DIE SIMULATION DER KOMMUTIERUNGSVORGÄNGE EINES MITTELFREQUENZWECHSELRICHTERS AUF DIGITALRECHNER

Von

A. KÁRPÁTI und T. KOVÁCS

Lehrstuhl für Automatisierung, Technische Universität Budapest

(Eingegangen am 4. Mai, 1971)

Vorgelegt von Prof. Dr. CSÁKI

1. Einführung

Die Schaltung in Abb. 1—1 ist das Schaltbild eines Wechselrichters der den Antriebsmotor eines hochdrehzahligen Antriebes speist. Die Frequenz ist zwischen 0 und 2 kHz stetig veränderbar. In dieser Arbeit wird darauf nicht hingewiesen, warum für die Speisung des Antriebs eine verhältnismäßig komplizierte Schaltung gewählt werden mußte, sondern es werden bloß die Kommutationsvorgänge des Wechselrichters betrachtet. Als Ergebnis der Untersuchung wird ein Rechenprogramm angegeben, das die zeitliche Änderung der Spannung und des Stromes des Kommutierungskondensators sowie die zeitliche Änderung des Stromes des Motor- und Hilfkreises während der Kommutation berechnet.

1.1. *Vereinfachende Voraussetzungen*

a) Da sich während der Kommutation die Spannung des Kondensators im Motorkreis nich wesentlicht ändert, wird angenommen, daß $u_{ct} = U_M =$ $=$ konst. ist.

b) Ebenso ist $u_{cs} = U_s =$ konstant.

c) Alle Dioden und Thyristoren in den Stromkreisen sind verlustlos. Die Thyristoren sperren sofort unter Einwirkung einer Anoden—Katoden—Sperrspannung, ohne Rückstrom und Trägheitseffekte.

d) Die Zeitdauer der Halbperiode ist länger als die Umladezeit des Kommutierungskondensators.

e) Die Thyristoren werden durch einen einzigen kurzen Impuls gezündet.

2. Grundlagen für den Aufbau des Programms

Die Funktionsweise des Wechselrichters wird in stationärem Zustand untersucht. Das bedeutet, daß sich die einzelnen Arbeitsphasen periodisch wiederholen und am Ende jeder Halbperiode die Werte von einigen Größen den Anfangswerten — abgesehen vom Vorzeichen — gleich sind.

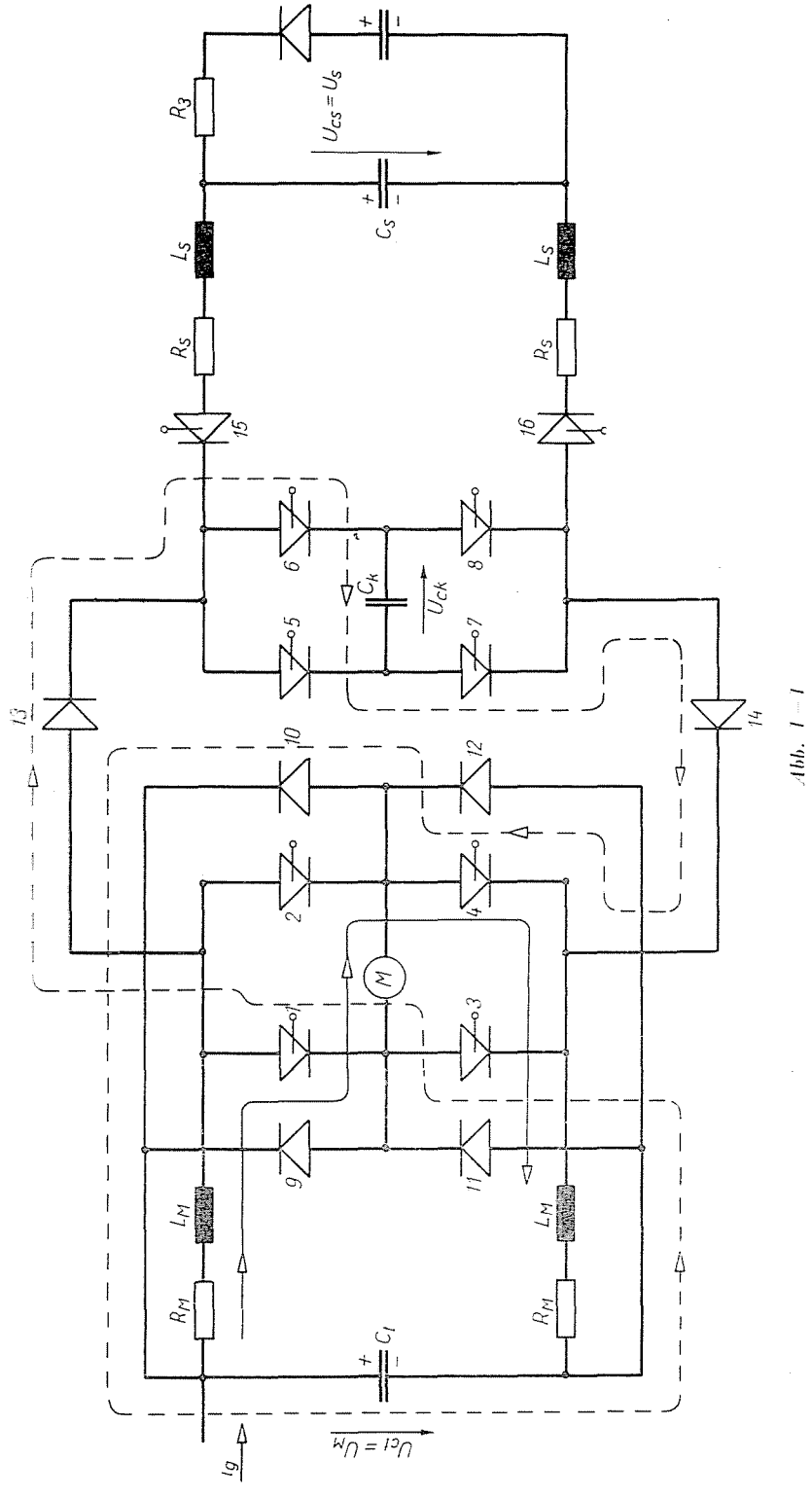
R_3
$U_{CS} = U_S$
C_S
L_S
R_S
15
16
6
8
C_k
U_{Ck}
5
7
13
14
10
12
2
4
M
1
3
9
11
L_M
R_M
C_1
$U_{C1} = U_M$
1g

Abb. 1 — 1

Es soll die Halbperiode untersucht werden, vor der der Strom durch die Thyristoren 1—4 geführt wurde. Am Anfang dieser Halbperiode schalten die Löschthyristoren 6—7, dann mit einer Verzögerung t_s die Hilfsthyristoren 15—16 ein. Schließlich schalten mit einer Verzögerung t_f die Hauptthyristoren 2—3 durch, und führen Strom bis zum Ende der untersuchten Halbperiode, d. h. bis zur folgenden Löschung.

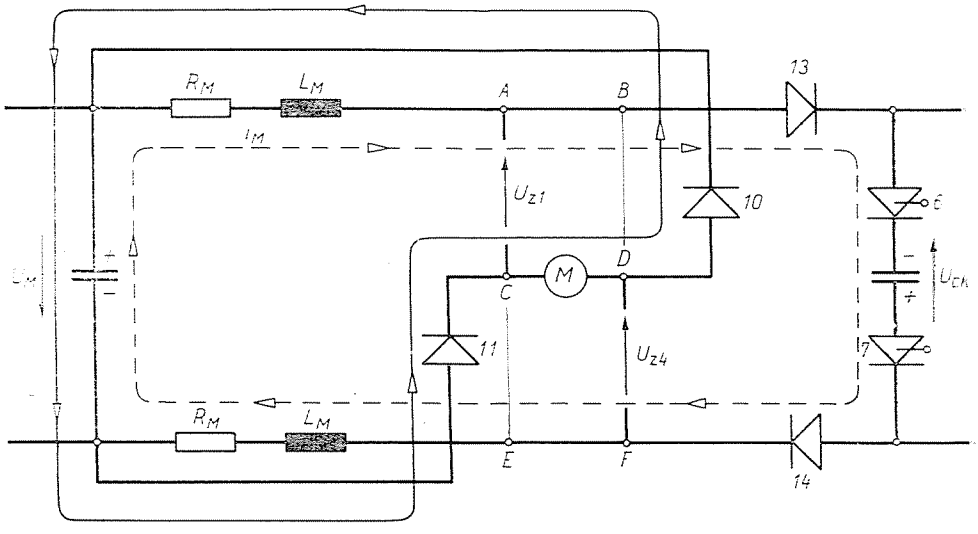

Abb. 2—1

Auf Grund von Abb. 1—1 und des Gesagten folgt, daß während der Kommutation der Kommutierungskondensator C_k in dreifacher Weise geladen werden kann.

Den Fall *a* zeigt Abb. 2—1. Das hier gültige Gleichungssystem lautet:

$$i'_M = (U_M + u_{ck})/2L_M - i_M R_M/L_M$$
$$u'_{ck} = -i_M/C_k \qquad (2-1)$$

Die Variante *b* ist in Abb. 2—2 zu sehen. Das Gleichungssystem lautet in diesem Falle:

$$i'_M = (U_M + u_{ck})/2L_M - i_M R_M/L_M$$
$$u'_{ck} = -(i_M + i_s)/C_k \qquad (2-2)$$
$$i'_s = (U_s + u_{ck})/2L_s - i_s R_s/L_s$$

Die Variante *c* ist in Abb. 2—3 dargestellt. Das Gleichungssystem lautet in diesem Falle:

$$i'_s = (U_s + u_{ck})/2L_s - i_s R_s/L_s$$
$$u'_{ck} = -i_c/C_k \qquad (2-3)$$

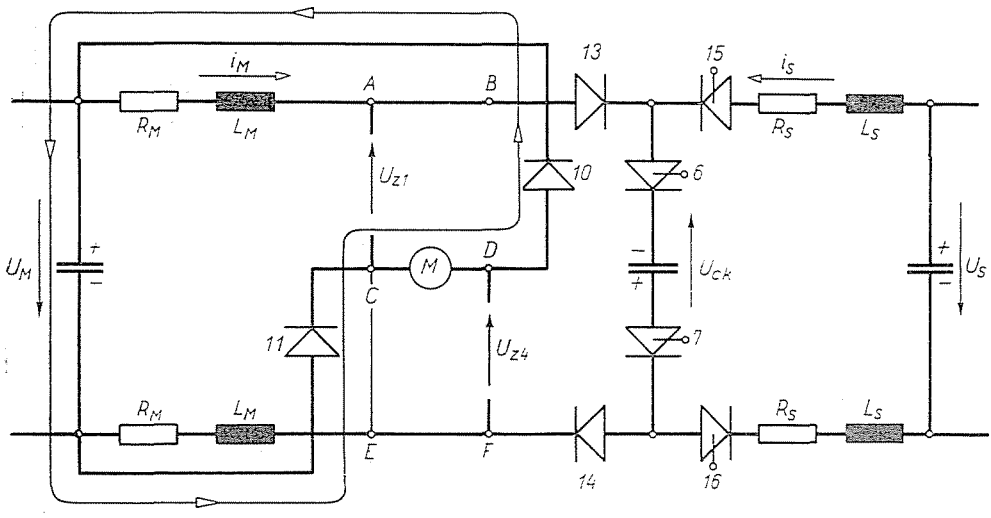

Abb. 2—2

Der stationäre Zustand kann nach zwei Methoden gesucht werden. Nach der einen Methode wird das gesamte digitale Modell des Wechselrichters hergestellt. Ausgehend von den Anfangswerten Null wird die Berechnung über so viele Perioden durchgeführt, bis der stationäre Zustand erreicht ist.

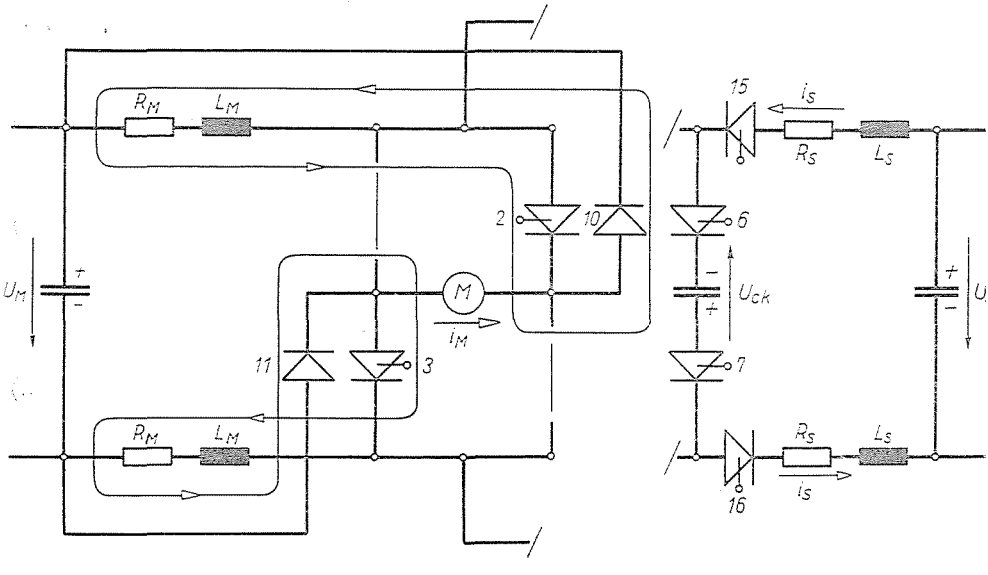

Abb. 2—3

Nach der anderen, einfacheren Methode wird nur der Kommutationskreis simuliert. Es werden die im stationären Zustand auftretenden Werte abgeschätzt und die Berechnung beginnt mit diesen Werten. Das Endergebnis von vorgeschriebener Genauigkeit wird durch Iteration bestimmt.

3. Das Blockschaltbild des Programms

Die Glieder des Blockschaltbilds des Programms sind mit Nummern versehen, die auf den Erklärungstext hinweisen. Das Blockschaltbild ist in Abb. 3—1 zu sehen.

Die Iteration werden mit Hilfe des Befehls

$$u_{ck0i+1} = u_{ck0i} \tag{3—1}$$

durchgeführt.

Die Deutung der einzelnen Glieder der Blockschaltbildes:

1. Es wird kontrolliert, ob die Löschthyristoren 6—7 eingeschaltet werden können.

2. Nach dem Einschalten der Löschthyristoren 6—7 im Zeitpunkt $t = 0$ bildet sich der in Abb. 2—1 angegebene Stromkreis und der Kondensator C_k wird gemäß Gleichungssystem *a* geladen.

3. Es wird untersucht, ob im Zeitintervall $0 - t_s$ der Strom i_M den Wert Null erreicht.

3.1. Im Bejahungsfall können die Hilfsthyristoren nicht einschalten, der Umladevorgang wird unterbrochen, der Wert der Spannung u_{ck} bleibt $u_{ck}(t_1)$.

4. Erreicht i_M im Intervall $0 - t_s$ den Wert Null nicht, so muß geprüft werden, ob die Hilfsthyristoren 15—16 einschaltbar sind.

4.1. Wenn dies nicht der Fall ist, dann wird der Kondensator C_k im Stromkreis gemäß Abb. 2—1 weitergeladen, Gleichungssystem *a*, bis der Strom im Zeitpunkt t_0 gleich null wird.

5. Sind die Hilfsthyristoren 15—16 einschaltbar, wird der Kondensator C_k im Stromkreis gemäß Abb. 2—2 geladen Gleichungssystem *b*.

6. Erreichen im Zeitintervall $t_s - t_f$ weder i_M noch i_s den Wert Null, bleibt das Gleichungssystem bis zum Zeitpunkt t_f gültig.

7. In diesem Falle ist im Zeitpunkt t_f zu prüfen, ob die Hauptthyristoren 2—3 eingeschaltet werden können. Die Bedingung 7 läßt sich auf Grund von Abb. 3—2 beweisen.

8. Können die Hauptthyristoren 2—3 eingeschaltet werden, wird gemäß Abb. 2—3 das Gleichungssystem solange gültig sein, bis der Strom i_s den Wert Null erreicht hat und die Aufladung des Kondensators C_k beendet ist.

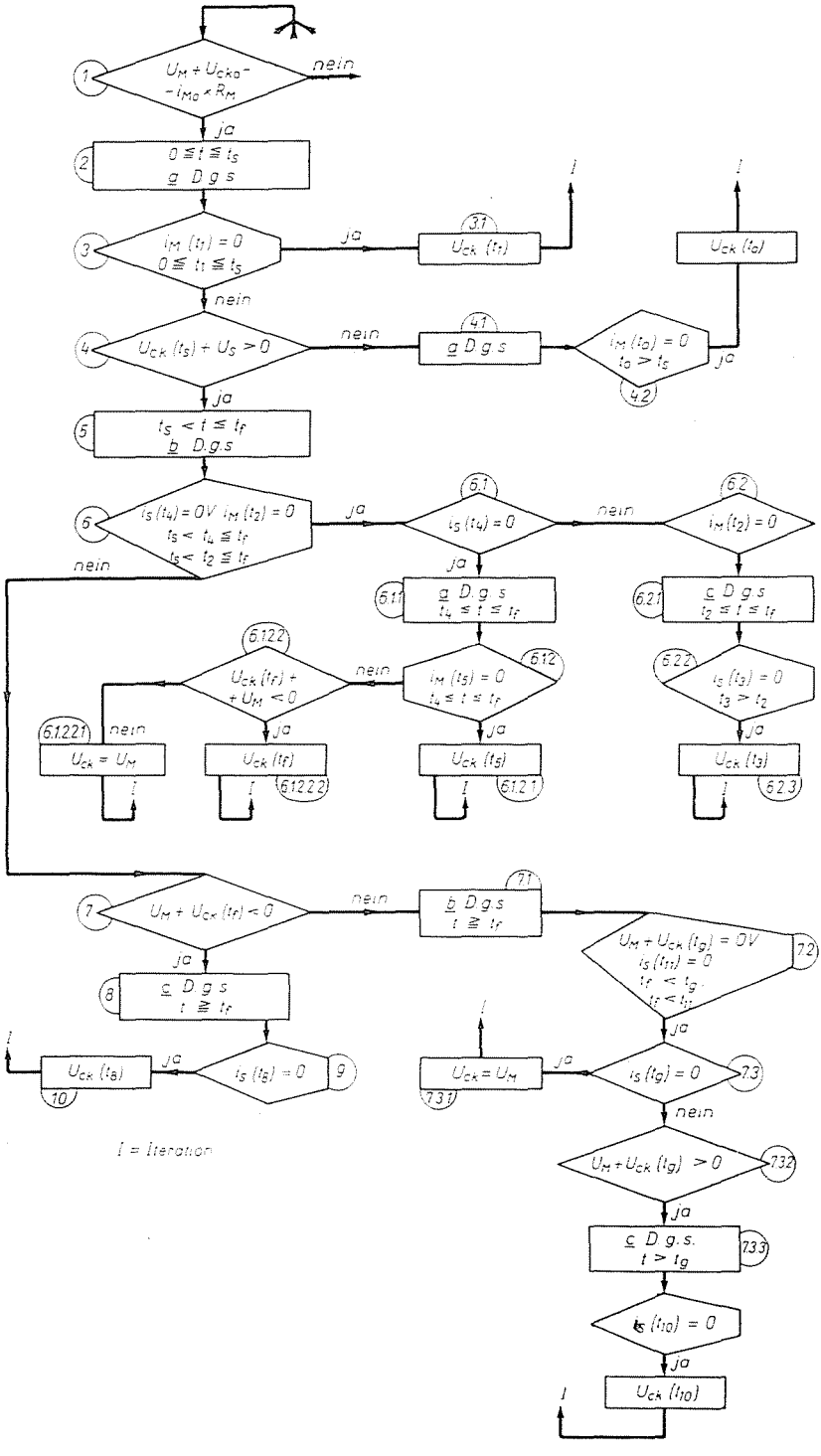
1
$U_M + U_{ck0} - i_{M0} \cdot R_M$
nein
ja
2
$0 \leqq t \leqq t_S$
a D g s
3
$i_M(t_1) = 0$
$0 \leqq t_1 \leqq t_S$
ja
3.1
$U_{ck}(t_1)$
I
nein
4
$U_{ck}(t_S) + U_S > 0$
nein
4.1
a D g s
$i_M(t_0) = 0$
$t_0 > t_S$
4.2
ja
$U_{ck}(t_0)$
I
5
$t_S < t \leqq t_f$
b D.g.s
6
$i_S(t_4) = 0V$ $i_M(t_2) = 0$
$t_S < t_4 \leqq t_f$
$t_S < t_2 \leqq t_f$
ja
6.1
$i_S(t_4) = 0$
nein
6.2
$i_M(t_2) = 0$
ja
6.1.1
a D g s
$t_4 \leqq t \leqq t_f$
6.2.1
c D g s
$t_2 \leqq t \leqq t_f$
6.1.2.2
$U_{ck}(t_f) + U_M < 0$
nein
6.1.2
$i_M(t_5) = 0$
$t_4 \leqq t \leqq t_f$
6.2.2
$i_S(t_3) = 0$
$t_3 > t_2$
6.1.2.2.1
$U_{ck} = U_M$
6.1.2.2.2
$U_{ck}(t_f)$
6.1.2.1
$U_{ck}(t_5)$
6.2.3
$U_{ck}(t_3)$
I
7
$U_M + U_{ck}(t_f) < 0$
nein
7.1
b D.g.s
$t \geqq t_f$
ja
8
c D g s
$t \geqq t_f$
7.2
$U_M + U_{ck}(t_g) = 0V$
$i_S(t_{11}) = 0$
$t_f < t_g$
$t_f < t_{11}$
ja
10
$U_{ck}(t_8)$
ja
9
$i_S(t_8) = 0$
7.3.1
$U_{ck} = U_M$
ja
7.3
$i_S(t_g) = 0$
nein
7.3.2
$U_M + U_{ck}(t_g) > 0$
ja
7.3.3
c D g.s.
$t > t_g$
$i_S(t_{10}) = 0$
ja
$U_{ck}(t_{10})$
I = Iteration

Abb. 3 – 1

In dieser Erklärung sind die Nebenzweige des Programms nicht berücksichtigt. Ihre Notwendigkeit kann aber auf Grund der Abbildungen und der vorherigen Überlegungen leicht nachgewiesen werden.

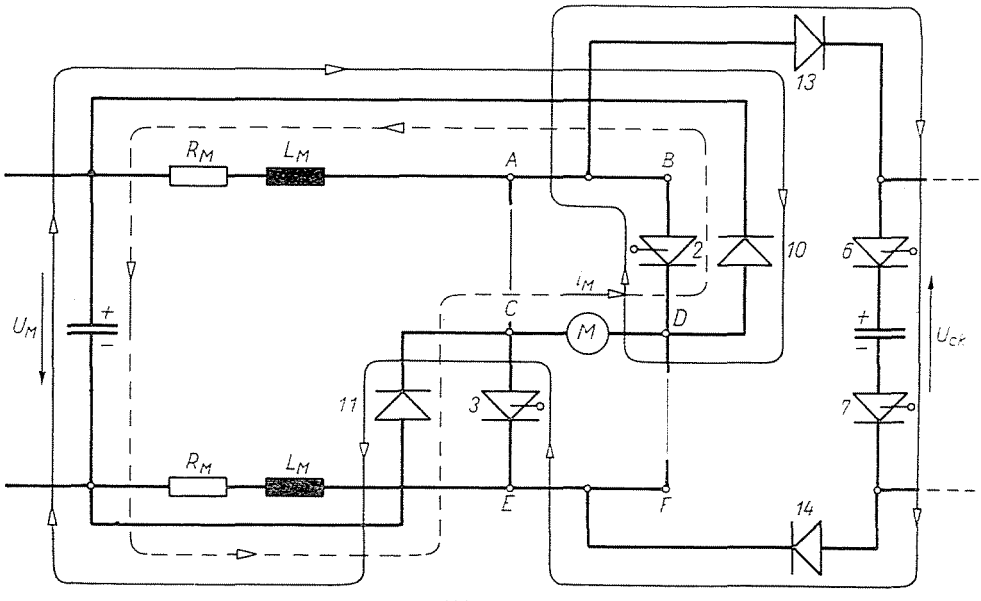

Abb. 3—2

4. Die Beschreibung des Programms

Das im folgenden angegebene Programm wurde in der Praxis mit gutem Ergebnis ausprobiert und simuliert den mit den Ziffern 1, 2 . . . 10 bezeichneten Hauptzweig. Weicht der Programmablauf auf irgendeinen Nebenzweig aus, erfolgt eine Meldung und der weitere Ablauf wird gestoppt.

Das Programm löst verschiedene Differentialgleichungssysteme in dem die End- und Anfangswerte miteinander verkettet werden. Für die Lösung der Differentialgleichungssysteme wurde das Verfahren von Kutta—Merson gewählt, weil es geeignet ist, durch die Absetzung des Fehlers den Schrittabstand automatisch zu ändern.

Der Programmaufbau ist im folgenden dargestellt:

```
begin real t, ts, tf, imo, ucko, uckol, um, us, rm, lm, lm2, fm, ck, rs, ls, ls2, fs,
          h, hl, ta, tb, a1, a2, a3, b1, b2, b3, d, error2, error3;
    integer n, nf, ns, k, j, pa, pb, pc, l, q2, j1;
    array y0[1:3], x[1:20, 1:3], tx[1:20], eps[1:3], eps1[1:3];
    boolean first, abc, ac;
procedure group1;
    begin hl: = (tb — ta)×0.5;
```

```
            y0[1]: = a1; y0[2]: = a2; y0[3]: = a3;
      end;
procedure group2;
      begin a1: = y0[1]; a2: = y0[2]; a3: = y0[3];
            ta: = t;
      end;
procedure group3;
      begin b1: = y0[1]; b2: = y0[2]; b3: = y0[3];
            tb: = t;
      end;
procedure fcta(t, fi, f);
      real array fi, f; real t;
      begin f[1]: = (um + fi[2])/lm2 — fi[1]×fm;
            f[2]: = —fi[1]/C_k;
      end;
procedure fctb(t, fi, f);
      real array fi, f; real t;
      begin f[1]: = (um + fi[2])/lm2 — fi[1]×fm;
            f[2]: = —(fi[1] + fi[3])/C_k;
            f[3]: = (us + fi[2])/ls2 — fi[3]×fs;
      end;
procedure fctc(t, fi, f);
      real array fi, f; real t;
      begin f[1]: = (us + fi[2])/ls2 — fi[1]×fs;
            f[2]: = —fi[1]/C_k;
      end;
procedure merson(t, eps, h, k, fct, p);
      integer p; real t, h, k;
      procedure fct; array eps;
      begin integer i, loc, l, q1;
      real error, t0, hc;
      array y1, y2, f0, f1, f2, 1:n;
      boolean increase;
      if first then begin p: = 1; first: = false; end;
      t0: = 1; q: = 1; loc: = 0;
next: if loc + q < p then q1: = q else q1: = p — loc;
      hc: = h×q1/p;
      fct(t, y0, f0),
      for i: = 1 step 1 until n do y1[i]: = y0[i] + hc/3×f0[i];
      fct(t + hc/3, y1, f1);
      for i: = 1 step 1 until n do y1[i]: = y0[i] + hc/6×f0[i] +
                                           hc/6×f1[i];
```

```
fct(t + hc/3, y1, f1);
for i: = 1 step 1 until n do y1[i]: = y0[i] + hc/8×f0[i] +
                                     3×hc/8×f1[i];
fct(t + hc/2, y1, f2);
for i: = 1 step 1 until n do y1[i]: = y0[i] + hc/2×f0[i] —
                                     3×hc/2×f1[i] + 2×hc×f2[i];
fct(t + hc, y1, f1);
for i: = 1 step 1 until n do y2[i]: = y0[i] + hc/6×f0[i] +
                                     2×hc/3×f2[i] + hc/6×f1[i];
increase: = true;
for i: = 1 step 1 until n do
  begin error: = abs(0.2×(y1[i] — y2[i])):
    if error > eps[i]×q1/p then
      begin if q ≠ 1 then q: = q + 2
                     else begin p: = 2×p; loc: = loc×2
                          end;
            goto next:
      end;
if error×64 > eps[i]×q1/p then increase: = false;
  end;
for i: = 1 step until n do y0[i]: = y2[i];
loc: = loc + q1; t: = t0 + h×loc/p;
if increase ∧ q1 = q then q: = 2×q;
if loc ≠ p then goto next else
  begin k: = h/p; if q < p then p: = 0 + q
                           else p: = 1;
  end;
end kutta-merson:
procedure itel(cimke, c, d);
  real c, d; label cimke;
  begin integer i;
  isml: groupl;
        for i: = 1 step 1 until 3 do epsl[i]: = eps[i]×hl/h;
        if abc then goto bb else if ac then goto aa else goto cc;
  aa: merson(ta, epsl, hl, k, fcta, pa);
      goto folyt;
  bb: merson(ta, epsl, hl, k, fctb, pb);
      goto folyt;
 cc: merson(ta, epsl, hl, k, fctc, pc);
  folyt: if c ≥ d then group2 else group3;
        if tb — ta < error2 then goto cimke else goto isml;
  end itel;
```

```
  read(n, ts, ns, tf, nf, eps1, eps2, eps3, error2, error3,
       um, rm, lm, us, ls, rs, ck, imo, ucko);
l: = 0;
first: = true; lm2: = 2×lm; fm: = rm/lm;
ls2: = 2×ls; fs: = rs/ls;
anfang:t: = 0; y0[1]: = imo; y0[2]: = ucko; yo[3]: = 0;
if um + ucko — imoxrm < 0 then
  begin q2: = 1; goto all;
  end;
h: = ts/ns;
for j: = 1 step 1 until ns do
  begin merson(t, eps, h, k, feta, pa);
    x[j, 1]: = y0[1]; x[j, 2]: = y0[2]; x[j, 3]: = 0;
    tx[j]: = t;
  end;
if y0[1] ≤ 0 then begin q2: = 2; goto all; end;
if y0[2] + us ≤ 0 then begin q2: = 3; goto all; end;
h: = (tf — ts)/nf; t: = ts; pb: = pa; n: = 3;
for j: = ns + 1 step 1 until nf + ns do
  begin merson(t, eps, h, k, fetb, pb);
    x[j, 1]: = y0[1]; x[j, 2]: = y0[2]; x[j, 3]: = y0[3]; tx[j]: = t;
  end;
if y0[1] ≤ 0 ∨ y0[3] ≤ 0 then begin q2: = 4; goto all;
                                end;
if um + y0[2] ≥ 0 then begin q2: = 5; goto all;
                        end;
y0[1]: = y0[3]; t: = tf; pc: = pb; h: = ts/ns;
n: = 2; j: = nf + ns;
for j: = j + 1 while yp[1] ≥ 0 do
  begin merson(t, eps, h, k, fetc, pc);
    x[j, 1]: = y0[1]; x[j, 2]: = y0[2]; x[j, 3]: = 0; tx[j]: = t; j1: = j;
  end;
abc: = false; ac: = false; d: = 0; group3;
ta: = tx[j1 — 1]; a1: = x[j1 — 1, 1]; a2: = x[j1 — 1, 2]; a3: = 0;
print(a3);
itel(ismet, y0[1], d);
ismet: uckol: = abs[y0[2]);
print(a3);
if abs(uckol — ucko) < error3 then goto veg
                              else
  begin ucko: = uckol; l: = l + 1;
        print(ucko, l); goto anfang;
```

```
  end;
all:print(q2); stop;
veg:print(x, tx, uckol, l);
end;
```

Die durch die Berechnung erhaltene Ergebnisse sind in Abb. 4—1 gezeigt. Die Ergebnisse stimmen mit guter Näherung mit dem Meßwerten überein.

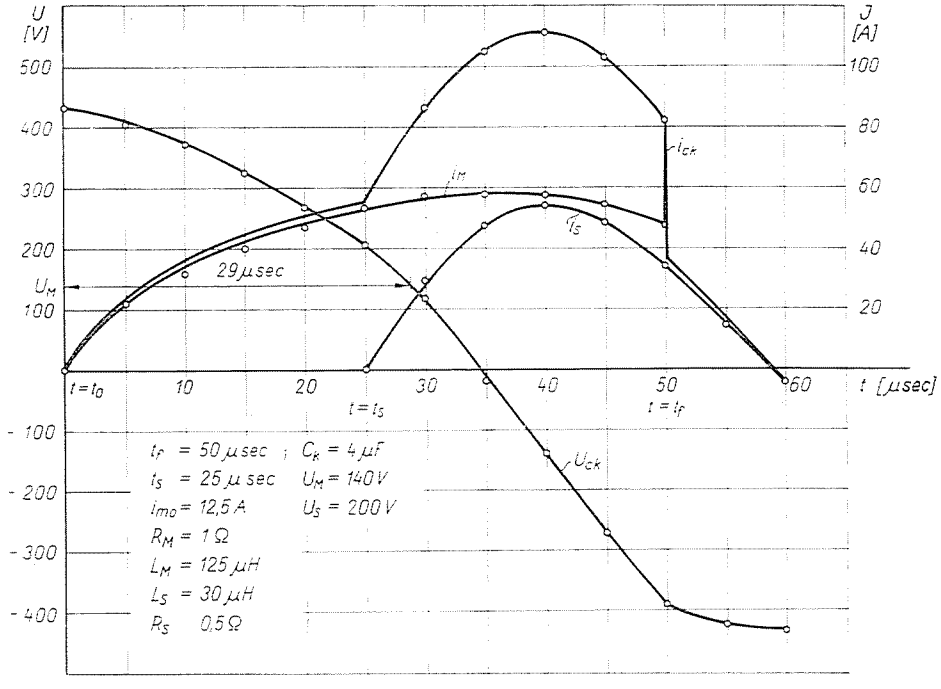

Abb. 4—1

5. Folgerungen

Auf Grund des Gesagten kann festgestellt werden, daß das für die Lösung von Differentialgleichungssystemen n-ter Ordnung geeignete Kutta—Merson Verfahren mit automatischer Schrittabstandsänderung für die Untersuchung der Betriebsverhältnisse von komplizierten Wechselrichterschaltungen vorteilhaft angewandt werden kann. Es ermöglicht nämlich, auf Grund einer allgemeinen Formel, die Integration von Differentialgleichungssystemen verschiedener Ordnungen, die durch ihre End- und Anfangswerte miteinander verkettet sind und die Arbeitsweise der Schaltung beschreiben.

Das Program simuliert die Kommutierungsvorgänge mit hinreichender Genauigkeit, so daß mit seiner Hilfe die Einwirkungen der einzelnen Wechselrichterparameter auf die Kommutation abgeschätzt werden können.

Die Aufgabe könnte auf einem Analogrechner wegen der großen Anzahl der logischen Bedingungen, die zu berücksichtigen sind, — wofür der Analogrechner wenig geeignet ist — nur durch den Aufbau von zusätzlichen logischen Kreisen gelöst werden.

Zusammenfassung

Die Arbeit behandelt die Simulation des Betriebs einer Mittelfrequenz-Wechselrichter-Schaltung. Es wird das Blockschaltbild der Simulation einer ausgeführten Schaltung angegeben, sodann wird die Lösung der Aufgabe mit Hilfe des Kutta-Merson-Verfahrens gezeigt.

Literatur

1. Lőcs, Gy.: Algol 60 programozási nyelv. Műszaki Könyvkiadó, Budapest, 1970 (in ungarischer Sprache).
2. Csáki, F.—Ipsits, I.—Barki, K.—Ganszky, K.: Ipari elektronika. Tankönyvkiadó, Budapest, 1966 (in ungarischer Sprache).
3. Parnaby, J.: Digital computing techniques. Part 2. Control, August, 1968.

Attila Kárpáti
Tivadar Kovács
Budapest XI., Garami E. tér 3., Ungarn