High-Speed
Memory Systems

Spring 2014

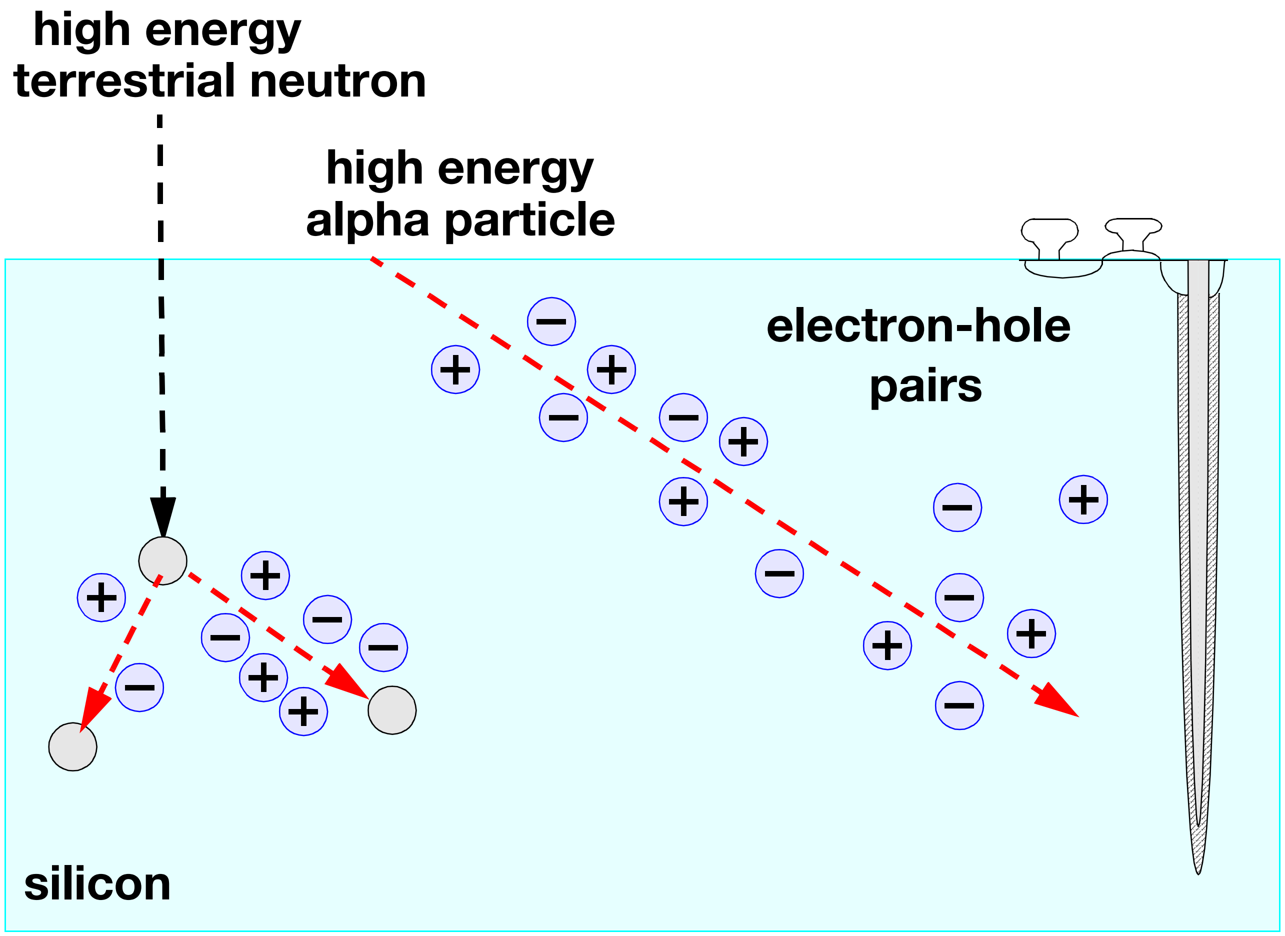
CS-590.26
Lecture F

Bruce Jacob
David Wang

University
of Crete

DRAM Reliability:

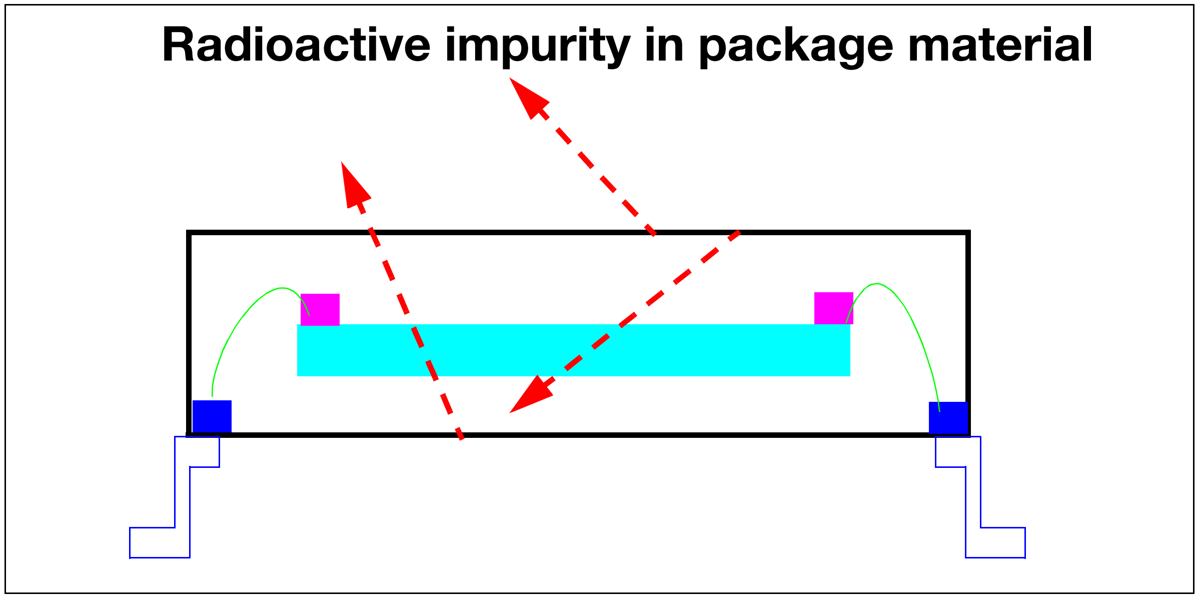
Parity, ECC, Chipkill, Scrubbing



Bruce Jacob
David Wang

University
of Crete

Alpha Particles:

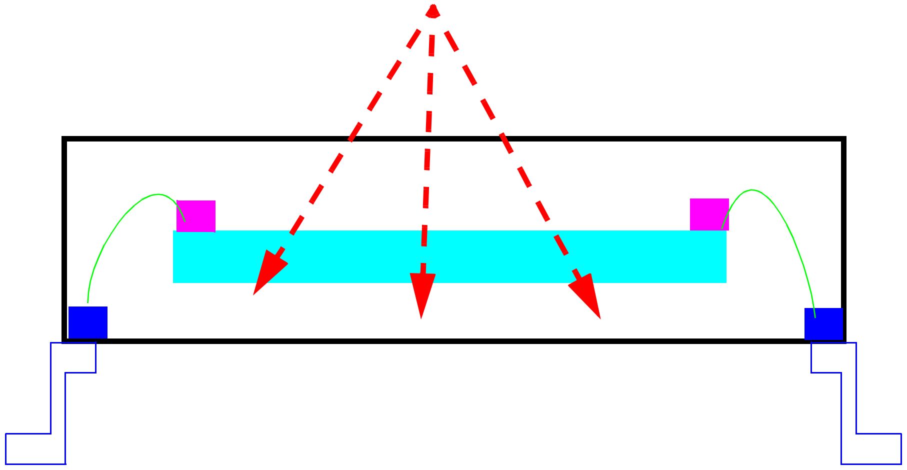


- **Soft errors were big problems for early DRAM chips.**
- **Low energy alpha particles were discovered to be the culprit, but where were they coming from?**
- **Intel published paper in 1979 caused industry to pay close attention to material purity in silicon processing and packaging.**
- **Now largely considered to be “solved problem”**

Bruce Jacob
David Wang
University of Crete

Terrestrial Neutrons:

Space aliens bent on destroying human technology



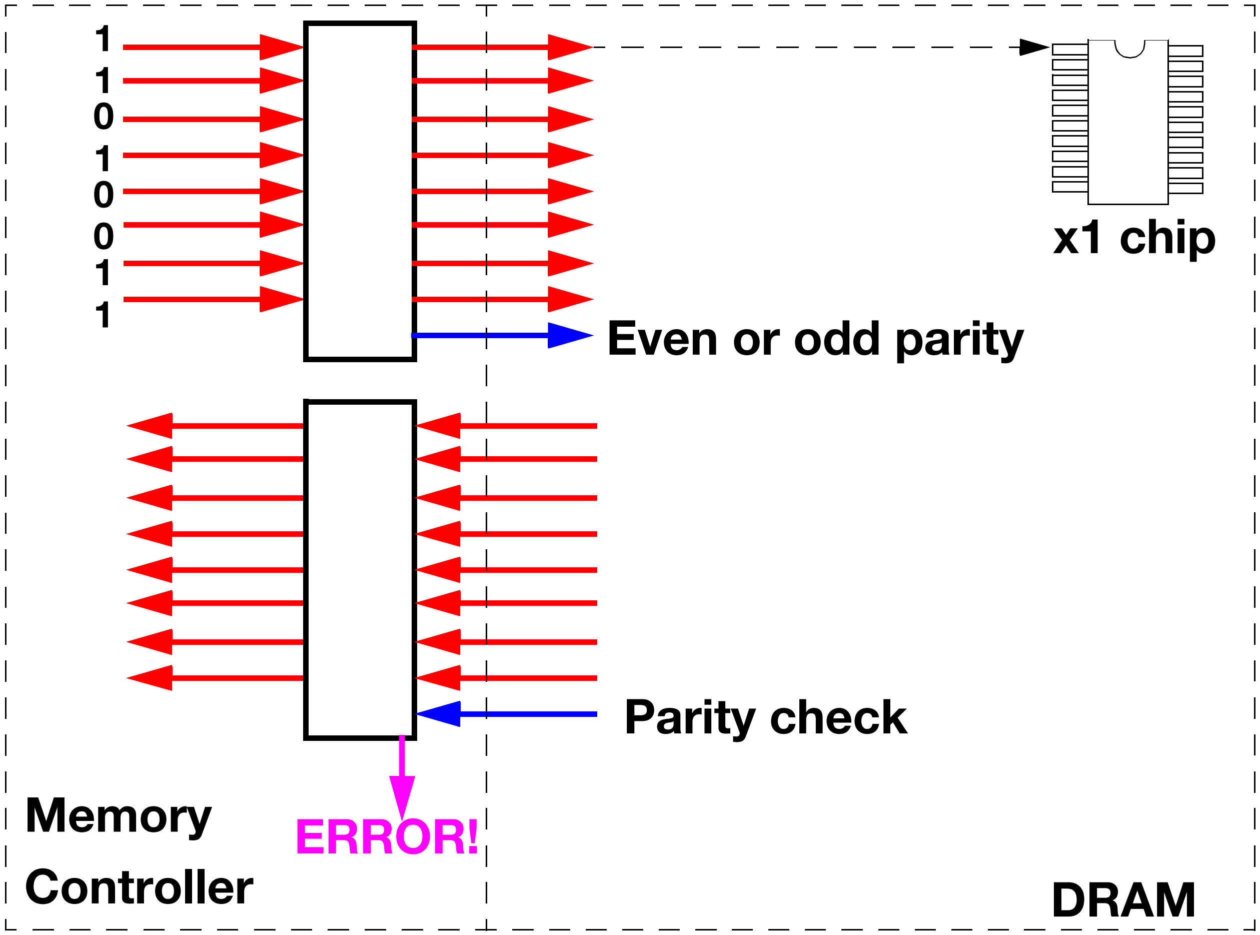
- **High energy cosmic rays originate in space, but ...**
- **collisions with atmosphere generates secondary particles. “Terrestrial Neutrons” main part of flux**
- **Flux of neutrons depend on altitude.**
- **IBM claims 5950 failures per billion device-hours at sea level, 0 failures in underground vault, with 50 feet of rocks completely shielding test setup.**

Bruce Jacob
David Wang

University
of Crete

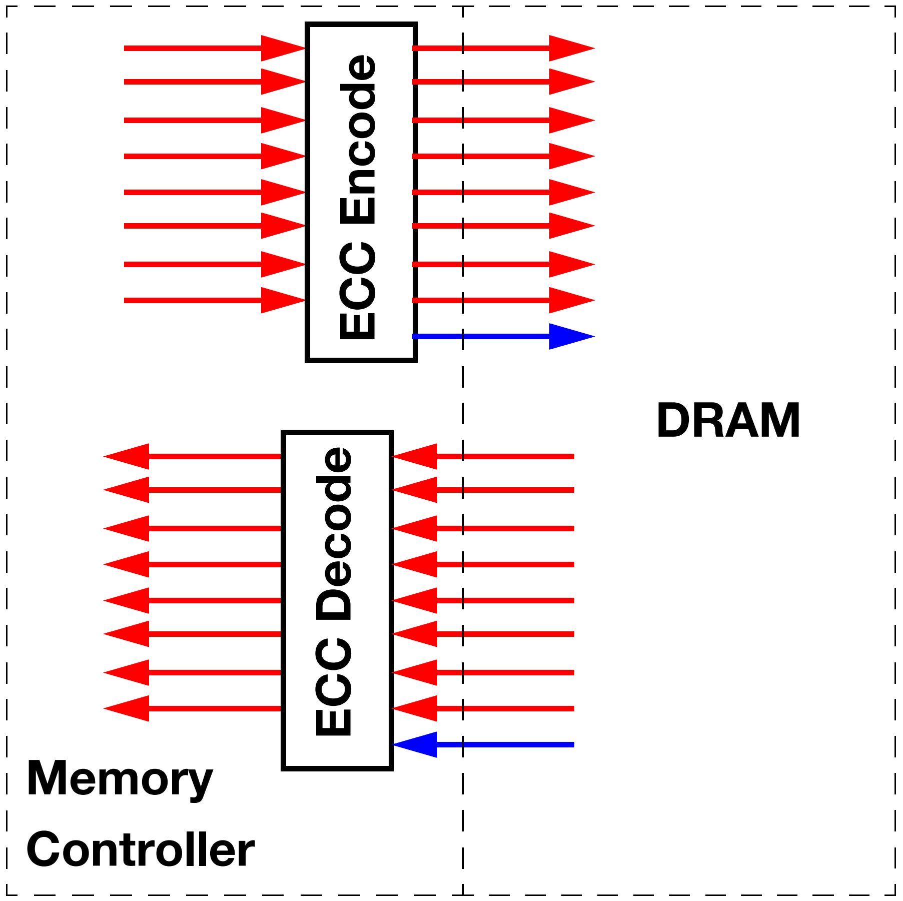
UNIVERSITY OF MARYLAND

Parity: “For Farmers”



- Odd bit error detection
- No error correction capability
- Overhead: 1 bit per byte

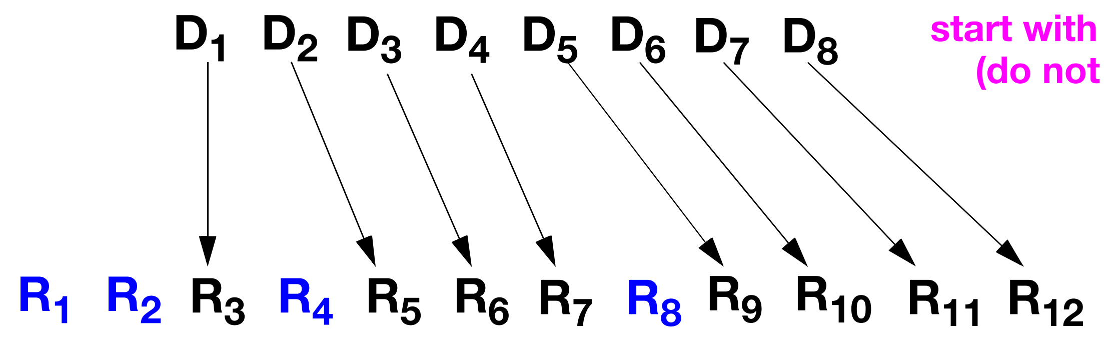
Error Correcting Code I



- Also based on "parity checking", but more sophisticated
- Error detection AND correction capability
- Overhead: depending on scheme

Error Correcting Code IIa

Single-bit Error Correction (SEC)


D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9 R_{10} R_{11} R_{12}

start with 8 data bits (do not use D_0)

Reserve R_m bit positions where m is a power of 2.
Move data bits into available bit positions. (skip R_0)
Display “m” in binary format.

R_{0001} R_{0010} R_{0011} R_{0100} R_{0101} R_{0110} R_{0111} R_{1000} R_{1001} R_{1010} R_{1011} R_{1100}

R_m bit positions will be the check bits, where each R_m bit will store the parity of the other bit positions where the m^{th} bit in the index is set

$$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011}$$
$$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011}$$

- requires n+1 check bits to provide SEC to 2^n data bits

UNIVERSITY OF MARYLAND

Error Correcting Code IIb

SEC Encoding Example

D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8

1 1 0 0 1 1 1 0

start with 8 data bits (do not use D_0)

R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9 R_{10} R_{11} R_{12}

Reserve R_m bit positions where m is a power of 2.
Move data bits into available bit positions. (skip R_0)
Display “m” in binary format.

R_{0001} R_{0010} 1 R_{0100} 1 0 0 R_{1000} 1 1 1 0

R_m bit positions will be the check bits, where each R_m bit will store the parity of the other bit positions where the m^th bit in the index is set

$$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$$
$$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$
$$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$\boxed{D = \{1\,1\,0\,0\,1\,1\,1\,0\} \longrightarrow R = \{0\,1\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\}}$$

UNIVERSITY OF MARYLAND

Error Correcting Code IIc

SEC Verification Example

R = { 0 1 1 0 1 0 0 1 1 1 1 0 }

R = { 0 1 1 0 1 0 0 1 1 1 0 0 }

One bit error. Can we detect and correct?

Recompute check bits

$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$

$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 0 = 0$

$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

XOR old check bits against new check bits

	R_{1000}	R_{0100}	R_{0010}	R_{0001}	
	1	0	1	0	Old
$\oplus$	0	0	0	1	New
	1	0	1	1	Difference ! Bit position 11 is rotten

Error Correcting Code IIIa

What about multi-bit errors?

R = { 0 1 1 0 1 0 0 1 1 1 1 0 }

R = { 0 1 1 0 1 0 0 1 1 1 0 1 }

Multi bit error. Can we detect and correct?

Recompute check bits

$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$

$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 1 = 1$

$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

XOR old check bits against new check bits

	R_{1000}	R_{0100}	R_{0010}	R_{0001}	
	1	0	1	0	Old
$\oplus$	1	1	0	1	New
	0	1	1	1	Difference !

Oops, Bit position 7 is NOT rotten

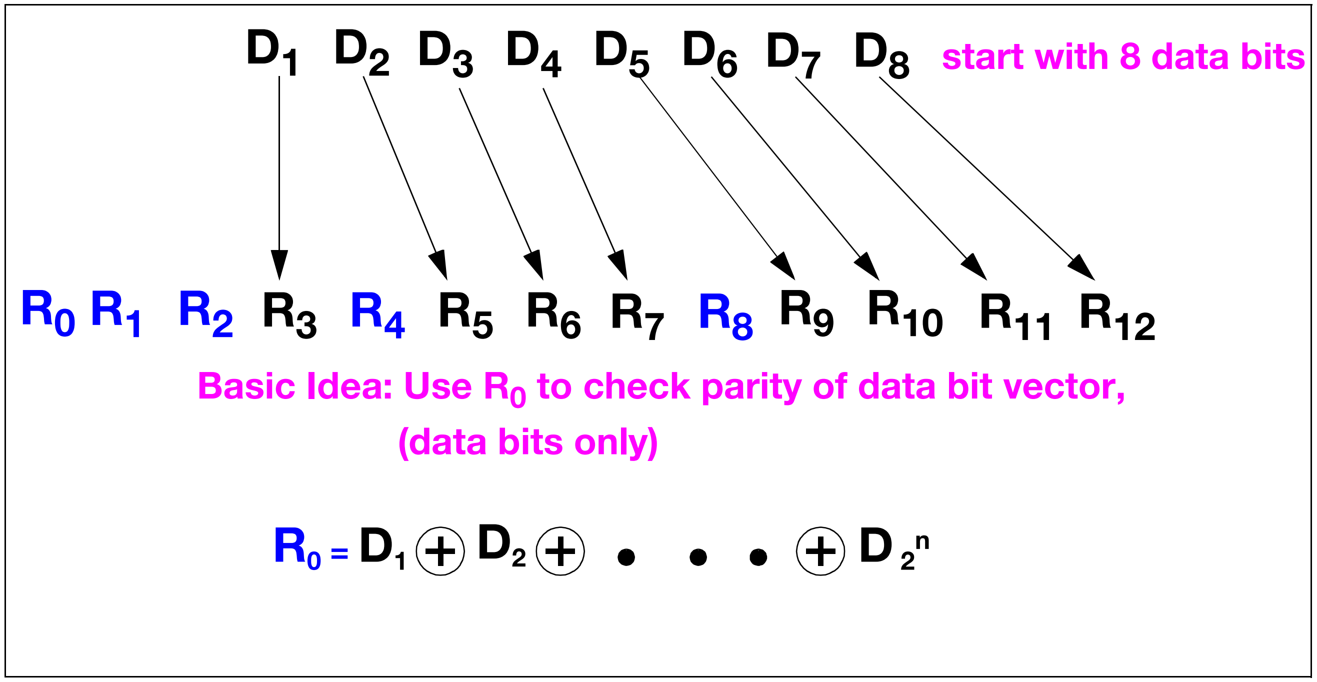
UNIVERSITY OF MARYLAND

UNIVERSITY OF MARYLAND

Error Correcting Code IIIb

What about multi-bit errors?

Single Error Correction Double Error Detection (SECDED)


D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8
start with 8 data bits
R_0 R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9 R_{10} R_{11} R_{12}
Basic Idea: Use R_0 to check parity of data bit vector, (data bits only)

$$R_0 = D_1 \oplus D_2 \oplus \bullet \bullet \bullet \oplus D_{2^n}$$

- requires n+2 check bits to provide **SECDED** to 2^n data bits

Error Correcting Code IIIc

What about multi-bit errors - Redux

R = { 1 0 1 1 0 1 0 0 1 1 1 1 0 }

R = { 1 0 1 1 0 1 0 0 1 1 1 0 1 }

Multi bit error. Can we detect and correct?

Recompute check bits

$R_{0001} = R_{0011} \oplus R_{0101} \oplus R_{0111} \oplus R_{1001} \oplus R_{1011} = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$

$R_{0010} = R_{0011} \oplus R_{0110} \oplus R_{0111} \oplus R_{1010} \oplus R_{1011} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

$R_{0100} = R_{0101} \oplus R_{0110} \oplus R_{0111} \oplus R_{1100} = 1 \oplus 0 \oplus 1 \oplus 1 = 1$

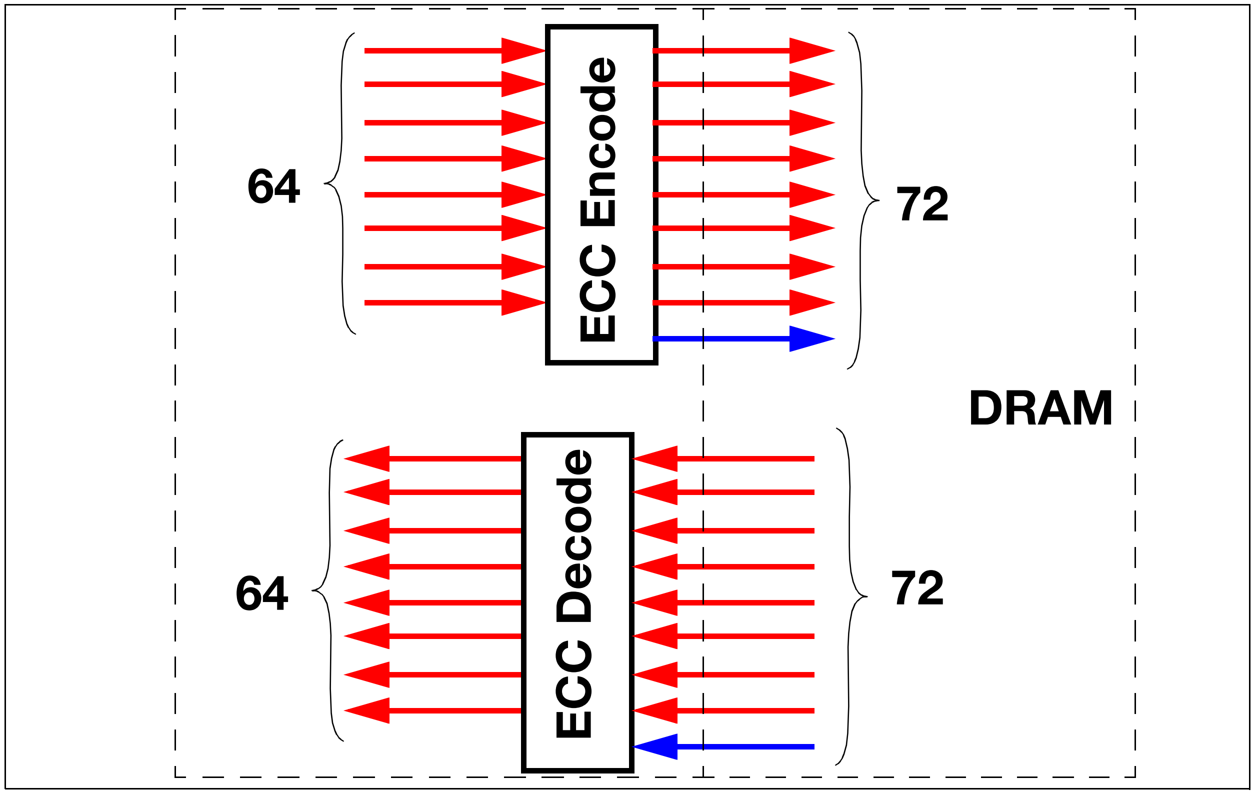
$R_{1000} = R_{1001} \oplus R_{1010} \oplus R_{1011} \oplus R_{1100} = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

XOR old check bits against new check bits

	R_{1000}	R_{0100}	R_{0010}	R_{0001}	
	1	0	1	0	Old
$\oplus$	1	1	0	1	New
	0	1	1	1	Difference !

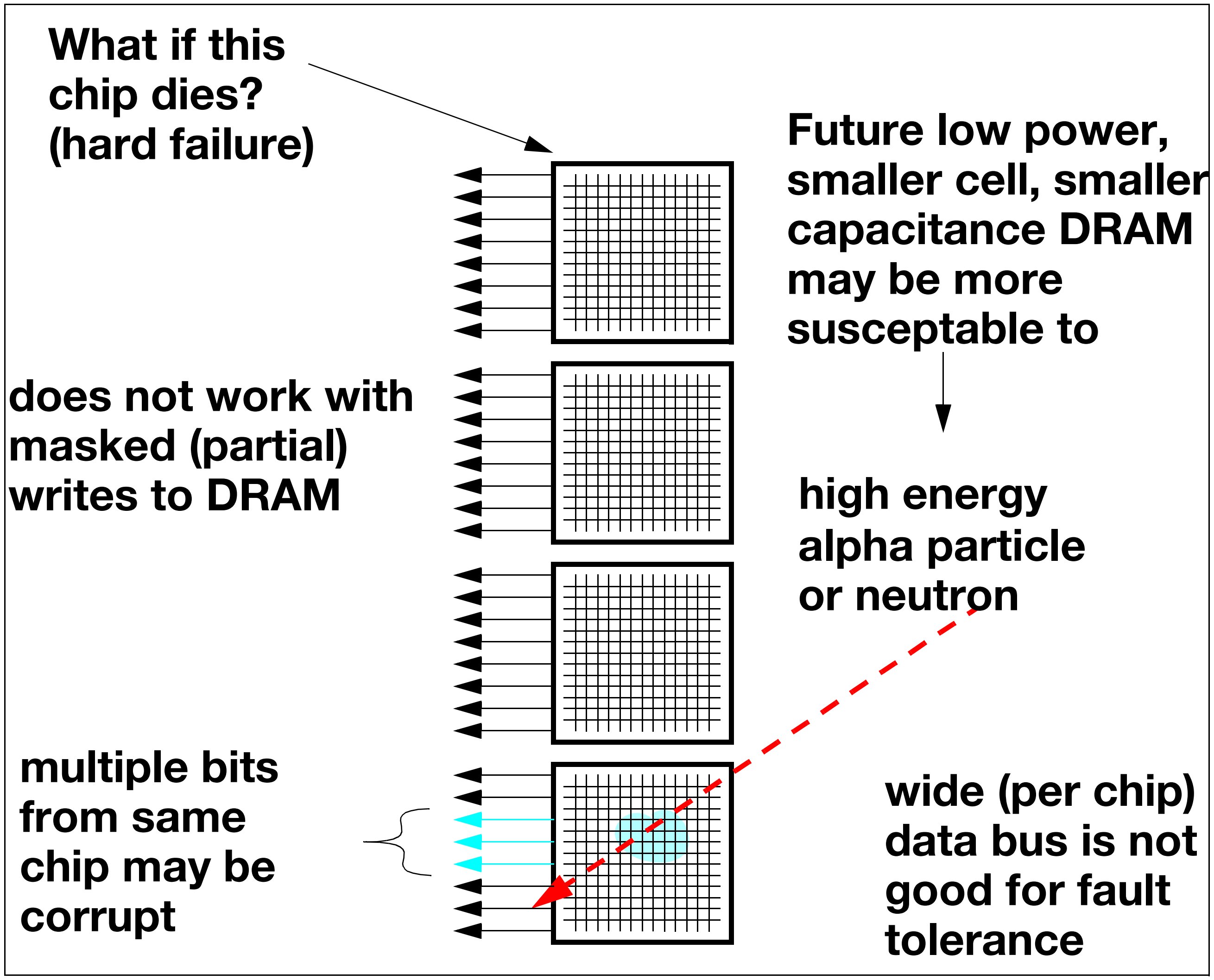
XOR check bits tell us there is error, but R_0 parity says all is well. This is a 2 bit error, cannot be corrected.

Error Correcting Code IV



- **SECDED needs n + 2 check bits to protect 2^n data bits**
- **Data bus width of 64 = 2^6 means 6 + 2 = 8 check bits to provide SECDED protection**
- **Logic depth of n + 1 = 7 to compute XOR parity for 0th bit**
- **May cost additional cycle(s) on read latency**

Weaknesses of ECC?


What if this chip dies? (hard failure)
Future low power, smaller cell, smaller capacitance DRAM may be more susceptable to
high energy alpha particle or neutron
does not work with masked (partial) writes to DRAM
multiple bits from same chip may be corrupt
wide (per chip) data bus is not good for fault tolerance

Error rate is given in failures per bit. There are always more DRAM storage bits in the next generation system.

Bruce Jacob
David Wang

University of Maryland
ECE Dept.

Multi-bit Error Correction I

$$0 = \begin{bmatrix}0\\0\end{bmatrix} \quad 1 = \begin{bmatrix}0\\1\end{bmatrix} \quad \alpha = \begin{bmatrix}1\\0\end{bmatrix} \quad \alpha^2 = \begin{bmatrix}1\\1\end{bmatrix}$$

$$\begin{bmatrix}C_{\alpha^2}\\C_{\alpha}\\C_1\\C_0\end{bmatrix} = \begin{bmatrix}
1&1&1&\alpha^2&\alpha&1&1&1&\alpha^2&\alpha&0&0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&\alpha^2&\alpha&1\\
\alpha^2&\alpha&1&0&0&0&1&1&1&1&\alpha^2&\alpha&0&0&1&1&1&1&0&1&0&0&0&\alpha^2&\alpha&1&1&1&1&0&0&0\\
0&0&0&1&1&1&\alpha&\alpha^2&1&1&1&1&\alpha^2&\alpha&0&0&0&1&1&1&1&1&1&0&0&0&\alpha^2&\alpha&1&0&0&0\\
0&0&0&0&0&0&\alpha^2&\alpha&0&0&1&1&1&1&\alpha^2&\alpha&1&0&1&1&\alpha^2&\alpha&1&1&1&1&0&0&0&1&1&1
\end{bmatrix}$$

Parity check matrix in GF(2^2)

Apply transform matrices

Parity check matrix in binary field

$$T_0 = \begin{bmatrix}0&0\\0&0\end{bmatrix} \quad T_1 = \begin{bmatrix}1&0\\0&1\end{bmatrix} \quad T_{\alpha} = \begin{bmatrix}1&1\\1&0\end{bmatrix} \quad T_{\alpha^2} = \begin{bmatrix}0&1\\1&1\end{bmatrix}$$

$$\begin{bmatrix}C_7\\C_6\\C_5\\C_4\\C_3\\C_2\\C_1\\C_0\end{bmatrix} = \begin{bmatrix}
1&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0\\
0&1&0&1&0&1&1&1&1&0&0&0&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&1\\
0&1&1&1&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0\\
1&1&1&0&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&1&0&0&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&1&0&0&0&1&0&1&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1
\end{bmatrix}$$

$$\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc}
0&&&&&&&&8&&&&&&&&16&&&&&&&&24&&&&&&&31&32&&&&&&&&40&&&&&&&&48&&&&&&&&56&&&&&&&63
\end{array}$$

$$\begin{bmatrix} C_7 \\ C_6 \\ C_5 \\ C_4 \\ C_3 \\ C_2 \\ C_1 \\ C_0 \end{bmatrix} = \left[\begin{array}{cccccccccccccccccccccccccccccccc|c|ccccccccccccccccccccccccccccccc}
1&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0 \\
0&1&0&1&0&1&1&1&1&0&0&0&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&1 \\
0&1&1&1&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0 \\
1&1&1&0&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&1&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&1&0&0&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&1&0&0&0&1&0&1&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1 \\
0&&&&&&&&8&&&&&&&&16&&&&&&&&24&&&&&&&31&32&&&&&&&&40&&&&&&&&48&&&&&&&&56&&&&&&&63
\end{array}\right]$$

$$\begin{bmatrix} C_7 \\ C_6 \\ C_5 \\ C_4 \\ C_3 \\ C_2 \\ C_1 \\ C_0 \end{bmatrix} = \left[\begin{array}{cccccccccccccccccccccccccccccccc|cc|cccccccccccccccccccccccccccccc}
1&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0 \\
0&1&0&1&0&1&1&1&1&0&0&0&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&1 \\
0&1&1&1&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0 \\
1&1&1&0&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&1&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&1&0&1&0&1&1&0&1&1&0&1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&1&1&1&0&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&0&1&0&1&0&1&0&0&1&1&1&1&0&0&0&1&0&1&0&0&1&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&0&1&0&1&0&1&1&1&1&0&0&1&0&0&0&1&0&1&1&1&1&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1 \\
0&&&&&&&&8&&&&&&&&16&&&&&&&&24&&&&&&&31&32&&&&&&&&40&&&&&&&&48&&&&&&&&56&&&&&&&63
\end{array}\right]$$

FIGURE 30.12: Locating a single bit and 2-adjacent bit errors in a 64-bit word.

A two-bit error in positions 32,33 results in 11110011

Table 30.3 Error location table for the 2-adjacent error correction algorithm, taken from US Patent #5,490,155 (Compaq's Advanced ECC implementation)

				S7:	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				S6:	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
s	s	s	s	s5:	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
3	2	1	0	s4:	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0			C4	C5		C6	5	3	1	C7	0	4	2		2,3	0.1	4,5
0	0	0	1		C0	51	49	47	63	33			61		28		59			30,31
0	0	1	0		C1	46	50	48	58	31			62		32		60			28,29
0	0	1	1			48,49	46,47	50,51	60,61	29			58,59				62,63			32,33
0	1	0	0		C2	57	52	54,55	11	35			9	19			7	17		
0	1	0	1		45	39	23	21	37											
0	1	1	0		43								24							12,13
0	1	1	1		41										14		26,27			
1	0	0	0		C3	55	56	52,53	6		16		10		34		8		18	
1	0	0	1		40				27											14,15
1	0	1	0		44	20	38	22					36							
1	0	1	1		42												24,25			
1	1	0	0			53	54	56,57	8,9			18,19	6,7			16,17	10,11			34,35
1	1	0	1		42,43				25						12					
1	1	1	0		40,41					15			26							
1	1	1	1		44,45	22,23	20,21	38,39									36,37			

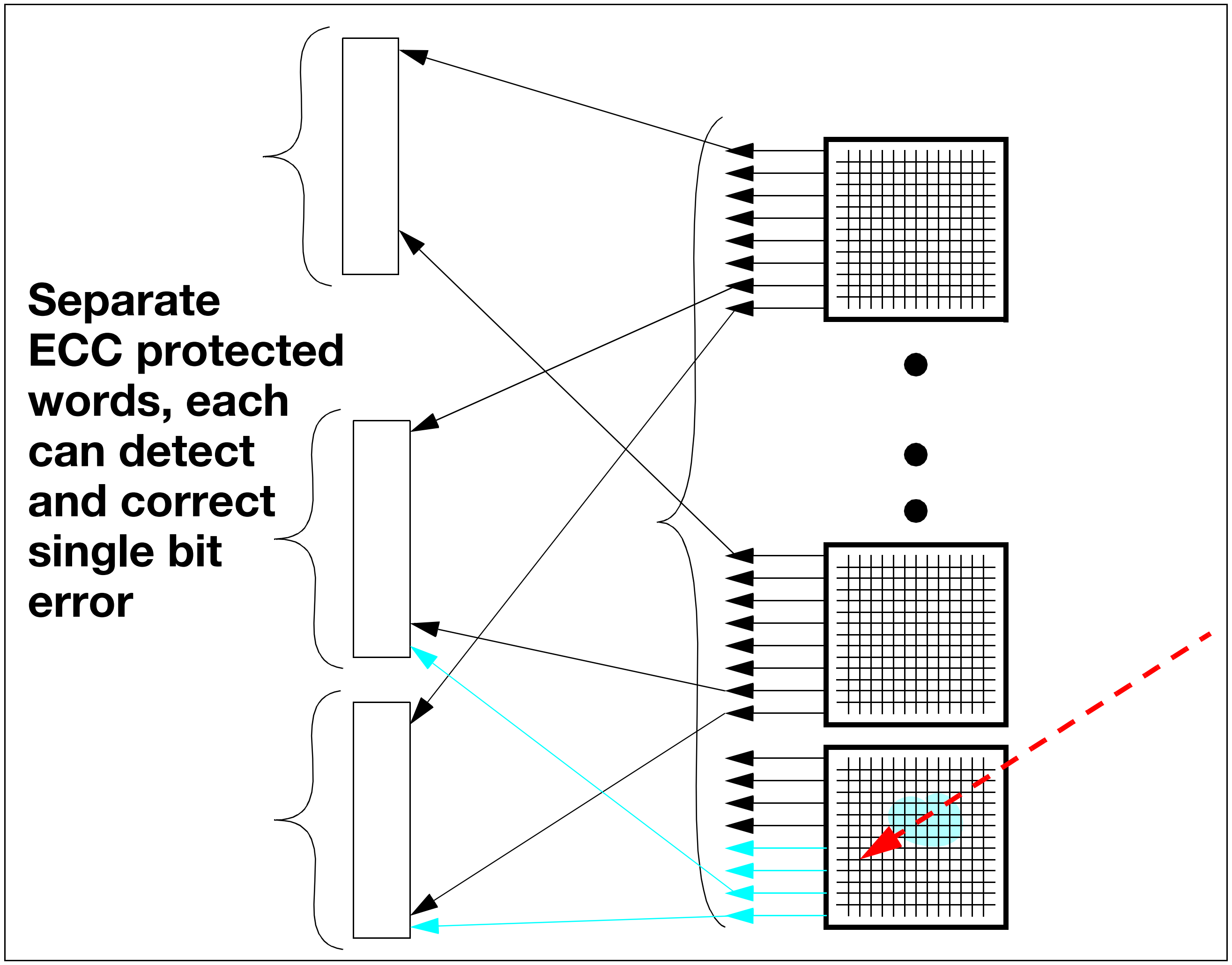
Syndrome of 11110011 points to bad bits 32,33

Multi-bit Error Correction II

- **Each pair of bit positions treated as a single symbol.**
- **Combine with bit steering to cover failure across address boundaries.**
- **Different algorithms exist with varying level of complexity**
- **Should try to work with established framework of (64, 72) DIMMs.**
- **Else, custom memory modules for specialized systems**

UNIVERSITY OF MARYLAND

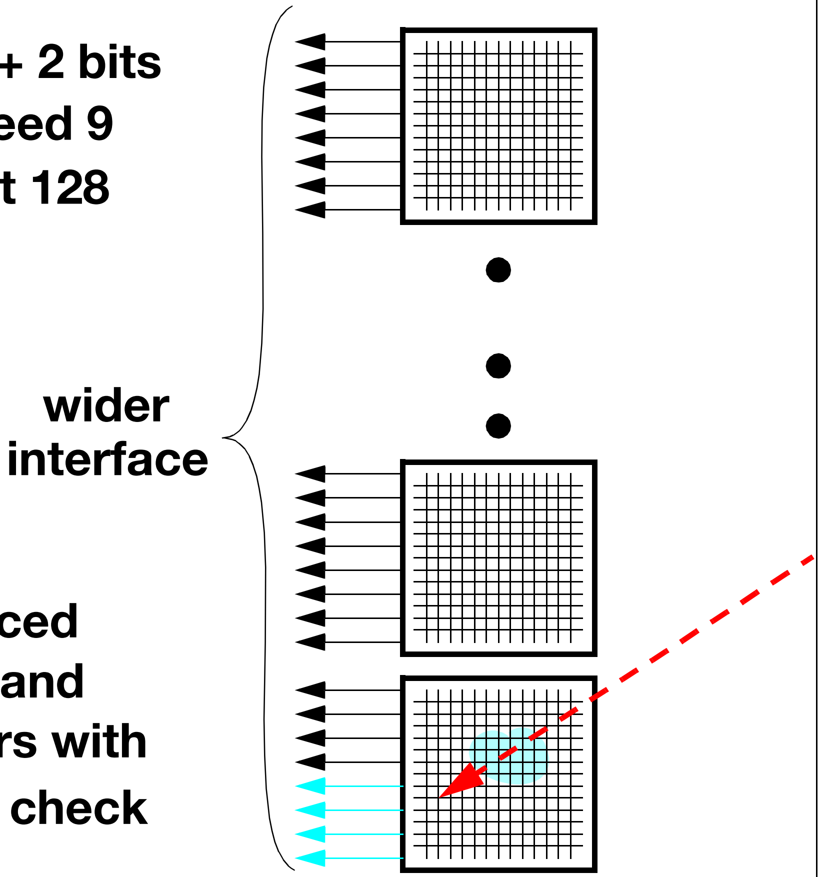
“Chipkill” I



Architect the memory system so there is no Single Point of Failure that could bring down the system

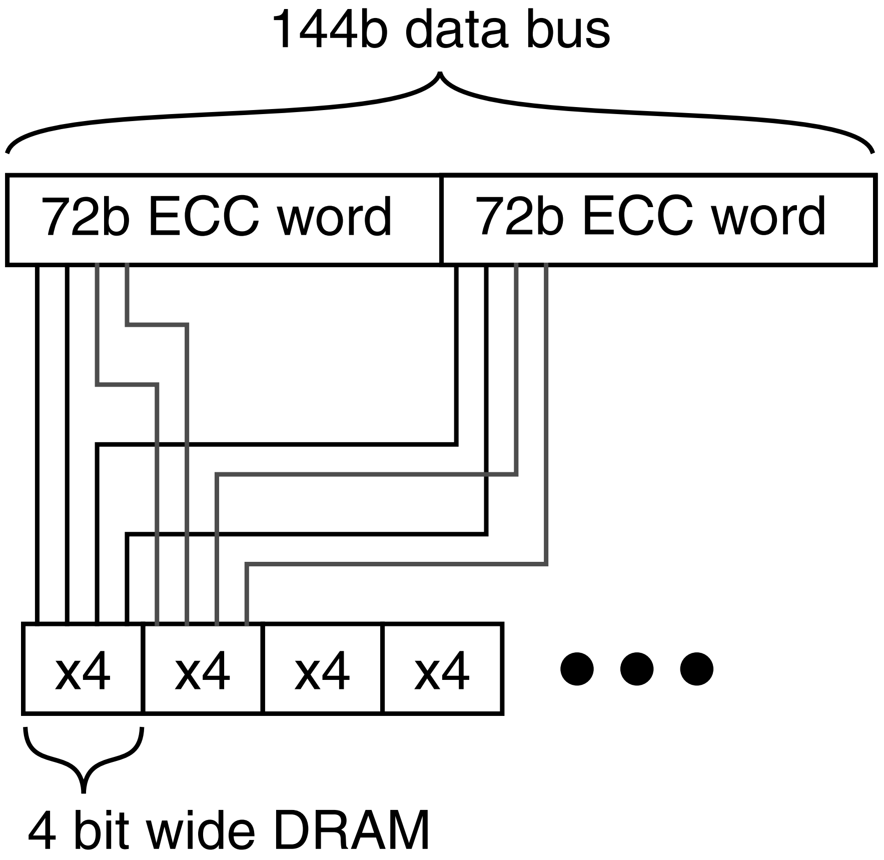
“Chipkill” II

SECDED requires n + 2 bits to protect 2^n bits. Need 9 check bits to protect 128 data bits.



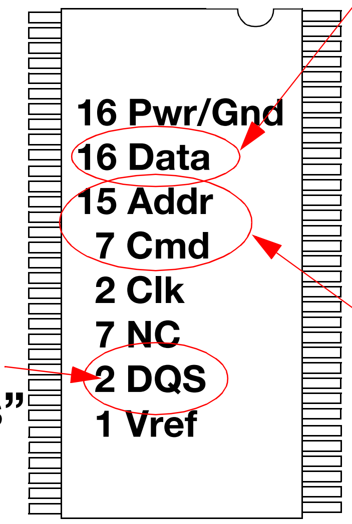
Deploy more advanced algorithm to detect and repair multi-bit errors with 128 data bits and 16 check bits, or 256:32.

Architect the memory system so there is **no Single Point of Failure** that could bring down the system. Deploy method 1, method 2, or combination of both to protect against multi-bit errors



Bit-Steering

Problems Remain



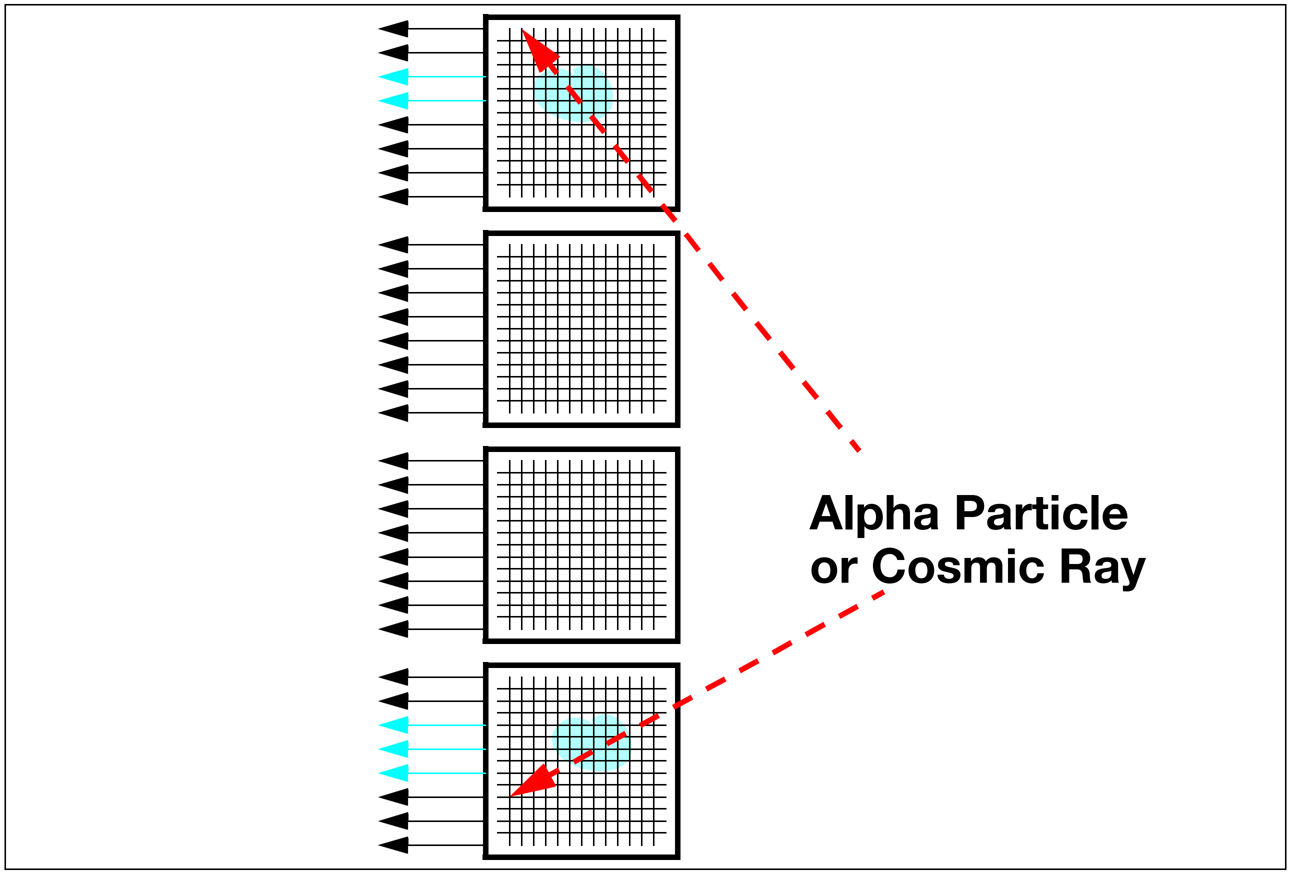
ECC/Chipkill protects data transmission error in addition to **S**ingle **E**vent **U**pset storage error.

Address/command transmission errors are not protected

DQS is per byte for the x8 and x16 chip. “topology matched, source synchronous” is a problem for certain types of chipkill implementation

DDR SDRAM

Scrubbing



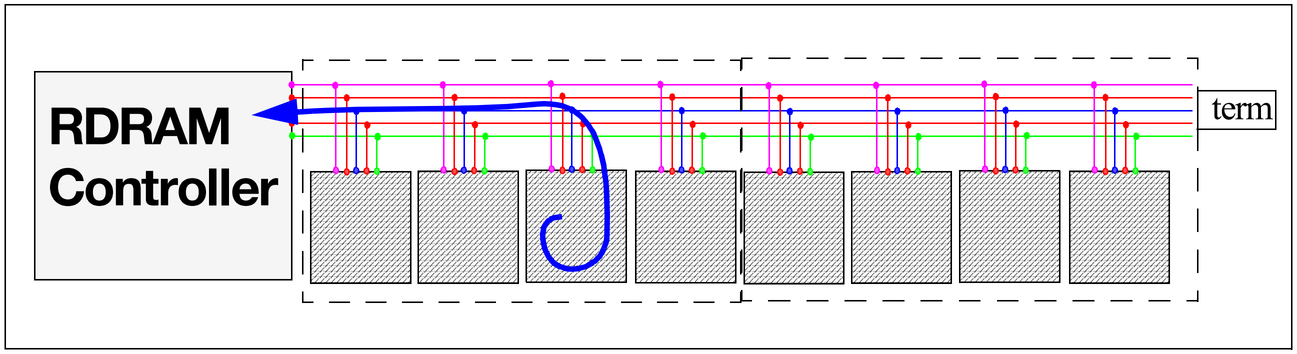
Soft error model based on **S**ingle **E**vent **U**pset
alpha particles or cosmic rays.

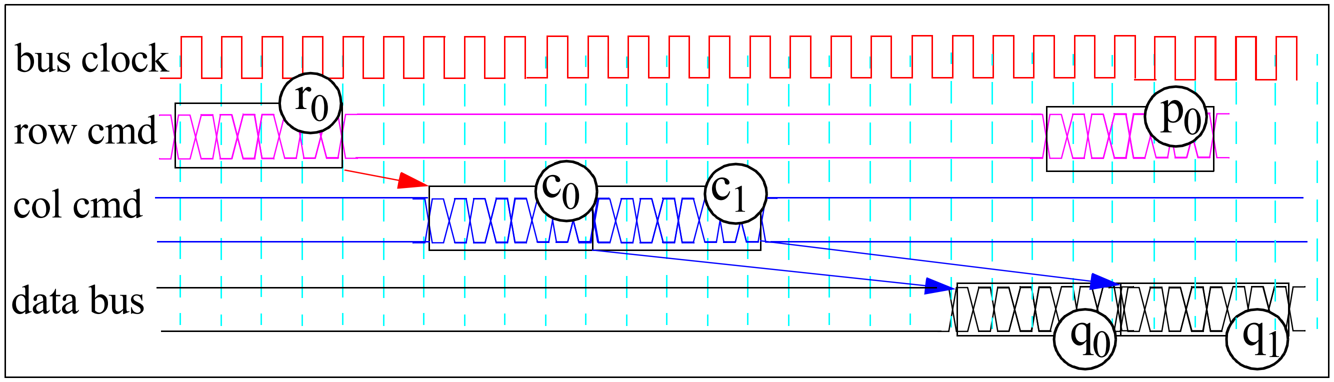
“Scrubbing” merely reads out data to controller, scrub out any correctable error(s), write it back into memory before multi-bit errors build up and become no longer correctable

Serverworks Grand Champion HE

- 128 bit ECC algorithm. 16 bit detection, 8 bit correction.
- Memory scrubbing
- Spare memory
- Memory mirroring
- Hot plug memory card

What about Rambus?





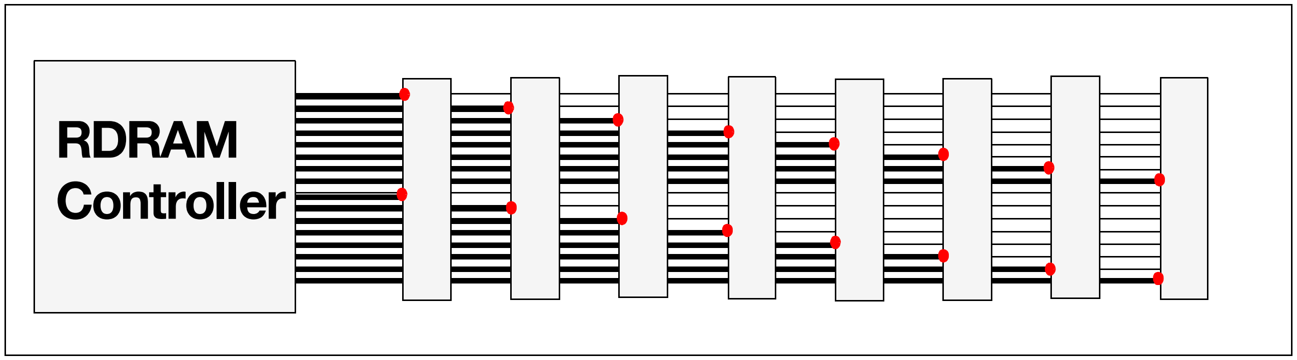
Each "access" to DRAM is serviced by a single DRAM chip. One DRAM chip will provide 8 consecutive beats of data, 16 bit wide per beat.

- Design ECC version, with 18 bit wide interface. provides SECDED protection, not chipkill

Bruce Jacob
David Wang

University
of Crete

Interleaved Device Mode



- Each chip provides 2 bits of data for every read request
- Provides effective chipkill capability when used in multiple channel configuration