Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 21

XV Seminarium
ZASTOSOWANIE KOMPUTERÓW W NAUCE I TECHNICE' 2005
Oddział Gdański PTETiS

ZASTOSOWANIE S-FUNKCJI W SYMULACJI TRÓJPOZIOMWEGO MODULATORA SVPWM W PROGRAMIE SIMULINK

Paweł SZCZEPANKOWSKI[1], Grzegorz GRABOWSKI[2], Janusz NIEZNAŃSKI[3]

1. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki, Katedra Energoelektroniki i Maszyn Elektrycznych, ul. G. Narutowicza 11/12, 80-952 Gdańsk tel: +48 (58) 348-60-76 fax: +48 (58) 341-08-80 e-mail: pszczep@ely.pg.gda.pl
2. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki, Katedra Energoelektroniki i Maszyn Elektrycznych, ul. G. Narutowicza 11/12, 80-952 Gdańsk tel: +48 (58) 348-60-75 fax: +48 (58) 341-08-80 e-mail: ggrab@ely.pg.gda.pl
3. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki, Katedra Energoelektroniki i Maszyn Elektrycznych, ul. G. Narutowicza 11/12, 80-952 Gdańsk tel: +48 (58) 347-16-75 fax: +48 (58) 341-08-80 e-mail: jniez@ely.pg.gda.pl

Blok s-funkcji w programie SIMULINK pozwala na efektywne wykorzystanie narzędzi tego środowiska. S-funkcja napisana w języku C wprowadzona do schematu symulacyjnego jako blok użytkownika jest odpowiednikiem bloku USERa z programu TCAD. Celem artykułu nie jest porównanie programów symulacyjnych, ale wskazanie zalet bloku s-funkcji na przykładzie symulacji działania trójpoziomowego modulatora napięcia. Artykuł, ze względu na swoją ograniczoną objętość nie zawiera szczegółowego opisu działania i budowy falownika wielopoziomowego. Niniejszy artykuł będzie poświęcony zagadnieniu wykorzystania s-funkcji do implementacji trójpoziomowego modulatora napięcia SVPWM i wykorzystaniu go, wraz z elementami z biblioteki SimPowerSystems, w przykładowym schemacie symulacyjnym.

1. WSTĘP

Funkcja, czy algorytm napisany w języku C mogą być szybko zaadoptowane przez inne środowiska symulacyjne czy docelowe, takie jak kompilatory dla procesorów DSP, narzędzia syntezy cyfrowej dla układów PLD. Blok s-funkcji wykonuje się szybciej niż pliki napisane w języku MATLABA i stanowi uzupełnienie SIMULINKA o bloki, których działanie jest w pełni definiowane przez programistę. Pakiet MATLAB, w porównaniu z programem TCAD czy PSPICE stanowi bardziej rozbudowane i bogatsze narzędzie analizy sygnałów dedykowane do badania widma czy filtracji. W kolejnym punkcie zawarto podstawowe informacje dotyczące s-funkcji.

Recenzent: Prof. dr hab. inż. Elżbieta Bogalecka - Wydział Elektrotechniki i Automatyki Politechniki Gdańskiej

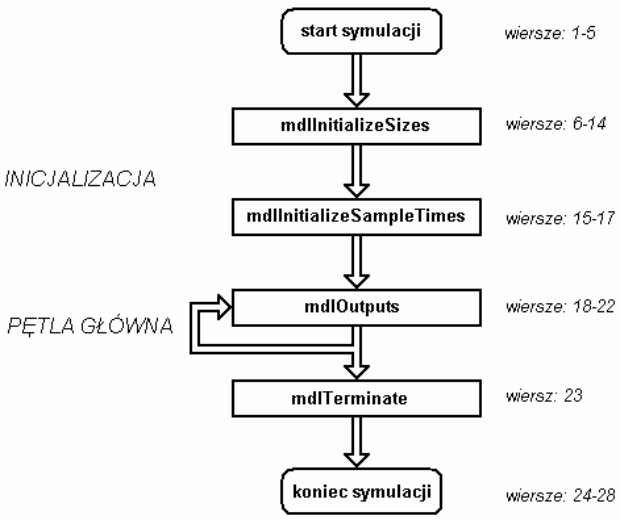
1.1. Podstawy s-funkcji

Schemat symulacyjny w programie Simulink zbudowany jest z połączonych ze sobą bloków wykonawczych, które znajdują się w bibliotekach osadzonych w programie bądź utworzonych przez użytkownika na zasadzie kompozycji kilku bloków podrzędnych w jeden nadrzędny. Przyporządkowany blokowi symbol graficzny posiada określoną liczbę wejść, wyjść oraz parametrów. Działanie bloku może być opisane przy pomocy jednego z języków strukturalnych. Aby umożliwić pracę bloku wykonawczego z s-funkcją należy ją skompilować. Środowisko MATLAB pozwala na wybór kompilatora. S-funkcja musi być napisana zgodnie z szablonem. Dla języka C istnieją dwa szablony – złożony i prosty. Znajdują się one w zasobach pomocy środowiska MATLAB. Są to pliki o nazwach *sfuntmpl_doc.c* oraz *sfuntmpl_basic.c*. Przygotowania kodu źródłowego można dokonać w dowolnym edytorze tekstowym. Przyporządkowanie kompilatora następuje po wywołaniu w linii poleceń programu MATLAB komendy *mex –setup*. Polecenie *mex nazwa_pliku.c* uruchamia kompilację. Blok s-funkcji znajduje się w grupie User-Defined Function w zasobach Simulinka. Działanie opisano w punkcie następnym, w którym blok s-funkcji posiada jedno wejście, jedno wyjście oraz jeden parametr, zaś algorytm programu realizuje funkcję $y(t)=k \cdot x(t)$, gdzie y oznacza wyjście, x wejście, k parametr, t czas. Poniżej zamieszczono kompletny kod wymienionej wcześniej s-funkcji wraz z numeracją wierszy.

```
1  #define S_FUNCTION_NAME  sfunkcja
2  #define S_FUNCTION_LEVEL 2
3  #define LICZBA_PARAMETROW   1
4  #define PARAMETR(S)    ssGetSFcnParam(S, 1)
5  #include "simstruc.h"
6  static void mdlInitializeSizes(SimStruct *S)
7  {   ssSetNumSFcnParams(S, 1);
8     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) { return; }
9     if (!ssSetNumInputPorts(S, 1)) return;
10    ssSetInputPortWidth(S, 0, 1);
11    ssSetInputPortDirectFeedThrough(S, 0, 1);
12    if (!ssSetNumOutputPorts(S,1)) return;
13    ssSetOutputPortWidth(S, 0, 1);
14    ssSetNumSampleTimes(S, 1);}
15 static void mdlInitializeSampleTimes(SimStruct *S)
16 {   ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
17    ssSetOffsetTime(S, 0, 0.0);}
18 static void mdlOutputs(SimStruct *S, int_T tid)
19 {  double                *p_parametr = (double*)mxGetPr(PARAMETR(S));
20    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
21    real_T                *y    = ssGetOutputPortRealSignal(S,0);
22    *y = *p_parametr *(*uPtrs[0]);}
23 static void mdlTerminate(SimStruct *S) {}
24 #ifdef  MATLAB_MEX_FILE
25 #include "simulink.c"
26 #else
27 #include "cg_sfun.h"
28 #endif
```

Algorytm s-funkcji można podzielić na dwa zasadnicze etapy: inicjalizację oraz pętlę główną programu, co ilustruje diagram umieszczony na następnej stronie. Diagram, zawiera również odnośniki do numerów wiersz z powyższego kodu przykładowego programu. Na wstępie symulacji ustala się liczbę parametrów, ilość wejść, wyjść jak również ich

szerokość (mdlInitializeSizes). W dalszym kroku należy w ciele funkcji mdlInitializeSampleTimes zadeklarować wektor czasu: rodzaj czasu, krok i opcjonalnie offset. Zagadnienie to szczegółowo opisane jest w [2].

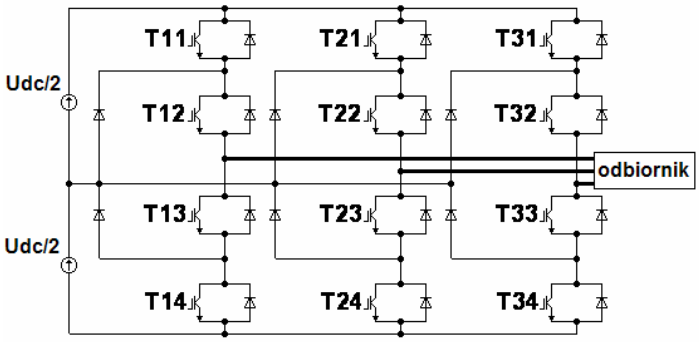


Rys. 1. Przebieg symulacji z wykorzystaniem s-funkcji [1]

W pętli głównej wykonuje się funkcja mdlOutputs. Należy zauważyć, że akceptowana jest tylko wskaźnikowa reprezentacja wyjść. Symulacja modulatora napięcia będzie bazowała na opisanym wcześniej szkielecie przykładowej s-funkcji. Budowę falownika trójpoziomowego oraz podstawowe informacje na temat modulatora zawarto w następnym punkcie.

1.2. Budowa falownika trójpoziomowego NPC

Budowę falownika trójpoziomowego NPC (ang. ***N**eutral-**P**oint-**C**lamped Inverter*) przedstawia rysunek 2. Składa się on z trzech gałęzi, z których każda składa się z czterech łączników energoelektronicznych (na rysunku przedstawione są tranzystory IGBT). Głównym zadaniem falownika jest generowanie odpowiedniego napięcia w celu zasilania podłączonego do jego zacisków wyjściowych odbiornika.



Rys. 2. Budowa falownika trójpoziomowego NPC [3]

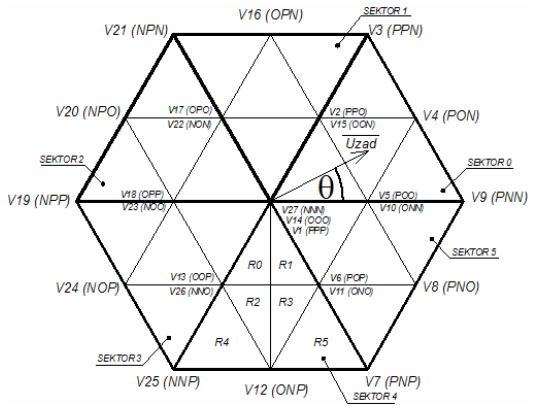
W gałęzi falownika wyróżnia się trzy szablony przełączeń: **P** (tranzystory T11 i T12 są włączone, T13 oraz T14 są wyłączone), **O** (tranzystory T12 i T13 są włączone, pozostałe są wyłączone), **N** (włączone są T13 i T14, pozostałe są wyłączone), gdzie $\boldsymbol{T_{ab}}$ oznacza *b*-ty tranzystor w gałęzi *a.* Modulator napięcia decyduje o wyborze odpowiedniego szablonu załączeń spośród 27 dostępnych. Więcej informacji znajduje się w opracowaniach [4] oraz [5]. W punkcie 1.3 zawarte są tylko podstawowe informacje na temat modulatora.

1.3. Zasada działania modulatora napięcia

W tabeli 1 dokonano zestawienia wszystkich możliwych kombinacji wektorów P, N i O wraz z odpowiadającymi im wartościami napięć fazowych odbiornika oraz szablonami załączeń tranzystorów dla każdej fazy. Relacje przestrzenne między wektorami oraz położenie przykładowego napięcia zadanego *Uzad* ilustruje rysunek 3. Do zadań modulatora należą: wyznaczenie kąta wektora *Uzad*, sektora, w którym znajduje się wektor oraz grupy wektorów bazowych – wektorów najbliżej położonych względem płaszczyzny z rysunku 3 [6]. Następnie, wyznacza się czasy trwania dla poszczególnych wektorów [7]. T1i w tabeli oznacza grupę łączników energoelektronicznych w gałęzi pierwszej, połączonej z fazą 1 obciążenia. Grupę tę stanowią tranzystory T11, T12, T13 oraz T14. Dla pozostałych dwóch gałęzi oznaczenia są analogiczne.

Tabela 1. Napięcia składowe wektorów wyjściowych oraz szablony przełączeń

Oznaczenie wektora	Szablon przełączeń	U1f	U2f	U3f	Szablon przełączeń w gałęzi		
					T1i	T2j	T3k
V1	PPP	0	0	0	1100	1100	1100
V2	PPO	1/6	1/6	-1/3	1100	1100	0110
V3	PPN	1/3	1/3	-2/3	1100	1100	0011
V4	PON	1/2	0	-1/2	1100	0110	0011
V5	POO	1/3	-1/6	1/6	1100	0110	0110
V6	POP	1/6	-1/3	1/6	1100	0110	1100
V7	PNP	-1/3	2/3	-1/3	1100	0011	1100
V8	PNO	1/2	-1/2	0	1100	0011	0110
V9	PNN	2/3	-1/3	-1/3	1100	0011	0011
V10	ONN	1/3	-1/6	-1/6	0110	0011	0011
V11	ONO	1/6	-1/3	1/6	0110	0011	0110
V12	ONP	0	-1/2	1/2	0110	0011	1100
V13	OOP	-1/6	-1/6	1/3	0110	0110	1100
V14	OOO	0	0	0	0110	0110	0110
V15	OON	1/6	1/6	-1/3	0110	0110	0011
V16	OPN	0	1/2	-1/2	0110	1100	0011
V17	OPO	-1/6	1/3	-1/6	0110	1100	0110
V18	OPP	-1/3	1/6	1/6	0110	1100	1100
V19	NPP	-2/3	1/3	1/3	0011	1100	1100
V20	NPO	-1/2	1/2	0	0011	1100	0110
V21	NPN	-1/3	2/3	-1/3	0011	1100	0011
V22	NON	-1/6	1/3	-1/6	0011	0110	0011
V23	NOO	-1/3	1/6	1/6	0011	0110	0110
V24	NOP	-1/2	1/2	0	0011	0110	1100
V25	NNP	-1/3	-1/3	2/3	0011	0011	1100
V26	NNO	-1/6	-1/6	1/3	0011	0011	0110
V27	NNN	0	0	0	0011	0011	0011



Rys. 3. Płaszczyzna wektorów bazowych falownika trójpoziomowego NPC

Aby wyznaczyć czasy trwania poszczególnych wektorów bazowych należy odnaleźć wektory, których położenie jest najbardziej korzystne, z punktu widzenia składania wektora zadanego. Wektory bazowe tworzą ściany trójkątów – rysunek 3. Warunek generacji wektora zadanego ***Vref*** w okresie czasu o długości ***T*** opisany jest równaniem (1) [7].

$$V_{ref} \cdot T = T_1 \cdot V_A + T_2 \cdot V_B + (T - T_1 - T_2) \cdot V_Z \tag{1}$$

gdzie: $\boldsymbol{T_1}$, $\boldsymbol{T_2}$, $\boldsymbol{T}$ – bezwzględne czasy trwania wektorów, $\boldsymbol{V_{ref}}$ – wektora zadany, $\boldsymbol{V_A}$, $\boldsymbol{V_B}$- wektory aktywne, $\boldsymbol{V_Z}$ – wektor zerowy

$$\begin{cases} t_a + t_b + t_c = 1 \\ \mathbf{V}_a \cdot t_a + \mathbf{V}_b \cdot t_b + \mathbf{V}_c \cdot t_a = \mathbf{V}_{ref} \end{cases} \tag{2}$$

gdzie: $\boldsymbol{t_a}$, $\boldsymbol{t_b}$, $\boldsymbol{t_c}$ – względne czasy trwania wektorów bazowych odniesione do $\boldsymbol{T}$
$\boldsymbol{V_a}$, $\boldsymbol{V_b}$, $\boldsymbol{V_c}$ – względne wektory odniesione do wektora $\boldsymbol{V_{ref}}$

Zastępując w równaniu (2) wyrażenia $\boldsymbol{V_a}$, $\boldsymbol{V_b}$, $\boldsymbol{V_c}$ odpowiednio przez (3) – patrz rysunek 3 –

$$\begin{cases} V_a = V_{5,10} = \dfrac{1}{2} \cdot e^{j0} \\ V_b = V_{2,15} = \dfrac{1}{2} \cdot e^{j\frac{\pi}{3}} \\ V_c = 0 \end{cases} \tag{3}$$

po rozkładzie wektorów $\boldsymbol{V_a}$, $\boldsymbol{V_b}$, $\boldsymbol{V_c}$ oraz $\boldsymbol{V_{ref}}$ na współrzędne αβ otrzymuje się:

$$\begin{cases} \dfrac{1}{2} \cdot \cos(0) \cdot t_a + \dfrac{1}{2} \cdot \cos\left(\dfrac{\pi}{3}\right) \cdot t_b + 0 \cdot t_c = V_{ref} \cdot \cos(\theta) \\ \dfrac{1}{2} \cdot \sin(0) \cdot t_a + \dfrac{1}{2} \cdot \sin\left(\dfrac{\pi}{3}\right) \cdot t_b + 0 \cdot t_c = V_{ref} \cdot \sin(\theta) \\ t_a + t_b + t_c = 1 \end{cases} \tag{4}$$

Rozwiązanie jest następujące:

$$\begin{cases} t_a = \frac{4}{\sqrt{3}} \cdot V_{ref} \cdot \sin\left(\frac{\pi}{3} - \theta\right) \\ t_b = \frac{4}{\sqrt{3}} \cdot V_{ref} \cdot \sin(\theta) \\ t_c = 1 - t_a - t_b \end{cases} \tag{5}$$

Układ równań (2) jest równoważny zapisowi macierzowemu (6).

$$\begin{bmatrix} u_\alpha \\ u_\beta \\ 1 \end{bmatrix} \cdot \begin{bmatrix} V_{1\alpha} & V_{2\alpha} & V_{3\alpha} \\ V_{1\beta} & V_{2\beta} & V_{3\beta} \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} t_a \\ t_b \\ t_c \end{bmatrix} \tag{6}$$

gdzie: u_α, u_β - składowe wektora zadanego $\alpha\beta$, $V_{1\alpha}, V_{1\beta}, V_{2\alpha}, V_{2\beta}\ V_{3\alpha}, V_{3\beta}$ – składowe $\alpha\beta$ macierzy odwrotnej dla każdego z trzech wektorów bazowych

Następny punkt opisuje implementację modulatora napięcia SVPWM za pomocą s-funkcji.

2. IMPLEMENTACJA MODULATORA SVPWM

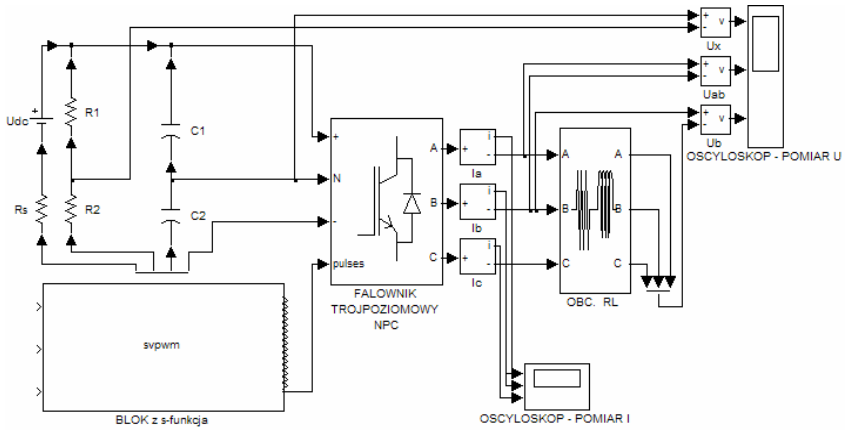
2.1. Omówienie wybranych fragmentów programu

Konstrukcja programu modulatora jest oparta na strukturze przedstawionej w punkcie 1.1. W wierszach 1-4 zdefiniowano nazwę programu, ilość parametrów oraz ich nazwy. Wiersze 5-6 definiują 32-bitowe stałe zapisane w formacie heksadecymalnym. Wartości te kodują stan wszystkich 12 łączników energoelektronicznych w falowniku. Następnie w linii nr 7 zapisano macierz odwrotną potrzebą do rozwiązania równania (6). Wiersz 8 znajduje się deklaracja tablicy szablonów załączeń. Sekcja mdlInitializeSizes obejmuje wiersze 13-27. W sekcji tej zadeklarowano 3 jednokanałowe wejścia bloku, 15 wyjść jednokanałowych oraz jedno wyjście 12-kanałowe. Szerokość ostatniego wyjścia wskazuje, że zostanie ono podłączone do bloku falownika trójpoziomowego dostępnego w bibliotece Power System.W wierszach 28-29 ustalono czas ciągły [2]. Deklaracje funkcji użytkownika – obliczenie sektora, kąta i regionu – zostały zawarte w wierszach 30-32. Główny algorytm lokuje się w wierszach 33-63 – jest to pętla główna programu. Program po wykonaniu pętli głównej nie wykonuje w sekcji mdlTerminate żadnych działań. Przebieg symulacji ilustruje rysunek 1. W dalszej części artykułu opisano pętlę główną.

```
1  #define S_FUNCTION_NAME svpwm
2  #define S_FUNCTION_LEVEL 2
3  #define LICZBA_PARAMETROW   4
4  #define A_PARAM(S)    ssGetSFcnParam(S, 0) …
5  #define V1              0x00000CCC
6  #define V27             0x00000333…
7  const double INV[6][6][3][3] = …
8  int TAB[6][6][7]= …
9  #include "simstruc.h"
10 #include <math.h>
```

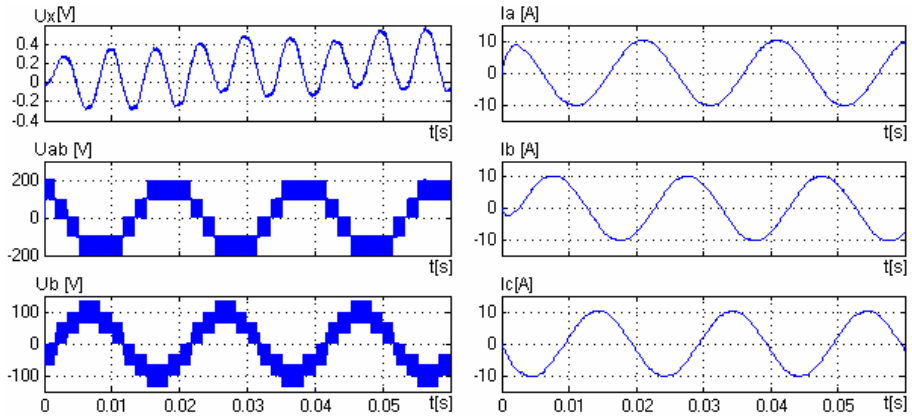
```
11  #include <stdlib.h>
12  #include <string.h>
13  static void mdlInitializeSizes(SimStruct *S){
14  ssSetNumSFcnParams(S,LICZBA_PARAMETROW);
15  if(ssGetNumSFcnPrams(S)!=ssGetSFcnParamsCount(S))
    return;
16  if (!ssSetNumInputPorts(S, 3)) return;
17  ssSetInputPortWidth(S, 0, 1);
18  ssSetInputPortWidth(S, 1, 1);
19  ssSetInputPortWidth(S, 2, 1);
20  ssSetInputPortDirectFeedThrough(S, 0, 1);
21  ssSetInputPortDirectFeedThrough(S, 1, 1);
22  ssSetInputPortDirectFeedThrough(S, 2, 1);
23  if (!ssSetNumOutputPorts(S,16)) return;
24  ssSetOutputPortWidth(S, 0, 1);
25  ssSetOutputPortWidth(S, 1, 1); …
26  ssSetOutputPortWidth(S, 15, 12);
27  ssSetNumSampleTimes(S, 1);}
28  static void mdlInitializeSampleTimes(SimStruct *S)
29  {ssSetSampleTime(S, 0, CONTINOUS_SAMPLE_TIME); ssSetOffsetTime(S, 0, 0.0);}
30  int get_sector(double phase) {…}
31  double get_teta(int s, double kat) {…}
32  int get_region( double t, double Vref, int sector) {…}
33  static void mdlOutputs(SimStruct *S, int_T tid) {
34  InputRealPtrsType in0 = ssGetInputPortRealSignalPtrs(S,0);
35  InputRealPtrsType in1 = ssGetInputPortRealSignalPtrs(S,1);
36  InputRealPtrsType in2 = ssGetInputPortRealSignalPtrs(S,2);
37  real_T          *out0   = ssGetOutputPortRealSignal(S,0);
38  real_T          *out1   = ssGetOutputPortRealSignal(S,1);
39  real_T          *out15   = ssGetOutputPortRealSignal(S,15);
40  int_T           width = ssGetOutputPortWidth(S,15);
41  double  *p_sym = (double*)mxGetPr(A_PARAM(S));
42  double  *p_f   = (double*)mxGetPr(B_PARAM(S));
43  double  *p_fs  = (double*)mxGetPr(C_PARAM(S));
44  double  *p_ma  = (double*)mxGetPr(D_PARAM(S));
45  static double f, fs, ma, sym, tc, Tc, fi, teta, xa, xb, Vref, ta, tb, tz, t0_4, time_vec[7], igbt[12], k;
46  static double s11,s12,s13,s14,s21,s22,s23,s24,s31,s32,s33,s34;
47  f=*p_f;   fs=*p_fs;   ma=*p_ma;   sym=*p_sym;   Tc=1.0/fs;
48  switch ((int)sym){
49  case 0:
50  if (ssGetT(S)/Tc < 1 ) {    tc=0;    }
51  if (ssGetT(S) >= tc ) {
52  i=0;  c=tc+Tc; fi=fmod(2.0*M_PI*f*ssGetT(S),2.0*M_PI);
53  teta=fmod(fi,M_PI/3); s=get_sector(fi); Vef=ma; region=get2_region(teta,Vref);
54  xa = ma*cos(fi); xb = ma*sin(fi);
55  tz=INV[region][s][0][0]*xa+INV[region][s][0][1]*xb+1*INV[region][s][0][2];
56  ta=INV[region][s][1][0]*xa+INV[region][s][1][1]*xb+1*INV[region][s][1][2];
57  tb=INV[region][s][2][0]*xa+INV[region][s][2][1]*xb+1*INV[region][s][2][2];
58  ta=(ta*0.5*Tc); tb=(tb*0.5*Tc); t0_4=0.25*tz*Tc;
59  time_vec[0] =   t0_4;  time_vec[1] =   t0_4+ta;  time_vec[2] =   t0_4+ta+tb;  time_vec[3] =   Tc-t0_4-ta-tb;
    time_vec[4] =  Tc-t0_4-ta; time_vec[5] =  Tc-t0_4;time_vec[6] =  Tc;}
60  if (ssGetT(S) > (tc -Tc  + time_vec[i])){ i++;
61  igbt[0] = (double)(TAB[region][s][i] & 2048);…
    igbt[10]= (double)(TAB[region][s][i] & 2);
    igbt[11]= (double)(TAB[region][s][i] & 1);
62  for (j=0; j<width; j++){ if (igbt[j]>=1) {igbt[j]=1} else igbt[j]=0; *out15++ = igbt[j]; }
63  *out0 =…; *out12 =…; return;}
64  static void mdlTerminate(SimStruct *S) {…}
```

Wiersze 34-36 zawierają definicję wskaźników do trzech wejść jednowymiarowych, zaś w kolejnych trzech znajdują się deklaracje wskaźników do użytych wyjść. W dalszej części znajdują się – wiersze 41-44 – deklaracje wskaźników do czterech parametrów bloku: rodzaju symulacji, częstotliwości zadanego napięcia, częstotliwości modulacji i współczynnika głębokości modulacji. Pierwszy z parametrów jest argumentem funkcji wyboru *switch*, pozwala to na wybór strategii modulacji. Powyżej przedstawiono część kodu programu dla przyjętej strategii sterowania. Więcej informacji na ten temat można odnaleźć w [6]. Dla zadanego wektora napięcia *Uzad* funkcje użytkownika obliczają kąt, sektor oraz region (wiersz 53) [4]. W celu wyznaczenia czasu trwania wektorów aktywnych i zerowego (V27, V14, V1 z rysunku 3) należy wyznaczyć składowe: poziomą α i pionową β. Wyznaczone wartości podstawia się do równań w wierszach 55-57.



Rys. 4. Schemat symulacyjny falownika trójpoziomowego NPC w programie SIMULINK (DSP – blok s-funkcji)

2.2. Wyniki symulacji



Rys. 5. Wyniki symulacji – oznaczenia jak na rysunku 4

W analizowanym przykładzie szablon załączeń składa się z 7 stanów, przy czym jako stan należy rozumieć stan łączników energoelektronicznych odpowiadający określonemu wektorowi z płaszczyzny na rysunku 3. Szablony załączeń zdefiniowane są w tabeli, w wierszu nr 8. Określony szablon siedmiu załączanych wektorów (na przykład w sektorze 0, regionie 0 - V14/V5/V2/V1/V2/V5/V14) zależy od aktualnego sektora i regionu, natomiast czasy trwania każdego wektora z szablonu zależą od składowych wektora napięcia zadanego. Wiersz 61 zawiera grupę 12 komparatorów czasowych przyporządkowanych każdemu tranzystorowi w falowniku. W kolejnych następuje aktualizacja wyjść.

4. PODSUMOWNIE

Algorytm modulatora został zaimplementowany w jednym, zwartym bloku z s-funkcją. Dzięki temu schematy symulacyjne w środowisku MATLAB zawierające nietypowe obiekty zyskują na przejrzystości. Użycie s-funkcji napisanej w języku C pozwala na łatwe przeniesienie algorytmu do innych środowisk symulacyjnych, jak również zastosowanie go w programie przeznaczonym dla procesora czy komputera.

5. BIBLIOGRAFIA

1. Pomoc programu MATLAB: Example of a Basic C MEX S-Function
2. Pomoc programu MATLAB: Sample Time
3. A. Nabae, H. Akagi, I. Takahashi: A New NPC PWM Inverter, IDEE Transaction on Industry Application, Vol. IA-17, NO. 5, September/October 1981
4. Nowacki Z.: *Modulacja szerokości impulsów w napędach przekształtnikowych prądu przemiennego*, PWN, Warszawa, 1991
5. Nowak M., Barlik R.: *Poradnik inżyniera energoelektronika*, WNT, Warszawa 1998
6. Keith A. Corzine, J. R. Baker: Multilevel Voltage-Source Duty-Cycle Modulation: Analysis and Implementation, IEEE Transactions on Industrial Electronics, VOL. 49, NO. 5, october 2002
7. Nikola Celanovic, Dushan Boroyevich: A Comprehensive Study of Neutral-Point Voltage Balancing Problem in Three-level NPC VS PWM Inverter*s*, IEEE Transactions on Power Electronics, VOL. 15, NO. 2, March 2000
8. J. Iglesias, A. Agudo, M. Lafoz: *Analysis and Simulation of Three-Level Voltage Source adn Its Application to Flywheel Energy Storage Systems*, EPE 99

APPLICATION OF S-FUNCTIONS FOR THE SIMULATION OF THREE-LEVEL SVPWM MODULATOR IN SIMULINK

The SIMULINK block descriptions in the form of S-functions permit very effective utilisation of the rich functionality offered by the SIMULINK environment. An S-function can be written in the C programming language and included in the SIMULINK model as a user-defined block, in which case it can be considered a close counterpart of the USER block in TCad. This paper outlines and illustrates the usefulness of the S-functions by example simulations of the three-level inverter. The modulator block is implemented as an S-function and incorporated into the overall inverter model composed of the SimPowerSystems library blocks. Due to space limitations, the paper does not provide a detailed description of the three-level inverter.