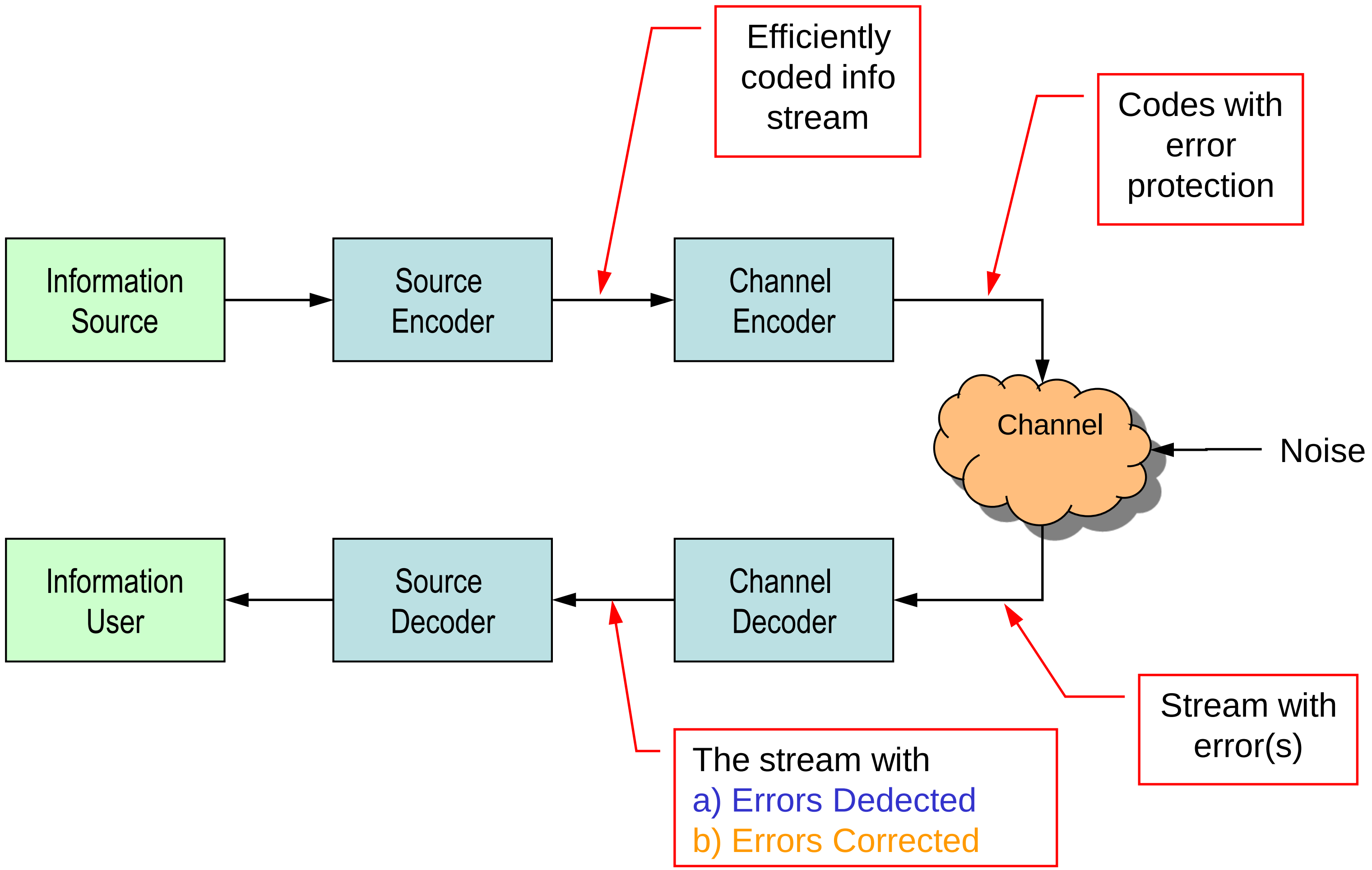
Channel Coding

by Erol Seke

For the course "Communications"

ESKİŞEHİR OSMANGAZİ UNIVERSITY

Information System



Error Detection

Parity Checking

One additional bit is added to each byte in the message

Longitudinal Redundancy Checking (LRC)

One additional character (block check character = BCC) to the end of each block of data

Polynomial Checking

A character or series of characters to the end of the message based on a mathematical algorithm.

Two most popular techniques are :

1. Checksum
2. Cyclic Redundancy Checking (CRC)

Error Correction

Reverse error correction

Stop and Wait ARQ

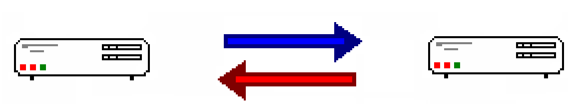
Continuous ARQ

Forward error correction

In forward error correction, there is no need to send retransmit requests to the sender. Because the errors are corrected at the receiver using the additional info transmitted (added redundancy)

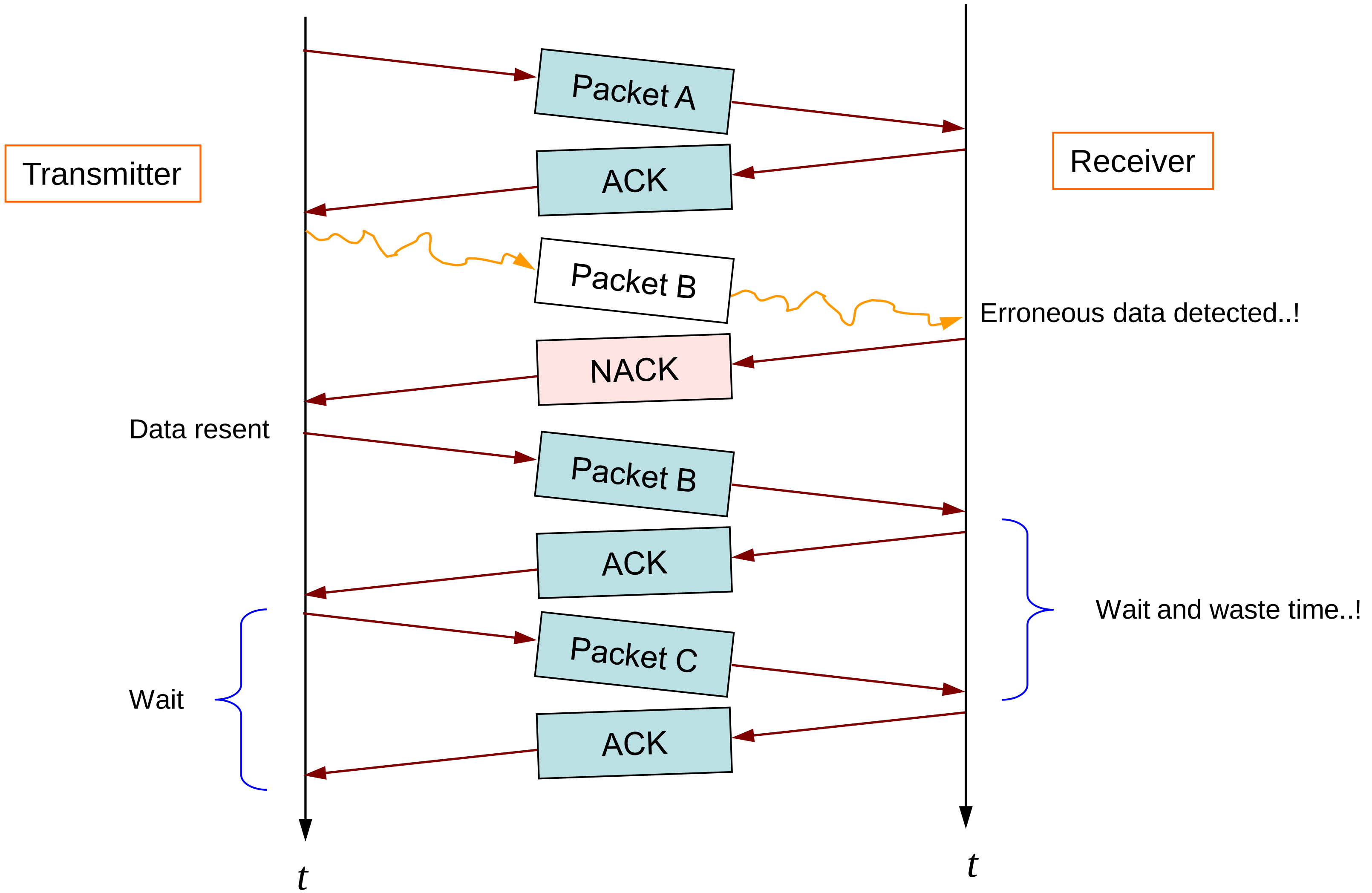
The simplest, most effective, least expensive, and most commonly used method is retransmission. Sometimes called Backward Error Correction

A receiver that detects an error simply asks the sender to ***retransmit*** the message until it is received without error.
This is often called **A**utomatic **R**epeat re**Q**uest (ARQ). But requires duplex channel.

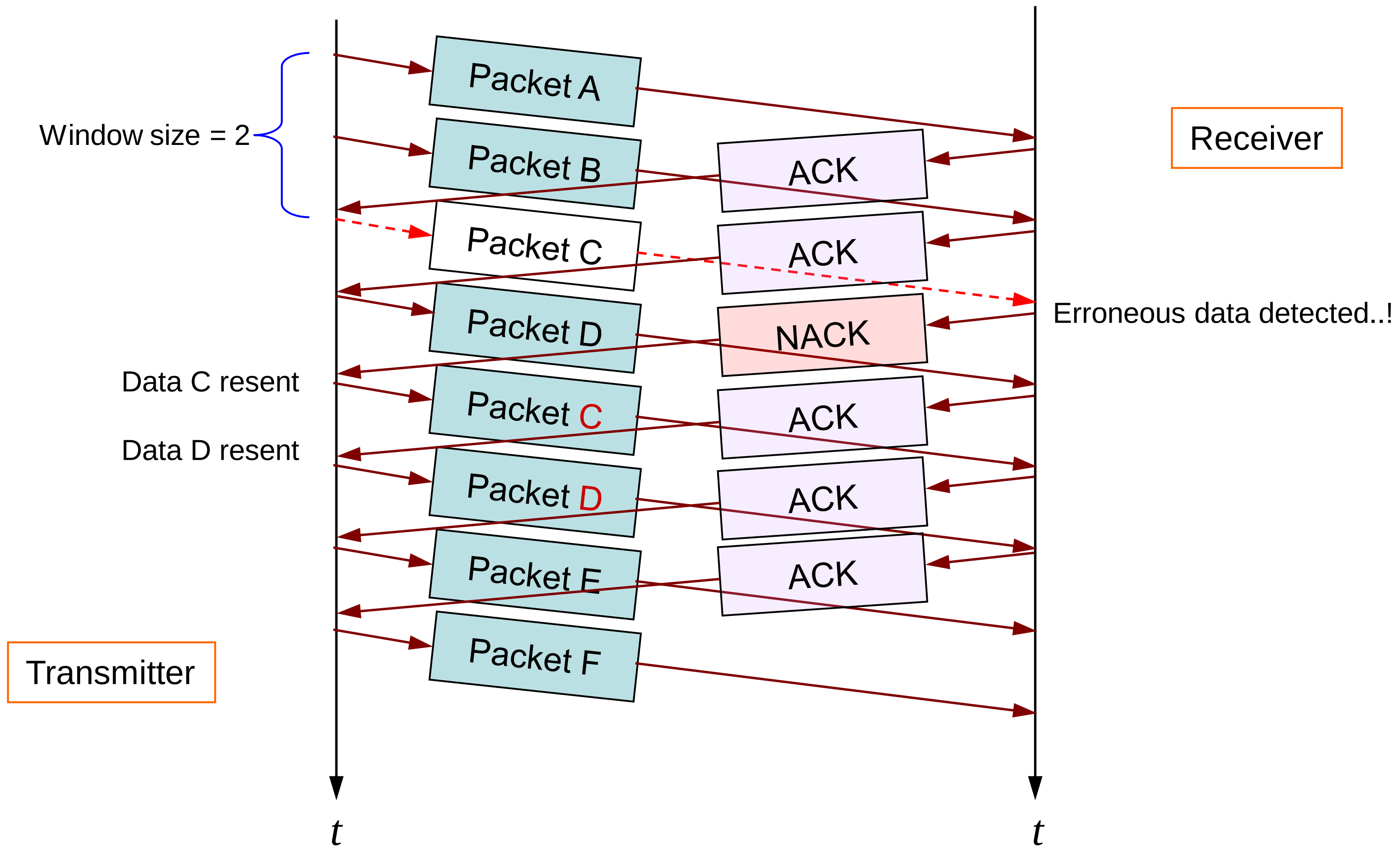


Full-Duplex : Both devices can transmit and receive simultaneously.

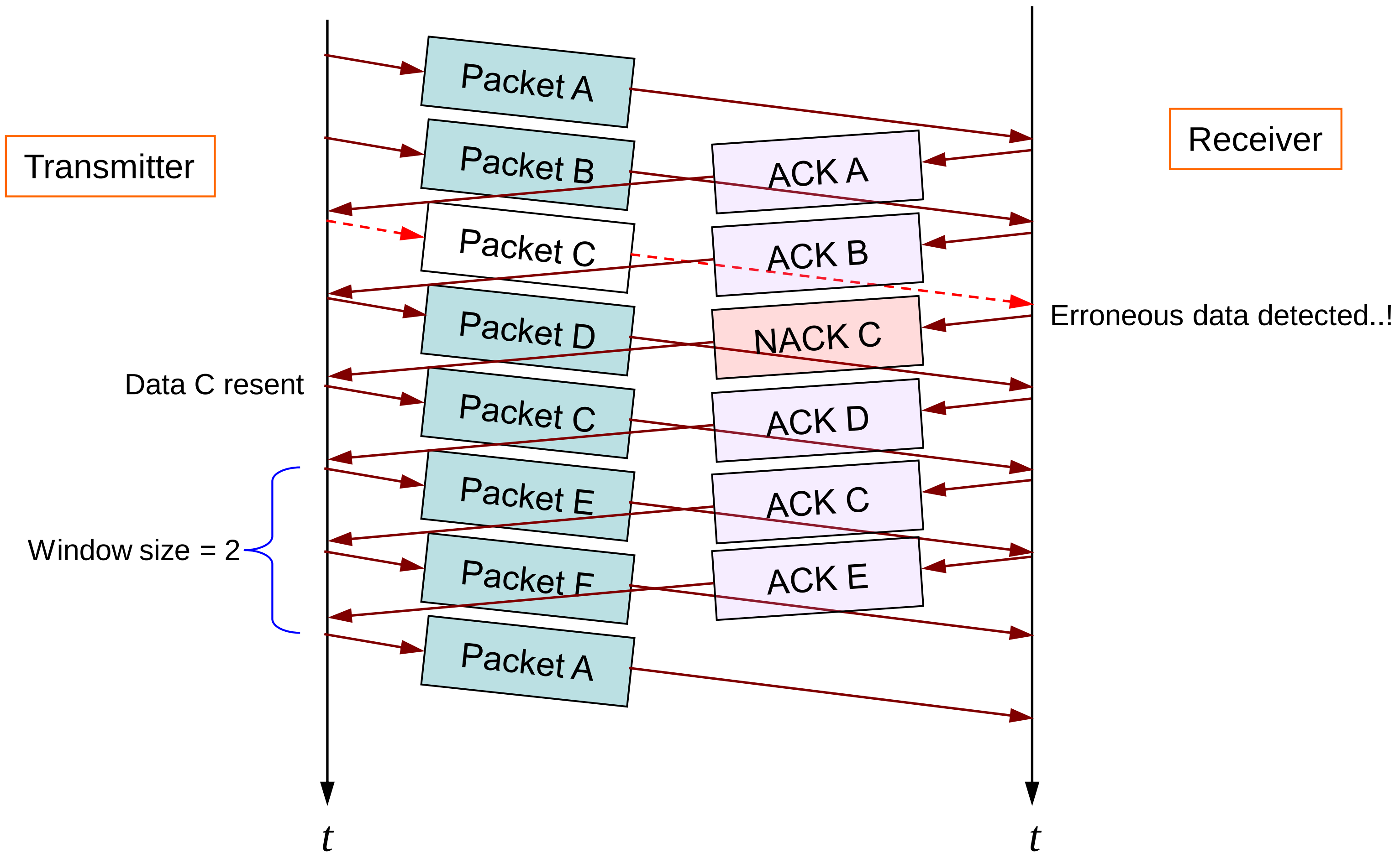
Stop and Wait ARQ



Continuous ARQ with Pullback – Sliding Window ARQ



Continuous ARQ with Selective Repeat – Sliding Window ARQ



Parity Checking

The additional parity bit is set to make the total number of ones in the byte (including the parity bit) either an even number or an odd number.

Seven data bits	Even Parity	Odd Parity
0101100	01011001	01011000

Additional bit (parity bit)

Same calculation is done at the receiver. If the same parity bit is found then it is assumed that received bits are OK

Example

1111011	11110110	11110111
	Two bit errors	Receiver think everything is OK
Receiver side	11101110	11010001

Longitudinal Redundancy Checking (LRC)

LRC adds one additional character, called the block check character (BCC) to the end of each block of data before the block is transmitted

	P	A	R	I	T	Y	BCC
BIT 1	1	1	1	1	1	1	**0**
BIT 2	0	0	0	0	0	0	**0**
BIT 3	1	0	1	0	1	1	**0**
BIT 4	0	0	0	1	0	1	**0**
BIT 5	0	0	0	0	1	0	**1**
BIT 6	0	0	1	0	0	0	**1**
BIT 7	0	1	0	1	0	1	**1**
PARITY	**0**	**0**	**1**	**1**	**1**	**0**	**1**

Parity of first bits

Parity of second bits

•
•
•

Parity of first byte

Parity of second byte

• • •

Even when used together, the parity and LRC will not catch all errors.

LRC will fail to detect errors that occur in an "even rectangular form", and other forms harder to describe as long as there are an even number of errors in each column and each row

Homework : Calculate the probability of detecting single bit and double bit errors in LRC

Polynomial Checking (1. Checksum)

The checksum is calculated by summing up the numerical value of each character, ignoring the carries if exist and using the remainder as the checksum that is transmitted to the other end of the communication circuit

Hexadecimal values of the character	checksum
12 40 05 80 FB 12 00 26 B4 BB 09 B4 12 28 74 11	BB
12 00 2E 22 12 00 26 75 00 00 FA 12 00 26 25 00	3A
F5 00 DA F7 12 00 26 B5 00 06 74 10 12 00 2E 22	F1
74 11 12 00 2E 22 74 13 12 00 2E 22	B4

Checksum is calculated in the same way in the receiver and compared with the received one.
If they are different then it is clear that the received data has error(s). It is obvious that the sum might come up the same even if the values are different. Therefore checksum detects only about 95% of the errors.

F5 00 DA F7 12 00 26 B5 00 06 74 12 10 00 2E 22	F1

Polynomial Checking (2. Cyclic Redundancy Checking)

A block of data is treated as one long binary polynomial P

The sender divides P by a fixed binary polynomial G, resulting in a whole polynomial Q, and a remainder, R/G.

$$\frac{P}{G} = Q + \frac{R}{G}$$

$x^5 + x^2 + 1$ (USB)

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

(IEEE 802.3)

The remainder R is appended to the message before transmission, as a check sequence k bits long (8, 16, 24, or 32 bits).

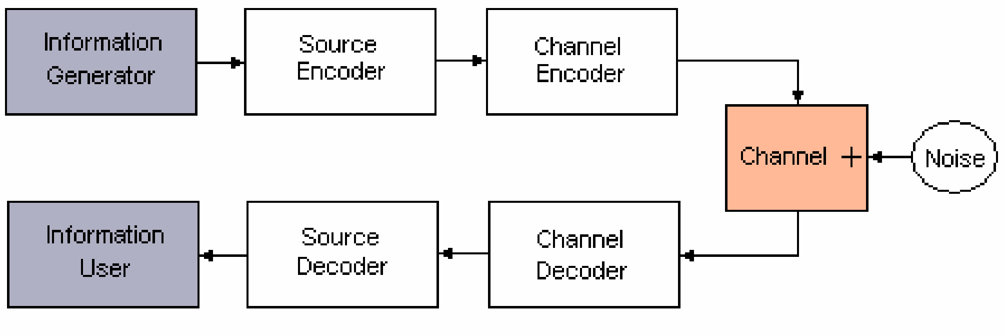
Same calculation is done at the receiver as the block is received. If the CRC numbers are the same, it is assumed that the data is error free.

The receiving hardware divides the received message by the same G, which generates an R.

The receiving hardware checks to ascertain whether the received R agrees with the locally generated R. If it does not, the message is assumed to be in error

CRC has become the standard method of error detection for block data transmission because of its high reliability in detecting transmission errors.
1. An 8 bit CRC detects 99,969 percent of the error.
2. CRC-16 (16 bits) detects at least 99.99 percent of them.
3. CRC-24 (24 bits) allows only three bits in 100 million to go undetected, and the error rate of 3 X 10-8.
4. Today 32 bit CRC codes are popular because they have an even higher error detection rate

Back to Information System



Source Encoder : For coding efficiency, removes coding redundancy, does compression

Channel Encoder : For reliability and robustness against channel noise and errors, adds coding redundancy

How do we add some redundancy to code so that we can recover from some errors?

Forward Error Correction (History)

1. Around 1947-1948, the subject of information theory was created by Claude Shannon.
2. During the same time period, Richard Hamming discovered and implemented a single-bit error correcting code.(Block coding or algebraic coding)
3. Soon after, Marcel Golay generalized Hamming's construction and constructed codes. He also constructed two very remarkable codes that correct multiple errors, and that now bear his name.
4. Another FEC technique, known as convolutional coding, was first introduced in 1955. Historically,the first type of convolutional decoding was sequentional decoding.
5. In 1960, researchers, including Irving Reed and Gustave Solomon, discovered how to construct error correcting codes that could correct for an arbitrary number of bits or an arbitrary number of "bytes". Even though the codes were discovered at this time, there still was no way known to decode the codes.
6. In 1967, Andrew Viterbi developed a decoding technique that has since become the standard for decoding convolutional codes.
7. In 1968, Elwyn Berlekamp and James Massey discovered algorithms needed to build decoders for multiple error correcting codes. They came to be known as the Berlekamp-Massey algorithm.
8. In 1974, Joseph Odenwalder combined these two coding techniques to form a concatenated code. In this arrangement, the encoder linked together an algebraic code followed by a convolutional code.
9. In 1993, Claude Berrou and his associates developed the turbo code, the most powerful forward error-correction code yet. Using the turbo code, communication systems can approach the theorethical limit of channel capacity, as characterized by the so-called Shannon Limit, which had been considered unreachable for more than four decades.

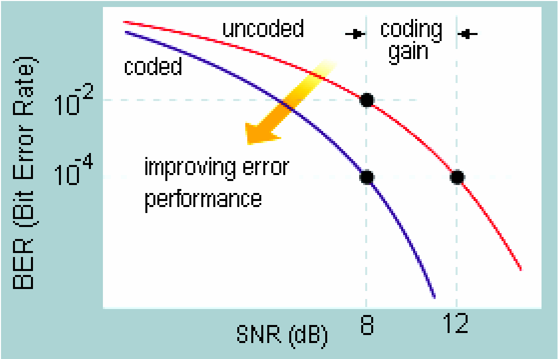
The codes are usually designated by (n,k) pairs, where n is the number of code bits (output) and k is the number of data bits (input)

Rate, $R = \frac{k}{n}$ is a measure of information (in bits) per output bit.

$$redundancy = \frac{n-k}{n}$$

is good for protection against channel errors but bad for channel utilization.

System performance improves (i.e., bit-error rate decreases) as SNR increases.



$$G(dB) = \left(\frac{E_b}{N_0}\right)_B (dB) - \left(\frac{E_b}{N_0}\right)_A (dB)$$

Homework : Read 6.3

Block Codes

C is a block code (n,k)

$$C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_i, \ldots, \mathbf{c}_M\} \qquad i = 1,2,\ldots,M\,, \quad M = 2^k$$

where $\mathbf{c}_i$ is a sequence of 0s and 1s of length n and is called a codeword.

if $\mathbf{c}_i \oplus \mathbf{c}_j$ is a codeword then C is called linear block code.

modulo 2 addition

Assumption

$\mathbf{x}_1 \oplus \mathbf{x}_2$ maps into $\mathbf{c}_1 \oplus \mathbf{c}_2$

if $\mathbf{x}_1$ and x_2 map into $\mathbf{c}_1$ and $\mathbf{c}_2$ respectively

Example

Given $C = \{00000, \quad 10100, \quad 01111, \quad 11011\}$, a (5,2) code

The code is linear since $\mathbf{c}_i \oplus \mathbf{c}_j$ is also a codeword

Given the mapping

$$\begin{aligned} 00 &\rightarrow 00000 \\ 01 &\rightarrow 01111 \\ 10 &\rightarrow 10100 \\ 11 &\rightarrow 11011 \end{aligned}$$

the assumption is correct

However with

$$\begin{aligned} 00 &\rightarrow 10100 \\ 01 &\rightarrow 01111 \\ 10 &\rightarrow 00000 \\ 11 &\rightarrow 11011 \end{aligned}$$

the assumption is incorrect

The Hamming Distance

The number of components that differ between $\mathbf{c}_i$ and $\mathbf{c}_j$ $\quad d(\mathbf{c}_i, \mathbf{c}_j)$

The Hamming Weight

The number of nonzero components of the codeword $\mathbf{c}_i$ $\quad w(\mathbf{c}_i)$

Minimum Hamming Distance

$$d_{\min} = \min_{\substack{\mathbf{c}_i, \mathbf{c}_j \\ i \neq j}} \{d(\mathbf{c}_i, \mathbf{c}_j)\}$$

Minimum Weight of the Code

$$w_{\min} = \min_{\mathbf{c}_i \neq 0} \{w(\mathbf{c}_i)\}$$

Theorem : $\quad d_{\min} = w_{\min}$ $\quad$ in any linear code

Let the information sequences be, in a (n,k) code

$$\mathbf{e}_1 = (1000...0)$$
$$\mathbf{e}_2 = (0100...0)$$
$$\mathbf{e}_3 = (0010...0)$$
$$\vdots$$
$$\mathbf{e}_k = (0000...1)$$

and their corresponding codewords be $\mathbf{g}_1, \mathbf{g}_2, \ldots \mathbf{g}_k$

Since any information sequence $\mathbf{x}$ can be written as $\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{e}_i$

the corresponding codeword can be written as $\mathbf{c} = \sum_{i=1}^{n} x_i \mathbf{g}_i$

Define

$$\mathbf{G} \overset{def}{=} \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}$$

generator matrix

so $\mathbf{c} = \mathbf{x}\mathbf{G}$

Any linear combination of the rows of the generator matrix is a codeword.

The generator matrix of a (n,k) code is a kxn matrix of rank k.

The generator matrix completely describes the code.

The generator matrix of the code $C = \{00000, \quad 10100, \quad 01111, \quad 11011\}$

is found by taking the codewords corresponding the information sequences (10) and (01)

$$\mathbf{G} = \begin{bmatrix} 10100 \\ 01111 \end{bmatrix}$$

The codeword for information sequence (x_1, x_2) is $(c_1, c_2, c_3, c_4, c_5) = (x_1, x_2)\mathbf{G}$

or

$$\begin{aligned} c_1 &= x_1 \\ c_2 &= x_2 \\ c_3 &= x_1 \oplus x_2 \\ c_4 &= x_2 \\ c_5 &= x_2 \end{aligned}$$

Such a code is called a *systematic code*

In such codes first k bits are just the copies of the information bits.

The generator matrix of systematic codes shall be in the form of $G = [I_k \mid P]$

where I_k is a k x k identity matrix (for first k codebits) and P is a k x (n-k) binary matrix called parity matrix. So,

$$c_i = \begin{cases} x_i, & 1 \le i \le k \\ \sum_{j=1}^{k} p_{ji} x_j, & k+1 \le i \le n \end{cases}$$

Hamming Codes (R.W.Hamming 1940)

Let the information bits be x_1, x_2, x_3 and x_4

And the code word bits be

$$c_1 = x_1$$

$$c_2 = x_2$$

$$c_3 = x_3$$

$$c_4 = x_4$$

$$\left.\begin{aligned} c_5 &= c_1 \oplus c_2 \oplus c_4 \\ c_6 &= c_1 \oplus c_3 \oplus c_4 \\ c_7 &= c_2 \oplus c_3 \oplus c_4 \end{aligned}\right\} \text{parity check bits}$$

where the summations are in modulo 2

In this code the receiver is able to correct single bit errors in a word

(7,4) Hamming code words

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

Let

$$\mathbf{r} = r_1 r_2 r_3 r_4 r_5 r_6 r_7$$

be the received word with a maximum of 1 bit in error although

$$\mathbf{c} = c_1 c_2 c_3 c_4 c_5 c_6 c_7$$

was sent.

We simply find the closest match (ML) from the code words table.

Example : The received word is 0 1 1 0 1 0 1

The closest word in the table (with one bit difference) is in the fifth row.

The information bits sent are the first 4 bits of this code word.

It is not efficient to search the code book for the closest code word. There are better algorithms.

Since $0 \oplus 0 = 1 \oplus 1 = 0$ İt is obvious that $c_i \oplus c_i = 0$

Let us apply this to parity check equations

$$0 = c_1 \oplus c_2 \oplus c_4 \oplus c_5$$
$$0 = c_1 \oplus c_3 \oplus c_4 \oplus c_6$$
$$0 = c_2 \oplus c_3 \oplus c_4 \oplus c_7$$

If $\mathbf{r}$ is received with a maximum of one bit in error,
then the results of above calculations become

$$s_1 = r_1 \oplus r_2 \oplus r_4 \oplus r_5$$
$$s_2 = r_1 \oplus r_3 \oplus r_4 \oplus r_6$$
$$s_3 = r_2 \oplus r_3 \oplus r_4 \oplus r_7$$

(s_1, s_2, s_3)

For no or single bit errors
in first four bits

$(0,0,0)$

$(1,1,0)$

$(1,0,1)$

$(0,1,1)$

$(1,1,1)$

$\mathbf{s} = (s_1, s_2, s_3)$ is called the *syndrome vector*

(s_1, s_2, s_3)

$(0,0,0) \longrightarrow$ no error

$(1,1,0) \longrightarrow$ r_1 is erronous (not same as s_1)

$(1,0,1) \longrightarrow$ r_2 is erronous (not same as s_2)

$(0,1,1) \longrightarrow$ r_3 is erronous (not same as s_3)

$(1,1,1) \longrightarrow$ r_4 is erronous (not same as s_4)

Just complement the erronous bit

$(0,0,1)$

$(0,1,0)$

$(1,0,0)$

Means that the error is in the parity bits, so no action necessary to find the correct information bits. Just take them.

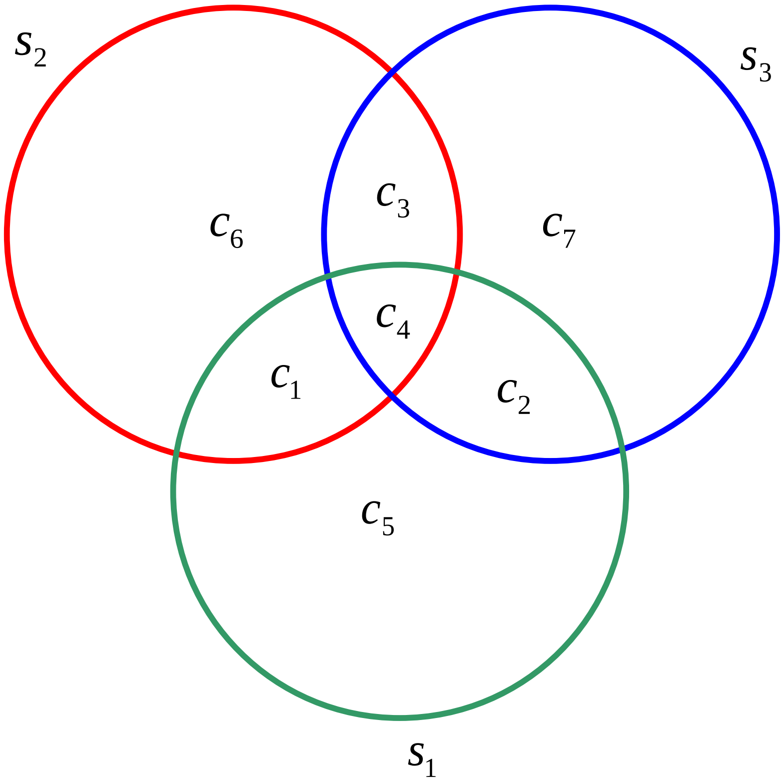
What happens if two bits were received in error?

If r_1 and r_2 are in error and a `0000000` was sent, then a `1100000` will be received.

$$s_1 = 1 \oplus 1 \oplus 0 \oplus 0$$
$$s_2 = 1 \oplus 0 \oplus 0 \oplus 0$$
$$s_3 = 1 \oplus 0 \oplus 0 \oplus 0$$

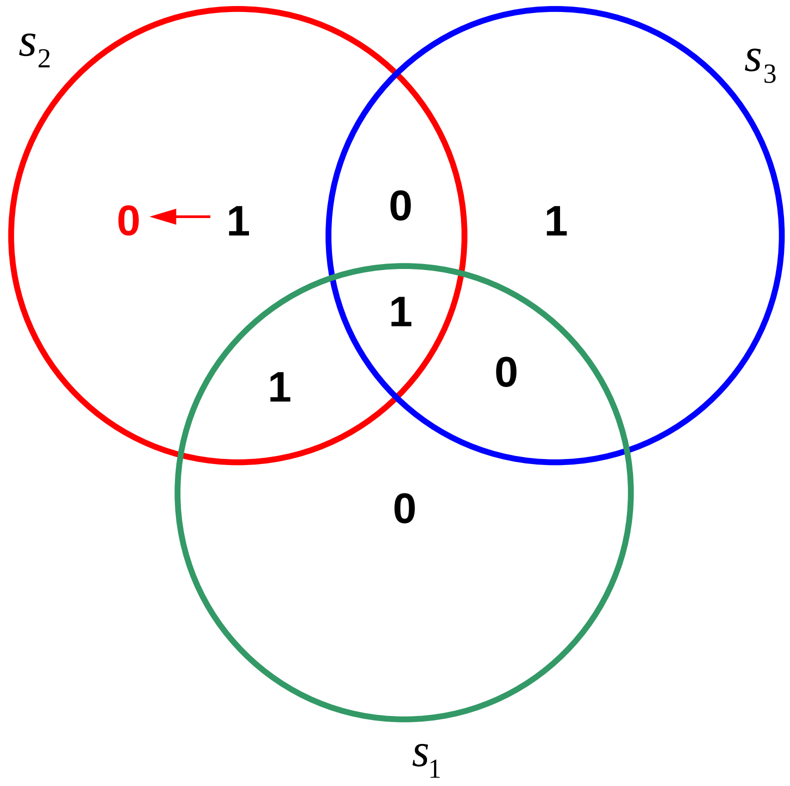
$\Rightarrow$ $(s_1, s_2, s_3) = (0,1,1)$ meaning that r_3 should be corrected !

McEliece's Diagrams



Since (s_1, s_2, s_3) must be all zero, we try to make the sums of the bits in each circle zero.

Example : The received word is $\mathbf{r} = 1001011$

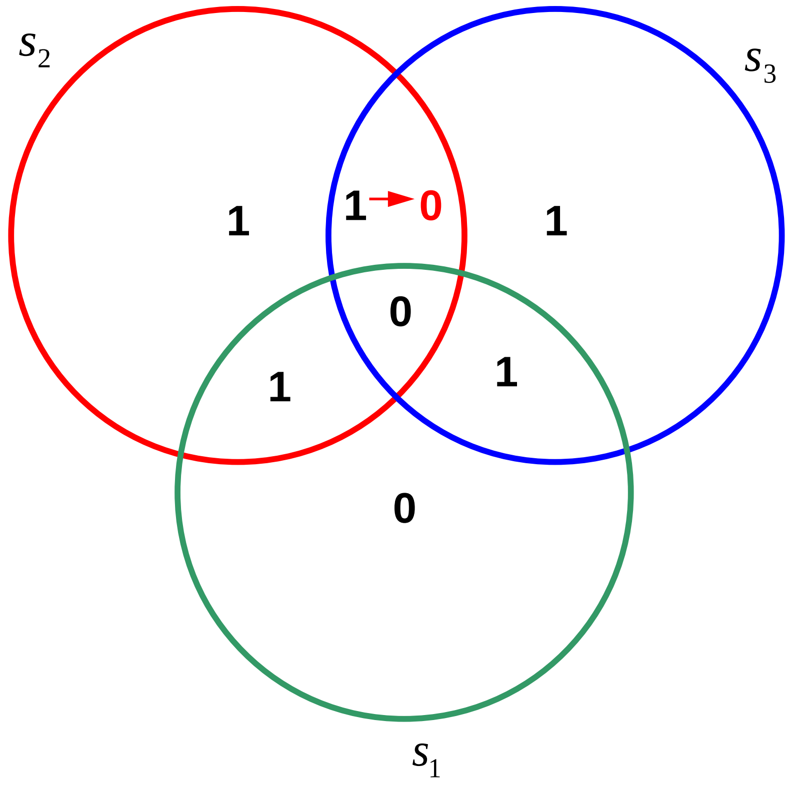


s_1 and s_3 are both zero (no problem there). But s_2 is 1.

In order to correct both s_2 and not change s_1 and s_3 we must make r_6=0.

The information bits are not affected from this : `1001`

Example : The received word is $\mathbf{r} = 1110011$



No problem with S_1, but S_2 and S_3 are 1. In order to correct both we must change r_3 to 0

The examples we have seen were using (7,4) Hamming code

(4 bit information and 7 bit code)

The next longer Hamming codes are (15,11), (31,26) and (63,57)

For each integer $m \geq 3$ There is an (n,k) Hamming code with

$n = 2^m - 1$ code bits of which

$k = 2^m - 1 - m$ are information bits and the remaining m are parity bits

$m = 4 \longrightarrow$ (15,11)

$m = 5 \longrightarrow$ (31,26)

$m = 6 \longrightarrow$ (63,57)

Note on error-correcting codes

Hamming codes are unable to correct multiple bit errors in a code word. For that we would need more complex codes like Reed-Solomon[hmw] codes.

Cyclic Codes

Cyclic codes are a subset of linear block codes

A cyclic code is a linear block code with the extra condition;

Cyclic shift of a codeword must also be a codeword

Example :

$$\{000, 110, 101, 011\}$$

Cyclic shifted versions

$$\{000, 101, 011, 110\}$$

are also codewords, so it is cyclic code

Cyclic codewords are thought of polynomials, called codeword polynomials

$$c(p) = \sum_{i=1}^{n} c_i p^{n-i} = c_1 p^{n-1} + c_2 p^{n-2} + \cdots c_{n-1} p + c_n \qquad c = (c_1, c_2, \cdots, c_{n-1}, c_n)$$

The polynomial

$$c^{(1)}(p) = c_2 p^{n-1} + c_3 p^{n-2} + \cdots c_{n-1} p^2 + c_n p + c_1 \quad \text{representing} \quad c^{(1)} = (c_2, c_3, \cdots, c_n, c_1)$$

is the cyclic shift of c and also a codeword in the code.

The mathematics are done in modulo arithmetic.

```
0+0 = 1+1 = 0-0 = 1-1 = 0
1+0 = 0+1 = 0-1 = 1-0 = 1
0x0 = 0x1 = 1x0 = 0
1x1 = 1
```

The interesting thing about these polynomials with modulo arithmetic is when $p^i c(p)$ is divided by $p^n + 1$ the remainder is $c^i(p)$

Let us show this when $i=1$

$$pc(p) = p(c_1 p^{n-1} + c_2 p^{n-2} + \cdots c_{n-1} p + c_n) = c_1 p^n + c_2 p^{n-1} + \cdots c_{n-1} p^2 + c_n p$$

and

$$\begin{array}{r|l} c_1 p^n + c_2 p^{n-1} + \cdots c_{n-1} p^2 + c_n p & p^n + 1 \\ \cline{2-2} - \quad c_1 p^n + c_1 & c_1 \\ \hline c_2 p^{n-1} + c_3 p^{n-2} + \cdots c_n p + c_1 & \end{array} \longrightarrow c^{(1)}(p)$$

(using modulo arithmetic properties)

similarly $$\frac{p^i c(p)}{p^n + 1} = \frac{c^{(i)}(p)}{p^n + 1} + c_i$$ or $$c^{(i)}(p) = p^i c(p) + c_i (p^n + 1)$$

and finally $$c^{(n)}(p) = p^n c(p) + c_n (p^n + 1) = c(p)$$

In a (n,k) cyclic code all codeword polynomials are multiples of a polynomial

Generator polynomial

$$g(p) = p^{n-k} + g_2 p^{n-k-1} + g_3 p^{n-k-2} + \cdots + g_{n-k} p + 1$$

which divides $p^n + 1$

If $X(p) = x_1 p^{k-1} + x_2 p^{k-2} + \cdots + x_{k-1} p + 1$

represents the information sequence $x = (x_1, x_2, \cdots, x_{k-1}, x_k)$

then the codeword polynomial is $c(p) = X(p) g(p)$

Example : $x = (1010)$ and $g = (1101)$ Find codeword

$$c(p) = (p^3 + p)(p^3 + p^2 + 1)$$
$$= p^6 + p^5 + p^3 + p^4 + p^3 + p$$
$$= p^6 + p^5 + p^4 + p$$

$c = (1110010)$ Since k = 4 and n-k = 3 this a codeword of (7,4)

Q : is it a systematic code?

Example : Generate a (7,4) cyclic code

$n-k=3$ ⟹ We need a 3rd degree generator polynomial and it has to divide p^7+1

$$p^7+1=(p+1)(p^3+p^2+1)(p^3+p+1)$$ choose this $$g(p)=p^3+p^2+1$$

and multiply by $X(p)=x_1p^3+x_2p^2+x_3p+x_4$ where (x_1,x_2,x_3,x_4) are info sequences.

information word	$X(p)$	$c(p)$	codeword
0000	0	0	0000000
0001	1	p^3+p^2+1	0001101
0010	p	p^4+p^3+p	0011010
0011	$p+1$	p^4+p^2+p+1	0010111
0100	p^2	.	.
0101	p^2+1	.	.
0110	p^2+p	.	.
0111	p^2+p+1		
1000	p^3		
1001	p^3+1		
1010	p^3+p		
1011	p^3+p+1		
1100	p^3+p^2		
1101	p^3+p^2+1		
1110	p^3+p^2+p		
1111	p^3+p^2+p+1		

Homework : Fill the rest of the table

For a systematic code $c(p) = p^{n-k} X(p) + \rho(p)$

where $\rho(p)$ is the remainder of the division $\dfrac{p^{n-k} X(p)}{g(p)}$

Example : given $g(p) = p^3 + p^2 + 1$ and $x = (1010)$ ((n,k) = (7,4))

$$p^{n-k} X(p) = p^{7-4}(p^3 + p) = p^6 + p^4$$

$$\begin{array}{r|l}
p^6 + p^4 & p^3 + p^2 + 1 \\ \cline{2-2}
-\ p^6 + p^5 + p^3 & p^3 + p^2 + 1 \\ \hline
p^5 + p^4 + p^3 & \\
-\ p^5 + p^4 + p^2 & \\ \hline
p^3 + p^2 & \\
-\ p^3 + p^2 + 1 & \\ \hline
1 & \text{(remainder)}
\end{array}$$

$$c(p) = p^6 + p^4 + 1$$

$$c = (1010001)$$

If the received codeword r has at most 1 bit error in (7,4) code then it is correctable.

For a code to be single-error-correcting, all single error patterns must be addressable by the syndrome vector. That is, the condition

$$2^{n-k} \geq n+1$$ is to be satisfied

Info bits	Code word
0000	0000000
0001	0001101
0010	0010111
0011	0011010
0100	0100011
0101	0101110
0110	0110100
0111	0111001
1000	1000110
1001	1001011
1010	1010001
1011	1011100
1100	1100101
1101	1101000
1110	1110010
1111	1111111

Note that;

1. First 4 bits of the codeword is the same with the info bits. (systematic)
2. The red-bits are parity which has the same size as the syndrome vector. 3 bits can address 7 different error positions and 1 no error condition.
3. Cyclic shifts of codewords are in the table too.
4. Code is linear.

$$G = \left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

Homework : complete the (7,4) systematic cyclic code table and *generator matrix* for $g(p) = p^3 + p + 1$

Syndrome vector is calculated by dividing the received word by the generator polynomial and taking the remainder.

$$s(p) = \mathrm{Rem}\frac{r(p)}{g(p)}$$

$$\frac{r(p)}{g(p)} = c(p) + \frac{s(p)}{g(p)}$$ remainder of long division

Example : correct the error in the received word 0101000, if there is any. Use the systematic code generated earlier.

$$r(p) = p^5 + p^3 \qquad g(p) = p^3 + p^2 + 1$$

$$\begin{array}{r|l} p^5 + p^3 & p^3 + p^2 + 1 \\ \cline{2-2} -\; p^5 + p^4 + p^2 & p^2 + p \\ \hline p^4 + p^3 + p^2 & \\ -\; p^4 + p^3 + p & \\ \hline p^2 + p & \end{array} \longleftarrow s(p)$$

We obtained the syndrome vector as $s = (110)$

But we do not know which bit it points to. Let us find syndrome vector for each possible error position.

$$\mathrm{Rem}\frac{p^6}{p^3 + p^2 + 1} = p^2 + p \longrightarrow 110$$

$$\mathrm{Rem}\frac{p^5}{p^3 + p^2 + 1} = p + 1 \longrightarrow 011$$

⋮ ⋮

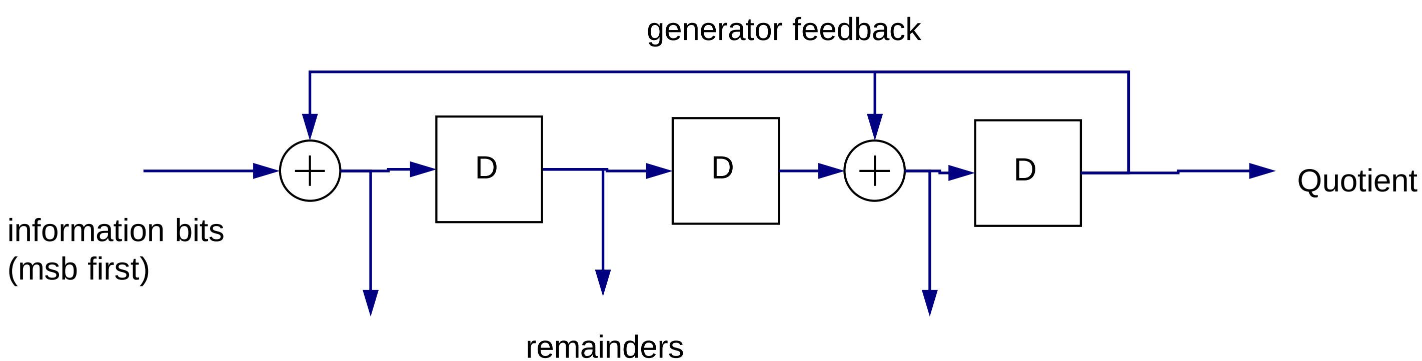
Syndrome Table

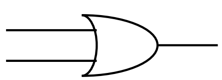
error position	syndrome
1000000	110
0100000	011
0010000	111
0001000	101
0000100	100
0000010	010
0000001	001

It is decoders job to calculate the syndrome vector and invert the bit which it points using the syndrome table. If there is no error, then the syndrome vector should be 000. The big assumption is that we have a single bit error.

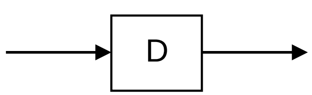
In the example (110) indicates that the first bit should be inverted. The corrected information word is (1101) instead of (0101)

Division Circuit for $g(p) = p^3 + p^2 + 1$



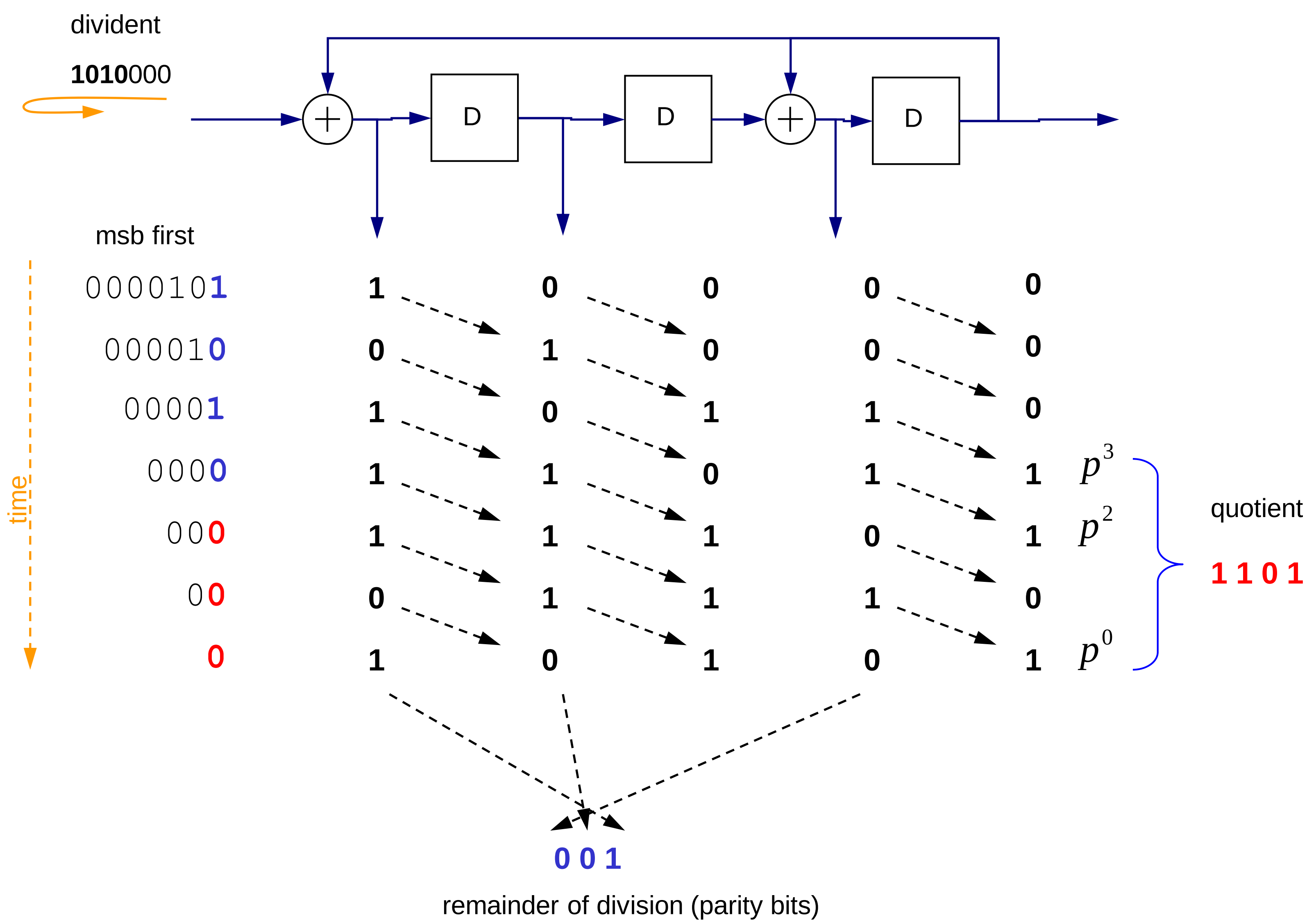


Modulo summations can easily be done by X-OR gates

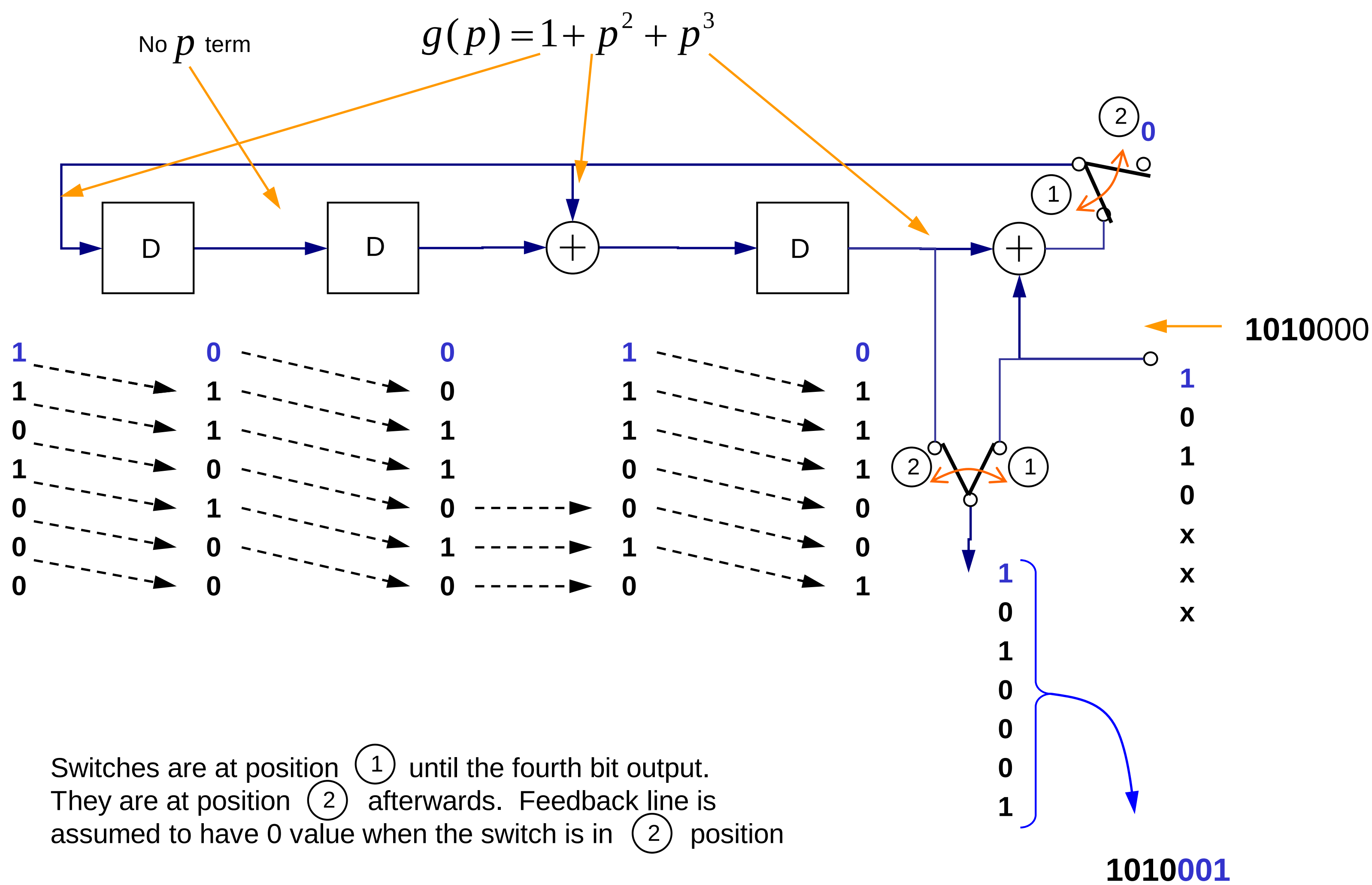


D-type flip-flop (1 bit register) represents a delay Z^{-1}

Binary Division (modulo) Circuitry (animated)



Systematic Codeword Generator Circuitry (animated)



Switches are at position 1 until the fourth bit output.
They are at position 2 afterwards. Feedback line is assumed to have 0 value when the switch is in 2 position

The generator $g(p) = p^{16} + p^{12} + p^{5} + 1$ standardized as V.41 by ITU-T is used in Wide-Area-Networks

The generator

$$g(p) = p^{32} + p^{26} + p^{23} + p^{22} + p^{16} + p^{12} + p^{11} + p^{10} + p^{8} + p^{7} + p^{5} + p^{4} + p^{2} + p + 1$$

is standardized by IEEE and is used in Local-Area-Networks and FDDIs.

Homework : Design a syndrome vector detection circuitry for the code previously analyzed.

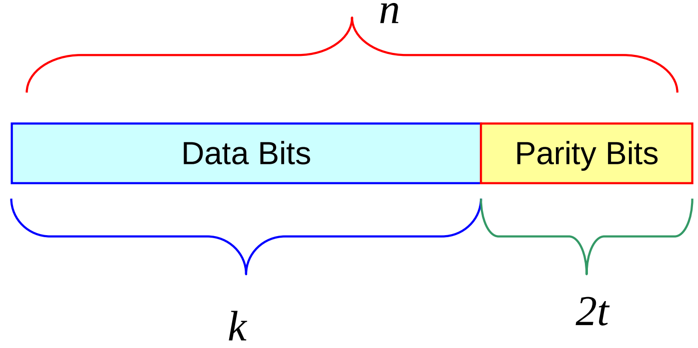
1. Divide received vector by the generator.
2. The remainder is the syndrome vector.
3. Use the syndrome table to determine the incorrect bit position.
4. Correct the errorenous bit if there is any (if the syndrome is nonzero).

Reed-Solomon

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. Reed-Solomon codes are used to correct errors in many systems including:

1. Storage devices (including tape, Compact Disk, DVD, barcodes, etc)
2. Wireless or mobile communications (including cellular telephones, microwave links, etc)
3. Satellite communications
4. Digital television
5. High-speed modems such as ADSL, xDSL, etc.

A Reed-Solomon code is specified as $\mathrm{RS}(n,k)$ with s-bit symbols. This means that the encoder takes k data symbols of s bits each and adds parity symbols to make an n symbol codeword. There are n-k parity symbols of s bits each.



A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, where $2t = n-k$.

Example : RS(255,223) with 8 bit symbols (s=8)

Bytes per codeword - Data bytes per codeword = 32 parity bytes

223 bytes (symbols)	32 bytes

$$\left.\begin{array}{l} n = 255 \\ k = 223 \\ s = 8 \end{array}\right\} \quad 2t = 32 \qquad t = 16$$

(the number of correctable symbols in a 255 symbol block)

The maximum codeword size is $n = 2^s - 1$

A codeword is, as usual, generated using a special polynomial called generator polynomial. All valid codewords are exactly divisible by it. The general form is:

$$g(x) = (x - a^{i})(x - a^{i+1}) \cdots (x - a^{i+2t})$$

a is the primitive element of the Galois field

A codeword is constructed using $$c(x) = g(x) \cdot i(x)$$

valid codeword

generator polynomial

information block

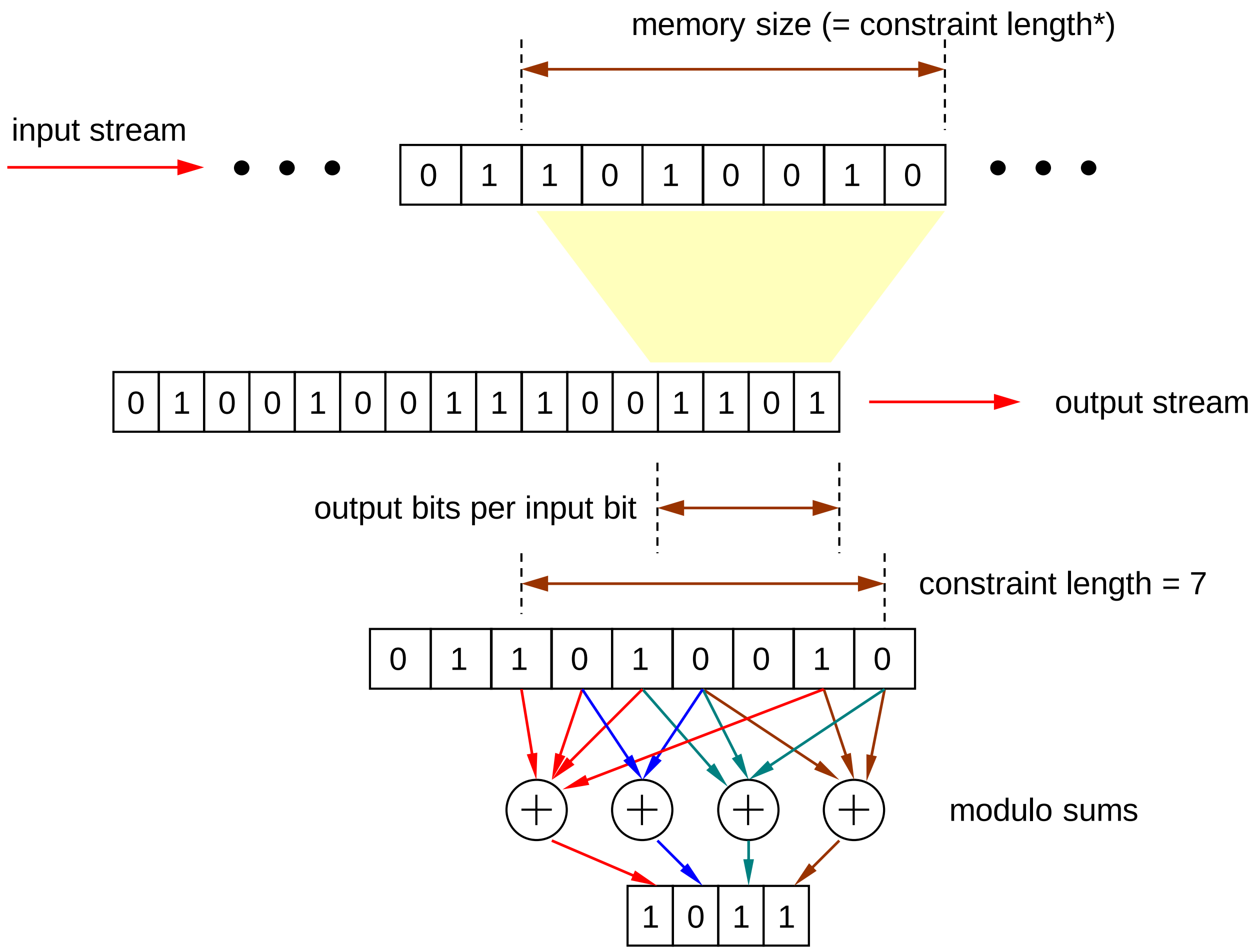
Example: Generator for RS(255,249) is

$$g(x) = (x - a^{0})(x - a^{1})(x - a^{2})(x - a^{3})(x - a^{4})(x - a^{5})$$

$$g(x) = x^{6} + g_{5}x^{5} + g_{4}x^{4} + g_{3}x^{3} + g_{2}x^{2} + g_{1}x + g_{0}$$

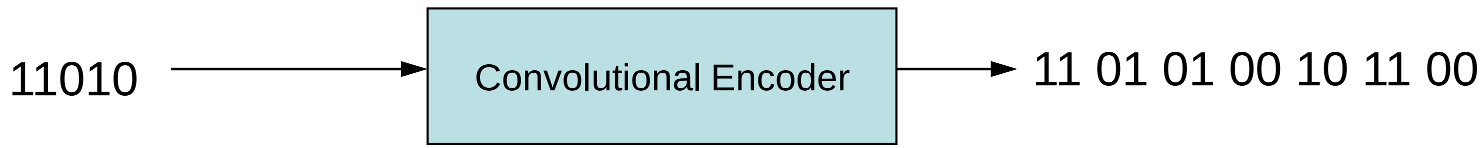
Convolutional Codes

Convolutional codes use not only the current symbol digits but also the previous N digits of the previous symbols. It does operate on streams not blocks.

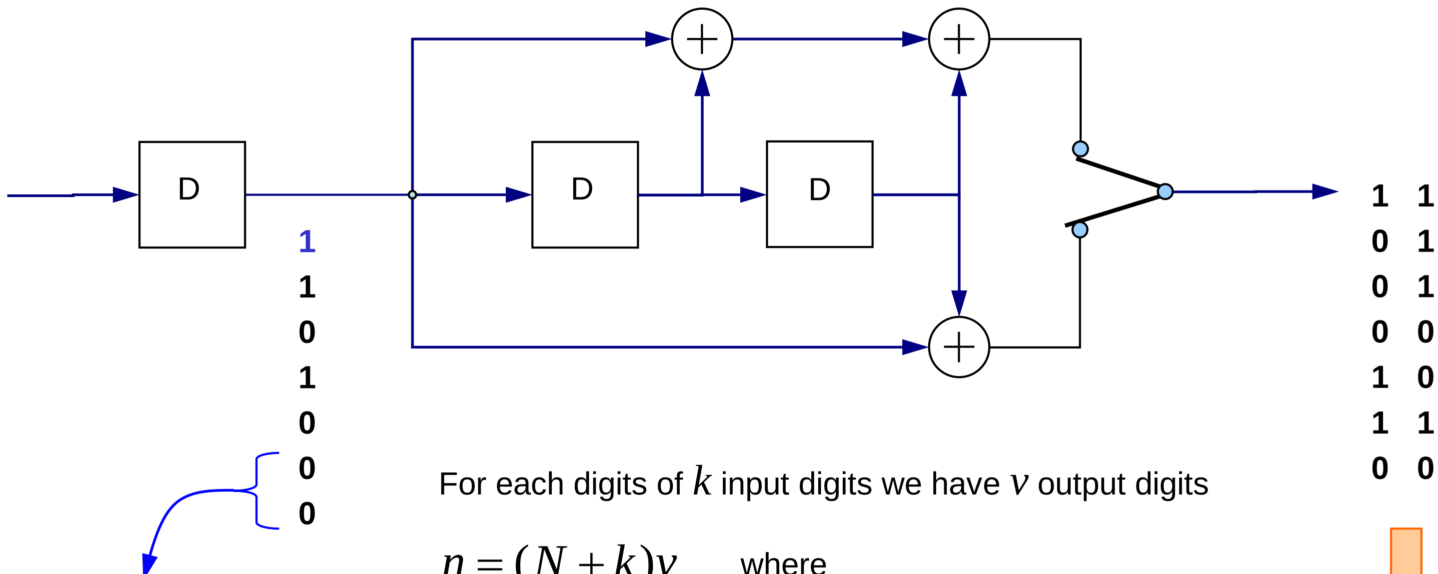


* : it is assumed that the bits are shifted-right one bit at a time in which case N=constraint length

(animated demo of a convolutional encoder)



$$o_{2i} = x_i \oplus x_{i-1} \oplus x_{i-2}$$
$$o_{2i+1} = x_i \oplus x_{i-2}$$



These are added to end the stream and make the system ready to accept another stream of bits. The entrance of the first bit of the next block shifts out the last remaining bit.

For each digits of k input digits we have v output digits

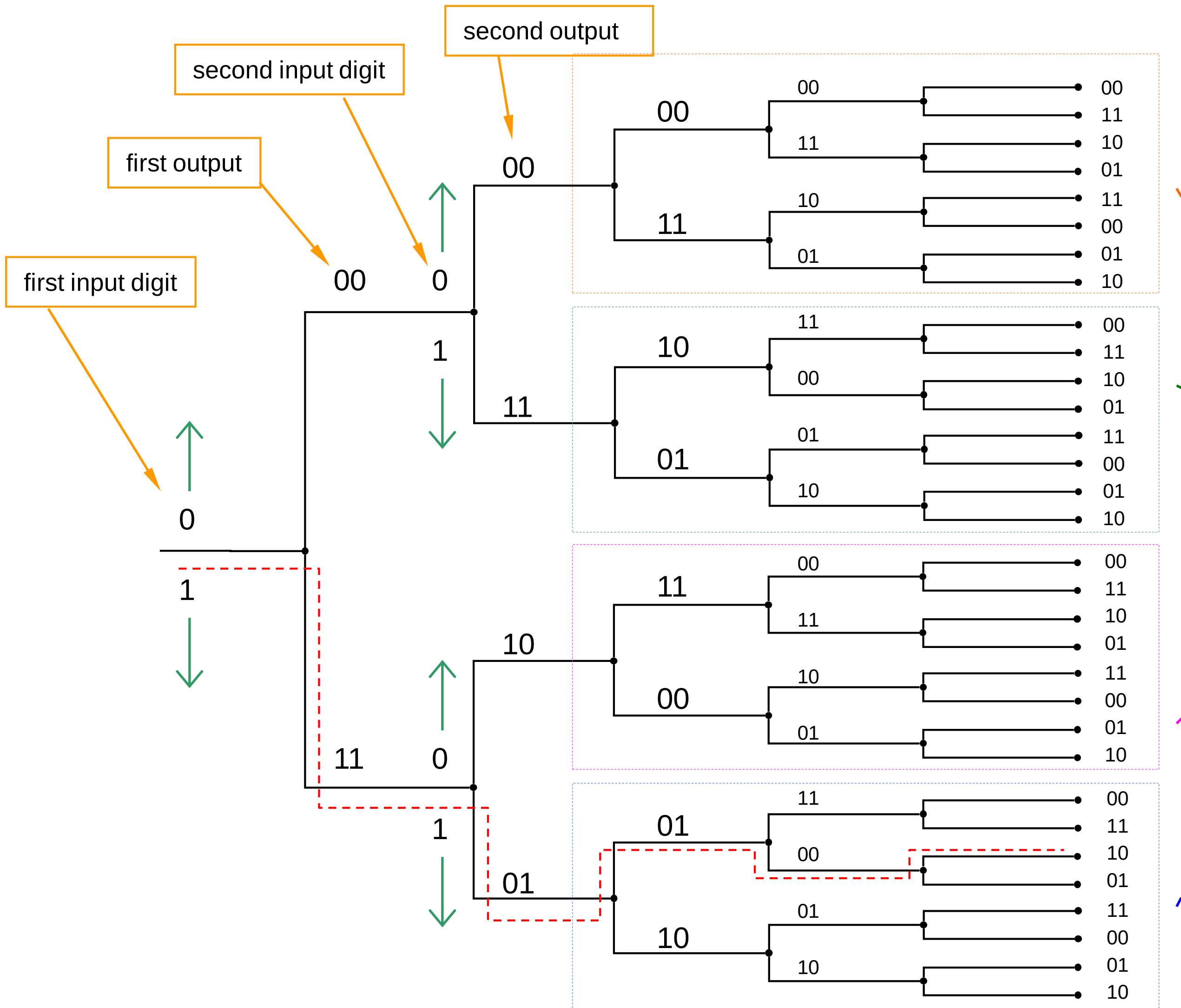
$$n = (N + k)v$$ where

n = number of output digits
N = the number of registers (memory)
k = number of input digits
v = number of sums (switch positions) at the output

(N becomes unimportant when k gets large)

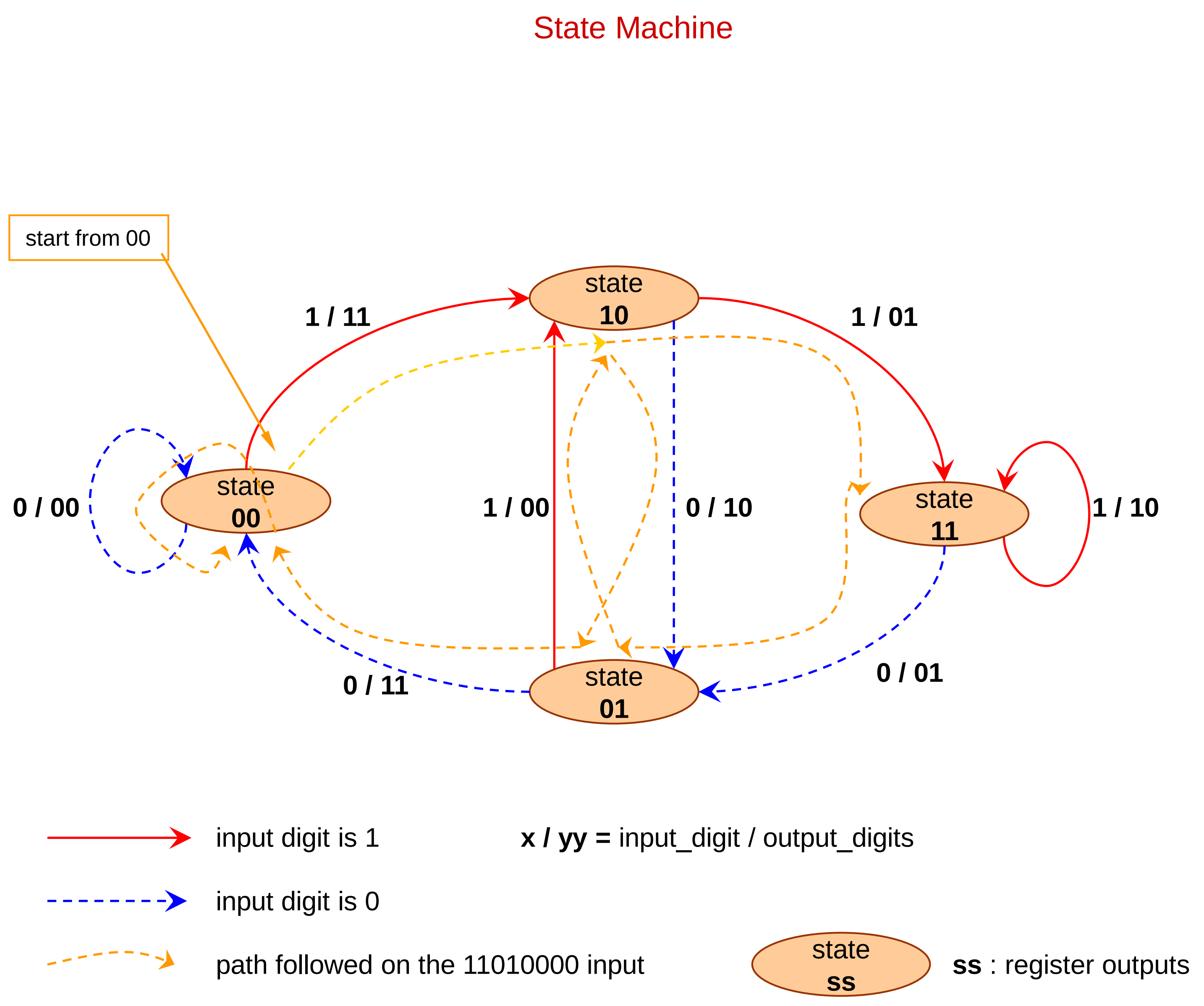
The rate is ½
(two output bits for each input bit)

The output sequence for any possible input can be shown on a code tree



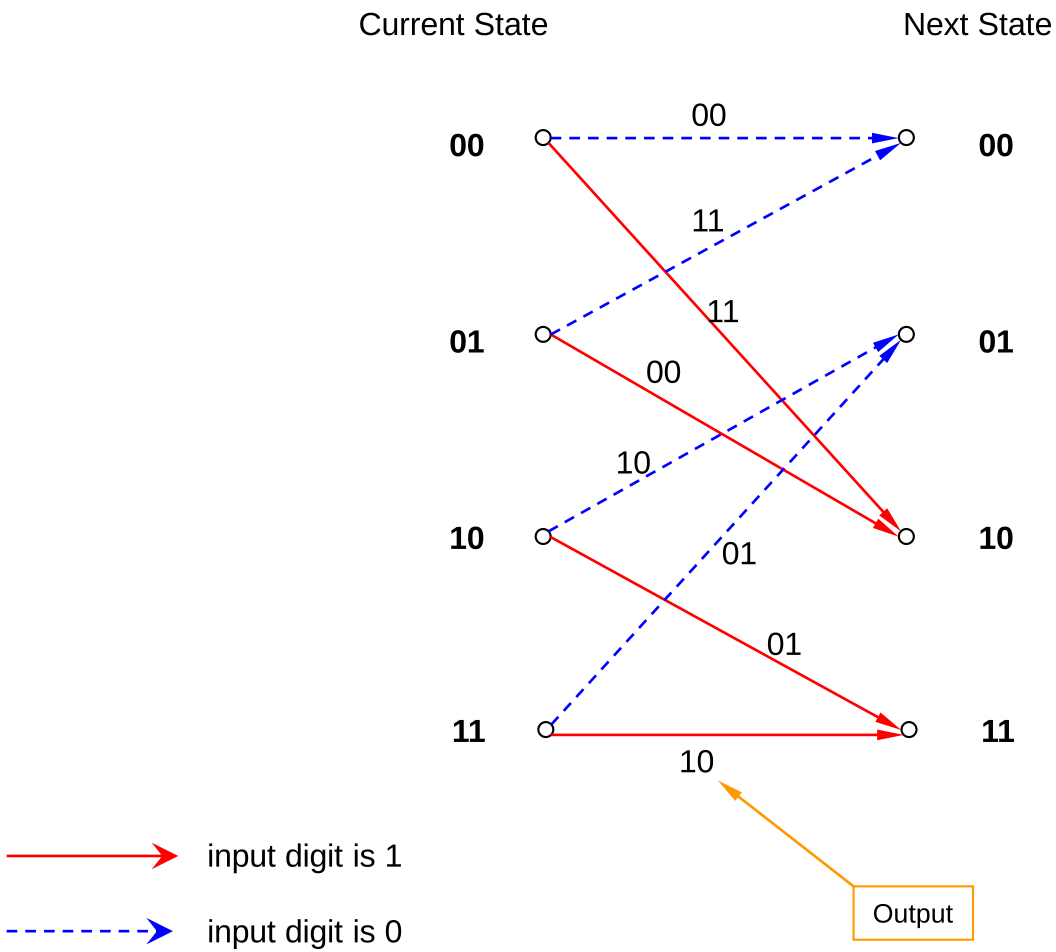
Final output of these blocks does not depend on the previous digits of the input sequence. The structure repeats itself.

So the input output relation can be shown on a state diagram

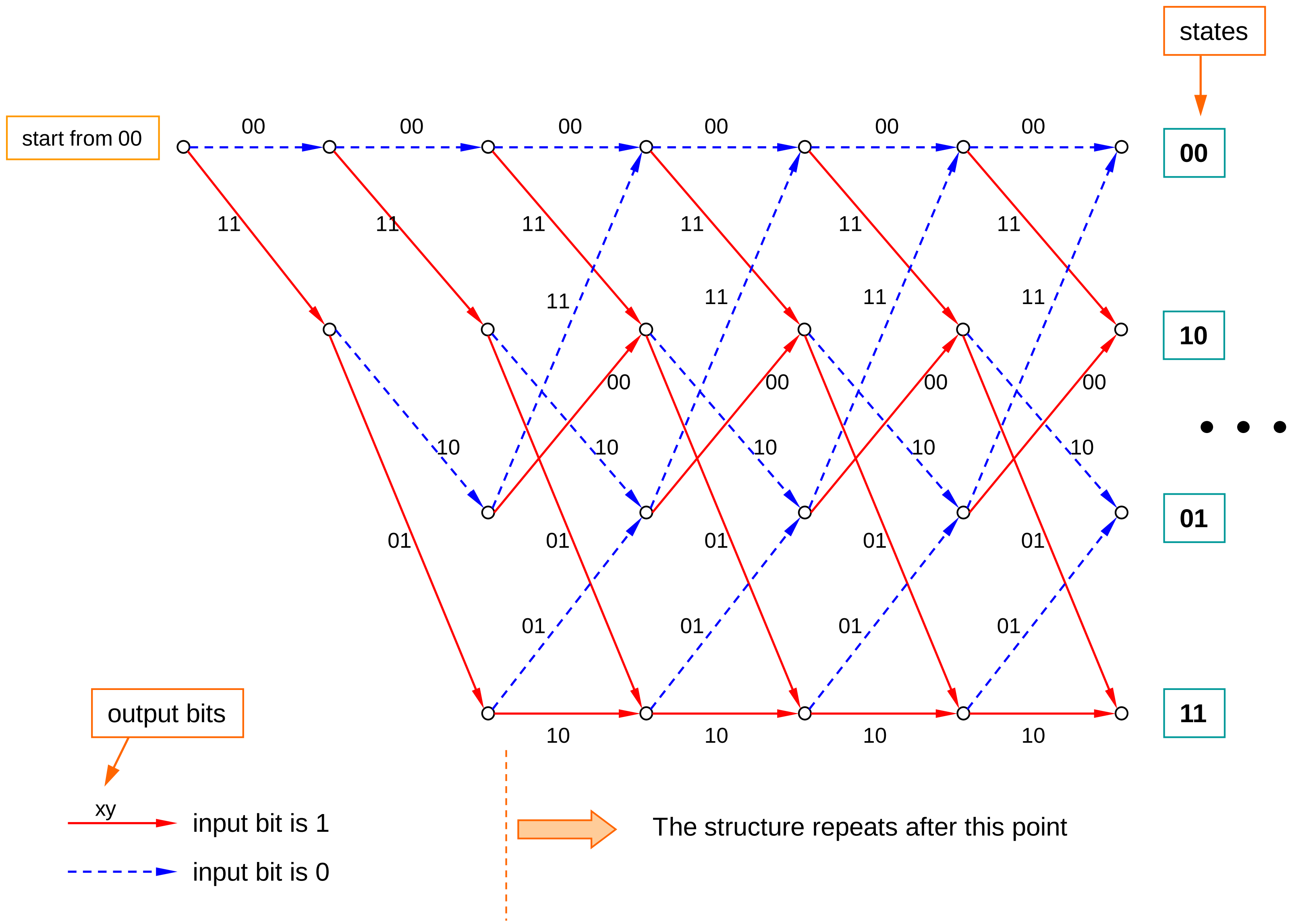


State Machine
start from 00
state
10
state
00
state
11
state
01
1 / 11
1 / 01
0 / 00
1 / 00
0 / 10
1 / 10
0 / 11
0 / 01
input digit is 1
x / yy = input_digit / output_digits
input digit is 0
path followed on the 11010000 input
state
ss
ss : register outputs

State Transition Graph

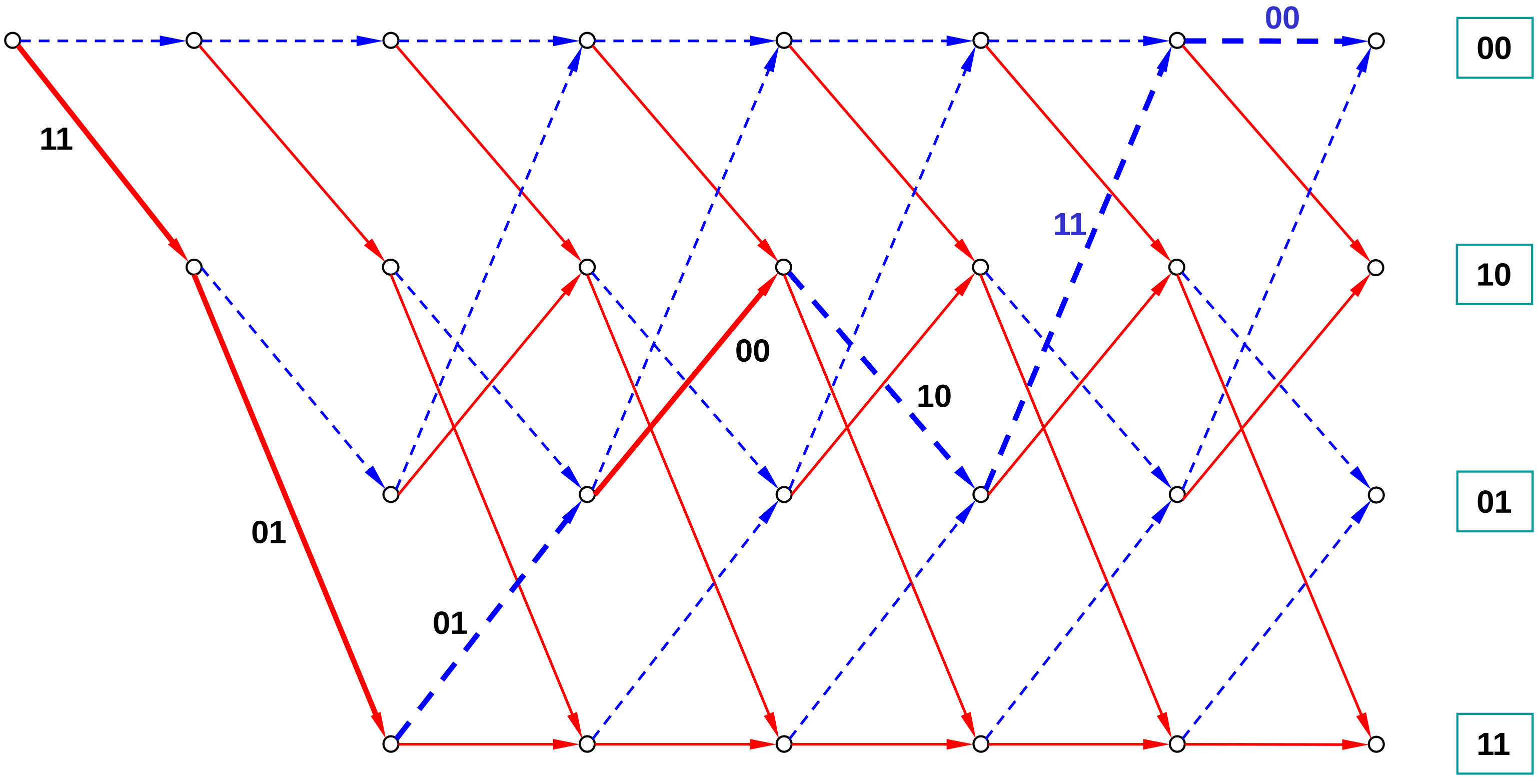


Trellis Diagram



Example Trellis

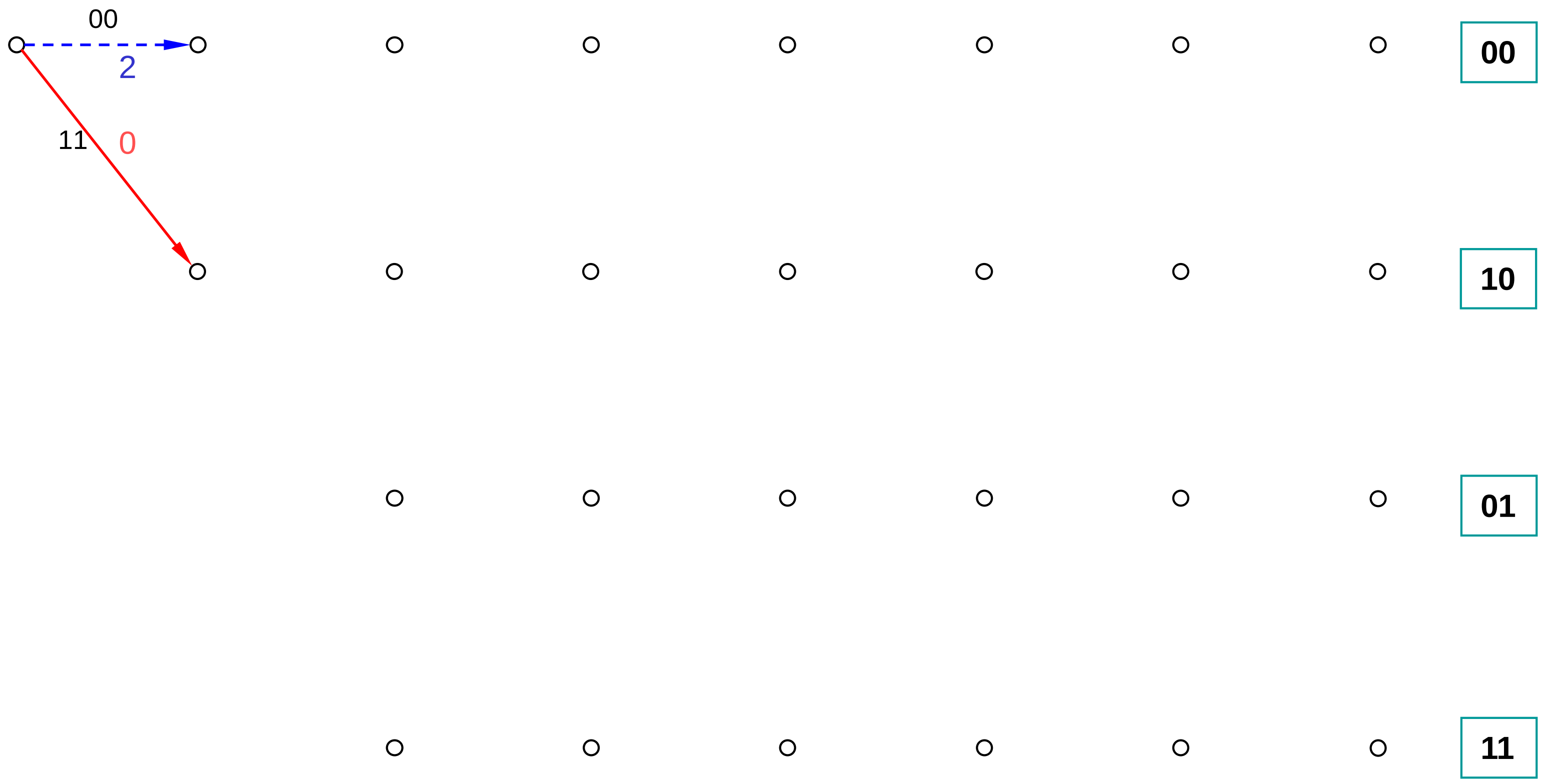
input = 1101000 output = 11 01 01 00 10 11 00



Trellis for the input 1101000 is marked with thick lines.
The last two input bits of 00 are appended to make the system ready for the next input stream/frame

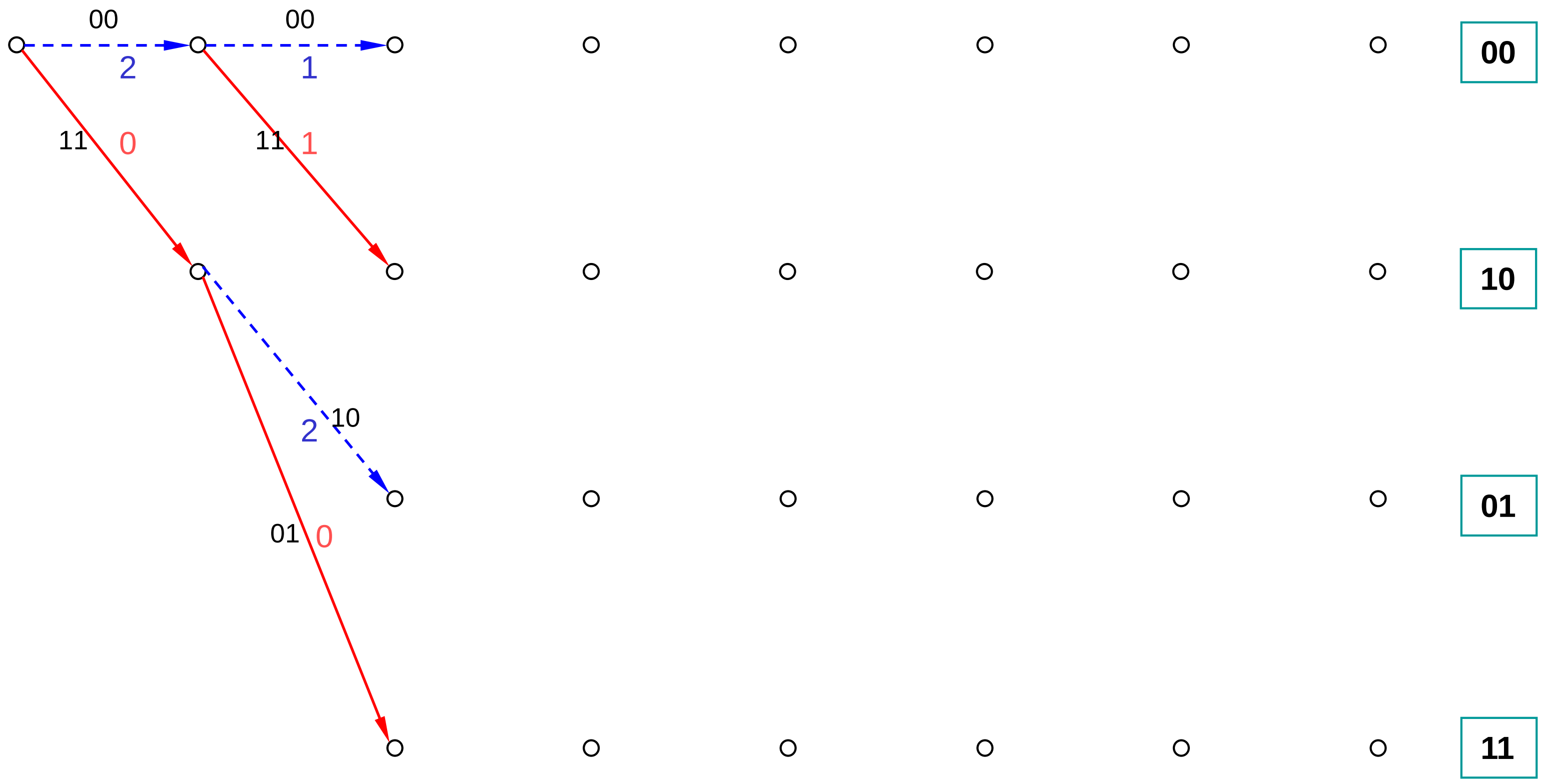
Decoding (Viterbi's Algorithm)

decoder input = **11** 01 01 01 10 11 00

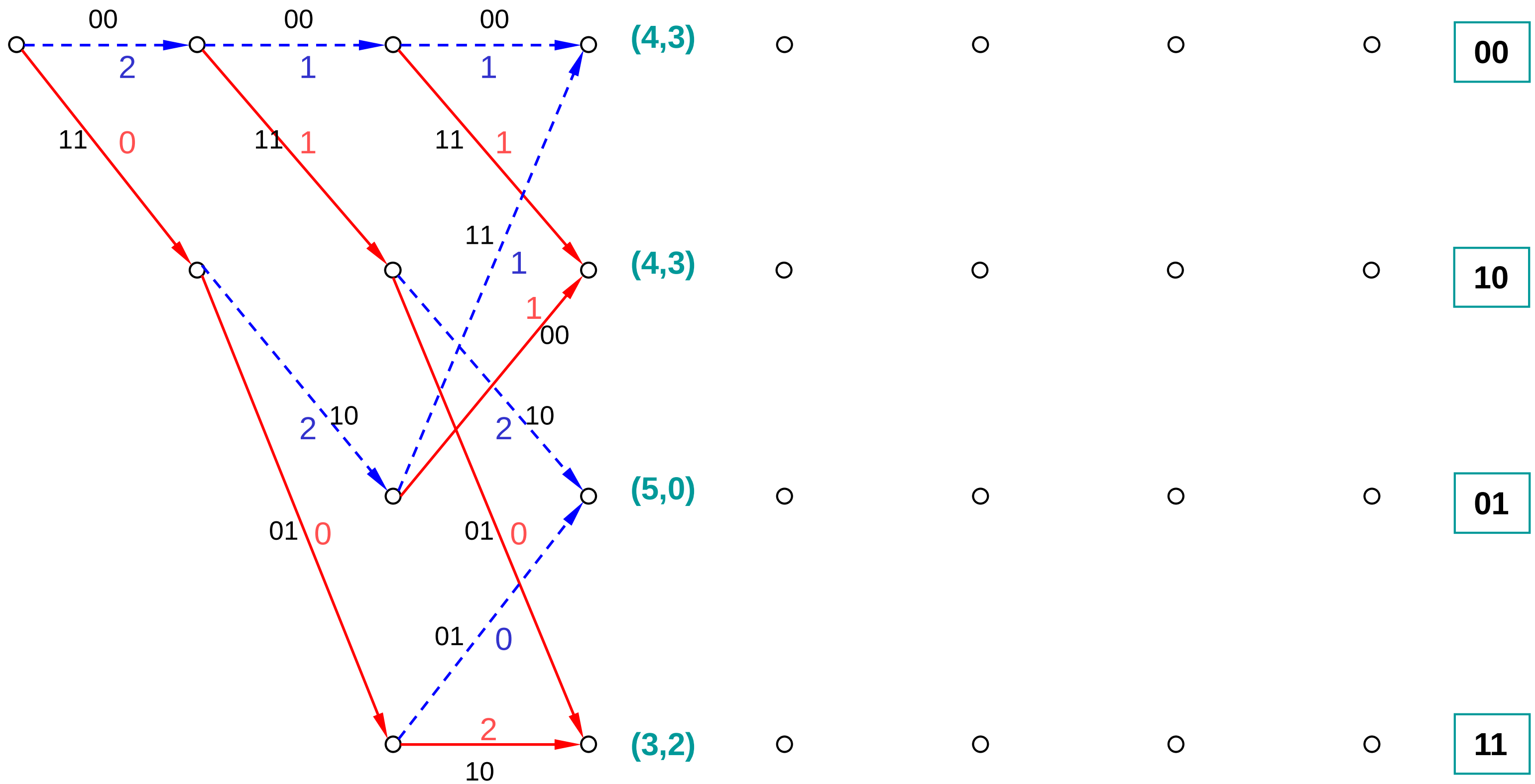


Find the Hamming distance between input bits and branch value. Mark this value on the branch.
Example: input value is 11, but the branch value is 00, then the Hamming distance is 2 as marked.

decoder input = 11 **01** 01 01 10 11 00

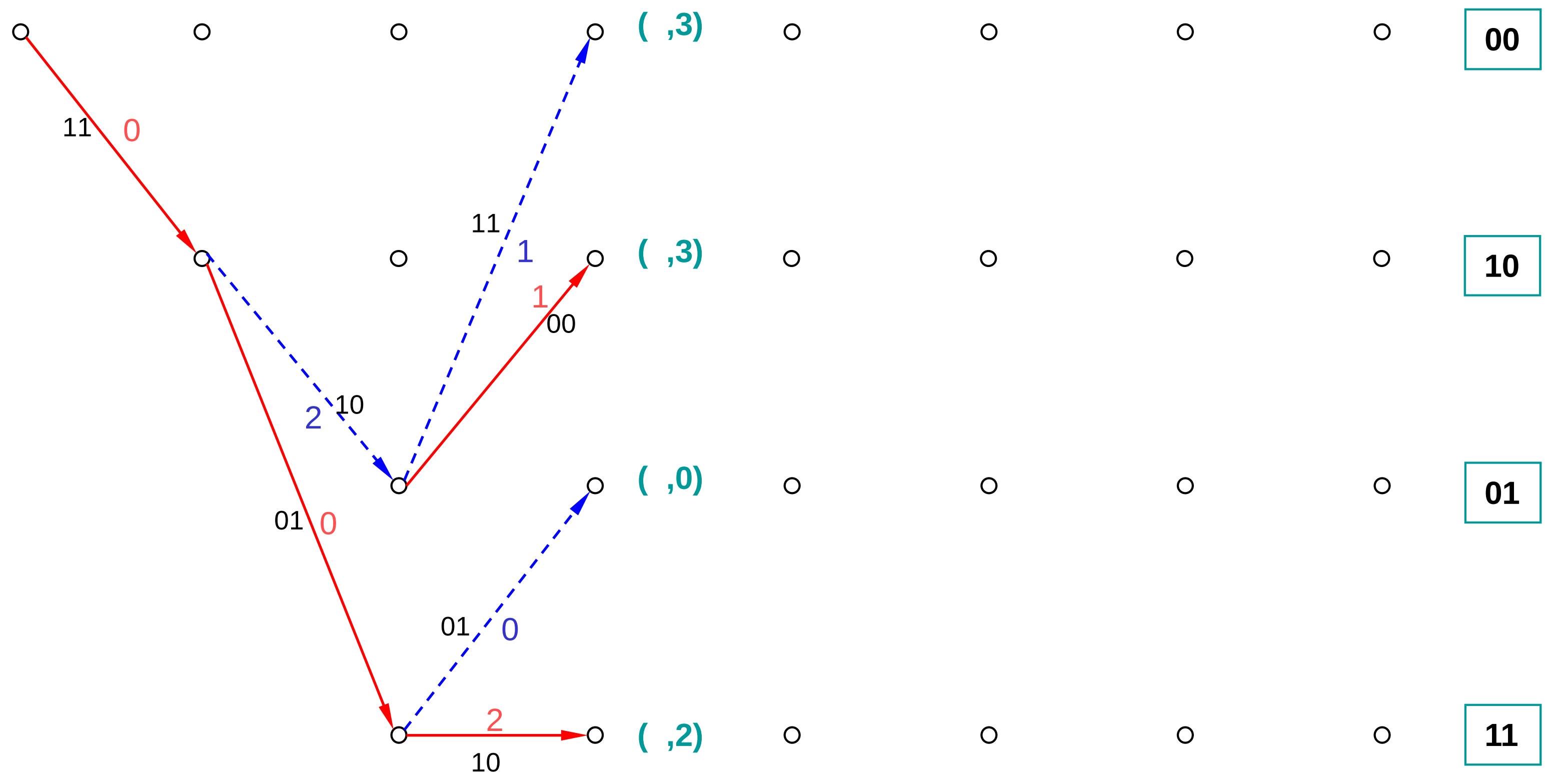


decoder input = 11 01 **01** 01 10 11 00



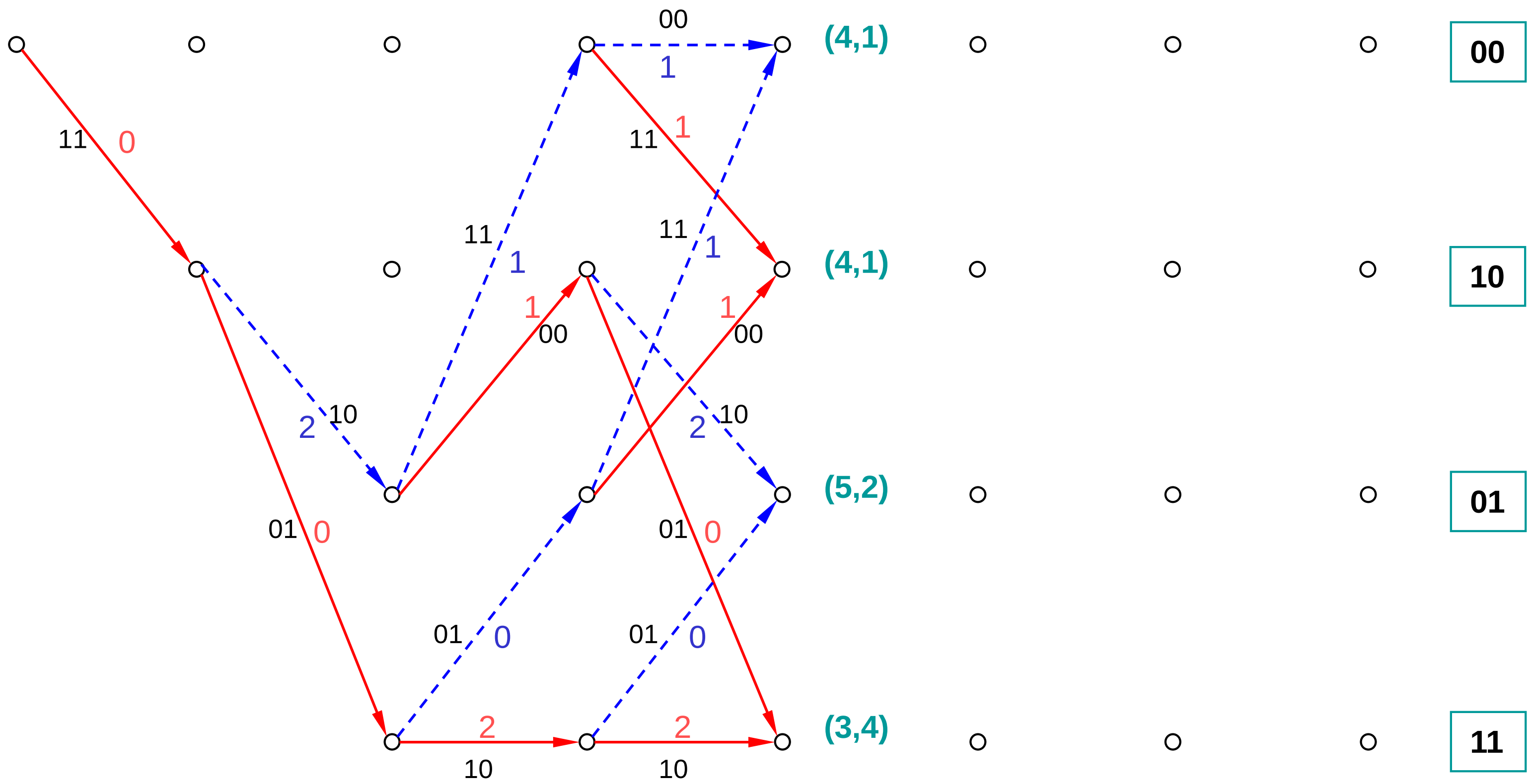
After the third branching, notice that each node is reached from only two predecessor node. Since what happens after this moment can not affect what happened up to this point, we compare the distance values of these two incoming branches and select the smaller one (Maximum Likelihood = Minimum Distance) and eliminate the other. We do this for each node. (The number of nodes = the number of states = 2^{CL-1})

decoder input = 11 01 **01** 01 10 11 00

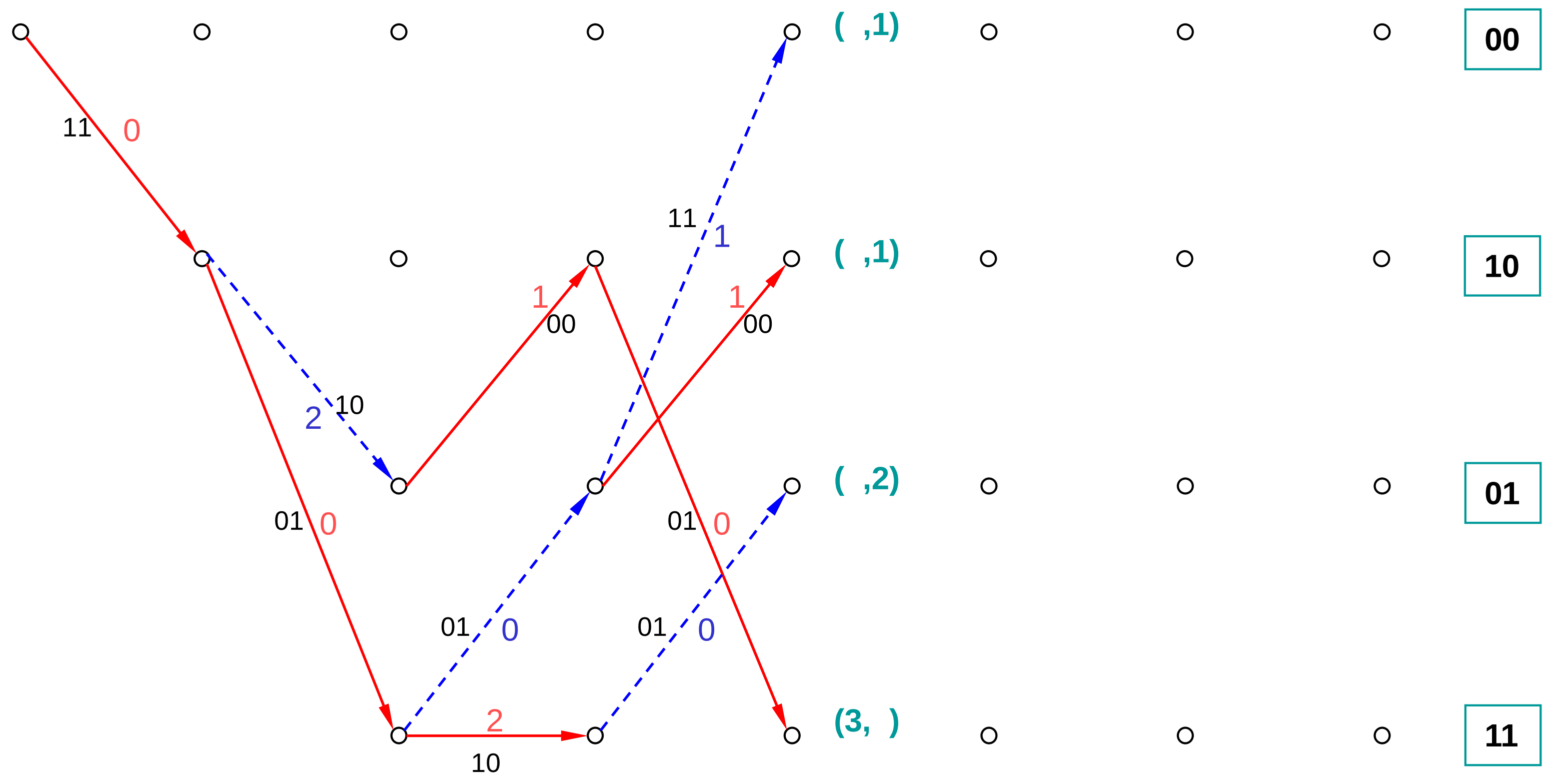


The remaining paths after elimination are called the "*survivors*".
Notice that we have a single branch survived at the beginning. It is called "*common stem*".
The decoder can output a data bit of "1" since 00->10 transition is caused by a "1".

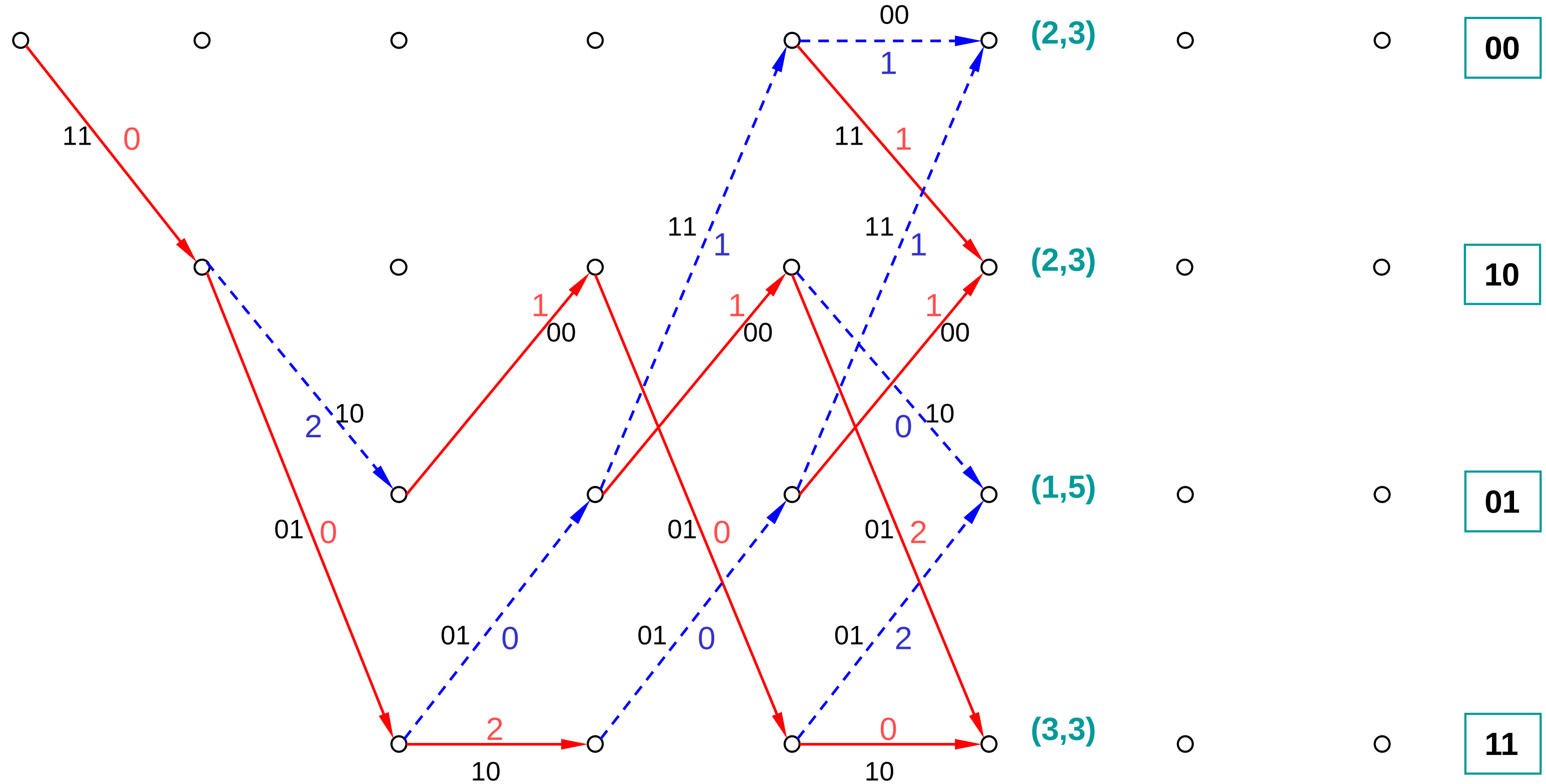
decoder input = 11 01 01 **01** 10 11 00



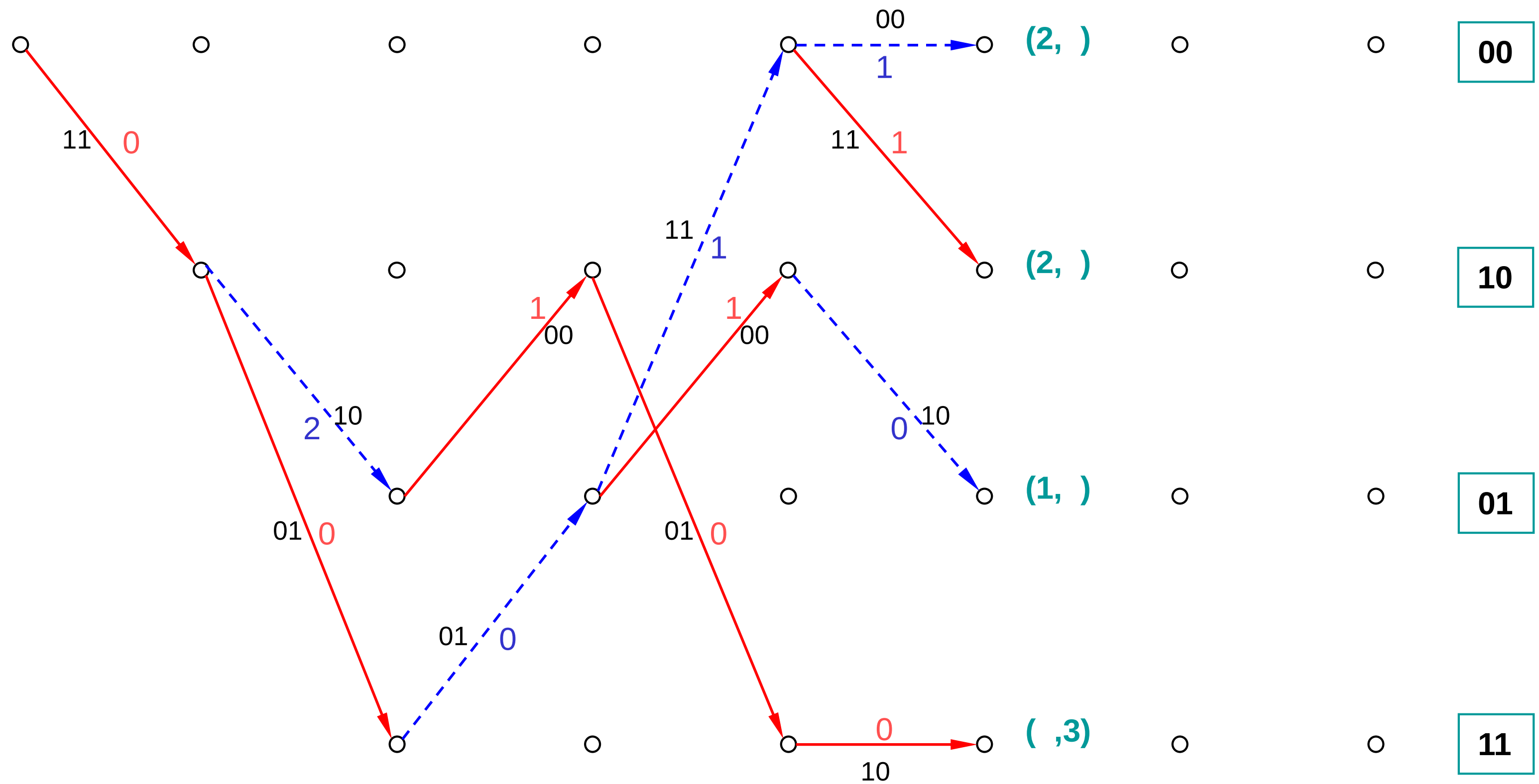
decoder input = 11 01 01 **01** 10 11 00



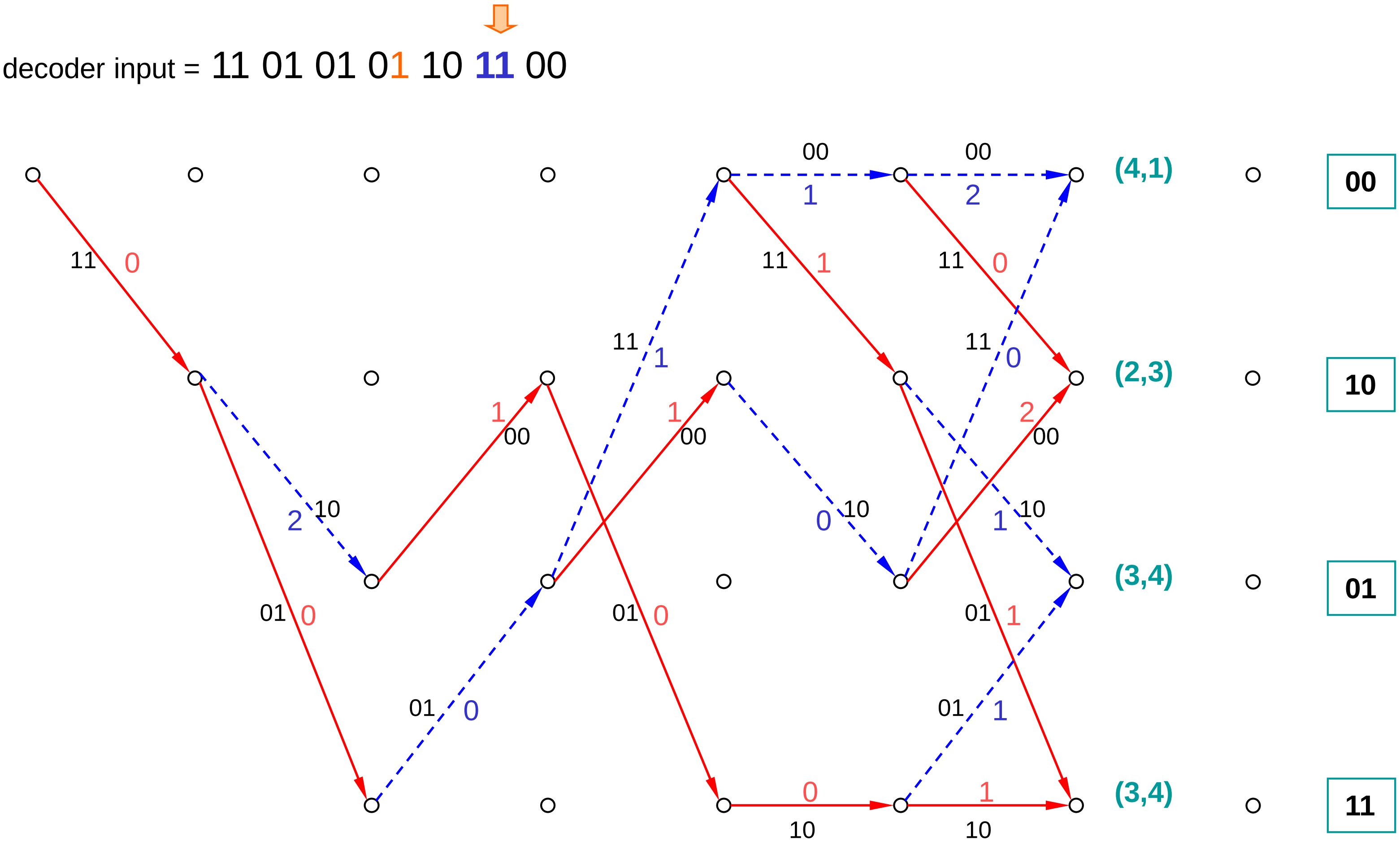
decoder input = 11 01 01 01 **10** 11 00



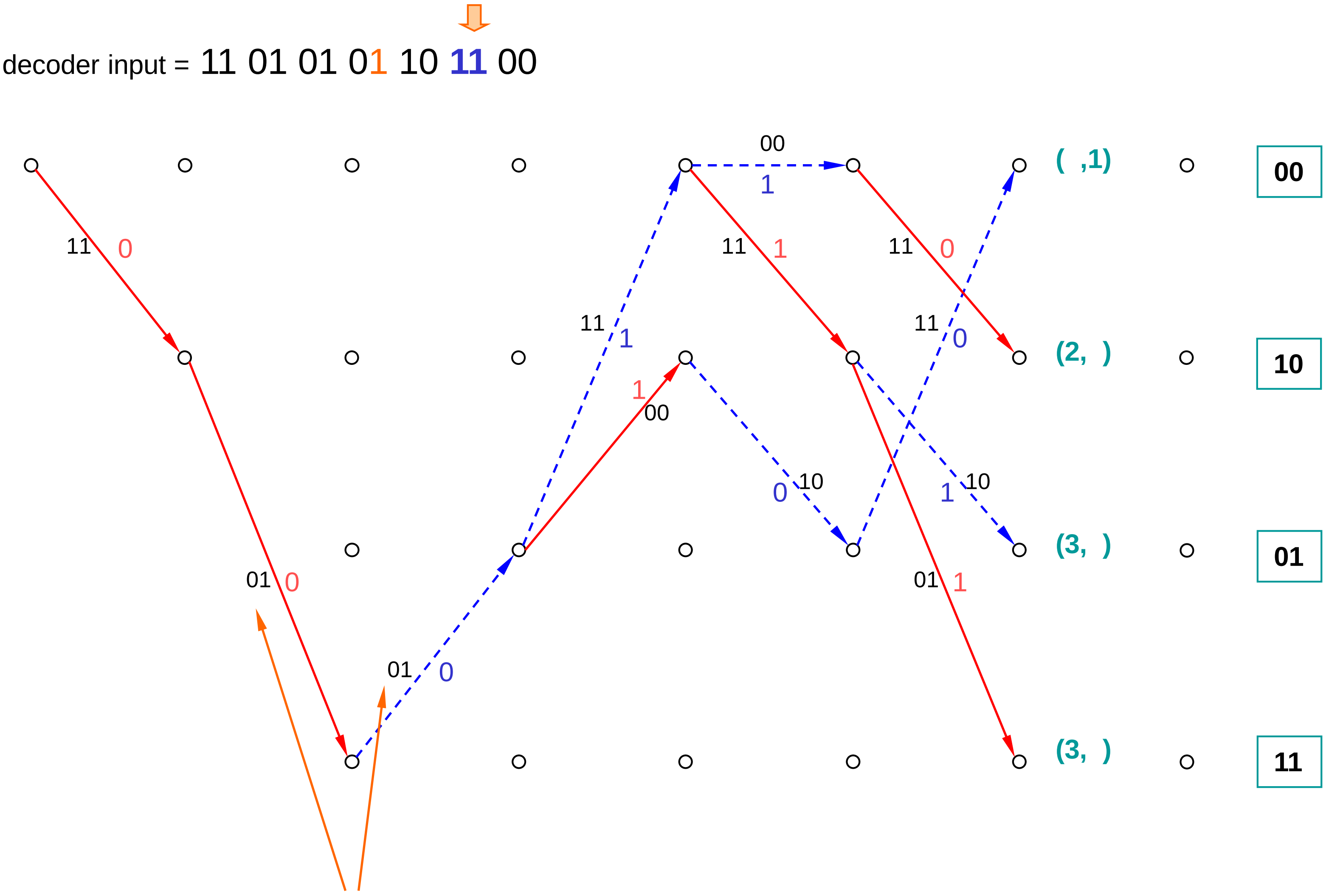
decoder input = 11 01 01 01 **10** 11 00



decoder input = 11 01 01 01 10 11 00

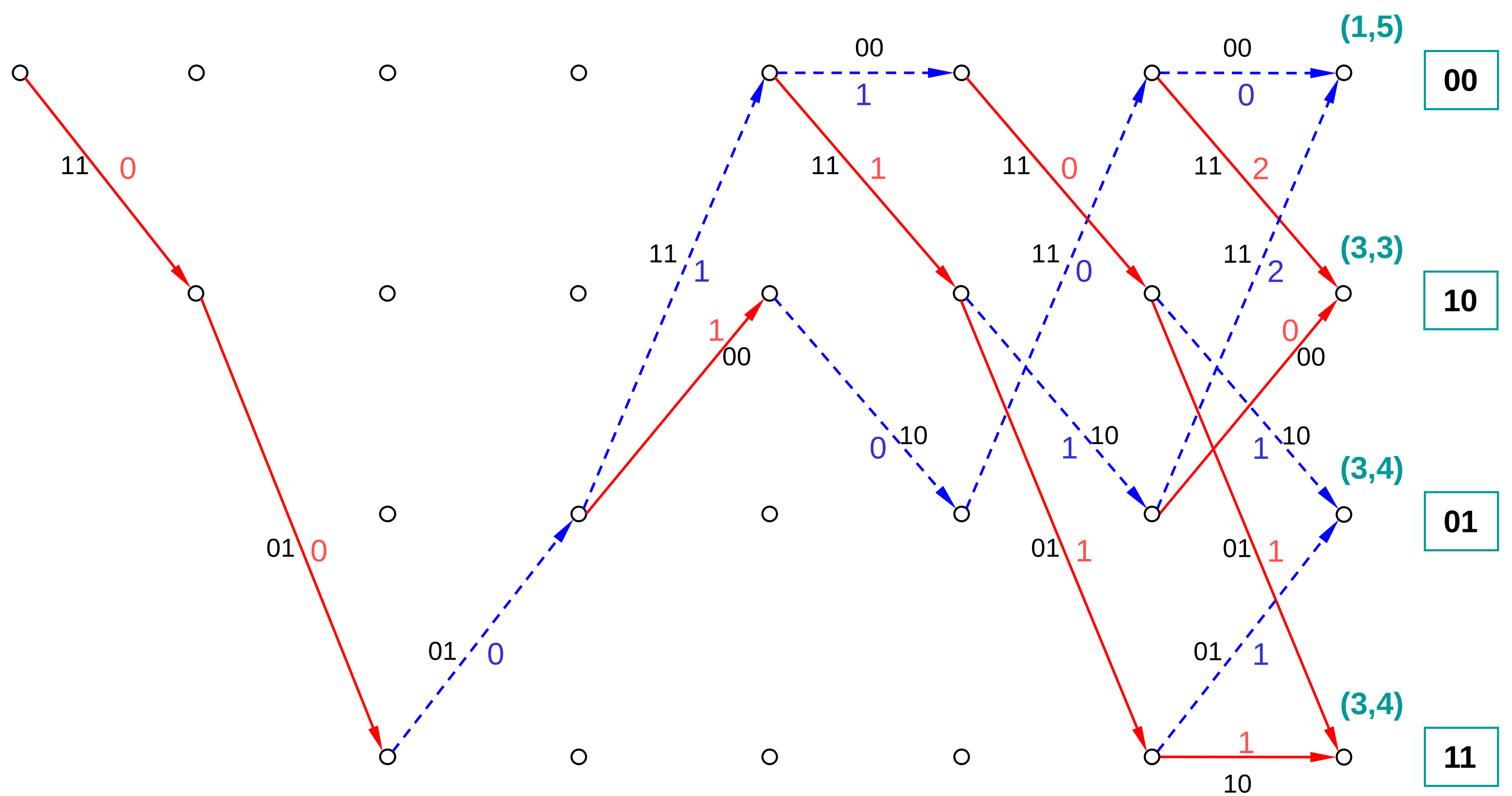


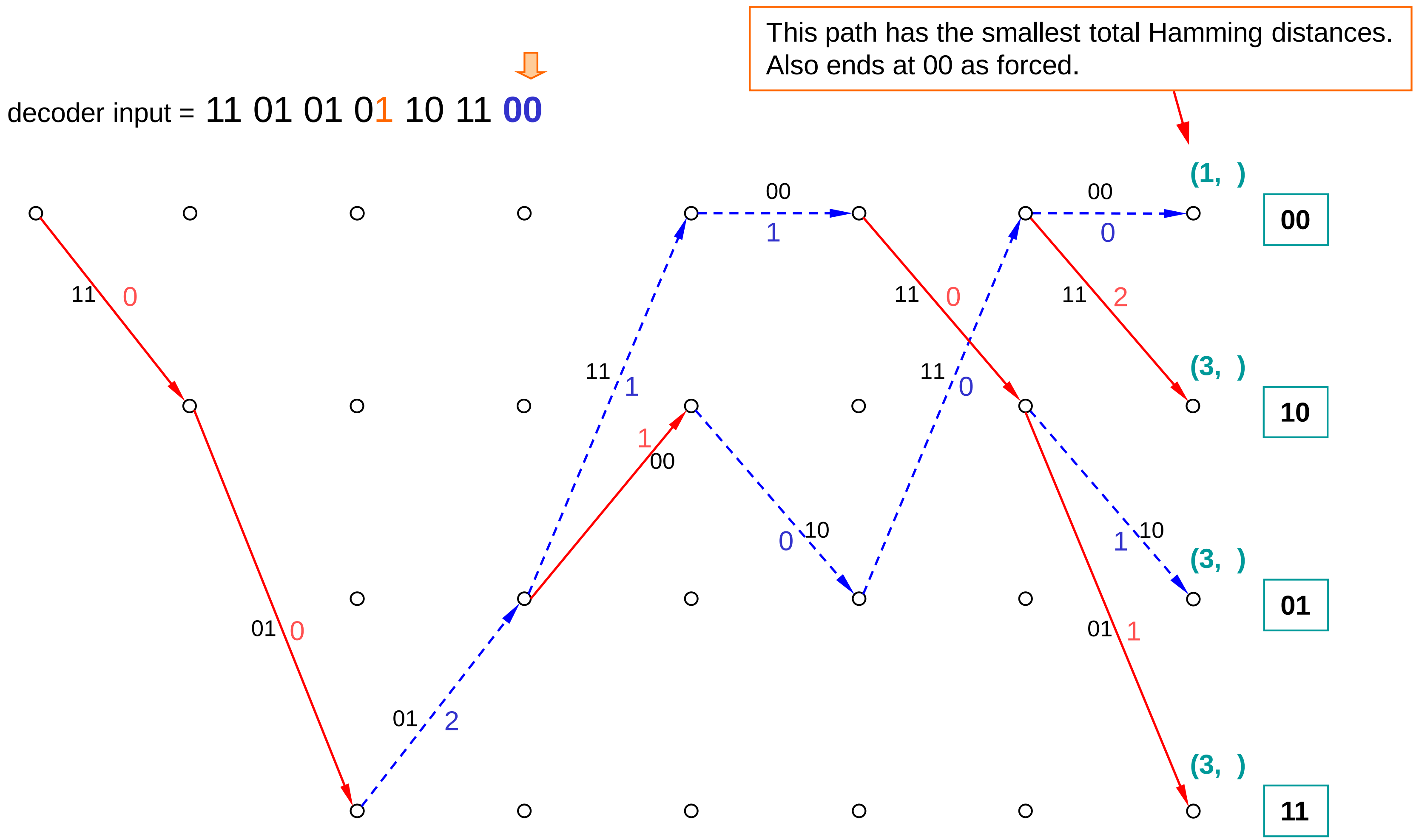
decoder input = 11 01 01 01 10 11 00



These two can be output now.
Depending on the errors the common stem may lag as much as 5 x constraint length.
This is a decoding delay. But still only 4 paths are kept in memory.

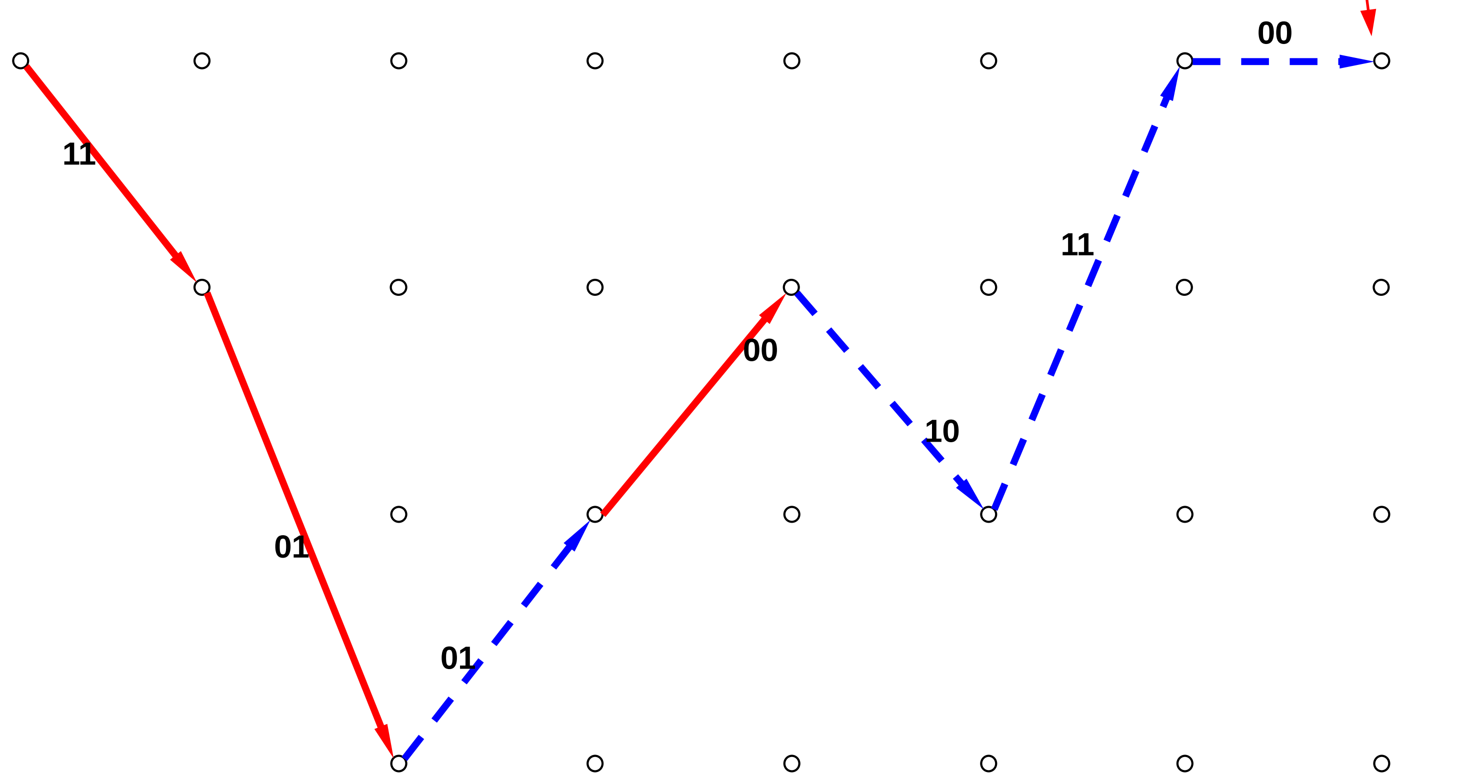
decoder input = 11 01 01 01 10 11 **00**



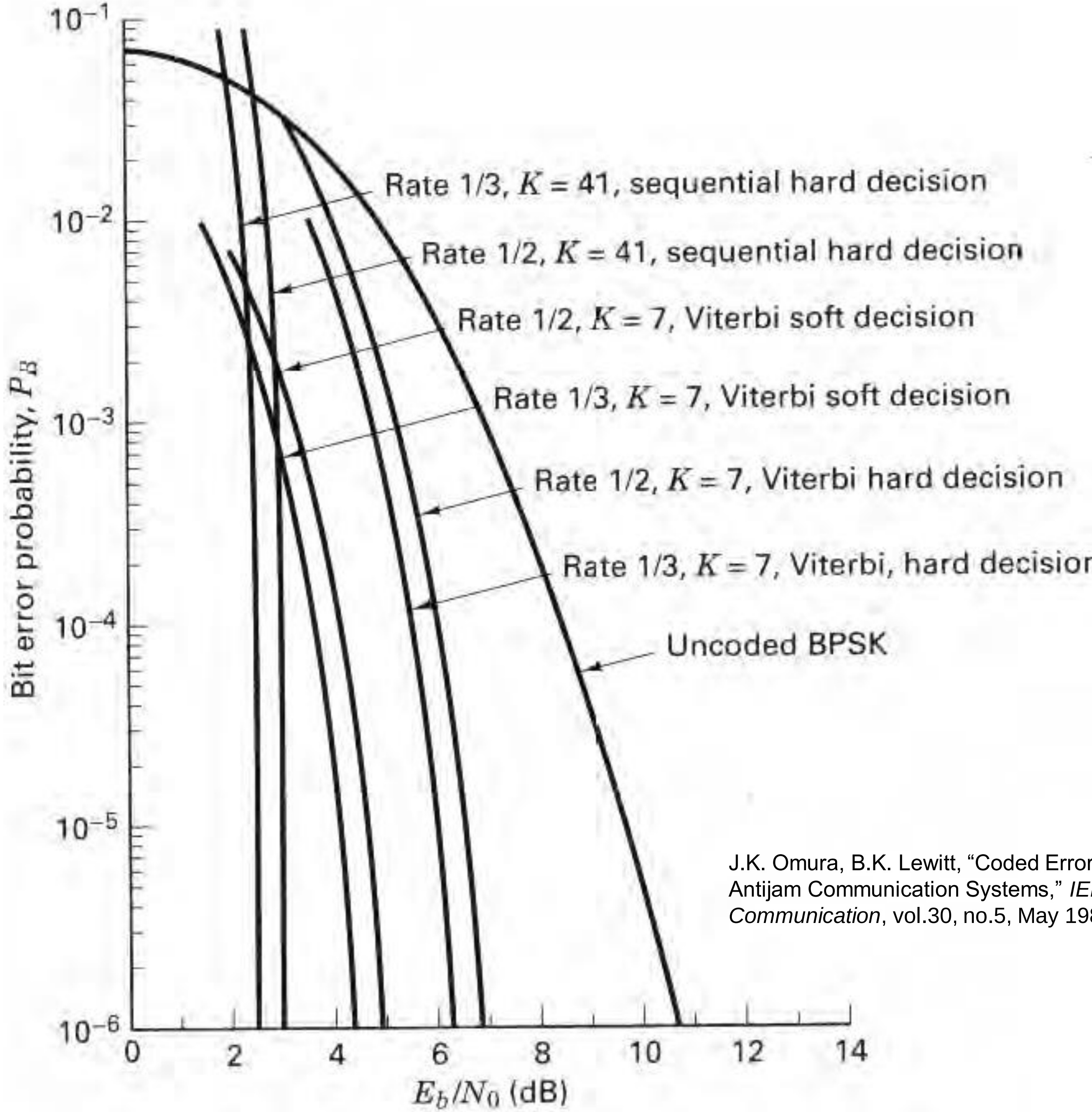

This path has the smallest total Hamming distances.
Also ends at 00 as forced.
decoder input = 11 01 01 01 10 11 00
(1,)
(3,)
(3,)
(3,)
00
10
01
11

Ready for the next input frame

decoder input = 11 01 01 01 10 11 00



decoder output = 11 01 01 00 10 11 00



J.K. Omura, B.K. Lewitt, "Coded Error Probability Evaluation for Antijam Communication Systems," *IEEE Transactions on Communication*, vol.30, no.5, May 1982

Optimum Rate 1/2 & 1/3 Convolutional Codes

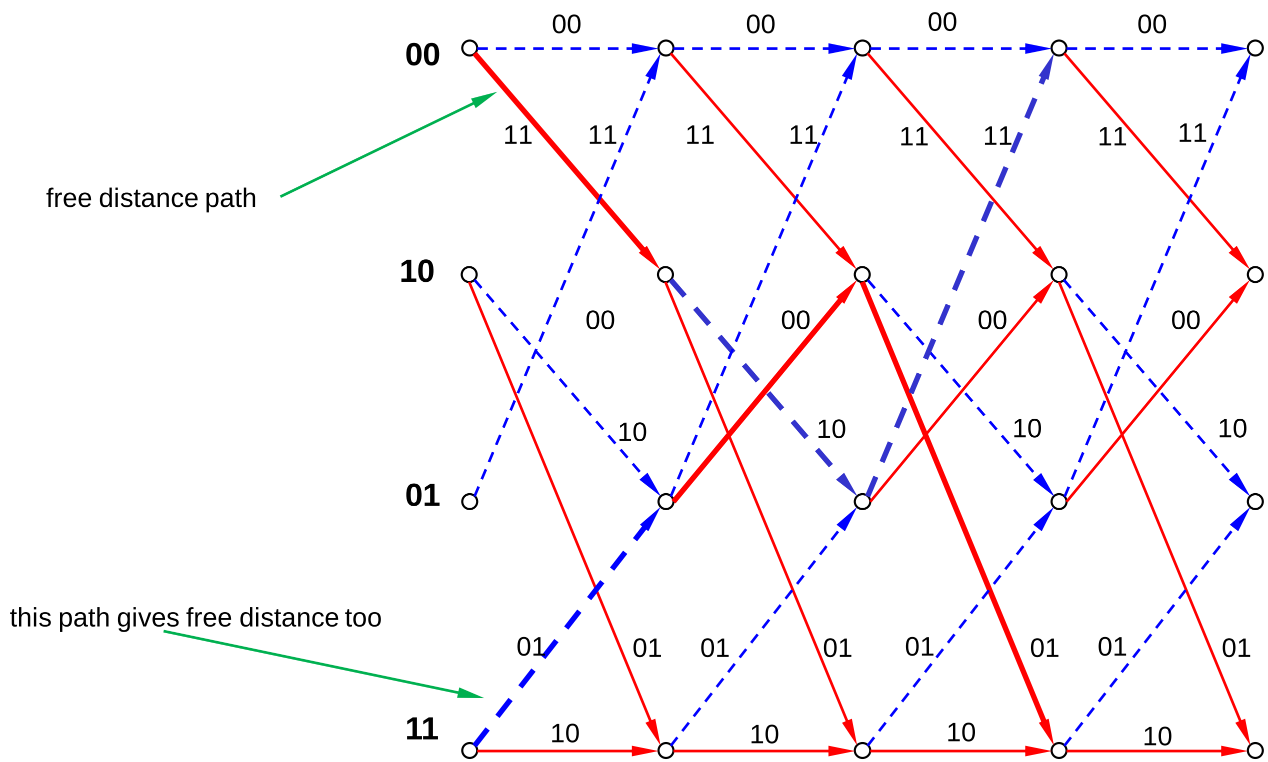
free distance =	5	6	7	8	10	10	12
	$\begin{bmatrix}111\\101\end{bmatrix}$	$\begin{bmatrix}1111\\1011\end{bmatrix}$	$\begin{bmatrix}10111\\11001\end{bmatrix}$	$\begin{bmatrix}101111\\110101\end{bmatrix}$	$\begin{bmatrix}1001111\\1101101\end{bmatrix}$	$\begin{bmatrix}10011111\\11100101\end{bmatrix}$	$\begin{bmatrix}110101111\\100011101\end{bmatrix}$

free distance =	8	10	12	13	15	16
	$\begin{bmatrix}111\\111\\101\end{bmatrix}$	$\begin{bmatrix}1111\\1011\\1101\end{bmatrix}$	$\begin{bmatrix}11111\\11011\\10101\end{bmatrix}$	$\begin{bmatrix}101111\\110101\\111001\end{bmatrix}$	$\begin{bmatrix}1001111\\1010111\\1101101\end{bmatrix}$	$\begin{bmatrix}11101111\\10011011\\10101001\end{bmatrix}$

J.P. Odenwalder, "Error Control Coding Handbook," Linabit Corp., *San Diego, Calif.*, July 1976

Free Distance

For block codes free distance is the minimum Hamming distance between codewords, and defines the error correcting capability of the code.
Convolutional codes work on streams, not the blocks info blocks. Free distance can be defined by sum of Hamming distances along the diverged (with an error) path, between …00… stream and the values on the path.
For the example coder, free distance path is the sum of 1s along the path shown thick below (since Hamming distances are calculated between the code and the 00 possibility). Therefore it is 5 for the example coder.



Error Correcting Capability

Error correcting capability of a block code is given by $t = \left\lfloor \frac{d_f - 1}{2} \right\rfloor$

t : the number of correctible errors in a codeword

d_f : minimum free distance (minimim distance between codewords)

Error correcting capability of a convolutional code is not that clear.
It obviously depends on the distribution of errors.

END