**Akademia Górniczo-Hutnicza**
**im. Stanisława Staszica w Krakowie**

Raport z projektu

# Generator funkcyjny wykonany na układzie FPGA wykorzystujący bezpośrednią syntezę cyfrową

z przedmiotu

## Metodyki projektowania i modelowania systemów

Elektronika i telekomunikacja - Systemy wbudowane, rok I studiów magisterskich

Piotr Kowol

15 czerwca 2025

# Spis treści



# Spis rysunków

# 1. Wstęp

Generatory funkcyjne pozwalają generować sygnały elektryczne o zadanych parametrach. Jednym ze sposobów na generowanie sygnału jest bezpośrednia synteza cyfrowa - DDS (*ang.* Direct Digital Synthesis). Jeden ze sposobów realizacji bezpośredniej syntezy cyfrowej wykorzystuje pamięć próbek działającą na zasadzie pamięci LUT (*ang.* Look-Up Table) oraz akumulator fazy. W pamięci znajdują się kolejne wartości cyfrowe sygnału, a akumulator fazy z każdym taktem zegara adresuje odpowiednie komórki pamięci. Następnie wartości próbek są przekazywana do przetwornika cyfrowo-analogowego, który przetwarza sygnał cyfrowy na postać analogową. W takich układach czesto wykorzystuje się nadrzędny procesor, który dostarcza sygnały sterujące. Dzięki temu układy DDS mogą być łatwo konfigurowane poprzez przeprogramowanie pamięci próbek lub jednorazowo programowane przez producenta do odtwarzania konkretnego typu sygnałów np. scalone układy DDS sygnałów sinusoidalnych. W ramach projektu zrealizowano układ bezpośredniej syntezy cyfrowej z programowalną pamięcią próbek, zaprojektowany w języku Verilog.

# 2. Założenia projektowe

Głównym założeniem projektu jest osiągnięcie dużej częstotliwości odtwarzania sygnału - większej niż częstotliwość głównego zegara taktującego logikę sterującą. Ponadto układ powinien posiadać możliwość zaprogramowania pamięci próbek oraz zadania kroku fazowego, który przekłada się na częstotliwość generowanego sygnału. Przesyłanie sygnałów sterujących zostanie zrealizowane za pomocą protokołu UART.

# 3. Analiza problemu

Generacja przebiegów o zadanych parametrach jest możliwa wykorzystując układy analogowe, jednak przestrajanie częstotliwości sygnału w szerokim zakresie wiąże się z projektowaniem wielosekcyjnych przełączanych filtrów. Takie układy zajmują sporo miejsca, ich miniaturyzacja nie jest taka prosta, a elementy przełączające zużywają się. Ponadto analogowe generatory funkcyjne najczęściej ograniczają się do generowania kilku rodzai sygnałów. Rozwiązaniem tych problemów może być generacja sygnału za pomocą syntezy cyfrowej, a następnie jego przetworzenie na postać analogową, poprzez przetwornik cyfrowo-analogowy. Taką funkcjonalność zapewniają układy bezpośredniej syntezy cyfrowej - DDS. Wykorzystują one pamięć do przechowywania wartości próbek, która jest adresowana za pomocą akumulatora fazy. Akumulator fazy z każdym taktem zegara inkrementuje adres pamięci o zadaną wartość, zwaną krokiem fazowym. Odczytywane próbki są przekazywane do przetwornika cyfrowo-analogowego, a następnie poddawane filtracji, filtrem odtwarzającym, którego pasmo przepustowe kończy się przed częstotliwością Nyquist'a. Dodatkowym atutem jest możliwość przeprogramowania pamięci próbek i wygenerowania dowolnego sygnału okresowego.
Układy DDS mają jedną kluczową wadę - nie są w stanie wygenerować sygnału o częstotliwości wyższej niż połowa ich częstotliwości taktowania, co wynika z twierdzenia o próbkowaniu. Okazuje się, że można temu zaradzić. Autorzy [1] zaproponowali wykorzystanie dedykowanych serializerów, dostępnych w układzie FPGA, do zwiększenia częstotliwości odtwarzania powyżej częstotliwości taktowania logiki sterującej. Dodatkowo zastosowanie trybu DDR pozwala jeszcze bardziej zwiększyć częstotliwość odtwarzania.

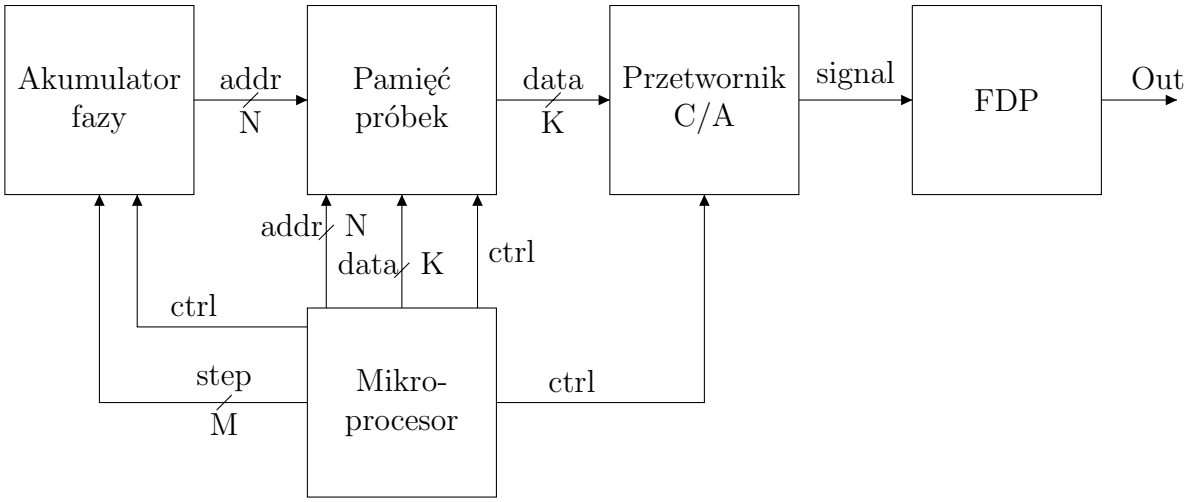
# 4. Realizacja

W tym rozdziale zostanie opisana struktura logiczna zaprojektowanego układu oraz algorytmiczny opis jego działania. Zostaną również opisane konfiguracje poszczególnych bloków. Projekt logiki programowalnej został zrealizowany w środowisku Vivado Design Suite 2018.3, a porogram na mikroprocesor został napisany w C w środowisku Vivado SDK. Projektowanie logiki przeprowadzono w kilku etapach:

- Stworzenie opisu behawioralnego lub strukturalnego poszczególnych bloków,
- Przeprowadzenie testów komponentów na poziomie symulacji behawioralnej,
- Weryfikacja poprawności działania po przeprowadzeniu implementacji - przeprowadzenie symulacji behawioralnych i czasowych w celu sprawdzenia działania i relacji czasowych między sygnałami,
- Naniesienie potrzebnych poprawek w projektowanych układach oraz powtórna weryfikacja,
- Połączenie zbudowanych bloków funkcjonalnych,
- Weryfikacja działania całego układu po implementacji na poziomie symulacji behawioralnej oraz czasowej

## 4.1. Budowa układu

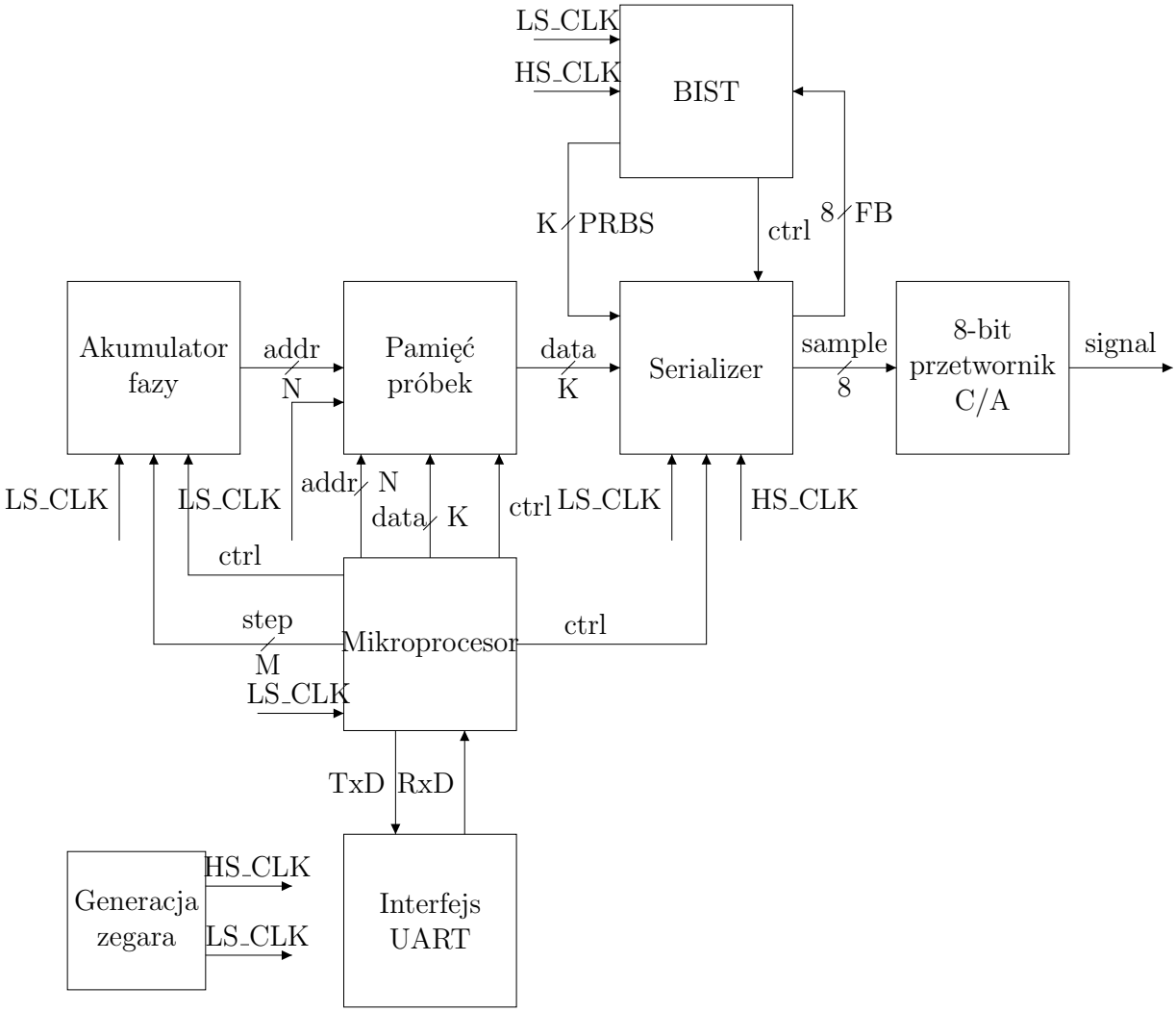
Układy DDS składają się z kilku kluczowych bloków: akumulatora fazy, pamięci próbek oraz przetwornika cyfrowo-analogowego. Często posiadają nadrzędny układ sterujący w postaci mikroprocesora lub mikrokontrolera. Sygnał wyjściowy z przetwornika należy poddać filtracji dolnoprzepustowym filtrem odtwarzającym w celu usunięcia składowych o częstotliwości wyższej od połowy częstotliwości zegara taktującego układ. Schemat blokowy podstawowego układu bezpośredniej syntezy cyfrowej przedstawiono na rysunku 4.1.




Rysunek 4.1: Schemat blokowy podstawowego układu DDS.

Układy FPGA w większości nie są dostosowane do pracy przy bardzo szybkich zegarach taktujących. Z tego powodu podstawowy schemat DDS'a wzbogacono o serializer oraz układ generacji zegara. Zastosowanie serializera pozwala uzyskać wyższą częstotliwość odtwarzania sygnału, niż wynikająca z taktowania logiki sterującej. Serializer wykorzystuje dedykowane IP

Core do serializacji danych i pozwala przekazywać próbki sygnału znacznie szybciej niż pozwalała by logika programowalna. Ponad to wykorzystuje tryb pracy DDR (*ang.* Double Data Rate), co umożliwia przekazywanie próbek na każdym zboczu szybkiego zegara taktującego serializer. Serializer wymaga dwóch sygnałów zegarowych - jeden wolniejszy, służący do taktowania obwodów wejściowych, oraz drugi szybszy, taktujący szybkie układy wyjściowe, służące do serializacji danych. Synchroniczność działania układów zapewniono poprzez generację obydwu sygnałów zegarowych za pomocą układu pętli fazowej, z tego samego referencyjnego zegara, doprowadzonego do układu FPGA za pomocą oscylatora kwarcowego. Sterowanie układem oraz ładowanie pamięci próbek z komputera odbywa się za pomocą procesora ARM dostępnego w układzie FPGA. Procesor zapewnia komunikację z komputerem poprzez protokół UART, oraz komunikację z układem DDS za pośrednictwem magistrali AXI - jest to konfigurowalna magistrala wykorzystywana w układach FPGA do komunikacji między zintegrowanymi mikroprocesorami, a zaprojektowaną logiką. Układ został zbudowany w oparciu o platformę ewaluacyjną ZedBoard od firmy AVNET. Płytka wykorzystuje układ SoC AMD Xilinx Zynq-7000. Układ nie posiada zintegrowanego przetwornika cyfrowo-analogowego, dlatego wykorzystano zewnętrzną płytkę rozszerzeń z 8-bitową drabinką rezystorową R-2R. W ramach projektu nie przewiduje się budowy filtru dolnoprzepustowego. Zmodyfikowany schemat blokowy układu bezpośredniej syntezy został przedstawiony na rysunku 4.2.



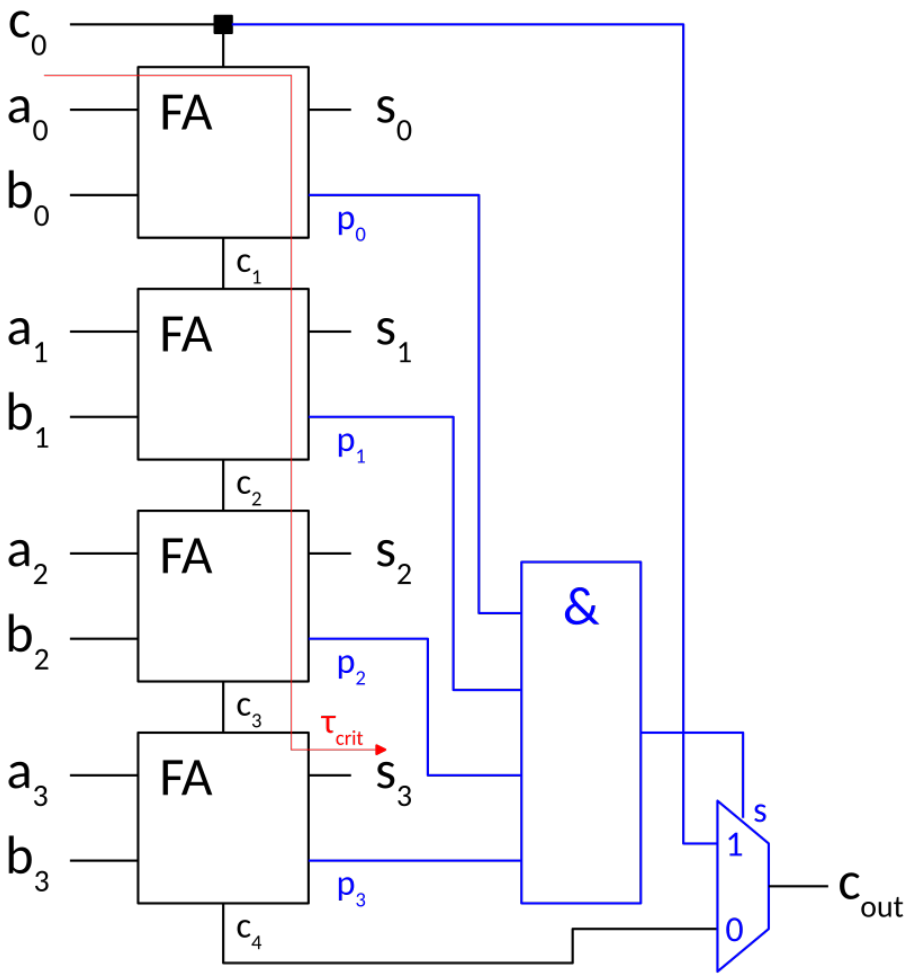

Rysunek 4.2: Schemat blokowy układu DDS.

Dodatkowo układ został wyposażony w logikę samo-testującą (*ang.* Build-In Self-Test). Układ BIST generuje binarną sekwencję pseudo-losową (*ang.*Pseudo-Random Binary Sequence)

i przekazuje ją do wejść serializerów. Umożliwia to sprawdzenie poprawności działania szybkiej serializacji danych, poprzez pobranie danych wyjściowych, ich deserializację za pomocą dedykowanych deserializerów oraz porównanie z danymi przekazanymi do wejść serializerów. Na podstawie porównania danych wysyłanych i odebranych układ zlicza ilość występujących błędów i przekazuje raport wynikowy do mikroporcesora, który wysyła dane do komputera.

### 4.1.1. Akumulator fazy

Układ akumulatora fazy składa się z rejestru oraz sumatora. Rejestr przechowuje aktualny indeks próbki, a sumator inkrementuje aktualną wartość o zadany krok fazowy. Cała operacja zajmuje mniej niż jeden takt LS_CLK. Rejestr został zrealizowany na przerzutnikach D, a sumator zbudowano w oparciu o 6 4-bitowych sumatorów w architekturze sumatorów z przeskokiem przeniesienia (*ang.* Carry Skip Adder). Schemat 4-bitowego sumatora został przedstawiony na rysunku 4.3. Sygnały $S_i$ są poszczególnymi bitami sumy, sygnały pomocnicze $p_i = a_i \oplus b_i$ służą




Rysunek 4.3: Schemat blokowy 4-bit CSA [2].

do wyboru przeniesienia wyjściowego. Sygnały przeniesień wewnętrznych są propagowane lub generowane według wzoru:

$$c_{i+1} = \begin{cases} c_i & \text{jeśli } p_i = 1 \text{ - propagacja} \\ a_i & \text{jeśli } p_i = 0 \text{ - generacja} \end{cases}$$

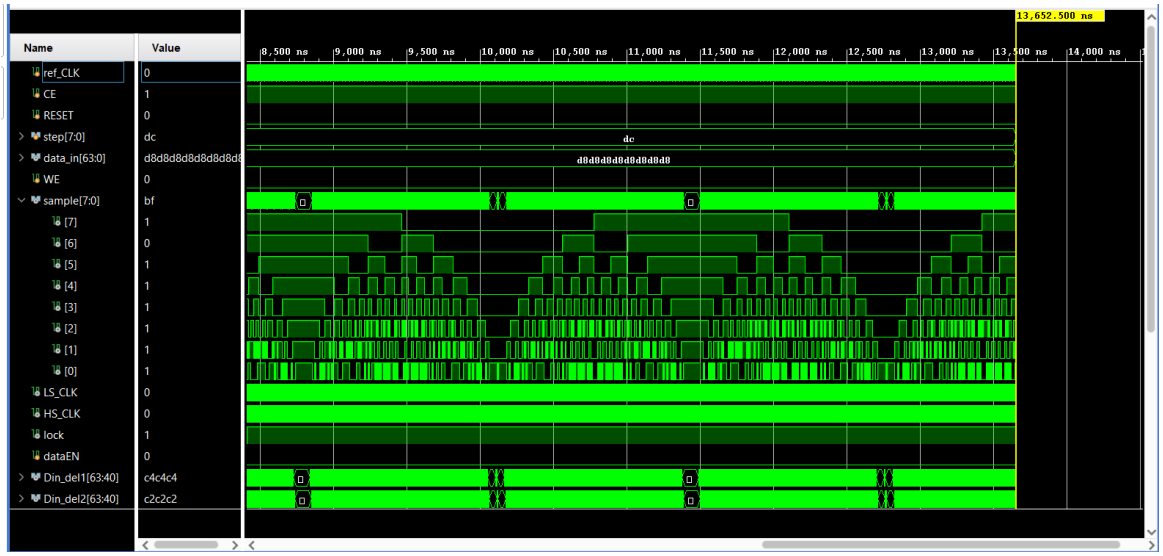
Zastosowanie CSA pozwala obliczyć następny indeks próbki nieco szybciej niż za pomocą standardowych sumatorów.

### 4.1.2. Pamięć próbek

Pamięć zrealizowano w postaci dedykowanych programowalnych układów BRAM (*ang.* Block Random-Access Memory). W projekcie wykorzystano pamięć dwu-portową o 10-bitowej magistrali adresowej oraz 64-bitowych komórkach, co odpowiada 8192 komórkom po 8 próbkom w każdej. Adresy komórek pamięci odpowiadają indeksom próbek odtwarzanego sygnału. Jeden port pamięci działa w trybie zapisu - służy do ładowania wartości próbek z komputera, a drugi port jest tylko do odczytu - jest adresowany przez akumulator fazy, a odczytane dane są przekazywane do serializerów. Podczas zapisu danych adres kolejnej komórki jest automatycznie inkrementowany na koniec pojedynczej operacji zapisu. Dzięki temu mechanizmowi nie jest potrzebne przesyłanie adresu komórki przez mikroprocesor.

### 4.1.3. Serializer

Do serializacji danych wykorzystano 8 dedykowanych serializerów OSERDESE2 [3], [4], [5] . Pozwalają one osiągnąć znacznie wyższą częstotliwość odtwarzania niż częstotliwość taktowania logiki sterującej. Podczas wystąpienia narastającego zbocza zegara taktującego logikę sterującą - LS_CLK (Low speed CLK) dane z pamięci są zatrzaskiwane w obwodach wejściowych układów OSERDESE2. Z kolei przy każdym zboczu szybkiego zegara - HS_CLK (High speed CLK) kolejne bity są przekazywane na wyjście serializerów. Układy OSERDESE2 zostały skonfigurowane do pracy w trybie DDR (*ang.* Double Data Rate) przy szerokości danych wejściowych równej 8 bitów. Pozwala to zmniejszyć częstotliwość zegara HS_CLK do połowy częstotliwości odtwarzania. W układzie nie jest wykorzystywana praca w trybie trój-stanowym - ogranicza ona szerokość danych do 4 bitów, co wymagałoby stosowania serializerów w trybie master-slave - potrzeba by 2 razy więcej układów OSERDESE2. Przykładowy przebieg serializacji przedstawiono na rysunku 4.4. Sygnał wyjściowy z serializerów to sample[7:0]. W zaprezentowanym przykładzie układ jest taktowany z dedykowanej pętli fazowej, a wartości próbek są odczytywane z wcześniej zaprogramowanej pamięci BRAM.

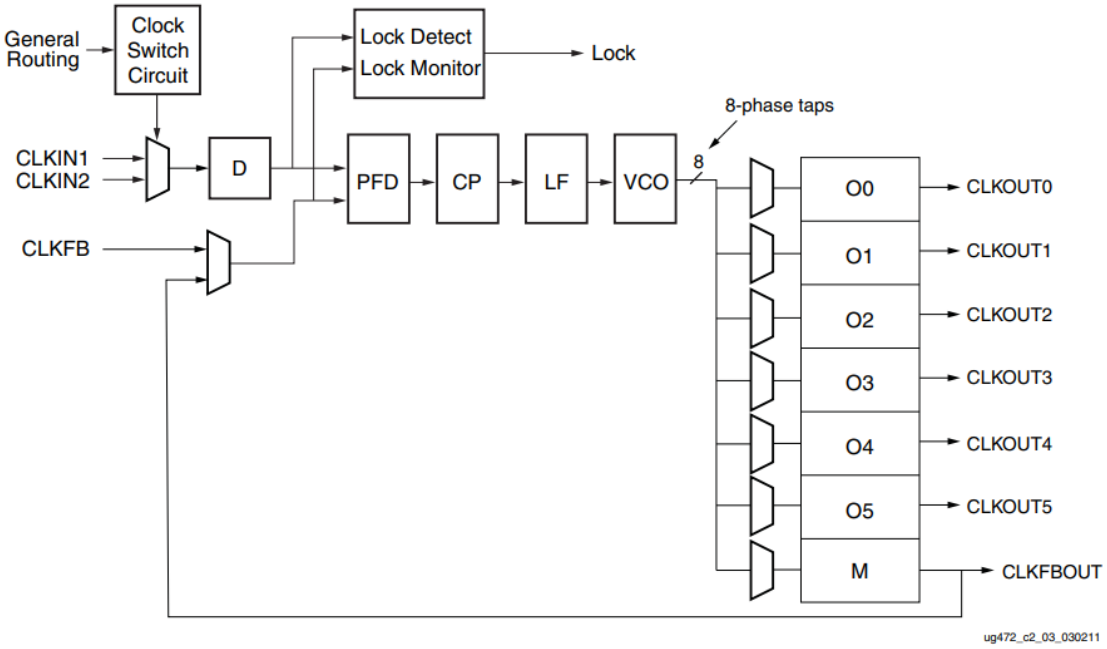



Rysunek 4.4: Serializacja sygnału sinusoidalnego za pomocą serializera składającego się z 8 układów OSERDESE2 - wynik symulacji czasowej przeprowadzonej po implementacji układu.

### 4.1.4. Generacja zegara

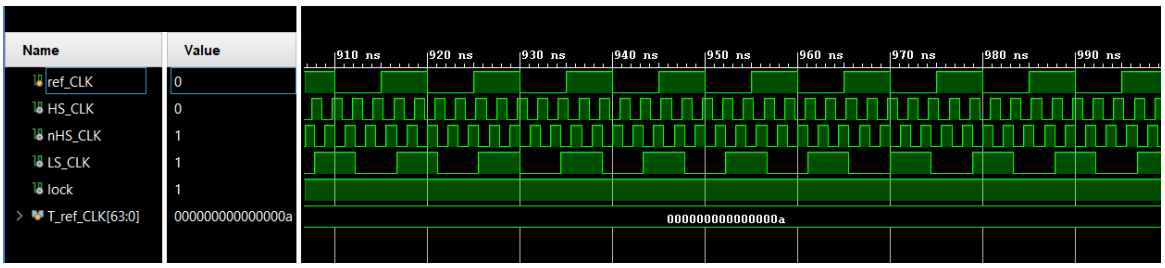
Sygnały zegarowe zostały wygenerowane za pomocą dedykowanego układu PLLE2_BASE. Układ składa się z detektora fazy i częstotliwości - PFD, pompy ładunkowej - CP, filtru dolno-

przepustowego - LF, oraz generatora przestrajanego napięciem - VCO oraz głównego dzielnika częstotliwości - M. Ponad to układ posiada aż 6 wyjść zegarowych, każde z nich jest wyposażone w indywidualny dzielnik częstotliwości. Dodatkowo możliwy jest wybór odpowiedniej fazy VCO dla każdego wyjścia niezależnie. Schemat blokowy układu PLLE2_BASE przedstawiono na rysunku 4.5. Wewnętrzny generator przestarajany napięciem może pracować w za-




Rysunek 4.5: Schemat blokowy układu PLLE2_BASE [6].

kresie $800 \div 2133\ MHz$. Górna częstotliwość graniczna może się różnić zależnie od możliwości układu FPGA - parametr Grade Speed [5]. Częstotliwość referencyjnego oscylatora kwarcowego umieszczonego na płytce ewaluacyjnej wynosi $f_{REF} = 100\ MHz$ [7]. Początkowo zakładano że częstotliwość odtwarzanie sygnału wyniesie 1 $GS/s$, co oznacza, że układy OSERDESE2 pracujący w trybie DDR, należy taktować zegarem 500 $MHz$, z kolei zegar taktujący logikę sterującą miałby częstotliwość 4 razy mniejszą, tj. 125 $MHz$. Wykorzystując te informacje, skonfigurowano dzielniki częstotliwości pętli fazowej. Dzielnik w sprzężeniu zwrotnym (*ang.* Feedback Divider) ustawiono na 10, co daje $f_{VCO} = 1000\ MHz$, a dzielniki wyjściowe ustawiono na 2 i 8 odpowiednio dla HS_CLK i LS_CLK. Podczas przeprowadzania symulacji wszystko przebiegało prawidłowo, jednak po wygenerowaniu raportu opóźnień, okazało się, że globalne bufory sygnałów zegarowych - BUFG - nie są w stanie prawidłowo przepropagować sygnału o częstotliwości 500 $MHz$. Według dokumentacji, ich częstotliwość graniczna wynosi $f_{gBUFG} = 464\ MHz$ przy speed grade wynoszącym $-1$ [5]. W związku z tym, zmniejszono częstotliwość HS_CLK do 450 $MHz$, co daje częstotliwość odtwarzania 900 $MS/s$. Częstotliwość zegara LS_CLK również została proporcjonalnie obniżona i wynosi 112.5 $MHz$. Zmiana częstotliwości wiązała się z modyfikacją konfiguracji dzielników częstotliwości w pętli fazowej - zmniejszono wartość feedback divider'a do 9, co przekłada się na pracę VCO na częstotliwości $f_{VCO} = 900\ MHz$. Aby uniknąć niezdefiniowanego zachowania układów podczas uruchamiania pętli fazowej, skorzystano z wewnętrznego układu kontrolującego zatrzaśnięcie pętli, a jego sygnał wyjściowy użyto do generowania sygnału CE (*ang.* Clock Enable) w sposób kombinacyjny. Przykładowe przebiegi wyjściowe pętli fazowej zostały przedstawione na rysunku 4.6. Istotnym aspektem projektowania ścieżki dystrybucji zegara jest zastosowanie odpowiednich buforów sygnałów zegarowych - dla wszystkich ścieżek powinny zostać użyte takie same bufory w celu uniknięcia wyścigu zboczy.




Rysunek 4.6: Przebiegi sygnałów zegarowych generowanych przez pętlę fazową - wyniki symulacji czasowej przeprowadzonej po implementacji układu.

### 4.1.5. Mikroprocesor i protokół UART

Mikroprocesor obsługuje kilka poleceń, są to:

- LOAD - przejscie do trybu ładowania danych
- STEP - załadowanie kroku fazowego
- GENERATE - przejscie do trybu generowania sygnału
- BIST - przejscie do trybu testowania
- STOP - zatrzymanie pracy układu
- RESET - software'owy reset DDS'a

Wywołanie polecenia powoduje uruchomienie odpowiedniej funkcji oraz przesłanie magistralą AXI odpowiednich sygnałów sterujących to rejestru sterującego. Rozpoznawanie poleceń odbywa się przez ich liczbowe zakodowanie - wystarczy odczytać dwa kolejne bajty. Ładowanie wartości próbek polega na przesyłaniu kolejnych 8-bitowych wartości - próbki mieszczą się w przedziale $0 \div 255$ i przyjmują wartości całkowite. Załadowanie kroku fazowego przebiega analogicznie, z tą różnicą, że jego rozdzielczość jest większa i wynosi $1 \div 4095$. Przesyłane wartości zawsze muszą zawierać odpowiednio 3 lub 4 znaki - odbiór przebiega podobnie jak w przypadku poleceń sterujących.

### 4.1.6. BIST

Układ testujący wykorzystuje 8 generatorów PRBS8 [8] oraz 8 dedykowanych deserializerów ISERDESE2. Generatory PRBS8 pracują równolegle z przesunięciem kroku początkowego o 32. Każdy serializer właściwie działa niezależnie od innych, dlatego nie jest wymagane generowanie sygnału PRBS64, co znacząco wydłużyłoby testowanie - w takim przypadku należałoby ograniczyć liczbę generowanych kodów podczas testu - przy zegarze taktującym 112.5 $MHz$ sprawdzenie wszystkich możliwych kombinacji zajełoby $\approx$ 5200 lat. Znajomość budowy układu, pozwala zredukować liczbę wektorów testowych do $2^8$, co przekłada się na czas testu wynoszący zaledwie $\approx$ 2.3 $\mu s$. Proces serializacji - deserializacji w obrębie pojedynczego kanału został przedstawiony na rysunku 4.7. Teoretycznie układy OSERDESE2 oraz ISERDESE2 powinny posiadać własne zegary. Podczas projektowania układu okazało się, że nie jest to możliwe - narzędzie do implementacji umieściło układy w obrębie jednego slice'u, co zapewnia niewielkie opóźnienia między serializerem i deserializerem, jednak zabrakło dedykowanych buforów zegara do taktowania układów. W prawdzie platforma Zynq 7000 posiada 32 takie bufory, jednak są one rozdzielone pomiędzy poszczególne slice'y. Rozwiązaniem problemu okazało się zastosowanie tych samych zegarów do taktowania układów - wystarczyło wygenerować zegar HS_CLK różnicowo. Niestety utrudnieniem okazały się czasy propagacji układów I/OSERDESE2 - w

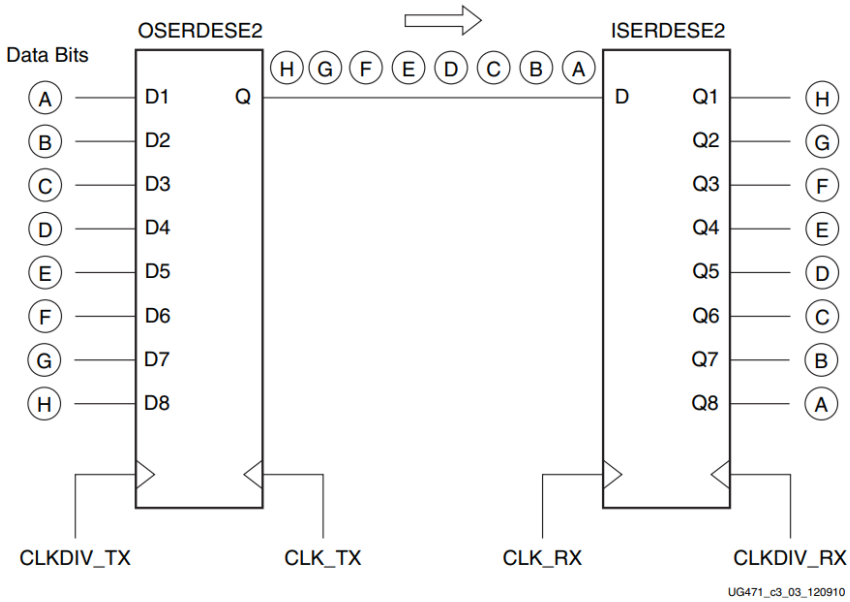


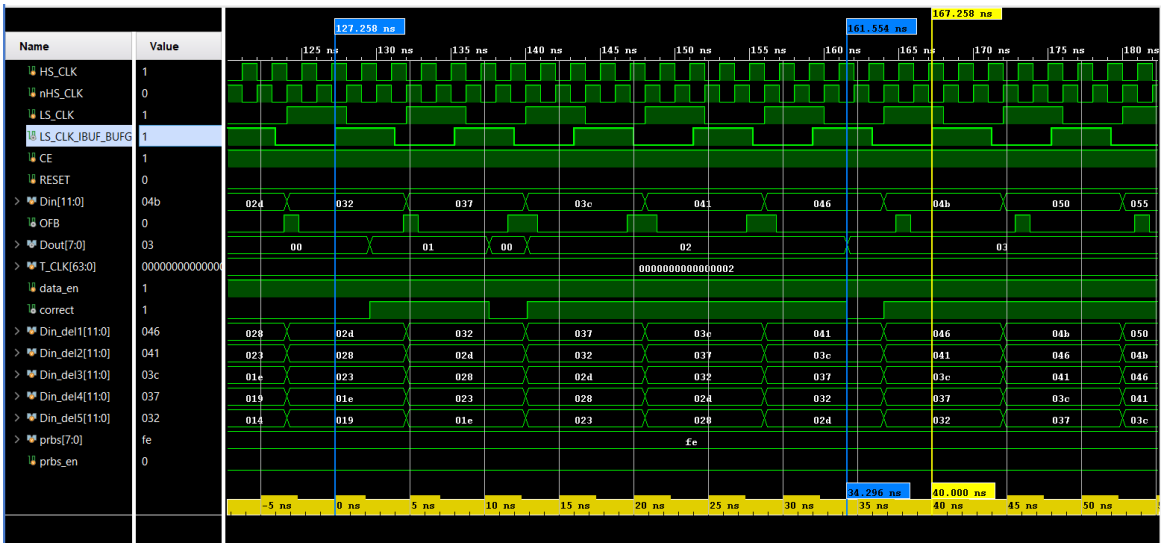

*Figure 3-3:* **Bit Ordering on Q1–Q8 Outputs of ISERDESE2 Ports**

Rysunek 4.7: Kolejność przepływu bitów przy serializacji i deserializacji danych za pomocą układów OSERDESE2 i ISERDESE2 [3].

celu prawidłowego porównania danych serializownych z deserializowanymi należy odczytać 7 bitów z aktualnej sekwencji wyjściowej i 1 najmłodszy bit z poprzedniej oraz sekwencję wzorcową należy opóźnić o 4 takty zegara. Wynik porównania jest realizowany przez komparator kombinacyjny, ale odczyt prawidłowej wartości jest możliwy przy piątym zboczu narastającym LS_CLK. Przykładowe przebiegi dla symulacji jednego kanału BIST zostały przedstawione na rysunku 4.8. Dane wejściowe to 8 najstarszych bitów sygnału Din[11:0], sygnał OFB, to sprzężenie zwrotne między srerilaizerem, a deserializerem, dane wyjściowe układy ISERDESE2 to sygnał Dout[7:0]. Sygnały Din_deli stanowią dane wejściowe opóźnione o odpowiednią liczbę taktów zegara. Są one synchroniczne z sygnałem zegarowym LS_CLK ponieważ są generowane w TestBenchu, a nie wewnątrz testowanego układu. Pozostałe sygnały są synchroniczne z zegarem buforowanym zegarem LS_CLK. Przykład zrealizowano dla zegara $f_{LS\_CLK} = 125\ MHz$ ze względu na wymogi środowiska - nie implementowano pętli fazowej, a podstawową jednostką czasu symulacji jest 1 $ns$ - okres zegara jest całkowitą wielokrotnością podstawowej jednostki czasu. W przypadku 8 kanałów BIST opóźnienie również wynosi 5 taktów LS_CLK. Poprawność serializacji i deserializacji jest sygnalizowana sygnałem correct.

## 4.2. Przygotowanie próbek

Próbki sygnału są przygotowywane za pomocą algorytmu napisanego w języku Python. Skrypt pozwala wygenerować sygnał sinusoidalny, prostokątny, trójkątny oraz piło-kształtny. Użytkownik podaje amplitudę, zdefiniowaną jako procent pełnego zakresu, który wynosi $V_{FS} = 3.3\ V$, offset zdefiniowany jako procentowa wartość pełnego zakresu oraz liczbę okresów, które mają się zmieścić w pamięci. Wprowadzenie większej amplitudy oraz offsetu jest możliwe, ale będzie się wiązało ze zniekształceniem sygnału w postaci obcięcia wartości wykraczających po za zakres. Użytkownik zostanie o tym poinformowany poprzez wysłanie ostrzeżenia. Algorytm ma zdefiniowaną pojemność pamięci oraz rozdzielczość przetwornika. Wartości próbek są obliczane na podstawie zadanych parametrów - sygnał pseudo-analogowy, a następnie są one normalizowane do pełnego zakresu bitowego i zaokrąglane do wartości całkowitych. Skrypt dodatkowo wyświetla obliczony przebieg w celu weryfikacji i akceptacji, oraz oblicza zalecany



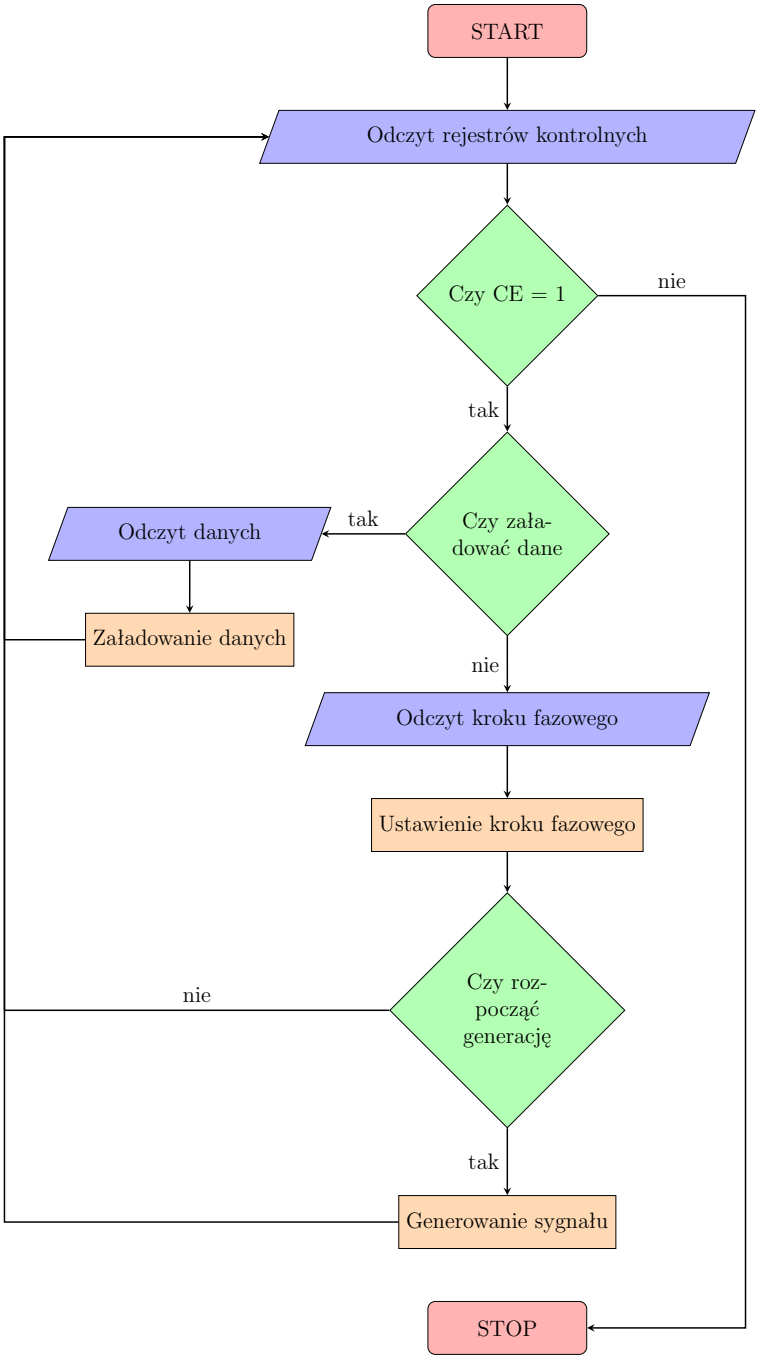

Rysunek 4.8: Przykładowy przebieg sygnałów dla jednego kanału BIST dla zegara taktującego 125 $MHz$ - wynik symulacji czasowej przeprowadzonej po implementacji układu.

zakres kroku fazowego dla proponowanego zestawu próbek. Ma to na celu zredukowanie błędów kwantyzacji oraz redukcję artefaktów występujących w widmie generowanego sygnału - spurs̀ów III rzędu. Po akceptacji dane są przesyłane przez port szeregowy do układu FPGA. Na koniec uzytkownik wprowadza krok fazowy. Dodatkowym atutem jest możliwość wyświetlenia dostępnych poleceń wraz z krótkim opisem poprzez wpisanie *help* lub znaku: *?*.

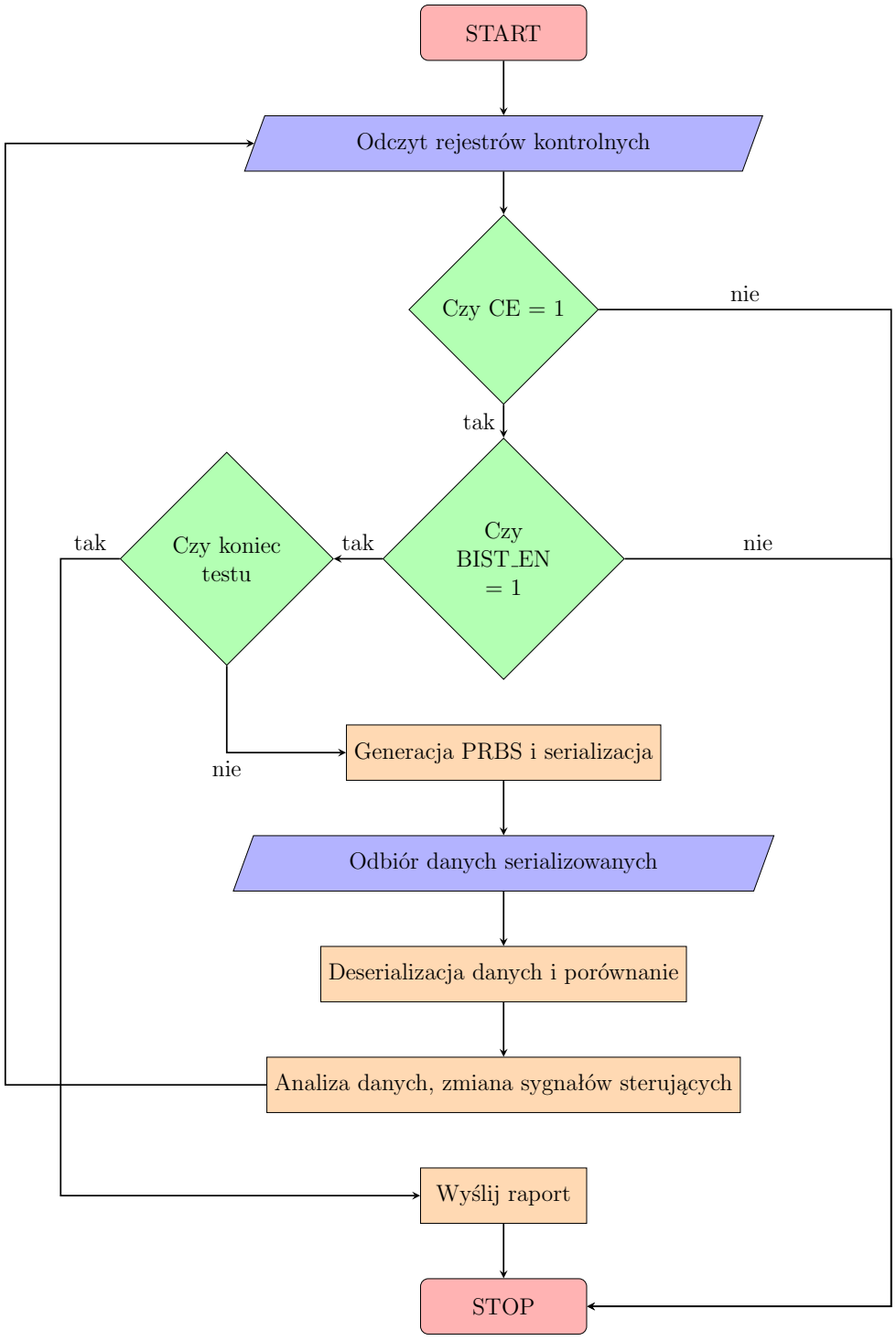
## 4.3. Działanie układu

Główny algorytm działania układu został przedstawiony na rysunku 4.9. Sygnały sterujące są przekazywane do rejestrów kontrolnych przez mikroprocesor za pośrednictwem magistrali AXI. W przypadku ładowania próbek do pamięci mikroporcesor ustawia bit WE w rejestrze kontrolnym DDS'a, z kolei układ wewnętrzny generuje impuls pozwalający na zapis otrzymanego pakietu danych trwający dokładnie jeden takt zegara LS_CLK. Pakiet danych liczy 8 próbek, czyli 64 bity.

Wyzwolenie samo-testowania układu powoduje zatrzymania wykonywania głównego algorytmu i przejście do algorytmu BIST. Zostaje on wyzwolony po odczycie danych z rejestrów sterujących, jeśli wystąpił sygnał BIST enable (BIST_EN). Działanie algorytmu BIST zostało przedstawione na rysunku 4.10.



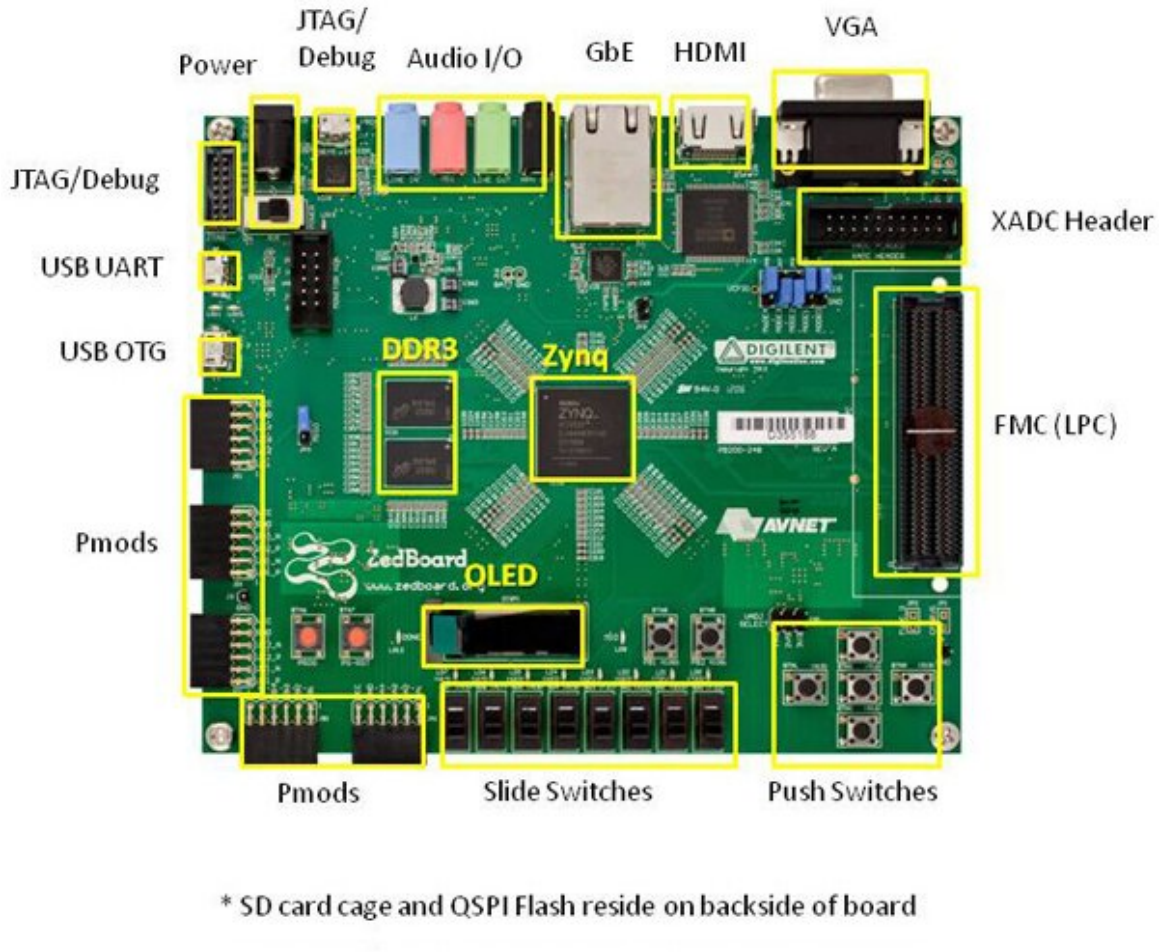

Rysunek 4.9: Uproszczony algorytm pracy układu.




Rysunek 4.10: Algorytm działania BIST.

# 5. Użytkowanie

Platformę ewaluacyjną ZedBoard należy podłączyć do zasilania za pomocą dołączonego zasilacza oraz podłączyć do komputera PC za pomocą kabla USB. Do komunikacji z komputerem jest wykorzystywane złącze microUSB służące jednocześnie do programowania układu FPGA - znajduje się ono po prawej od złącza zasilania i jest oznaczone *JTAG/Debug*. Widok płytki został przedstawiony na rysunku 5.1 Po uruchomieniu platformy można uruchomić skrypt służący




Rysunek 5.1: Widok z góry płytki ewaluacyjnej ZedBoard z zaznaczonymi portami [7].

do konfiguracji układu bezpośredniej syntezy cyfrowej. Skrypt posiada kilka poleceń:

- *help* lub *?* - wyświetla informacje o dostępnych poleceniach,
- *connect* - umożliwia połączenie z układem DDS za pomocą portu szeregowego COM,
- *prepare* - służy do ustawienie parametrów generowanego sygnału,
- *step* - ładuje zadany krok fazowy do układu FPGA,
- *load* - ładuje wygenerowany sygnał do pamięci próbek,
- *generate* - uruchamia generacje sygnału,
- *reset* - przywraca stan początkowy układu DDS,
- *stop* - zatrzymuje generacje sygnału,

- *bist* - uruchamia self-test układu DDS, a następnie odbiera raport testu,
- *quit* - zamyka połaczenie z urządzeniem , a następnie kończy działanie programu

Polecenia pozwalają na proste w obsłudze zarządzanie układem bezpośredniej syntezy cyfrowej.

## 5.1. Opis poleceń

Poniżej zaprezentowano krótki opis poleceń oraz przyjmowane argumenty. W przypadku wprowadzenia polecenia, którego nie ma na liście poleceń, żadne nie zostanie wykonane, a użytkownik zostanie poinformowany o wprowadzeniu błędnej informacji.

### 5.1.1. Poecenie *help*

Wyświetla listę wszystkich poleceń wraz z przyjmowanymi argumentami. Wynik działania zaprezentowano na rysunku 5.2.

```
>?
help:                                   prints all possible commands with short description
?:                                      same as help
bist:                                   starts build-in self test and reads test report
connect <serial_port>:                  connect to device connected to specified COM port
generate:                               starts signal generation
load:                                   loads samples data to DDS memory
reset:                                  sets all DDS counters and phase accumulator to 0, resets BIST and serializers state
step <value>:                           loads <value> to DDS as phase step, only positive integers values are allowed, default 1
stop:                                   disables signal generation
prepare <signal> <amplitude> <offset> <N>:      starts data generation process
                                                <signal> - signal types: sine, rectangle, triangle, sawtooth, default sine
                                                <amplitude> - signal amplitude defined as percent of full scale, only positive values, floats are allowed, default 50%
                                                <offset> - DC offset defined as percent of full scale, floats are allowed, default 50%
                                                <N> - number of periods of signal written in memory, only positive integers values are allowed, default 1
quit:                                   disconnects target device, closes serial port session end exits code
>
```

Rysunek 5.2: Wynik działania polecenia *help*.

### 5.1.2. Polecenie *connect*

Polecenie przyjmuje jeden argument - port szeregowy. Przykładowe wywołanie *connect com1* spowoduje próbę połaczenia z urządzeniem podłaczonym do portu COM1. W przypadku wystąpienia błędu z połaczeniem, użytkownik zostanie poinformowany oraz otrzyma listę dostępnych urządzeń. Jeśli urządzenie nie zostanie znalezione, można wywołać najwyższy numer portu, co spowoduje ponowne wyszukanie urządzeń. W sytuacji, gdy użytkownik wywoła polecenia *connect*, ale nie zdefiniuje portu, zostanie użyty port domyślny - COM1.

### 5.1.3. Polecenie *prepare*

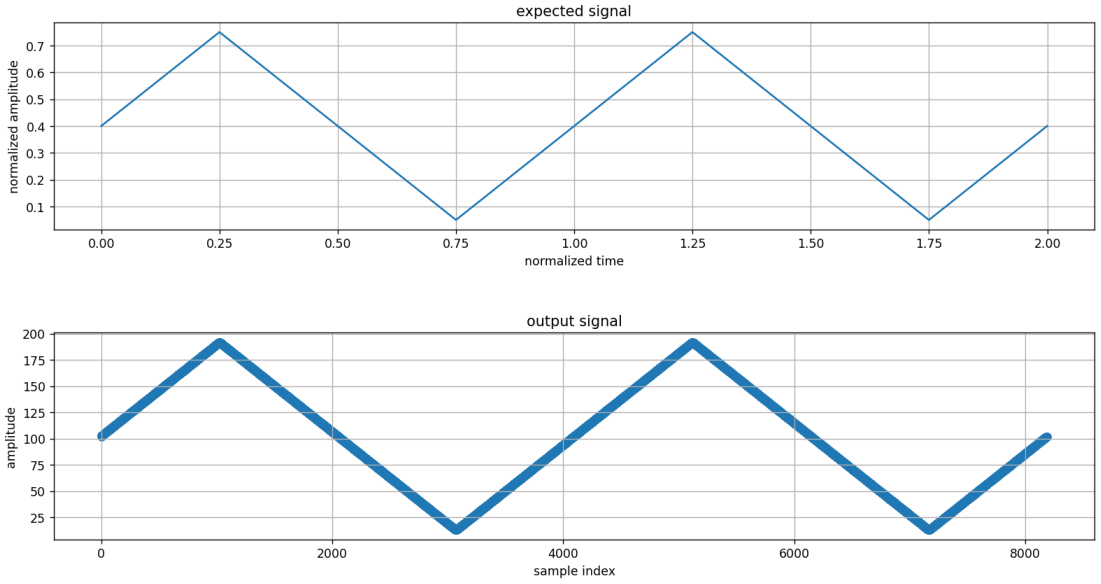
Polecenie przyjmuje 4 argumenty: kształt sygnału, amplitudę wyrażoną w % pełnego zakresu, offset wyrażony w % pełnego zakresu oraz liczbę okresów, które zostaną załadowane do pamięci. Skrypt pozwala wygenerować podstawowe przebiegi:

- sygnał sinusoidalny - *sine*,
- sygnał prostokątny - *rectangle*,
- sygnał trójkątny - *triangle*,
- sygnał piło-kształtny - *sawtooth*

Inne kształty przebiegów zostaną zaimplementowane w kolejnych wersjach oprogramowania. Dzięki oprogramowaniu, można szybko przygotować próbki, bez obaw o ich rozmieszczenie w pamięci próbek, oraz ich reprezentacje binarną. W przypadku podania błędnych parametrów, zostaną użyte wartości domyślne. Domyślnym sygnałem jest 1 okres sinusa o amplitudzie 50 % pełnego zakresu i offsecie wynoszącym 50 %. Jeśli tylko część argumentów jest błędna, pozostałe zostaną wprowadzone prawidłowo. Użytkownik zostanie poinformowany o błędach odpowiednimi komunikatami. Możliwe jest też podanie amplitudy lub offsetu wykraczającego po za zakres. W takiej sytuacji podane wartości zostaną wprowadzone, a użytkownik zostanie poinformowany odpowiednim ostrzeżeniem - nie jest to traktowane jako błąd, gdyż zakłada się, że użytkownik może celowo wygenerować sygnał zniekształcony. Po wygenerowaniu wartości próbek, zostanie wyświetlony wykres z idealnym sygnałem wprowadzonym przez użytkownika oraz sygnał poddany kwantyzacji i ograniczeniu do pełnego zakresu - szczyty sygnału zostaną odpowiednio obcięte do skrajnych wartości, tj. 0 lub 255. Liczba okresów sygnału powinna być liczbą całkowitą z zakresu $1 \div 1024$, co odpowiada od 8 do 8192 próbek na okres. Wprowadzenie większej liczby okresów będzie skutkowało ostrzeżeniem i zastosowaniem wartości domyślnej. Przykład prawidłowego zastosowania polecenia został przedstawiony na rysunku 5.3, a przykład wprowadzenia błędnych danych na rysunku 5.5. Sygnały wygenerowane na podstawie podanych wartości zostały przedstawione na rysunkach 5.4 oraz 5.6.

```
>prepare triangle 35 40 2
SignalType: triangle    Amplitude: 35.0 Offset: 40.0    Periods: 2
Total errors: 0
Total warnings: 0
recommended min phase step is 1
recommended max phase step is 16384
>
```

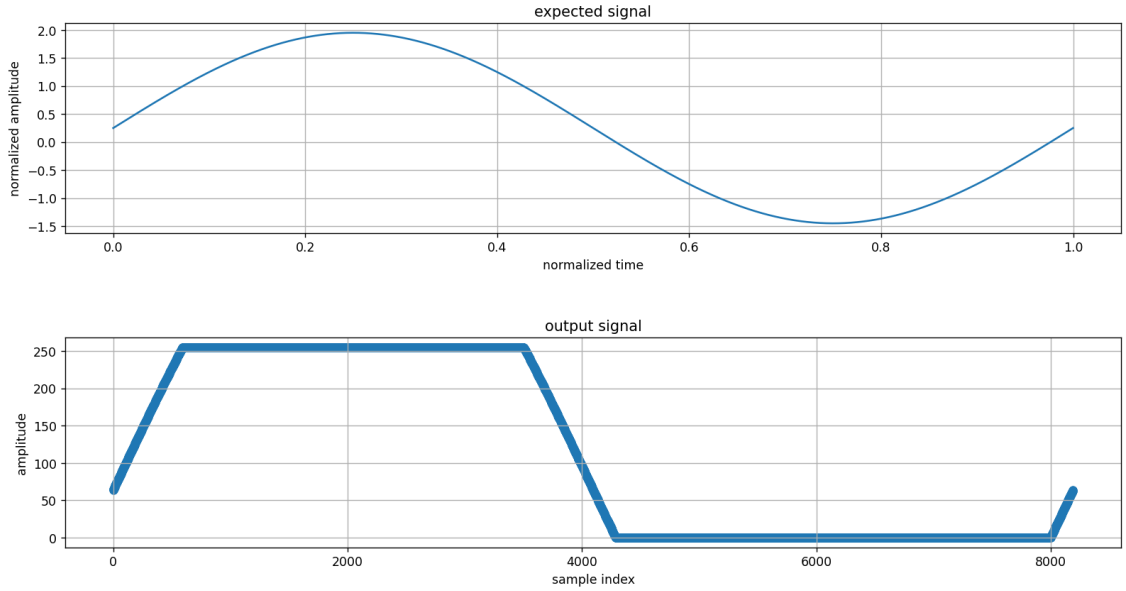
Rysunek 5.3: Przykład prawidłowego użycia polecenia *prepare*.




Rysunek 5.4: Przykład prawidłowo wygenerowanego sygnału.

```
>prepare tri 170 25
Signal type error: wrong type, default type will be used
Signal warning: max value exceeds full scale, signal will be limited
Signal warning: min value exceeds full scale, signal will be limited
Number of periods error: expected positive integer value, default value will be used
SignalType: sine        Amplitude: 170.0        Offset: 25.0    Periods: 1
Total errors: 2
Total warnings: 2
recommended min phase step is 16384
recommended max phase step is 65536
>
```

Rysunek 5.5: Przykład błędnego użycia polecenia *prepare* - błędna nazwa sygnału, oraz nie podana liczba okresów.




Rysunek 5.6: Przykład nieprawidłowo wygenerowanego sygnału - za duża amplituda i offset spowodowały niesymetryczne obcięcie szczytów sinusa (zakładając, że użytkownik chciał otrzymać sygnał bez zniekształceń).

### 5.1.4. Polecenie *step*

Polecenie przyjmuje jeden argument - wartość kroku fazowego w postaci dodatniej liczby całkowitej z zakresu $1 \div 65535$ - wartości powyżej 16383 pozwalają pomijać komórki pamięci - jest to zalecane przy generowaniu szybkich sygnałów. W przypadku podania błędnej wartości lub nie podania żadnej, użytkownik zostanie poinformowany oraz zostanie użyta wartość domyślna wynosząca 1.

### 5.1.5. Polecenie *load*

Wywołanie polecenia powoduje załadownia wygenerowanych danych do pamięci próbek. W przypadku problemów z komunikacją z urządzeniem lub nie wygenerowania danych, użytkownik zostanie poinformowany odpowiednim komunikatem błędu.

### 5.1.6. Polecenie *generate*

Polecenie uruchamia generację sygnału. W przypadku braku komunikacji, użytkownik zostanie poinformowany odpowiednim błędem.

### 5.1.7. Polecenie *reset*

Polecenie daje możliwość wyzerowania akumulatora fazy, licznika błędów oraz układów do serializacji danych. W przypadku błędu komunikacji z urządzeniem informuje użytkownika odpowiednim komunikatem błędu.

### 5.1.8. Polecenie *stop*

Zatrzymuje generację sygnału oraz wyprowadza 0 na wyjściach serializerów. Informuje użytkownika jeśli wystąpi błąd komunikacji.

### 5.1.9. Polecenie *bist*

Polecenie pozwala na przetestowanie poprawności działania układów wyjściowych poprzez serializację i deserializację sygnału pseudolosowego, generowanego wewnątrz układu FPGA. Wynik testu zostaje przekazany użytkownikowi w postaci liczby wektorów testowych, liczby prawidłowo oraz błędnie przetworzonych wektorów. W kolejnych wersjach oprogramowania zostanie zaimplementowana możliwość odczytywania dla jakich wartości wystąpiły błędy w celu analizy wystąpienia potencjalnych uszkodzeń.

### 5.1.10. Polecenie *quit*

Polecenie kończy komunikację z układem FPGA, zamyka otwartą sesję i wyłącza oprogramowanie sterujące.

# Historia wersji

Tabela 5.1: Historia wersji dokumentacji wraz z opisem zmian

| Data wydania | Wersja | Opis |
|---|---|---|
| 15.06.2025 | 1.0.0 | Pierwsza wersja dokumentacji |

# Bibliografia


[1] Hong-fei Zhang Chun-li Luo Peng-yi Tang Ke Cui Sheng-zhao Lin Ge Jin. "High-speed arbitrary waveform generator based on FPGA". W: *IEEE Nuclear Science Symposium and Medical Imaging Conference* (2014).

[2] *Carry-skip adder*. Ostatni dostęp 25 maj 2025. 2024. URL: https://en.wikipedia.org/wiki/Carry-skip_adder.

[3] AMD/Xilinx. *7 Series FPGAs SelectIO Resources User Guide (UG471)*.

[4] AMD/Xilinx. *Vivado Design Suite 7 series FPGA and Zynq 7000 SoC Libraries Guide (UG935)*.

[5] AMD/Xilinx. *Zynq-7000 All Programmable SoC: DC and AC Switching Characteristics*.

[6] AMD/Xilinx. *7 Series FPGAs Clocking Resources User Guide (UG472)*.

[7] AVNET. *ZedBoard Technical Documents*.

[8] Kevin Fronczak. "Stability Analysis of Switched DC-DC BoostConverters for Integrated Circuits". Ostatni dostęp 13 cze 2025. Prac. mag. KATE GLEASON COLLEGE OF ENGINEERING ROCHESTER INSTITUTE OF TECHNOLOGY, 2013. URL: https://www.researchgate.net/publication/293518409_Stability_Analysis_of_Switched_DC-DC_Boost_Converters_for_Integrated_Circuits#pf91.

[9] AMD/Xilinx. *Zynq-7000 SoC Data Sheet (DS190)*.

[10] AMD/Xilinx. *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*.

[11] AMD/Xilinx. *UltraFast Design Methodology Guide for FPGAs and SoCs (UG949)*.

[12] AMD/Xilinx. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)*.

[13] AMD/Xilinx. *OS and Libraries Document Collection (UG643)*.


Piotr Kowol