Sprawozdanie z projektu końcowego

Systemy wbudowane i czasu rzeczywsitego — Laboratorium 2013L

Data	2013-05-15
Autorzy	Tomasz Cudziło, Mateusz Malicki

1 Treść zadania

W ramach projektu mieliśmy zamodelować w środowisku MATLAB układ pętli synchronizacji fali. Układ miał na pojedynczym wejściu przyjmować sumę trzech sygnałów:

- $f_1 = 50$ Hz i przesunięciu fazowym $\varphi_1 = 5°$,
- $f_3 = 150$ Hz i przesunięciu fazowym $\varphi_3 = 10°$,
- $f_5 = 250$ Hz i przesunięciu fazowym $\varphi_5 = 30°$.

W otrzymanym sygnale układ miał zsynchronizować się do sygnału o częstotliwości f_5 i podać go na wyjście.

2 Wykonanie

2.1 Prototyp na blokach Simulink

Stworzyliśmy w pełni działający układ korzystając z niezależnych bloków z pakietów dostarczanych przez Simulink. Miało to ułatwić wprowadzanie zmian parametrów sterujących podzespołami i dodatkowych punktów kontrolnych wewnątrz układu. Schemat prototypu znajduje się na rysunku 1 na stronie 7.

Układ jest zbudowany identycznie z modelem zaprezentowanym w czasie zajęć laboratoryjnych. Szczegóły implementacyjne podzespołów i dobrane parametry są identyczne ze szczegółami modelu końcowego opisanego w sekcji 2.2.

2.2 Wersja końcowa zaimplementowana w C

Układ w wersji końcowej składa się z jednego bloku łączącego podzespoły z prototypu wewnątrz swojej implementacji w C. Schemat układu PLL w wersji końcowej znajduje się na stronie 8.
Kod C bloku `PLL` realizujący cały układ jest przedstawiony poniżej na listingu 1. W następnych sekcjach opisane są szczegółowo funkcje realizujące kolejne podzespoły układu.

```
// blok PLL
static double feedback_in = 1.0;

lpf_out[0] = low_pass_filter(signal_in[0] * feedback_in);
```

```
pi_ctrl_out[0] = pi_controller(lpf_out[0]);
saw_out[0] = saw_tooth_generator();

pll_out[0] = sin(saw_out[0] + pi_ctrl_out[0]);
feedback_in = cos(saw_out[0] + pi_ctrl_out[0]);
```

Listing 1: Implementacja układu PLL w C.

2.2.1 Detektor przesunięcia fazy

Został zaimplementowany zgodnie z wytycznymi z zajęć. Zasada działania jest prosta. Iloczyn wartości sygnałów referencyjnego i porównywanego jest podawany na wejście filtra dolnoprzepustowego. Pozwala to z wystarczająco dobrym przybliżeniem wyznaczyć wartość przesunięcia fazowego.

Filtr dolnoprzepustowy wykorzystany w układzie został wygenerowany za pomocą programu WinFilter w wersji 0.8 [1]. Do naszych potrzeb najlepszy okazał się filtr dolnoprzepustowy Butterwortha czwartego rzędu. Parametry operacyjne filtru dobraliśmy eksperymentalnie. Ich wartości są wypisane w tabeli 2.

Parametr	**Wartość**
Model filtru	Butterworth
Rząd filtru	4
Częstotliwość próbkowania (narzucona)	10 kHz
Częstotliwość odcięcia	0,025 kHz

Tabela 2: Parametry operacyjne filtru dolnoprzepustowego wchodzącego w skład detektora fazy.

Funkcjonalność filtru jest zrealizowana w jednej funkcji `low_pass_filter`. Jej definicja znajduje się w listingu 2. Jako jedyny parametr przyjmuje wartość nowej próbki sygnału. Zwraca wartość sygnału po przefiltrowaniu.

```
// low_pass_filter.c
#define NCoef 4

double low_pass_filter(double newSample) {
    double ACoef[NCoef+1] = {
        0.00000000378517015213,
        0.00000001514068060852,
        0.00000002271102091279,
        0.00000001514068060852,
        0.00000000378517015213
    };

    double BCoef[NCoef+1] = {
        1.00000000000000000000,
```

```
        -3.95895331864708360000,
        5.87770027353614570000,
        -3.87853054905173390000,
        0.95978365381149922000
    };

    static double y[NCoef+1]; // output samples
    static double x[NCoef+1]; // input samples
    int n;

    // shift the old samples
    for (n = NCoef; n > 0; n--) {
       x[n] = x[n-1];
       y[n] = y[n-1];
    }

    // calculate the new output
    x[0] = newSample;
    y[0] = ACoef[0] * x[0];
    for (n = 1; n <= NCoef; n++)
        y[0] += ACoef[n] * x[n] - BCoef[n] * y[n];

    return y[0];
}
```

Listing 2: Implementacja filtru dolnoprzepustowego wchodzącego w skład dektora fazy.

2.2.2 Kontroler proporcjonalny-integrujący

Wyjście z detektora fazy jest podawane na wejście kontrolera propocjonalnego- integrującego. Jest on realizowany przez funkcję `pi_controller`, która przyjmuje wartość kolejnej próbki sygnału i zwraca sygnał po przeprowadzeniu obu operacji. Parametry kontrolera również zostały dobrane eksperymentalnie, zgodnie ze wskazówkami z konsultacji. Wybrane wartości parametrów są zestawione w tabeli 3. Tak dobrane parametry pozwalają na najszybsze zsynchronizowanie układu dla przypadku opisanego w naszym zadaniu (vide sekcja 1).

Parametr	**Wartość**
Waga członu integrującego	0,0075
Waga członu proporcjonalnego	0,3000

Tabela 3: Parametry operacyjne kontrolera PI przetwarzającego sygnał detektora fazy.

Implementacja funkcji realizującej kontroler PI jest identyczna z implementacją wykonaną podczas pierwszych zajęć laboratoryjnych. Jest przedstawiona poniżej na listingu 3.

```c
// pi_controller.c
#define K_i 0.0075
#define K_p 0.3

double pi_controller(double u)
{
    static double u_p, y_p;

    double y = (K_p + K_i) * u - K_p * u_p + y_p;
    y_p = y;
    u_p = u;
    return y;
}
```

Listing 3: Implementacja kontrolera PI przetwarzającego sygnał z detektora fazy.

2.2.3 Generator sygnału referencyjnego piło-kształtnego

Sygnał z detektora przesunięcia fazy, przetworzony przez kontroler PI jest dodawany jako korekcja do sygnału referencyjnego nadawanego przez generator sygnału piło-kształtnego. Zgodnie z wytycznymi zadania generator na wyjściu podawał sygnał o częstotliwości $f_{ref} = 250\,\text{Hz}$ i amplitudzie $|s_{ref}| = 2\pi$.
Jest on realizowany przez funkcję `saw_tooth_generator`, która nie przyjmuje argumentów i zwraca wartość kolejnych próbek sygnału referencyjnego. Jej implementacja znajduje się w listingu 4.

```c
#define pi 3.14159265358979323846264433
#define f_smp 10000
#define f_target 250

double saw_tooth_generator()
{
    const double dy = 2*pi / f_smp * f_target;
    static double y = -2*pi / f_smp * f_target; // == -dy

    y = fmod(y+dy, 2*pi);

    return y;
}
```

Listing 4: Implementacja generatora sygnału referencyjnego.

Tak skorygowany sygnał jest podawany na wyjście układu oraz w sprzężeniu zwrotnym na wejście detektora fazy.

3 Analiza działania

3.1 Czas synchronizacji układu PLL

Zaimplementowany układ PLL synchronizuje się z sygnałem $f_5 = 250\,\text{Hz}$ i $\varphi_5 = 30°$ w czasie krótszym niż $0,042$ s. Jest to sygnał docelowy podany w treści zadania projektowego z sekcji 1. Wykres obrazujący proces synchronizacji jest generowany w MATLABie przez scope o nazwie *Scope for target and PLL output signals*. Scope wyświetla trzy sygnały:

- yellow – sygnał docelowy z zadania,
- magenta – sygnał wyjściowy układu PLL,
- cyan – sygnał referencyjny.

Parametry operacyjne podzespołów układu zostały dobrane tak, by uzyskać jak najszybszą synchronizację dla sygnału o fazie $\varphi_5 = 30°$. Zmiana przesunięcia fazy φ_5 powoduje zmianę czasu synchronizacji, który jest niekoniecznie optymalny.

Wartość φ_5 [°]	**Przybliżony czas synchronizacji [s]**
5	$0,033$
10	$0,035$
15	$0,038$
30	$0,042$
45	$0,046$
60	$0,050$
90	$0,064$
120	$0,084$
180	$0,148$

Tabela 4: Przedstawienie zależności czasu synchronizacji od przesunięcia fazowego sygnału docelowego.

3.2 Stabilność układu PLL

Układ jest stabilny dla dobranych parametrów operacyjnych podzespołów opisanych w sekcji 2. Wartości te zostały dobrane specyficznie dla przypadku z zadania projektowego. Zwiększenie współczynnika członu integracyjnego kontrolera PI szybko prowadzi do utraty stabilności całego układu. Obniżenie częstotliwości odcięcia filtru dolnoprzepustowego ma identyczny efekt. Dobrane przez nas parametry operacyjne podzespołów są unikalne dla częstotliwości sygnału, do którego układ PLL ma się synchronizować.

4 Uwagi

Oba modele układów PLL załączone do sprawozdania zostały zrealizowane z sukcesem. Oba działają identycznie. Oba wykazują się tym samym błędem po osiągnięciu synchronizacji. Sygnał generowa-

ny przez układ PLL jest przesunięty w fazie o jeden krok próbkowania do przodu względem sygnału docelowego.

Nie jesteśmy pewni źródła tego błędu. Pomimo licznych prób nie udało nam się go wyeliminować. Zakładamy, że jest to dowolna kombinacja poniższych elementów:

4.1 Dokładność kontrolera PI

Sądzimy, że sygnał wyjściowy kontrolera PI zbiega do nieznacznie większej wartości niż powinien. Niestety, pomimo licznych prób nie udało nam się dobrać parametrów dla kontrolera, które niwelowały błąd — sekcja 2.2.2.

4.2 Dokładność generatora sygnału referencyjnego

Dokładność wartości sygnału referencyjnego ma krytyczne znaczenie dla uzyskania synchronizacji. Pomimo, wydawałoby się, trywialnej i niezawodnej implementacji generatora (sekcja 2.2.3) możliwe jest, że przy aktualnej kombinacji częstliwości próbkowania $f_{smp} = 10\,\text{kHz}$ i częstotliwości sygnału referencyjnego $f_{target} = 0.25\,\text{kHz}$ odchodzimy od właściwej wartości w sąsiedztwie wartości brzegowych 0 lub 2π.

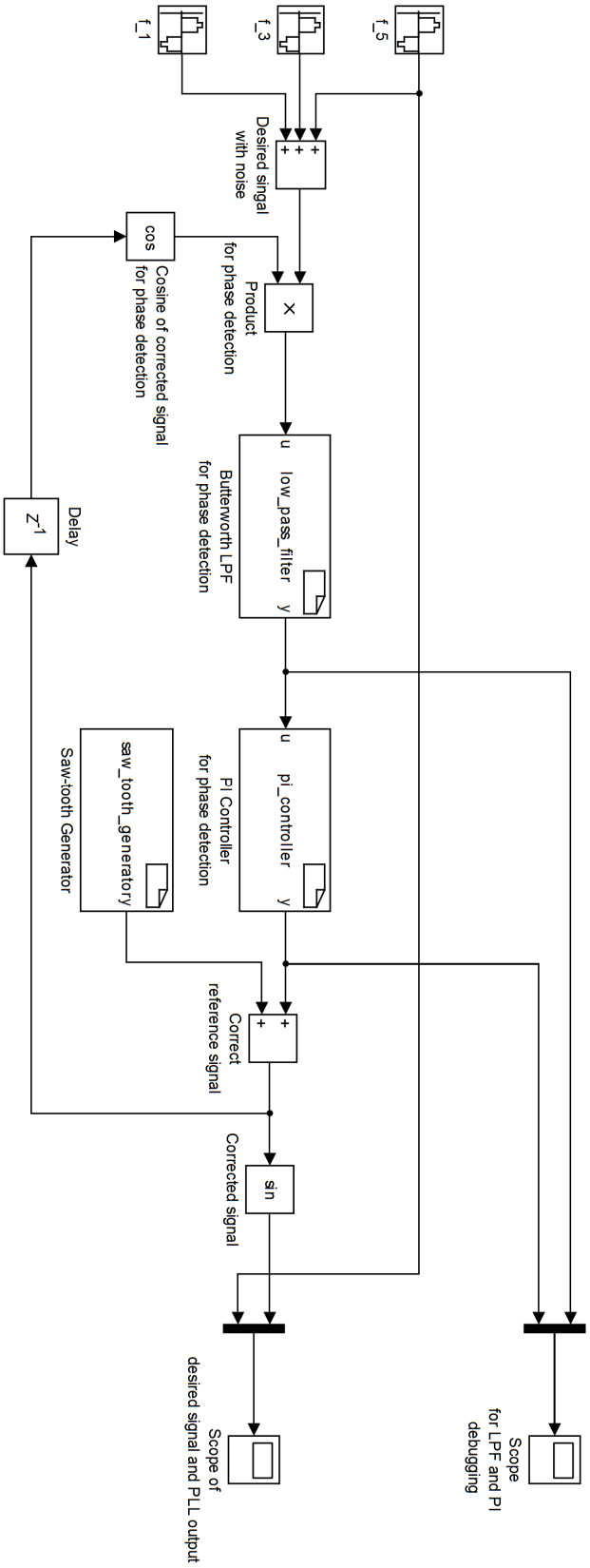
4.3 Dokładność filtru dolnoprzepustowego

Przy aktualnie dobranych parametrach, częstotliwość odcięcia na filtrze wynosi $f_{cut} = 0,025\,\text{kHz}$. Idealnie byłoby ustalić jeszcze niższą częstotliwość odcięcia rzędu $0,001\,\text{kHz}$. Jednak dla narzuconych parametrów filtru to jest model Butterwortha, rzędu 4. i częstotliwości próbkowania $f_{smp} = 10\,\text{kHz}$ ustalenie f_{cut} poniżej wartości $0,020\,\text{kHz}$ prowadzi do utraty stabilności filtru, a przez to całego układu. W efekcie jesteśmy zmuszeni ustalić częstotliwość odcięcia na poziomie $f_{cut} = 0,025\,\text{kHz}$.

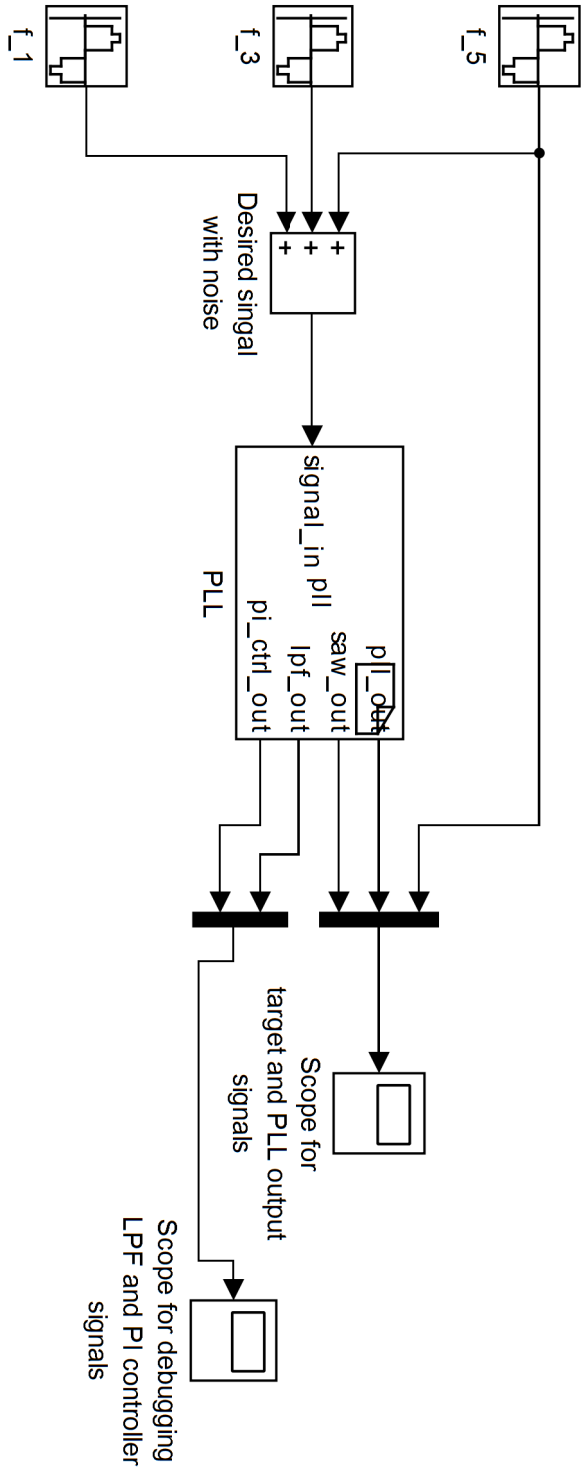
W efekcie wartość sygnału wychodzącego z filtru po początkowej korygacji może być wciąż zbyt wysoka. Idea działania naszego detektora fazy, zakłada, że filtr dolnoprzepustowy odcina wszystkie sygnały poza sygnałami o częstotliwościach bliskich $0\,\text{Hz}$. Poprzez ograniczenie na f_{cut} łamiemy to założenie. Sukcesywnie może to powodować błąd w wartości sygnału wychodzącego z kontrolera PI, opisany w 4.1.

Literatura

[1] http://www.winfilter.20m.com/



Rysunek 1: Schemat prototypu układu PLL zbudowany z niezależnych bloków z pakietów Simulink.



Rysunek 2: Schemat układu PLL zaimplementowanego w C.