EECS150 - Digital Design

Lecture 26 - Faults and Error Correction

April 25, 2013

John Wawrzynek

Types of Faults in Digital Designs

- Design Bugs (function, timing, power draw)
  - detected and corrected at design time through testing and verification (simulation, static checks)
  - Sometimes in 3rd party design blocks
- Manufacturing Defects (violation of design rules, impurities in processing, statistical variations)
  - post production testing for sorting
  - spare on-chip resources for repair
- Runtime Failures (physical effects and environmental conditions)
  - assuming design is correct and no manufacturing defects

Runtime Faults

- All digital systems suffer occasional runtime faults.
  - Fault tolerant design methodologies are employed to tolerate faults in critical applications (avionics, space exploration, medical, …)
  - Error detection and correction is commonly used in memory systems and communication networks.
- Deeply scaled CMOS devices will suffer reliability problems due to a variety of physical effects (processing, aging, environmental susceptibility)
  - Lower supply voltage for energy efficiency makes matters worse.

Physical Fault Mechanisms

- IC Faults can be classified as permanent, transient, and intermittent:
  - Permanent faults reflect irreversible physical changes (like fused wire or shorted transistor)
  - Transients are induced by temporary environmental conditions (like cosmic rays and electromagnetic interference)
  - Intermittent faults occur due to unstable or marginal hardware (temporary ΔV_t resulting in timing error)
- Intermittent faults often occurs repeatedly at the same location while transients affect random locations.
- Intermittents often occur in bursts.
- Intermittent faults track changes in voltage and temperature, and may become permanent.

Physical Fault Mechanisms

- Intermittent: Aging, Voltage/Temperature Dependent
  - NBTI (negative bias temperature instability) & PBTI
  - HCI (hot carrier injection)
  - TDDB (time-dependent dielectric breakdown)
  - Electromigration

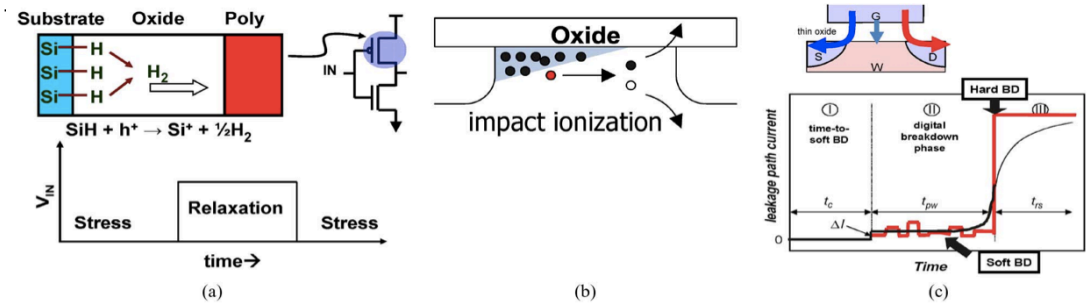


Fig. 5. *Temporal variations: (a) NBTI degradation process in PMOS. Breaking of hydrogen bonds creates dangling Si that acts as a defect trap near Si-SiO_2 interface—increasing V_{TH} of the transistor. V_{TH} degradation and recovery mechanism under NBTI stress is also shown. (b) Impact ionization due to HCI. (c) Percolation path due to TDDB. The behavior of leakage current after soft and hard breakdown is also plotted.*

Physical Fault Mechanisms

- NBTI, PBTI, & HCI increase Vt and decrease mobility
  - Leads to decreased performance, lower noise margins, mismatching (in SRAM), …
- TDDB causes soft or hard gate shorts resulting in degraded transistor performance and can lead to complete transistor failure
- Electromigration reduces interconnect conductivity and can lead to open circuit.

[Ghosh and Roy: Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era, Proceedings of the IEEE | Vol. 98, No. 10, October 2010]

Single Event Effects on digital integrated circuits: Origins and Mitigation Techniques

Dr. Raoul Velazco
TIMA Laboratory
ARIS (Reliable Architectures of Integrated Systems)
Grenoble – France
http://tima.imag.fr
raoul.velazco@imag.fr

Ecole de Microélectronique et Microsystèmes
Fréjus, 19/5/2011

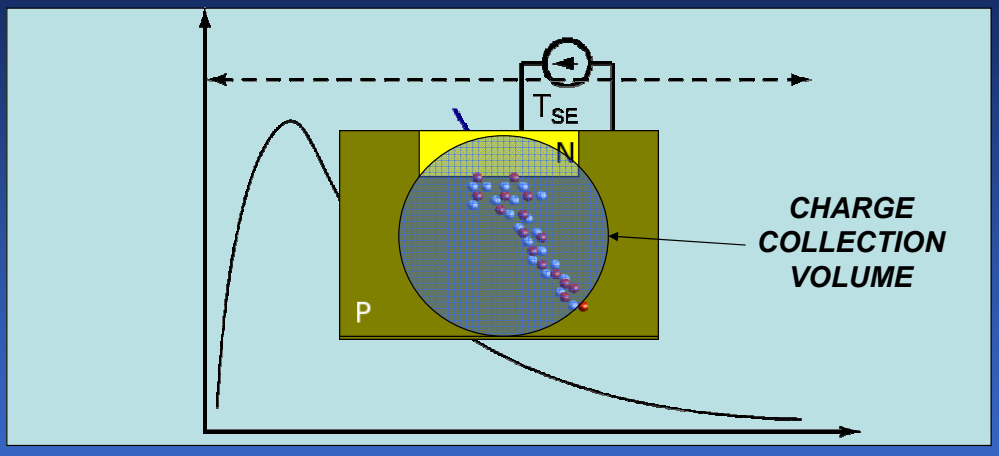
2. A Description of SEE's

What you always wanted to know about Single Event Effects (SEE's)

- *What are they?:*
  One of the result of the interaction between the radiation and the electronic devices
- *How do they act?:*
  Creating free charge in the silicon bulk that, in practical, behaves as a short-life but intense current pulse
- *Which are the ultimate consequences?*
  From simple bitflips or noise-like signals until the physical destruction of the device

2. A Description of SEE's

The Physical Mechanism



The incident particle generates a dense track of electron hole pairs and this ionization cause a transient current pulse if the strike occurs near a sensitive volume.

2. A Description of SEE's

The Classification of SEE's

SINGLE EVENT UPSET (SEU): CHANGE OF DATA OF MEMORY CELLS

MULTIPLE BIT UPSET (MBU): SEVERAL SIMULTANEOUS SEU'S

SINGLE EVENT TRANSIENT (SET): PEAKS IN COMBINATIONAL IC's

FUNCTIONAL INTERRUPTION (SEFI): PHENOMENA IN CRITICAL PARTS

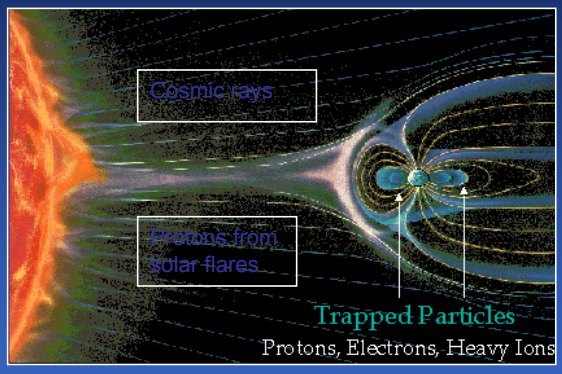
SINGLE EVENT LATCH-UP (SEL): PARASITIC THYRISTOR TRIGGER

AND OTHERS…

HARD ERRORS vs SOFT ERRORS

3. Sources of SEE's

Usually, SEE's have been associated with space missions because of the absence of the atmospheric shield…



Unfortunately, our quiet oasis seems to be vanishing since the enemy is knocking on the door…

- ***Alpha particle from vestigial U or Th traces***
- ***Atmospheric neutrons and other cosmic rays***

3. Sources of SEE's

Alpha Particles

– Sometimes, they appeared without a warning and, after some months and spending a lot of money, the source is detected*.

- In 1978, Intel had to stop a factory because water was extracted from a nearby river that, upstream, is too close to an old uranium mine.

** J. F. Ziegler and H. Puchner, "SER – History, Trends and Challenges. A guide for Designing with Memory ICs", Cypress Semiconductor, USA, 2004.*

3. Sources of SEE's

Alpha Particles

– Sometimes, they appeared without a warning and, after some months and spending a lot of money, the source is detected*.

- In 1986, IBM detected a high rate of useless devices and related it to the phosphoric acid, the bottles of which were cleaned with a ^{210}P deionizer gadget…hundreds of kms far.

** J. F. Ziegler and H. Puchner, "SER – History, Trends and Challenges. A guide for Designing with Memory ICs", Cypress Semiconductor, USA, 2004.*

3. Sources of SEE's

Alpha Particles

– Sometimes, they appeared without a warning and, after some months and spending a lot of money, the source is detected*.

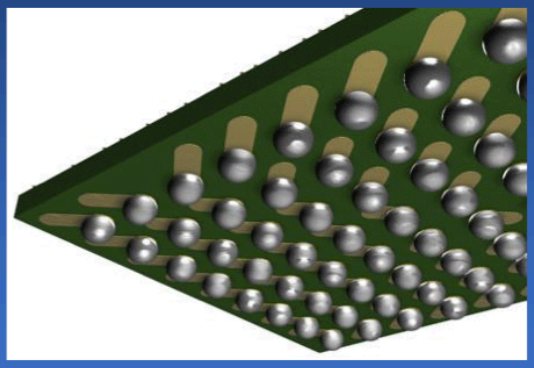
- In 1992, the problem came from the use of bat droppings living in cavern with traces of Th and U to obtain phosphorus.

** J. F. Ziegler and H. Puchner, "SER – History, Trends and Challenges. A guide for Designing with Memory ICs", Cypress Semiconductor, USA, 2004.*

3. Sources of SEE's

Alpha Particles

– But sometimes, we are a little naive...
  - Solder balls are usually made from Sn and Pb, *which come from minerals where there may be uranium and thorium traces.*



Nevertheless, the designer forgets this detail and places the solder balls too close to critical nodes!

3. Sources of SEE's

Cosmic Rays

Usually, they had been a headache for the designers of electronics boarded in space missions...

Here you are some of their practical jokes*...

- Cassini Mission (1997).- Some information was lost because of MBUs.
- Deep Space 1.- An SEU caused a solar panel to stop opening out.
- Mars Odyssey (2001).- Two weeks after the launch, alarms went off because some errors lately attributed to an SEU.
- GPS satellite network.- One of the satellites is out of work, probably because of a latch-up.

** B. E. Pritchard, IEEE NSREC 2002 Data Workshop Proceedings, pp. 7-17, 2002*

3. Sources of SEE's

Cosmic Rays at Ground Level

- The highest fluence is reached between 15-20 km of altitude.
- Less than 1% of this particle rain reaches the sea level.
- The composition has also changed…
  - Basically, neutrons and some pions

Usually, the neutron flux is referenced to that of New York City, its value been of (in appearance) only 15 n/cm^2/h

- This value depends on the altitude (approximately, x10 each 3 km until saturation at 15-20 km).
- And also on latitude, since the nearer the Poles, the higher rate.
- South America Anomaly (SAA), close to Argentina
- 1.5 m of concrete reduces the flux to a half.

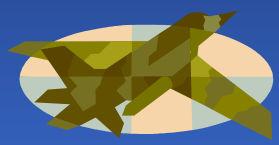
What a weak foe, really should be we afraid of?

3. Sources of SEE's

Cosmics Rays at Ground Level

Perhaps, we may believe that we are in a safe shelter but…

– 1992.- The PERFORM system, used by airplanes to manage the taking-off manoeuvre had to be suddenly replaced because of the SEUs in their SRAMs*.



– 1998.- A study reported that, every day, the 1 out of 10000 SRAMs attached to pacemakers underwent bitflips**.

This factor being 300 times higher if the patient had taken an transoceanic aircraft.

** J. Olsen, IEEE Trans. Nucl. Sci., 1993, 40, 74-77*

*** P. D. Bradley, IEEE Trans. Nucl. Sci., 45 (6), 2829-2940*

3. Sources of SEE's

Cosmic Rays at Ground Level

- The call of the Thousand (2000).- Sun Unix server systems crashed in dozens of places all over the USA because of SEU's happening in their cache memory, costing several millions of dollars*.
- 2005.- After 102 days, the ASC Q Cluster supercomputer showed 7170 errors in its 81-Gb cache memory, 243 of which led to a crash of the programs or the operating system**.

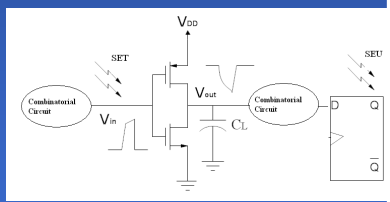
** FORBES, 2000*

*** K. W. Harris, IEEE Trans. Dev. Mat. Reliab., 2005, 5, 336-342*

4. Mitigation of SEE's

First of all, Where must we expect SEEs?

- All the combinational stages are supposed to be affected by SETs.
- Everything having SRAM cells is a candidate to show SEUs, MBU's:
  - SRAM's, Microprocessors, FPGAs, ASICs, etc.
- Other devices seem to be quite SEE-tolerant because of their way of building:
  - DRAMs, PSRAMs, NAND memories, etc.



Which are the strategies to mitigate SEE's?

1. *Technological*
2. *Design*
3. *Software and Hardware Redundancy*

Fault Models

- Low-level Fault Models:

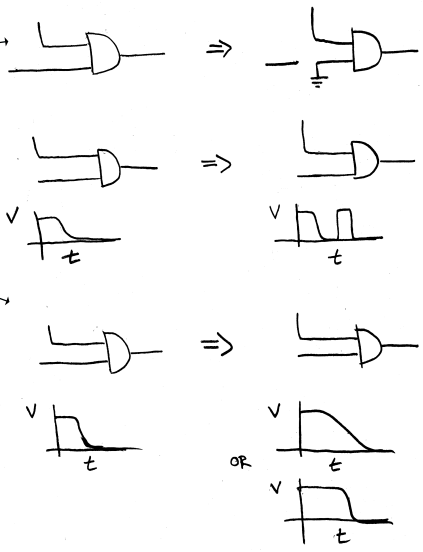
For logic circuit nodes

1. Permanent stuck at 0 or 1
2. Glitches
3. Slow transitions

For memory blocks

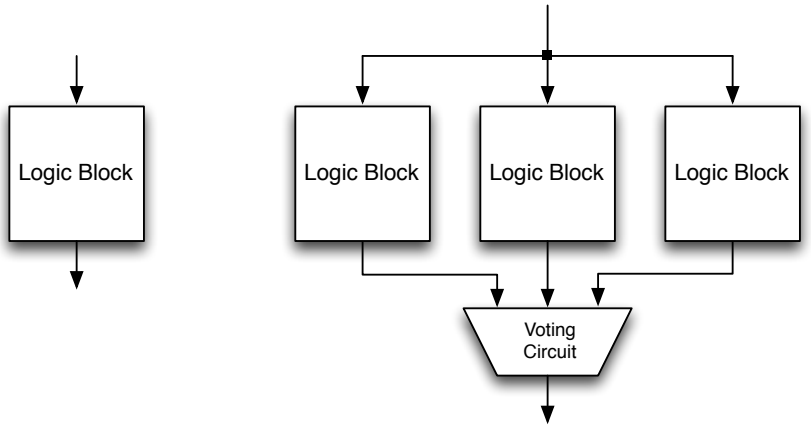
(and flip-flops, registers)

1. Permanent stuck at 0 or 1
2. Hold failure
3. Read upset
4. Slow read
5. Write failure



A Fault-Tolerant Design Methodology

Triple Modular Redundancy (TMR)



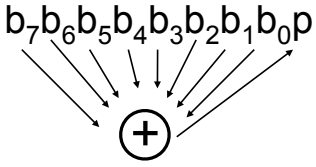
Error Correction Codes (ECC)

- Memory systems exhibit errors (accidentally flipped-bits)
  - Large concentration of sensitive nodes
  - "Soft" errors occur occasionally when cells are struck by alpha particles or other environmental upsets.
  - Less frequently, "hard" errors can occur when chips permanently fail.
- Where "perfect" memory is required
  - servers, spacecraft/military computers, …
- Memories are protected against failures with ECCs
- Extra bits are added to each data-word
  - extra bits are used to detect and/or correct faults in the memory system
  - in general, each possible data word value is mapped to a unique "code word". A fault changes a valid code word to an invalid one - which can be detected.

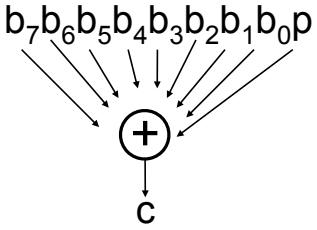
Simple Error Detection Coding

Parity Bit

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:



- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).



- A non-zero parity indicates an error occurred:
  - two errors (on different bits) is not detected (nor any even number of errors)
  - odd numbers of errors are detected.

Hamming Error Correcting Code

- Use more parity bits to pinpoint bit(s) in error, so they can be corrected.
- Example: Single error correction (SEC) on 4-bit data
  - use 3 parity bits, with 4-data bits results in 7-bit code word
  - 3 parity bits sufficient to identify any one of 7 code word bits
  - overlap the assignment of parity bits so that a single error in the 7-bit word can be corrected
- **Procedure**: group parity bits so they correspond to subsets of the 7 bits:
  - p_1 protects bits 1,3,5,7
  - p_2 protects bits 2,3,6,7
  - p_3 protects bits 4,5,6,7

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Note: number bits from left to right.

Bit position number

$001 = 1_{10}$
$011 = 3_{10}$
$101 = 5_{10}$
$111 = 7_{10}$ } p_1

$010 = 2_{10}$
$011 = 3_{10}$
$110 = 6_{10}$
$111 = 7_{10}$ } p_2

$100 = 4_{10}$
$101 = 5_{10}$
$110 = 6_{10}$
$111 = 7_{10}$ } p_3

Hamming Code Example

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

- Note: parity bits occupy power-of-two bit positions in code-word.
- On writing to memory:
  - parity bits are assigned to force even parity over their respective groups.
- On reading from memory:
  - check bits (c_3,c_2,c_1) are generated by finding the parity of the group and its parity bit. If an error occurred in a group, the corresponding check bit will be 1, if no error the check bit will be 0.
  - check bits (c_3,c_2,c_1) form the position of the bit in error.

- Example: $c = c_3c_2c_1 = 101$
  - error in 4,5,6, or 7 (by c_3=1)
  - error in 1,3,5, or 7 (by c_1=1)
  - no error in 2, 3, 6, or 7 (by c_2=0)
- Therefore error must be in bit 5.
- *Note the check bits point to 5*

- By our clever positioning and assignment of parity bits, the check bits always address the position of the error!

- c=000 indicates no error

Hamming Error Correcting Code

- Overhead involved in single error correction code:
  - let *p* be the total number of parity bits and *d* the number of data bits in a *p* + *d* bit word.
  - If p error correction bits are to point to the error bit (*p* + *d* cases) plus indicate that no error exists (1 case), we need:

    $2^p >= p + d + 1,$

    thus $p >= \log(p + d + 1)$

    for large *d*, *p* approaches log(*d)*

- Adding on extra parity bit covering the entire word can provide double error detection

  1 2 3 4 5 6 7 8

  p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4

- On reading the C bits are computed (as usual) plus the parity over the entire word, P:

  C=0 P=0, no error

  C!=0 P=1, correctable single error

  C!=0 P=0, a double error occurred

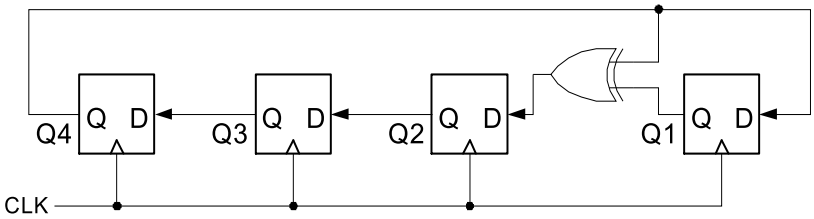
  C=0 P=1, an error occurred in p_4 bit

Typical modern codes in DRAM memory systems:
64-bit data blocks (8 bytes) with 72-bit code words (9 bytes), results in SEC, DED.

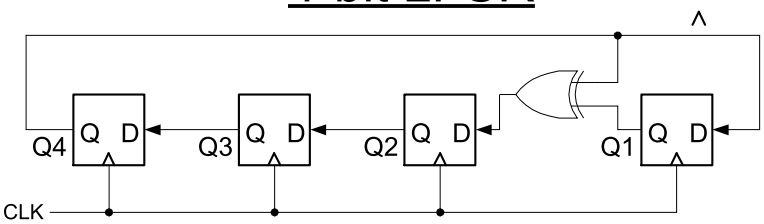
LFSRs

Linear Feedback Shift Registers (LFSRs)

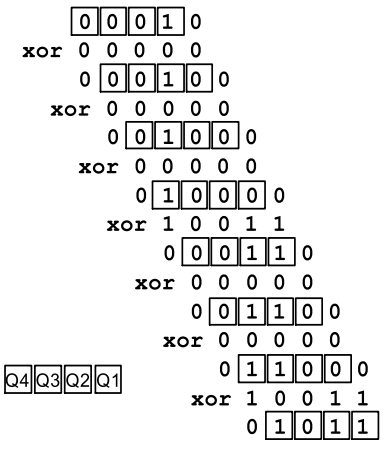
- These are n-bit counters exhibiting *pseudo-random* behavior.
- Built from simple shift-registers with a small number of xor gates.
- Used for:
  - random number generation
  - counters
  - error checking and correction
- Advantages:
  - very little hardware
  - high speed operation
- Example 4-bit LFSR:



4-bit LFSR



- Circuit counts through 2^4-1 different non-zero bit patterns.
- Leftmost bit decides whether the "10011" xor pattern is used to compute the next value or if the register just shifts left.
- Can build a similar circuit with any number of FFs, may need more xor gates.
- In general, with n flip-flops, 2^n-1 different non-zero bit patterns.
- (Intuitively, this is a counter that *wraps around* many times and in a strange way.)



0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

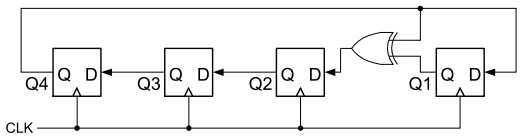
Primitive Polynomials

$x^2 + x + 1$
$x^3 + x + 1$
$x^4 + x + 1$
$x^5 + x^2 + 1$
$x^6 + x + 1$
$x^7 + x^3 + 1$
$x^8 + x^4 + x^3 + x^2 + 1$
$x^9 + x^4 + 1$
$x^{10} + x^3 + 1$
$x^{11} + x^2 + 1$
$x^{12} + x^6 + x^4 + x + 1$
$x^{13} + x^4 + x^3 + x + 1$
$x^{14} + x^{10} + x^6 + x + 1$
$x^{15} + x + 1$
$x^{16} + x^{12} + x^3 + x + 1$
$x^{17} + x^3 + 1$
$x^{18} + x^7 + 1$
$x^{19} + x^5 + x^2 + x + 1$
$x^{20} + x^3 + 1$
$x^{21} + x^2 + 1$
$x^{22} + x + 1$
$x^{23} + x^5 + 1$
$x^{24} + x^7 + x^2 + x + 1$
$x^{25} + x^3 + 1$
$x^{26} + x^6 + x^2 + x + 1$
$x^{27} + x^5 + x^2 + x + 1$
$x^{28} + x^3 + 1$
$x^{29} + x + 1$
$x^{30} + x^6 + x^4 + x + 1$
$x^{31} + x^3 + 1$
$x^{32} + x^7 + x^6 + x^2 + 1$

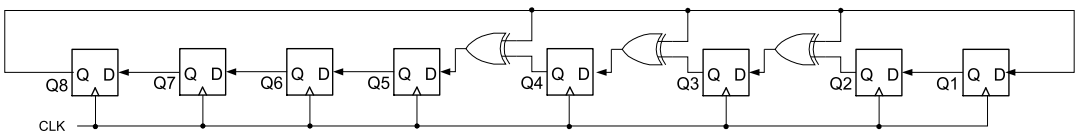
Galois Field		**Hardware**
Multiplication by x	⇔	shift left
Taking the result mod $p(x)$	⇔	XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, ..., 2^n-1$	⇔	Shifting and XOR-ing 2^n-1 times.

Building an LFSR from a Primitive Polynomial

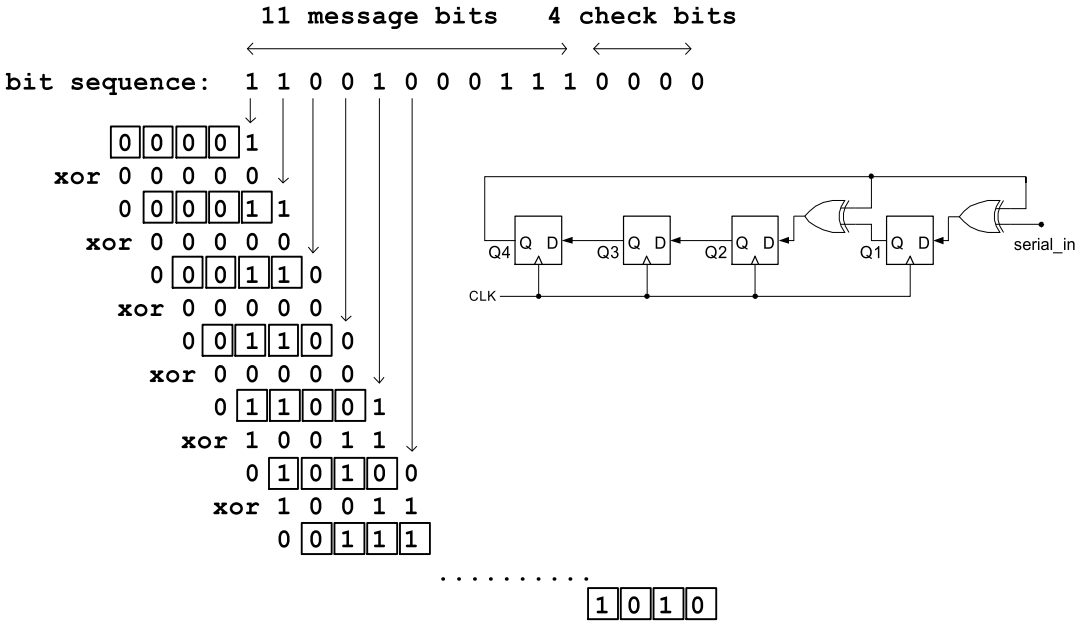
- For *k-bit* LFSR number the flip-flops with FF1 on the right.
- The feedback path comes from the Q output of the leftmost FF.
- Find the primitive polynomial of the form $x^k + ... + 1$.
- The $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of FF 1.
- Each term of the form x^n corresponds to connecting an xor between FF n and $n+1$.
- 4-bit example, uses $x^4 + x + 1$
  - x^4 ⇔ FF4's Q output
  - x ⇔ xor between FF1 and FF2
  - 1 ⇔ FF1's D input



- To build an 8-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect xors between FF2 and FF3, FF3 and FF4, and FF4 and FF5.



Error Correction with LFSRs



Error Correction with LFSRs

- XOR Q4 with incoming bit sequence. Now values of shift-register don't follow a fixed pattern. Dependent on input sequence.
- Look at the value of the register after 15 cycles: "1010"
- Note the length of the input sequence is 2^4-1 = 15 (same as the number of different nonzero patters for the original LFSR)
- Binary message occupies only 11 bits, the remaining 4 bits are "0000".
  - They would be replaced by the final result of our LFSR: "1010"
  - If we run the sequence back through the LFSR with the replaced bits, we would get "0000" for the final result.
  - 4 parity bits "neutralize" the sequence with respect to the LFSR.

    1 1 0 0 1 0 0 0 1 1 1 0 0 0 0 ⇒ 1 0 1 0

    1 1 0 0 1 0 0 0 1 1 1 1 0 1 0 ⇒ 0 0 0 0
- If parity bits not all zero, an error occurred in transmission.
- If number of parity bits = log total number of bits, then single bit errors can be corrected.
- Using more parity bits allows more errors to be detected.
- Ethernet uses 32 parity bits per frame (packet) with 16-bit LFSR.