Integer Codes Correcting Single Errors and Detecting Burst Errors within a Byte

Aleksandar Radonjic, *IEEE Member*

***Abstract*-Correcting single and detecting adjacent errors has become important in memory systems using high density DRAM chips. The reason is that, in these systems, the strike of a single energetic particle can upset one or more adjacent bits. In this paper, we present a simple solution for this problem based on integer codes capable of correcting single errors and detecting *l*-bit burst errors within a *b*-bit byte ($1 < l \leq b$). Unlike the classical approach, the proposed one does not rely on the use of dedicated encoding/decoding hardware. Instead, it uses the processor as both encoder and decoder. The effectiveness of such solution is demonstrated on a theoretical model of an eight-core processor. The obtained results show that it has the potential to be used in future DDR5 systems.**

***Index Terms*-Integer codes, single errors, burst errors, DRAM chips, eight-core processor.**

I. Introduction

The last 40 years have seen a great progress in the design of DRAM chips. They became more and more dense, which reflected an increase in their capacity [1]. Along with these advances, the need to protect DRAM data from bit errors has also increased. Unlike older generations of DRAMs, the newer ones were (and still are) organized in a *b*-bit-per-chip manner (*b* = 4, 8, 16 and 32 bits). Because of this, they became more susceptible to bit errors caused by radiation effects [1]-[4].

In an attempt to mitigate this problem, engineers began to protect DRAM data with linear error correcting codes (ECCs). Among them, the most widely used are single error correcting and double error detecting (SEC-DED) codes [1]. In practice, these codes are always implemented in hardware. Specifically, the encoding/decoding (E/D) circuits are added to the DRAM controller, while one or two additional DRAM chips are used to store the check bits (DDR4 systems use one additional chip, while DDR5 systems will use two additional chips).

In contrast to this hardware-oriented approach, this paper proposes a different and much cheaper approach. The essence of our idea is to use codes that have low redundancy, that can be easily implemented in software, and that can correct/detect small number of adjacent errors (which are the most common types of errors in DRAM memories [1]-[4]). Since none of the existing ECCs meets these requirements, it is necessary to construct codes with quite unique properties. In this paper, we will show how one class of such codes can be constructed. In addition, we will show how these codes can be used in future DDR5 systems.

The author is with the Institute of Technical Sciences of the Serbian Academy of Sciences and Arts (SASA), 11000 Belgrade, Serbia (e-mail: sasa_radonjic@yahoo.com).

The organization of this paper is as follows: Section II deals with the construction of integer codes correcting single errors and detecting all *l*-bit burst errors confined to one *b*-bit byte ($1 < l \leq b$) (*l/b* burst errors) (integer SEC-B$_{l/b}$ED codes). The error control procedure for these codes is described in Section III, while Section IV is devoted to the evaluation and application of integer SEC-B$_{l/b}$ED codes in future DDR5 systems. Finally, Section V concludes the paper.

II. Codes Construction

In the coding literature, one can find many ECCs capable of correcting single errors and detecting adjacent errors [5]-[13]. The common feature for all these codes is that they use finite field (FF) operations to encode/decode data bits. For this reason, they are practically applicable only if implemented in dedicated chips (modern processors do not equip hardware for FF arithmetic). In contrast to [5]-[13], the proposed codes are suited for software implementation, since they are defined over the ring of integers modulo $2^b - 1$. This is formally described by the following definitions.

Definition 1. [15] *Let* $Z_{2^b-1} = \{0, 1, ..., 2^b - 2\}$ *be the ring of integers modulo* $2^b - 1$ *and let* $B_i = \sum_{n=0}^{b-1} a_n \cdot 2^n$ *be the integer representation of a b-bit byte, where* $a_n \in \{0, 1\}$ *and* $1 \leq i \leq k$. *Then, the code* $C(b, k, c)$, *defined as*

$$C(b, k, c) = \left\{ x \in Z_{2^b-1}^{k+1} : \sum_{i=1}^{k} C_i \cdot B_i \equiv B_{k+1} \pmod{2^b - 1} \right\} \quad (1)$$

is an $(kb + b, kb)$ *integer code, where* $x = (B_1, B_2, ..., B_k, B_{k+1}) \in Z_{2^b-1}^{k+1}$ *is the codeword vector,* $c = (C_1, C_2, ..., C_k, 1) \in Z_{2^b-1}^{k+1}$ *is the coefficient vector and* $B_{k+1} \in Z_{2^b-1}$ *is an integer.*

Definition 2. Let $x = (B_1, B_2, ..., B_k, B_{k+1}) \in Z_{2^b-1}^{k+1}$, $y = (\underline{B}_1, \underline{B}_2, ..., \underline{B}_k, \underline{B}_{k+1}) \in Z_{2^b-1}^{k+1}$ *and* $e = (\underline{B}_1 - B_1, \underline{B}_2 - B_2, ..., \underline{B}_k - B_k, B_{k+1} - \underline{B}_{k+1}) = (e_1, e_2, ..., e_k, e_{k+1}) \in Z_{2^b-1}^{k+1}$ *be the written codeword, the readout codeword and the error vector, respectively. Then, the syndrome S of the readout codeword is defined as*

$$S = \sum_{i=1}^{k} C_i \cdot \underline{B}_i - \underline{B}_{k+1} \pmod{2^b - 1} = \sum_{i=1}^{k+1} e_i \cdot C_i \pmod{2^b - 1} \quad (2)$$

The first step in constructing integer SEC-B$_{l/b}$ED codes is to determine the integer values of single and *l/b* burst errors. For that purpose, we will rely on the analysis from [14] and [15]. In the first paper, it was shown that the integer value of a *l/b* burst error is equal to $e_i = \pm 2^r \cdot (2m - 1)$, where $0 \leq r \leq b - l$, $1 \leq m \leq 2^{t-1}$ and $1 \leq t \leq l$. On the other hand, from [15] we know that the integer value of a single error is equal to $e_i = \pm 2^r$, where $0 \leq r \leq b - 1$. Having these in mind, we can state the

TABLE I
NUMBER OF COEFFICIENTS FOR SOME INTEGER SEC-B$_{l/b}$ED CODES.

		$b = 6$	$b = 7$	$b = 8$	$b = 9$	$b = 10$	$b = 11$	$b = 12$	$b = 13$
$l = 3$	k	0	1	6	8	13	27	85	99
	$\lvert s_2\rvert$	0	56	104	244	504	964	1758	4252
	$\lvert s_3\rvert$	0	42	38	104	238	466	272	1338
$l = 4$	k	0	0	2	3	7	17	38	58
	$\lvert s_2\rvert$	0	0	142	72	574	1284	2376	5176
	$\lvert s_3\rvert$	0	0	66	186	288	366	782	1480

following definitions and theorem.

Definition 3. The set of syndromes corresponding to single errors is defined as

$$s_1 = \bigcup_{r=0}^{b-1}\bigcup_{i=1}^{k+1}\left\{\pm 2^r \cdot C_i \ (\mathrm{mod}\ 2^b - 1)\right\} \tag{3}$$

Definition 4. The set of syndromes corresponding to l/b burst errors is defined as

$$s_2 = \bigcup_{r=0}^{b-l}\bigcup_{m=1}^{2^{t-1}}\bigcup_{t=1}^{l}\bigcup_{i=1}^{k+1}\left\{\pm 2^r \cdot (2m-1) \cdot C_i \ (\mathrm{mod}\ 2^b - 1)\right\} \tag{4}$$

Theorem 1. The cardinality of the set s_1 is
$|s_1| = 2 \cdot b \cdot (k+1)$.

Proof. The proof is given in [15]. □

Theorem 2. The codes defined by (1) can correct all single errors and detect all l/b burst errors if there exists k mutually different coefficients C_i such that

1. $|s_1| = 2 \cdot b \cdot (k+1)$,
2. $s_1 \cap s_2 = \varnothing$,
3. $s_2 \subseteq Z_{2^b-1} \backslash \{0,1\}$.

where $|s_1|$ denotes the cardinality of s_1.

Proof. From [15] we know that Condition 1 is the necessary and sufficient condition for correcting single errors. On the other hand, Conditions 2 and 3 ensure that the syndromes caused by *l/b* burst errors are nonzero and different from those generated by single errors. This asserts that *l/b* burst errors are detectable, and therefore, the codes satisfying the conditions 1 to 3 are (*kb* + *b*, *kb*) integer SEC-B$_{l/b}$ED codes. □

From Theorem 2 we see that the cardinality of the set s_2 is not known in advance. The reason is that some *l/b* burst errors may generate the same detectable syndrome.

Theorem 3. Let s_3 be the set of syndromes corresponding to multiple errors (m-errors) excluding l/b burst errors. Then, for any (*kb* + *b*, *kb*) *integer SEC-B$_{l/b}$ED code it holds that*

$$k \le \left\lfloor \frac{2^b - |s_2| - |s_3| - 2 \cdot b - 2}{2 \cdot b} \right\rfloor.$$

Proof. Definition 1 states that the total number of nonzero syndromes is equal to $2^b - 2$. On the other hand, the number of nonzero syndromes that respectively corresponds to single errors, *l/b* burst errors and *m*-errors is equal to $|s_1|$, $|s_2|$ and $|s_3|$. As a result, we have the inequality

$$2 \cdot b \cdot (k+1) + |s_2| + |s_3| \le 2^b - 2$$

where from it follows that

$$k \le \left\lfloor \frac{2^b - |s_2| - |s_3| - 2 \cdot b - 2}{2 \cdot b} \right\rfloor . \square$$

The last step in constructing integer SEC-B$_{l/b}$ED codes is to find the C_i's satisfying the conditions of Theorem 1. As this task cannot be performed without using a computer, it was necessary to write a suitable computer program. By using it, it is also possible to find out how the number of the C_i's depends on the byte length. Some of the obtained results are shown in Tables I-III.

TABLE II
COEFFICIENTS FOR SOME INTEGER SEC-B$_{3/b}$ED CODES.

$b = 7$											
11											
$b = 8$											
9	13	19	21	25	43						
$b = 9$											
9	11	15	19	23	29	35	37				
$b = 10$											
9	11	13	19	21	23	35	47	49	51	59	71
89											
$b = 11$											
9	11	13	15	17	19	21	23	25	29	35	41
49	53	67	71	83	93	101	111	113	141	155	163
199	211	217									
$b = 12$											
9	11	13	15	17	19	21	23	25	29	31	35
37	41	43	47	53	59	61	67	71	73	79	81
83	89	97	99	101	107	109	113	121	137	139	143
149	151	153	157	163	165	167	169	173	181	187	197
199	209	211	221	227	229	231	233	239	277	279	281
283	285	293	299	307	311	313	331	341	347	349	359
361	397	403	409	421	437	439	587	589	619	661	683
691											
$b = 13$											
9	11	13	15	17	19	21	23	25	29	31	35
37	41	43	47	49	53	59	61	67	71	79	81
83	89	99	101	107	109	113	117	121	131	135	139
143	151	153	163	165	167	179	181	187	189	195	197
199	209	211	219	225	227	229	233	241	275	277	279
283	313	315	323	325	331	337	341	347	349	357	359
361	373	431	455	457	465	477	549	555	557	587	589
603	611	615	651	683	697	733	743	843	873	1171	1173
1189	1229	1331									

TABLE III
COEFFICIENTS FOR SOME INTEGER SEC-B$_{4/b}$ED CODES.

$b = 8$											
21	43										
$b = 9$											
19	25	83									
$b = 10$											
17	19	21	23	43	47	53					
$b = 11$											
17	19	21	25	27	33	41	45	59	73	83	10
163	173	181	211	309							
$b = 12$											
17	19	21	23	25	27	29	31	33	37	41	45
53	55	61	67	73	77	79	83	89	97	121	137
141	167	181	197	211	227	293	301	307	309	311	327
587	683										
$b = 13$											
17	19	21	23	25	27	29	33	35	37	39	41
43	45	47	49	53	55	61	71	73	77	83	89
91	93	101	107	109	121	131	143	151	167	169	173
181	211	217	277	293	309	311	361	373	421	439	467
547	571	601	603	627	711	739	1235	1323	1325		

III. ERROR CONTROL PROCEDURE

The error control procedure (ECP) for the proposed codes is similar to those described in [14]-[17]. In short, when $S \neq 0$, the decoder will lookup the syndrome table (ST) to obtain the error correction data (ECD). If such data are not present in the ST, the decoder will declare a decoding failure. Otherwise, it will perform the operation

$$B_i = \underline{B}_i + E \ (\mathrm{mod}\ 2^b - 1) \tag{5}$$

TABLE IV
THE COMPLETE ST FOR THE (40, 32) INTEGER SEC-$B_{3/8}$ED CODE.

	S	i	E		S	i	E		S	i	E		S	i	E		S	i	E		S	i	E
1	1	5	1	**44**	44	0	0	**87**	87	4	8	**130**	130	0	0	**173**	173	0	0	**216**	216	0	0
2	2	5	2	**45**	45	0	0	**88**	88	0	0	**131**	131	0	0	**174**	174	4	16	**217**	217	3	2
3	3	0	0	**46**	46	0	0	**89**	89	0	0	**132**	132	1	127	**175**	175	0	0	**218**	218	0	0
4	4	5	4	**47**	47	2	16	**90**	90	0	0	**133**	133	0	0	**176**	176	0	0	**219**	219	1	4
5	5	0	0	**48**	48	0	0	**91**	91	0	0	**134**	134	2	127	**177**	177	0	0	**220**	220	0	0
6	6	0	0	**49**	49	3	239	**92**	92	0	0	**135**	135	0	0	**178**	178	0	0	**221**	221	0	0
7	7	0	0	**50**	50	0	0	**93**	93	4	32	**136**	136	0	0	**179**	179	3	4	**222**	222	1	32
8	8	5	8	**51**	51	0	0	**94**	94	2	32	**137**	137	3	127	**180**	180	0	0	**223**	223	5	223
9	9	1	254	**52**	52	2	251	**95**	95	0	0	**138**	138	4	127	**181**	181	0	0	**224**	224	0	0
10	10	0	0	**53**	53	0	0	**96**	96	0	0	**139**	139	0	0	**182**	182	0	0	**225**	225	0	0
11	11	0	0	**54**	54	0	0	**97**	97	0	0	**140**	140	0	0	**183**	183	1	8	**226**	226	0	0
12	12	0	0	**55**	55	0	0	**98**	98	3	223	**141**	141	0	0	**184**	184	0	0	**227**	227	0	0
13	13	2	254	**56**	56	0	0	**99**	99	0	0	**142**	142	0	0	**185**	185	0	0	**228**	228	0	0
14	14	0	0	**57**	57	0	0	**100**	100	0	0	**143**	143	0	0	**186**	186	4	64	**229**	229	2	2
15	15	0	0	**58**	58	0	0	**101**	101	0	0	**144**	144	1	239	**187**	187	0	0	**230**	230	0	0
16	16	5	16	**59**	59	3	64	**102**	102	0	0	**145**	145	0	0	**188**	188	2	64	**231**	231	0	0
17	17	0	0	**60**	60	0	0	**103**	103	3	8	**146**	146	0	0	**189**	189	1	64	**232**	232	0	0
18	18	1	253	**61**	61	0	0	**104**	104	2	247	**147**	147	0	0	**190**	190	0	0	**233**	233	0	0
19	19	3	254	**62**	62	0	0	**105**	105	0	0	**148**	148	0	0	**191**	191	5	191	**234**	234	4	1
20	20	0	0	**63**	63	0	0	**106**	106	0	0	**149**	149	0	0	**192**	192	0	0	**235**	235	0	0
21	21	4	254	**64**	64	5	64	**107**	107	0	0	**150**	150	0	0	**193**	193	0	0	**236**	236	3	1
22	22	0	0	**65**	65	0	0	**108**	108	0	0	**151**	151	2	8	**194**	194	0	0	**237**	237	1	2
23	23	0	0	**66**	66	1	191	**109**	109	0	0	**152**	152	3	247	**195**	195	0	0	**238**	238	0	0
24	24	0	0	**67**	67	2	191	**110**	110	0	0	**153**	153	0	0	**196**	196	3	191	**239**	239	5	239
25	25	0	0	**68**	68	0	0	**111**	111	1	16	**154**	154	0	0	**197**	197	0	0	**240**	240	0	0
26	26	2	253	**69**	69	4	191	**112**	112	0	0	**155**	155	0	0	**198**	198	0	0	**241**	241	0	0
27	27	0	0	**70**	70	0	0	**113**	113	0	0	**156**	156	0	0	**199**	199	0	0	**242**	242	2	1
28	28	0	0	**71**	71	0	0	**114**	114	0	0	**157**	157	3	32	**200**	200	0	0	**243**	243	0	0
29	29	0	0	**72**	72	1	247	**115**	115	0	0	**158**	158	0	0	**201**	201	0	0	**244**	244	0	0
30	30	0	0	**73**	73	0	0	**116**	116	0	0	**159**	159	0	0	**202**	202	0	0	**245**	245	0	0
31	31	0	0	**74**	74	0	0	**117**	117	4	128	**160**	160	0	0	**203**	203	2	4	**246**	246	1	1
32	32	5	32	**75**	75	0	0	**118**	118	3	128	**161**	161	2	223	**204**	204	0	0	**247**	247	5	247
33	33	1	223	**76**	76	3	251	**119**	119	0	0	**162**	162	4	223	**205**	205	0	0	**248**	248	0	0
34	34	0	0	**77**	77	0	0	**120**	120	0	0	**163**	163	0	0	**206**	206	3	16	**249**	249	0	0
35	35	0	0	**78**	78	0	0	**121**	121	2	128	**164**	164	0	0	**207**	207	0	0	**250**	250	0	0
36	36	1	251	**79**	79	0	0	**122**	122	0	0	**165**	165	0	0	**208**	208	2	239	**251**	251	5	251
37	37	0	0	**80**	80	0	0	**123**	123	1	128	**166**	166	0	0	**209**	209	0	0	**252**	252	0	0
38	38	3	253	**81**	81	4	239	**124**	124	0	0	**167**	167	0	0	**210**	210	0	0	**253**	253	5	253
39	39	0	0	**82**	82	0	0	**125**	125	0	0	**168**	168	4	247	**211**	211	0	0	**254**	254	5	254
40	40	0	0	**83**	83	0	0	**126**	126	0	0	**169**	169	0	0	**212**	212	0	0				
41	41	0	0	**84**	84	4	251	**127**	127	5	127	**170**	170	0	0	**213**	213	4	2				
42	42	4	253	**85**	85	0	0	**128**	128	5	128	**171**	171	4	4	**214**	214	0	0				
43	43	0	0	**86**	86	0	0	**129**	129	0	0	**172**	172	0	0	**215**	215	0	0				

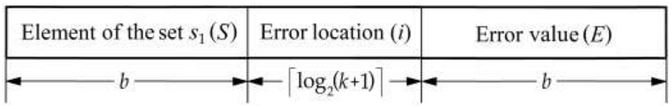

Fig. 1. Bit-width of one ST entry.

where $E \in \{\mp 2^r \pmod{2^b - 1}: 0 \le r \le b - 1\}$. To avoid multiple table lookups, and thus speed up ECP, we will employ a different strategy than in [14]-[17]. It is based on the use of a ST that has $2^b - 2$ entries (Definition 1) (Fig. 1), of which $|s_1|$ contain the ECD (Theorem 2), while the remaining $2^b - |s_1| - 2$ indicate the presence of multiple errors (l/b burst errors and m-errors). The pseudocode for generating such a ST is presented in Algorithm 1, while an illustration of its use is provided by the following example.

Example 1. Let $b = 8$, $l = 3$, $k = 4$, $C_1 = 9$, $C_2 = 13$, $C_3 = 19$ and $C_4 = 21$. Now, assume that the encoder needs to encode 32 data bits, 10101001 11001001 10100010 10101010. From (1) we know that the integer value of the last (fifth) byte will be equal to

$$B_{k+1} = B_5 = \sum_{i=1}^{4} C_i \cdot B_i \pmod{255} = 72 = 01001000_2$$

and the codeword will have the following form: $x = (B_1, B_2, B_3, B_4, B_5) = (10101001_2, 11001001_2, 10100010_2, 10101010_2, 01001000_2) = (169, 201, 162, 170, 72)$.

Algorithm 1 The pseudocode for generating the ST

```
Input: $b, k, c = \{C_1, C_2, \ldots, C_k, 1\}$;
Output: the syndrome table (ST);
ST = [ ];
for $S = 1$ to $2^b - 2$
    ST($S$) = [$S$, 0, 0]; // table entries indicating multiple errors //
end
for $i = 1$ to $|c|$
    for $r = 0$ to $b - 1$
        $S = [\pm 2^r \cdot c(i) \pmod{2^b - 1}]$;
        $E = \bmod\ [\mp 2^r \pmod{2^b - 1}]$;
        ST($S$) = [$S$, $i$, $E$]; // table entries with the ECD //
    end
end
```

Scenario 1: Suppose that the 6th bit is flipped. In that case, the codeword will have the form: $y = (\underline{B}_1, \underline{B}_2, \underline{B}_3, \underline{B}_4, \underline{B}_5) = (10101\underline{1}01_2, 11001001_2, 10100010_2, 10101010_2, 01001000_2) = (173, 201, 162, 170, 72)$. As explained previously, the decoder will perform the operation

$$S=\sum_{i=1}^{4} C_i \cdot \underline{B}_i - \underline{B}_5 \pmod{255} = 108-72 \pmod{255} = 36$$

after which it will conclude that the first byte is in error (Table IV). As a result, it will execute the operation

$$B_1 = \underline{B}_1 + E \pmod{255} = 173 + 251 \pmod{255} = 169.$$

Scenario 2: Assume now that the 10th, 11th and 12th bits are flipped. In that case, the codeword will have the following form: y = ($\underline{B}_1$, $\underline{B}_2$, $\underline{B}_3$, $\underline{B}_4$, $\underline{B}_5$) = (10101001_2, $\underline{001}01001_2$, 10100010_2, 10101010_2, 01001000_2) = (169, 41, 162, 170, 72). As in the previous case, the decoder will perform the operation

$$S=\sum_{i=1}^{4} C_i \cdot \underline{B}_i - \underline{B}_5 \pmod{255} = 32-72 \pmod{255} = 215$$

after which it will conclude that an uncorrectable error has occurred (Table IV). As a result, it will declare a decoding failure.

Scenario 3: Suppose that the 17th and 18th bits are flipped. In that case, the codeword will have the form: y = ($\underline{B}_1$, $\underline{B}_2$, $\underline{B}_3$, $\underline{B}_4$, $\underline{B}_5$) = (10101001_2, 11001001_2, $\underline{01}100010_2$, 10101010_2, 01001000_2) = (169, 201, 98, 170, 72). Now, after calculating

$$S=\sum_{i=1}^{4} C_i \cdot \underline{B}_i - \underline{B}_5 \pmod{255} = 131-72 \pmod{255} = 59$$

the decoder will conclude that the error has occurred within the third byte. As a result, it will perform the operation

$$B_3 = \underline{B}_3 + E \pmod{255} = 98 + 64 \pmod{255} = 162.$$

Scenario 4: Finally, assume that the 19th, 20th and 21th bits are flipped. In that case, the readout codeword will have the form: y = ($\underline{B}_1$, $\underline{B}_2$, $\underline{B}_3$, $\underline{B}_4$, $\underline{B}_5$) = (10101001_2, 11001001_2, $10\underline{011}010_2$, 10101010_2, 01001000_2) = (169, 201, 154, 170, 72). As in the previous case, after calculating

$$S=\sum_{i=1}^{4} C_i \cdot \underline{B}_i - \underline{B}_5 \pmod{255} = 175-72 \pmod{255} = 103$$

the decoder will conclude that the error has occurred within the third byte. As a result, it will execute the operation

$$B_3 = \underline{B}_3 + E \pmod{255} = 154 + 8 \pmod{255} = 162.$$

From the last two scenarios we see that the decoder can correct byte errors having the same integer values as single errors. In other words, in addition to correcting single errors, the decoder can correct 50% of double adjacent (DA) errors, 25% of triple adjacent errors and so on.

IV. Evaluation and Application in DDR5 Systems

A. Evaluation and Implementation Strategy

As mentioned in Section I, there are many codes that can correct single errors and detect adjacent errors. However, only two of them belong to a class of single error correcting and l-bit burst error detecting (SEC-B$_l$ED) codes. The first codes are obtained by modifying the OLS codes [10], while the second ones are designed to be optimal in terms of redundancy [11]. For this reason, in the following, we will focus only on the codes from [11], and compare them with the proposed codes.

At the outset, let us analyze the redundancy of both codes. According to [11], the optimal linear SEC-B$_l$ED codes have parameters [$l\cdot(2^{r-l}-1)$, $l\cdot(2^{r-l}-1)-r$], where $l \geq 3$. In contrast, in Section II, we have seen that integer SEC-B$_{l/b}$ED codes are characterized by the parameters ($kb+b$, kb). If we combine this with the data given in Tables I-III, we easily come to the conclusion that the proposed codes require 1 to 2 check bits more than the linear SEC-B$_l$ED codes (Table V). Besides this, in Section II we have seen that the proposed codes can detect l/b burst errors, while the codes from [11] can detect all l-bit burst errors, including those affecting two adjacent bytes.

TABLE V
Check-bit Lengths of the Optimal Linear SEC-B$_l$ED Codes and Proposed Codes.

Codes		Data word length (bits)						
		8	16	32	64	128	256	512
Codes from [11]	$l = 3$	6	7	7	8	9	10	11
	$l = 4$	7	7	8	9	10	11	12
Proposed codes	$l = 3$	8	8	8	9	10	11	12
	$l = 4$	8	8	10	10	11	12	13

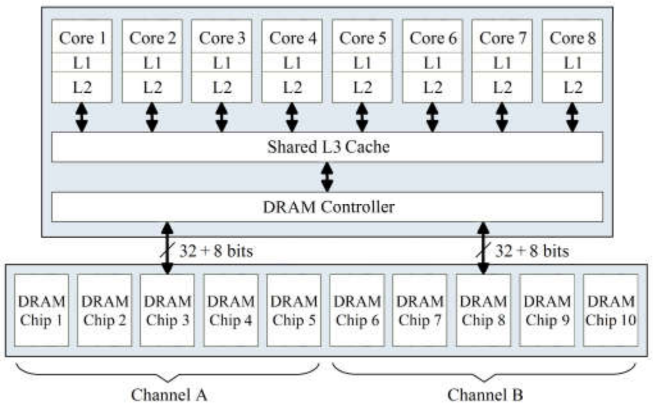

Fig. 2. Block diagram of an eight-core server processor supporting one DDR5 ECC memory module.

Although the mentioned drawbacks may seem important, from the standpoint of potential application in DDR5 systems, they are practically irrelevant. The reason for this lies in the architectural organization of the ECC DDR5 memory module. Namely, it is known that this module has two independent channels, each of which is 40 bits wide (32 data bits + 8 ECC bits) [18] (Fig. 2). This means that, in both channels, four chips are used to hold user data, while the fifth chip stores the check bits of the ECC. From this it is easy to conclude that both, the (40, 32) linear SEC-B$_4$ED code and the (40, 32) integer SEC-B$_{3/8}$ED code, could be used in DDR5 systems. The main advantage of the (40, 32) linear SEC-B$_4$ED code is that it guarantees detection of all bursts of length up to four bits (in future DDR5 systems these errors will be limited to 8-bit bytes, since DRAM chips are physically separated and independent). On the other hand, the advantage of the (40, 32) integer SEC-B$_{3/8}$ED code is that it can correct some additional byte errors, such as 50% of all DA errors.

However, the most important advantage of the proposed codes is their ability to run fast on modern processors. This feature does not have any linear code, including [13]. On the other hand, existing integer ECCs [14]-[17] have the potential to be implemented fast in software, but do not meet the criteria for use in DDR5 systems (they have either high redundancy [14], [16], [17], or weak error control capabilities [15]). With all this in mind, below we will consider how the proposed codes could be used to improve the reliability of the future DDR5 system. In this regard, suppose that we want to improve the reliability of the memory system of an eight-core processor that has four integer units per core (IU$_1$, IU$_2$, IU$_3$ and IU$_4$) and that supports four DDR5 ECC memory modules. Considering that the DRAM access latency has not changed significantly over the last decade [19], we can assume that this processor has the following specifications [20], [21]:

1) clock cycle: $3.3\cdot 10^9$ Hz,
2) integer addition latency: 1 clock cycle,
3) integer multiplication latency: 3 clock cycles,
4) modulo reduction latency: 1 clock cycle,
5) comparison operation latency: 1 clock cycle,
6) L1 cache (64 KB) access latency: 4 clock cycles,
7) L2 cache (1 MB) access latency: 14 clock cycles,
8) L3 cache (11 MB) access latency: 34 clock cycles,
9) DRAM (32 GB) access latency: 100-150 clock cycles.

Suppose now that eight integer (40, 32) SEC-$B_{3/8}$ED codes are used to encode 32 data bytes (B_1, B_2, ..., B_{32}). In addition, suppose the STs (Table IV) are placed in all L1 caches, and that four registers within each core store the coefficients C_1 = 9, C_2 = 13, C_3 = 19, and C_4 = 21 [note that one ST occupies $\left|2^b - 2\right| \times (2\cdot b + log_2\lceil k+1\rceil) \approx 0.6$ KB of memory]. In that case, each IU_1 will perform the following operations:

- *Core* 1 (IU_1)

$$B_{33} = \sum_{i=1}^{4} C_i \cdot B_i \text{ (mod 255)} \tag{6}$$

- *Core* 2 (IU_1)

$$B_{34} = \sum_{i=1}^{4} C_i \cdot B_{i+4} \text{ (mod 255)} \tag{7}$$

$$\vdots$$

- *Core* 8 (IU_1)

$$B_{40} = \sum_{i=1}^{4} C_i \cdot B_{i+28} \text{ (mod 255)} \tag{8}$$

After finishing this task, the processor will generate the 320-bit super codeword [x = (B_1, B_2, ..., B_{32},..., B_{40})], which will then be stored into the DRAM memory.

Suppose now that the DRAM controller reads the super codeword from the DRAM memory and then passes it on to the processor. In that case, each IU_1 will compute the value of the corresponding syndrome:

- *Core* 1 (IU_1)

$$S_1 = \sum_{i=1}^{4} C_i \cdot \underline{B}_i - \underline{B}_{33} \text{ (mod 255)} \tag{9}$$

- *Core* 2 (IU_1)

$$S_2 = \sum_{i=1}^{4} C_i \cdot \underline{B}_{i+4} - \underline{B}_{34} \text{ (mod 255)} \tag{10}$$

$$\vdots$$

- *Core* 8 (IU_1)

$$S_8 = \sum_{i=1}^{4} C_i \cdot \underline{B}_{i+28} - \underline{B}_{40} \text{ (mod 255)} \tag{11}$$

If the u syndromes ($1 \le u \le 8$) are different from zero, the IU_1s will accesses (in parallel) to the L1 caches and read out the corresponding entries from the STs. If any of these entries has the form [S, 0, 0], the processor will declare a decoding failure. Otherwise, the corresponding IU_1(s) will perform the operation(s) $B_i = \underline{B}_i + E$ (mod 255), after which the processor will accept the super codeword as error-free.

B. Analysis and Discussion

From the above, it can be seen that the E/D procedures are highly parallelized. This significantly reduces the number of clock cycles needed to process DRAM bits. In particular, to generate eight check-bytes, the processor needs T_{EN} = 16 clock cycles (4 integer multiplications, 3 integer additions and 1 modulo reduction), which is only 2 cycles greater than the L2 cache access latency. The decoding procedure, on the other hand, is more complicated than the encoding one and, thus, more time-consuming. The first step in decoding is to generate eight syndromes. For that purpose, the processor needs T_1 = 17 clock cycles (4 integer multiplications, 3 integer additions, 1 integer subtraction and 1 modulo reduction). In the next step, the processor will take T_2 = 1 clock cycle to check if all the syndromes are equal to zero (1 comparison operation). If this is not the case, and if the corresponding ST entries are not of the form [S, 0, 0], the processor will spend T_3 = 7 clock cycles (1 access to the L1 cache, 1 comparison operation, 1 integer addition and 1 modulo reduction) to correct the errors. So overall, the processor needs $T_{DE} = T_1 + T_2 + T_3 = 25$ clock cycles to decode the super codeword. Although this time is more than 50 percent longer than the encoding time, it is still negligible. This can be best seen from the fact that the DRAM access latency is 4 to 6 times greater than the time needed for decoding.

On the other hand, it should be borne in mind that the above example represents only one possible application of the proposed codes. So, for instance, if the processor supports eight DDR5 ECC memory modules (the maximum number of memory modules allowed by the DDR5 standard [18]) one needs to use sixteen integer (40, 32) SEC-$B_{3/8}$ED codes. Then the E/D operations will be performed (in parallel) by the IU_1s and IU_2s, which means that the 640-bit super codeword will be encoded and decoded in 16 and 29 clock cycles, respectively. The increased decoding latency of four clock cycles is a consequence of the fact that two threads share the same L1 cache (one thread must wait for another to get the ECD). However, even then, the decoding latency will be 5 clock cycles less than the L3 cache access latency. At the end, it should be noted that the use of sixteen IUs (two IUs per core) for E/D purposes cannot be considered as a burden for the processor, since each core has four IUs and at least two units for floating point calculations [20].

V. Conclusion

This paper has presented a class of integer codes that are suitable for use in future DDR5 systems. We have shown that the presented codes have two important characteristics: first, they can correct single errors and detect burst errors within a byte, and second, they use integer and lookup table operations to encode/decode data bits. Thanks to these, the presented codes have the potential to be implemented in software. To further confirm this, we have performed a theoretical analysis in the case of an eight-core processor supporting four DDR5 ECC memory modules. The obtained results have shown that the time penalty for using eight integer (40, 32) SEC-$B_{3/8}$ED codes is 25 clock cycles. It has been also indirectly shown that these results are not final, i.e. the time penalty can be further reduced by using faster and/or more powerful processors.

REFERENCES

[1] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*, John Wiley & Sons, Inc., 2006.

[2] B. Schroeder *et al*., “DRAM Errors in the Wild: A Large-Scale Field Study,” in *Proc. SIGMETRICS 2009*, pp. 193-204, June 2009.

[3] V. Sridharan *et al*., “Memory Errors in Modern Systems: The Good, The Bad, and The Ugly,” in *Proc. ASPLOS 2015*, pp. 297-310, Mar. 2015.

[4] J. Meza *et al*., “Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends From the Field,” in *Proc. DSN 2015*, pp. 415-426, Sept. 2015.

[5] D. Bossen, L. Chang and C. Chen, “Measurement and Generation of Error Correcting Codes for Package Failures,” *IEEE Trans. Comput.*, vol. 27, no. 3, pp. 201-204, March 1978.

[6] S. Reddy, “A Class of Linear Codes for Error Control in Byte-per-Card Organized Digital Systems,” *IEEE Trans. Comput*., vol. 27, no. 5, pp. 455-459, May 1978.

[7] J. Schwartz and J. Wolf, “A Systematic (12, 8) Code for Correcting Single Errors and Detecting Adjacent Errors,” *IEEE Trans. Comput.,* vol. 39, no. 11, pp. 1403-1404, Nov. 1990.

[8] T. Etzion, “Optimal Codes for Correcting Single Errors and Detecting Adjacent Errors,” *IEEE Trans. Inform. Theory*, vol. 38, no. 7, pp. 1357–1360, July 1992.

[9] R. Johansson, “A Class of (12, 8) Codes for Correcting Single Errors and Detecting Double Errors within a Nibble,” *IEEE Trans. Comput.,* vol. 42, no. 12, pp. 1504-1506, Dec. 1993.

[10] L. Everett *et al*., “Check Bit Code Circuit for Simultaneous Single Bit Error Correction and Burst Error Detection,” U.S. Patent 5,457,702, Oct. 10, 1995.

[11] B. Fu and P. Ampadu, “Burst Error Detection Hybrid ARQ with Crosstalk-Delay Reduction for Reliable On-Chip Interconnects,” in *Proc. 24th IEEE Int. Symp. Defect Fault Toler. VLSI Syst*., pp. 440-448, Oct. 2009.

[12] A. Sánchez-Macián, P. Reviriego and J. Maestro, “Enhanced Detection of Double and Triple Adjacent Errors in Hamming Codes Through Selective Bit Placement,” *IEEE Trans. Device Mater. Rel.*, vol. 12, no. 2, pp. 357-362, Jun. 2012.

[13] A. Sánchez-Macián, P. Reviriego, J. Maestro, “Hamming SEC-DAED and Extended Hamming SEC-DED-TAED Codes Through Selective Shortening and Bit Placement", *IEEE Trans. Device Mater. Rel.*, vol. 14, no. 1, pp. 574-576, Mar. 2014.

[14] A. Radonjic and V. Vujicic, “Integer Codes Correcting Burst Errors within a Byte,” *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 411-415, Feb. 2013.

[15] A. Radonjic, “(Perfect) Integer Codes Correcting Single Errors,” *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 17-20, Jan. 2018.

[16] A. Radonjic and V. Vujicic, “Integer Codes Correcting Sparse Byte Errors,” *Cryptogr. Commun.*, vol. 11, no. 5, pp. 1069-1077, Sept. 2019.

[17] A. Radonjic, “Integer Codes Double Errors and Triple-Adjacent Errors within a Byte,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 8, pp. 1901-1908, Aug. 2020.

[18] K. Kim *et al.*, “Using Dual Channel Memory as Single Channel Memory with Spares,” U.S. Patent 20200075079A1, March 2, 2020.

[19] K. Chang, “Understanding and Improving the Latency of DRAM-Based Memory Systems,” Ph.D. dissertation, Carnegie Mellon Univ., 2017.

[20] A. Fog. (Mar. 8, 2020). The Microarchitecture of Intel, AMD and via CPUs: An Optimization Guide for Assembly Programmers and Compiler Makers. Technical University of Denmark. [Online]. Available: https://www.agner.org/optimize/microarchitecture.pdf

[21] Intel Corp. (2017). Intel Xeon W-2145 Processor. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/126707/intel-xeon-w-2145-processor-11m-cache-3-70-ghz.html