. *IFAC Proceedings Volumes*, 34:161–166, 2001.

[98] D. N. Sonawane, M. S. Sutaone, I. Malek. Systolic architecture for integer point matrix multiplication using fpga. *2009 4th IEEE Conference on Industrial Electronics and Applications, Xi'an, Chiny*, strony 3822–3825, 2009.

[99] U. Sridhar, A. Govindarajan, R. Russell. Demonstration of leapfrogging for implementing nonlinear model predictive control on a heat exchanger. *ISA Transactions*, 60:218–227, 2016.

[100] U. Sridhar, A. Govindarajan, R. Russell. Application of multi-model switching predictive functional control on the temperature system of an electric heating furnace. *ISA Transactions*, 68:287–292, 2017.

[101] K. S. Stadler, J. Poland, E. Gallestey. Model predictive control of a rotary cement kiln. *Journal of Process Control*, 1:1–9, 2011.

[102] D. Sturzenegger, D. Gyalistras, M. Morari, R. Smith. Model predictive climate control of a swiss office building: implementation, results, and cost benefit analysis. *IEEE Transactions on Control system technology*, 24:1–12, 2016.

[103] A. Suardi, S. Longo, E. C. Kerrigan, G. A. Constantinides. Explicit mpc: Hard constraint satisfation under low precision arithmetic. *Control Engineering Practice*, 47:60–69, 2016.

[104] A. Taheri, M. H. Zhalebaghi. A new model predictive control algorithm by reducing the computing time of cost function minimization for npc inverter in three-phase power grids. *ISA Transactions*, 71:391–402, 2017.

[105] G. Takács, G. Batista, M. Gulan, B. Rohal'-Ilkiv. Embedded explicit model predictive vibration control. *Mechatronics*, 36:54–62, 2016.

[106] J. Tarnawski. Implementacja algorytmu regulacji predykcyjnej mpc w sterownikach programowalnych. *Pomiary Automatyka Robotyka*, 17:100–107, 2013.

[107] P. Tatjewski. *Advanced control of industrial processes, structures and algorithms*. Springer, London, 2007.

[108] P. Tatjewski. *Sterowanie zaawansowane obiektow przemyslowych. Struktury i algorytmy. Wydanie drugie zmienione*. Akademicka Oficyna Wydawnicza EXIT, 2016.

[109] S. Thielemans, T. J. Vyncke, M. Jacxsens, J. A. Melkebeek. Fpga implementation of online finite-set model based predictive control for power electronics. *2011 Workshop on Predictive Control of Electrical Drives and Power Electronics, Monachium, Niemcy*, strony 63–69, 2011.

[110] Z. Trajanoski, P. Wach. Neural predictive control for insulin delivery using the subcutaneous route. *Journal of Process Control*, 9:1122–1134, 1998.

[111] G. Valencia-Palomo, J. A. Rossiter. Programmable logic controller implementation of an auto-tuned predictive control based on minimal plant information. *ISA Transactions*, 50:92–100, 2011.

[112] K. Varnavas, W. H. Sims, J. Casas. The use of field programmable gate arrays (FPGA) in small satellite communication systems. Raport instytutowy, NASA, 2015.

[113] R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, C. Kitchen. An overview of fpgas and fpga programming; initial experiences at daresbury. *Council for the Central Laboratory of the Research Councils*, 2006.

[114] A. Wojtulewicz. Implementation of dynamic matrix control algorithm using field programmable gate array: Preliminary results. W. Mitkowski, J. Kacprzyk, K. Oprzędkiewicz, P. Skruch, redaktorzy, *Trends in Advanced Intelligent Control, Optimization and Automation*, strony 325–334. Springer International Publishing, 2017.

[115] A. Wojtulewicz, P. Chaber, M. Ławryńczuk. Stanowisko laboratoryjne do badania

wielowymiarowych algorytmów regulacji. *Pomiary Automatyka Robotyka*, 19:15–20, 2015.

[116] A. Wojtulewicz, P. Chaber, M. Ławryńczuk. Multiple-input multiple-output laboratory stand for process control education. *The 21st International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje*, strony 466–471, 2016.

[117] A. Wojtulewicz, M. Ławryńczuk. Computationally efficient implementation of dynamic matrix control algorithm for very fast processes using programmable logic controller. *The 23rd International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje*, strony 579–584, 2018.

[118] A. Wojtulewicz, M. Ławryńczuk. Implementation of multiple-input multiple-output dynamic matrix control algorithm for fast processes using field programmable gate array. *IFAC-PapersOnLine*, 51:324–329, 2018.

[119] F. Xu, H. Chen, W. Jin, Y. Xu. FPGA implementation of nonlinear model predictive control. *2014 26th Chinese Control and Decision Conference (CCDC), Changsha, Chiny*, strony 108–113, 2014.

[120] A. S. Yamashita, A. C. Zanin, D. Odloak. Tuning the model predictive control of a crude distillation unit. *ISA Transactions*, 60:178–190, 2016.

[121] C. Yang, V. Vyatkin. Design and validation of distributed control with decentralized intelligence in process industries: A survey. *The IEEE International Conference on Industrial Informatics*, strony 1395–1400, 2008.

[122] N. Yang, D. Li, J. Zhang, Y. Xi. Model predictive controller design and implementation on FPGA with application to motor servo system. *Control Engineering Practice*, 20:1229–1235, 2014.

[123] S. Yang, M. P. Wan, B. F. Ng, T. Zhang, S. Babu, Z. Zhang, W. Chen, S. Dubey. A state-space thermal model incorporating humidity and thermal comfort for model predictive control in buildings. *Energy and Buildings*, 170:25–39, 2018.

[124] W. Zhao, T. H. Go. Quadcopter formation flighht control combining mpc and robust feedback linearization. *Journal of the Franklin Institute*, 351:1335–1355, 2013.

[125] G. Y. Zhu, M. A. Henson, L. Megan. Model predictive control of a rotary cement kiln. *Journal of Process Control*, 2:129–148, 2001.

[126] P. Zometa, M. Kögel, T. Faulwasser, R. Findeisen. Implementation aspects of model predictive control for embedded systems. *American Control Conference (ACC), Montreal, Kanada*, strony 1205–1210, 2012.