

Generator sygnału 1 kHz

Raport Techniczny

Szymon Januszek, szymon_j@tutanota.com

Kwiecień 2023

Streszczenie

Głównym celem poniższej konstrukcji jest otrzymanie sygnału możliwie najwyższej jakości, zdefiniowanych w założeniach konkursowych, tzn. dokładność częstotliwości, stopień zniekształceń harmonicznym oraz dokładność amplitudy sygnału.

Spis treści

1	Bezpośrednia synteza cyfrowa	3
1.1	Mikrokontroler	3
1.2	Przetwornik Cyfrowo-Analogowy	3
1.3	Analiza sygnału wyjściowego	4
2	Kształtowanie	4
3	Kontrola Amplitudy	5
3.1	Pomiar amplitudy	5
3.2	Element sprzęgający	5
3.3	Źródło odniesienia	6
4	Oprogramowanie	6
4.1	Kod źródłowy	8

1 Bezpośrednia synteza cyfrowa

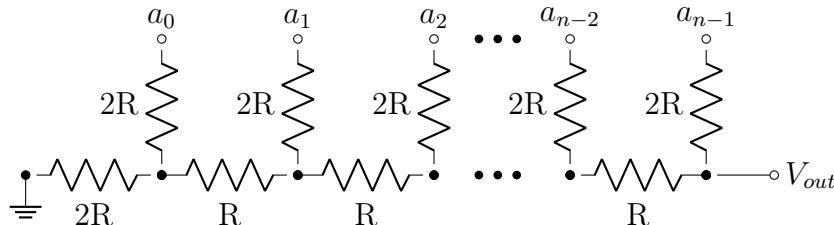
Aby wygenerować maksymalnie stabilny i dokładny względem częstotliwości sygnał, zdecydowałem się na wykorzystanie techniki **Bezpośredniej Syntezy Cyfrowej**, gdzie mikroprocesor wraz z układem Przetwornika Cyfrowo-Analogowego (DAC) generuje sygnał zbliżony do sygnału oczekiwanego. Umożliwia to zastosowanie wysokiej jakości generatora kwarcowego, jako głównego źródła odniesienia, o precyzji rzędu 50ppm¹, w całym zakresie temperatur. Oznacza to, iż częstotliwość uzyskanego sygnału nie będzie odbiegać o więcej niż ± 50 mHz od wartości oczekiwanej.

1.1 Mikrokontroler

Jako główny mikrokontroler wybrany został układ **AVR32DA28**[2] należący do rodziny 8-bitowych układów AVR. W porównaniu ze starszymi generacjami, producent dopuszcza pracę układu przy częstotliwościach przekraczających 8 MHz z napięciem 3 V. Jednocześnie dokonana została unifikacja modelu pamięci, co okaże się przydatne przy pisaniu oprogramowania(4).

1.2 Przetwornik Cyfrowo-Analogowy

Do implementacji przetwornika zrealizowany został układ drabinki R-2R (Rys.1). Umożliwiając łatwą zmianę wartości liczbowej, przedstawionej jako liczba binarna na pinach mikrokontrolera, na ułamek napięcia zasilania.



Rysunek 1: Schemat drabiny R-2R [1]

Napięcie na węźle wyjściowym przetwornika można opisać w następujący sposób:

$$V_{out} = V_{ref} \times \frac{a_0 \times 2^0 + a_1 \times 2^1 + a_2 \times 2^2 + \dots + a_{N-1} \times 2^{N-1}}{2^N}$$

Jak widać, przy $N = 8$, przetwornik ten pozwala na uzyskanie 256 różnych wartości. Dla $V_{ref} = 3$ V, oznacza to, iż różnica pomiędzy dwoma dowolnymi stopniami wynosi ok. 11.72 mV.

Co ciekawe, okazuje się, iż impedancja wyjściowa takiego układu jest stała dla każdego poziomu ².

¹Części na milion

²Wynika to z Twierdzenia Thévenina, zakładającego zerową impedancję źródła oraz mikrokontrolera. W rzeczywistości ich wartości są znacznie mniejsze od wartości R.

Aby zachować użyteczność najniższych bitów, precyzja wykorzystanych rezystorów musi być lepsza niż $\frac{1}{2^N} \times 100\% \approx 0.3\%$. Na szczęście, w sprzedaży dostępne są rezystory przeznaczone do montażu powierzchniowego o precyzji 0.1%.

1.3 Analiza sygnału wyjściowego

Sygnał generowany w ten sposób nie opowiada jednak perfekcyjnej sinusoidzie (Rys. 2). Obecne jest tzw. zniekształcenie kwantyzacji, wynikające z faktu, iż sygnał złożony jest z serii dyskretnych wartości. Jednak jeśli dokładnie przyjrzymy się różnicy pomiędzy naszym sygnałem a idealnej sinusoidzie, to okaże się, że amplituda takiego sygnału jest nie wielka ($\Delta V \leq \frac{V_{ref}}{2^N}$), a także składa się on z częstotliwości wielokrotnie większych niż częstotliwość fundamentalna generowanego sygnału.



Rysunek 2: Przebieg sinusoidy generowanej przy pomocy przetwornika R-2R.

Możemy uznać, iż dobrym przybliżeniem takiego sygnału będzie suma sinusoidy o amplitudzie równej połowie napięcia zasilania V_{ref} i o częstotliwości f_0 oraz sygnału prostokątnego o częstotliwości $f_0 \times N_{smp}$ i amplitudzie $0.5 \times \frac{V_{ref}}{2^N}$, gdzie N_{smp} oznacza ilość dyskretnych zmian napięcia w czasie jednego okresu.

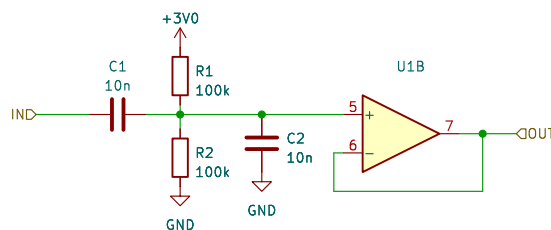
Widzimy więc że częstotliwość już częstotliwość drugiej składowej jest N_{smp} - krotnie większa, a jej amplituda $2^{N-2}\pi$ -krotnie mniejsza w porównaniu ze składową fundamentalną. Oznacza to, iż przy zastosowaniu filtra dolnoprzepustowego niskiego stopnia, możliwe jest efektywne usunięcie składowych wyższej częstotliwości.

2 Kształtowanie

Mimo wszystko należy usunąć składowe wyższej częstotliwości. Funkcję tą realizuje kondensator C2 (Rys. 3).

Dodatkowo, rzeczywiste wzmacniacze operacyjne nie są w stanie generować sygnałów bardzo bliskich < 300 mV szyn zasilania. Tłumienia tego dokonują C1, R1, R2,

zachowując wartość środkową na poziomie 1.5 V



Rysunek 3: Schemat filtru

3 Kontrola Amplitudy

Ostatnią zmienną którą należy rozpatrzyć jest amplituda sygnału wyjściowego. Na tym etapie amplituda sygnału jest większa od oczekiwanej wartości 0.5 V. Potrzebna jest więc metoda dokładnego tłumienia, nie wprowadzająca dodatkowych zniekształceń i zakłócenia.

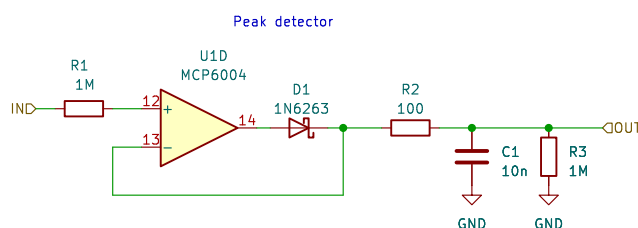
Rozwiązanie oparte o nawet precyzyjny potencjometr pozostaje jednak wrażliwe na takie zmienne jak napięcie zasilania, czy też dryft parametrów elementów związany ze zmianą temperatury.

Tak więc, niezbędny jest układ aktywnie korygujący amplitudę sygnału. Cyfrowa implementacja takiego rozwiązania nie wchodzi w grę, ponieważ przetwornik DAC nie posiada wystarczającej głębokości bitowej. Stąd też, proponuję rozwiązanie w pełni analogowe.

Przy użyciu wzmacniaczy operacyjnych możemy uzyskać pętle sprzężenia zwrotnego, korygującą amplitudę sygnału względem stałego napięcia odniesienia. Do realizacji tego rozwiązania potrzebujemy następujących elementów.

3.1 Pomiar amplitudy

Konieczny jest pomiar amplitudy sygnału wyjściowego. To tego celu służy tzw. Detektor szczytowy (Rys. 4).



Rysunek 4: Przykładowy schemat połówkowego detektora szczytowego

3.2 Element sprzęgający

Wymagany jest układ umożliwiający tłumienie sygnału zmiennego względem sygnału sterującego:

$$V_{wyj} \propto \frac{V_{wej}}{V_{ster}} \quad (1)$$

Fizyczna implementacja takiego układu okazuje się być niebanalna. Możliwy jest układ realizujący operacje mnożenia wartości sygnałów. Jego realizacja jednak jest wysoce złożona i wymaga relatywnie dużo części, ponieważ w jego skład wchodzi układy wyznaczające logarytm naturalny obu wartości, dodające te wartości do siebie, a na koniec wyznaczające funkcje wykładniczą e^x tej sumy.

Alternatywna metoda wykorzystuje element zmieniający swój opór elektryczny względem jakiegoś innego czynnika. Przykładem historycznej realizacji takiego układu jest zastosowanie niewielkiej lampy żarowej, której opór włókna jest wprost proporcjonalny do jego temperatury.

Istnieje jednak znacznie prostsze i wydajniejsze rozwiązanie.

Popularnym elementem o zmiennym oporze jest fotorezystor. W przeciwieństwie do innych elementów światłoczułych, przy stałym natężeniu światła, zachowuje się on jak faktyczny rezystor, tzn. prąd przepływający jest wprost proporcjonalny do napięcia i odwrotnie do jego oporu $I \propto \frac{V}{R_\phi}$.

Tak więc, jeśli zamknie się fotorezystor w obudowie światłoszczelnej wraz z diodą LED, uzyskamy układ o zmiennym oporze elektrycznym, kontrolowany przez prąd diody³.

$$R \propto \frac{1}{I_{Led}} \quad (2)$$

Umożliwia to łatwą implementację układu sprzężenia (Eq. 1).

Co prawda powyższy model ignoruje fakt, iż tak na prawdę opór fotorezystora zależy wykładniczo od natężenia padającego światła $R = \left(\frac{1}{\phi}\right)^A$. Nie stanowi jednak to dużego problemu ponieważ układ ten znajduje się w pętli sprzężenia zwrotnego.

$$R \propto \frac{1}{(I_{Led})^A} \quad (3)$$

3.3 Źródło odniesienia

W tym momencie potrzebne jest jedynie stabilne oraz dostrajalne źródło napięcia odniesienia. Do tego celu z zupełności wystarczy układ LM4041 wraz z precyzyjnym potencjometrem.

4 Oprogramowanie

Czas przejść do opisu kodu źródłowego. W pliku `sine_tab.cpp` znajduje się definicja tabeli funkcji sinus. Składa się ona z 1024 próbek, a wartość i -tej próbki zdefiniowana jest w następujący sposób:

$$f(i) = (2^N - 1) + \left\lfloor (2^N - 1) \times \sin\left(\frac{2\pi \times i}{N_{smp}}\right) \right\rfloor \quad (4)$$

³Zakładam $\phi \propto I$

$$f(i) = 128 + \left\lfloor 127 \times \sin\left(\frac{2\pi \times i}{1024}\right) \right\rfloor \quad (5)$$

Każda próbka jest 8-bitową liczbą całkowitą z zakresu $\in \{1 \dots 255\}$, jednocześnie tabela przedstawia cały jeden okres funkcji sinus, poza wartością $\sin(2\pi)$. Następną próbkę stanowi wartość $\sin(0)$. Dzięki temu reprodukowany sygnał nie zawiera przeskoku co cały okres.

Zadaniem oprogramowania mikrokontrolera pozostaje wczytanie kolejnych wartości z pamięci, a następnie ustawienie pinów na odpowiedni stan. Wykorzystując 8-bitową naturę architektury AVR możliwe jest ustawienie stanu całego rejestru (tj. grupy 8 pinów) poprzez pojedynczy zapis do obszaru peryferialnego pamięci (Linia 35)

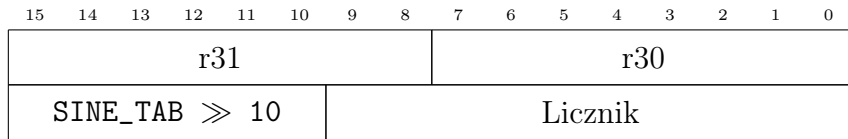
W tym miejscu napotykamy jednak pewien problem, synteza powyższego sygnału wymaga produkcji ponad miliona próbek na sekundę, tym czasem zastosowany mikroprocesor jest przystosowany do pracy przy częstotliwości nie przekraczających 24 MHz. Przy zastosowanym generatorze kwarcowym o częstotliwości pracy 12.288 MHz oznacza to, że mamy dokładnie 12 cykli maszynowych na wygenerowanie następnej próbki, co więcej aby zachować precyzję częstotliwości musimy wykorzystać wszystkie 12 cykli za każdym razem. Stąd też decyzja o implementacji całego algorytmu bezpośrednio w niskopoziomowym assemblerze. Pozwala to na dokładną kontrolę nad czasem wykonania całego cyklu, a jednocześnie umożliwia szereg optymalizacji będących poza zasięgiem kompilatora.

Kolejne bajty pobierane są przez instrukcje `ld` (Linia 32), która jednocześnie inkrementuje wskaźnik tworzony przez parę rejestrów `r30` i `r31`.

Problemem pozostaje jedynie wydajne zawijanie wskaźnika po dotarciu do 1024-tej wartości. Ponownie wykorzystując 8-bitową naturę układów AVR, możliwe jest bardzo wydajne rozwiązanie tego problemu.

Jednym z atrybutów udostępnianych przez kompilator GCC jest `__attribute__((aligned (1024)))`, który pozwala na wymuszenie wyrównania pozycji pierwszego bajtu danej zmiennej do wielokrotności liczby podanej jako argument. Oznacza to iż pozycja tablicy przyjmuje postać $A \times 1024$.

Tak więc, niższy bajt 16-bitowego wskaźnika będzie ulegał stałym zmianą na całej swojej długości, natomiast w bajcie wyższym, dzięki wyrównaniu tabeli, zmieniać się będą jedynie dwa najniższe bity (Rys. 5).



Rysunek 5: Schemat rozkładu bitów we wskaźniku

Tak więc jedyne co tak na prawdę musimy robić to nie pozwolić na zmianę stanu 6 najwyższych bitów wskaźnika. Możemy to osiągnąć używając sprytniej sztuczki, zajmującej łącznie 2 cykle zegarowe (Linia 38).

$$r31' = ((r31 \wedge 11_{bin}) \vee r20) \quad (6)$$

Po pierwsze zerujemy sześć najwyższych bitów górnego rejestru instrukcją ANDI - (And with Immediate) z liczbą 3_{dec} . Następnie instrukcja OR przywraca poprawny stan wyższych bitów, które na początku działania programu zostały zapisane w rejestrze 20 (Linijka 26).

4.1 Kod źródłowy

```
1 ;© Szymon Januszek 2023
2 #include <avr/io.h>
3 #include "config.h"
4
5 .extern SINE_TAB
6 .global main
7
8 main:
9     ;temporarily use r1 for ZERO
10    eor r1, r1
11
12    ;select external clock as Main Clock source
13    ldi r16, 0xD8; MAGIC VALUE
14    ldi r17, 0x83
15    out CPU_CCP, r16
16    sts CLKCTRL_MCLKCTRLA, r17; use external clock
17
18    out CPU_CCP, r16
19    sts CLKCTRL_MCLKCTRLB, r1; disable prescaler
20
21    ;set port direction and clear
22    ldi r16, 0xFF
23    sts PORTD_DIRSET, r16
24    out PORT_OUT, r1
25
26    ;load pointer to Z reg. pair
27    ldi r20, hi8(SINE_TAB)
28    mov r31, r20
29    mov r30, r1
30
31 loop:
32    ;pull next byte out of memory and increment the pointer
33    ld r2, Z+
34
35    ;output data
36    out PORT_OUT, r2
37
38    ;cyclize data pointer
39    andi r31, 0x03
40    or r31, r20
```



```
41
42      ;pad for 12.288 MHz
43      nop
44      nop
45      nop
46      nop
47
48      rjmp loop
```

References

- [1] CC BY-SA 2.0. URL: <https://commons.wikimedia.org/wiki/File:R2r-ladder.png>.
- [2] Microchip corp. *AVR32DA Final Data Sheet*. URL: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/AVR32DA28-32-48-Data-Sheet-40002228B.pdf>.



Rysunek 6: Kot autora wspomagający proces twórczy