Błażej Wieczorek
Rok akademicki: 2019-2020

Implementacja generatora przebiegów sinusoidalnych na układ FPGA przy użyciu generatora kodu z pakietu Matlab Simulink.

Streszczenie

W tej pracy zostały przedstawione sposoby pracy z generatorem kodu, jako elementem składowym pakietu Matlab oraz jego praktyczne wykorzystanie w projekcie. Zostanie także poruszony temat optymalizacji modelu w celu umieszczenia algorytmu na rzeczywistym układzie. W dalszej części przejdę do sposobów weryfikacji wygenerowanego kodu w programie symulacyjnym dostarczonym wraz z ISE® Design Suite (który nie jest bezpośrednio wspierany przez pakiet Matlab HDL Coder) oraz weryfikacja przy użyciu narzędzi wspieranych przez to środowisko. Zwrócę także uwagę na zasady tworzenia poprawnych modeli oraz zamieszczę dodatkowe odnośniki rozszerzające opisy znajdujące się w tej pracy. Pakiet HDL Coder wspiera język VHDL i Verilog, przez co możliwe jest proste przenoszenie modelu pomiędzy projektami korzystających z wyżej wymienionych języków opisu sprzętu.

1. Wprowadzenie

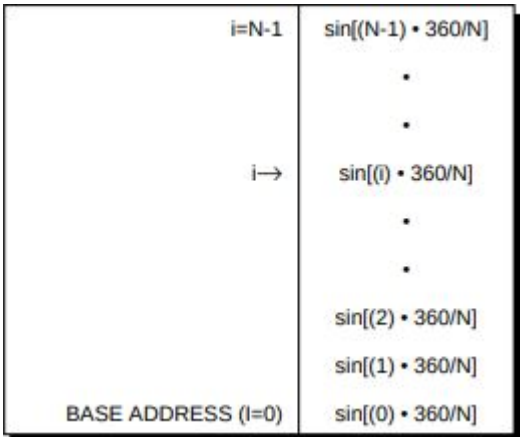
Rynek układów programowalnych FPGA, który jest aktualnie warty około 6 mld dolarów, według przewidywań ma dalej zwiększać swoją wartość do około 13 mld dolarów w 2026 roku. Stały wzrost wartości tego rynku pozwala przewidywać, że układy tego typu będą coraz częściej używane w aktualnych aplikacjach oraz będą przystosowywane do przejęcia innych rynków zdominowanych przez mniej wydajne i tańsze układy. Przez co możemy założyć wzrost zainteresowania sposobami organizacji pracy projektantów stosujących układy FPGA w swoich urządzeniach.

Układy FPGA najczęściej wykorzystywane w aplikacjach wymagających równoległości działania oraz przeprowadzania bardzo szybkich obliczeń (np. przetwarzanie sygnałów telekomunikacyjnych, przetwarzanie obrazu oraz jako sterowniki w bardzo obciążonych systemach czasu rzeczywistego). Coraz częściej także można układać te spotkać w centrach danych, gdzie równoległość przetwarzania jest kluczowa.

Generacja dokładnej wartości funkcji sinus jest ważna w wielu algorytmach sterowania, gdzie poza dokładności samej wartości, kluczowy jest czas jej obliczania. Jednym z przykładów realnej implementacji, gdzie potrzebna jest wartość funkcji sinus i cosinus jest obliczanie transformaty Clarka i Parka w układach sterowania wektorowego silnika BLAC. Transformaty te pozwalają na przejście z układy 3 fazowego prądów do przestrzeni 0, d, q, gdzie możliwa jest implementacja sterowania wektorowego przy użyciu regulatorów PI lub PID. Więcej informacji o tym zastosowaniu znajduje się w odnośniki [2].

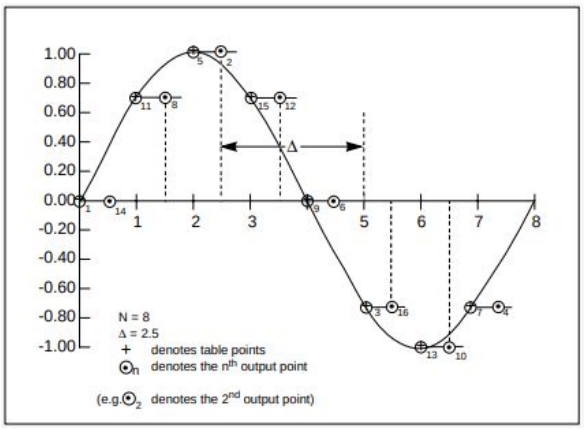
2. Zastosowany algorytm - Lookup Table

Zastosowany algorytm wykorzystuje wartości tablicy składającą się z jednego okresu wartości funkcji sinus. Kąt 2π reprezentowany jest jako każdy indeks tablicy. Dodatkowo podczas implementacji na układ fpga, trzeba zmienić sposób reprezentacji wartości tablicy z wartości zmiennoprzecinkowej na stałoprzecinkową, przez co znacznie zmniejszamy ilość pamięci na przechowywanie tablicy oraz zmniejszamy ilość elementów logicznych wykorzystywanych dla prawidłowego odczytu tablicy.



Rys 1. Zestawienie indeksów z wartościami [1]

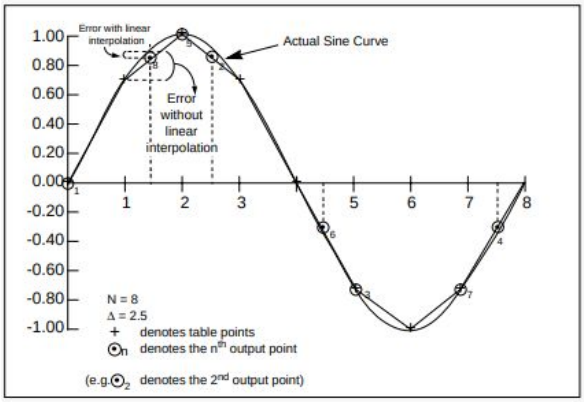
Generacja przebiegu sinusoidalnego (o stałej częstotliwości) polega na iteracją po tablicy z stałą wartością dodaną do indeksu (co takt zegara wewnętrznego), gdy wartość indeksu wykracza poza tablicę następuje odjęcie wielkości tablicy od indeksu, przez co dodatkowo ogranicza się ilość bitów potrzebnych do reprezentacji indeksu.



Rys 2. Przedstawienie iteracji po tablicy przy niecałkowitej wartości zwiększającej indeks [1]

Algorytm Lookup pozwala na generację przebiegu sinusoidalnego w pewnym zakresie częstotliwości, dolnym ograniczeniem algorytmu jest ograniczenie ilości bitów na których zapisany jest indeks (część po przecinku liczby stałoprzecinkowej), natomiast górnym ograniczeniem częstotliwości iterowania spowodowanych jest spadkiem dokładności sinusa, wynikającym z znacznym zmniejszeniem ilości próbek na okres.

Kolejnym problemem występującym podczas odczytywania z tablicy jest przypadek w którym wartości dodawana do indeksu nie jest wartością całkowitą zgodną z układem indeksów (wartość odczytywana znajduje się pomiędzy dwoma indeksami). Problem ten można rozwiązać poprzez zastosowanie interpolacji, zależnie od wymaganej dokładności możemy wybrać różne metody interpolacji. W tym opracowaniu przedstawiam wyłącznie rozwiązanie oparte na liniowej interpolacji wartości sinusoidalnej.



Rys 3. Interpolacja wartości sinusoidy pomiędzy wartościami z tablicy [1]

$$y = m \cdot x + b \quad (1)$$

$$m = S[i+1] - S[i] \quad (2)$$

$$b = S[i] \quad (3)$$

$$x = v - \lfloor v \rfloor \quad (4)$$

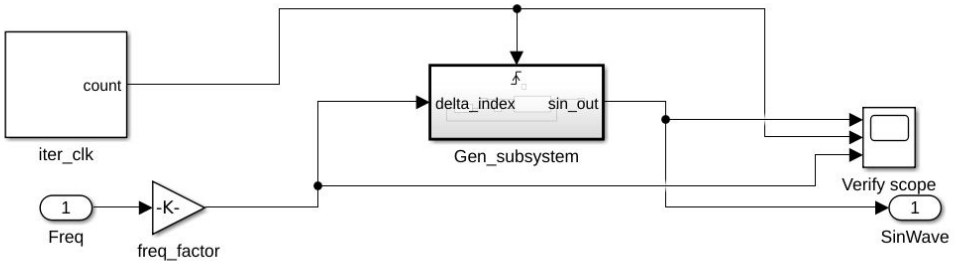
$$i = \lfloor v \rfloor \quad (5)$$

v - wartość indeksu przekazana do odczytu z tablicy

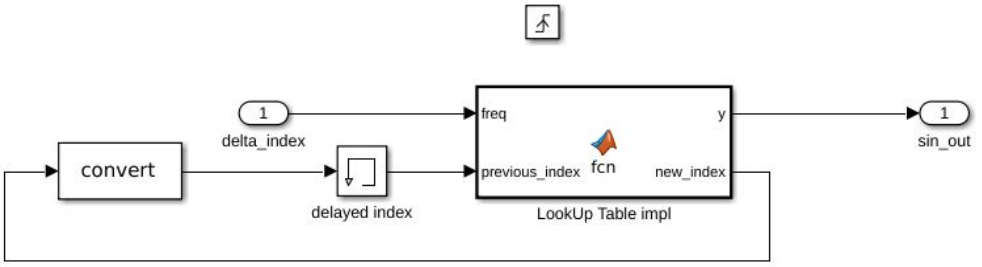
Zastosowanie interpolacji wartości tablicowych oraz przeznaczenie więcej bitów części ułamkowej indeksu można zwiększyć zakres obsługiwanych częstotliwości, które możemy generować poprzez zwiększenie zegara taktującego układ odczytujący, przy tym zmniejszając Δ indeksu. Taki zabieg pozwoli generować przebiegi ze znacznie szerszego spektrum częstotliwości, gdyż błędy spowodowane pomijaniem wartości tablicowych są znacznie wyższe niż błędy interpolacji liniowej na tym zakresie.

3. Implementacja algorytmu w pakiecie Simulink®

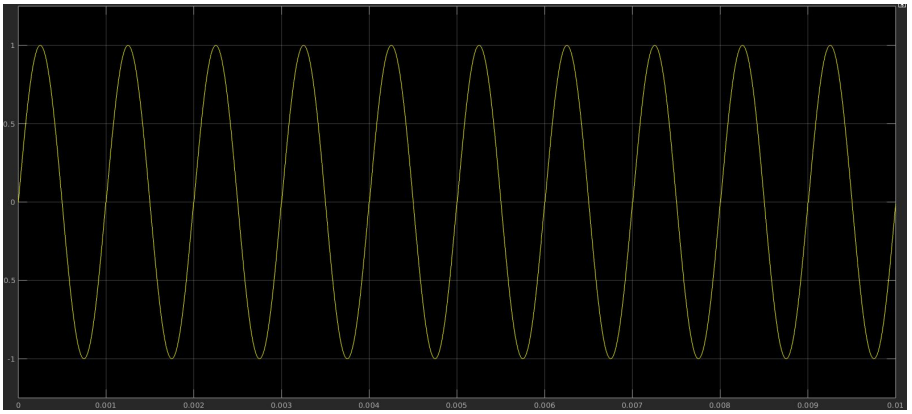
Implementację rozpoczęto od wygenerowania 255 próbek (1 pełny okres) przebiegu sinusoidalnego dla amplitudy równej 1. Dzięki czemu możliwe było zapisanie wartości generowanej jako liczby stałoprzecinkowej z znakiem i tylko jednym bitem części całkowitej i jednego bitu znaku, reszta z wykorzystywanych 18 bitów zostało zaalokowanych jako część ułamkową (limit 18 bitów stosuje się w implementacjach docelowo będących syntezowanych na układy firmy Xilinx, ponieważ większość z nich posiada wbudowane bloki DSP48, które nakładają pewne ograniczenia na wielkość składników operacji arytmetycznych, więcej informacji w źródle [3]).



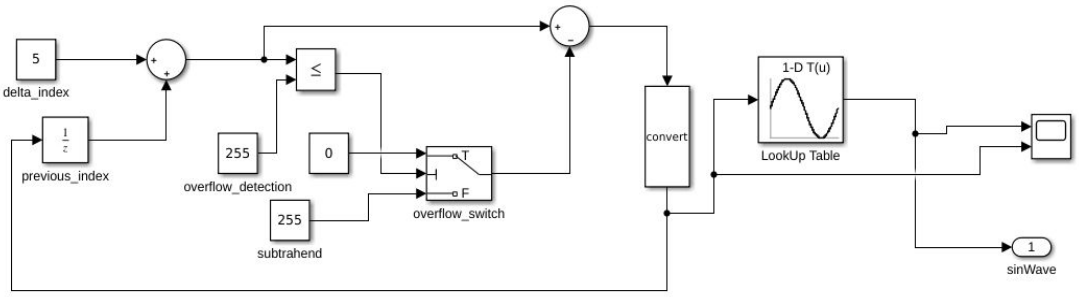
Rys 4. Układ wykorzystujący podsystem aktywowany zboczem narastającym.



Rys 5. Wnętrze podsystemu z blokiem funkcyjnym jako element wykonawczy.



Rys 6. Przebieg generowany na wyjściu układu dla wartości 1 na wejściu układu

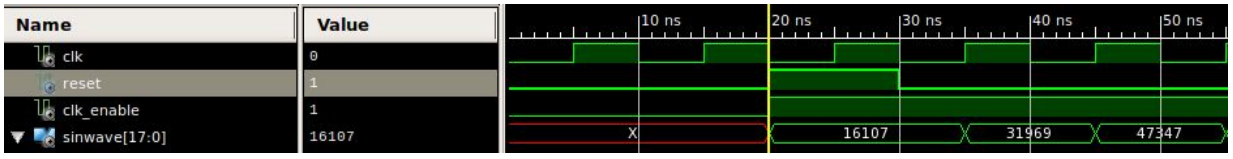


Rys 7. Alternatywny układ zbudowany wyłącznie na blokach dostępnych w pakiecie Simulink

Kod w języku Matlab znajdujący się w bloku LookUp Table impl:

```
function [y, new_index]= fcn(freq, previous_index)
%#codegen
loader = coder.load('tableSin.mat');
table = loader.tableSin;
new_index = fi(previous_index + freq, 0, 19, 8);
if(new_index >= 256)  %check if index overflows the table
new_index = fi(new_index - fi(256,0,9,0),0,19,8);
end
if(mod(new_index, fi(1,0,1,0)) == 0 )%check if index is int
y = fi(table((mod(new_index,fi(256,0,9,0)))+1),1,18,17);
else
index_floor = floor(new_index);
if(index_floor == 255)  %check if ceil value overflows the table
index_ceil = fi(0,0,12,0);
else
index_ceil = ceil(new_index);
end
sin_floor = table(index_floor+1);
sin_ceil = table(index_ceil+1);
a = sin_ceil - sin_floor;
y = fi(a*mod(new_index,fi(1,0,1,0)) + sin_floor, 1, 18, 17);
end
```

Powyższe rysunki pokazują sposób w jaki możemy stworzyć syntezowalny system w środowisku Simulink korzystając z bloków z pakietu HDL Coder. Możliwa jest implementacja jednego układu na wiele sposobów, każdy z nich zależy od preferencji samego użytkownika, zastosowanie bloków funkcyjnych pozwala na tworzenie bardziej skomplikowanych układów i obliczeń, lecz układy zbudowane wyłącznie w oparciu o gotowe bloki, będą zazwyczaj lepiej zoptymalizowane pod kątem danego układu. W ustawieniach modelu, jest możliwość wybrania docelowego układu, taktowania zegara doprowadzonego do układu oraz metod optymalizacji kodu. W przypadku pracy na liczbach zmiennoprzecinkowych możliwe jest korzystanie z zewnętrznych bibliotek (IP Core) dedykowanych do układu którym się posługujemy zamiast standardowej biblioteki zawartej w pakiecie Matlab.



Rys 8. Sekwencja rozpoczynająca pracę generatora

Aby rozpocząć pracę układu należy podać impuls na wejście reset oraz stan wysoki na wejście clk_enable. Do czasu podania impulsu na wejście reset wyjście układu znajduje się w stanie nieustalonym, po podaniu impulsu układ przechodzi do stanu ustalonego. Aby rozpocząć generacje należy podać stan wysoki na wejście układu, jest to sygnał aktywujący zegar dla układów wewnętrznych generatora (część środowisk jak ISE DS, przy tworzeniu pliku Test Bench będzie traktować sygnał clk_enable jako zegar, nie jest to poprawne, należy wtedy zmienić ręcznie tą wartość w pliku, jest to spowodowane wyrazem clk w nazwie, który jest zarezerwowany dla sygnałów zegarowych, można tę nazwę zmienić w opcjach modelu w pakiecie Simulink).

```
Selected Device : 6slx9tqg144-2


Slice Logic Utilization:
 Number of Slice Registers:               8  out of  11440     0%
 Number of Slice LUTs:                  198  out of   5720     3%
    Number used as Logic:               198  out of   5720     3%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    198
   Number with an unused Flip Flop:     190  out of    198    95%
   Number with an unused LUT:             0  out of    198     0%
   Number of fully used LUT-FF pairs:     8  out of    198     4%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                        22
 Number of bonded IOBs:                22  out of    102    21%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:              1  out of     16     6%
 Number of DSP48A1s:                    2  out of     16    12%
```

Rys 9. Raport z syntezy układu z rys nr 7 dla układy Spartan6

```
Device utilization summary:
---------------------------

Selected Device : 6slx9tqg144-2


Slice Logic Utilization:
 Number of Slice Registers:              35  out of  11440     0%
 Number of Slice LUTs:                  410  out of   5720     7%
    Number used as Logic:               410  out of   5720     7%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    426
   Number with an unused Flip Flop:     391  out of    426    91%
   Number with an unused LUT:            16  out of    426     3%
   Number of fully used LUT-FF pairs:    19  out of    426     4%
   Number of unique control sets:         2

IO Utilization:
 Number of IOs:                         22
 Number of bonded IOBs:                 22  out of    102    21%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     16     6%
 Number of DSP48A1s:                     1  out of     16     6%
```

Rys 10. Raport z syntezy układu z rys nr 4 dla układy Spartan6

```
Device utilization summary:
---------------------------

Selected Device : 6slx9tqg144-2


Slice Logic Utilization:
 Number of Slice Registers:            8086  out of  11440    70%
 Number of Slice LUTs:                15290  out of   5720   267% (*)
    Number used as Logic:             14732  out of   5720   257% (*)
    Number used as Memory:              558  out of   1440    38%
       Number used as SRL:              558

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  18267
   Number with an unused Flip Flop:   10181  out of  18267    55%
   Number with an unused LUT:          2977  out of  18267    16%
   Number of fully used LUT-FF pairs:  5109  out of  18267    27%
   Number of unique control sets:         2

IO Utilization:
 Number of IOs:                         68
 Number of bonded IOBs:                 68  out of    102    66%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     16     6%
 Number of DSP48A1s:                     8  out of     16    50%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used

---------------------------
```

Rys 11. Raport z syntezy układu z rys nr 4 (liczby zmiennoprzecinkowe)

Porównując raporty utworzony przez narzędzie syntezujące wygenerowany kod możemy dojść do wniosku, że oba układu zużywają podobną ilość zasobów układu. Największą różnicą jest mniejsza ilość wykorzystywanych modułów DSP w układzie z rys 4, co jest bardzo pożądane, gdyż ilość tych bloków jest bardzo ograniczona, a ich wykorzystanie w tym przypadku zdaje się nadmiarowe. Jakość optymalizacji modelu wykorzystującego bloki funkcyjne zależy w głównej mierze od przemyślanego modelu funkcji znajdującej się w tym bloku, jeśli funkcja jest dobrze zoptymalizowana i nie zawiera żadnych nadmiarowych obliczeń, możliwe jest osiągnięcie poziomu zużycia elementów logicznych na podobnym poziomie ja bloki dedykowane, co jest dobrym prognostykiem, gdy problem który rozwiązujemy jest bardziej złożony. Efekty braku optymalizacji stałoprzecinkowej możemy zauważyć na rys 11.

4. Wnioski:

Układy FPGA będą zwiększać swój udział w rynku, przez co będzie zwiększać się ilość narzędzi stosowanych przez programistów i architektów tych układów. Zastosowanie generatora kodu ma wiele zalet, między innymi bardzo szybki czas tworzenia modelu, możliwość wysokopoziomowej weryfikacji modelu oraz zapewnia możliwość projektowania układu na wyższym poziomie niż przy użyciu docelowych języków opisy sprzętu. Narzędzie to może być także przydatne osobom, które nie potrafią pisać w żadnym języku opisu sprzętu, lecz znają język Matlab oraz potrafią obsługiwać pakiet Simulink, są w stanie zaimplementować algorytmy, które mogą być wykorzystywana jako część innego systemu pisanego w języku docelowym. Generowany kod jest także możliwy do przeczytania i zrozumienia przez człowieka z średnią znajomością języka generowanego, lecz problemem jest zrozumienie zapisu bloków funkcyjnych, w tych przypadkach kod był nieczytelny i chaotyczny.

5. Literatura:

[1]http://home.agh.edu.pl/~turcza/pinz/gen/APR1_SIN_GEN.PDF
[2]http://www.komel.katowice.pl/ZRODLA/FULL/113/ref_03.pdf
[3]https://www.xilinx.com/support/documentation/user_guides/ug389.pdf