

# **AN INTRODUCTION TO TURBO CODES**

**Jack Keil Wolf  
ECE 154C**

**Spring 2009**

# **TURBO CODES AND ITERATIVE DECODING (CONTINUED)**

# TURBO CODES

- **Turbo codes** were introduced by C. Berrou, A. Glavieux and P. Thitimajshima in 1993.
- The name is derived from an iterative decoding algorithm used to decode these codes where, like a **turbo engine**, part of the output is reintroduced at the input and processed again.
- An **interleaver** is used to connect two or more **simple** constituent codes.

# TURBO CODES

- **The decoders for the constituent codes are very easy to implement.**
- **The decoders need to have soft inputs and outputs.**
- **The output of one decoder is used as the input of the other decoder.**
- **Many decoding iterations are required.**

# TURBO CODES

- **Although the original turbo code consisted of the parallel combination of two constituent codes, other variations exist.**
- **Some things we need to know about:**
  - Iterative decoding
  - Recursive convolutional encoders
  - Random-like interleavers
  - Extrinsic and intrinsic information
  - Soft-in, soft-out (SISO) decoders

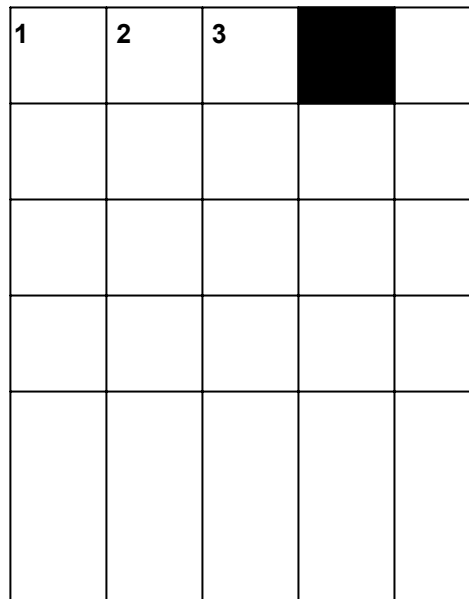
# ITERATIVE DECODING AND CROSSWORD PUZZLES

- Consider two people doing a crossword puzzle where one person sees only the **row clues** and the other person sees only the **column clues**.
- Also assume that the two persons take turns in filling in the puzzle: rows, columns, rows, columns, etc.
- To make the example closer to decoding turbo codes, assume that each person fills in the spaces with their best guess as to the **probabilities** of the possible letters in that space.

# ITERATIVE DECODING AND CROSSWORD PUZZLES

- For example if the row clue for a 3 letter word is “household pet”, the first person might guess that the first letter is a “d” (for dog) with probability 0.5, “c” (for cat) with probability 0.4, or “e” (for eel) with probability 0.1.
- The other person would take these probabilities and refine them based upon the column clues.
- The refined probabilities are fed back to the first person and the process continues until either the crossword puzzle is completed or a failure occurs.

# CROSSWORD PUZZLE



## Across

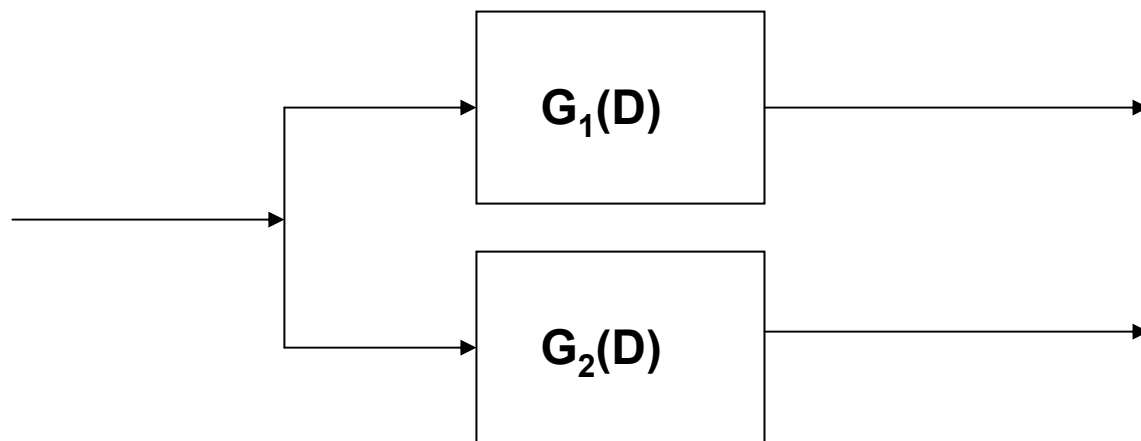
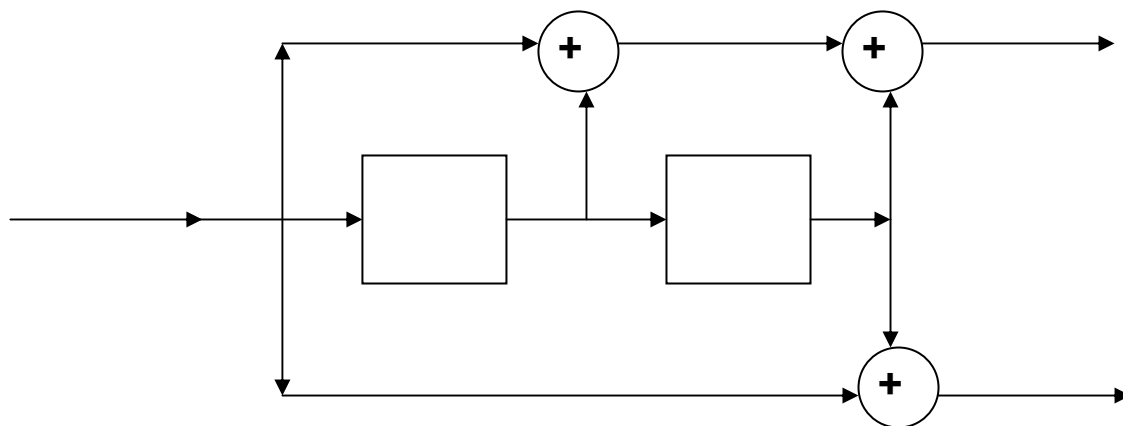
1. Household pet

## Down

1. Opposite of "up"

# SYSTEMATIC, RECURSIVE CONVOLUTIONAL ENCODER

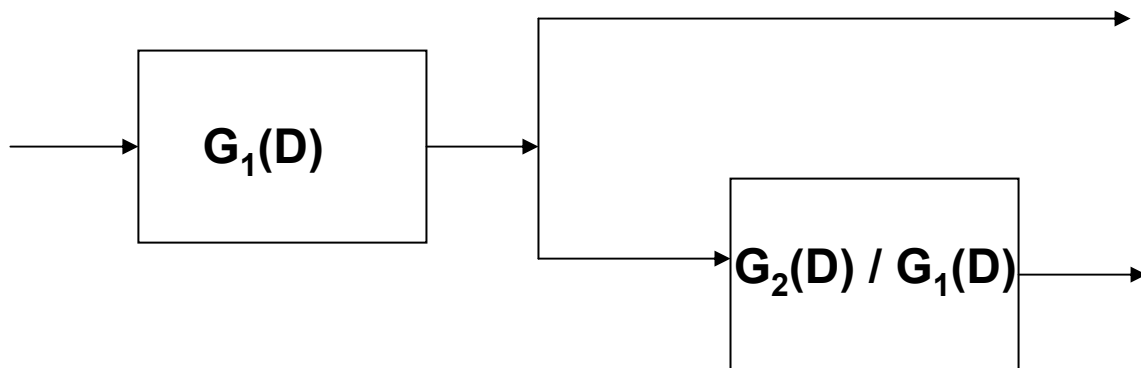
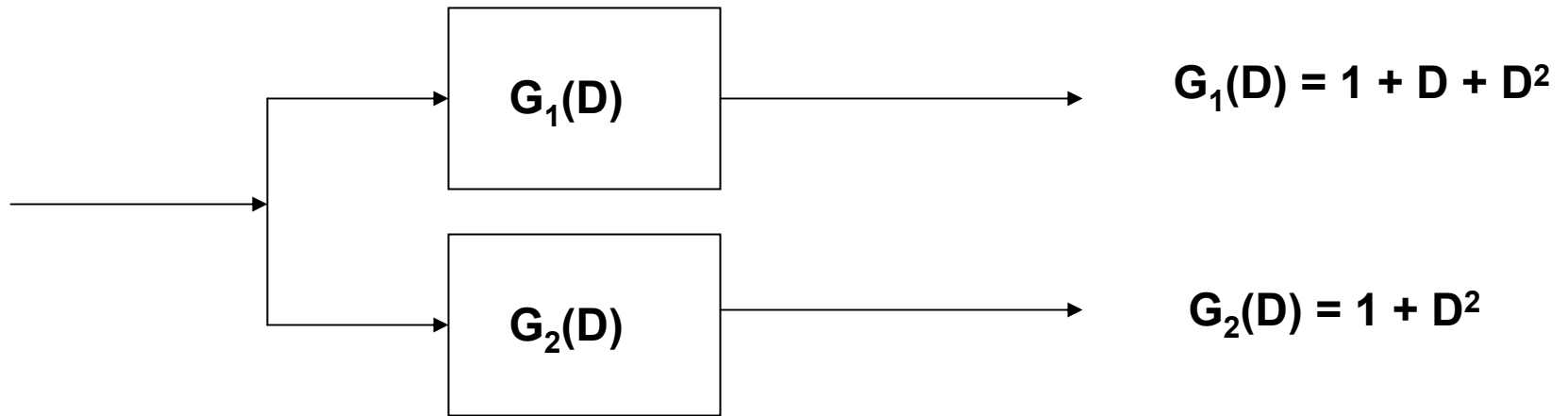
- **Example: Nonsystematic rate  $\frac{1}{2}$  4 state encoder.**



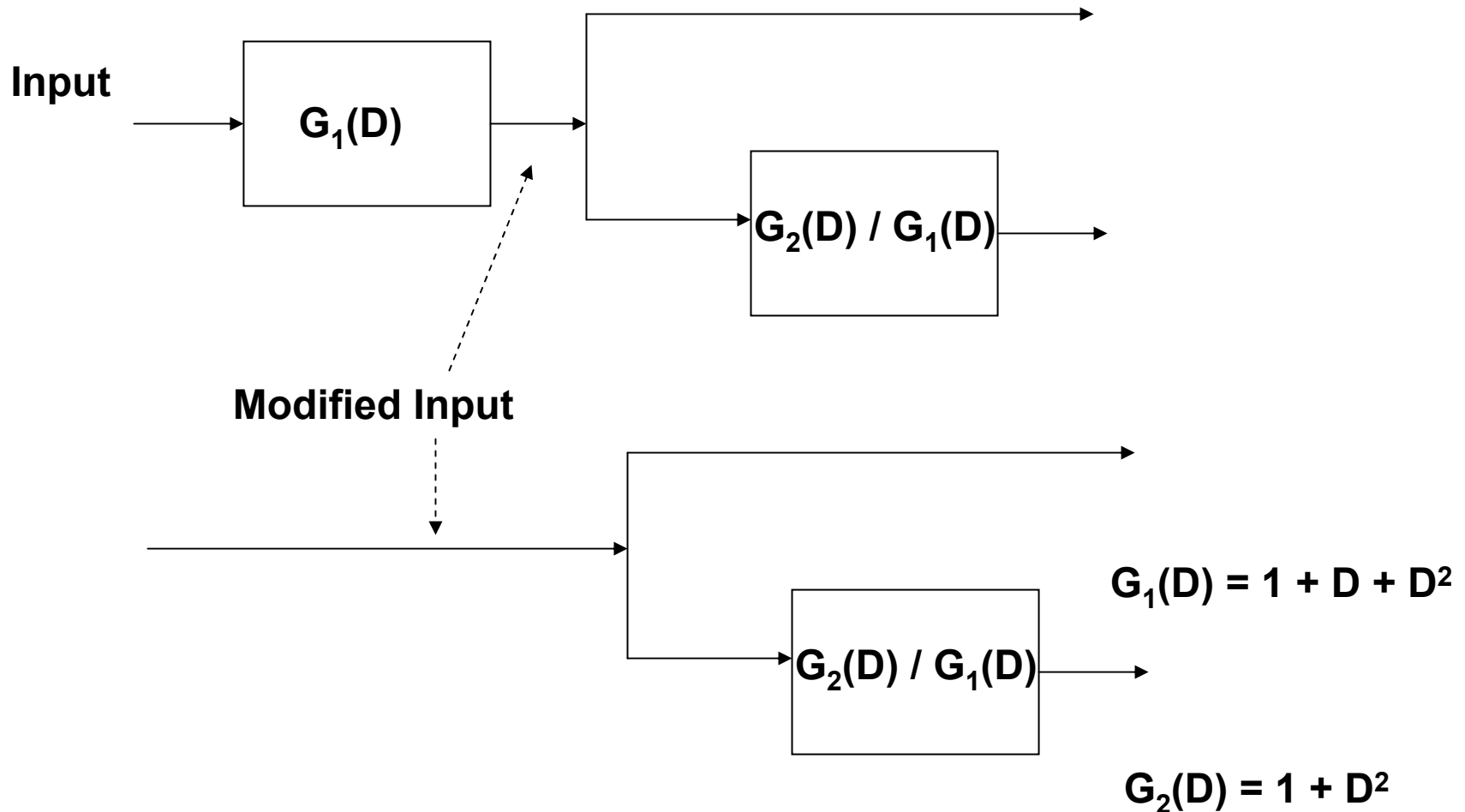
$$G_1(D) = 1 + D + D^2$$

$$G_2(D) = 1 + D^2$$

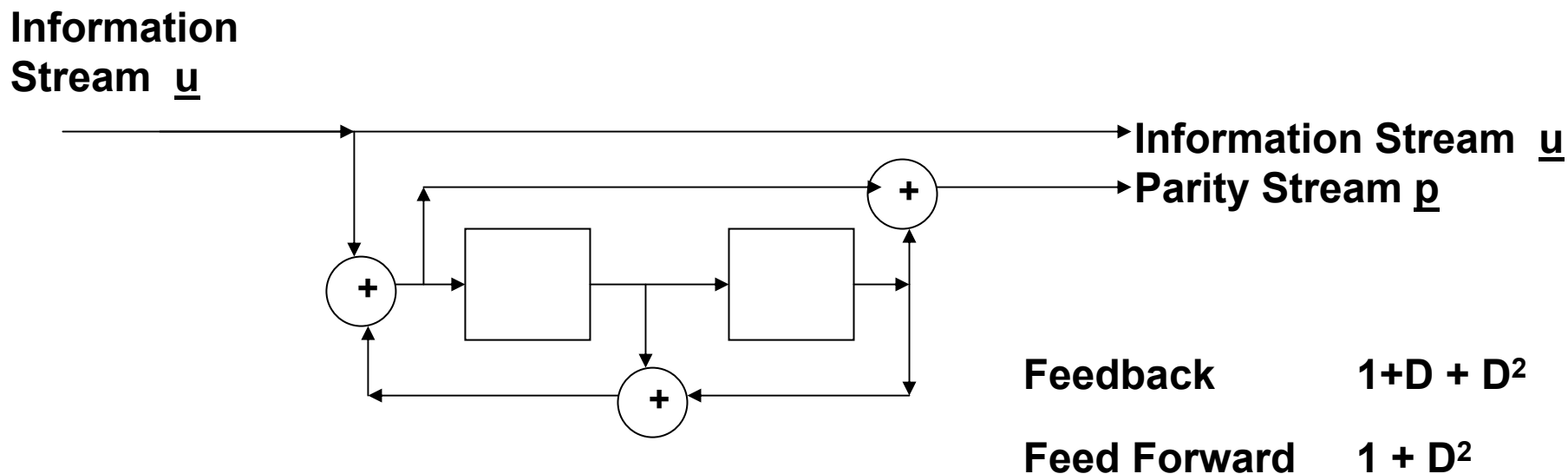
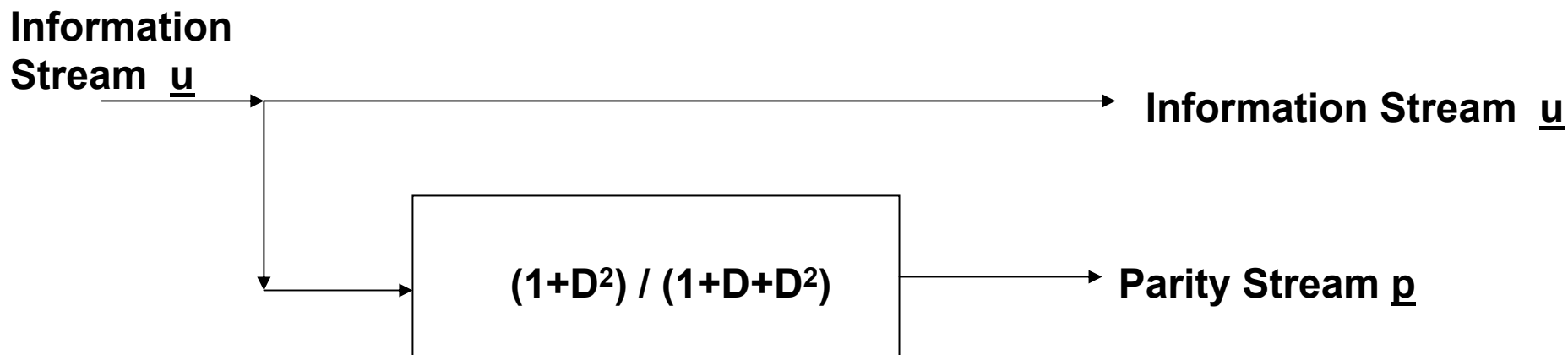
# SYSTEMATIC, RECURSIVE CONVOLUTIONAL ENCODER



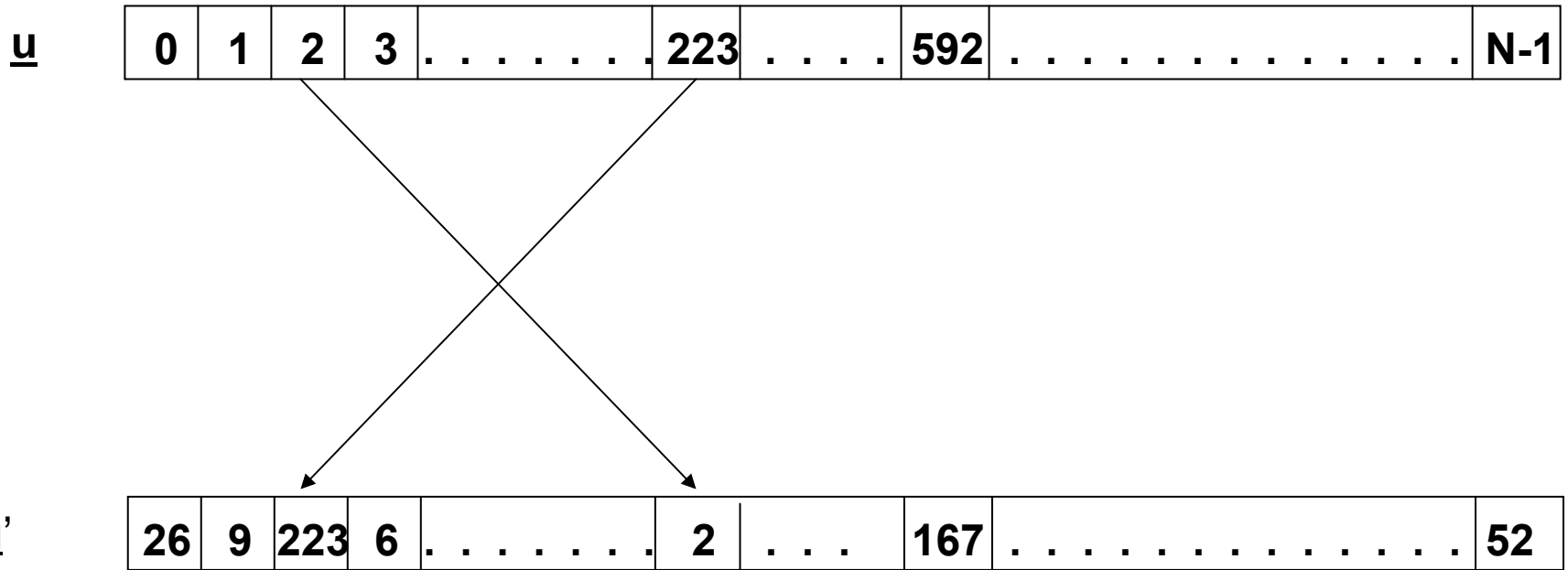
# SYSTEMATIC, RECURSIVE CONVOLUTIONAL ENCODER



# SYSTEMATIC, RECURSIVE CONVOLUTIONAL ENCODER



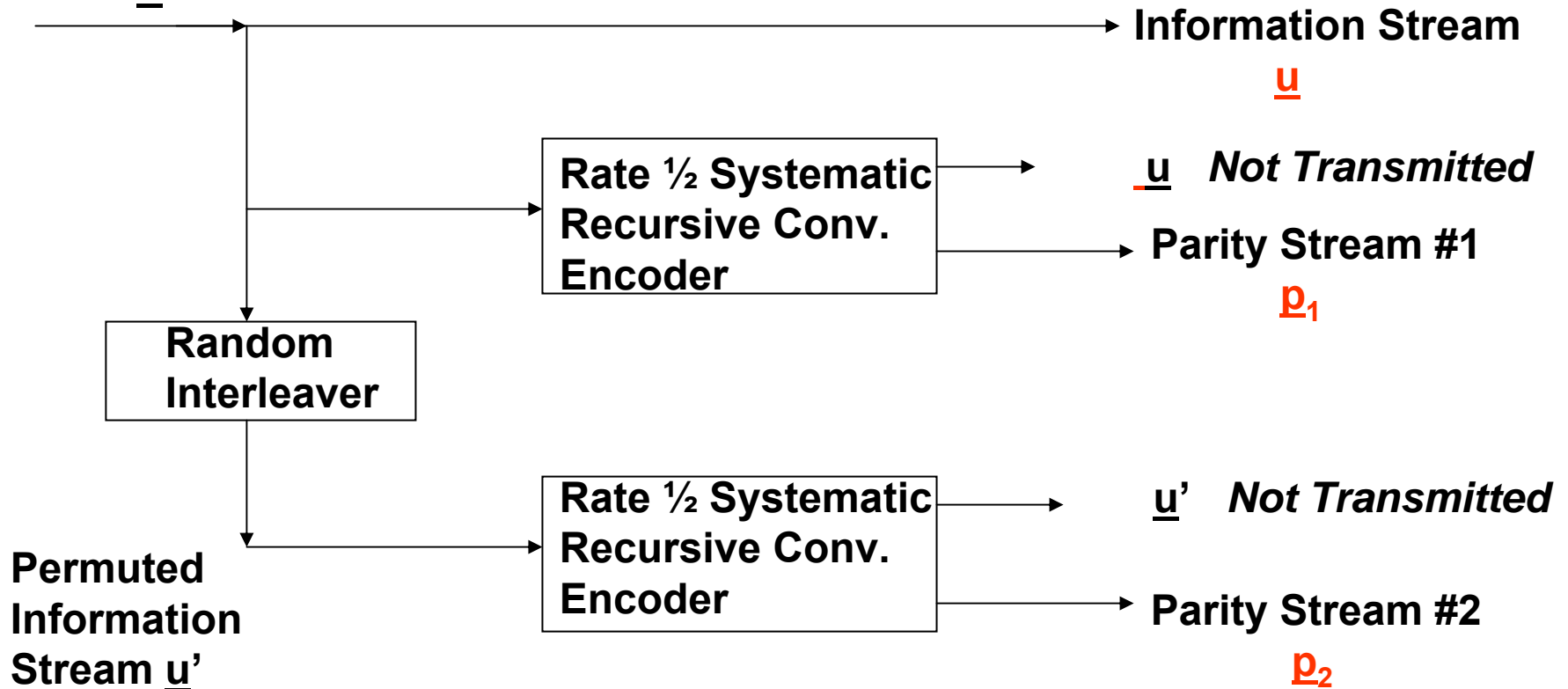
# RANDOM-LIKE INTERLEAVER



# A PARALLEL TURBO ENCODER

- Example of a rate 1/3 Turbo encoder:

Information Stream  $\underline{u}$

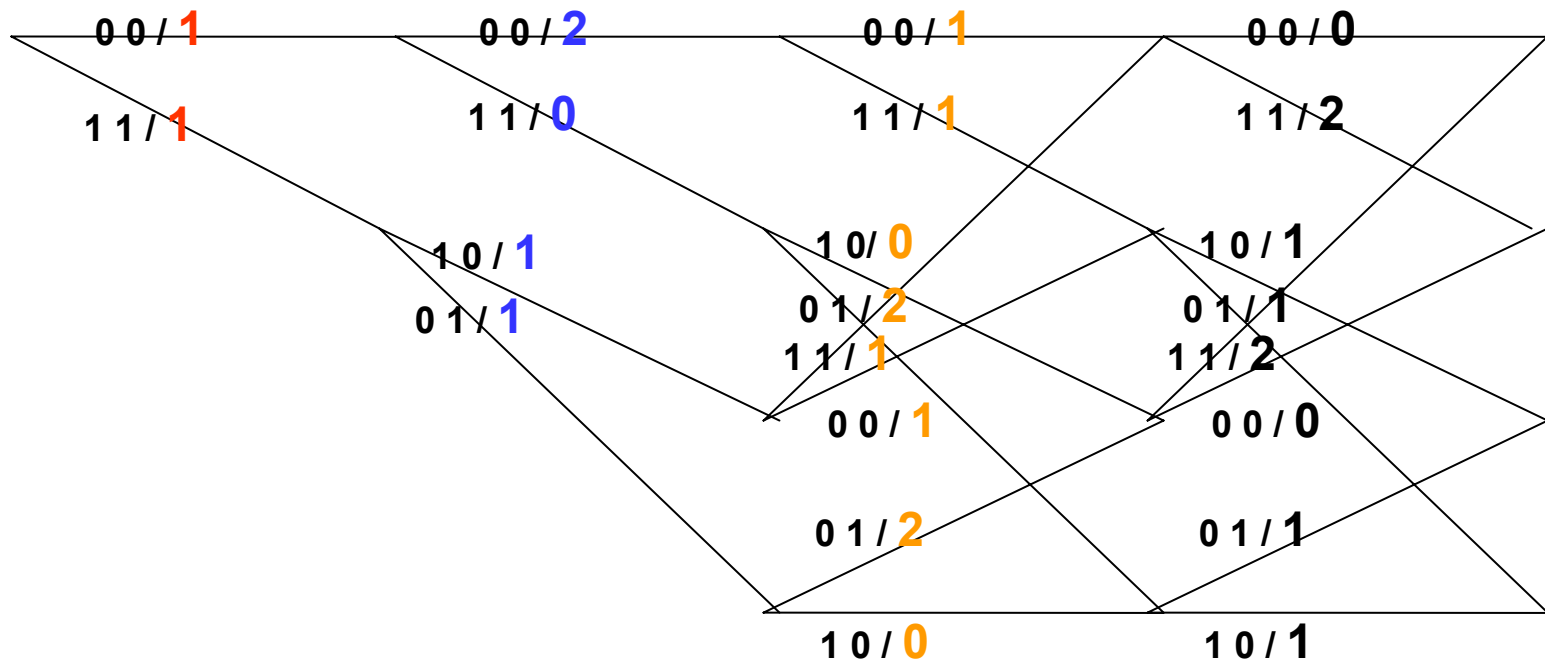


# A PARALLEL TURBO ENCODER

- The result is a rate  $1/3$  parallel turbo encoder.
- For each input bit  $u_i$ , the encoder outputs 3 binary digits:  $u_i$ ,  $p_{1,i}$ ,  $p_{2,i}$ .
- To understand decoding, we must first understand how a convolutional decoder can use soft channel information.

# SOFT-IN, SOFT-OUT DECODERS

- The previously described Viterbi decoder used hard-outputs (i.e., 0's and 1's) from the channel.
- For received sequence 0 1 1 1 1 0 0 0 ...



# SOFT-INPUT VITERBI DECODER

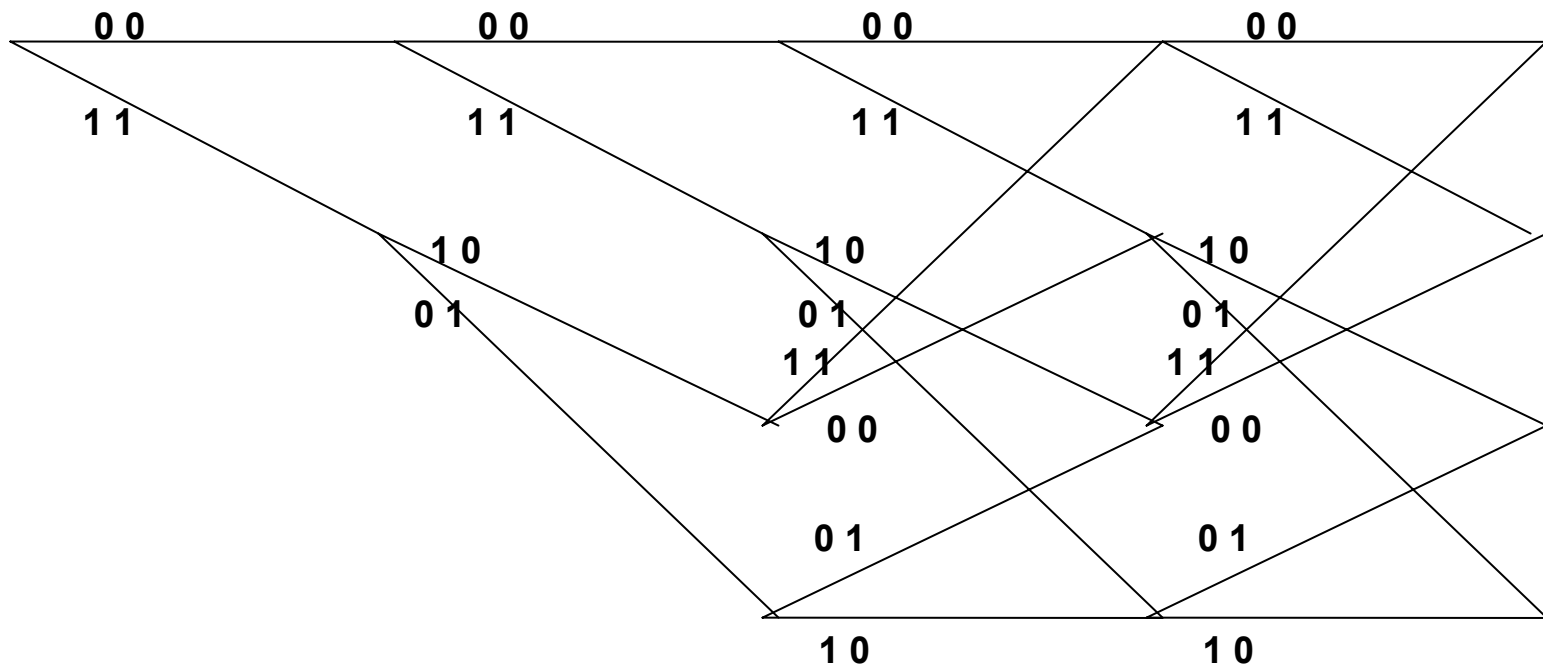
- A **soft-input** Viterbi decoder would use the probabilities (or the logarithm of the probabilities) of the bits.
- Then one would take the path corresponding to the sequence of bits with the highest probability.
- Using logarithms allows the decoder to use the **Add-Compare-Select** algorithm since the probability of a sequence is the product of the probabilities but the logarithm of the probability of a sequence is the sum of the logarithms of the probabilities.

# SOFT-INPUT VITERBI DECODER

- For a Gaussian noise channel, the unquantized soft branch metrics corresponding to  $(r_i, r_{i+1})$  should be:

$$0\ 0 \rightarrow (r_i+1)^2+(r_{i+1}+1)^2 \quad 0\ 1 \rightarrow (r_i+1)^2+(r_{i+1}-1)^2$$

$$1\ 0 \rightarrow (r_i-1)^2+(r_{i+1}+1)^2 \quad 1\ 1 \rightarrow (r_i-1)^2+(r_{i+1}-1)^2$$

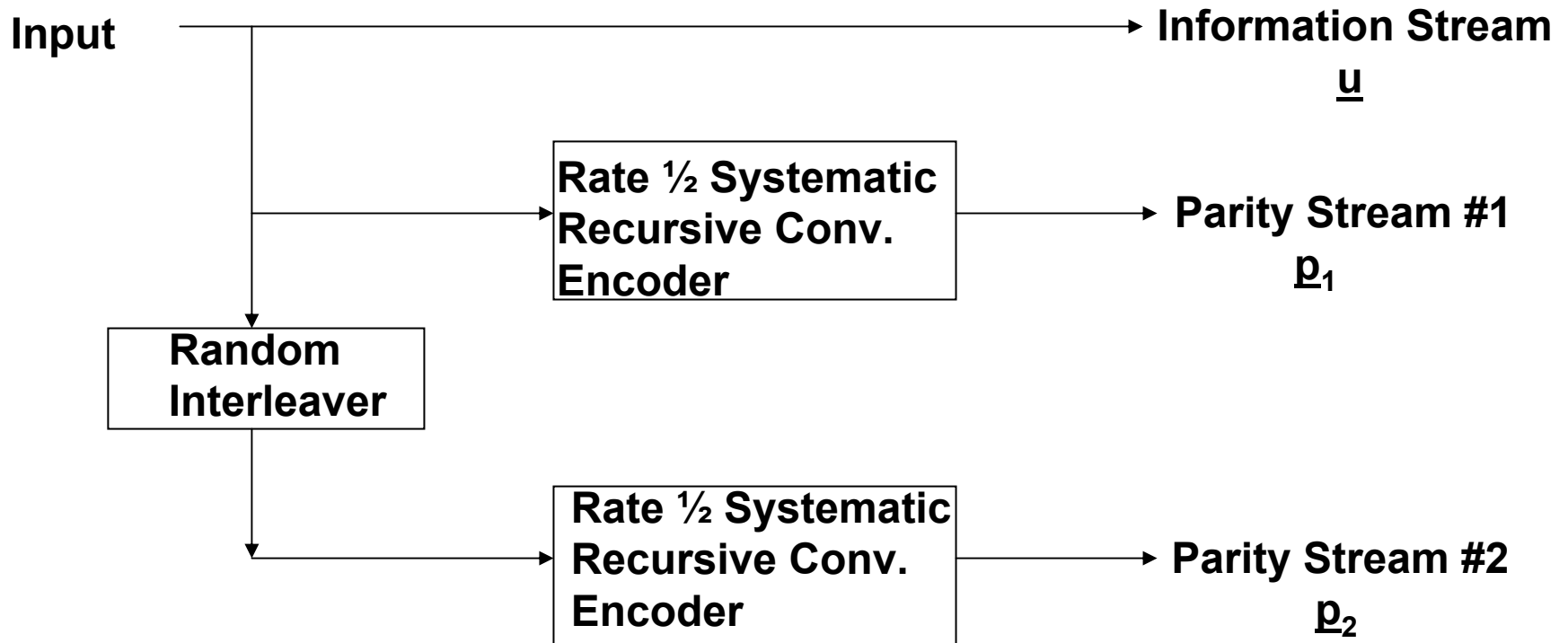


# SOFT-OUTPUT VITERBI DECODER

- **Hagenhauer modified the soft-input Viterbi decoder to give soft outputs. This means the decoder outputs the probability (or the logarithm of the probability) that each bit is a 0 or a 1.**
- **Bahl, Cocke, Jelinek and Raviv gave an algorithm (called the BCJR algorithm) that is both soft-input and soft-output. It is also called the MAP (maximum a posteriori probability) algorithm or the APP (aposteriori probability) algorithm.**
- **Some properties of the BCJR algorithm will be discussed later.**

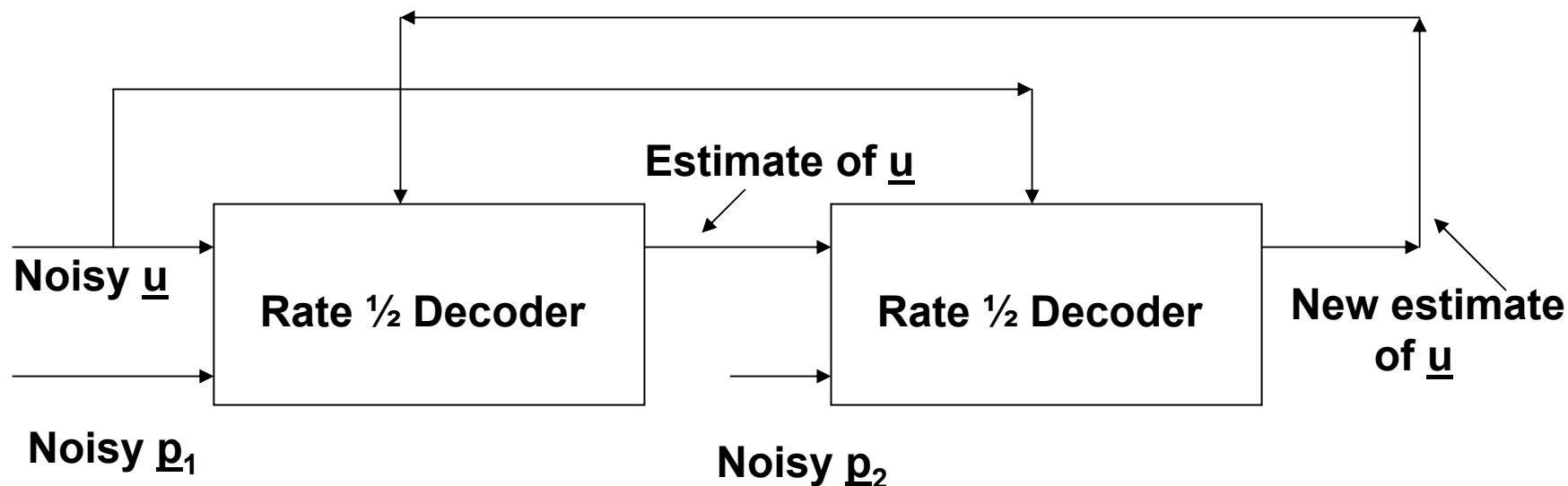
# DECODING TURBO CODES

- From before, the encoder is:



# DECODING TURBO CODES (WITH MANY DETAILS OMITTED)

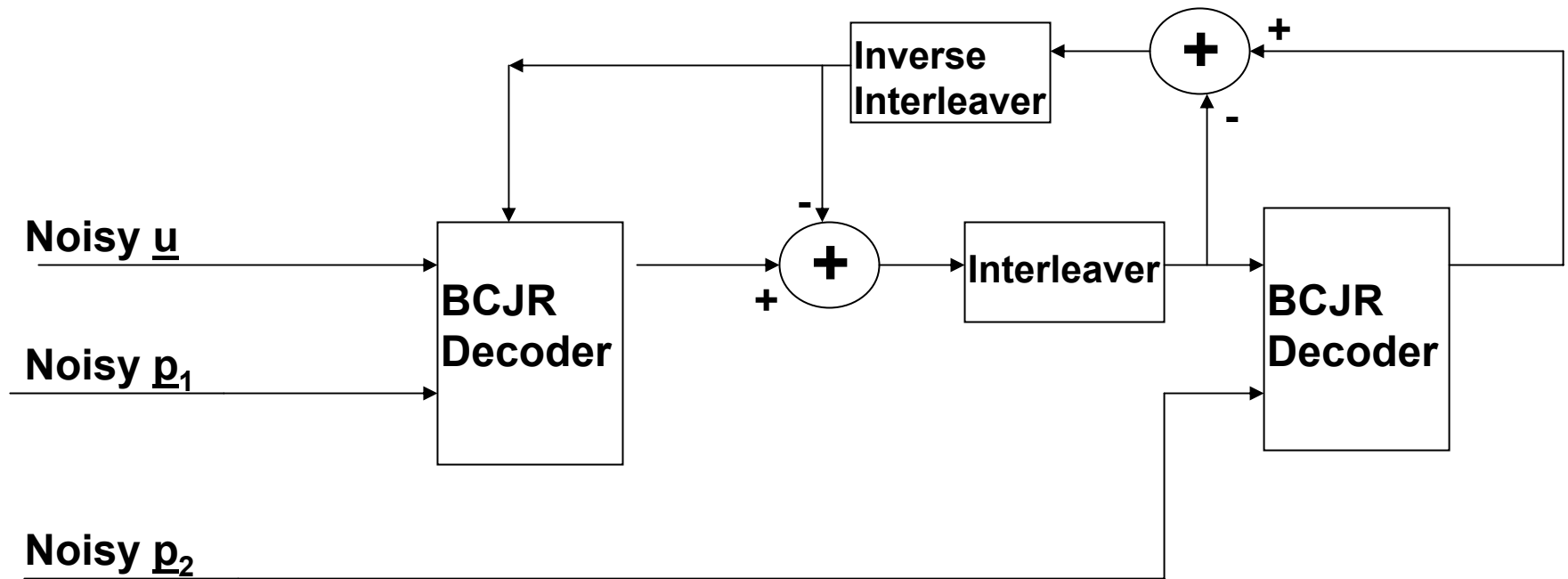
- Both decoders are soft-in and soft-out.
- The soft-output of one decoder becomes soft-input of other decoder.
- After several iterations, the final output is a hard decision on the information sequence  $\underline{u}$ .



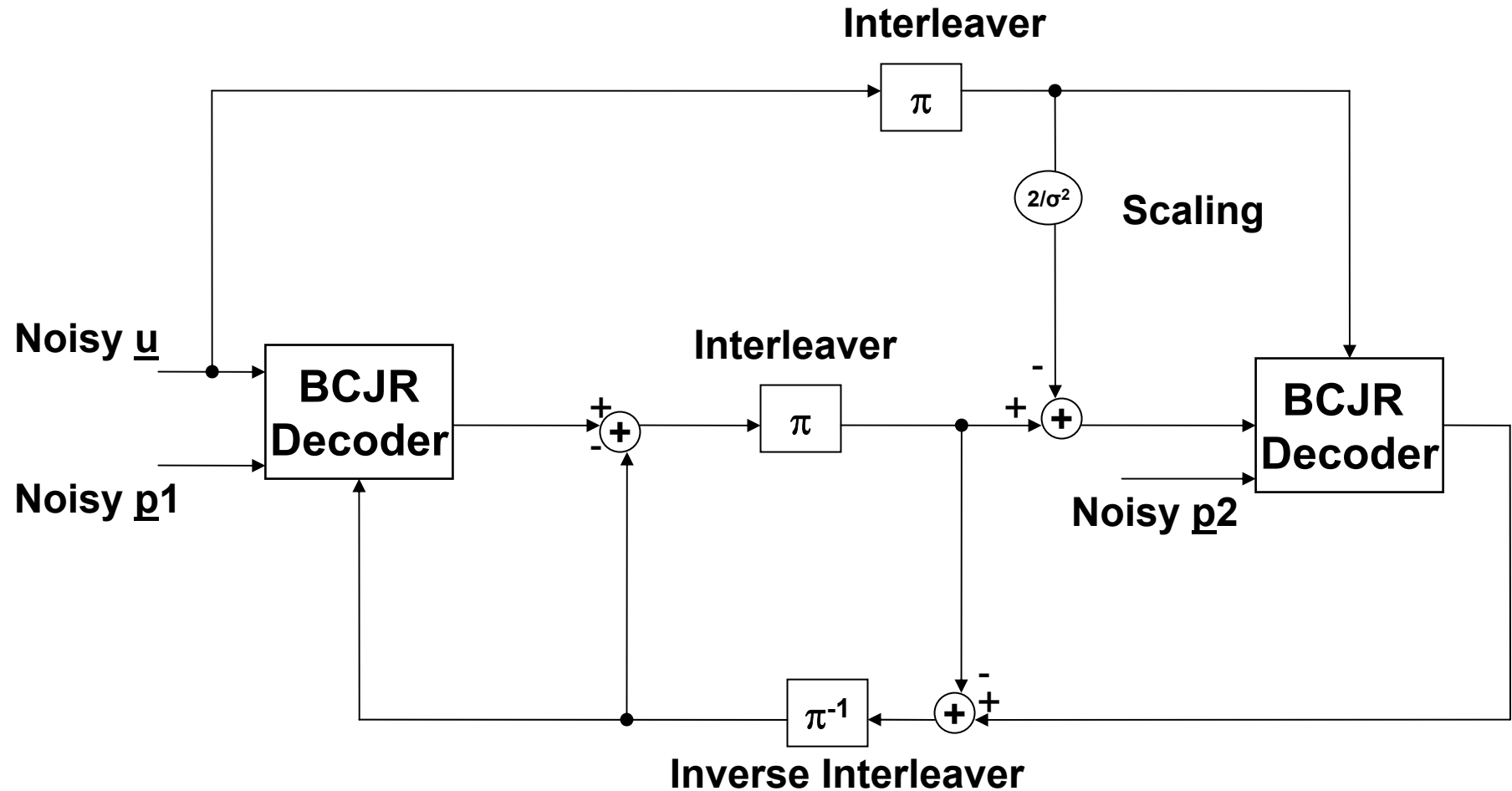
# SOME MISSING DETAILS

- One must reorder the information sequence coming out of each decoder to take into account the random-like permutation at the encoder.
- One must make sure that each decoder is acting only on new information from the other decoder and not just repeating the processing it already did.
- New information is called **extrinsic information** and old information is called **intrinsic information** .

# A SLIGHTLY MORE COMPLETE DECODER: PARALLEL CASE



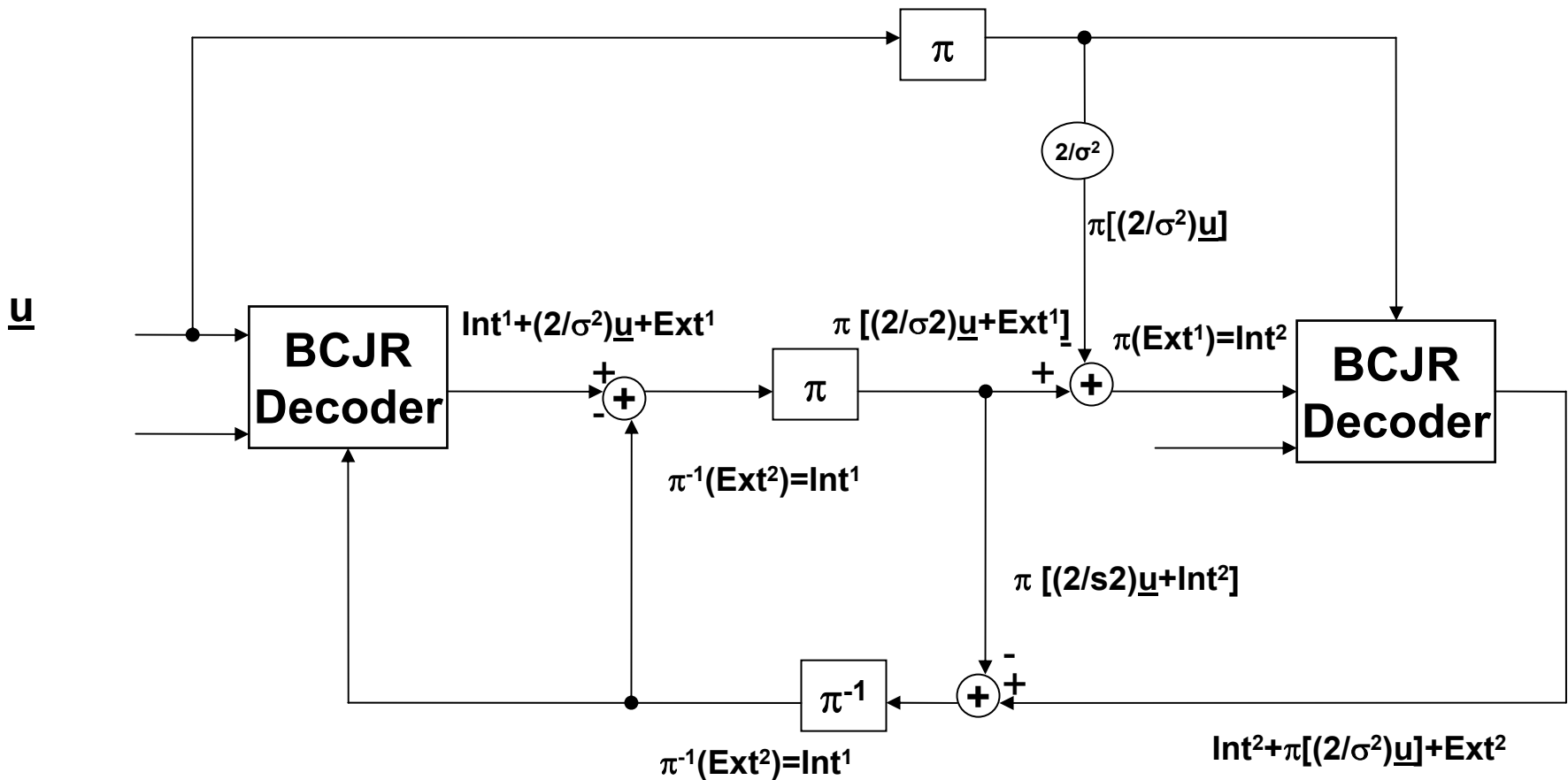
# MORE LIKE THE REAL THING: PARALLEL CASE



# SOME PROPERTIES OF THE BCJR ALGORITHM

- The BCJR decoder outputs the logarithm of the ratio of two (conditional) probabilities: the (conditional) probability that a bit is a 1 and the (conditional) probability that a bit is a 0.
- For an AWGN channel, the output can be written as the sum of three terms: (1) the **intrinsic information** (i.e., the logarithm of the ratio of the *apriori* probabilities), (2)  $(2/\sigma^2)$ (**channel value**), and (3) the **extrinsic information**.
- The extrinsic information from one decoder becomes the intrinsic information for the other decoder.

# MORE DETAILS ON THE ITERATIVE DECODER



# MORE PROPERTIES OF THE BCJR ALGORITHM

- Like the Viterbi algorithm, the BCJR algorithm is based upon a trellis description of the code.
- The algorithm traverses the trellis in two directions: forward and backward. Thus the alternate name, the **forward-backward** algorithm.
- It is very closely related to another algorithm called the Baum-Welch algorithm.

# MORE PROPERTIES OF THE BCJR ALGORITHM

- The **forward** algorithm recursively calculates the best estimate of the probability of the states at each depth, based upon **previous** channel outputs.
- The algorithm is recursive in that the estimates of the probabilities of the states at depth  $i$  are a function of the probability of the states at depth  $(i-1)$  and the received noisy samples corresponding to the bits on the branches that connect the states at depth  $(i-1)$  to the states at depth  $i$ .

# MORE PROPERTIES OF THE BCJR ALGORITHM

- The **backward** algorithm recursively calculates the best estimate of the probability of the states at each depth, based upon the **future** channel outputs.
- The algorithm is recursive in that the estimates of the probabilities of the states at depth  $i$  are a function of the probability of the states at depth  $(i+1)$  and the received noisy samples corresponding to the bits on the branches that connect the states at depth  $(i)$  to the states at depth  $(i+1)$ .

# MORE PROPERTIES OF THE BCJR ALGORITHM

- The estimate of the probability that  $u_i$  is a 0 or a 1, is then obtained by the forward algorithm's estimates of the probabilities of the **states** at depth  $(i-1)$ , the backward algorithm's estimates of the probabilities of the **states** at depth  $i$ , and the noisy received values of the bits corresponding to the branches that connect these states.
- Finally a hard decision can be made on the decoder's output using a slicer. That is a bit is a 1, if and only if the probability of it being a 1 is larger than  $\frac{1}{2}$ .

# MORE PROPERTIES OF THE BCJR ALGORITHM

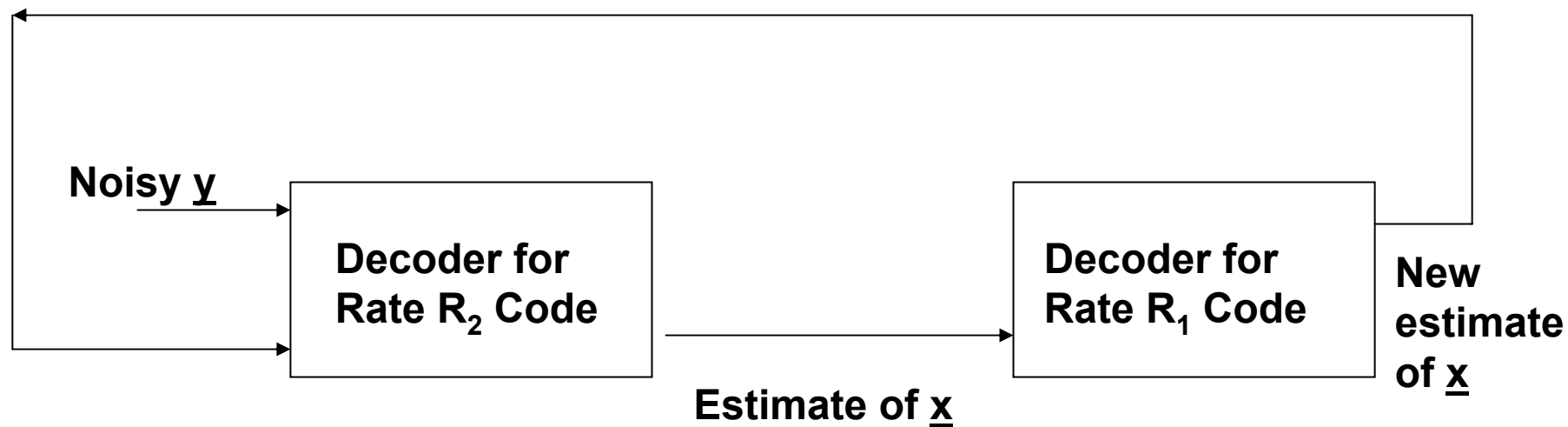
- **There are modifications to the BCJR algorithm that make the recursions similar to that of the Viterbi algorithm.**
- **Just as in the Viterbi algorithm, the performance is best when the beginning and ending state of the trellis is known (usually the all-zero state).**
- **For turbo codes, it is easy to force the first of the encoders to the all-zero state. However, because of the interleaver, it is not so easy to do this for the second encoder.**

# SERIAL TURBO CODE (CONCATENATED CODE)

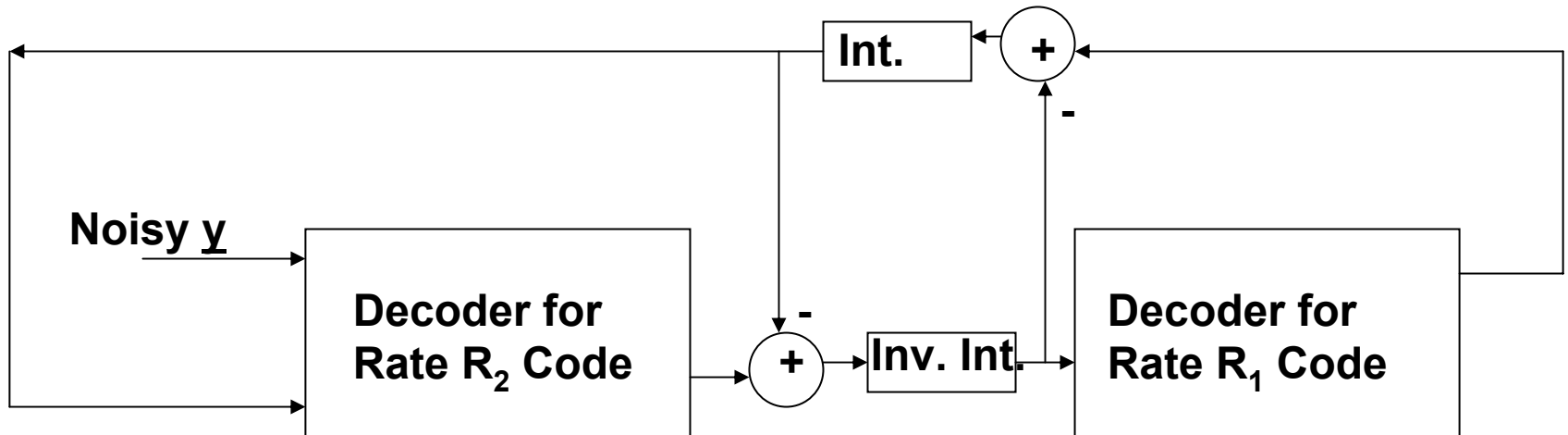


$$\text{Overall rate} = R_1 R_2$$

# SERIAL TURBO DECODER (WITH DETAILS OMITTED)



# A SLIGHTLY MORE COMPLETE DECODER: SERIAL CASE



**RANDOM AND PSEUDO-**  
**RANDOM BINARY**  
**SEQUENCES**

# RANDOM BINARY SEQUENCES

- A random binary sequence is a binary sequence produced by tossing a perfect coin. A typical sequence might be: H H H T H T T H ...
- There are two conventions for the binary values. {H,T} can be represented by {+1,-1} or by {0,1}.
- The probability of a head and a tail are assumed to be equal (i.e., 50-50). Furthermore, successive tosses are assumed to be **statistically independent**. That is, the next outcome does **not** depend on what already happened.

# RANDOM AND PSEUDO-RANDOM BINARY SEQUENCES

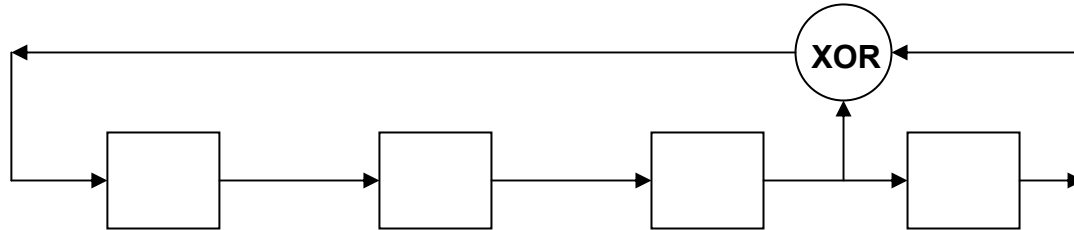
- **Pseudo-random binary sequences are binary sequences that have properties similar to random binary sequences.**
- **For example, a random binary sequence has the property that**
  - **Short runs of heads or tails are more likely than long runs. Specifically  $\frac{1}{2}$  the runs are of length 1,  $\frac{1}{4}$  of the runs are of length 2,  $\frac{1}{8}$  of the runs are of length 3, ...**
- **We would want a pseudo-random sequence to approximate this property.**

# RANDOM AND PSEUDO-RANDOM BINARY SEQUENCES

- However, there is **nothing random** about a pseudo-random sequence. In particular, an entire pseudo-random sequence of very long length may be totally predictable from very few past samples.
- There are ways to make predicting future values of the sequence difficult. Although these ways are useful in creating sequences used in cryptography, they have no relevance to CDMA.
- Pseudo-random sequences are often called pseudo-noise (or **PN**) sequences.

# LINEAR SHIFT REGISTER PN SEQUENCES

- The most common type of PN sequences are those generated by a linear feedback shift register:



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

The PN sequence can be taken as the **periodic** sequence at the output of the last stage:  
**000100110101111**

# LINEAR SHIFT REGISTER PN SEQUENCES

- Note that 000100110101111 has one more 1 than 0.
- Also note that the sequence has 8 runs of consecutive 0's or consecutive 1's.

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

Four of the 8 runs are of length 1, 2 of the 8 runs are of length 2, 1 of the 8 runs is of length 3 and 1 of the 8 runs is of length 4. Except for the run of length 4, the proportion of runs of a given length is that which is expected of random sequences.

# LINEAR SHIFT REGISTER PN SEQUENCES

- If the feedback shift register (FBSR) has its feedback taps positioned in a certain way, the feedback shift register is said to generate maximal length (i.e., ML) sequences.
- A ML sequence is periodic but its period is equal to  $(2^{(\text{\# of stages in FBSR})} - 1)$ .

# MAXIMAL LENGTH LINEAR SHIFT REGISTER SEQUENCES

No. of stages in FBSR	Length of Sequence
4	15
5	31
6	63
...	...
10	1,023
20	1,048,575
30	1,073,741,823

# LINEAR SHIFT REGISTER PN SEQUENCES

- There are many FBSR's of a given length that will generate a maximal length sequence.
- As a matter of fact, there are more than 24,000 shift registers of length 20 that will generate distinct maximal length sequences of period  $(2^{20} - 1)$  .
- If one does not know the tap connections of the FBSR, one can still predict the sequence if one has a short set of consecutive digits of the sequence.
- In particular, if one knows any  $2n$  consecutive bits of an ML sequence, of period  $2^n - 1$ , one can predict all the rest of the sequence. For example, if  $n=30$ , any 60 consecutive digits of the sequence will predict the entire sequence of length 1,073,741,823.

# ORTHOGONAL SEQUENCES

- A pair of binary sequences of even length is said to be **orthogonal** if they differ in exactly  $\frac{1}{2}$  of the positions. For example:

1 1 0 1 0 0

1 0 1 1 0 1

are orthogonal.

- A set of binary sequences are said to be an **orthogonal set** if every pair of sequences in the set are orthogonal.

# ORTHOGONAL SEQUENCES

- One set of orthogonal sequences are the Hadamard sequences. The construction goes as follows:

length 2            1 1  
                         1 0

length 4            1 1 | 1 1  
                         1 0 | 1 0  
                         

---

                         1 1 | 0 0  
                         1 0 | 0 1

# ORTHOGONAL SEQUENCES

length 8

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

length n

length 2n

$$\begin{array}{c|c}
 & \mathbf{A}_n \\
 \hline
 \mathbf{A}_n & \mathbf{A}_n \\
 \hline
 \mathbf{A}_n & \mathbf{B}_n \text{ where } \mathbf{B}_n = \text{complement } (\mathbf{A}_n)
 \end{array}$$

# ORTHOGONAL SEQUENCES

- Note that if two sequences are orthogonal, they will remain orthogonal even after some other common sequence is added (bit by bit, modulo 2) to both of them.

- For example,

1 1 0 0 1 1 0 0

1 0 0 1 1 0 0 1

are orthogonal so that

$$1\ 1\ 0\ 0\ 1\ 1\ 0\ 0 + \underline{0\ 0\ 1\ 1\ 0\ 1\ 0\ 0} = 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0$$

$$1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 + \underline{0\ 0\ 1\ 1\ 0\ 1\ 0\ 0} = 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1$$

are also orthogonal.

# **ORTHOGONAL SEQUENCES**

- **Thus, if one starts with a portion of a pseudo-noise sequence and one adds this sequence to a set of orthogonal sequences the result is a set of orthogonal sequences.**
- **This is what is done in CDMA to create orthogonal subcarriers.**