Pipelined and Partitionable Forward Error Correction and Cyclic Redundancy Check Circuitry Implementation for PCI Express® 6.0

Dr. Debendra Das Sharma and Swadesh Choudhary

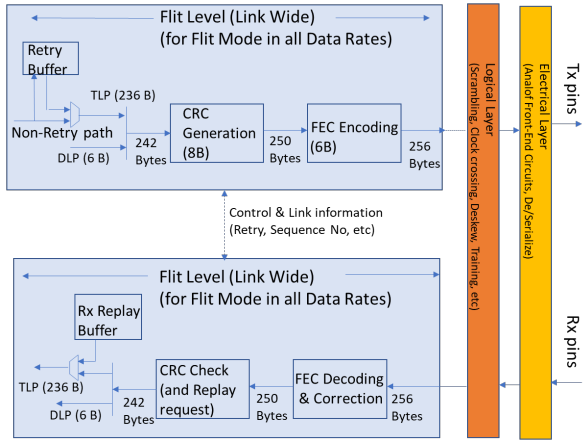
Intel Corporation

Agenda

- PCIe 6.0 FEC and CRC recap
- FEC implementation
- CRC implementation
- Results

PCIe FEC and CRC Recap

x8 Lanes	0	1	2	3	4	5	6	7
256 UI								
TLP Bytes	0	1	2	3	4	5	6	7
(0-299)	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23
	24	25	26	27	28	29	30	31
	32	33	34	35	36	37	38	39
	40	41	42	43	44	45	46	47
	48	49	50	51	52	53	54	55
	56	57	58	59	60	61	62	63
	64	65	66	67	68	69	70	71
	72	73	74	75	76	77	78	79
	80	81	82	83	84	85	86	87
	88	89	90	91	92	93	94	95
	96	97	98	99	100	101	102	103
	104	105	106	107	108	109	110	111
	112	113	114	115	116	117	118	119
	120	121	122	123	124	125	126	127
	128	129	130	131	132	133	134	135
	136	137	138	139	140	141	142	143
	144	145	146	147	148	149	150	151
	152	153	154	155	156	157	158	159
	160	161	162	163	164	165	166	167
	168	169	170	171	172	173	174	175
	176	177	178	179	180	181	182	183
	184	185	186	187	188	189	190	191
	192	193	194	195	196	197	198	199
	200	201	202	203	204	205	206	207
	208	209	210	211	212	213	214	215
	216	217	218	219	220	221	222	223
	224	225	226	227	228	229	230	231
	232	233	234	235	dlp0	dlp1	dlp2	dlp3
	dlp4	dlp5	crc0	crc1	crc2	crc3	crc4	crc5
	crc6	crc7	ecc0	ecc0	ecc0	ecc1	ecc1	ecc1



- 256B Flit, with 3 groups of FEC (interleaved bytes)
- 8B of CRC, 6B of FEC
- FEC covers the CRC bytes as well
- Each FEC is a (86, 84) code
  - Syndrome and Parity bytes added by the encoder
- CRC is a Reed Solomon (250, 242) code

FEC encoder (Flat)

	B_0	⋯	B_{82}	B_{83}	B_{84}	B_{85}
0	α^{84}	⋯	α^{2}	α	1	p_0
1	α^{85}	⋯	α^{3}	α^{2}	α	p_1
2	α^{86}	⋯	α^{4}	α^{3}	α^{2}	p_2
3	α^{87}	⋯	α^{5}	α^{4}	α^{3}	p_3
4	α^{88}	⋯	α^{6}	α^{5}	α^{4}	p_4
5	α^{89}	⋯	α^{7}	α^{6}	α^{5}	p_5
6	α^{90}	⋯	α^{8}	α^{7}	α^{6}	p_6
7	α^{91}	⋯	α^{9}	α^{8}	α^{7}	p_7
					Check Bits	Row Parity [N-3:0]

- Parity byte is a simple XOR of all the bytes
- α is the root of the primitive polynomial ($x^8+x^4+x^3+x^2+1$) that is used to generate this code
- The check bits are computed using the following equation:

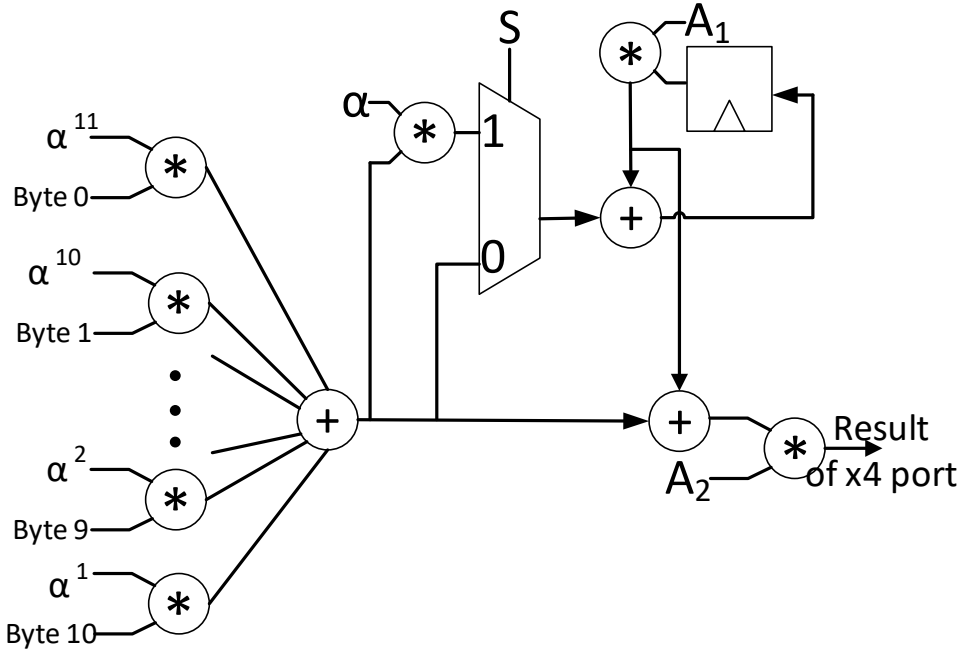
$$C = \sum_{i=0}^{83} B_i \times \alpha^{(84-i)}$$

FEC encoder parity byte (pipelined)

Width	Cycle	FEC Group		
		0	1	2
X16	0	B [42:0]	B [42:0]	B [41:0]
	1	B[83:43]	00h,B[82:43]	00h,B[82:42]
X8	0	B[21:0]	B[20:0]	B[20:0]
	1	B[42:22]	B[42:21]	B[41:21]
	2	B[63:43]	B[63:43]	B[63:42]
	3	B[83:64]	00h,B[82:64]	00h,B[82:64]
X4	0	B[10:0]	B[10:0]	B[9:0]
	1	B[21:11]	B[20:11]	B[20:10]
	2	B[31:22]	B[31:21]	B[31:21]
	3	B[42:32]	B[42:32]	B[41:32]
	4	B[53:43]	B[52:43]	B[52:42]
	5	B[63:54]	B[63:53]	B[63:53]
	6	B[74:64]	B[74:64]	B[73:64]
	7	B[83:75]	00h,B[82:75]	00h,B[82:74]

- Depending on the link sub-division and FEC group, different bytes are available at different clock cycles (shown in the table)
- Parity computation is accumulated every cycle and is ready for consumption after the last chunk of the data is available

FEC encoder check byte (pipelined for x4)



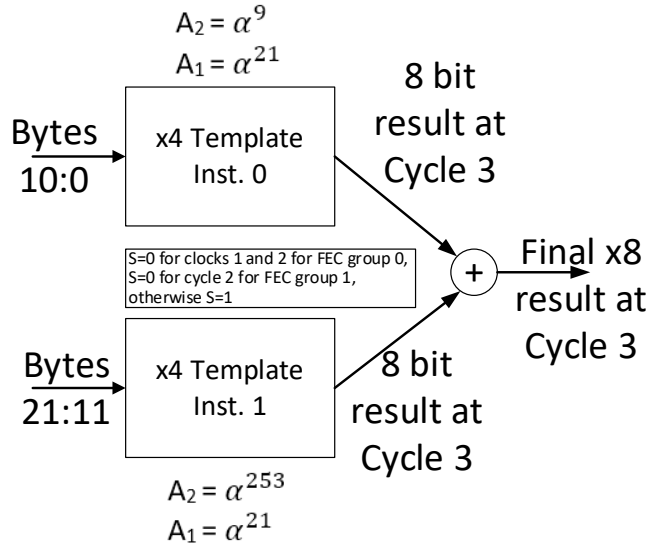
- Key property utilized is that powers of α can be accumulated over a partial result to give an equivalent check value :

$$B_i * \alpha^{x+y} + B_j * \alpha^{z+y} = \left(B_i * \alpha^{x} + B_j * \alpha^{z}\right) * \alpha^{y}$$

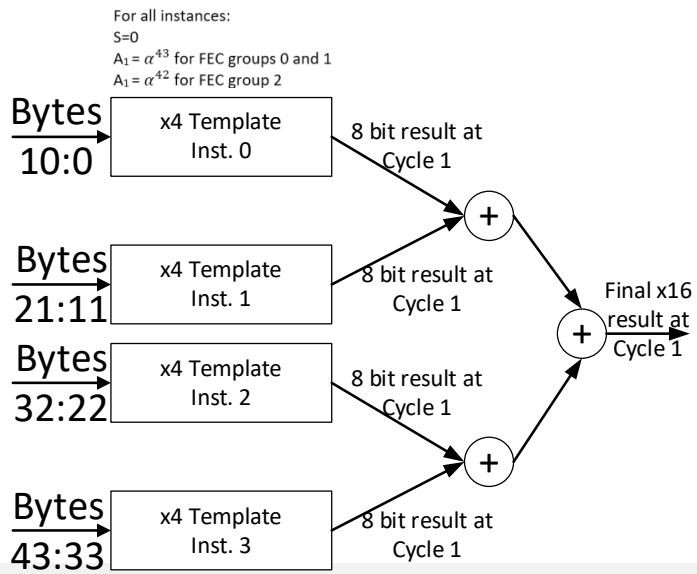
- A_1 and A_2 are parameters that map to specific powers of α and are selected to perform adjustments to account for the slight differences between the data byte ordering for the different FEC groups

FEC encoder check byte (pipelined for x8 and x16)

x8



x16



- Use x4 as a building block
- Multiple instances of them follow as it follows the data path split configuration
- S, A_1 and A_2 are selected to ensure the check computation is consistent and they account for slight differences in the byte order between different FEC groups

FEC Decoder

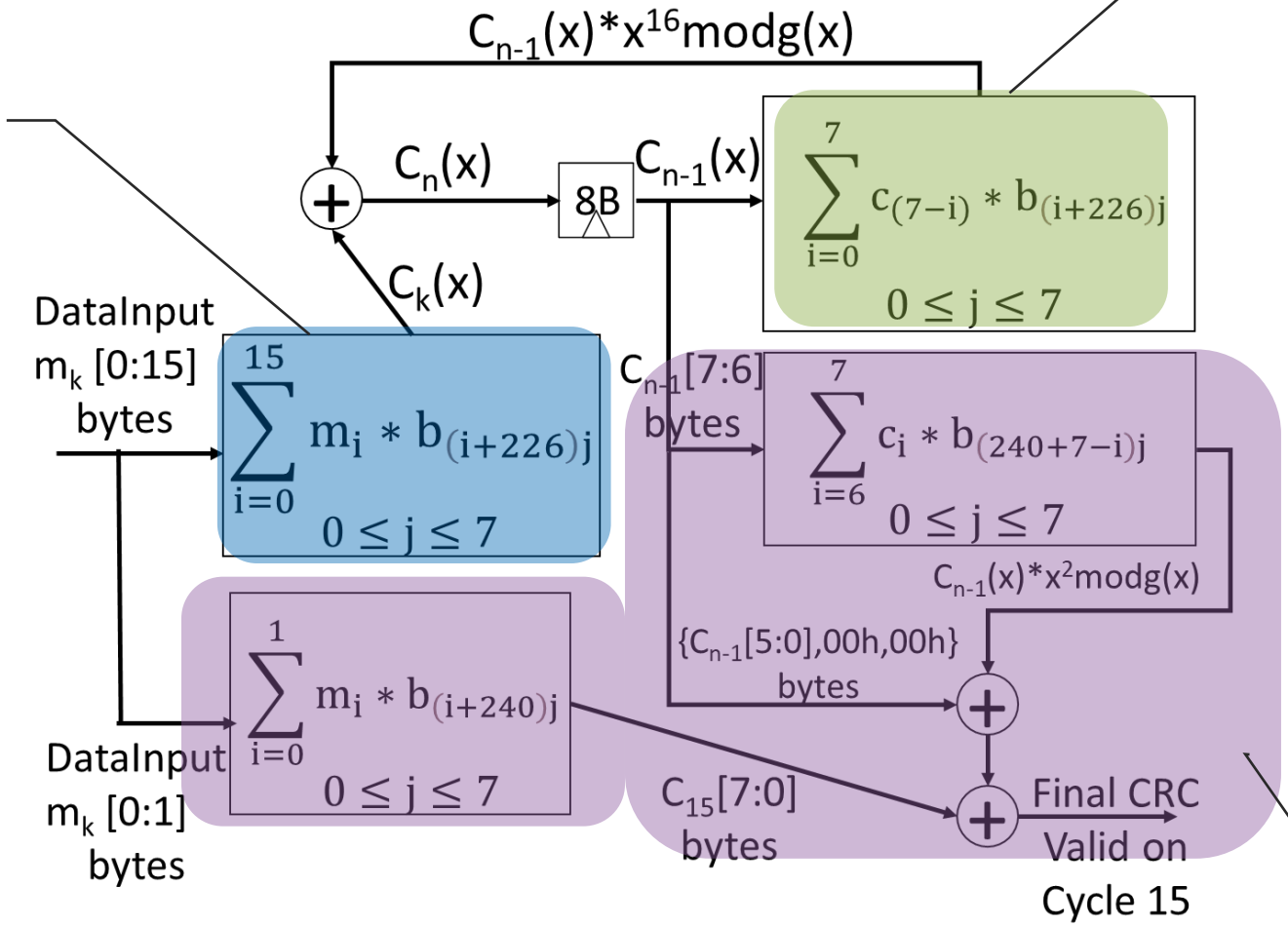
- Syndromes of Parity and Check are computed
  - XOR of computed Parity, Check with the received Parity, Check respectively
- If Data[k] is the true message byte in kth position, and E[k] is the error byte that corrupted the kth position and A[k] is the corresponding power of α
  - Syndrome of Parity = E[k]
  - Syndrome of Check = E[k] * A[k]
- Kth byte is in error if syndrome of Parity * A[k] = Syndrome of Check
- The corresponding Data byte is corrected by an XOR with syndrome of parity

CRC Pipelining

- PCIe uses 8B CRC derived from a Reed Solomon code (250, 242)
- Given its generator matrix, each message symbol's contribution to the CRC is the XOR of the message with the corresponding byte in a specific row from the generator matrix
  - Mathematical properties of polynomial division permit us to operate a message symbol on Row "x" of the generator matrix, followed by a translation operation on the result to get a value that is equivalent to operating the message symbol on Row "y"
- As data bytes stream in over multiple clock cycles
  - Compute intermediate results using a subset of CRC computation and accumulate in eight bytes of flip-flops
  - Final translation operation is performed to get a result equivalent to the Flat implementation (this translation also maps to a subset of CRC computation)

CRC pipelining for a x4 port

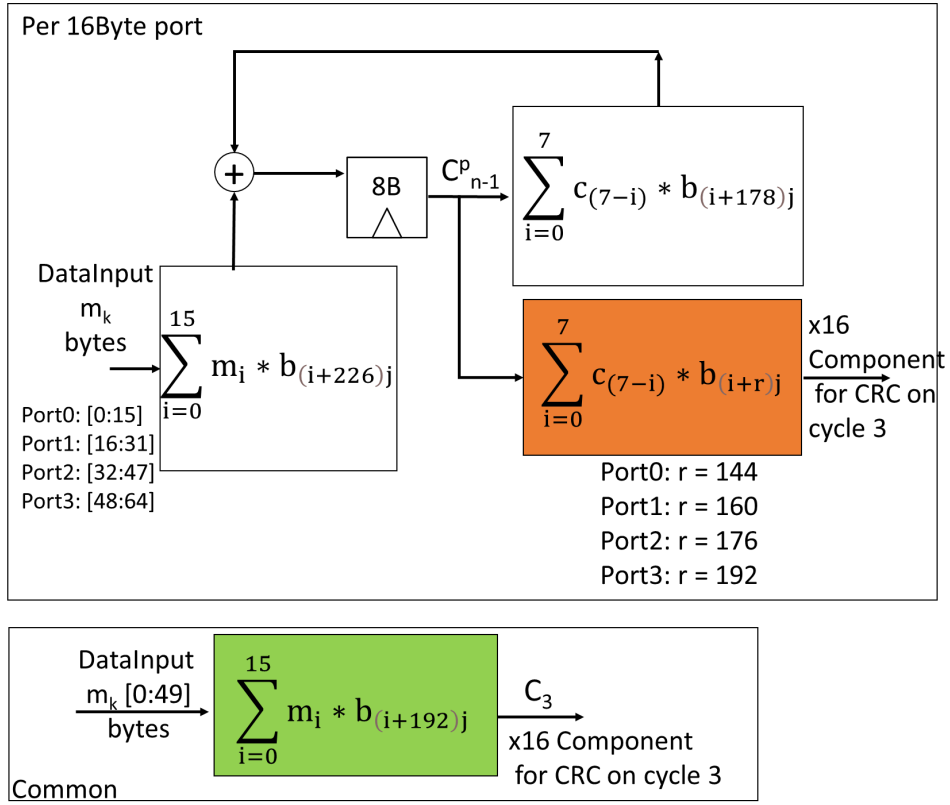
Incoming Bytes for cycles 0 to 14 operate on rows 226:241 of the generator matrix



Intermediate results continue translation and accumulation every clock cycle

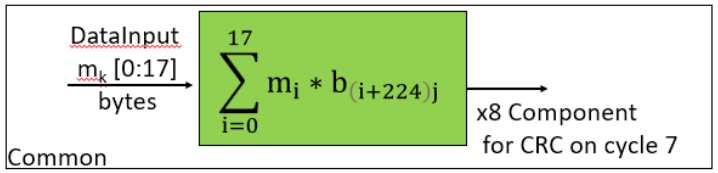
Final computation and adjustment to get CRC value on cycle 15

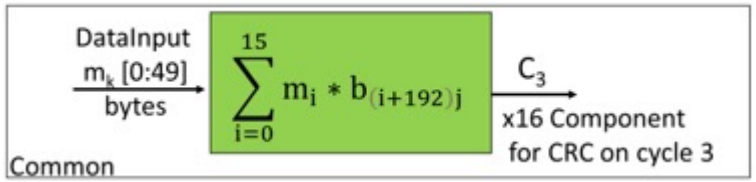
CRC pipelining example of building x16 using x4 component

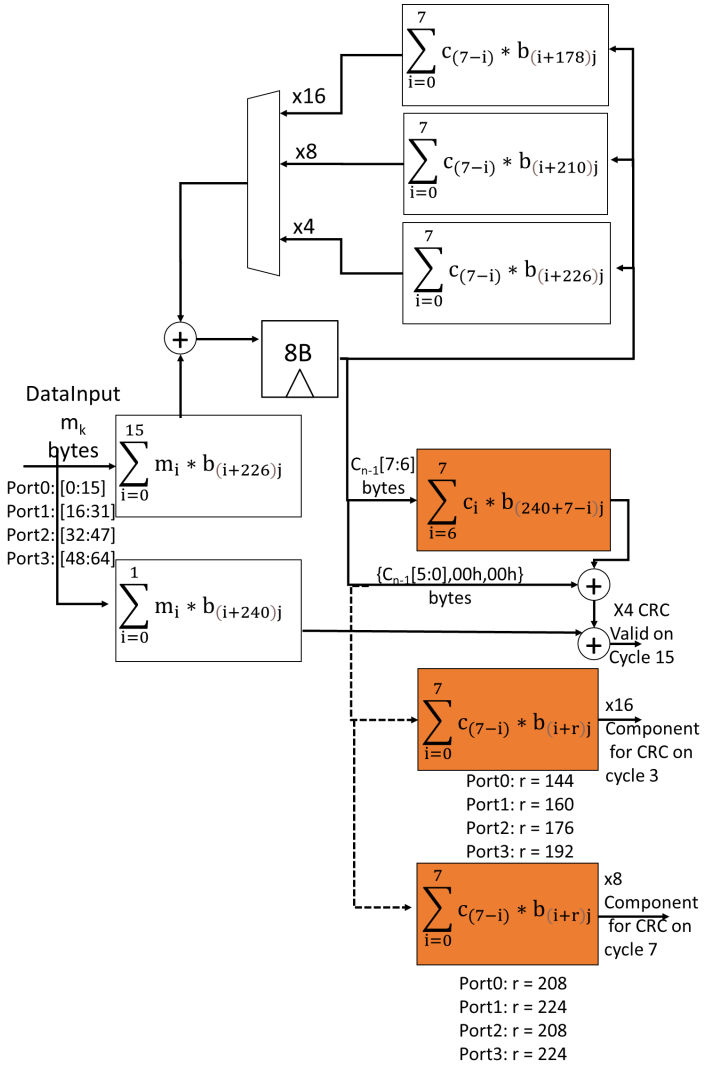


- x16 built using x4 components, but each port gets a different translation computation for the intermediate results
- There is common logic for computing the result of bytes coming in on the last cycle (cycle 3)

CRC pipelined implementation for x4, x8 and x16 together







Results

Logic Levels

	FEC (vs flat)	CRC (vs flat)
Encoder	12 (vs 9)	9 (vs 10)
Decoder	25 (vs 25)	9 (vs 10)

Area Improvement

	Flat logic (single port)	Flat logic (4 ports) ~(4* 1 port)	Pipelined logic (4 ports)	Area improvement
FEC encoder	6,185	24,740	9,619	>2.6x
FEC decoder	25,969	103,876	88,755	>1.17x
CRC Encoder/ decoder	17,887	71,548	28,082	>2.5x

Thank You!