Programe de aplicații

UN MODEL DE PROIECTARE ȘI OPTIMIZARE A CIRCUITELOR DE COMUTAȚIE

Conf.dr.Moise Cocan
Asistent Dorin Bocu

Universitatea din Brașov

În lucrarea de față ne propunem să prezentăm o modalitate de proiectare a circuitelor de comutație precum și optimizarea acestora.

Vom prezenta în acest sens un algoritm care realizează proiectarea și optimizarea unui circuit de comutație utilizînd structurile Reed-Muller, deci făcînd apel la operațiile modulo-2.

Pornim de la ideea că cerințele de bază ale oricărei structuri capabile să permită descrierea și optimizarea circuitelor de comutație sînt următoarele:

(i) completa funcționalitate, care constă în posibilitatea de reprezentare algebrică distinctă a fiecărei funcții booleene de un număr dat de variabile;
(ii) flexibilitatea care presupune maleabilitate în funcționare astfel ca algoritmii design-ului circuitelor să poată fi realizați și utilizați cu ușurință;
(iii) posibilitatea de realizare care presupune faptul că operațiile de bază au corespondențe fizice în circuitele de comutație.

Toate aceste cerințe sînt satisfăcute de algebra modulo-2 și chiar pot fi extinse pentru structurile GF(q), cu q-prim, sau $q=p^k$ cu p-prim, $k\in N$.

Ne propunem, deci, să realizăm sinteza unui circuit de comutație, care să realizeze o funcție booleană dată. Presupunem că se cunoaște funcția booleană și, implicit, și forma sa normală disjunctivă:

$$f(x_1, x_2, \dots, x_n) = \sum_{i=0}^{2^n-1} d_i m_i, \qquad (1)$$

unde m_i sînt minitermenii funcției f, deci au forma:

$m_i = x_1^{\alpha_{i,1}}, x_2^{\alpha_{i,2}}, \dots, x_n^{\alpha_{i,n}}$, $\alpha_{i,j} \in \{0,1\}$, $x^0 = \bar{x}$, $x^1 = x$

iar d_i sînt valorile funcției f:

$f(\alpha_{i,1},\alpha_{i,2},\dots,\alpha_{i,n}) = d_i$ unde $<\alpha_{i,1},\alpha_{i,2},\dots,\alpha_{i,n}> 2 \le i > 10$

Exemple

(i) **Cazul n=2.**

$f(x_2,x_1) = d_0\bar{x}_1\bar{x}_2 + d_1\bar{x}_1x_2 + d_2x_1\bar{x}_2 + d_3x_1x_2$.

(ii) **Cazul n=3**

$$f(x_3,x_2,x_1) = d_0\bar{x}_1\bar{x}_2\bar{x}_3 + d_1\bar{x}_1\bar{x}_2x_3 + d_2\bar{x}_1x_2\bar{x}_3 + d_3\bar{x}_1x_2x_3 + d_4x_1\bar{x}_2\bar{x}_3 + d_5x_1\bar{x}_2x_3 + d_6x_1x_2\bar{x}_3 + d_7x_1x_2x_3$$

Observație. Fără restrîngerea generalității se poate presupune că funcția booleană f este simplificată, adică expresia booleană este expresie minimală, în caz contrar aplicîndu-se algoritmul Quine-McCluskey.

Teorema 1. Oricărei expresii booleeene de forma (1) îi corespunde o expresie Reed-Muller de forma (2).

$$f(x_n, x_{n-1}, \dots, x_1) = \sum_{i=0}^{2^n-1} c_i\pi_i \qquad (2)$$

Demonstrație. Menționăm faptul că în expresia booleană suma și produsul sînt, respectiv, suma booleană și produsul boolean, în timp ce în expresia Reed-Muller acestea sînt suma, respectiv produsul modulo-2.

Se procedează prin inducție după n (n fiind numărul variabilelor).

Pentru n=2 avem:

$$f(x_2, x_1) = \sum_{i=0}^{3} d_i m_i = \sum_{i=0}^{3} c_i\pi_i$$

unde

$c = T_2 \bullet d$

$$c = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}, T_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, d = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

Se remarcă faptul că $T_2 = T_1 * T_1$, unde "*" reprezintă produsul Kronecker.

Pentru n=3 avem:

$$f(x_3, x_2, x_1) = \sum_{i=0}^{7} d_i m_i = \sum_{i=0}^{7} c_i\pi_i$$

$c = T_3 \bullet d$; $T_3 = (T_1 * T_1) * T_1 = T_2 * T_1$.

În cazul a n variabile avem:

$$f(x_n, x_{n-1}, \dots, x_1) = \sum_{i=0}^{2^n-1} d_i m_i = \sum_{i=0}^{2^n-1} c_i\pi_i,$$

unde

$\pi_i = x_n^{\alpha_{i,n}} x_{n-1}^{\alpha_{i,n-1}} \dots x_1^{\alpha_{i,1}}$

suma este sumă peste GF(2), $\alpha_{i,j} \in \{0,1\}$, $x^0=1$, $x^1=x$ iar vectorul c se obține:

$c = T_n \bullet d$; $T_n = T_1 * T_1 * \dots * T_1$;

$$T_n = \begin{pmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{pmatrix}.$$

Pentru cazul a n+1 variabile rezultă:

$$f(x_{n+1}, x_n, \dots, x_1) = \sum_{i=0}^{2^{n+1}-1} d_i m_i = \sum_{i=0}^{2^{n+1}-1} c_i\pi_i,$$

cu

$$c = \underbrace{\begin{pmatrix} T_n & 0 \\ T_n & T_n \end{pmatrix} \bullet d}_{} = T_{n+1} \bullet d;$$

$T_{n+1} = T_1 * T_1 * \dots * T_1$

n+1 ori

Observație. Deoarece $T_n^{-1} = T_n$ $(n \in N)$, cu $T_0=(1)$, corespondența c⇔D este bijectivă unde

$$C = \left\{c/c \in \{0,1\}^p, p = 2^n, n \in N\right\},$$

$$D = \left\{d/d \in \{0,1\}^p, p = 2^n, n \in N\right\}.$$

Considerăm acum expresiile Reed-Muller generalizate, adică expresiile de forma:

$$f\left(\overset{\bullet}{x}_n, \overset{\bullet}{x}_{n-1}, \dots, \overset{\bullet}{x}_1\right) = \sum_{i=0}^{2^n-1} c_i \overset{\bullet}{\pi}_i$$

unde $\overset{\bullet}{\pi}_i$ sunt pitermii generalizați, obținuți din pitermii corespunzători π_i substituind variabilele booleene x_j (j=1,...,n) prin variabilele $\overset{\bullet}{x}_j$ (j=1,...,n) respectiv, unde $\overset{\bullet}{x}_j = x_j$ sau $\overset{\bullet}{x}_j = \overline{x}_j = x_j \oplus 1$.

Pentru o funcție de n variabile există 2^n variante posibile, numite expresii Reed-Muller generalizate, fiecăreia corespunzîndu-i o polaritate bine determinată $p \in \{0, 1, 2, \dots, 2^n-1\}$, unde

$$<p>_{10} = <\alpha_1 \alpha_2 \dots \alpha_n>_2, \ \alpha_j \in \{0,1\}, \ j = 1, \dots, n$$

$$\alpha_j = \begin{cases} 1, & \text{dacă } \overset{\bullet}{x}_j = \overline{x}_j \\ 0, & \text{dacă } \overset{\bullet}{x}_j = x_j \end{cases}.$$

Observație. Expresiile Reed-Muller sînt expresii Reed-Muller generalizate de polaritate zero.

Teorema 2. Expresia Reed-Muller generalizată de polaritate p, pentru o funcție de n variabile are forma:

$$f\left(x_n, x_{n-1}, \dots, x_1\right) = \sum_{i=0}^{2^n-1} a_i \overset{\bullet}{\pi}_i,$$

unde:

$$a = Z_{<p>} \bullet c;$$

$$Z_{<p>} = Z_{\alpha_n} * Z_{\alpha_{n-1}} * Z_{\alpha_1};$$

$$Z_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ Z_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Demonstrația se realizează prin inducție după n.

Algoritmul de generare al expresiilor Reed-Muller generalizate constă, în esență, în apelul succesiv sau recursiv al unui set de proceduri dintre care menționăm:

- procedura Pcfbfcd, care preia coeficienții funcției booleene reprezentate sub formă canonică disjunctivă;
- procedura Metpas, care determină coeficienții expresiilor Reed- Muller utilizînd metoda triunghiului lui Pascal;
- procedura Typemax16, care permite generarea matricelor pătratice de dimensiune inferioară lui 16, implicate în calculul expresiilor Reed-Muller;
- procedura Typemin23, care are același rol ca procedura precedentă în cazul dimensiunii superioare lui 32;
- procedura Prodk, ce determină produsul Kronecker etc.

```
{*****************************************}
{Procedura Expand                         }
{*****************************************}
  PROCEDURE Expand
              (VAR af,bf:tablou;VAR kf:BYTE);
  VAR
     limp,limc,i,j,l,m:BYTE;
     pc:BYTE;
     vinci,vincj:BYTE;
  BEGIN
{*****************************************}
{Salvare pas precedent                    }
{*****************************************}
     pc:=kf-1;
     Doilap(pc,limp);
     FOR i:=1 TO limp DO
        BEGIN
           FOR j:=1 TO limp DO
           BEGIN
              bf [i,j] :=af [i,j];
           END;
        END;
{*****************************************}
{Generare stînga sus                      }
{*****************************************}
     Doilap (kf,limc);
     FOR i:=1 TO limp DO
     BEGIN
        FOR j:=1 TO limp DO
           BEGIN
              af [i,j]:=bf [i,j];
           END;
     END;
{*****************************************}
{Generare dreapta sus                     }
{*****************************************}
     vinci:=limp+1;
     FOR i:=1 TO limp DO
        BEGIN
           FOR j:=vinci TO limc DO
           BEGIN
              af [i,j]:=0;
           END;
        END;
{*****************************************}
{Generare stînga jos                      }
{*****************************************}
     l:=1;
     vinci:=limp+1;
     FOR i:=vinci TO limc DO
        BEGIN
           FOR j:= TO limp DO
              BEGIN
                 af[ i,j]:=bf [l,j];
              END;
              l:=l+1;
        END;
```

```
{*****************************************}
{Generare dreapta jos                     }
{*****************************************}
        l:=1;
        m:=1;
        vinci:=limp+1;
        vincj:=limp+1;
        FOR i:=vinci TO limc DO
          BEGIN
            FOR j:=vincj TO limc DO
              BEGIN
                af [i,j]:=bf [l,m];
                m:=m+1;
              END
              l:=l+1;
          END;
END;
{*****************************************}
{Sfîrşit procedura Expand.                }
{*****************************************}
{*****************************************}
{Procedura permite calculul vectorului dezvoltării}
{Reed-Muller de polaritate zero utilizînd metoda}
{triunghiului lui Pascal                  }
{*****************************************}
PROCEDURE MetPas
                (VAR df,cf:vect;VAR nf:BYTE);
VAR
        i,j,k:BYTE;
        t:vect;
BEGIN;
        CLRSCR;
        Pcfbfcd(nf,df);
        cf [1]:=df [nf];
        FOR k:=nf-1 DOWNTO 1 DO
          BEGIN
            FOR i:=1 TO k DO
              BEGIN
                t [i]:=df [i]+df [i+1];
                t [i]:=t [i] MOD 2;
              END;
              j:=nf-k+1;
              cf [j] :=t [i];
          END;
END;
{*****************************************}
{Sfîrşit procedura MetPas                 }
{*****************************************}
```

```
PROCEDURE Typemin32
                (VAR ma:tablou;VAR dma:BYTE);
{*****************************************}
{Procedura afişează matrici pătratice de dimensiuni >16}
{care sînt puteri ale lui doi.            }
{*****************************************}
VAR
        k,l,i,j,n:BYTE;
        npar:BYTE;
BEGIN
        npar:=dma DIV 16-1;
        FOR k:=0 TO npar DO
          BEGIN
          FOR 1:=0 TO npar DO
            BEGIN
              CLRSCR;
              n:=0;
              FOR i:=1 TO 16 DO
                BEGIN
                  FOR j:=1 TO 16 DO
                    BEGIN
                      GOTOXY(j+n,i);
                      WRITE (ma
                      [k*16+i,1*16+j]);
                      n:=n+1;
                    END;
                  n:=0;
                END;
                GOTOXY(10,20);
                WRITE('Apăsaţi o tastă!!!');
                READLN;
            END;
          END;
END;
{*****************************************}
{Sfîrşit procedura Typemin32.             }
{*****************************************}
PROCEDURE Typemax16
                (VAR ma:tablou;VAR dma:BYTE);
{*****************************************}
{Procedura afişează matrici pătratice de dimensiuni ≤ 16}
{*****************************************}
VAR
        i,j,k:BYTE;
BEGIN
        CLRSCR;
        k:=0;
        FOR i:=1 TO dma DO
          BEGIN
            FOR j:=1 TO dma DO
              BEGIN
                GOTOXY (j+k,i);
                WRITE (ma [i,j]);
                k:=k+1;
              END;
              k:=0;
          END;
          GOTOXY(10,20);
          WRITE('Apăsaţi o tastă!!');
          READLN;
END;
```

```
{*****************************************}
  Sfirşit procedura Typemax16.
{*****************************************}
{PROCEDURE Prodk(VAR fd1,fd2:BYTE;VAR}
{fa,fb,fc:tablou);                       }
{*****************************************}
{Procedura permite produsul Kronecker pentru două}
{matrici pătratice                       }
{Acesta este algoritmul specific al programului per
{ansamblu                                }
{*****************************************}
VAR
i,j,k,l,m,n,p:BYTE;
  BEGIN
    FOR i:=1 TO fd1 DO
      BEGIN
        FOR j:=1 TO fd1 DO
          BEGIN
            k:=fd2*(i-1)+1;
            l:=fd2*(j-1)+1;
            IF fa [i,j] =1 THEN
              BEGIN
                FOR m:=1 TO fd2 DO
                  BEGIN
                    p:=1;
                    FOR n:=1 TO fd2 DO
                      BEGIN
                        fc [k,p] :=fb [m,n] ;
                        p:=p+1;
                      END;
                    k:=k+1;
                  END;
              END
              ELSE
              BEGIN
                FOR m:=1 TO fd2 DO
                  BEGIN
                    p:=1;
                    FOR n:=1 TO fd2 DO
                      BEGIN
                        fc [k,p] :=0;
                        p:=p+1;
                      END;
                    k:=k+1;
                  END;
              END;
          END;
      END;
  END;
{*****************************************}
{Sfirşit procedură Prodk                 }
{*****************************************}
```

Pornind de la forma generală a unei expresii Reed-Muller ne propunem să realizăm un algoritm pentru sinteza circuitelor de comutație.

Conexiunile fundamentale, fiind conexiunile serie și conexiunile paralel, orice conexiune se reprezintă ca o combinație a celor două.

Pe baza acestor conexiuni fundamentale construim conexiunile corespunzătoare operațiilor modulo-2:

Conexiunile corespunzătoare pitermilor $\pi_i(i=0,1,2,...,2^n-1)$ au forma:
unde

$$<i> = <\alpha_1, \alpha_2, \ldots, \alpha_n> 2$$

$$\{j_1,\ldots,j_m\}=\{h/h\in\{1,2\ldots,n\}, \alpha_h=1\}$$

și au reprezentările:

Algoritmul pentru sinteza acestor circuite constă, în esență, în utilizarea procedurilor:

- Genpolgrm care generează polaritatea expresiilor Reed-Muller;
- Sumbin care realizează calculul recursiv al unei sume modulo-2;
- Prodbin care realizează calculul recursiv al produsului modulo-2;
- Proda care calculează vectorul a, al coeficienților expresiei Reed-Muller generalizată ($a=Z_{<p>}\bullet c$).

Bibliografie

1. GREEN, D.: Modern Logic Design, Addison-Wesley Publishing Company, 1986.