

Tema 1.B.

1. Descrierea programului

În acest proiect se implementează funcția nr. 4:

```
void complex_mult(Word16 *out, Word16 *in, Word16 m, Word16 sc_re, Word16 sc_im);
```

Funcția permite înmulțirea unui vector cu valori complexe cu un scalar complex conform formulei:

$$\begin{aligned} \text{out_re} &= (\text{in_re} * \text{sc_re} - \text{in_im} * \text{sc_im}) / 2 \\ \text{out_im} &= (\text{in_im} * \text{sc_re} + \text{in_re} * \text{sc_im}) / 2 \end{aligned}$$

Figura 1. Formula de calcul a datelor de ieșire

Word16 *out reprezintă un pointer la vectorul complex rezultat. Părțile reale și imaginare ale elementelor vectorului sunt stocate astfel: out_re(0), out_im(0), out_re(1), out_im(1), etc.

Word16 *in reprezintă un pointer la vectorul complex de înmulțit. Părțile reale și imaginare ale elementelor vectorului sunt stocate astfel: in_re(0), in_im(0), in_re(1), in_im(1), etc.

Word16 m reprezintă numărul de elemente complexe în vectorul in, multiplu de 4.

Word16 sc_re reprezintă partea reală a scalarului cu care se înmulțește.

Word16 sc_im reprezintă partea imaginară a scalarului cu care se înmulțește.

2. Detalii de implementare

a. Structura proiectului

Proiectul este structurat în următoarele foldere:

bin – în acest folder se găsește varianta compilată a proiectului.

cw – aici sunt stocate fișierele *.mcp și alte metadate necesare IDE-ului CodeWarrior. În acest folder se găsește fișierul necesar deschiderii proiectului „441F_MARCULESCU_Tudor.mcp”.

m – fișierul matlab în care există codul de generare al vectorului de intrare și apoi cel de comparare al vectorului de ieșire din programul C CodeWarrior cu varianta vectorului de ieșire calculată în matlab.

src – stochează fișierele tip sursă *.c și cele tip header *.h ale programului C CodeWarrior.

b. Programul Matlab de generare al vectorului de intrare și verificare al vectorului de ieșire calculat în programul C CodeWarrior.

Pentru generarea vectorului de intrare s-a folosit următorul cod în Matlab.

```

1      % nr total de numere in vector de intrare + 2
2      DataBlockSize=34;
3
4      % generare vector numere complexe
5      in = rand(1,DataBlockSize);
6
7      % se genereaza un fisier input.dat in care se vor scrie datele de intrare
8      fid=fopen('..\input.dat','w','b');
9
10     % se scrie vectorul de intrare in 'in' in fisier
11     fwrite(fid,in.*2^15,'int16');
12     fclose(fid);
13

```

Figura 2. Cod Matlab responsabil de generare valorilor de intrare

Linia 2: Se declară o variabilă care determină numărul total de date de intrare + 2. Acel +2 este datorat faptului că numărul complex cu care se înmulțește vectorul de intrare este generat în același vector de intrare, structura datelor de intrare fiind:

sc_re, sc_im, in_re_0, in_im_0, in_re_1, in_im_1, etc.

Linia 5: Datele sunt generate în mod complet aleator, acestea fiind valori subunitare cuprinse între 0 și 1.

Linia 7-12: Datele generate sunt stocate într-un fișier în format *.dat, compatibil cu programul C din CodeWarrior. Ele sunt înmulțite cu 2^{15} pentru a menține compatibilitatea și a putea fi interpretate în mod corect de programul scris în C.

```

14 % se ruleaza proiectul CW apoi se continua in Matlab
15 % se citesc datele de iesire din programul in C
16 fid=fopen('..\output.dat','r','b');
17 out_cw=fread(fid, DataBlockSize-2, 'int16');
18 out_cw=out_cw/(2^15);
19 fclose(fid);
20
21 % se recalculeaza datele de iesire in matlab
22 sc_re = in(1);
23 sc_im = in(2);
24 i = 3;
25 while i <= DataBlockSize
26     out(i-2) = ((in(i) * sc_re) - (in(i+1) * sc_im)) / 2;
27     out(i-1) = ((in(i+1) * sc_re) + (in(i) * sc_im)) / 2;
28     i = i + 2;
29 end
30 out = out';
31
32 % se afiseaza in consola side by side vectorul calculat in matlab
33 % versus vectorul citit din fisierul de iesire output.dat al programului
34 % in C din CodeWarrior
35 [out, out_cw]
36
37 % se calculeaza vectorul de erori relative
38 for i = 1:(DataBlockSize - 2)
39     err_relative(i) = (out_cw(i) - out(i)) / out(i);
40 end
41
42 % se afiseaza un grafic al valorilor erorii relative
43 plot(err_relative)
44 title('relative error')

```

Figura 3. Cod Matlab responsabil de comparația valorilor de ieșire din programul C cu cele recalculate în Matlab

A doua parte a programului se referă la metoda de verificare a datelor de ieșire obținute din programul C implementat în CodeWarrior.

Linia 16-19: Se citesc datele de ieșire din programul C și se stochează în vectorul out_cw.

Linia 22-30: Se calculează folosind formula de la punctul 1. datele de ieșire pe baza datelor de intrare stocate în vectorul in din matlab.

Linia 35: Se afișează în fereastra Command Window vectorul de ieșire calculat în matlab comparat cu vectorul de ieșire generat de programul C.

Command Window

```

ans =

    0.0435    0.0435
    0.0339    0.0339
    0.0278    0.0278
    0.0552    0.0552
    0.0819    0.0819
    0.0258    0.0258
    0.1455    0.1455

```

Figura 4. Exemplu date de ieșire ale programului Matlab din Command Window

Linia 37-44: Se calculează un vector de valori ale erorii relative ale valorilor calculate în programul C comparativ cu cele calculate în matlab. Se afișează un grafic pe baza valorilor erorii relative.

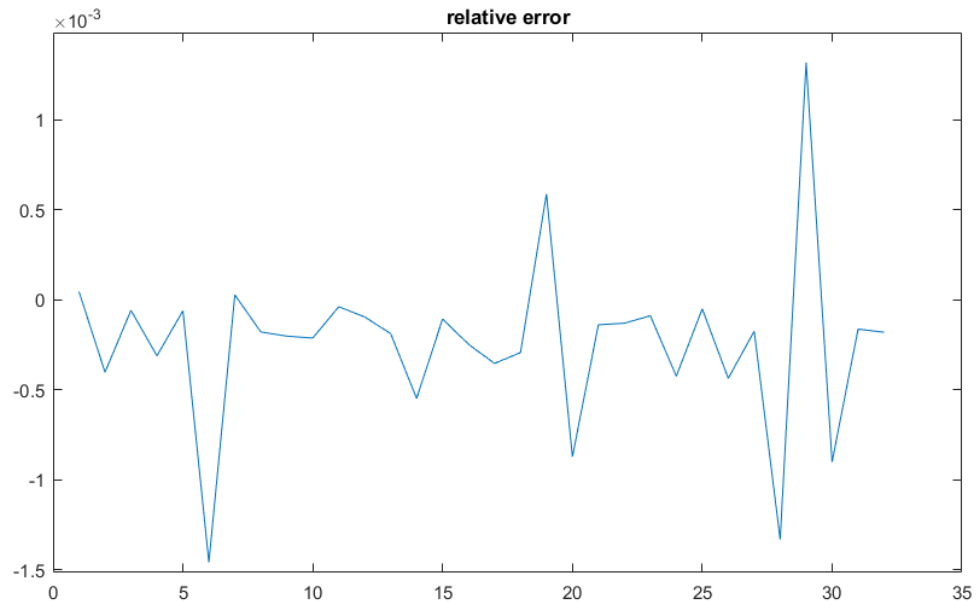


Figura 5. Exemplu date de grafic generat de programul Matlab

2.4 Programul C CodeWarrior de calcul al vectorului de ieșire

Programul în C este împărțit în mai multe fișiere:

- constants.h - în el sunt stocate constantele globale tip macro utilizate în cod.
- functions.h - în el este stocat prototipul funcției dată spre implementare, în cazul acestui proiect fiind vorba de „complex_mult”, cu directiva extern care permite la faza de link-are a programului, să se aducă funcția complex_mult în fișierul în care este inclus functions.h.
- complex_mult.c - aici este scrisă implantarea funcției complex_mult.
- main.c - fișierul sursă principal în care are este scrisă logica programului de generare a datelor de ieșire.

În continuare voi prezenta funcția principală `int_main()` scrisă în fișierul sursă `main.c`.

```

1  #include <prototype.h>
2  #include <stdio.h>
3  #include "constants.h"
4  #include "functions.h"
5
6  int main()
7  {
8
9      /* declare variables */
10     Word16 in[DataBlockSize] = {0};
11
12     Word16 in_aligned[DataBlockSize] = {0};
13     #pragma align in_aligned 8
14
15     Word16 out[DataBlockSize] = {0};
16     #pragma align out 8
17
18     Word16 sc_re          = 0;
19     Word16 sc_im          = 0;
20     FILE *fp              = NULL;
21     short i                = 0;
22
23     /* open the file generated in matlab */
24     fp=fopen("../input.dat","r+b");
25
26     /* read from the file */
27     fread(in,sizeof(Word16),DataBlockSize,fp);
28     if (!fp)
29         printf("\nNu s-a deschis input.dat");
30     fclose(fp);
31
32     /* assign the real and the imaginary part from the buffer */
33     sc_re = in[0];
34     sc_im = in[1];
35
36     /* generate the in_aligned[] vector */
37     memcpy(&in_aligned[0], &in[2], DataBlockSize-2 );
38
39     /* compute the output */
40     complex_mult(&out[0], &in[2], DataBlockSize - 2, sc_re, sc_im);
41
42     /* print the results to the console */
43     printf("Input: in_re[], in_im[], sc_re, sc_im.   Result out_re[], out_im[]\n");
44
45     for (i = 0; i < DataBlockSize - 2; i++)
46     {
47         printf("%d, %d, %d, %d, %d, %d\n", in[i+2], in[i+3], sc_re, sc_im, out[i], out[i+1]);
48     }
49
50     /* create an output file */
51     fp=fopen("../output.dat","w+b");
52
53     /* write the resulting vector in it */
54     fwrite(out,sizeof(Word16),DataBlockSize - 2,fp);
55     if (!fp)
56         printf("\nNu s-a deschis output.dat");
57     fclose(fp);
58
59     return(0);
60 }
61

```

Figura 6. Funcția principală int_main() a programului C

Linia 13 și linia 16: Pentru optimizare, se folosesc directivele de aliniere în memorie a vectorului de intrare, respectiv al vectorului de ieșire. Acest lucru va duce la utilizarea de către compilator a comenzii în asamblare:

```
move.2f (r1)+,d2:d3      ;[27,1]
```

care preia într-un singur ciclu de mașină atât partea fracționară, cât și partea imaginară din memorie.

Linia 27: Se observă cum datele generate anterior cu ajutorul matlab sunt citite de către programul C și sunt stocate în vectorul de intrare `in[DataBlockSize]`.

Linia 33-34: Conform a ceea ce am prezentat mai sus, `sc_re` și `sc_in` sunt preluate din datele de intrare stocate în vectorul `in`. Am ales ca acestea să fie primele din vector pentru a menține compatibilitatea cu programul matlab prezentat anterior.

Linia 37: Pentru a elimina primele 2 intrări ale `sc_re` și `sc_im` din vectorul de intrare, se face o copie a acestuia în vectorul aliniat în memorie „`in_aligned`”.

Linia 40: Este apelată funcția `complex_mult()` în interiorul căreia se calculează datele de ieșire stocate în vectorul `out[DataBlockSize]`.

Linia 45-48: Sunt afișate în consolă datele de ieșire.

Linia 54-57: Se generează fișierul de ieșire de tip `*.dat` în care se vor stoca datele de ieșire calculate în programul C. Acest fișier este preluat de programul matlab, după cum se poate observa în figura 3.

```

1  #include <prototype.h>
2
3  /* prototypes of global functions */
4  void complex_mult(Word16 *out, Word16 *in, Word16 m, Word16 sc_re, Word16 sc_im);
5
6
7  /* definition of global functions */
8  void complex_mult(Word16 *out, Word16 *in, Word16 m, Word16 sc_re, Word16 sc_im)
9  {
10     /* declare aligned vectors of input/output */
11     #pragma align *in 8
12     #pragma align *out 8
13
14     /* initialize buffers to store the mult results */
15     Word16 shResult1 = 0;
16     Word16 shResult2 = 0;
17     Word16 shResult3 = 0;
18     Word16 shResult4 = 0;
19     short i          = 0;
20
21     for (i = 0; i < m; i += 2)
22     {
23
24         /* first optimized cycle 4 alu operations + 2 mov */
25         /* compute the real part */
26         /* compute the first multiplication in_re * sc_re */
27         shResult1 = mult(*(in+i), sc_re);
28
29         /* compute the second multiplication in_im * sc_im */
30         shResult2 = mult(*(in+i+1), sc_im);
31
32         /* compute the imaginary part */
33         /* compute the third multiplication in_im * sc_re */
34         shResult3 = mult(*(in+i+1), sc_re);
35
36         /* compute the fourth multiplication in_re * sc_im */
37         shResult4 = mult(*(in+i), sc_im);
38
39
40         /* second optimized cycle 4 operations */
41         *(out+i) = sub(shResult1, shResult2);
42         *(out+i+1) = add(shResult3, shResult4);
43         *(out+i) = shr(*(out+i), 1);
44         *(out+i+1) = shr(*(out+i+1), 1);
45     }
46 }

```

Figura 7. Funcția principală complex_mult() a programului C

Linia 11-12: Se utilizează directivele de aliniere în memorie a datelor de intrare și de ieșire accesate prin intermediul pointerilor *in și *out.

Linia 27-42: Se utilizează funcțiile intrinseci ale suitei de dezvoltare pentru procesorul StarCore: „mult()”, „sub()”.

Linia 43-44: Se utilizează funcția intrinsecă shr() pentru împărțire la 2 a datelor calculate.

În funcția complex_mult() se menține formatul de stocare al datelor impusă de cerința problemei. Astfel, la finalul buclei for(), datele din vectorul out vor fi stocate sub forma:

out_re_0, out_im_0, out_re_1, out_im_1, etc.

În urma optimizării funcției, se obține următorul cod de asamblare din care voi prezenta bucla for():

```

96      FALIGN
97      LOOPSTART3
98      L7
99      DW6
100     move.2f  (r1)+,d2:d3      ;[27,1]
101     DW7
102     [
103     mpy      d2,d6,d8          ;[37,1]
104     mpy      d3,d6,d7          ;[30,1]
105     mpy      d2,d4,d5          ;[27,1]
106     ]
107     DW8
108     [
109     and      #-65536,d8,d8      ;[34,1]
110     and      #-65536,d7,d7      ;[41,1]
111     ]
112     DW9
113     [
114     mac      d3,d4,d8          ;[34,1]
115     sub      d7,d5,d10         ;[41,1]
116     ]
117     DW10
118     tfr      d8,d11            ;[41,1]
119     DW11
120     [
121     and      #-65536,d10,d10     ;[43,1]
122     and      #-65536,d11,d11     ;[44,1]
123     moves.2f d10:d11,(r0)       ;[41,1]
124     ]
125     DW12
126     [
127     asr      d10,d12            ;[43,1]
128     asr      d11,d13            ;[44,1]
129     ]
130     DW13
131     moves.2f d12:d13,(r0)+      ;[41,1]
132     LOOPEND3

```

Figura 8. Bucla for() din funcția complex_mult() optimizată în asamblare

Se observă că sunt utilizate operațiile pe numere fracționare, întrucât se folosește funcția „mpy” în locul „impy”.

De asemenea se utilizează funcția de stocare a datelor în memorie moves.2f d12:d13, (r0)+, care este mai optimă decât utilizarea a 2 move-uri.

Codul de asamblare compilat cu optimizare de nivel 2 s-a redus astfel la doar 151 de linii, față de programul neoptimizat care ar fi avut 424 de linii de cod.

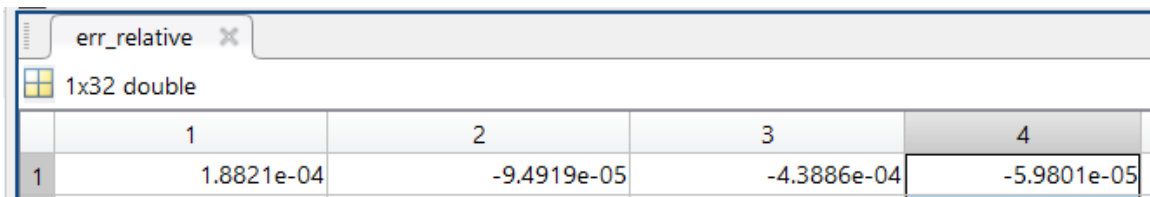
complex_mult.c	complex_mult.sl	complex_mult.sl
255	move.l	(sp-20),d7 ;[36,1]
256	DW90	
257	iadd	d7,d5 ;[36,1]
258	DW91	
259	move.l	d5,(sp-4) ;[36,1]
260	DW92	
261	move.w	#<2,d2 ;[36,1]
262	DW93	
263	move.l	(sp-4),d8 ;[36,1]
264	DW94	
265	iadd	d8,d2 ;[36,1]
266	DW95	
267	move.l	d2,(sp-8) ;[36,1]
268	DW96	
269	move.l	(sp-8),r4 ;[36,1]
270	DW97	
271	nop	;[0,0] AGU stal
272	DW98	
273	move.f	(r4),d8 ;[36,1]
274	DW99	
275	moves.f	d8,(sp-4) ;[36,1]
276	DW100	
277	move.f	(sp-52),d0 ;[36,1]
278	DW101	
279	move.f	(sp-4),d2 ;[36,1]
280	DW102	
281	mpy	d2,d0,d11 ;[36,1]
282	DW103	
283	moves.f	d11,(sp-14) ;[36,1]
284	DW104	
285	move.w	(sp-10),r4 ;[39,1]
286	DW105	
287	move.w	r4,(sp-8) ;[39,1]
288	DW106	
289	move.w	(sp-8),d2 ;[39,1]
290	DW107	
291	asll	d4,d2 ;[39,1]
292	DW108	
293	move.l	d2,(sp-8) ;[39,1]
294	DW109	
295	move.l	(sp-8),d13 ;[39,1]
296	DW110	
297	move.l	(sp-20),d14 ;[39,1]
298	DW111	
299	iadd	d14,d13 ;[39,1]
300	DW112	
301	move.l	d13,(sp-8) ;[39,1]
302	DW113	
303	move.l	(sp-8),r6 ;[39,1]
304	DW114	
305	nop	;[0,0] AGU stal
306	DW115	
307	move.f	(r6),d15 ;[39,1]
308	DW116	
309	moves.f	d15,(sp-4) ;[39,1]
310	DW117	
311	move.f	(sp-54),d2 ;[39,1]

Figura 9. Bucla for() din functia complex_mult() neoptimizată în asamblare

3. Rezultatele testării

În urma rulării proiectului se observă că datele de ieșire calculate în programul matlab coincid cu cele calculate în programul C din CodeWarrior. Acest lucru este susținut și de valorile obținute în vectorul de erori relative pe care l-am calculat folosind formula:

$$\text{err_relative}(i) = (\text{out_cw}(i) - \text{out}(i)) / \text{out}(i);$$



	1	2	3	4
1	1.8821e-04	-9.4919e-05	-4.3886e-04	-5.9801e-05

Figura 8. Valori eroare relativă

Pentru o mai bună inspecție a erorilor relative, se generează un grafic al acestora, în care pe axa OY sunt reprezentate valorile erorilor, iar pe axa OX sunt reprezentate pozițiile elementelor în vectorul de date de ieșire.

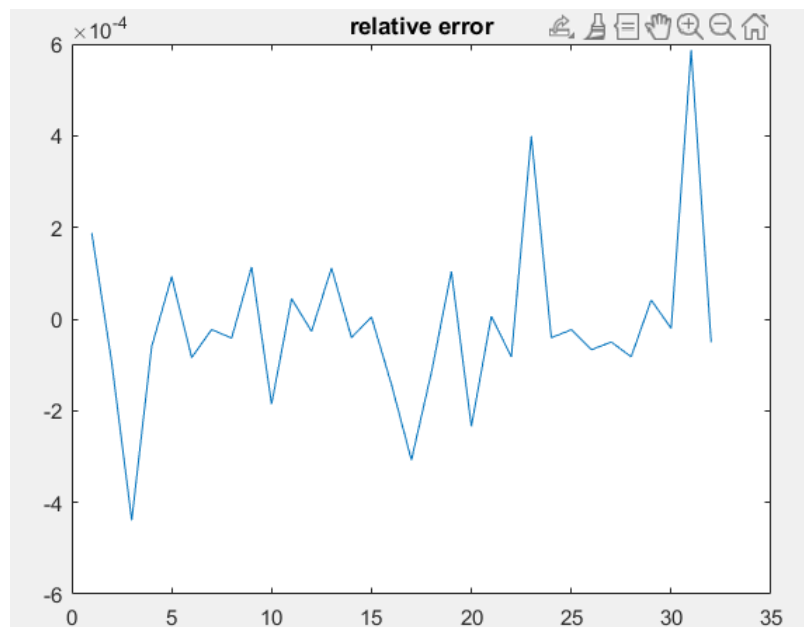


Figura 9. Valori eroare relativă