KOMUNIKASI DATA

PENDETEKSI KESALAHAN
CRC
Pertemuan ke-10

Dosen
Lie Jasa

PENDETEKSI KESALAHAN

- **Tujuannya adalah mengetahui adanya kesalahan yang terjadi pada frame-frame yang ditransmisikan**

- **Pb : Probabilitas kesalahan bit tunggal, sering disebut dengan BER (*bit error rate*)**
- **P1 : Probabilitas dimana frame tiba tanpa kesalahan.**
- **P2 : Probabilitas dimana frame tiba dengan satu atau lebih kesalahan bit yang tak terdeteksi**
- **P3 : Probabilitas dimana frame tiba dengan satu atau lebih kesalahan bit yang terdeteksi namun tanpa kesalahan bit yang tak terdeteksi**

PENDETEKSI KESALAHAN

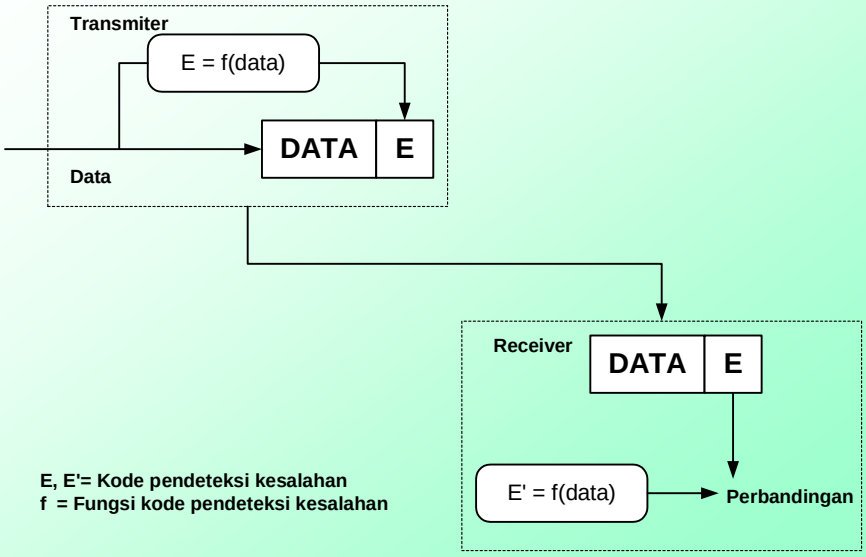
- **Probabilitas kesalahan yang terdeteksi (P3) menjadi NOL, Untuk menyatakan probabilitas yang tersisa, diasumsikan dimana bit-bit tersebut mengalami kesalahan (Pb), konstan dan bebas untuk masing-masing bit**

$$P1 = (1 – Pb)^F$$

$$P2 = (1 – P1)$$

- **F adalah jumla bit per frame**

PENDETEKSI KESALAHAN



E, E'= Kode pendeteksi kesalahan
f = Fungsi kode pendeteksi kesalahan

PENDETEKSI KESALAHAN

CEK PARITAS

Adalah mendeteksi kesalahan dengan melampirkan bit paritas ke ujung blok data. Ada paritas genap atau paritas ganjil. Biasanya paritas genap dipergunakan untuk transmisi Synchronous sedangkan paritas ganjil untuk transmisi Asynchronous.

CRC (*Cyclic Redudancy Check*)

Merupakan pendeteksi kesalahan yang paling umum dan dianggap paling handal.

PENDETEKSI KESALAHAN

Dalam CRC adanya blok bit K-bit atau Pesan, Transmiter mengirimkan suatu deretan (n-k) bit disebut sebagai sebagai *Frame Check Sequence (FCS).* Hal ini dapat dibagi dengan jelas oleh beberapa nomer yang sebelumnya sudah ditetapkan, kemudian pada receiver membagi frame yang datang dengan nomer tersebut. Bila tidak ada sisa diasumsikan tidak terjadi kesalahan

Menyajikan prosedurnya dalam 3 cara :

1). Modulo 2 Aritmatik, 2). Polynomial, 3). Digital Logic.

PENDETEKSI KESALAHAN

1). Modulo 2 Aritmatik

Menggunakan penambahan Biner tanpa pembawa, yang dikenal dengan operasi XOR (*Exclusive Or*)

Pengurangan biner tanpa pembawa yang dituliskan sebagai operasi XOR

```
  1111        1111          11001
+ 1010      - 0101        x    11
---------   ----------    -------------
  0101        1010          11001
                           110010   xor
                           ------------
                           101011
```

PENDETEKSI KESALAHAN

Sekarang menetapkan :

T = (k + n) bit frame untuk ditransmisikan, diman n < k

M = k bit pesan, k bit pertama dari T

F = n bit FCS, n bit terakhir dari T

P = pola n+1 bit, merupakan pembagi yang sudah ditetapkan sebelumnya.

Kita inginkan T/P tidak memiliki sisa, dimana

$$T = 2^n M + F$$

Dengan cara Mengalikan M dengan 2^n, sebenarnya kita telah memindahkannya ke kiri sebanyak n bit dan menambahkan hasilnya dengan NOL. Dengan menambahkan F menghasilkan deretan M dan F, yang merupakan T. Kita ingin T bisa dibagi oleh P

PENDETEKSI KESALAHAN

Anggap saja $2^n M$ dengan P

$$\frac{2^n M}{P} = Q + \frac{R}{P}$$

Menghasilkan hasil bagi dan sisa, karena pembaginya berupa modulo 2, sisanya selalu satu bit lebih rendah dari pembagi. Sisanya ini kita gunakan sebagai FCS.

$$T = 2^n M + R$$

Apabila R memenuhi syarat T/P tidak ada sisa sbb :

PENDETEKSI KESALAHAN

$$\frac{T}{P} = \frac{2^n M + R}{P}$$

Di substitusi dalam persamaan (1) kita dapatkan

$$\frac{T}{P} = [Q + R/P] + R/P$$

Bagaimanapun juga angka biner yang ditambahkan dengan modulo 2 akan menghasilkan NOL sehingga

$$\frac{T}{P} = Q + (R+ R)/P = Q$$

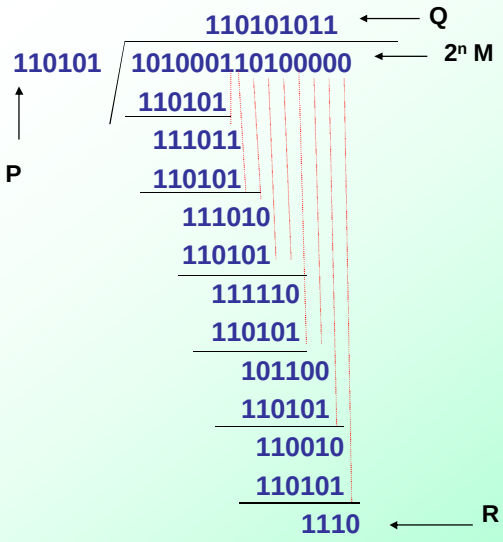
PENDETEKSI KESALAHAN

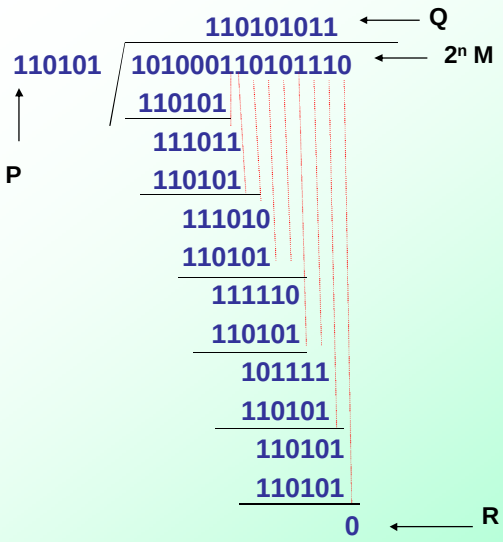
Contoh :

1) Pesan M = 1010001101 (10 bits)
Pola P = 110101 (6 bits)
FCS R = akan dihitung (5 bits)

2). Pesan dilakikan dengan n^5 maka didapatkan
101000110100000

3). Hasil point (2). Dibagi dengan P





PENDETEKSI KESALAHAN

Polynomial

Cara kedua mengamati proses CRC dengan menyatakan seluruh nilai sebagai Polynomial dalam suatu model Variabel X, dengan koefisien biner. Koefisien berhubungan dengan bit-bit dalam angka biner. Jadi untuk M = 110011, kita peroleh $M(x) = X^5 + X^4 + X + 1$, dan untuk P = 11001, kita peroleh $P(x) = X^4 + X^3 + 1$. Operasi aritmatik lagi berupa modulo 2.

Prose CRC digambarkan sbb.

$$\frac{X^n M(x)}{P(x)} = Q(x) + R(x)/P(x) \qquad T(x) = X^n M(x) + R(x)$$

PENDETEKSI KESALAHAN

CRC-12 = $X^{12} + X^{11} + X^3 + X^2 + X + 1$

CRC-16 = $x^{16} + x^{15} + x^2 + 1$

CRC-CCITT = $X^{16} + X^{12} + X^5 + 1$

CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

PENDETEKSI KESALAHAN

Digital Logic

Proses CRC ditunjukkan dan sekaligus diimplementasikan sebagai rangkaian pembagi yang terdiri dari gate Eksklusif OR dan register penggeser. Register penggeser adalah tempat penyimpan 1 bit, dengan satu input dan satu output. Semua perangkat register diletakkan secara simultas, sehingga menyebabkan pergeseran 1-bit disemua register.

Rangkaiannnya diimplementasikan sbb:

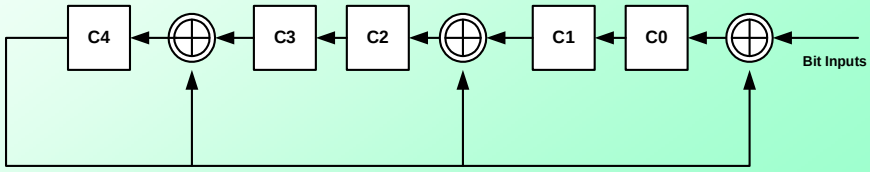
1. **Register memuat n-bit, setara dengan panjang FCS**
2. **Terdapat n atau lebih gate XOR**
3. **Ada atau tidaknya gate berkaitan dengan ada atau tidaknya term dalam pembagi pelinomial, P(X), tidak termasuk term X^n**

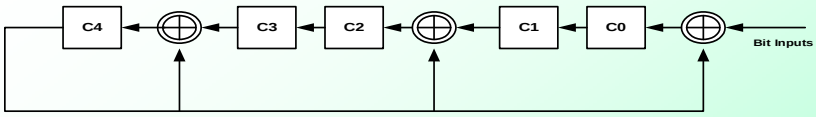
PENDETEKSI KESALAHAN

Arsitektur rangkaian dijelaskan dengan cara mengamati contoh gambar berikut sbb:

Pesan M = 1010001101 — **M(x) = X9 + X7 + X4 + X3+ X2 + 1**

Pembagi P = 110101 — **P(x) = X5 + X4 + X2 + 1**





	C4	C3	C2	C1	C0	C4 XOR C3	C4 XOR C1	C4XOR Input	Input	
Awal	0	0	0	0	0	0	0	1	1	
Tahap 1	0	0	0	0	1	0	0	0	0	Pesan Dikirim
Tahap 2	0	0	0	1	0	0	1	1	1	
Tahap 3	0	0	1	0	1	0	0	0	0	
Tahap 4	0	1	0	1	0	1	1	0	0	
Tahap 5	1	0	1	0	0	1	1	1	0	
Tahap 6	1	1	1	0	1	0	1	0	1	
Tahap 7	0	1	1	1	0	1	1	1	1	
Tahap 8	1	1	1	0	1	0	1	1	0	
Tahap 9	0	1	1	1	1	1	1	1	1	
Tahap 10	1	1	1	1	1	0	0	1	0	Lima Nol tambahan
Tahap 11	0	1	0	1	1	1	1	0	0	
Tahap 12	1	0	1	1	0	1	0	1	0	
Tahap 13	1	1	0	0	1	0	1	1	0	
Tahap 14	0	0	1	1	1	0	1	0	0	
Tahap 15	0	1	1	1	0	1	1	0	-	

TUGAS 10 MEI 2019

- Sebuah **CRC** disusun untuk membangkitkan FCS 4-Bit untuk sebuah pesan 11-Bit. Polinomial pembangkitnya adalah X4 + X3 + 1
  - Gambarkan rangkaian register-penggeser yang menampilkan fungsi ini.
  - Tuliskan deretan kode bit data 10110100110 (bit paling kiri paling tidak signifikan) menggunakan pembangkit polinomial dan berikan kata kodenya.
  - Asumsikan bahwa bit 7 (dihitung dari LSB) dalam kata kode mengalami kesalahan dan tunjukkan bahwa algoritma pendeteksian mendeteksi adanya kesalahan.

Sumber :

David Culler

Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://www-inst.eecs.berkeley.edu/~cs150

Review

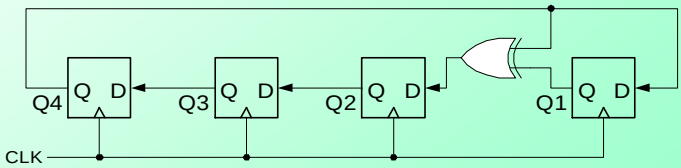
- **Concept of error coding**
  - Tambahkan beberapa bit tambahan (memperbesar ruang nilai-nilai) yang membawa informasi tentang semua bit
  - Detect : Fungsi sederhana untuk memeriksa seluruh data + cek diterima dengan benar
  - Correct : Algoritma untuk mencari simbol terdekat yang valid
- **Hamming codes**
  - Selektif menggunakan fungsi paritas
  - Jarak + # bit flips
  - Paritas : XOR dari bit => deteksi error tunggal
  - SECDED
    databits + p +1 <2p

Outline

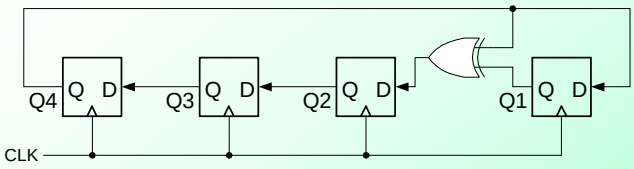
- **Memperkenalkan LFSR sebagai fancy counter**
- **Praktek Check Redudancy Cyclik**
  -Burst kesalahan dalam jaringan, disk, dll
- **Teori LFSRs**

Linear Feedback Shift Registers
(LFSRs)

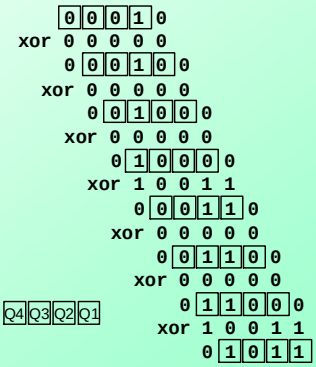
- Adalah sebuah n-bit counter menunjukkan perilaku *pseudo-random*.
- Dibangun dari shift-register sederhana dengan sejumlah gerbang xor.
- Digunakan untuk :
  - random number generation
  - counters
  - error checking and correction
- Keuntungan:
  - Hardware sangat sedikit
  - Operasi kecepatan tinggi
- Contoh 4-bit LFSR:



4-bit LFSR



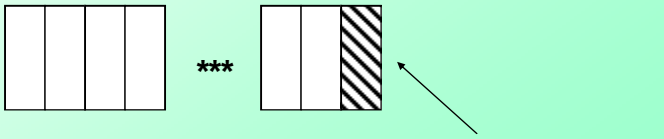
- Rangkaian Menghitung melalui 2^4-1 perbedaan non-zero bit patterns.
- Bit Waktu yang paling menentukan operasi pergeseran atau lebih kompleks
- Dapat dibangun dari similar circuit dengan beberapa FFs, atau gerbang XOR.
- Umumnya dengan n flip-flops, 2^n-1 perbedaan non-zero bit patterns.
- (intuitif, ini adalah counter yang mengerjakan secara berulang-ulang dan dengan cara yang aneh)



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

Concept : Redundant Check

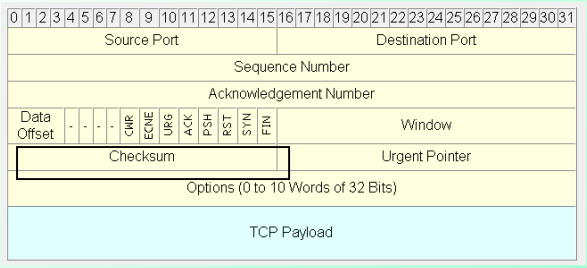
- Mengirim message M dan “mengecek” word C
- Secara Sederhana <M,C> dihitung jika keduanya diterima dengan benar (dengan probabilitas tinggi)
- Contoh: XOR semua bytes di M dan tambahkan dengan “checksum” byte, C, di akhir
  - Receiver XORs <M,C>
  - What should result be?
  - What errors are caught?



Contoh : TCP Checksum

7	Application (HTTP,FTP, DNS)
4	Transport (TCP, UDP)
3	Network (IP)
2	Data Link (Ethernet, 802.11b)
1	Physical

TCP Packet Format

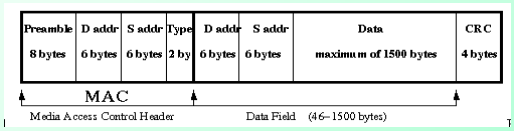


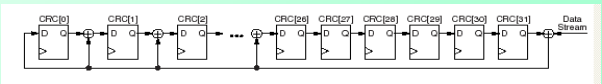
• TCP Checksum a 16-bit checksum, terdiri dari one's complement dari one's complement sum dari isi TCP segment header dan data, ini dihitung oleh sender, dan termasuk segment transmission. (note end-around carry)

• Jumlahkan semua words, termasuk checksum word, harus menghasilkan NOL

Example: Ethernet CRC-32

7	Application (HTTP,FTP, DNS)
4	Transport (TCP, UDP)
3	Network (IP)
2	Data Link (Ethernet, 802.11b)
1	Physical





CRC concept

- I have a msg polynomial M(x) of degree m
- We both have a generator poly G(x) of degree m
- Let r(x) = remainder of M(x) x^n / G(x)
  - M(x) x^n = G(x)p(x) + r(x)
  - r(x) is of degree n
- What is (M(x) x^n – r(x)) / G(x) ?

n bits of zero at the end

- So I send you M(x) x^n – r(x)

tack on n bits of remainder
Instead of the zeros

  - m+n degree polynomial
  - You divide by G(x) to check
  - M(x) is just the m most signficant coefficients, r(x) the lower m
- n-bit Message is viewed as coefficients of n-degree polynomial over binary numbers

Galois Fields - the theory behind LFSRs

- LFSR circuits performs multiplication on a *field*.
- A field is defined as a *set* with the following:
  - two operations defined on it:
    - "addition" and "multiplication"
  - closed under these operations
  - associative and distributive laws hold
  - additive and multiplicative identity elements
  - additive inverse for every element
  - multiplicative inverse for every non-zero element
- Example fields:
  - set of rational numbers
  - set of real numbers
  - set of integers is *not* a field (why?)
- Finite fields are called *Galois fields*.
- Example:
  - Binary numbers 0,1 with XOR as "addition" and AND as "multiplication".
  - Called GF(2).
  - 0+1 = 1
  - 1+1 = 0
  - 0-1 = ?
  - 1-1 = ?

Galois Fields - The theory behind LFSRs

- Consider *polynomials* whose coefficients come from GF(2).
- Each term of the form x^n is either present or absent.
- Examples: 0, 1, x, x^2, and $x^7 + x^6 + 1$

  $= 1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$
- With addition and multiplication these form a field:
- "Add": XOR each element individually with no carry:

$$\begin{array}{r} x^4 + x^3 + \quad + x + 1 \\ + \quad x^4 + \quad + x^2 + x \\ \hline x^3 + x^2 \quad + 1 \end{array}$$

- "Multiply": multiplying by x^n is like shifting to the left.

$$\begin{array}{r} x^2 + x + 1 \\ \times \quad x + 1 \\ \hline x^2 + x + 1 \\ x^3 + x^2 + x \\ \hline x^3 \quad + 1 \end{array}$$

So what about division (mod)

$\frac{x^4 + x^2}{x} = x^3 + x$ with remainder 0

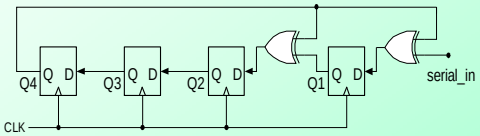
$\frac{x^4 + x^2 + 1}{X + 1} = x^3 + x^2$ with remainder 1

$$
\begin{array}{r|l}
 & x^3 + x^2 + 0x + 0 \\
\hline
X + 1 & x^4 + 0x^3 + x^2 + 0x + 1 \\
 & x^4 + x^3 \\
\hline
 & x^3 + x^2 \\
 & x^3 + x^2 \\
\hline
 & 0x^2 + 0x \\
 & 0x + 1
\end{array}
$$

Remainder 1

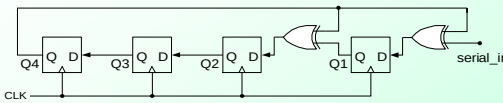
Polynomial division

```
                0 0 0 0 1 0 1
          ---------------------
10011   | 1 0 1 1 0 0 1 0 0 0 0
          1 0 0 1 1
          -----------
          0 0 1 0 1
            0 1 0 1 0
              1 0 1 0 1
              ---------
              1 0 0 1 1
              ---------
              0 0 1 0 0
```



- When MSB is zero, just shift left, bringing in next bit
- When MSB is 1, XOR with divisor and shiftl

CRC encoding



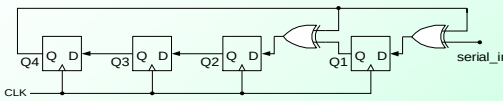
1 0 1 1 0 0 1 0 0 0 0

0	0	0	0	
0	0	0	1	0 1 1 0 0 1 0 0 0 0
0	0	1	0	1 1 0 0 1 0 0 0 0
0	1	0	1	1 0 0 1 0 0 0 0
1	0	1	1	0 0 1 0 0 0 0
0	1	0	1	0 1 0 0 0 0
1	0	1	0	1 0 0 0 0
0	1	1	0	0 0 0 0
1	1	0	0	0 0 0
1	0	1	1	0 0
0	1	0	1	0
1	0	1	0	

Message sent:

1 0 1 1 0 0 1 1 0 1 0

CRC decoding



1 0 1 1 0 0 1 1 0 1 0

0	0	0	0	
0	0	0	1	0 1 1 0 0 1 1 0 1 0
0	0	1	0	1 1 0 0 1 1 0 1 0
0	1	0	1	1 0 0 1 1 0 1 0
1	0	1	1	0 0 1 1 0 1 0
0	1	0	1	0 1 1 0 1 0
1	0	1	0	1 1 0 1 0
0	1	1	0	1 0 1 0
1	1	0	1	0 1 0
1	0	0	1	1 0
0	0	0	0	0
0	0	0	0	

Galois Fields - The theory behind LFSRs

- These polynomials form a *Galois (finite)* field if we take the results of this multiplication modulo a prime polynomial $p(x)$.
  - A prime polynomial is one that cannot be written as the product of two non-trivial polynomials $q(x)r(x)$
  - Perform modulo operation by subtracting a (polynomial) multiple of $p(x)$ from the result. If the multiple is 1, this corresponds to XOR-ing the result with $p(x)$.
- For any degree, there exists at least one prime polynomial.
- With it we can form $GF(2^n)$

- Additionally, …
- Every Galois field has a primitive element, α, such that all non-zero elements of the field can be expressed as a power of α. By raising α to powers (modulo $p(x)$), all non-zero field elements can be formed.
- Certain choices of $p(x)$ make the simple polynomial x the primitive element. These polynomials are called *primitive*, and one exists for every degree.
- For example, $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element and successive powers of α will generate all non-zero elements of GF(16). *Example on next slide.*

Galois Fields – Primitives

$\alpha^0 = 1$
$\alpha^1 = x$
$\alpha^2 = x^2$
$\alpha^3 = x^3$
$\alpha^4 = x + 1$
$\alpha^5 = x^2 + x$
$\alpha^6 = x^3 + x^2$
$\alpha^7 = x^3 + x + 1$
$\alpha^8 = x^2 + 1$
$\alpha^9 = x^3 + x$
$\alpha^{10} = x^2 + x + 1$
$\alpha^{11} = x^3 + x^2 + x$
$\alpha^{12} = x^3 + x^2 + x + 1$
$\alpha^{13} = x^3 + x^2 + 1$
$\alpha^{14} = x^3 + 1$
$\alpha^{15} = 1$

- Note this pattern of coefficients matches the bits from our 4-bit LFSR example.

$$\alpha^4 = x^4 \bmod x^4 + x + 1$$
$$= x^4 \text{ xor } x^4 + x + 1$$
$$= x + 1$$

- In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:

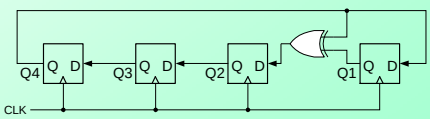
Primitive Polynomials

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

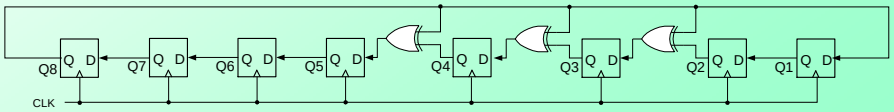
Galois Field		Hardware
Multiplication by x	$\Leftrightarrow$	shift left
Taking the result mod $p(x)$	$\Leftrightarrow$	XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n-1$	$\Leftrightarrow$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

Building an LFSR from a Primitive Poly

- 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 $x^k + \ldots + 1$.
- 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 $x^4 + x + 1$
  - $x^4 \Leftrightarrow$ FF4's Q output
  - $x \Leftrightarrow$ xor between FF1 and FF2
  - $1 \Leftrightarrow$ 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.



Generating Polynomials

- CRC-16: $G(x) = x^{16} + x^{15} + x^2 + 1$
  - detects single and double bit errors
  - All errors with an odd number of bits
  - Burst errors of length 16 or less
  - Most errors for longer bursts
- CRC-32: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  - Used in ethernet
  - Also 32 bits of 1 added on front of the message
    - Initialize the LFSR to all 1s