AUTOTESTAREA PSEUDOINELARĂ A MICROCONTROLERELOR NANOSATELITULUI SATUM

Serghei Grițcov, Alexandru Ghincul, Ghenadie Bodean
Universitatea Tehnică a Moldovei
{gritscov, aghincul, gbodean}@gmail.com

Abstract. *În lucrare se analizează autotestarea memoriei incorporate a microcontrolerelor din familia MSP430 bazată pe metoda de testare pseudo-inelară a dispozitivelor de memorie. Sunt expuse exemple de programe FLASH şi SRAM de testare a memoriilor microcontrolerului. Sunt prezentate programe-teste pentru determinarea defectărilor constante ale celulelor memoriei şi defectărilor legate de înregistrarea (intrarea) multiplă în memorie la viteza maximă.*

Cuvinte-cheie: *autotestare pseudoinelară, memorii operative, microcotroler RISC.*

I. Introducere

Memoria digitală este una din cele mai importante componente ale microcontrolerului (MCU). În microcontrolerele moderne este utilizată memoria FLASH (de tip EEPROM) şi memoria cu acces aleatoriu (RAM). În memoria FLASH, de regulă, se păstrează codurile de instrucțiune ale programului. Memoria RAM este destinată pentru stocarea temporară a datelor. La executarea programului MCU se efectuează eşantionarea secvențială a instrucțiunilor şi datelor din dispozitivele de memorie. De aceea, corectitudinea funcțiilor îndeplinite ale microcontrolerului va fi determinată de corectitudinea executării instrucțiunilor de înregistrare, stocare şi citire din memorie.

Microcontrolerele moderne conțin memorie încorporată cu un volum de la unități la sute de kilobaiți. Pentru implementarea memoriei de capacitate mare se utilizează conductoare lungi şi înguste cu un număr mare de intersecții. La o astfel de topologie a conductorilor cele mai frecvente defecte sunt înregistrarea multiplă în memorie la o rată maximă [1]. Pentru detectarea acestui tip de defectări se aplică testarea **at-speed**, anume **Back-to-Back** (BtB), în care se repetă înregistrarea datelor în memorie pentru un număr minim de cicluri ale generatorului microcontrolerului [2,3].

În [4] sunt expuşi algoritmii de autotestare a memoriei microcontrolerelor din familia AVR, elaborate în baza March-testelor. Testele March sunt o adaptare a tehnicii iterative de testare a memoriei cu celule *unipoziționale* (i.e. bit-orientate) la memoria cu celule *multipoziționale* (i.e. word-orientate). În lucru [5] a fost propusă o tehnică a autotestării memoriei numită testarea pseudo-inelară, care este *invariantă* în raport cu organizarea celulelor memoriei. Testarea pseudo-inelară sau π-testarea se bazează pe emularea automatului liniar, anume a registrului de deplasare cu legătură de reacție (LFSR), de însăşi memoria. Structura LFSR urmează structura polinomului generator ce este ales în dependență de binaritatea celulelor şi particularitățile defectărilor RAM. Parametrii principali pentru sintetizarea algoritmului π-testării sunt: (1) polinomul generator, de regulă, este ireductibil asupra câmpului Galois (simplu sau extins); (2) stările inițiale ale automatului LFSR pentru fiecare ciclu (iterație) al π-testării; (3) direcția de deplasare a LFSR-ului (în creştere, descreştere sau arbitrar) în spațiul de adrese al matricei memoriei.

În lucrarea prezentă sunt propuse şi analizate particularitățile autotestării pseudo-inelare a microcontrolerelor MSP430, folosite în proiectul nanosatelitului universitar SARUM. În secțiunea 2 sunt prezentate succint resursele microcontrolerelor MSP430, necesare pentru realizarea π-testării. În secțiunea 3 sunt propuşi şi descrişi algoritmii de bază ai π-testelor. În secțiunea 4 sunt prezentate exemple de implementare a π-testării microcontrolerelor MSP430.

II. Microcontrolerele din familia MSP430

A. Particularităţile arhitecturii procesorului central MSP430[6]:

- Arhitectura RISC de 16-biţi cu 27 de comenzi şi 7 regimuri de adresare;
- Arhitectura ortogonală, în care fiecare comandă este potrivită pentru fiecare regim de adresare;
- Acces complet la toate registrele;
- Instrucţiuni cu executarea într-un tact;
- Schimbul direct între celulele memoriei fără înregistrarea intermediară în registru;
- Comenzile şi adresările în formatele „cuvânt" şi „bait".

B. Registrele de lucru MSP430. MCU conţine 16 regiştri de 16 biţi. Registrele R0, R1, R2 şi R3 au o funcţie specială. Contorul de program (Program Counter, PC/R0) indică următoarea comandă ce va fi exectuată. Fiecare comandă constă dintr-un număr par de baiţi (doi, patru sau şase). Indicatorul stivei (Stack pointer, SP/R1) se utilizează de către MCU pentru stocarea adreselor de returnare din subprograme şi întreruperi. Registrul stării (State Register, SR/R2), utilizat ca registrul sursei sau destinatarului, poate fi adresat în regimul de înregistrare doar cu ajutorul cuvintelor-comenzi. R2 şi R3 – registrele generatorului constantelor CG1 şi CG2. R4 - R15 sunt registre cu destinaţie generală..

C. Regimuri de adresare în MSP430. Şapte regimuri de adresare pentru operandul sursei şi patru regimuri de adresare pentru operandul destinaţiei au posibilitatea de a adresa tot spaţiul de adrese. În tabelul 1 este prezentată configurarea biţilor pentru regimurile As (sursă) şi Ad (destinaţie).

Tabelul 1. Regimurile de adresare a operanzilor sursei/ destinatarului.

As/Ad	Regimuri de adresare	Sintaxă
00 / 0	Regim de registru	Rn
01 / 1	Regim de adrese	X(Rn)
01 / 1	Regim de simboluri	ADDR
01 / 1	Regim absolut (necondiţionat)	&ADDR
10 / -	Regim de registru indirect	@Rn
11 / -	Autoincrement indirect	@Rn+
11 / -	Regim direct (nemijlocit)	#N

D. Setul de instrucţiuni MCU. Setul complet de instrucţiuni din familia MSP430 conţine 27 de instrucţiuni ale nucleului şi 24 instrucţiuni emulate. Instrucţiunile nucleului au un cod unic. Instrucţiunile emulate sunt instrucţiuni care simplifică citirea şi înscrierea codului, dar care nu au un cod propriu de executare, de aceea asamblorul le transformă automat în instrucţiuni echivalente ale nucleului. Utilizarea instrucţiunilor emulate nu duce la mărirea volumului codului sau la micşorarea productivităţii MCU.

Există trei tipuri de comenzi ale nucleului:

- cu operand dublu;
- cu operand unic;
- comenzi de salt.

Setul de comenzi, necesare pentru realizarea testării în baza π-algoritmului, este prezentat în tab. 2.

Tabelul 2. Setul de comenzi MSP430 pentru realizarea π-testării

Mnemonica			**V**	**N**	**Z**	**C**

BIT(.B)	src,dst	src .and. dst	0	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
DEC(.B)*	dst	dst - 1 -> dst	*	*	*	*
JEQ/JZ	label		-	-	-	-
JMP	label	PC + 2 decalare -> PC	-	-	-	-
MOV(.B)	src,dst	src -> dst	-	-	-	-
RLC(.B)*	dst		*	*	*	*
TST(.B)*	dst	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	src .xor. dst -> dst	*	*	*	*

unde: * – flagul este setat; - – flagul este resetat; 0/1 – flagul este citit ca valoare de rezervă.

III. Testarea pseudo-inelară a memoriei MSP430

A. Sintetizarea schemelor testării pseudo-inelare

Testarea pseudo-inelară sau testarea π se bazează pe emularea registrului de deplasare cu legătura de reacție (LFSR) a memoriei. Memoria testată este reprezentată ca un masiv unidimensional, care conține celule *m*-poziționale. Celulele se indexează cu valori ale adreselor de la 0 până la *n*-1, unde *n* – capacitatea matricei de memorie. Ideea π-testării se bazează pe utilizarea a *k* celule de memorie în calitate de celule ale automatului LFSR şi deplasarea (virtuală a) registrului via toate celulele matricei de memorie. În cazul π-testării se realizează deplasarea LFSR-lui virtual în raport cu datele vs. a datelor în celulele registrului.

În procedura de π-testare se execută aşa-numitele π-iterații. Iterația π-tesării constă din următoarele: *inițializarea* LFSR-ului virtual cu starea **Init**, *tranzitarea* automatului LFSR în spațiul de adrese a memoriei microcontrolerului, *citirea* stării finale **Fin** a registrului virtual LFSR şi *analiza* rezultatului obținut. Calitatea π-testării se evstimează prin compararea stării finale **Fin** cu cea de referință, care poate fi egală cu **Init**. Pentru estimarea stării de referință folosim ecuația recurentă cunoscută Kolmogorov-Chapmen:

$$\mathbf{S_n} = \mathbf{S_0} * \mathbf{A^n}, \qquad (1)$$

unde $\mathbf{S}_0$ este vectorul de dimensiunea *k* al stării LFSR în momentul inițial de timp; **A** – matricea de însoțire bidimensională în formatul:

$$A = \begin{bmatrix} g_1 & 1 & 0 & \cdots & 0 \\ g_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \\ g_{k-1} & 0 & 0 & \cdots & 1 \\ g_k & 0 & 0 & \cdots & 0 \end{bmatrix}$$

unde $g_{(\bullet)}$ sunt coeficienții polinomului $g(z)$ de gradul *k* asupra $\mathbf{GF}(2^m)$, prezentat în formatul standart, i.e. al cărui coeficient liber g_0 este egal cu unu.

În cazul când numărul operațiilor de deplasare este proporțional perioadei *T* a polinomului $g(z)$, atunci comparaea va arăta astfel **Init** ≦ **Fin**. Dacă starea finală **Fin** va coincide cu cea inițială **Init**, atunci se va concluziona că nu au fost găsite detectate defectări la tipul specificat.

Structura LFSR-virtual urmează structura polinomului ireductibil $g(z) = \sum_{i=0}^{k} g_i z^i$, unde $g_i \in \mathbf{GF}(2^m) / p(x)$, $p(z) = \sum_{i=0}^{m} p_i x^i$, m= 2, 3,…, $p_i \in \{0,1\}$. În fig.1 este prezentat LFSR-ul cu sumatorii externi, legătura de reacție a căruia este determinată de structura polinomului

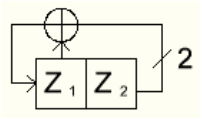


Figura 1. Registrul de deplasare pe **GF**(2^8) cu $g(z)=1+z+2z^2$, unde simbolul "/" indică operația de modulare înmulțită la constanta 2.

$g(z)=1+z+2z^2$ asupra extensiei câmpului Galois **GF**(2^8) generat de polinomul ireductibil $p(x)=1+x+x^6+x^7+x^8$.

La realizarea π-testării trebuie de efectuat înmulțirea cu 2 modulo $p(x)$ a celulei de 8 biți a memoriei. Pentru realizarea compactă a înmulțirii cu constanta se va aplica algoritmul de sintetizare a schemei înmulțirii modulară cu constanta [7]. Rezultatul aplicării algoritmului pentru exemplul analizat este prezentat în fig.2. Din figură rezultă că pentru implementarea "multiply_by_2" pe microcontrolerul MSP430, vor fi necesare o instrucțiune XOR şi o instrucțiune de deplasare.

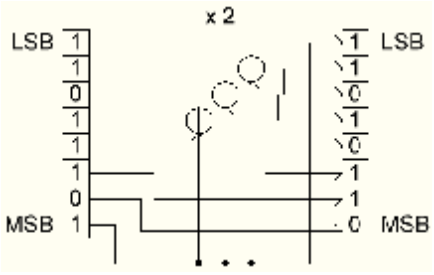


Figura 2. Schema înmulțirii modulare cu 2 (stânga – intrare: 187, dreapta – ieşire: 107)

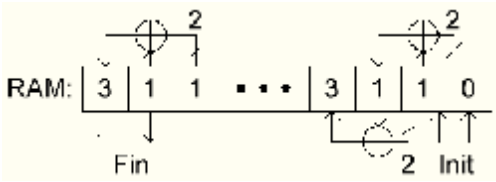
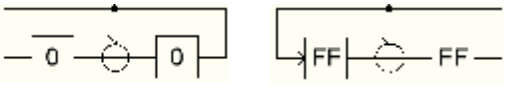


Figura 3. Iterația π-tesării a memoriei

În fig.3 este prezentat exemplul executării unei π-iterații de testare cu LFSR-ul virtual, descris de polinomul ireductibil $g(z)=1+z+2z^2$ asupra **GF**(2^8) cu $p(x)=1+x+x^6+x^7+x^8$. Parametrii π-iterației sunt: starea inițială **Init** = <1, 0> în formatul MSB..LSB; modul de deplasare LFSR – consecutiv (valorile adreselor în creştere sau descreştere); numărul tactelor (sau deplasărilor) automatului LFSR virtual pentru MSP430F2274 – egală cu 1023. În fiecare tact rezultatul legăturii de reacție, adică 1-$g(z)$, se înregistrează în celula succesivă a memoriei, care, împreună cu k-1 celule precedente specifică stările noi ale automatului LFSR.

B. Scheme de π-testare a defectărilor.

Pentru detectarea defectărilor constante s@0 şi s@1 (constanta zero şi respectiv constanta unu) se aplică automatul LFSR, prezentat în fig.4. Pentru detectarea constantei zero se foloseşte LFSR cu starea inițială **Init** = <0,0> (fig. 4, *a*), iar pentru detectarea constantei unu se foloseşte LFSR cu starea inițială **Init** = <255,255> (fig. 4, *b*) .



a) *b*)

Figura 4. LFSR cu sumatorii incorporați pentru detectarea defectărilor s@0 (*a*) şi s@1 (*b*), unde simbolul $\underline{\oplus}$ specifică operația XNOR.

Pentru detectarea defectărilor de înscriere multiplă în memorie la viteza maximă trebuie implementată metoda de π-testare **at-speed** de tipul **Back-to-Back** (BtB) [3,4]. Pentru aceasta**,** matricea de memorie se împarte într-un anumit număr de blocuri şi se efectuează testarea paralelă a tuturor blocurilor, fiecare din acestea având valori individuale de inițializare **Init** şi finale **Fin**. Deoarece microcontrolerul MSP430 nu susține executarea paralelă a comenzilor, apriori se efectuează calculul valorilor **Init**, apoi depanarea succesivă a π-testelor cu aceste valori.

IV. Π-testarea microcontrolerelor MSP430

În fig.5 este prezentată schema-bloc a algoritmului π-testării memoriei MSP430.

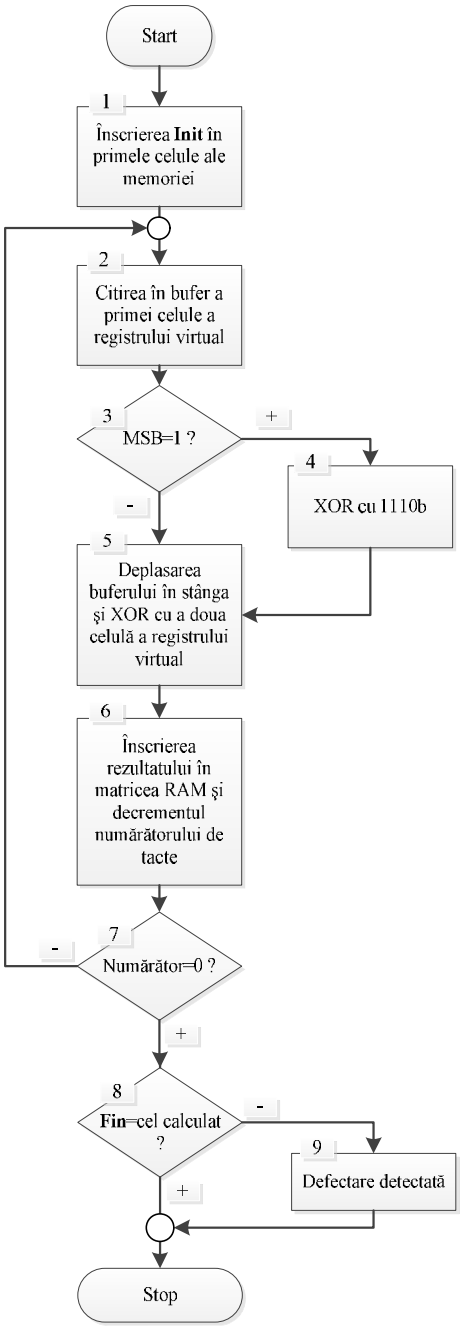


Figura 5. Algoritmul realizării π-testării memoriei MSP430

Modul de funcționare a algoritmului constă în următoarele: blocul 1 - în primele două celule ale memoriei se înscrie valoarea inițială a registrul virtual LFSR; blocul 2 - se citeşte valoarea MSB a celulei LFSR; blocurile 3..5 - se efectuează înmulțirea modulară la 2 (vezi secțiunea precedentă) după care se efectuează operația XOR cu celula LSB; blocul 6 - rezultatul obținut se înregistrează în următoarea celulă a memoriei, astfel se execută deplasarea în stânga a automatului LFSR; blocurile 7..8 - dacă numărătorul este în zero, valoarea rezultată din ultimele două celule ale memoriei se compară cu valoarea de referință (aşteptată). Remarcăm că numărătorul iterațiilor se setează cu valoarea capacității memoriei testate.

Mai jos este prezentat exemplul listingului iterației π-testării a memoriei RAM a microcontrolerului MSP430.

```
//*************************************************
Test_RAM:
   mov    #0200h,R6    ; fixăm începutul RAM
   mov    #1021,r5     ; fixăm valoarea RAM - 3
   mov.b #0,0(R6)      ; înscriem în RAM
   mov.b #1,1(R6)      ; starea inițială [0 1]
Cycle1:                ; ciclul inițial al testului
   mov.b @R6+,R7       ; citim valoarea celulei curente
   bit.b    #0080h,R7  ; verificăm MSB în prima celulă
   jnc      Cycle2     ; dacă MSB=0, trecem la Cycle2
   xor.b   #1110b,R7   ; dacă MSB=1, executăm xor
Cycle2:
   rlc.b   R7          ; deplasăm valoarea R7 în stânga
   xor.b  @R6,R7       ; î xor între valoarea
                              ; R7 şi următoarea celulă RAM
      mov.b R7,1(R6)   ; înregistrăm rezultatul în
                       ; următoarea celulă a memoriei
   dec      R5         ; micşorăm contorul ciclurilor cu 1
   jnz      Cycle1     ; repetăm toate operațiile dacă
                       ; contorul nu a atins valoarea 0
Finsh:                 ; dacă contorul arată 0, atunci
   cmp.b #1, 1(R6)     ; comparăm valoarea finală cu
                       ; cea de referință
   jne      Error      ; dacă nu coincid trecem la Detectat
//*************************************************
```

Listingul iterațiilor π-testului al memoriei FLASH MSP430 este identic listingului iterațiilor π-testului memoriei RAM, cu excepția că după fiecare operație de înregistrare în memoria FLASH, trebuie introdus următorul cod al programului:

```
//*************************************************
//---------------
   mov.       b #255,0(R6)     ; înregistrăm în Flash "255"
//---------------
   mov.       b #255,1(R6)     ; înregistrăm în Flash "255"
//---------------
   Xor        .b   @R6,R7      ; îndeplinim xor între valorile celulelor curentă şi următoare Flash
   inv.       b   R7           ; inversăm valoarea rezultatului
 //---------------
   cmp        #255, 1(R6)      ; comparăm valoarea finală cu cea de referință, în caz de neegalitate
```

```
    jne     Error           ; trecem la Detectat
  //*************************************************
```

Exemplul codului programului de testare RAM, în baza testelor pseudo-inelare prin metoda at-speed de tip BtB, este prezentat mai jos, unde *seg* – semnifică segmentele în care a fost divizată memoria RAM:

```
Test_RAM:               // seg = 204,8 bytes
   mov   #0200h,R6      ; fixăm începutul RAM pentru seg1
   mov   #02CCh,R8      ; fixăm începutul RAM pentru seg2
   mov   #0398h,R10     ; fixăm începutul RAM pentru seg3
   mov   #0464h,R12     ; fixăm începutul RAM pentru seg4
   mov   #0530h,R14     ; fixăm începutul RAM pentru seg5
   mov   #202,r5        ; fixăm mărimea RAM Segment-3
 T0:
   mov.b #0,0(R6)       ; înscriem în RAM seg1
   mov.b #1,1(R6)       ; starea inițială [0 1]
   //--------------------
   // similar repetăm fragmentul codului T0 pentru
   // registrele seg2 – seg5
Cycle01:
   mov.b @R6+,R7        ; citim valoarea din celula curentă

; seg1
   //--------------------
   // similar citim valorile din celulele seg2-seg4
   cmp   #1,R5          ; verificăm, dacă acesta este ultimul
   jeq   T1             ; ciclu al testului, permitem
                        ; următoarea comandă
   mov.b @R14+,R15      ; citim valoarea din celula 1 seg5
T1:
   bit.b #0080h,R7      ; verificăm MSB în celula 1 seg1
   jnc   F1             ; dacă MSB=0, trecem la F1
   xor.b #1110b,R7      ; dacă MSB=1, îndeplinim xor
F1:
   rlc.b R7             ; deplasăm valoarea R7 în stânga
   xor.b @R6,R7         ; xor între valoarea
                        ; R7 şi urm. celulă RAM
 // similar repetăm fragmentul codului T1 pentru  R8-R13
   cmp   #1,R5          ; verificăm, dacă acesta este ultimul
                        ; ciclu al testului, trecem la
   jeq   WriteD         ; eticheta WriteD
   bit.b #0080h,R15     ; verificăm MSB în celula 1 seg5
   jnc   F5             ; dacă MSB=0, trecem la F5
   xor.b #1110b,R15     ; dacă MSB=1, îndeplinim xor
F5:
   rlc.b R15            ; deplasăm valoarea R15 în stânga
   xor.b @R14,R15       ; xor între valoarea
                        ; R15 şi urm. celulă RAM
WriteD:
   cmp   #1,R5          ; verificăm, dacă acesta este ultimul
   jeq   T4             ; ciclu al testului, trecem la T4
   mov.b R7,1(R6)       ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg1
   // similar înscriem rezultatele în
   // următoarele celule ale memoriei pentru seg2-seg4
   jmp   Decr           ; trecem la eticheta Decr
T4:
   mov.b R7,1(R6)       ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg1
   mov.b R9,1(R8)       ; înscriem rezultatul în urm.
                        ;celulă a memoriei seg2
   mov.b R11,1(R10)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg3
   mov.b R13,1(R12)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg4
   // similar înscriem rezultatele în
   // următoarele celule ale memoriei pentru seg2-seg4
   jmp   Decr           ; trecem la eticheta Decr
T4:
   mov.b R7,1(R6)       ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg1
   mov.b R9,1(R8)       ; înscriem rezultatul în urm.
                        ;celulă a memoriei seg2
   mov.b R11,1(R10)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg3
   mov.b R13,1(R12)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg4
                        ; ciclu al testului, trecem la
   jeq   WriteD         ; eticheta WriteD
   bit.b #0080h,R15     ; verificăm MSB în celula 1 seg5
   jnc   F5             ; dacă MSB=0, trecem la F5
   xor.b #1110b,R15     ; dacă MSB=1, îndeplinim xor
F5:
   rlc.b R15            ; deplasăm valoarea R15 în stânga
   xor.b @R14,R15       ; îndeplinim xor între valoarea
                        ; R15 şi urm. celulă RAM
WriteD:
   cmp   #1,R5          ; verificăm, dacă acesta este ultimul
   jeq   T4             ; ciclu al testului, trecem la T4
   mov.b R7,1(R6)       ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg1
   // similar înscriem rezultatele în
   // următoarele celule ale memoriei pentru seg2-seg4
   jmp   Decr           ; trecem la eticheta Decr
T4:
   mov.b R7,1(R6)       ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg1
   mov.b R9,1(R8)       ; înscriem rezultatul în urm.
                        celulă a memoriei seg2
   mov.b R11,1(R10)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg3
   mov.b R13,1(R12)     ; înscriem rezultatul în urm.
                        ; celulă a memoriei seg4
Decr:
   dec   R5             ; micşorăm contorul ciclului cu 1
   jnz   Cycle01        ; repetăm toate operațiunile dacă
                        ; contorul nu indică valoarea 0
Finsh:
   cmp.b #109, 1(R6)    ; comparăm valoarea finală cu cea
                        ; de referință, în caz de  neegalitate
   jne   Detect         ; trecem la Detect
   //--------------------
   // similar repetăm fragmentul codului Finsh pentru
   // registrele R8, R10, R12
   cmp.b #214, 1(R14)   ; comparăm valoarea finală cu cea
                        ; de referință, în caz de  neegalitate
   jne   Detect         ; trecem la Detect
 //*************************************************
```

Notă: memoria MCU de 1024 baiți a fost împărțită în 5 segmente câte 2 registre per segment.

V. Analiza rezultatelor

Pentru estimarea eficacității testării pseudo-inelare vom compara π-testul constantei zero (s@0) cu respectivul testul March [4]:

```
(A) L1:
    LD      R0,Z        ; citirea "0" în R0
    ST      Z+,R17      ; înregistrarea în aceeaşi celulă a memoriei "1" cu incrementarea adresei memoriei
    OR      R20,R0      ; acumularea rezultatului în R20
    DEC     R18         ; decrementul contorul ciclului cu 1
    BRNE L1             ; repetare dacă numărătorul nu-i 0
-------------------------------------------------------------------------
    TST     R20         ; comparăm valoarea finală cu "0"
    BRNE  DETECT        ; dacă nu sunt egale, trecem la Detect
```

Notă: în programul [A] instrucțiunea "ST Z+,R17" este pregătitoare pentru următorul ciclu de citire a unității.

Testul exemplificat execută citirea zerourilor şi înscrierea unităților în ordinea crescătoare a adreselor ⇑(r0,w1). Mai jos este prezentat exemplul π-testării a detectării s@0:

```
(B) Cycle:
    mov.b @R6+,R7  ; citim valoarea din celula curentă
    xor.b @R6,R7   ; xor între val. celulei  Flash precedentă şi următoare
    mov.b R7,1(R6) ; înscriem rezultatul în urm. celulă ; a memoriei
    dec   R5       ; micşorăm numărătorul ciclului cu 1
    jnz     Cycle  ; repetăm toate operațiunile, până când numărătorul devine 0
-------------------------------------------------------------------------
    tst.b   1(R6)  ; comparăm starea finală cu "0"
    jnz     Error  ; dacă nu sunt egale, trecem la Detect
```

Complexitatea algoritmică a testului March şi a π-testului este aceeaşi şi constituie $7n$, unde n este numărul celulelor matricei de memori. Numărul iterațiilor este egal cu $8n$ la executarea testului March şi $11n$ – la executarea π-testului. Un număr mai mare de iterații pentru testarea pseudo-inelară se explică prin executarea comenzii "mov.b R7,1(R6)" a adresării prin indexare (în AVR lipseşte acest tip de adresare).

Notă: La executarea testului March după citirea "0" trebuie efectuată compararea valorilor citite şi "0".

În exemplul [A], la efectuarea instrucțiunii OR între valoarea citită curentă şi rezultatele anterioare, stocate în acumulator. Compararea poate fi efectuată doar după citirea întregului bloc de memorie. Astfel, pentru detectarea defectării s@0 în testul March este utilizat principiul similar cu principiul folosit în testarea pseudo-inelară, adică fiecare valoare nouă înregistrată în memorie este rezultatul calculelor a două valori anterioare. În ambele teste verificarea rezultatului se efectuează la sfârşitul testului, prin compararea conținutului ultimei celule a memoriei cu "0".

VI. Concluzii

În lucrare sunt analizați algoritmii de autotestare pseudo-inelară, numită π-testare, a memoriilor microcontrolerului MSP430. Sunt prezentate exemple de program ale π-testării pentru detectarea defectărilor singulare blocat 0 şi 1 şi a defectărilor cu înscriere multiplă în memorie la viteza maximă. Testul pseudo-inelar pentru memoria operativă de capacitatea 1024 baiți se execută în 18448 tacte, iar pentru memoria FLASH de capacitatea 1024 baiți – 22690 tacte.

VII. Referințe

1. Powell, T.J., Wu-Tung Cheng, Rayhawk, J., Samman, O., Policke, P., Lai, S., *Bist for deep submicron asic memories with high performance application* // *IEEE Int. Test Conf.*, pp. 386-392, 2003.

2. Xiaogang Du, Mukherjee, N., Wu-Tung Cheng, Reddy, S.M., *Full-speed field-programmable memory BIST architecture* // *IEEE Int. Test Conf*. paper 45.3, 2005.

3. Powell, T., Kumar, A., Rayhawk, J., Mukherjee, N., *Chasing subtle embedded RAM defects for nanometer technologies* // IEEE Int. Test Conf. paper 33.4, 2005.

4. Van de Goor, A., Gaydadjiev, G., Hamdioui, S., *Memory testing with a RISC microcontroller* // *IEEE Int. Test Conf*., pp. 214-219, 2010.

5. Bodean Gh., Bodean D., *Labunetz A., New Schemes for Self-Testing RAM* // DATE-2005.

6. MSP430x1xx Family, User's Guard http://focus.ti.com

7. Bodean Gh., *Generarea multiplicatoarelor asupra cîmpurilor Galois* - Meridian Ingeneresc, №3, pp. 14-21, 2006.