Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Mikroelektroniki i Optoelektroniki

Praca dyplomowa inżynierska

na kierunku Elektronika
w specjalności Elektronika i Inżynieria Komputerowa

Optymalizacja modułu sterowania i obliczania estymat
dla iteracyjnego przetwornika A/C

Aleksandra Antoszewska
Numer albumu 300292

promotor
dr inż. Zbigniew Jaworski

WARSZAWA 2022

Optymalizacja modułu sterowania i obliczania estymat dla iteracyjnego przetwornika A/C

Streszczenie.
Niniejsza praca jest poświęcona zagadnieniu projektowania modułu sterowania i obliczania estymat dla iteracyjnego przetwornika A/C. Zostały w niej przedstawione dwa podejścia: klasyczne oraz adaptacyjne. W ramach pracy zaimplementowano w języku VHDL modele przetworników A/C o budowie modułowej, składające się z części cyfrowej, odpowiedzialnej za sterowanie, a także części symulującej zachowanie układu analogowego. Zbadano wpływ liczby wykonywanych iteracji na parametry modeli przetworników, ze szczególnym uwzględnieniem efektywnej liczby bitów.

Słowa kluczowe: przetwornik A/C, optymalizacja, algorytmy adaptacyjne

Optimisation of control unit and calculation for iterative ADC

Abstract.
The following paper is devoted to the designing and optimisation of control unit for iterative ADC. Two algorithms were discussed: classical and adaptive. Models of iterative ADC, consisting of digital part, responsible for controlling the device and analog part's behavioral symulation were implemented as well. The influence of number of iterations for ADC's parameters, with particular reference to effective number of bits was analysed.

Keywords: analog-to-digital converter, optimisation, adaptive algorithms

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

Warszawa, 16.01.2022 r.
miejscowość i data

Aleksandra Antoszewska
imię i nazwisko studenta
300292
numer albumu
Elektronika
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Aleksandra Antoszewska
czytelny podpis studenta

Załącznik nr 3 do zarządzenia nr 109 /2021
Rektora PW z dnia 9 listopada 2021 r.

„załącznik nr 9 do zarządzenia nr 42 /2020 Rektora PW

Logo PW **Politechnika Warszawska**

Warszawa, 29.01.2022 r.
miejscowość i data

Aleksandra Antoszewska
imię i nazwisko studenta
300292
numer albumu
MEiTI, Elektronika
Wydział i kierunek studiów

Oświadczenie studenta w przedmiocie udzielenia licencji
Politechnice Warszawskiej

Oświadczam, że jako autor/współautor* pracy dyplomowej pt. Optymalizacja modułu sterowania i obliczania estymat dla iteracyjnego przetwornika A/ udzielam/~~nie udzielam*~~ Politechnice Warszawskiej nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elektronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym Politechniki Warszawskiej osobom zainteresowanym.

Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości lub w części, utrwalania w innej formie czy zwielokrotniania.

Aleksandra Antoszewska
czytelny podpis studenta

* niepotrzebne skreślić ".

Spis treści

1. Wstęp

Przetworniki A/C są istotnym elementem elektronicznym w wielu współczesnych układach. Od dziesiątek lat ludzie dążą do optymalizacji ich działania oraz uzyskania jak najlepszego stosunku jakości przetwornika do kosztu jego wytworzenia. Idealny przetwornik powinien cechować się dużą rozdzielczością, wysokim stosunkiem sygnału do szumu oraz możliwie jak najwyższą efektywną liczbą bitów.

Obecnie na rynku jest dostępne wiele typów przetworników A/C, jednak nadal dąży się do ulepszania istniejących już rozwiązań. Niewątpliwą innowacją okazały się przetworniki cykliczne. Są one niejako „wzbogacone" o część cyfrową, dzięki której wynikowa rozdzielczość takiego przetwornika jest większa niż wskazywałyby na to parametry wewnętrznego przetwornika A/C będącego częścią układu [3]. Odbywa się to kosztem prędkości działania – wykonanie trzech iteracji zajmuje więcej czasu niż jednokrotne użycie przetwornika o niskiej rozdzielczości.

Powstało wiele wariantów działania cyklicznych przetworników A/C. Niniejsza praca skupia się na implementacji oraz zbadaniu poprawności działania zaledwie ułamka rozwiązań opisanych w literaturze.

Cele i założenia pracy:

- implementacja cyfrowego bloku sterowania i obliczania estymat dla klasycznego oraz adaptacyjnego cyklicznego przetwornika A/C;
- opracowanie modelu części analogowej i zintegrowanie jej z blokiem cyfrowym;
- zbadanie poprawności działania zaimplementowanych algorytmów;
- zbadanie wpływu liczby wykonywanych iteracji na wartość efektywnej liczby bitów w kodzie estymaty.

Spodziewanym efektem realizacji pracy są dwa modele cyfrowego bloku sterowania cyklicznego przetwornika A/C, które można wykorzystać przy tworzeniu rzeczywistego przetwornika o budowie modułowej.

2. Zagadnienia teoretyczne

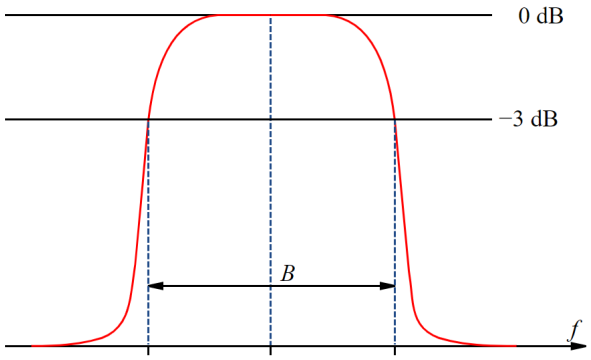
2.1. Przetwornik A/C

Cykl działania podstawowego przetwornika A/C może być podzielony na sekwencję następujących po sobie kroków. Można wyróżnić:

- filtrowanie antyaliasingowe;
- próbkowanie;
- kwantyzacja;
- kodowanie danych.

Filtr antyaliasingowy

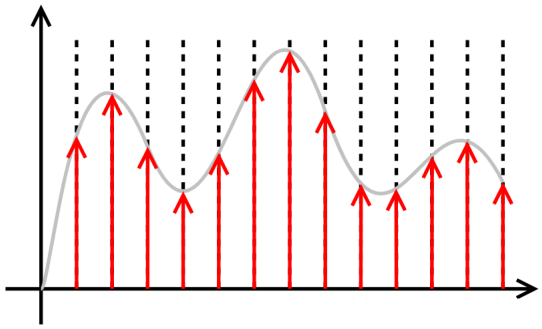
Zastosowanie na wejściu układu filtra antyaliasingowego stanowi zabezpieczenie sygnału przed niechcianymi interferencjami poza pasmem, na którym przenoszone są informacje. Aby uniknąć zjawiska aliasingu należy użyć filtru dolnoprzepustowego lub pasmowoprzepustowego [1].



Rys. 2.1. Przykład charakterystyki przenoszenia filtru pasmowoprzepustowego.

Próbkowanie

Zadaniem układu próbkującego jest przekształcenie ciągłego sygnału wejściowego na sygnał dyskretny, o skończonej liczbie próbek. Próbkowanie odbywa się z pewną częstotliwością, której wartość musi spełniać warunek twierdzenia o próbkowaniu, zwany warunkiem Nyquista. Mówi on, że częstotliwość próbkowania musi być niemniejsza niż dwukrotna wartość szerokości pasma sygnału przed spróbkowaniem [1].



Rys. 2.2. Przykład próbkowania idealnego.

Kwantyzacja

Kwantyzacja amplitudy polega na zamianie wartości próbek z dziedziny ciągłej na dyskretną. Zakres układu kwantyzującego jest dzielony na określoną liczbę części, tworząc zbiór przedziałów analogowych amplitud. Wartość próbki wejściowej jest modyfikowana do wartości odpowiadającej przedziałowi, w którym się znajduje. Wartości odpowiadające przedziałom mogą być liczbą odpowiadającą jego środkowi, a także jego górnej lub dolnej granicy [1].

Tworzenie kodów estymaty

Ostatnim etapem pracy przetwornika A/C jest zapisanie skwantyzowanych danych w postaci kodów estymaty. Jednym z często stosowanych i najprostszych sposobów kodowania jest USB (*ang. Unipolar Straight Binary*). Najniższy poziom kwantyzacji jest określony jako:

$$-V_{ref} + \frac{1}{2} V_{LSB} \tag{2.1}$$

Odpowiadający mu kod estymaty składa się z samych zer. Wraz z kolejnymi poziomami kwantyzacji zmieniają się kody estymaty, każdy kolejny to inkrementacja poprzedniego. Kod estymaty odpowiadający najwyższemu poziomowi kwantyzacji składa się z samych jedynek, a poziom ten jest określony jako:

$$V_{ref} + \frac{1}{2} V_{LSB} \tag{2.2}$$

Skala kwantyzacji to $-V_{ref} ... + V_{ref}$ [1].

2.2. Podstawowe rodzaje przetworników A/C

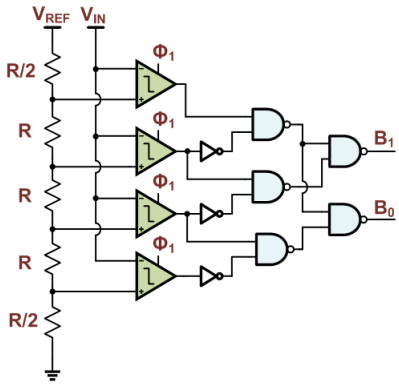
Przedstawiony powyżej cykl działania stanowi swego rodzaju uproszczenie, istnieje znacznie więcej niż jeden typ przetworników A/C. Do najpopularniejszych należą:

- przetwornik o porównaniu bezpośrednim;
- przetwornik z sukcesywną aproksymacją;
- przetwornik sigma-delta;
- przetwornik cykliczny.

2.2.1. Przetwornik o porównaniu bezpośrednim

Przetwornik o porównaniu bezpośrednim, nazywany również przetwornikiem *flash*, działa na zasadzie porównywania wartości napięcia wejściowego z napięciami odniesienia za pomocą komparatorów analogowych. Wynik operacji porównania jest zamieniany na ciąg binarny, będący kodem estymaty.

Ten typ przetworników A/C cechuje się znacznie wyższą szybkością działania niż inne rodzaje, różnica może sięgać nawet kilku rzędów wielkości. Takie przetworniki nie sprawdzają się, gdy jest wymagana duża rozdzielczość - zwiększenie jej o jeden bit wymaga podwojenia liczby użytych komparatorów oraz wymusza zwiększenie precyzji określenia poziomów napięcia odniesienia [1].



Rys. 2.3. Przykład budowy 2-bitowego przetwornika A/C o porównaniu bezpośrednim.

2.2.2. Przetwornik z sukcesywną aproksymacją

Przetwornik z sukcesywną aproksymacją wykorzystuje algorytm wyszukiwania binarnego (*ang. binary search*). Wartość napięcia wejściowego jest porównywana z napięciem odniesienia pochodzącym z wewnętrznego przetwornika C/A, za działanie którego odpowiada układ sterujący. Cykl działania jest podzielony na iteracje: w każdej z nich układ sterujący ustawia wartość '1' na kolejnych bitach kodu estymaty, jeśli wartość napięcia wejściowego przekracza napięcie z wewnętrznego przetwornika C/A oraz '0' w przeciwnym wypadku. Bity są przetwarzane w kolejności od najbardziej (MSB) do najmniej znaczącego (LSB).

Prędkość działania jest znacznie niższa niż w przypadku przetworników o porównaniu bezpośrednim, co przekłada się na niższą częstotliwość próbkowania. Im większa rozdzielczość takiego przetwornika, tym dłuższy czas wykonywania jednego cyklu przetwarzania danych [1].

2.2.3. Przetwornik sigma-delta

Przetwornik sigma-delta (zwany także *delta-sigma*) stosuje nadpróbkowanie (*eng. oversampling*) i kształtowanie szumu (*ang. noise shaping*). Stosowanie nadpróbkowania umożliwia przetwarzanie różnicy między kolejnymi próbkami zamiast wartości kolejnych próbek, dzięki czemu na wyjściu pojawia się jednobitowa informacja o sygnale przyrostowym.

Układ przetwornika sigma-delta zawiera w sobie generator, komparator, układ ekstrapolujący oraz układy logiczne. Jego budowa jest skomplikowana, ale cechuje go bardzo wysoka dokładność [1].

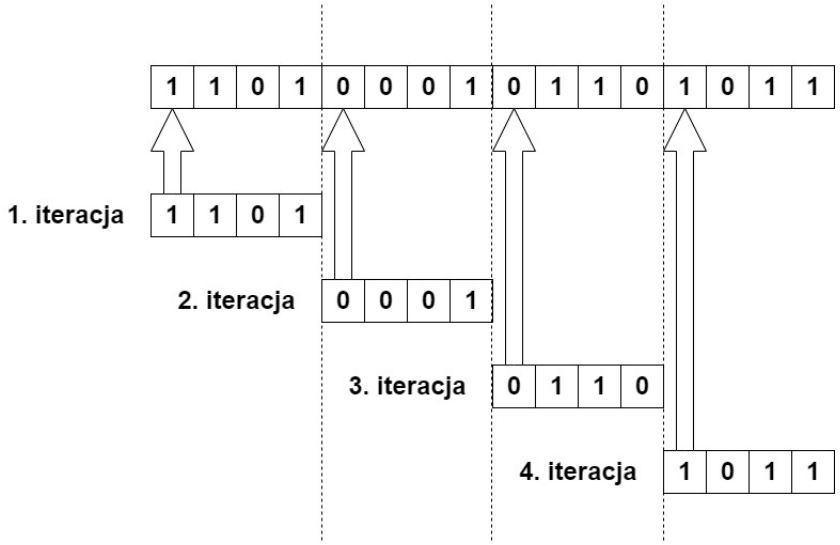
2.2.4. Przetwornik cykliczny: klasyczny i adaptacyjny

Cykliczne przetworniki A/C oprócz części analogowej zawierają również cyfrowy blok sterowania, zawierający algorytm obliczający kody estymat. Konwersja próbki wejściowej następuje iteracyjnie, dzięki czemu wynikowa rozdzielczość jest większa niż rozdzielczość wewnętrznego przetwornika. Wyjście wewnętrznego przetwornika A/C o niskiej rozdzielczości jest przekazywane na wejście cyfrowego bloku sterowania, w celu przetworzenia danych i obliczenia estymat. Co więcej, takie przetworniki zapewniają relatywnie niski poziom rozproszenia energii oraz mają mniejszy rozmiar dzięki zastosowaniu mniejszej liczby komparatorów w porównaniu z innymi typami przetworników [3][4].

Istnieje szereg rozwiązań, jeśli chodzi o strukturę i tworzenie kodów estymat w cyklicznym przetworniku A/C, dzięki czemu cały czas są tworzone ulepszone wersje istniejących metod. Wciąż prowadzone są prace badawczo-rozwojowe, których celem jest osiągnięcie jak najwyższej rozdzielczości, efektywnej liczbę bitów oraz prędkości działania przy jednoczesnym niskim poborze mocy.

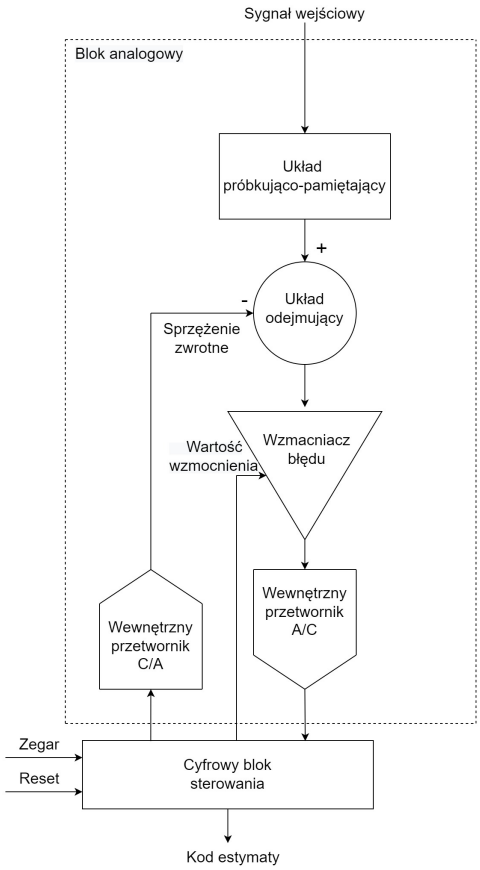
Pomimo mnogości istniejących rozwiązań można wyróżnić podejście klasyczne, opierające się na tworzeniu w części cyfrowej kodów estymaty metodą konkatenacji wyników cząstkowych przychodzących z wewnętrznego przetwornika A/C w kolejnych iteracjach [4].

Konwersja spróbkowanego sygnału wejściowego jest przeprowadzana przez pewną liczbę iteracji. Maksymalna ich liczba w przestawionym przykładzie jest 4 razy mniejsza niż rozdzielczość wewnętrznego przetwornika C/A, ponieważ wewnętrzny przetwornik A/C ma rozdzielczość na poziomie 4 bitów [4].



Rys. 2.4. Zasada tworzenia wynikowych kodów estymaty w cyklicznym przetworniku A/C.

Model klasyczny



Rys. 2.5. Schemat blokowy klasycznego cyklicznego przetwornika A/C.

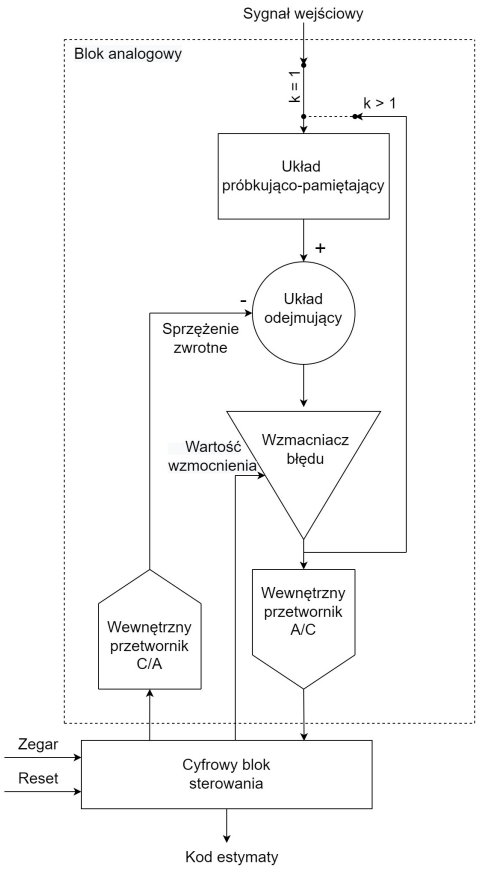
Schemat blokowy został przedstawiony na Rys. 2.5. Początkiem cyklu pracy klasycznego iteracyjnego przetwornika A/C jest spróbkowanie sygnału przez układ próbkująco-pamiętający. Okres próbkowania nie może być krótszy niż suma czasów trwania wszystkich cykli.

Spróbkowany sygnał jest przekazywany do układu odejmującego. Trafia tam również sprzężenie zwrotne, czyli wyjście wewnętrznego przetwornika C/A, będące estymatą kodu otrzymanego w poprzedniej iteracji. W pierwszej iteracji analogowa estymata powinna mieć wartość możliwie bliską zeru, aby nie zaburzyć działania układu. Wynikiem przejścia próbki przez układ odejmujący jest sygnał błędu, który trafia na wejście wzmacniacza [4].

Wartość wzmocnienia jest związana z numerem iteracji i przychodzi do wzmacniacza błędu w postaci sygnału z cyfrowego bloku sterowania. W kolejnych iteracjach sygnał błędu powinien mieć coraz niższą wartość, więc w celu wykorzystania możliwie największego zakresu wewnętrznego przetwornika A/C wzmocnienie powinno być coraz wyższe, ale na tyle niewielkie, aby nie doszło do przesterowania elementu.

Po wyjściu ze wzmacniacza błędu próbka trafia na wewnętrzny przetwornik A/C o niskiej rozdzielczości. Wynik jego pracy jest przekazywany do cyfrowego bloku sterowania, gdzie jest zapisywany jako fragment wynikowego kodu estymaty. W kolejnych iteracjach zmienia się wartość wzmocnienia we wzmacniaczu błędu, sprzężenie zwrotne w jest niezerowe, a kolejne wartości otrzymywane na wyjściu wewnętrznego przetwornika A/C są zapisywane na coraz młodszych bitach wynikowego kodu estymaty. Po ukończeniu cyklu na wyjściu bloku sterowania pojawia się cyfrowy sygnał o wysokiej rozdzielczości [4].

Model adaptacyjny



Rys. 2.6. Schemat blokowy adaptacyjnego cyklicznego przetwornika A/C.

Głównymi powodami, dla których zdecydowano się na modyfikację modelu klasycznego są konieczność użycia wewnętrznego przetwornika C/A o wysokiej rozdzielczości oraz duże wartości wzmocnień we wzmacniaczu sygnału błędu.

Model adaptacyjny rozwiązuje te problemy. Rozdzielczość wewnętrznego przetwornika C/A jest równa rozdzielczości wewnętrznego przetwornika A/C. Układ próbkująco–pamiętający zmienia wartość pamiętanej próbki w każdej kolejnej iteracji: w pierwszym cyklu działa tak, jak w modelu klasycznym, a w każdym kolejnym "przechwytuje"wartość sygnału wyjściowego wzmacniacza błędu z poprzedniej iteracji. Sygnał pełniący rolę sprzężenia zwrotnego jest analogową estymatą wartości otrzymanej w przetworniku A/C w poprzedniej iteracji [4].

3. Projekt

3.1. Cele i założenia pracy

Moduł sterowania i obliczania estymat stanowi część układu, jakim jest cykliczny przetwornik A/C. W celu zbadania poprawności jego działania niezbędne jest stworzenie modelu części analogowej.

Opracowanie modeli rzeczywistych elementów elektronicznych w języku VHDL nie należy do zadań, które można zrealizować szybko i bezproblemowo. Elektronikę cyfrową od analogowej odróżnia przede wszystkim typ arytmetyki. W przypadku elektroniki analogowej nie mamy do czynienia wyłącznie z liczbami stałoprzecinkowymi, jak bywa to w elektronice cyfrowej. Język VHDL daje możliwość wykorzystania typu *real*, jednak kod napisany z wykorzystaniem sygnałów lub zmiennych tego typu jest niesyntezowalny.

W celu wykorzystania arytmetyki dla liczb zmiennoprzecinkowych, model cyklicznego przetwornika A/C został podzielony na syntezowalny cyfrowy blok sterowania, będący głównym elementem zainteresowania, oraz niesyntezowalny model części analogowej, stanowiący jednocześnie testbench.

Projekt został wykonany przy użyciu programu Quartus Prime. Urządzenie, na którym odbyły się symulacje to układ FPGA z rodziny MAX 10, model 10M02DCU324A6G.

Rys. 3.1. Wygląd układu FPGA z rodziny MAX 10 na przykładzie 10M08DAF256C8GES.

Zasady przyjęte podczas implementacji algorytmów

W celu zachowania jak najlepszej czytelności kodu w języku VHDL zastosowano zasadę dodawania przedrostków:

- i_ sygnał wejściowy do układu / bloku;
- o_ sygnał wyjściowy z układu / bloku;
- r_ rejestr;
- G_ generic;
- C_ stała;
- T_ typ zdefiniowany przez użytkownika;
- P_ proces.

Działanie układu zostało opisane z użyciem procesów: sekwencyjnych w przypadku cyfrowego bloku sterowania, a także zarówno sekwencyjnych, jak i kombinacyjnych w przypadku modelu bloku analogowego. Brak procesów kombinacyjnych w części syntezowalnej jest spowodowany chęcią uniknięcia zatrzasków oraz zminimalizowania ryzyka wystąpienia metastabilności.

3.2. Cyfrowy blok sterowania

Część cyfrowa cyklicznego przetwornika A/C decyduje w znacznej części o jego poprawności działania oraz parametrach. W ogólności blok ten posiada trzy wejścia (zegar, reset oraz kod estymaty przychodzący z wewnętrznego przetwornika), a także trzy wyjścia (wartość wzmocnienia, wartość, na podstawie której jest obliczane sprzężenie zwrotne oraz wynikowy kod estymaty).

Modele klasyczny i adaptacyjny wykorzystują ten sam algorytm tworzenia wynikowego kodu estymaty na podstawie kodów estymaty z wewnętrznego przetwornika A/C. W przypadku wartości wzmocnienia oraz sprzężenia zwrotnego występują istotne różnice.

W celu minimalizacji poboru mocy, blok cyfrowy został opracowany tak, aby zużywał możliwie jak najmniej zasobów. Kierowano się takimi parametrami jak:

- liczba wykorzystywanych elementów logicznych;
- liczba wykorzystywanych rejestrów;
- liczba wykorzystywanych bitów pamięci.

3.2.1. Kod estymaty z wewnętrzego przetwornika A/C

Działanie cyklicznego przetwornika A/C w dużej mierze zależy od zastosowanego wewnętrznego przetwornika A/C o niskiej rozdzielczości. Równie istotną rolę gra sposób przechwycenia oraz wykorzystania danych wchodzących do cyfrowego bloku sterowania.

Jednym z podstawowych problemów napotkanych podczas implementacji części cyfrowej był sposób odczytu danych przychodzących z części analogowej. Początkowo brano pod uwagę dwie opcje:

- reagowanie na zmianę danych przychodzących i założenie, że zmiana danych oznacza, że została wykonana kolejna iteracja;
- wykonywanie kolejnych iteracji co pewiem określony czas, bez względu na to, czy kod estymaty wewnętrzengo przetwornika A/C uległ zmianie.

Jak łatwo zauważyć, pierwsza z wymienionych metod jest szybsza - reakcja części cyfrowej na zmianę sygnału wejściowego jest niemal natychmiastowa, następuje przy kolejnym zboczu narastającym zegara. Nie uwzględnia ona jednak sytuacji, gdy częściowe kody estymaty w kolejnych iteracjach są identyczne, jak ma to miejsce np. przy wartości minimalnej rozpatrywanego zakresu.

W modelach zdecydowano się na użycie drugiej metody. Jest zauważalnie wolniejsza, ale odpowiednio dobrana częstotliwość wykonywania kolejnych iteracji gwarantuje stabilne działanie układu.

Na wybranie odpowiedniej częstotliwości wykonywania kolejnych cykli części cyfrowej ma wpływ kilka czynników. Można wyróżnić:

- czas propagacji sygnału przez część analogową układu;
- czas działania części cyfrowej układu.

Część cyfrowana jest taktowana zegarem o częstotliwości 50 MHz, a jeden cykl pracy to maksymalnie kilka taktów zegara. Można założyć, że wpływ tego czynnika jest znikomy w porównaniu z czasem propagacji sygnału przez część analogową.

Czas propagacji to jeden z parametrów, który posiada każdy rzeczywisty element elektroniczny. Jest to czas, który upływa od chwili zmiany wejścia układu lub elementu logicznego do chwili ustalenia stanu wyjść, będącej reakcją na zmianę wejścia [1]. W modelach założono, że czas propagacji każdego z elementów jest równy i wynosi $t_p = 100$ ns.

Część analogowa układu składa się z kilku elementów, więc czas propagacji całego bloku jest kilkukrotnie wyższy. Jaka jest zatem odpowiednia częstotliwość wykonywania kolejnych cykli przez część cyfrową układu? Założono, że stabilność działania układu gra większą rolę niż osiągnięcie maksymalnej przepustowości. Dobrana częstotliwość uwzględnia zapewnienie pewnego bufora czasowego, aby upewnić się, że pobranie danych nie nastąpi za wcześnie. Przyjęto wartość $f_s = 1$ µs.

Aby umożliwić szybką adaptację modelu do rzeczywistych elementów o innych parametrach, częstotliwość pobierania nowych danych została zdefiniowana w kodzie jako stała, której wartość można łatwo zmodyfikować, a jej zmiana nie spowoduje zaburzenia działania układu.

3.2.2. Wynikowy kod estymaty

Kody estymaty są tworzone metodą konkatenacji opisaną w punkcie 2.2.4. Wyjście bloku cyfrowego jest aktualizowane dopiero po wykonaniu wszystkich iteracji, co oznacza, że cykl pracy zawierający cztery iteracje trwa $t = 4$ µs. Otrzymany model pracuje z częstotliwością:

$$f = 250\ kHz \tag{3.1}$$

Taka prędkość działania jest w pełni satysfakcjonująca dla sygnałów okresowych o częstotliwości do kilkudziesięciu kHz.

3.2.3. Wartość wzmocnienia

Wartość wzmocnienia jest przekazywana do wzmacniacza sygnału błędu, a jej wielkość zależy od numeru aktualnej iteracji. Należy zauważyć, że wartości wzmocnień w modelu klasycznym różnią się od wartości w modelu adaptacyjnym.

Model klasyczny

W idealnym przypadku, przy założeniu, że wewnętrzny przetwornik A/C jest 4-bitowy, wzmocnienie miałoby wartość:

$$C_k = 2^{3k-3} \tag{3.2}$$

gdzie k jest numerem aktualnej iteracji [4]. Z racji, że sytuacje idealne nie znajdują odwzorowania w rzeczywistości, zastosowanie takich wartości wzmocnienia prowadzi do przesterowania wewnętrznego przetwornika w 3. i 4. iteracji. Nowe wartości wzmocnień, czyli takie, dla których nie dochodzi do przesterowania, dobrano metodą inżynierską. Ich wartość zaprezentowano w tabeli 3.1.

Tab. 3.1. Zależność wartości wzmocnienia od numeru iteracji w modelu klasycznym.

Numer iteracji	Wartość wzmocnienia
1	1
2	8
3	16
4	16

Model adaptacyjny

Wartości wzmocnienia w modelu adaptacyjnym dobrano metodą inżynierską. Musiały one spełniać te same wymagania, jak w modelu klasycznym. Należy wykorzystać jak największą część zakresu pracy wewnętrznego przetwornika A/C, ale nie może dojść do jego przesterowania. Uzyskane wartości zaprezentowano w tabeli 3.2.

Tab. 3.2. Zależność wartości wzmocnienia od numeru iteracji w modelu adaptacyjnym.

Numer iteracji	Wartość wzmocnienia
1	1
2	2
3	2
4	2

3.2.4. Sprzężenie zwrotne

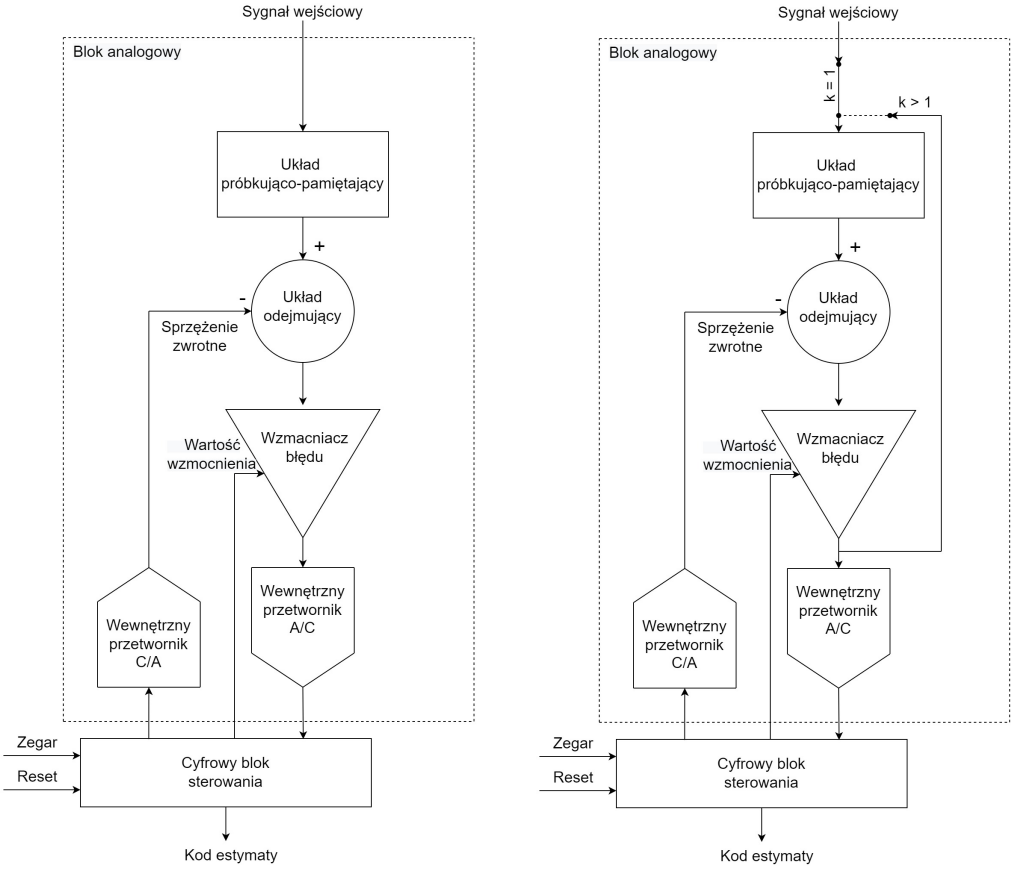
Zarówno algorytm klasyczny, jak i adaptacyjny wykorzystują sygnał sprzężenia zwrotnego w układzie odejmującym. W modelu klasycznym jako sprzężenie zwrotne wykorzystano estymatę kodu uzyskanego poprzedniej iteracji, a w modelu adaptacyjnym użyto estymaty sygnału wyjściowego z wewnętrznego przetwornika A/C o niskiej rozdzielczości.

Jak łatwo zauważyć, w modelu klasycznym kod estymaty sprzężenia zwrotnego jest 16-bitowy, a w modelu klasycznym 4-bitowy.

3.3. Model części analogowej badanego układu

Zarówno algorytm klasyczny, jak i adaptacyjny wykorzystują obecność układu odejmującego, wzmacniacza sygnału błędu, wewnętrznego przetwornika A/C o niskiej rozdzielczości, układu próbująco-pamiętającego oraz wewnętrznego przetwornika C/A.

Algorytmy działania części analogowych modeli zarówno klasycznego, jak i adaptacyjnego są tożsame ze sposobami opisanymi w punkcie 2.2.4. Oba rozwiązania mają liczne zalety: algorytmy działania nie są skomplikowane, a także mogą być łatwo modyfikowane i adaptowane pod wiele rzeczywistych elementów elektronicznych.



Rys. 3.2. Prównanie struktury klasycznego (a) i adaptacyjnego (b) cyklicznego przetwornika A/C.

Układ próbkująco-pamiętający

Istotą działania układu próbkująco-pamiętającego jest podtrzymywanie tej samej wartości próbki przez określony czas. Maksymalna długość takiego odcinka czasowego jest określona przez *czas akwizycji*. W modelu klasycznym ma on wartość:

$$t_{akwizycji} = 4\,\mu s \tag{3.3}$$

Wartość ta jest nieprzypadkowa – mieści się w zakresie dozwolonych czasów akwizycji dla układu próbkująco-pamiętającego firmy *Analog Devices*, model *LF198*. W modelu adaptacyjnym czas akwizycji jest krótszy i wynosi:

$$t_{akwizycji} = 1\,\mu s \tag{3.4}$$

Ta wartość również mieści się w zakresie pracy modelu *LF198*. Oznacza to, że uzyskanie w rzeczywistym układzie parametrów przyjętych w obu modelach jest możliwe.

Układ odejmujący

Sygnał błędu jest tworzony w każdej iteracji jako różnica wartości próbki w układzie próbkująco–pamiętającym i sprzężenia zwrotnego. Dzięki zastosowaniu w modelu arytmetyki liczb zmiennoprzecinkowych można w prosty sposób zasymulować działanie układu odejmującego jako pro-

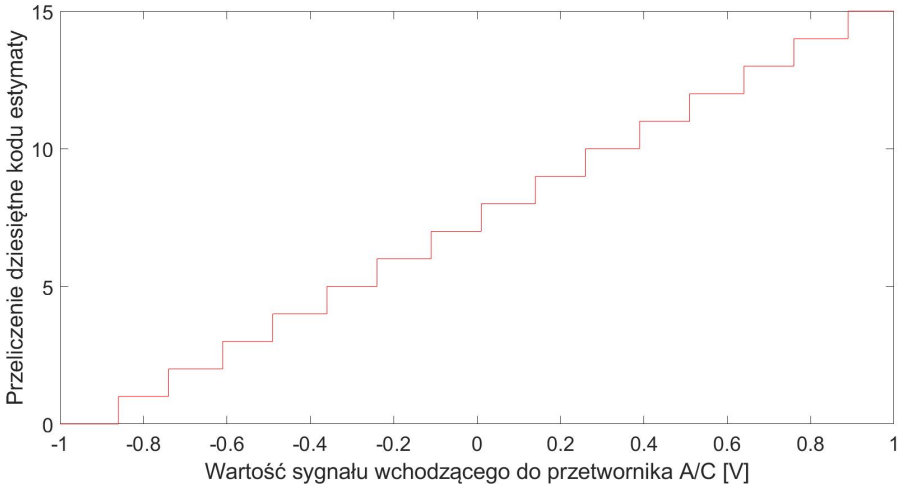
ces kombinacyjny, na którego liście wrażliwości znajdują się sygnały imitujące wyjście układu próbkującom-pamiętającego oraz wyjście wewnętrznego przetwornika C/A.

Wzmacniacz sygnału błędu

Sygnał błędu trafia na wejście wzmacniacza o zmiennym wzmocnieniu, którego wartość zależy od numeru iteracji oraz używanego modelu. Wartość wzmocnienia jest obliczana w każdej iteracji w cyfrowym bloku sterowania. W wyniku działania wzmacniacza powstaje wzmocniony sygnał błędu, który trafia na wejście wewnętrznego przetwornika A/C o niskiej rozdzielczości.

Wewnętrzny przetwornik A/C o niskiej rozdzielczości

Zarówno w przypadku modelu klasycznego, jak i adaptacyjnego użyto 4-bitowego wewnętrznego przetwornika A/C. Symulowany element ma właściwości zbliżone do przetwornika typu *flash*. W modelu zastosowano kodowanie *USB*. Zależność kodu estymaty od napięcia wejściowego została pokazana na Rysunku 3.3.



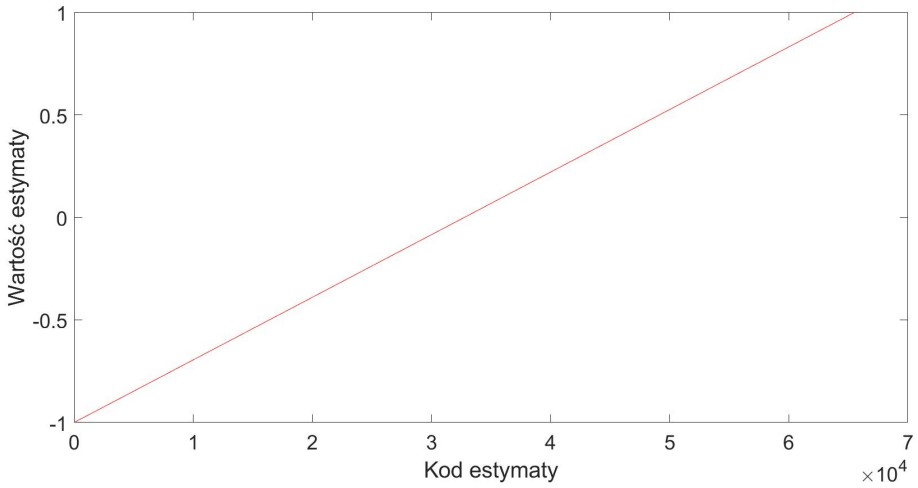
Rys. 3.3. Kod estymaty wewnętrznego przetwornika A/C w zależności od napięcia wejściowego.

Wewnętrzny przetwornik C/A

Największą różnicą w budowie bloku analogowego między modelem klasycznym a adaptacyjnym jest zastosowany wewnętrzny przetwornik C/A. W modelu klasycznym użyto przetwornika o rozdzielczości szesnastu bitów, a w modelu adaptacyjnym o rozdzielczości czterech bitów.

Sposób dekodowania jest tożsamy do sposobu kodowania zastosowanego w wewnętrznym przetworniku A/C, dzięki czemu nie występują błędy związane z różnymi metodami kodowania.

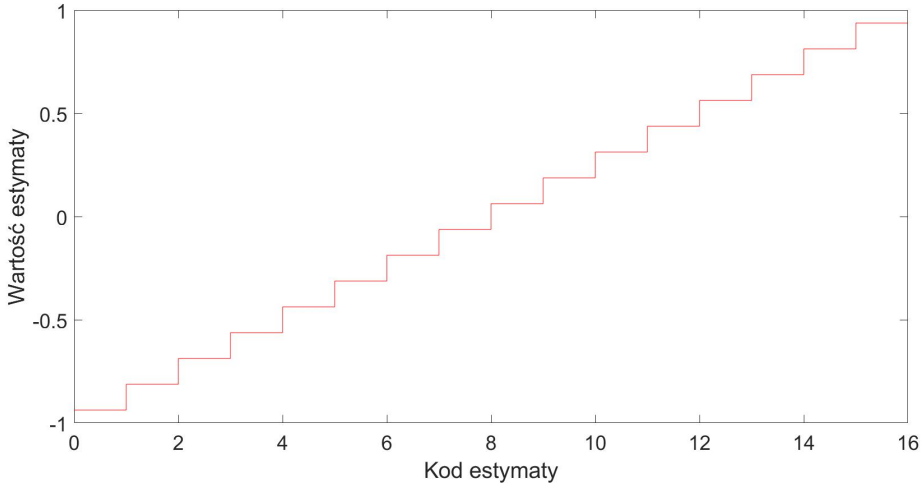
Model klasyczny



Rys. 3.4. Napięcie wyjściowe wewnętrznego 16-bitowego przetwornika C/A w zależności od kodu estymaty.

W 16-bitowym przetworniku C/A występuje aż 65536 poziomów napięcia, dzięki czemu przeskoki na kolejny poziom są praktycznie niezauważalne. Otrzymane estymaty kolejnych kodów są do siebie bardzo zbliżone.

Model adaptacyjny



Rys. 3.5. Napięcie wyjściowe wewnętrznego 4-bitowego przetwornika C/A w zależności od kodu estymaty.

W 4-bitowym przetworniku C/A można łatwo wyróżnić 16 poziomów napięcia wyjściowego odpowiadających 16 kodom estymaty. Zmiany są skokowe i łatwo je zauważyć z powodu niskiej rozdzielczości przetwornika.

4. Weryfikacja projektu i opracowanie wyników

Sprawdzenie poprawności działania modeli układów jest kluczowym elementem w ocenie ich użyteczności. Weryfikacja modeli została podzielona na 2 etapy:

- pomiary zużycia zasobów przez część cyfrową;
- badanie odpowiedzi przetwornika A/C na różne dane wejściowe.

Pomiary zużycia zasobów odbyły się przy użyciu oprogramowania Quartus Prime z użyciem układu FPGA z rodziny Max 10, a przedstawione wyniki są efektem działania procesu analizy i syntezy.

Kluczowy etap weryfikacji modeli, czyli badanie odpowiedzi przetwornika A/C, wykonano za pomocą symulacji behawioralnej w programie ModelSim. Nie jest to idealna metoda – nie uwzględnia m.in. opóźnień w rzeczywistych układach, ale jej zastosowanie pozwala wychwycić wiele błędów w procesie tworzenia modeli.

Jako zegar przyjęto wygenerowany w testbenchu sygnał prostokątny o częstotliwości 50 MHz i wypełnieniu na poziomie 50%. Takie zegary są szeroko stosowane w rzeczywistych układach – okres jest wystarczająco długi, aby założyć, że sygnały zdążą się przepropagować do kolejnego zbocza zegara. Do opracowania wyników wykorzystano środowisko Matlab oraz zapewnione przez nie narzędzia do przetwarzania sygnałów.

4.1. Przykładowe implementacje

Przed przystąpieniem do weryfikacji zaimplementowano modele klasyczny i adaptacyjny zgodnie z założeniami opisanymi w rozdziale 3. W sekcji 3.2.1 opisano tworzenie kodu estymaty jako wykonywanie kolejnych iteracji co pewien określony czas, bez względu na to, czy kod estymaty wewnętrznego przetwornika A/C uległ zmianie. Podczas implementacji zastosowano tę motodę i przyjęto częstotliwość próbkowania 1 MHz. Tę wartość zapisano do stałej, na której opiera się licznik inkrementowany przy każdym takcie zegara systemowego, dzięki czemu może być w prosty sposób modyfikowana, jednakże nie może przekroczyć częstotliwości zegara 50 MHz. Zbyt szybkie próbkowanie może również spowodować nieprawidłowe działanie bloku sterowania – w każdej iteracji wykonanie zapisu zajmuje kilka taktów zegara, więc okres, co jaki następuje próbkowanie nie powinien być krótszy niż czas trwania około 10 taktów, czyli 200 ns. Każdorazowe odliczenie przez licznik do jego wartości granicznej skutkuje ustawieniem na jeden takt zegara flagi o nadejściu nowych danych i w efekcie zapis kolejnego cząstkowego kodu estymaty.

Określanie numeru aktualnie trwającej iteracji również zostało zrealizowane za pomocą licznika, którego wartość jest inkrementowana po każdym zapisie cząstkowego kodu estymaty. Na podstawie numeru iteracji określa się wartość wzmocnienia wysyłaną do wzmacniacza sygnału błędu oraz część wynikowego kodu estymaty, do którego następuje zapis danych przychodzących z wewnętrznego przetwornika A/C.

Moduł sterowania decyduje także o sprzężeniu zwrotnym, które jest wysyłane do wewnętrznego przetwornika C/A o rozdzielczości 16 bitów w modeku klasycznym oraz 4 bitów w modelu adaptacyjnym. Na początku każdego cyklu pracy sprzężenie zwrotne ma wartość możliwie bliską zeru, a w kolejnych iteracjach jego wartość zmienia się po zapisie każdego z cząstkowych kodów estymaty.

Kod estymaty jest wystawiany na wyjście bloku sterowania po zakończeniu każdego cyklu pracy. W zaimplementowanych modelach dla 4 iteracji oznacza to aktualizację co 4 µs. Nie jest

to wartość graniczna, da się osiągnąć większą szybkość, zmieniając częstotliwość próbkowania lub częstotliwość pracy zegara. Należy jednak pamiętać, że poza ograniczeniem w postaci czasu działania bloku cyfrowego istnieje znacznie większe ograniczenie: czas propagacji sygnału w części analogowej układu. Każdy z elementów elektronicznych powoduje zwiększenie czasu propagacji.

W opisanym w rozdziale 3 projekcie przyjęto, że czas propagacji przez każdy z elementów wynosi 100 ns. W rzeczywistości są to czasy o rząd wielkości niższe, jednakże takie założenie pozwala na osiągnięcie lepszej stabilności modułów sterowania. W celu osiągnięcia najwyższej możliwej szybkości działania należy dobrać elementy elektroniczne, z których jest zbudowana część analogowa tak, aby wartość czasu propagacji przez każdy z nich była możliwie niska. Co więcej, aby zapewnić stabilność działania należy zwrócić szczególną uwagę na parametry *setup time* oraz *hold time*. Odpowiednio dobrane zależności czasowe w układzie mogą spowodować, że ograniczenie związane z czasem propagacji sygnału w częsci analogowej będzie mniejsze niż ograniczenie części cyfrowej. W takiej sytuacji o maksymalnej częstotliwości próbkowania zadecydują częstotliwość zegara oraz liczba taktów, jaka przypada na jeden cykl zapisu. Aby przyspieszyć działanie modułu sterowania można zastosować zegar o wyższej częstotliwości, na przykład wygenerowany przy pomocy układu PLL.

W prezentowanych przykładach nie było konieczności stosowania powyższych metod – założono, że częstotliwość próbkowania na poziomie 1 MHz jest satysfakcjonująca.

4.2. Pomiar wykorzystania zasobów układów cyfrowych

Zużycie zasobów przez część cyfrową opracowanych modeli pozwala ocenić, jakiej wielkości rzeczywisty układ cyfrowy jest niezbędny do zapewnienia pełnej funkcjonalności przy implementacji sprzętowej. Im mniejsze zużycie zasobów, tym bardziej prawdopodobne, że moc wydzielana w układzie cyfrowym będzie niewielka. Do oceny wykorzystania zasobów wzięto pod uwagę następujące parametry:

- liczba wykorzystywanych elementów logicznych;
- liczba wykorzystywanych rejestrów;
- liczba wykorzystywanych bitów pamięci.

Model klasyczny

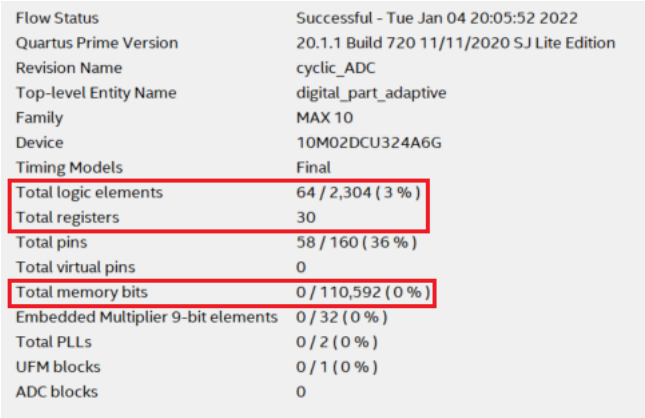
Flow Status	Successful - Tue Jan 04 19:58:31 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	cyclic_ADC
Top-level Entity Name	digital_part_classic
Family	MAX 10
Device	10M02DCU324A6G
Timing Models	Final
Total logic elements	77 / 2,304 (3 %)
Total registers	31
Total pins	70 / 160 (44 %)
Total virtual pins	0
Total memory bits	0 / 110,592 (0 %)
Embedded Multiplier 9-bit elements	0 / 32 (0 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0

Rys. 4.1. Zużycie zasobów układu cyfrowego w modelu klasycznym.

W modelu klasycznym blok cyfrowy wykorzystuje 77 elementów logicznych i 31 rejestrów. Nie został użyty żaden bit pamięci. Zakładając, że część cyfrowa cyklicznego przetwornika A/C byłaby

częścią jakiegoś większego układu cyfrowego, np. układu FPGA, dołączenie takiego komponentu nie powinno spowodować konieczności zastosowania układu o większych zasobach. Różnice między wielkością poszczególnych układów cyfrowych stosowanych w dużych projektach są znacznie większe niż kilkadziesiąt elementów logicznych i rejestrów.

Model adaptacyjny



Flow Status	Successful - Tue Jan 04 20:05:52 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	cyclic_ADC
Top-level Entity Name	digital_part_adaptive
Family	MAX 10
Device	10M02DCU324A6G
Timing Models	Final
Total logic elements	64 / 2,304 (3 %)
Total registers	30
Total pins	58 / 160 (36 %)
Total virtual pins	0
Total memory bits	0 / 110,592 (0 %)
Embedded Multiplier 9-bit elements	0 / 32 (0 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0

Rys. 4.2. Zużycie zasobów układu cyfrowego w modelu adaptacyjnym.

Zasoby wykorzystywane przez część cyfrową modelu adaptacyjnego są na zbliżonych poziomie do modelu klasycznego. W tym przypadku również można założyć, że zużycie zasobów układu cyfrowego jest marginalne.

4.3. Weryfikacja poprawności działania zbudowanych modeli

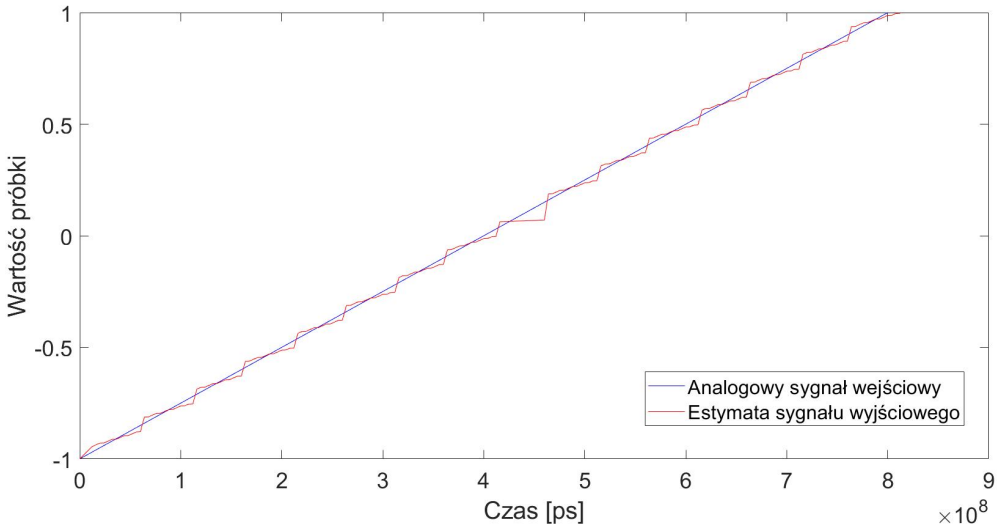
Wybrano kilka typów sygnałów, na których zostały dokonane pomiary:

- sygnał quasi-liniowy, przyjmujący możliwie dużo wartości między dolnym a górnym zakresem działania przetwornika;
- sygnał sinusoidalny o częstotliwości 1 kHz;
- sygnał sinusoidalny o częstotliwości 10 kHz;
- sygnał trójkątny o częstotliwości 1 kHz;
- sygnał trójkątny o częstotliwości 10 kHz.

4.3.1. Charakterystyka przetwarzania w całym zakresie

W pierwszym podejściu dokonano analizy zachowania modeli dla możliwie dużej liczby punktów znajdujących się w zakresie pracy cyklicznego przetwornika A/C. Utworzono 201 równo od siebie oddalonych punktów pomiarowych rozłożonych na cały zakres pracy i zbadano odpowiedź modelu cyklicznego przetwornika A/C.

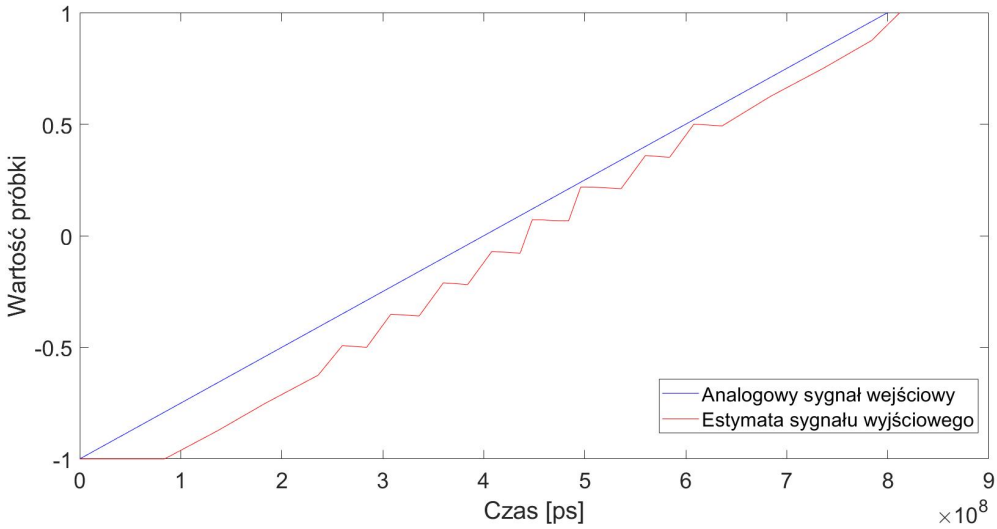
Model klasyczny



Rys. 4.3. Estymata klasycznego przetwornika A/C w całym zakresie pracy.

W modelu klasycznym udało się osiągnąć estymaty bardzo zbliżone do wartości analogowych napięcia wejściowego. Jest to zasługa m.in. zastosowania wewnętrznego przetwornika C/A o wysokiej rozdzielczości.

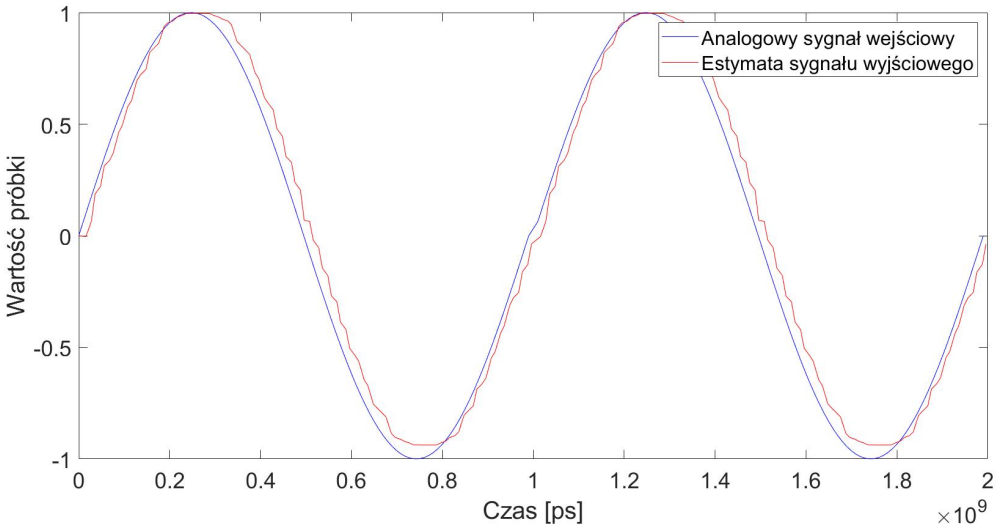
Model adaptacyjny



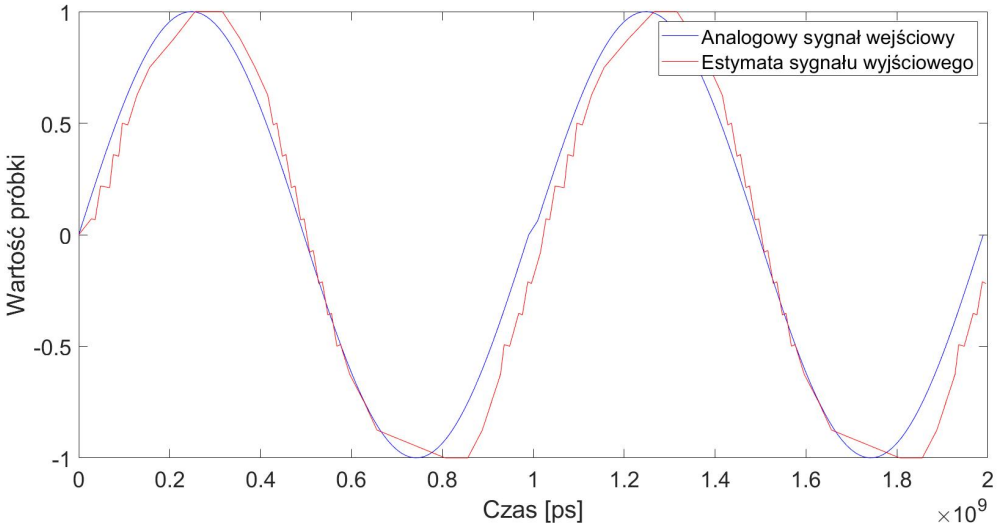
Rys. 4.4. Estymata adaptacyjnego przetwornika A/C w całym zakresie pracy.

Wynik wstępnej weryfikacji modelu adaptacyjnego początkowo nie napajał optymizmem – otrzymane estymaty miały wartości mniejsze od wejściowych nawet o 5%. Na taki efekt mogło mieć wpływ zastosowanie wewnętrznego przetwornika C/A o niskiej rozdzielczości, a także zmiana sposobu działania układu próbkująco-pamiętającego.

4.3.2. Sygnał sinusoidalny o częstotliwości 1 kHz



Rys. 4.5. Estymata klasycznego przetwornika A/C dla sygnału sinusoidalnego o częstotliwości 1 kHz.

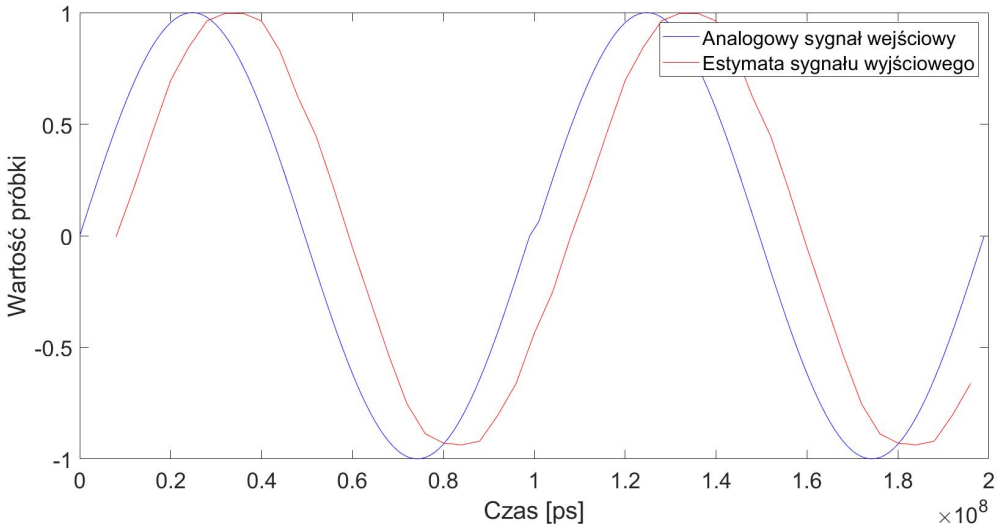


Rys. 4.6. Estymata adaptacyjnego przetwornika A/C dla sygnału sinusoidalnego o częstotliwości 1 kHz.

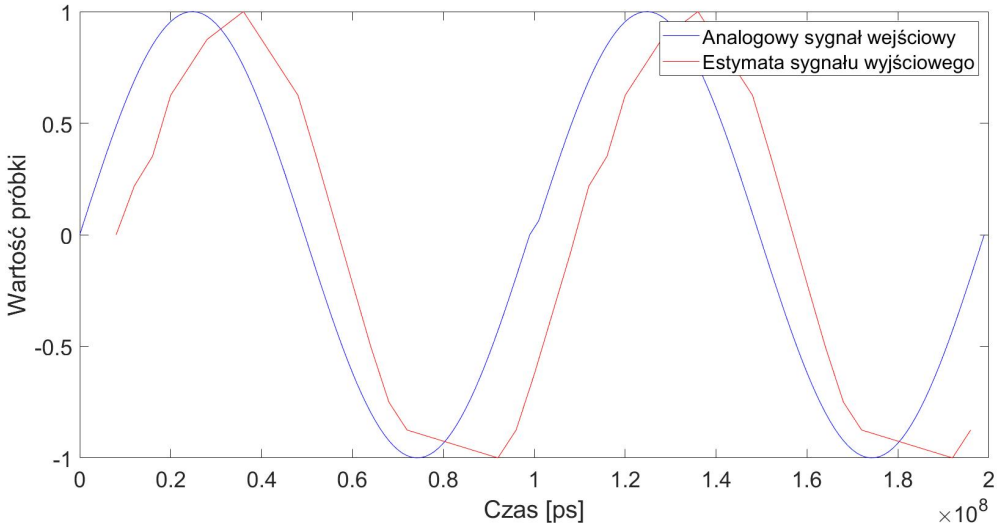
Na jeden okres trwania sygnału wejściowego przypada 250 próbek, z których została obliczona estymata. Estymaty wyjściowe w obu przypadkach przypominają kształtem sinusoidy. W modelu klasycznym sygnał jest bardziej gładki, osiąga wartość maksymalną analogowego sygnału wejściowego, jednak nie osiąga wartości minimalnej. W modelu klasycznym estymata nie jest tak gładka, ale osiąga obie krańcowe wartości.

W obu przypadkach estymata jest przesunięta w czasie względem sygnału wejściowego – jest to naturalne zjawisko, wynika to z czasu propagacji sygnału przez modele cyklicznych przetworników A/C.

4.3.3. Sygnał sinusoidalny o częstotliwości 10 kHz



Rys. 4.7. Estymata klasycznego przetwornika A/C dla sygnału sinusoidalnego o częstotliwości 10 kHz.

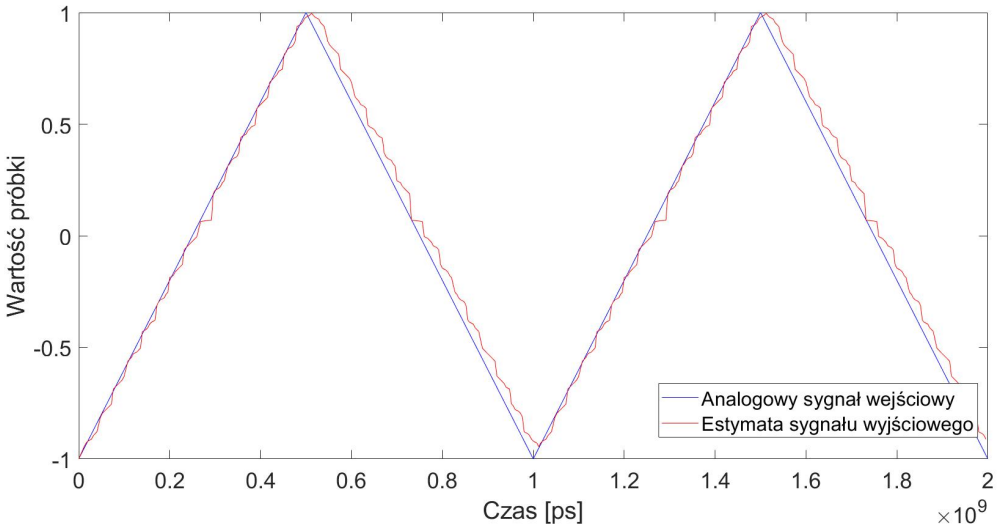


Rys. 4.8. Estymata adaptacyjnego przetwornika A/C dla sygnału sinusoidalnego o częstotliwości 10 kHz.

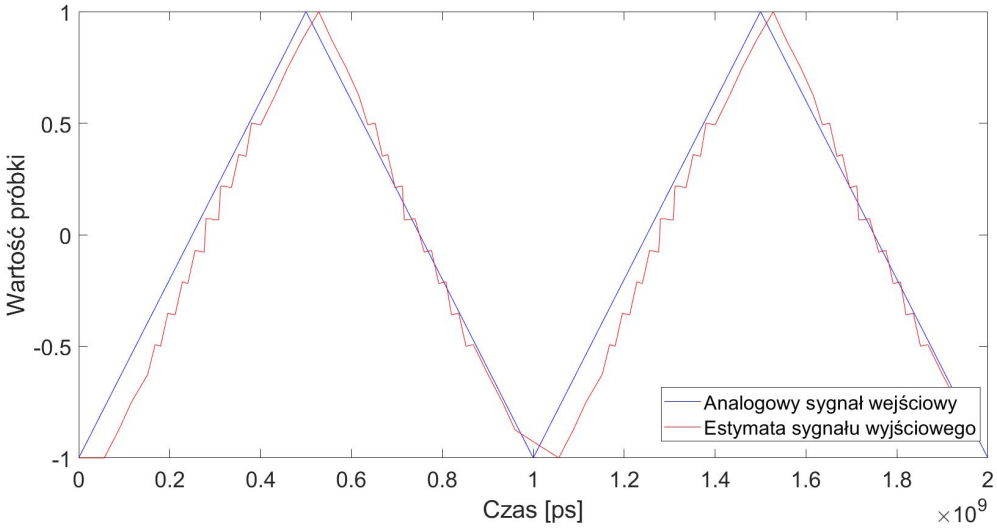
W przypadku sygnału wejściowego o częstotliwości 10kHz na jeden okres trwania sygnału przypada 25 próbek, a więc 10 razy mniej niż w poprzednim przypadku. W praktyce oznacza to bardziej gładkie przebiegi estymat. Podobnie jak poprzednio, w modelu klasycznym estymata nie osiąga wartości minimalnej, a w modelu adaptacyjnym osiągane są obie wartości graniczne.

W tym przypadku również mamy do czynienia z przesunięciem czasowym estymaty względem sygnału wejściowego. Chociaż wydaje się, że jest ono znacznie większe niż w przypadku sygnału o częstotliwości 1 kHz, to tak naprawdę przesunięcie ma identyczną wartość. Różnica wynika z przyjętej skali: pomiar sygnału o częstotliwości 1 kHz trwał 2 ms, a pomiar sygnału o częstotliwości 10 kHz to czas rzędu 200 µs.

4.3.4. Sygnał trójkątny o częstotliwości 1 kHz



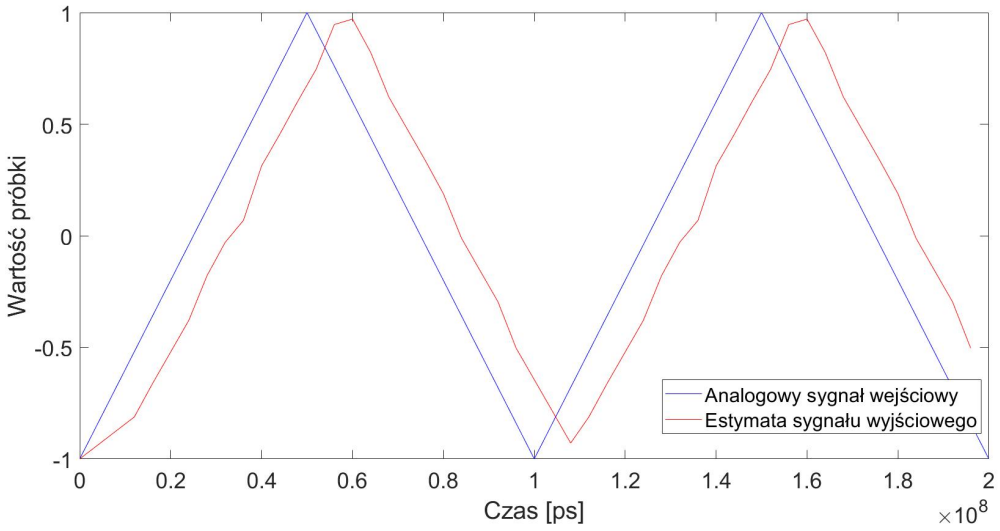
Rys. 4.9. Estymata klasycznego przetwornika A/C dla sygnału trójkątnego o częstotliwości 1 kHz.



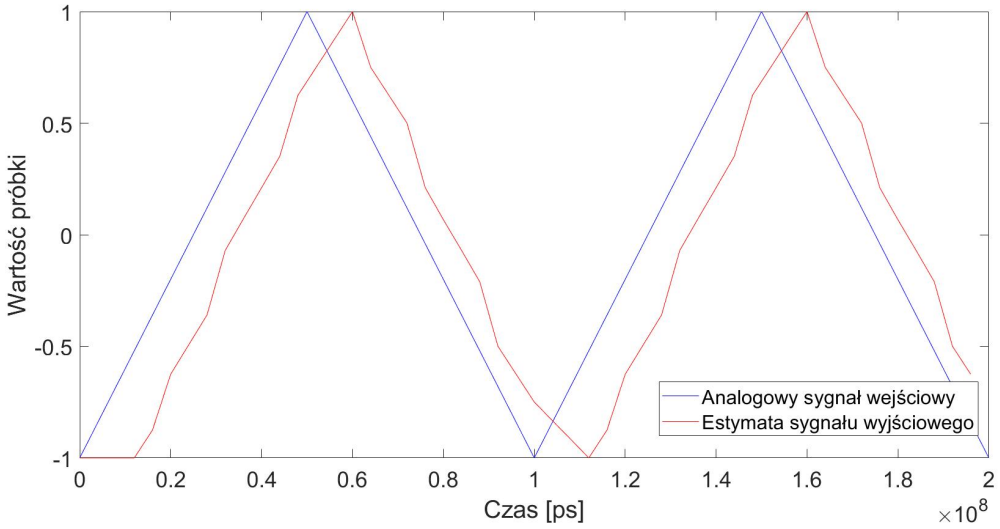
Rys. 4.10. Estymata adaptacyjnego przetwornika A/C dla sygnału trójkątnego o częstotliwości 1 kHz.

Kolejnym typem sygnału wejściowego, dla którego przeprowadzono testy jest sygnał trójkątny. Podobnie jak przy sygnale sinusoidalnym, na każdy okres sygnału przypada 250 próbek. Zarówno w modelu klasycznym, jak i adaptacyjnym estymata przypomina zniekształcony sygnał trójkątny. W modelu adaptacyjnym estymata jest fragmentami gładka, w modelu klasycznym zniekształcenia są zauważalne w całym zakresie.

4.3.5. Sygnał trójkątny o częstotliwości 10 kHz



Rys. 4.11. Estymata klasycznego przetwornika A/C dla sygnału trójkątnego o częstotliwości 10 kHz.



Rys. 4.12. Estymata adaptacyjnego przetwornika A/C dla sygnału trójkątnego o częstotliwości 10 kHz.

W przypadku sygnału trójkątnego o częstotliwości 10 kHz można wyciągnąć wnioski analogiczne do poprzednich przypadków. Większa częstotliwość oznacza mniejszą liczbę próbek na okres, więc wykres estymaty w obu modelach jest bardziej gładki. Przesunięcie czasowe między sygnałem wejściowym a estymatą również wydaje się większe niż dla sygnału o częstotliwości 1 kHz, ale w tym przypadku ponownie mamy do czynienia ze zmianą skali, przesunięcia czasowe nie wzrosły.

4.4. Badanie zależności efektywnej liczby bitów od liczby iteracji

Efektywna liczba bitów (ENOB, *ang. effective number of bits*) określa liczbę bitów sygnału cyfrowego powyżej poziomu szumów. W idealnym przypadku jej wartość byłaby równa rozdzielczości przetwornika A/C, ale w rzeczywistości jest mniejsza. Wartość tego parametru jest jednym z kluczowych elementów w ocenie jakości wykonania elementu.

Zachodzi równość:

$$ENOB = \frac{SINAD - 1{,}76}{6{,}02} \tag{4.1}$$

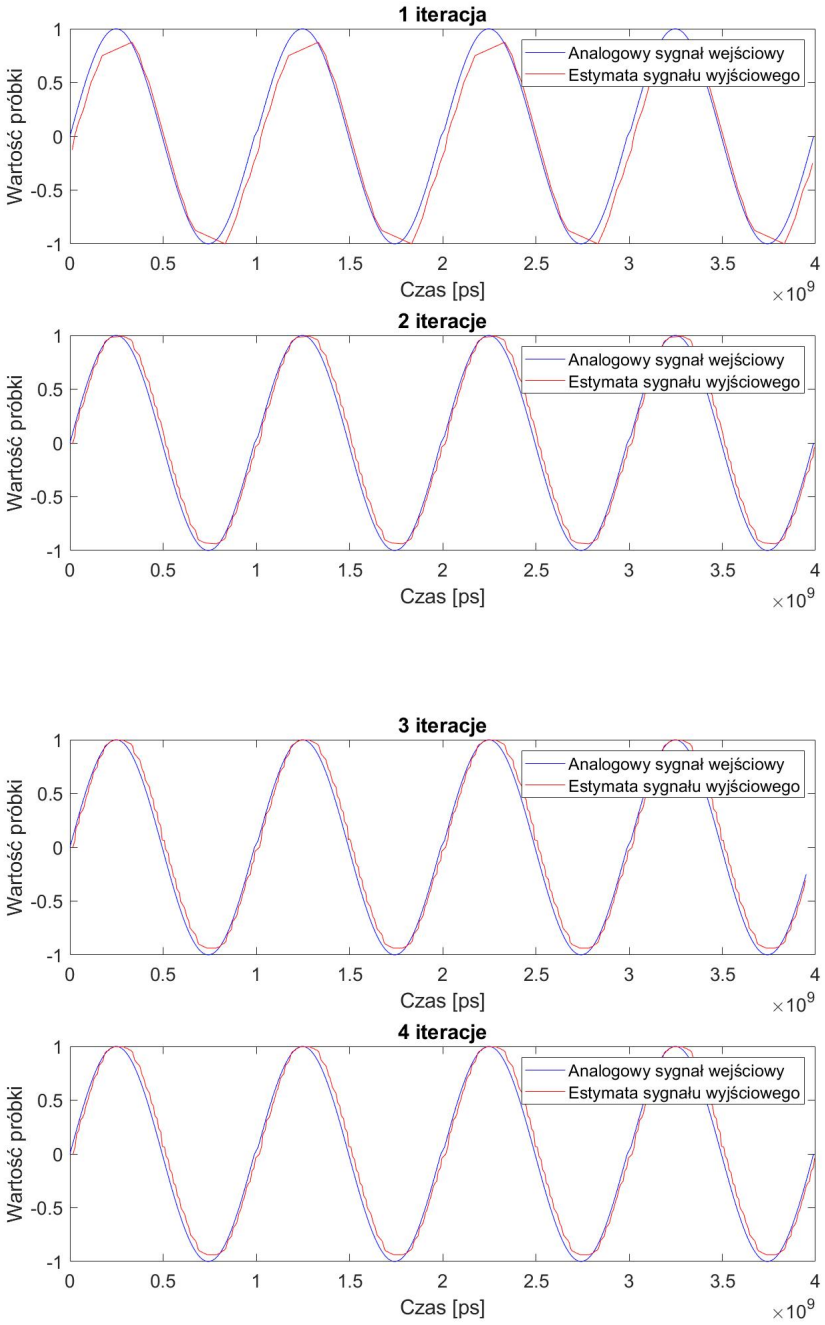
gdzie:

- wartość ENOB jest wyrażona w bitach;
- wartość SINAD (stosunek sygnału do szumu i zniekształcenia, *ang. signal-to-noise and distortion ratio*) jest wyrażona w dB.

Obliczenie wartości SINAD jest skomplikowanym procesem, ale istnieje wiele rozwiązań, które go ułatwiają. Wykorzystano pakiet zawierający narzędzia do przetwarzania sygnałów w środowisku Matlab, a w szczególności komendę $sinad(x)$, która oblicza wartość SINAD dla sygnału sinusoidalnego, używając zmodyfikowanego periodogramu oraz okna Kaisera.

Wykorzystane narzędzie nie jest idealne – do obliczenia wartości SINAD jest potrzebna duża liczba próbek. Na potrzeby pomiarów zdecydowano się na wykorzystanie ośmiu okresów sygnału sinusoidalnego o częstotliwości 1 kHz. Przyjęcie takiej metody nie daje stuprocentowej gwarancji otrzymania poprawnych wyników, jest to zgrubny pomiar jakościowy. Na jego podstawie można stwierdzić, jak zmienia się efektywna liczba bitów wraz ze zwiększaniem liczby iteracji.

Model klasyczny



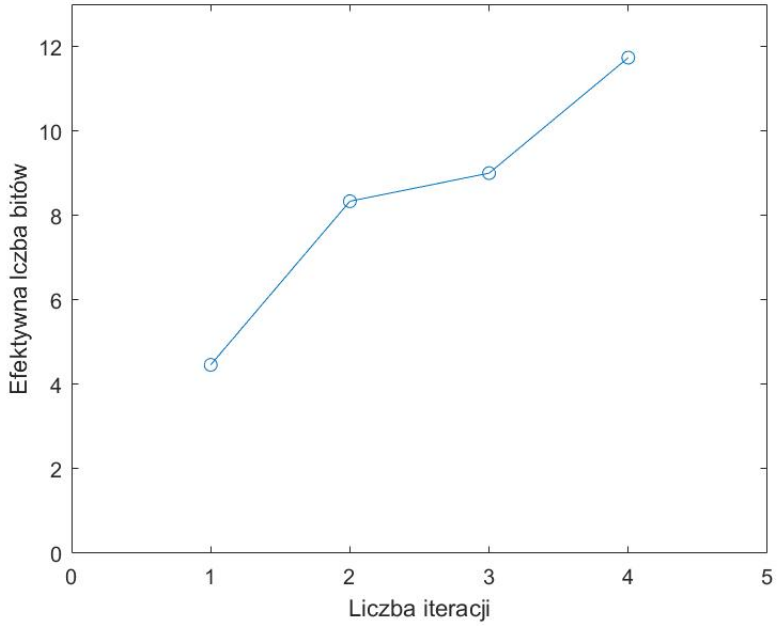
Rys. 4.13. Estymata sygnału wyjściowego w zależności od liczby iteracji w modelu klasycznym.

Wraz ze zwiększaniem liczby iteracji w modelu klasycznym można zauważyć, że sygnał nie osiąga zawsze wartości maksymalnej, mimo że we wcześniejszych testach ją osiągał. Wynika to z faktu, że kod estymaty wartości maksymalnej składa się z szestastu jedynek, a dopiero przy zastosowaniu czterech iteracji staje się możliwy zapis na wszystkich szesnastu bitach. Co więcej, wraz ze zwiększaniem liczby iteracji estymata coraz bardziej zbliża się kształtem do analogowego sygnału wejściowego.

Tab. 4.1. Zależność wartości efektywnej liczby bitów od liczby iteracji w modelu klasycznym.

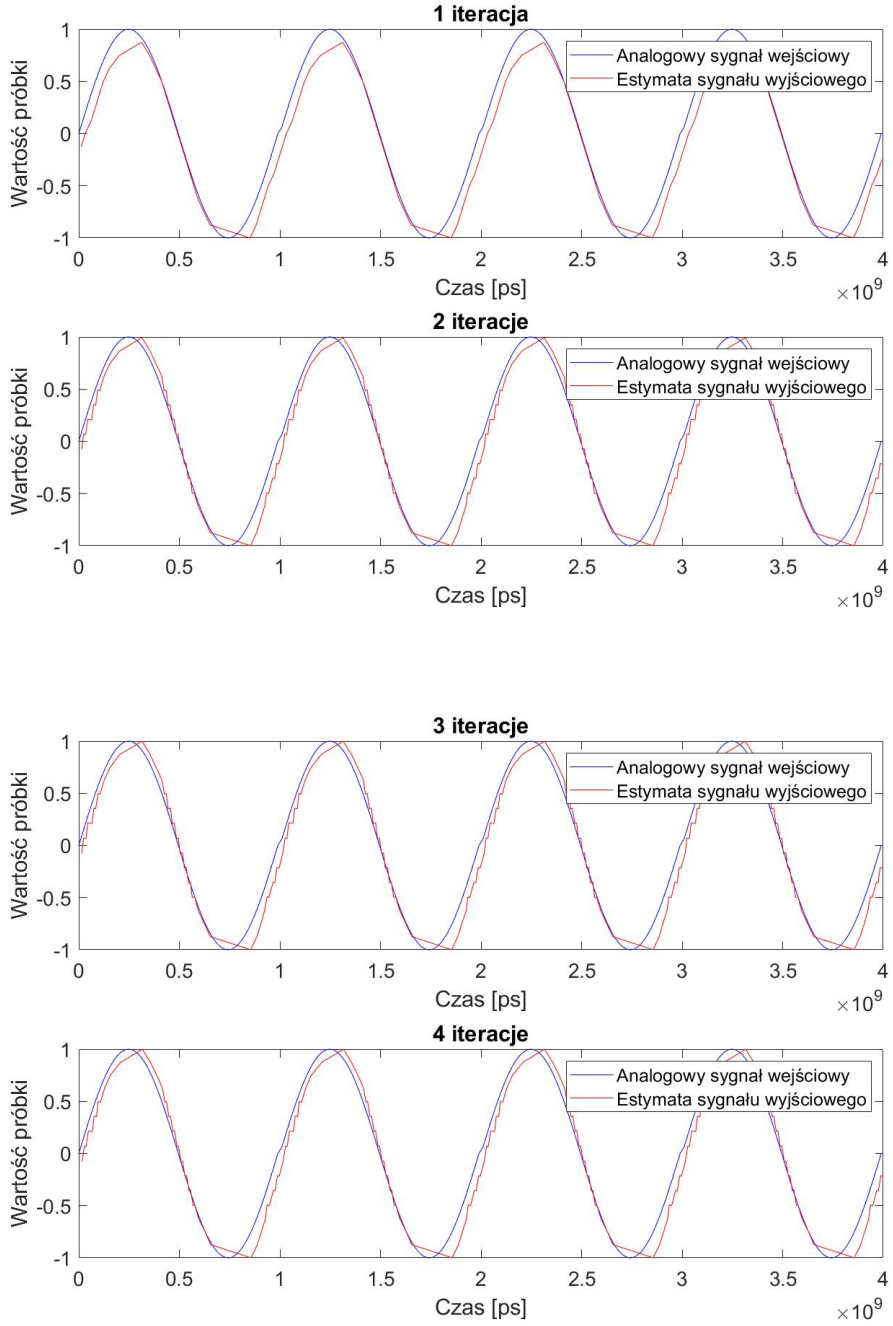
Liczba iteracji	Efektywna liczba bitów
1	4,4608
2	8,3349
3	9,0004
4	11,7353

Efektywna liczba bitów rośnie wraz z każdą dodaną iteracją. Otrzymane wyniki są zadowalające – nawet w przypadku jednej iteracji otrzymano wartość przekraczającą rozdzielczość wewnętrznego przetwornika A/C. Najmniejsza zmiana jest widoczna między dwiema a trzema iteracjami. Zastosowanie maksymalnej liczby iteracji pozwala uzyskać prawie trzy razy większą efektywną liczbę bitów niż rozdzielczość wewnętrznego przetwornika A/C.



Rys. 4.14. Wartość efektywnej liczby bitów w zależności od liczby iteracji w modelu klasycznym.

Model adaptacyjny



Rys. 4.15. Estymata sygnału wyjściowego w zależności od liczby iteracji w modelu adaptacyjnym.

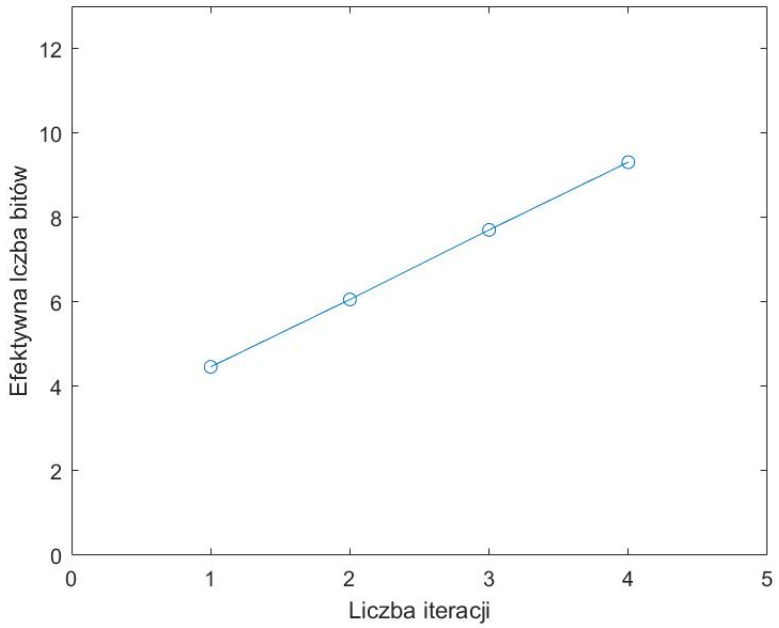
W przypadku modelu adaptacyjnego można wyciągnąć wnioski analogiczne do modelu klasycznego. Dla niepełnej liczby iteracji estymata nie osiąga wartości maksymalnej, a wraz ze zwiększaniem liczby iteracji estymata przypomina kształtem sygnał wejściowy w coraz większym stopniu.

Tab. 4.2. Zależność wartości efektywnej liczby bitów od liczby iteracji w modelu adaptacyjnym.

Liczba iteracji	Efektywna liczba bitów
1	4,4608
2	6,0571
3	7,7036
4	9,3070

Można dostrzec, że w przypadku zatosowania jednej iteracji wartość efektywnej liczby bitów jest identyczna jak w modelu klasycznym. Jest to spowodowane tożsamym mechanizmem działania obu modeli w takiej konfiguracji – modyfikacje modelu adaptacyjnego względem klasycznego występują od drugiej iteracji wzwyż. Dopiero wtedy uwidacznia się wpływ zastosowania wewnętrznego przetwornika C/A o niskiej rozdzielczości oraz zmiany sposobu działania układu próbkująco-pamiętającego.

Wartości efektywnej liczby bitów dla więcej niż jednej iteracji są mniejsze niż dla modelu klasycznego. Takie zjawisko nie jest zaskakujące – zastosowanie przetwornika C/A o mniejszej rozdzielczości wiąże się ze zmniejszeniem kosztu wykonania układu, ale niesie za sobą konsekwencje w postaci gorszych parametrów.



Rys. 4.16. Estymata sygnału wyjściowego w zależności od liczby iteracji w modelu klasycznym.

Chociaż charakterystyka działania w całym zakresie zbadana w punkcie 4.2.1 była wyraźnie gorsza w przypadku modelu adaptacyjnego, to wpływ na efektywną liczbę bitów okazał się mniejszy niż można by się spodziewać.

Tab. 4.3. Zależność wartości efektywnej liczby bitów od liczby iteracji w modelu adaptacyjnym.

Liczba iteracji	Model klasyczny	Model adaptacyjny	Różnica [%]
1	4,4608	4,4608	0
2	8,3349	6,0571	27.3285
3	9,0004	7,7036	14.4082
4	11,7353	9,3070	20,6923

Średnia różnica między wartościami efektywnej liczby bitów wynosi 15,6%. Biorąc pod uwagę fakt, że w modelu adaptacyjnym zastosowano wewnętrzny przetwornik C/A o cztery razy niższej rozdzielczości, otrzymane wyniki są satysfakcjonujące.

5. Podsumowanie

Zaimplementowano moduł sterowania i obliczania estymat dla klasycznego oraz adaptacyjnego cyklicznego przetwornika A/C. Każdy z cyfrowych bloków sterowania został zintegrowany z modelem części analogowej układu, a następnie poddany kilkuetapowej weryfikacji poprawności działania.

Modele cyklicznych przetworników A/C składają się z cyfrowego bloku sterowania oraz części analogowej zawierającej układ próbkująco-pamiętający, układ odejmujący, wzmacniacz sygnału błędu, wewnętrzny przetwornik A/C o niskiej rozdzielczości oraz wewnętrzny przetwornik C/A o rozdzielczości zależnej od modelu (wysoka rozdzielczość w modelu klasycznym oraz niska rozdzielczość w modelu adaptacyjnym). Model adaptacyjny stanowi modyfikację modelu klasycznego – użycie innego przetwornika C/A pozwala zmniejszyć koszt układu, jednak odbywa się to kosztem parametrów cyklicznego przetwornika A/C.

Oba moduły sterowania wykorzystują wewnętrzny przetwornik A/C o rozdzielczości 4 bitów do utworzenia 16-bitowego kodu estymaty bitów metodą konkatenacji kodów cząstkowych. W procesie weryfikacji dowiedziono, że takie algorytmy działania tworzą poprawnie działające cykliczne przetworniki A/C.

Parametrem, któremu poświęcono szczególną uwagę jest efektywna rozdzielczość. Dowiedziono, że jej wartość ulega zmianie, gdy stosuje się różną liczbę iteracji w cyklu pracy przetworników A/C. W obu modelach jednokrotne użycie 4-bitowego wewnętrznego przetwornika A/C pozwalało uzyskać efektywną liczbę bitów powyżej rozdzielczości używanego elementu. W modelu klasycznym, wykorzystującym wewnętrzny przetwornik C/A o wysokiej rozdzielczości, zaobserwowano wzrost efektywnej liczby bitów wraz ze wzrostem liczby wykonywanych iteracji do poziomu maksymalnego będącego niemal 3-krotnie większego niż rozdzielczość wewnętrznego przetwornika A/C. W przypadku modelu adaptacyjnego maksymalna efektywna liczba bitów była niespełna 2,4 raza większa, co również można uznać za satysfakcjonujący wynik.

Przedstawione rozwiązania mogą być zastosowane w rzeczywistych cyklicznych przetwornikach A/C. Opracowane modele cyfrowe zostały zoptymalizowane pod kątem zużycia zasobów układów cyfrowych, dołączenie ich instancji do układu nie powinno spowodować znacznego wzrostu poboru mocy. Stabilność i szybkość działania modułów sterowania zostały przetestowane metodą symulacji behawioralnej, przy użyciu sygnałów o wartościach stosowanych w rzeczywistych układach elektronicznych.

Kod estymaty, będący sygnałem wyjściowym modułów sterowania, może być łatwo zaadaptowany do wysłania do rzeczywistego elementu za pomocą dowolnego protokołu. Stworzone modele są przystosowane do modyfikacji, w celu uzyskania jak największej uniwersalności.

Nie można jednoznacznie stwierdzić, który z przedstawionych modeli jest lepszy, biorąc pod uwagę tylko złożoność i koszt implementacji części cyfrowej. Największe różnice występują bowiem w części analogowej, zwłaszcza w koszcie realizacji wewnętrznego przetwornika C/A. Oba modele charakteryzują się taką samą szybkością działania i stabilnością. Zużycie zasobów oraz łatwość modyfikacji również są na podobnym poziomie. Za modelem klasycznym przemawia wyższa efektywna liczba bitów, ale wytworzenie takiego układu wiąże się z większym kosztem spowodowanym użyciem wewnętrznego przetwornika C/A o wyższej rozdzielczości. Średnia różnica między wartościami efektywnej liczby bitów w obu modelach wynosi 15,6%, więc jest to zmiana zauważalna, ale nie kolosalna. Oba modele mają swoje zalety, a podstawą dokonania wyboru powinno być ustalenie, czy priorytetem są możliwie najlepsze parametry czy całkowity koszt wytworzenia układu.

Bibliografia

[1] Maloberti, F. (2014) Data Converters, Springer.

[2] Kester, W. (2005) The Data Conversion Handbook, Analog Devices, Inc.

[3] Murmann, B. and Boser, B.E. (2010) Digitally Assisted Pipeline ADCs. Theory and Implementation, Kluwer Academic Publishers, Boston.

[4] Jędrzejewski, K. (2014) Evolution of architectures and conversion algorithms in adaptive sub-ranging A/D converters, Proc. SPIE 9290, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014.

[5] Ahmed, I. (2010) Pipelined ADC Design and Enhancement Techniques, Springer.

[6] Ruiz-Amaya, J., Delgado-Restituto, M. and Rodríguez-Vázquez, Á. (2011) Device-Level Modeling and Synthesis of High-Performance Pipeline ADCs, Springer.

[7] Kulka Z., Libura A., Nadachowski M. (1987) Przetworniki analogowo-cyfrowe i cyfrowo-analogowe, WKŁ, Warszawa.

[8] Analog Devices, Inc., LF198 Data Sheet, https://www.analog.com/media/en/technical-documentation/data-sheets/lt0198.pdf

[9] Ilustracja charakterystyki przenoszenia filtru pasmowoprzepustowego (Rys. 2.1), Dostęp zdalny (08.01.2022): https://pl.wikipedia.org/wiki/Filtr_%C5%9Brodkowoprzepustowy#/media/Plik:Bandwidth_2.svg.

[10] Ilustracja próbkowania idealnego (Rys. 2.2), Dostęp zdalny (08.01.2022): https://pl.wikipedia.org/wiki/Pr%C3%B3bkowanie#/media/Plik:Sampled.signal.svg

[11] Ilustracja budowy 2-bitowego przetwornika A/C o porównaniu bezpośrednim (Rys. 2.3), Dostęp zdalny (08.01.2022): https://en.wikipedia.org/wiki/Flash_ADC#/media/File:Flash_ADC.png

[12] Ilustracja zasady tworzenia wynikowych kodów estymat w cyklicznym przetworniku A/C (Rys. 2.4), Opracowanie własne na podstawie: Jędrzejewski, K. (2014) Evolution of architectures and conversion algorithms in adaptive sub-ranging A/D converters, Proc. SPIE 9290, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014.

[13] Ilustracja koncepcji budowy klasycznego cyklicznego przetwornika A/C (Rys. 2.5), Opracowanie własne na podstawie: Jędrzejewski, K. (2014) Evolution of architectures and conversion algorithms in adaptive sub-ranging A/D converters, Proc. SPIE 9290, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014.

[14] Ilustracja koncepcji budowy adaptacyjnego cyklicznego przetwornika A/C (Rys. 2.6), Opracowanie własne na podstawie: Jędrzejewski, K. (2014) Evolution of architectures and conversion algorithms in adaptive sub-ranging A/D converters, Proc. SPIE 9290, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014.

Spis rysunków

Spis tabel

Spis załączników

Wykaz symboli i skrótów

ENOB – efektywna liczba bitów (*ang. effective number of bits*)
LSB – najmniej znaczący bit (*ang. least significant bit*)
MSB – najbardziej znaczący bit (*ang. most significant bit*)
USB – jeden z systemów kodowania unipolarnego (*ang. Unipolar Straight Binary*)

Załącznik 1. Część cyfrowa modelu klasycznego przetwornika A/C

```
1  ------------------------------------------------
2  --
3  -- Classic cyclic ADC
4  --
5  -- Author: Aleksandra Antoszewska
6  --
7  ------------------------------------------------
8  LIBRARY IEEE;
9  USE IEEE.STD_LOGIC_1164.ALL;
10 USE IEEE.NUMERIC_STD.ALL;
11
12 ENTITY digital_part_classic IS
13     GENERIC (
14         -- number of iterations in cyclic ADC
15         G_number_of_iterations      : integer RANGE 1 TO 4 := 4
16     );
17     PORT (
18         -- SYSTEM
19         i_clk                       : IN    std_logic;
20         i_reset                     : IN    std_logic;
21
22         -- INPUT
23         i_yk_SADC                   : IN    std_logic_vector(3 DOWNTO 0);
24
25         --OUTPUT
26         o_Ck                        : OUT   integer;
                   -- multiplier
27         o_V_digital_part            : OUT   std_logic_vector(15 DOWNTO 0);
                -- loopback
28         o_output_data               : OUT   std_logic_vector(15 DOWNTO 0)
                 -- final data
29     );
30 END ENTITY;
31
32 ARCHITECTURE behav OF digital_part_classic IS
33     ---------------
34     -- Constants --
35     ---------------
36
37     -- Clock frequency
38     CONSTANT C_clk_freq         :   integer := 50_000_000;
39     -- Sampling frequency
40     CONSTANT C_sampling_freq    :   integer := 1_000_000;
41     -- Sampling counter range
42     CONSTANT C_sampling_cnt_mod :   integer := C_clk_freq / C_sampling_freq
           ;
43
44     -------------
45     -- Signals --
46     -------------
47
48     -- Counters --
49     -- Iteration counter
50     SIGNAL r_iteration_cnt          : integer RANGE 1 TO
           G_number_of_iterations := 1;
51     -- Sampling time counter
52     SIGNAL r_sampling_cnt           : integer RANGE 0 TO C_sampling_cnt_mod
           -1   := 0;
53
54     -- Flags --
55     -- data ready flag
56     SIGNAL r_data_ready             : std_logic := '0';
57     -- data ready for SDAC flag
58     SIGNAL r_data_ready_sdac        : std_logic := '0';
```

```
59      -- new data arrived flag
60      SIGNAL r_new_data               : std_logic := '0';
61
62      -- Multiplier register
63      SIGNAL r_Ck                     : integer   := 1;
64      -- Output data register
65      SIGNAL r_output_data            : std_logic_vector(15 DOWNTO 0) := (
            OTHERS => '0');
66
67  BEGIN
68      ------------------
69      -- Assignmnents --
70      ------------------
71
72      -- Output enabled after all iterations done
73      o_output_data       <=  r_output_data WHEN (r_data_ready = '1');
74
75      -- SDAC input enabled when flag is set, otherwise input closest to 0V
76      o_V_digital_part    <=  r_output_data WHEN (r_data_ready_sdac = '1')
            ELSE
77                              (15 => '1', OTHERS => '0');
78
79      -- Multiplier
80      o_Ck                <= r_Ck;
81
82      ------------------------------
83      -- Processes and procedures --
84      ------------------------------
85
86      P_Ck: PROCESS (i_clk)
87      BEGIN
88          IF (rising_edge(i_clk)) THEN
89              CASE r_iteration_cnt IS
90                  WHEN 1 =>
91                      r_Ck <= 1;
92                  WHEN 2 =>
93                      r_Ck <= 8;
94                  WHEN 3 =>
95                      r_Ck <= 16;
96                  WHEN 4 =>
97                      r_Ck <= 16;
98                  WHEN OTHERS =>
99                      r_Ck <= 1;
100             END CASE;
101         END IF;
102     END PROCESS;
103
104     P_write_to_vector: PROCESS (i_clk)
105     BEGIN
106         IF (rising_edge(i_clk)) THEN
107             IF(i_reset = '1') THEN
108                 r_output_data <= (OTHERS => '0');
109             ELSE
110                 IF (r_new_data = '1') THEN -- protection from accidential
                        output change
111                     IF (G_number_of_iterations = 1) THEN
112                         r_output_data(15 DOWNTO 12) <= i_yk_SADC;
113                         r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
114                         r_data_ready  <= '1';
115                         r_data_ready_sdac <= '0';
116
117                     ELSIF (G_number_of_iterations = 2) THEN
118                         IF (r_iteration_cnt = 1) THEN
119                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
120                             r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
121                             r_data_ready  <= '0';
122                             r_data_ready_sdac <= '1';
```

```
123                             r_iteration_cnt <= 2;
124                         ELSIF (r_iteration_cnt = 2) THEN
125                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
126                             r_output_data( 7 DOWNTO 0) <= (OTHERS => '0');
127                             r_data_ready  <= '1';
128                             r_data_ready_sdac <= '0';
129                             r_iteration_cnt <= 1;
130                         END IF;
131
132                     ELSIF (G_number_of_iterations = 3) THEN
133                         IF (r_iteration_cnt = 1) THEN
134                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
135                             r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
136                             r_data_ready  <= '0';
137                             r_data_ready_sdac <= '1';
138                             r_iteration_cnt <= 2;
139                         ELSIF (r_iteration_cnt = 2) THEN
140                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
141                             r_output_data( 7 DOWNTO 0) <= (OTHERS => '0');
142                             r_iteration_cnt <= 3;
143                         ELSIF (r_iteration_cnt = 3) THEN
144                             r_output_data(7 DOWNTO 4) <= i_yk_SADC;
145                             r_output_data(3 DOWNTO 0) <= (OTHERS => '0');
146                             r_data_ready  <= '1';
147                             r_data_ready_sdac <= '0';
148                             r_iteration_cnt <= 1;
149                         END IF;
150
151                     ELSIF (G_number_of_iterations = 4) THEN
152                         IF (r_iteration_cnt = 1) THEN
153                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
154                             r_output_data(11 DOWNTO 0) <= (OTHERS => '0');
155                             r_data_ready  <= '0';
156                             r_data_ready_sdac <= '1';
157                             r_iteration_cnt <= 2;
158                         ELSIF (r_iteration_cnt = 2) THEN
159                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
160                             r_output_data(7 DOWNTO 0) <= (OTHERS => '0');
161                             r_iteration_cnt <= 3;
162                         ELSIF (r_iteration_cnt = 3) THEN
163                             r_output_data(7 DOWNTO 4) <= i_yk_SADC;
164                             r_output_data(3 DOWNTO 0) <= (OTHERS => '0');
165                             r_iteration_cnt <= 4;
166                         ELSIF (r_iteration_cnt = 4) THEN
167                             r_output_data(3 DOWNTO 0) <= i_yk_SADC;
168                             r_data_ready  <= '1';
169                             r_data_ready_sdac <= '0';
170                             r_iteration_cnt <= 1;
171                         END IF;
172                     END IF;
173                 END IF;
174             END IF;
175         END IF;
176     END PROCESS;
177
178     -- Sampling time measurement based on system clock
179     P_sampling_cnt: PROCESS (i_clk)
180     BEGIN
181         IF (rising_edge(i_clk)) THEN
182             IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
183                 r_sampling_cnt <= 0;
184             ELSE
185                 r_sampling_cnt <= r_sampling_cnt + 1;
186             END IF;
187         END IF;
188     END PROCESS;
189
```

```
190      -- Sampling
191      P_new_data: PROCESS (i_clk)
192      BEGIN
193          IF (rising_edge(i_clk)) THEN
194              IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
195                  r_new_data <= '1';
196              ELSE
197                  r_new_data <= '0';
198              END IF;
199          END IF;
200      END PROCESS;
201
202  END ARCHITECTURE;
```

Załącznik 2. Testbench z modelem części analogowej modelu klasycznego przetwornika A/C

```
1  ------------------------------------------------
2  --
3  -- Classic cyclic ADC testbench
4  --
5  -- Author: Aleksandra Antoszewska
6  --
7  ------------------------------------------------
8  LIBRARY IEEE;
9  USE IEEE.STD_LOGIC_1164.ALL;
10 USE IEEE.NUMERIC_STD.ALL;
11
12 ENTITY classic_cyclic_ADC IS
13 END classic_cyclic_ADC;
14
15 ARCHITECTURE classic_cyclic_ADC_arch OF classic_cyclic_ADC IS
16     -----------
17     -- Types --
18     -----------
19
20     TYPE   T_data_array     IS ARRAY(0 TO  99) OF real;
21     TYPE   T_data_array_2   IS ARRAY(0 TO 200) OF real;
22
23     ---------------
24     -- Constants --
25     ---------------
26
27     -- Clock frequency
28     CONSTANT C_clk_freq         :   integer := 50_000_000;
29     -- Number of iterations
30     CONSTANT C_iteration_number :   integer := 4;
31
32     -- Sample and hold circuit frequency can be changed due to usage of
           timer
33     -- Sampling frequency [Hz]
34     CONSTANT C_sampling_freq    :   integer := 250_000;
35     -- Sampling counter range
36     CONSTANT C_sampling_cnt_mod :   integer := C_clk_freq / C_sampling_freq;
37
38     -- ADC range
39     CONSTANT C_ADC_range        :   real    := 1.0;
40     -- ADC step
41     CONSTANT C_ADC_step         :   real    := (2.0*C_ADC_range)/16.0;
42     -- DAC range
43     CONSTANT C_DAC_range        :   real    := 1.0;
44     -- DAC step
45     CONSTANT C_DAC_step         :   real    := (2.0*C_DAC_range)/2.0**16;
46
47     -------------
48     -- Signals --
49     -------------
50
51     -- System --
52
53     -- Clock
54     SIGNAL r_clk                        :   std_logic := '0';
55     -- Reset
56     SIGNAL r_reset                      :   std_logic := '0';
57     -- Sampling time counter
58     SIGNAL r_sampling_cnt               :   integer RANGE 0 TO
           C_sampling_cnt_mod-1 := 0;
59
60     -- Analog circuit registers --
```

```
61
62      -- Input signal register
63      SIGNAL r_Vt                         :   real    := 0.0;
64      -- Sample and hold output register
65      SIGNAL r_V_SH                       :   real    := 0.0;
66      -- Substraction output register
67      SIGNAL r_ek                         :   real    := 0.0;
68      -- Amplifier output register
69      SIGNAL r_Ck_ek                      :   real    := 0.0;
70      -- Multiplier - digital part output
71      SIGNAL r_Ck                         :   integer := 1;
72      -- ADC - quantum counter register
73      SIGNAL r_quantum_cnt                :   real    := 0.0;
74      -- integer quantum counter register used to activate SADC
75      SIGNAL r_int_quantum_cnt            :   integer := 0;
76      -- SADC output register
77      SIGNAL r_yk_SADC                    :   std_logic_vector(3 DOWNTO 0) :=
            (OTHERS => '0');
78      -- SDAC input register
79      SIGNAL r_V_digital_part             :   unsigned(15 DOWNTO 0) := (OTHERS
            => '0');
80      -- SDAC input converted to std_logic_vector
81      SIGNAL r_V_digital_part_vector      :   std_logic_vector(15 DOWNTO 0) :=
            (OTHERS => '0');
82      -- SDAC output register
83      SIGNAL r_V_SDAC                     :   real := 0.0;
84
85      -- Output data and final estimation --
86
87      -- Final data vector
88      SIGNAL r_output_data                :   std_logic_vector(15 DOWNTO 0) :=
            (OTHERS => '0');
89      -- Final data converted to unsigned
90      SIGNAL r_output_data_unsigned       :   unsigned(15 DOWNTO 0) := (OTHERS
            => '0');
91      -- Final data estimation
92      SIGNAL r_estimation                 :   real := 0.0;
93
94      -- Simulated data --
95
96      -- Counter used to change input data
97      SIGNAL r_cnt            : integer RANGE 0 TO 200            := 0;
98      -- Input data (sine wave)
99      SIGNAL r_sine   : T_data_array      := (0.0, 0.0634239196565645,
            0.126592453573749, 0.189251244360410, 0.251147987181079,
            0.312033445698487, 0.371662455660328, 0.429794912089172,
            0.486196736100469, 0.540640817455598, 0.592907929054640,
            0.642787609686539, 0.690079011482112, 0.734591708657533,
            0.776146464291757, 0.814575952050336, 0.849725429949514,
            0.881453363447582, 0.909631995354518, 0.934147860265107,
            0.954902241444074, 0.971811568323542, 0.984807753012208,
            0.993838464461254, 0.998867339183008, 0.999874127673875,
            0.996854775951942, 0.989821441880933, 0.978802446214779,
            0.963842158559942, 0.945000818714669, 0.922354294104582,
            0.895993774291336, 0.866025403784439, 0.832569854634771,
            0.795761840530832, 0.755749574354258, 0.712694171378863,
            0.666769000516292, 0.618158986220606, 0.567059863862771,
            0.513677391573407, 0.458226521727411, 0.400930535406614,
            0.342020143325669, 0.281732556841430, 0.220310532786541,
            0.158001395973350, 0.0950560433041829, 0.0317279334980681,
            -0.0317279334980679, -0.0950560433041826, -0.158001395973350,
            -0.220310532786541, -0.281732556841429, -0.342020143325669,
            -0.400930535406613, -0.458226521727410, -0.513677391573406,
            -0.567059863862771, -0.618158986220605, -0.666769000516292,
            -0.712694171378863, -0.755749574354258, -0.795761840530832,
            -0.832569854634771, -0.866025403784439, -0.895993774291336,
            -0.922354294104581, -0.945000818714668, -0.963842158559942,
```

```
            -0.978802446214779, -0.989821441880933, -0.996854775951942,
            -0.999874127673875, -0.998867339183008, -0.993838464461254,
            -0.984807753012208, -0.971811568323542, -0.954902241444074,
            -0.934147860265107, -0.909631995354519, -0.881453363447582,
            -0.849725429949514, -0.814575952050336, -0.776146464291757,
            -0.734591708657534, -0.690079011482113, -0.642787609686540,
            -0.592907929054640, -0.540640817455597, -0.486196736100469,
            -0.429794912089172, -0.371662455660328, -0.312033445698487,
            -0.251147987181079, -0.189251244360411, -0.126592453573750,
            -0.0634239196565654, 0.0);
100     -- Input data (linear)
101     SIGNAL r_linear : T_data_array_2    := (-1.0, -0.99, -0.98, -0.97,
            -0.96, -0.95, -0.94, -0.93, -0.92, -0.91, -0.90, -0.89, -0.88,
            -0.87, -0.86, -0.85, -0.84, -0.83, -0.82, -0.81, -0.80, -0.79,
            -0.78, -0.77, -0.76, -0.75, -0.74, -0.73, -0.72, -0.71, -0.70,
            -0.69, -0.68, -0.67, -0.66, -0.65, -0.64, -0.63, -0.62, -0.61,
            -0.60, -0.59, -0.58, -0.57, -0.56, -0.55, -0.54, -0.53, -0.52,
            -0.51, -0.50, -0.49, -0.48, -0.47, -0.46, -0.45, -0.44, -0.43,
            -0.42, -0.41, -0.40, -0.39, -0.38, -0.37, -0.36, -0.35, -0.34,
            -0.33, -0.32, -0.31, -0.30, -0.29, -0.28, -0.27, -0.26, -0.25,
            -0.24, -0.23, -0.22, -0.21, -0.20, -0.19, -0.18, -0.17, -0.16,
            -0.15, -0.14, -0.13, -0.12, -0.11, -0.10, -0.09, -0.08, -0.07,
            -0.06, -0.05, -0.04, -0.03, -0.02, -0.01, 0.0, 0.01, 0.02, 0.03,
            0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14,
            0.15, 0.16, 0.17, 0.18, 0.19, 0.20, 0.21, 0.22, 0.23, 0.24, 0.25,
            0.26, 0.27, 0.28, 0.29, 0.30, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36,
            0.37, 0.38, 0.39, 0.40, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47,
            0.48, 0.49, 0.50, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58,
            0.59, 0.60, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69,
            0.70, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.80,
            0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.90, 0.91,
            0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0);
102
103 BEGIN
104
105     ------------------
106     -- Assignmnents --
107     ------------------
108
109     r_quantum_cnt <= (r_Ck_ek + C_ADC_range) / C_ADC_step;
110
111     -- Conversions --
112
113     r_int_quantum_cnt <= integer(r_quantum_cnt);
114
115     r_V_digital_part <= unsigned(r_V_digital_part_vector);
116
117     r_output_data_unsigned <= unsigned(r_output_data);
118
119     ---------------
120     -- Instances --
121     ---------------
122
123     Digital_part_inst: ENTITY work.digital_part_classic
124     GENERIC  MAP (
125         G_number_of_iterations  => C_iteration_number
126     )
127     PORT MAP (
128         i_clk                   => r_clk,
129         i_reset                 => r_reset,
130         i_yk_SADC               => r_yk_SADC,
131         o_Ck                    => r_Ck,
132         o_V_digital_part        => r_V_digital_part_vector,
133         o_output_data           => r_output_data
134     );
135
136     ------------------------------
```

```
137     -- Processes and procedures --
138     ------------------------------
139
140     P_clk: PROCESS
141     BEGIN
142         r_clk <= '0';
143         WAIT FOR 10 ns;
144         r_clk <= '1';
145         WAIT FOR 10 ns;
146     END PROCESS;
147
148     P_sampling_cnt: PROCESS (r_clk)
149     BEGIN
150         IF (rising_edge(r_clk)) THEN
151             IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
152                 r_sampling_cnt <= 0;
153             ELSE
154                 r_sampling_cnt <= r_sampling_cnt + 1;
155             END IF;
156         END IF;
157     END PROCESS;
158
159
160     P_SH: PROCESS (r_clk)
161     BEGIN
162         IF (rising_edge(r_clk)) THEN
163             IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
164                 r_V_SH <= r_Vt;
165             END IF;
166         END IF;
167     END PROCESS;
168
169     -- Propagation delay set to 100ns
170     P_Substraction: PROCESS (r_V_SH, r_V_SDAC)
171     BEGIN
172         r_ek <= r_V_SH - r_V_SDAC AFTER 100 ns;
173     END PROCESS;
174
175     -- Propagation delay set to 100ns
176     P_Amplifying: PROCESS (r_ek) --r_Ck
177     BEGIN
178         r_Ck_ek <= real(r_Ck) * r_ek AFTER 100 ns;
179     END PROCESS;
180
181     -- Propagation delay set to 100ns
182     P_ADC: PROCESS (r_quantum_cnt)
183     BEGIN
184         IF    (r_quantum_cnt < 1.0) THEN
185             r_yk_SADC <= "0000" AFTER 100 ns;
186         ELSIF (r_quantum_cnt < 2.0) THEN
187             r_yk_SADC <= "0001" AFTER 100 ns;
188         ELSIF (r_quantum_cnt < 3.0) THEN
189             r_yk_SADC <= "0010" AFTER 100 ns;
190         ELSIF (r_quantum_cnt < 4.0) THEN
191             r_yk_SADC <= "0011" AFTER 100 ns;
192         ELSIF (r_quantum_cnt < 5.0) THEN
193             r_yk_SADC <= "0100" AFTER 100 ns;
194         ELSIF (r_quantum_cnt < 6.0) THEN
195             r_yk_SADC <= "0101" AFTER 100 ns;
196         ELSIF (r_quantum_cnt < 7.0) THEN
197             r_yk_SADC <= "0110" AFTER 100 ns;
198         ELSIF (r_quantum_cnt < 8.0) THEN
199             r_yk_SADC <= "0111" AFTER 100 ns;
200         ELSIF (r_quantum_cnt < 9.0) THEN
201             r_yk_SADC <= "1000" AFTER 100 ns;
202         ELSIF (r_quantum_cnt < 10.0) THEN
203             r_yk_SADC <= "1001" AFTER 100 ns;
```

```
204         ELSIF (r_quantum_cnt < 11.0) THEN
205             r_yk_SADC <= "1010" AFTER 100 ns;
206         ELSIF (r_quantum_cnt < 12.0) THEN
207             r_yk_SADC <= "1011" AFTER 100 ns;
208         ELSIF (r_quantum_cnt < 13.0) THEN
209             r_yk_SADC <= "1100" AFTER 100 ns;
210         ELSIF (r_quantum_cnt < 14.0) THEN
211             r_yk_SADC <= "1101" AFTER 100 ns;
212         ELSIF (r_quantum_cnt < 15.0) THEN
213             r_yk_SADC <= "1110" AFTER 100 ns;
214         ELSIF (r_quantum_cnt > 15.0) THEN
215             r_yk_SADC <= "1111" AFTER 100 ns;
216         END IF;
217     END PROCESS;
218
219     -- Propagation delay set to 100ns
220     P_DAC: PROCESS (r_V_digital_part)
221     BEGIN
222         r_V_SDAC <= (real(to_integer(r_V_digital_part))*C_DAC_step -
                C_DAC_range + C_DAC_step/2.0) AFTER 100 ns;
223     END PROCESS;
224
225     -- Final estimation count
226     P_estimation: PROCESS (r_output_data)
227     BEGIN
228         r_estimation <= (real(to_integer(r_output_data_unsigned))*C_DAC_step
                - C_DAC_range + C_DAC_step/2.0);
229     END PROCESS;
230
231     -- Sine wave simulation
232     P_input_data_simulation: PROCESS
233     BEGIN
234         IF (r_cnt < 99) THEN
235             r_cnt <= r_cnt + 1;
236             r_Vt <= r_sine(r_cnt);
237             WAIT FOR 4 us;
238         ELSE
239             r_cnt <= 0;
240             r_Vt <= r_sine(r_cnt);
241             WAIT FOR 4 us;
242         END IF;
243     END PROCESS;
244
245 END classic_cyclic_ADC_arch;
```

Załącznik 3. Część cyfrowa modelu adaptacyjnego przetwornika A/C

```
1  ------------------------------------------------
2  --
3  -- Adaptive cyclic ADC
4  --
5  -- Author: Aleksandra Antoszewska
6  --
7  ------------------------------------------------
8  LIBRARY IEEE;
9  USE IEEE.STD_LOGIC_1164.ALL;
10 USE IEEE.NUMERIC_STD.ALL;
11
12 ENTITY digital_part_adaptive IS
13     GENERIC (
14         -- number of iterations in cyclic ADC
15         G_number_of_iterations      : integer RANGE 1 TO 4 := 4
16     );
17     PORT (
18         -- SYSTEM
19         i_clk                       : IN    std_logic;
20         i_reset                     : IN    std_logic;
21
22         -- INPUT
23         i_yk_SADC                   : IN    std_logic_vector(3 DOWNTO 0);
24
25         --OUTPUT
26         o_Ck                        : OUT   integer;
               -- multiplier
27         o_V_digital_part            : OUT   std_logic_vector( 3 DOWNTO 0);
               -- loopback
28         o_output_data               : OUT   std_logic_vector(15 DOWNTO 0)
               -- final data
29     );
30 END ENTITY;
31
32 ARCHITECTURE behav OF digital_part_adaptive IS
33     ---------------
34     -- Constants --
35     ---------------
36
37     -- Clock frequency
38     CONSTANT C_clk_freq         :   integer := 50_000_000;
39     -- Sampling frequency
40     CONSTANT C_sampling_freq    :   integer := 1_000_000;
41     -- Sampling counter range
42     CONSTANT C_sampling_cnt_mod :   integer := C_clk_freq / C_sampling_freq;
43
44     -------------
45     -- Signals --
46     -------------
47
48     -- Counters --
49     -- Iteration counter
50     SIGNAL r_iteration_cnt          : integer RANGE 1 TO
           G_number_of_iterations := 1;
51     -- Sampling time counter
52     SIGNAL r_sampling_cnt           : integer RANGE 0 TO C_sampling_cnt_mod
           -1   := 0;
53
54     -- Flags --
55     -- data ready flag
56     SIGNAL r_data_ready             : std_logic := '0';
57     -- data ready for SDAC flag
58     SIGNAL r_data_ready_sdac        : std_logic := '0';
59     -- new data arrived flag
```

```
60      SIGNAL r_new_data               : std_logic := '0';
61
62      -- Multiplier register
63      SIGNAL r_Ck                     : integer   := 1;
64      -- Output data register
65      SIGNAL r_output_data            : std_logic_vector(15 DOWNTO 0) := (
            OTHERS => '0');
66
67  BEGIN
68      ------------------
69      -- Assignmnents --
70      ------------------
71
72      -- Output enabled after all iterations done
73      o_output_data       <=  r_output_data WHEN (r_data_ready = '1');
74
75      -- SDAC input enabled when flag is set, otherwise input closest to 0V
76      o_V_digital_part    <=  i_yk_SADC WHEN (r_data_ready_sdac = '1') ELSE
77                              (3 => '1', OTHERS => '0');
78
79      -- Multiplier
80      o_Ck                <= r_Ck;
81
82      ------------------------------
83      -- Processes and procedures --
84      ------------------------------
85
86      P_Ck: PROCESS (i_clk)
87      BEGIN
88          IF (rising_edge(i_clk)) THEN
89              CASE r_iteration_cnt IS
90                  WHEN 1 =>
91                      r_Ck <= 1;
92                  WHEN 2 =>
93                      r_Ck <= 2;
94                  WHEN 3 =>
95                      r_Ck <= 2;
96                  WHEN 4 =>
97                      r_Ck <= 2;
98                  WHEN OTHERS =>
99                      r_Ck <= 1;
100             END CASE;
101         END IF;
102     END PROCESS;
103
104     P_write_to_vector: PROCESS (i_clk)
105     BEGIN
106         IF (rising_edge(i_clk)) THEN
107             IF(i_reset = '1') THEN
108                 r_output_data <= (OTHERS => '0');
109             ELSE
110                 IF (r_new_data = '1') THEN -- protection from accidential
                        output change
111                     IF (G_number_of_iterations = 1) THEN
112                         r_output_data(15 DOWNTO 12) <= i_yk_SADC;
113                         r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
114                         r_data_ready  <= '1';
115                         r_data_ready_sdac <= '0';
116
117                     ELSIF (G_number_of_iterations = 2) THEN
118                         IF (r_iteration_cnt = 1) THEN
119                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
120                             r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
121                             r_data_ready  <= '0';
122                             r_data_ready_sdac <= '1';
123                             r_iteration_cnt <= 2;
124                         ELSIF (r_iteration_cnt = 2) THEN
```

```
125                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
126                             r_output_data( 7 DOWNTO 0) <= (OTHERS => '0');
127                             r_data_ready  <= '1';
128                             r_data_ready_sdac <= '0';
129                             r_iteration_cnt <= 1;
130                         END IF;
131
132                     ELSIF (G_number_of_iterations = 3) THEN
133                         IF (r_iteration_cnt = 1) THEN
134                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
135                             r_output_data(11 DOWNTO  0) <= (OTHERS => '0');
136                             r_data_ready  <= '0';
137                             r_data_ready_sdac <= '1';
138                             r_iteration_cnt <= 2;
139                         ELSIF (r_iteration_cnt = 2) THEN
140                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
141                             r_output_data( 7 DOWNTO 0) <= (OTHERS => '0');
142                             r_iteration_cnt <= 3;
143                         ELSIF (r_iteration_cnt = 3) THEN
144                             r_output_data(7 DOWNTO 4) <= i_yk_SADC;
145                             r_output_data(3 DOWNTO 0) <= (OTHERS => '0');
146                             r_data_ready  <= '1';
147                             r_data_ready_sdac <= '0';
148                             r_iteration_cnt <= 1;
149                         END IF;
150
151                     ELSIF (G_number_of_iterations = 4) THEN
152                         IF (r_iteration_cnt = 1) THEN
153                             r_output_data(15 DOWNTO 12) <= i_yk_SADC;
154                             r_output_data(11 DOWNTO 0) <= (OTHERS => '0');
155                             r_data_ready  <= '0';
156                             r_data_ready_sdac <= '1';
157                             r_iteration_cnt <= 2;
158                         ELSIF (r_iteration_cnt = 2) THEN
159                             r_output_data(11 DOWNTO 8) <= i_yk_SADC;
160                             r_output_data(7 DOWNTO 0) <= (OTHERS => '0');
161                             r_iteration_cnt <= 3;
162                         ELSIF (r_iteration_cnt = 3) THEN
163                             r_output_data(7 DOWNTO 4) <= i_yk_SADC;
164                             r_output_data(3 DOWNTO 0) <= (OTHERS => '0');
165                             r_iteration_cnt <= 4;
166                         ELSIF (r_iteration_cnt = 4) THEN
167                             r_output_data(3 DOWNTO 0) <= i_yk_SADC;
168                             r_data_ready  <= '1';
169                             r_data_ready_sdac <= '0';
170                             r_iteration_cnt <= 1;
171                         END IF;
172                     END IF;
173                 END IF;
174             END IF;
175         END IF;
176     END PROCESS;
177
178     -- Sampling time measurement based on system clock
179     P_sampling_cnt: PROCESS (i_clk)
180     BEGIN
181         IF (rising_edge(i_clk)) THEN
182             IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
183                 r_sampling_cnt <= 0;
184             ELSE
185                 r_sampling_cnt <= r_sampling_cnt + 1;
186             END IF;
187         END IF;
188     END PROCESS;
189
190     -- Sampling
191     P_new_data: PROCESS (i_clk)
```

```
192     BEGIN
193         IF (rising_edge(i_clk)) THEN
194             IF (r_sampling_cnt = C_sampling_cnt_mod-1) THEN
195                 r_new_data <= '1';
196             ELSE
197                 r_new_data <= '0';
198             END IF;
199         END IF;
200     END PROCESS;
201
202 END ARCHITECTURE;
```

Załącznik 4. Testbench z modelem części analogowej modelu adaptacyjnego przetwornika A/C

```
1  ------------------------------------------------
2  --
3  -- Adaptive cyclic ADC testbench
4  --
5  -- Author: Aleksandra Antoszewska
6  --
7  ------------------------------------------------
8  LIBRARY IEEE;
9  USE IEEE.STD_LOGIC_1164.ALL;
10 USE IEEE.NUMERIC_STD.ALL;
11
12 ENTITY adaptive_cyclic_ADC IS
13 END adaptive_cyclic_ADC;
14
15 ARCHITECTURE adaptive_cyclic_ADC_arch OF adaptive_cyclic_ADC IS
16     -----------
17     -- Types --
18     -----------
19
20     TYPE   T_data_array     IS ARRAY(0 TO  99) OF real;
21     TYPE   T_data_array_2   IS ARRAY(0 TO 200) OF real;
22
23     ---------------
24     -- Constants --
25     ---------------
26
27     -- Clock frequency
28     CONSTANT C_clk_freq         :   integer := 50_000_000;
29     -- Number of iterations
30     CONSTANT C_iteration_number :   integer := 4;
31     -- Sampling frequency [Hz]
32     CONSTANT C_sampling_freq    :   integer := 250_000;
33
34     -- ADC range
35     CONSTANT C_ADC_range        :   real    := 1.0;
36     -- ADC step
37     CONSTANT C_ADC_step         :   real    := (2.0*C_ADC_range)/16.0;
38     -- 4-bit DAC range
39     CONSTANT C_DAC_range        :   real    := 1.0;
40     -- 4-bit DAC step
41     CONSTANT C_DAC_step         :   real    := (2.0*C_DAC_range)/2.0**4;
42     -- 16-bit DAC range
43     CONSTANT C_16_bit_DAC_range :   real    := 1.0;
44     -- DAC step
45     CONSTANT C_16_bit_DAC_step  :   real    := (2.0*C_DAC_range)/2.0**16;
46
47     -------------
48     -- Signals --
49     -------------
50
51     -- System --
52
53     -- Clock
54     SIGNAL r_clk                        :   std_logic := '0';
55     -- Reset
56     SIGNAL r_reset                      :   std_logic := '0';
57
58     -- Analog circuit registers --
59
60     -- Input signal register
61     SIGNAL r_Vt                         :   real    := 0.0;
62     -- Sample and hold output register
```

```
63      SIGNAL r_V_SH                       :   real    := 0.0;
64      -- Substraction output register
65      SIGNAL r_ek                         :   real    := 0.0;
66      -- Amplifier output register
67      SIGNAL r_Ck_ek                      :   real    := 0.0;
68      -- Multiplier - digital part output
69      SIGNAL r_Ck                         :   integer := 1;
70      -- ADC - quantum counter register
71      SIGNAL r_quantum_cnt                :   real    := 0.0;
72      -- SADC output register
73      SIGNAL r_yk_SADC                    :   std_logic_vector(3 DOWNTO 0) :=
            (OTHERS => '0');
74      -- SDAC input register
75      SIGNAL r_V_digital_part             :   unsigned(3 DOWNTO 0)         :=
            (OTHERS => '0');
76      -- SDAC input converted to std_logic_vector
77      SIGNAL r_V_digital_part_vector      :   std_logic_vector(3 DOWNTO 0) :=
            (OTHERS => '0');
78      -- SDAC output register
79      SIGNAL r_V_SDAC                     :   real    := 0.0;
80
81      -- Output data and final estimation --
82
83      -- Final data vector
84      SIGNAL r_output_data                :   std_logic_vector(15 DOWNTO 0) :=
             (OTHERS => '0');
85      -- Final data converted to unsigned
86      SIGNAL r_output_data_unsigned       :   unsigned(15 DOWNTO 0) := (OTHERS
             => '0');
87      -- Final data estimation
88      SIGNAL r_estimation                 :   real := 0.0;
89
90      -- Simulated data --
91
92      -- Counter used to change input data
93      SIGNAL r_cnt    : integer RANGE 0 TO 200    := 0;
94      -- Input data (sine wave)
95      SIGNAL r_sine   : T_data_array      := (0.0, 0.0634239196565645,
            0.126592453573749, 0.189251244360410, 0.251147987181079,
            0.312033445698487, 0.371662455660328, 0.429794912089172,
            0.486196736100469, 0.540640817455598, 0.592907929054640,
            0.642787609686539, 0.690079011482112, 0.734591708657533,
            0.776146464291757, 0.814575952050336, 0.849725429949514,
            0.881453363447582, 0.909631995354518, 0.934147860265107,
            0.954902241444074, 0.971811568323542, 0.984807753012208,
            0.993838464461254, 0.998867339183008, 0.999874127673875,
            0.996854775951942, 0.989821441880933, 0.978802446214779,
            0.963842158559942, 0.945000818714669, 0.922354294104582,
            0.895993774291336, 0.866025403784439, 0.832569854634771,
            0.795761840530832, 0.755749574354258, 0.712694171378863,
            0.666769000516292, 0.618158986220606, 0.567059863862771,
            0.513677391573407, 0.458226521727411, 0.400930535406614,
            0.342020143325669, 0.281732556841430, 0.220310532786541,
            0.158001395973350, 0.0950560433041829, 0.0317279334980681,
            -0.0317279334980679, -0.0950560433041826, -0.158001395973350,
            -0.220310532786541, -0.281732556841429, -0.342020143325669,
            -0.400930535406613, -0.458226521727410, -0.513677391573406,
            -0.567059863862771, -0.618158986220605, -0.666769000516292,
            -0.712694171378863, -0.755749574354258, -0.795761840530832,
            -0.832569854634771, -0.866025403784439, -0.895993774291336,
            -0.922354294104581, -0.945000818714668, -0.963842158559942,
            -0.978802446214779, -0.989821441880933, -0.996854775951942,
            -0.999874127673875, -0.998867339183008, -0.993838464461254,
            -0.984807753012208, -0.971811568323542, -0.954902241444074,
            -0.934147860265107, -0.909631995354519, -0.881453363447582,
            -0.849725429949514, -0.814575952050336, -0.776146464291757,
            -0.734591708657534, -0.690079011482113, -0.642787609686540,
```

```
            -0.592907929054640, -0.540640817455597, -0.486196736100469,
            -0.429794912089172, -0.371662455660328, -0.312033445698487,
            -0.251147987181079, -0.189251244360411, -0.126592453573750,
            -0.0634239196565654, 0.0);
    -- Input data (linear)
    SIGNAL r_linear : T_data_array_2    := (-1.0, -0.99, -0.98, -0.97,
            -0.96, -0.95, -0.94, -0.93, -0.92, -0.91, -0.90, -0.89, -0.88,
            -0.87, -0.86, -0.85, -0.84, -0.83, -0.82, -0.81, -0.80, -0.79,
            -0.78, -0.77, -0.76, -0.75, -0.74, -0.73, -0.72, -0.71, -0.70,
            -0.69, -0.68, -0.67, -0.66, -0.65, -0.64, -0.63, -0.62, -0.61,
            -0.60, -0.59, -0.58, -0.57, -0.56, -0.55, -0.54, -0.53, -0.52,
            -0.51, -0.50, -0.49, -0.48, -0.47, -0.46, -0.45, -0.44, -0.43,
            -0.42, -0.41, -0.40, -0.39, -0.38, -0.37, -0.36, -0.35, -0.34,
            -0.33, -0.32, -0.31, -0.30, -0.29, -0.28, -0.27, -0.26, -0.25,
            -0.24, -0.23, -0.22, -0.21, -0.20, -0.19, -0.18, -0.17, -0.16,
            -0.15, -0.14, -0.13, -0.12, -0.11, -0.10, -0.090, -0.080, -0.070,
            -0.060, -0.050, -0.040, -0.030, -0.020, -0.010, 0.0, 0.010, 0.020,
            0.030, 0.040, 0.050, 0.060, 0.070, 0.080, 0.090, 0.10, 0.11, 0.12,
            0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20, 0.21, 0.22, 0.23,
            0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30, 0.31, 0.32, 0.33, 0.34,
            0.35, 0.36, 0.37, 0.38, 0.39, 0.40, 0.41, 0.42, 0.43, 0.44, 0.45,
            0.46, 0.47, 0.48, 0.49, 0.50, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56,
            0.57, 0.58, 0.59, 0.60, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67,
            0.68, 0.69, 0.70, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78,
            0.79, 0.80, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89,
            0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0);

BEGIN

    ------------------
    -- Assignmnents --
    ------------------

    r_quantum_cnt <= (r_Ck_ek + C_ADC_range) / C_ADC_step;

    -- Conversions --

    r_V_digital_part <= unsigned(r_V_digital_part_vector);

    r_output_data_unsigned <= unsigned(r_output_data);

    ---------------
    -- Instances --
    ---------------

    Digital_part_inst: ENTITY work.digital_part_adaptive
    GENERIC  MAP (
        G_number_of_iterations  => C_iteration_number
    )
    PORT MAP (
        i_clk                   => r_clk,
        i_reset                 => r_reset,
        i_yk_SADC               => r_yk_SADC,
        o_Ck                    => r_Ck,
        o_V_digital_part        => r_V_digital_part_vector,
        o_output_data           => r_output_data
    );

    ------------------------------
    -- Processes and procedures --
    ------------------------------

    P_clk: PROCESS
    BEGIN
        r_clk <= '0';
        WAIT FOR 10 ns;
        r_clk <= '1';
```

```
139         WAIT FOR 10 ns;
140     END PROCESS;
141
142     P_SH: PROCESS
143     BEGIN
144         r_V_SH <= r_Vt;
145         WAIT FOR 1 us;
146         r_V_SH <= r_Ck_ek;
147         WAIT FOR 1 us;
148         r_V_SH <= r_Ck_ek;
149         WAIT FOR 1 us;
150         r_V_SH <= r_Ck_ek;
151         WAIT FOR 1 us;
152     END PROCESS;
153
154     -- Propagation delay set to 100ns
155     P_Substraction: PROCESS (r_V_SH, r_V_SDAC)
156     BEGIN
157         r_ek <= r_V_SH - r_V_SDAC AFTER 100 ns;
158     END PROCESS;
159
160     -- Propagation delay set to 100ns
161     P_Amplifying: PROCESS (r_ek)
162     BEGIN
163         r_Ck_ek <= real(r_Ck) * r_ek AFTER 100 ns;
164     END PROCESS;
165
166     -- Propagation delay set to 100ns
167     P_ADC: PROCESS (r_quantum_cnt)
168     BEGIN
169         IF    (r_quantum_cnt < 1.0) THEN
170             r_yk_SADC <= "0000" AFTER 100 ns;
171         ELSIF (r_quantum_cnt < 2.0) THEN
172             r_yk_SADC <= "0001" AFTER 100 ns;
173         ELSIF (r_quantum_cnt < 3.0) THEN
174             r_yk_SADC <= "0010" AFTER 100 ns;
175         ELSIF (r_quantum_cnt < 4.0) THEN
176             r_yk_SADC <= "0011" AFTER 100 ns;
177         ELSIF (r_quantum_cnt < 5.0) THEN
178             r_yk_SADC <= "0100" AFTER 100 ns;
179         ELSIF (r_quantum_cnt < 6.0) THEN
180             r_yk_SADC <= "0101" AFTER 100 ns;
181         ELSIF (r_quantum_cnt < 7.0) THEN
182             r_yk_SADC <= "0110" AFTER 100 ns;
183         ELSIF (r_quantum_cnt < 8.0) THEN
184             r_yk_SADC <= "0111" AFTER 100 ns;
185         ELSIF (r_quantum_cnt < 9.0) THEN
186             r_yk_SADC <= "1000" AFTER 100 ns;
187         ELSIF (r_quantum_cnt < 10.0) THEN
188             r_yk_SADC <= "1001" AFTER 100 ns;
189         ELSIF (r_quantum_cnt < 11.0) THEN
190             r_yk_SADC <= "1010" AFTER 100 ns;
191         ELSIF (r_quantum_cnt < 12.0) THEN
192             r_yk_SADC <= "1011" AFTER 100 ns;
193         ELSIF (r_quantum_cnt < 13.0) THEN
194             r_yk_SADC <= "1100" AFTER 100 ns;
195         ELSIF (r_quantum_cnt < 14.0) THEN
196             r_yk_SADC <= "1101" AFTER 100 ns;
197         ELSIF (r_quantum_cnt < 15.0) THEN
198             r_yk_SADC <= "1110" AFTER 100 ns;
199         ELSIF (r_quantum_cnt > 15.0) THEN
200             r_yk_SADC <= "1111" AFTER 100 ns;
201         END IF;
202     END PROCESS;
203
204     -- Propagation delay set to 100ns
205     P_DAC: PROCESS (r_V_digital_part)
```

```
206     BEGIN
207         r_V_SDAC <= (real(to_integer(r_V_digital_part))*C_DAC_step -
                C_DAC_range + C_DAC_step/2.0) AFTER 100 ns;
208     END PROCESS;
209
210     -- Final estimation count
211     P_estimation: PROCESS (r_output_data)
212     BEGIN
213         r_estimation <= (real(to_integer(r_output_data_unsigned))*
                C_16_bit_DAC_step - C_16_bit_DAC_range + C_16_bit_DAC_step/2.0)
                ;
214     END PROCESS;
215
216     -- Sine wave simulation
217     P_input_data_simulation: PROCESS
218     BEGIN
219         IF (r_cnt < 99) THEN
220             r_cnt <= r_cnt + 1;
221             r_Vt <= r_sine(r_cnt);
222             WAIT FOR 4 us;
223         ELSE
224             r_cnt <= 0;
225             r_Vt <= r_sine(r_cnt);
226             WAIT FOR 4 us;
227         END IF;
228     END PROCESS;
229
230 END adaptive_cyclic_ADC_arch;
```