Frame Synchronization and Channel Coding

Chapter 8

Introduction

We will cover the topics of **frame synchronization** and channel coding.

It requires that the **signal has been timing and frequency corrected**.

Once **frame synchronization** has been **completed** we can fully **decode data** over our wireless link.

Once decoded,we can move on toward **channel coding**,

Where are we now?

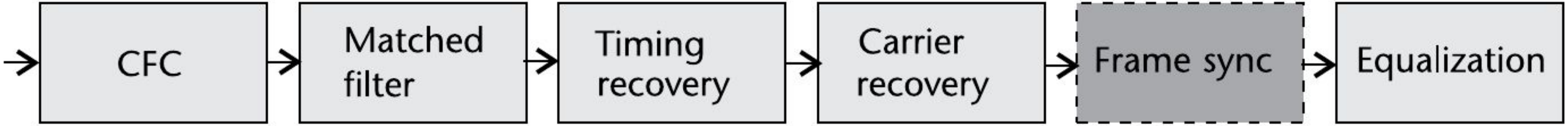


Figure 8.1 Receiver block diagram.

8.1 O Frame, Where Art Thou?

It is assumed that the available **samples represent single symbols** and are **corrected for timing, frequency, and phase offsets**.

The **start of a frame** will still be **unknown**, **we need to perform** an additional **correction** or estimation.

Mathematically, this is simply an **unknown delay in our signal** y:

$$u[n] = y[n - p]$$

Where $p \in \mathbb{Z}$. Once **we have an estimate** $\hat{p}$ **we can extract data** from the desired frame, demodulated to bits, and perform any additional channel decoding or source decode originally applied to the signal.

8.1 O Frame, Where Art Thou?

There are **various way to accomplish this estimation** but the implemented algorithm **we will use** is based on using **cross-correlation**.

Depending on the **receiver structure** and **waveform** it may be **possible** to perform **frame synchronization after demodulation**.

This **cannot** be used **if symbols** are **required** downstream **for an equalizer** or if the **preamble contains** configuration **parameters** for **downstream modulation**. (Like IEEE 802.11 or packet-based systems).

8.2 Frame Synchronization

The **common method** of **determining** the **start** of a given **frame** is with the **use** of **markers**, even in wired networking.

In the case of **wireless signals**, this problem becomes **more difficult**.

8.2 Frame Synchronization

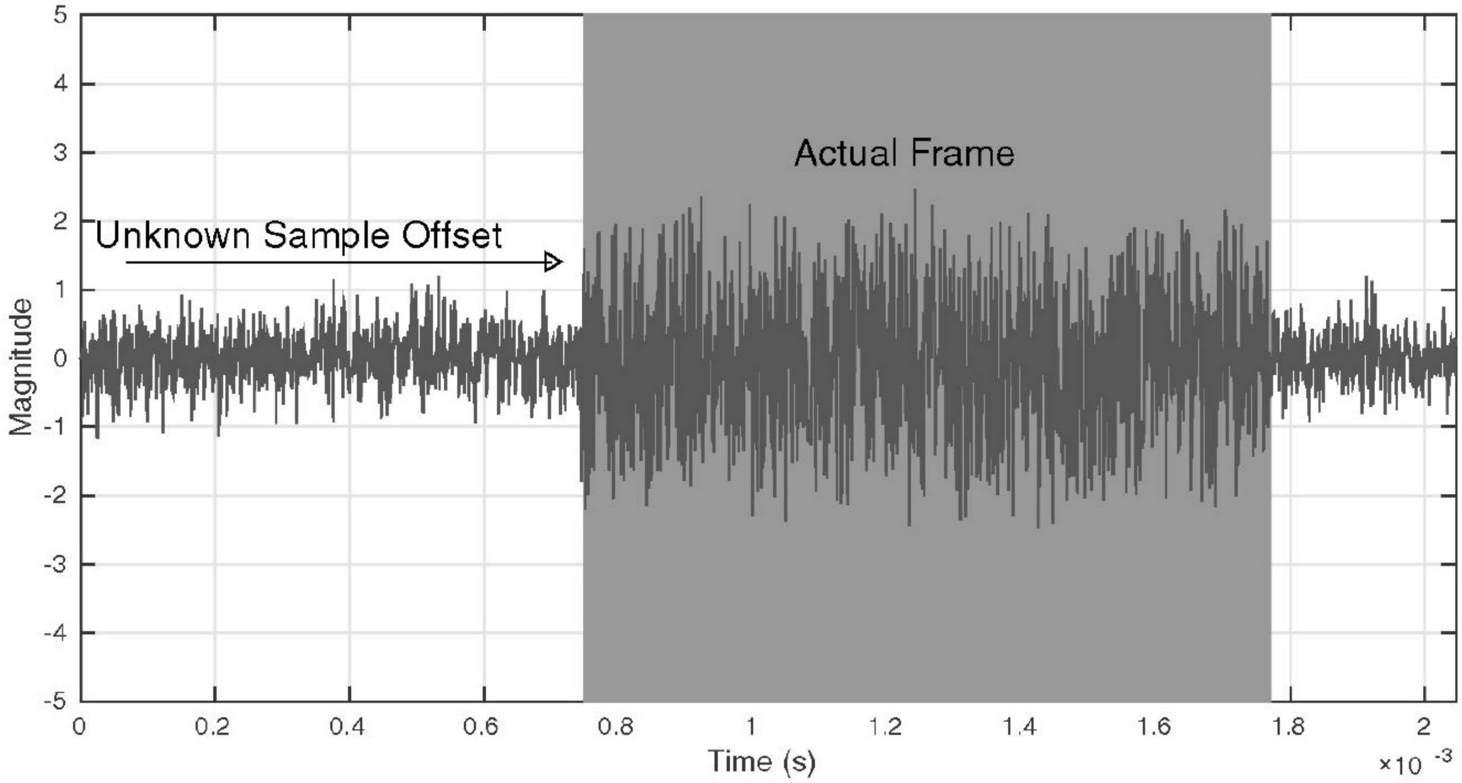


Figure 8.2 Example received frame in AWGN with an unknown sample offset.

8.2 Frame Synchronization

Due to the **high** degree of **noise** content in the signal, **specifically** designed **preamble sequences** are **appended** to frames before modulation.

Such **sequences** are **known** exactly **at the receiver** and have certain qualities that make frame estimation accurate.

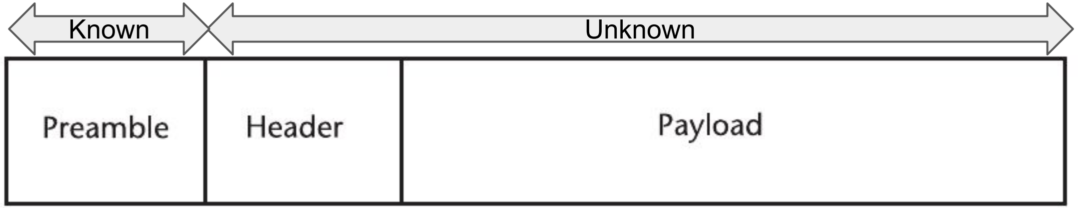


Figure 8.3 Common frame structure of wireless packet with preamble followed by header and payload data.

8.2 Frame Synchronization

Let's study a **technique** for **estimation** of the **start** of a **known sequence** starting at an **unknown sample in time**. (Before studying the common sequences used.)

Consider:

- A set of **N different binary sequences $\mathbf{b}_{\mathbf{n}}$** , where $n \in [1, \ldots, N]$, each of length L.
- An additional **binary sequence $\mathbf{d}$**.
- We want to **determine how similar $\mathbf{d}$** is to the existing **N sequences**.

8.2 Frame Synchronization

The use of a **cross correlation** would **provide** us the **appropriate estimate**, which we perform as

$$C_{d,b}(k) = \sum_{m} d^*(m) b_n(m+k)$$

When $d = b_n$ for a given n, **$C_{d,b}$ will be maximized** compared with the other $n - 1$ sequences, and **produce a peak at L_{th}** index at least.

We use this concept to build our frame start estimator.

8.2 Frame Synchronization -Barker Codes

- Utilized in **preambles** for **narrowband communications**.
- They have **unique autocorrelation** properties that have **minimal or ideal off-peak correlation**.
- For a sequence $a(i)$ the autocorrelation functions are **defined as**

$$c(k) = \sum_{i=1}^{N-k} a(i)a(i+k)$$

Such that

$$|c(v)| \leq 1, \quad 1 \leq v < N.$$

8.2 Frame Synchronization -Barker Codes

- Only **nine sequences** are **known** $N \in [1, 2, 3, 4, 5, 7, 11, 13]$

Table 8.1 Barker Codes from `comm.BarkerCode`

N	*Code*
2	−1, +1
3	−1, −1, +1
4	−1, −1, +1, −1
5	−1, −1, −1, +1, −1
7	−1, −1, −1, +1, +1, −1, +1
11	−1, −1, −1, +1, +1, +1, −1, +1, +1, −1, +1
13	−1, −1, −1, −1, −1, +1, +1, −1, −1, +1, −1, +1, −1

8.2 Frame Synchronization -Barker Codes

Let’s look at some MATLAB code on *barker_code.m* and *barker_code_variable_length.m* files.

8.2 Frame Synchronization -Barker Codes

Now let's try to see how we can **estimate the delay** in our signal $\hat{p}$

1. We have a **received signal r[n]** and a **Barker corde a[n]**
2. The received signal **r[n]** is of length L_r
3. The Barker corde **a[n]** is of length L_a
4. We will use **MATLAB's `xcorr`** function
5. MATLAB's `xcorr` function will **pad L_r-L_a zeros to a[n]** to perform the cross-correlation
6. The **cross correlation** will be of **size $2L_r$-1**
7. The **offset position** will be at

$$\hat{p} = \underset{k}{\operatorname{argmax}}\ C_{ra}(k) - L_r$$

8.2 Frame Synchronization -Barker Codes

Let’s look at some MATLAB code on *barkerBits13.m*

8.2 Frame Synchronization -Barker Codes

Some **notes** on the **cross-correlation estimation**

- For better **performance** the **FFT** can be used.
- The **correlation inflates** the **data** processed since the sequences must be of equal length.
- A **more efficient implementation** would be to **utilize a filter**.

$$y[n] = \sum_{i=0}^{N} b_i \, u[n-i].$$

Where **b_i** are the **filter taps** and **$u[n]$** is our **received signal** that contains the sequence of interest.

8.2 Frame Synchronization -Barker Codes

Some **notes** on the **cross-correlation estimation**

- For better **performance** the **FFT** can be used.
- The **correlation inflates** the **data** processed since the sequences must be of equal length.
- A **more efficient implementation** would be to **utilize a filter**.

Replace bi with the sequence of interest, but in reverse order.

Hardware efficient!

$$y[n] = \sum_{i=0}^{N} b_i \, u[n-i].$$

Where b_i are the **filter taps** and **u[n]** is our **received signal** that contains the sequence of interest.

8.2.1 Signal Detection

We can define a minimum **received power** or **power sensitivity**.

Sensitivity will be **based** on some **source waveform** and **cannot be generalized** in most cases.

Sensitivity should **never** be **given on its own**, **unless** given with respect to **some standard transmission**.

Must have **some knowledge** or reference to the **source signal**.

In the IEEE 802.11ac standard it is the minimum received signal power to maintain a packet error rate of 10%, for a give modulation and coding scheme

8.2.1 Signal Detection -Hypothesis Testing Framework

Let's define an **hypothesis testing framework** for **detecting our signal**.

A simple **binary hypothesis test**:

$$\mathcal{H}_0 : \text{no signals},$$
$$\mathcal{H}_1 : \text{signals exist},$$

8.2.1 Signal Detection -Hypothesis Testing Framework

Let’s define an **hypothesis testing framework** for **detecting our signal**.

A simple **binary hypothesis test**:

$$\mathcal{H}_0 : r[n] = n[n],$$
$$\mathcal{H}_1 : r[n] = x[n] + n[n],$$

where **r[n]** is the **received signal**, **n[k]** is the **noise in the RF environment**, and **x[n]** is the **signal we are trying to detect**.

8.2.1 Signal Detection -Hypothesis Testing Framework

Let's define an **hypothesis testing framework** for **detecting our signal**.

To decide between $\mathcal{H}_0$ or $\mathcal{H}_1$ we create a decision rule.

```
if r in Γ₁:
   ℋ = ℋ₁
else if in Γ₁ᶜ:
   ℋ = ℋ₀
```

In the context of packet detection, **thresholding** is actually the **implementation of a decision rule**.

8.2.1 Signal Detection -Hypothesis Testing Framework

Sensing errors are inevitable due to:

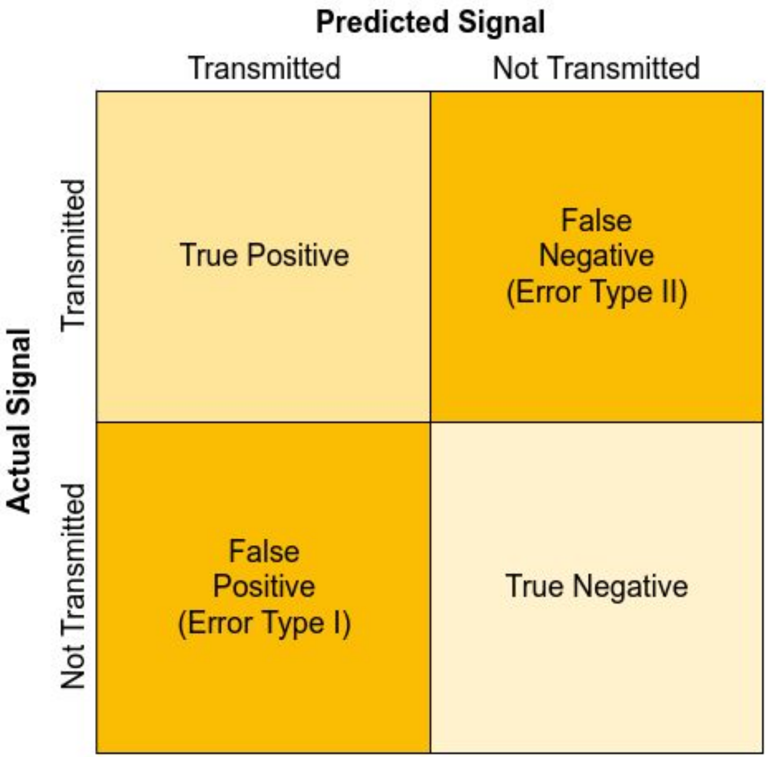
- **additive noise**,
- limited **observations**,
- **randomness** of the observed data

Which gives rise to **two types of error**:

- **Error Type I or False Alarm:** there are actually **no signals** in the channel, but the testing **detects** an **occupied channel**.
- **Error Type II or Missed Detection:** there exist **signals in the channel**, but the testing **detects** only a **vacant channel**.

8.2.1 Signal Detection -Hypothesis Testing Framework

Confusion Matrix



8.2.1 Signal Detection -Hypothesis Testing Framework

The **performance** of a **detector** can be **characterized by two parameters**

Probability of **false alarm** (PF)

$$P_F = P\{\text{Decide } \mathcal{H}_1 | \mathcal{H}_0\}$$

Type I Error

Probability of **missed detection** (PM) Type II Error

$$P_M = P\{\text{Decide } \mathcal{H}_0 | \mathcal{H}_1\}$$

Type II Error

8.2.1 Signal Detection -Hypothesis Testing Framework

Another frequently used parameter is the **probability of detection** (PD)

$$P_D = 1 - P_M = P\{\text{Decide } \mathcal{H}_1 | \mathcal{H}_1\}$$

- Characterizes the **detector's ability to identify** the **primary signals** in the **channel**.
- PD is usually referred to as the **power of the detector**.

8.2.1 Signal Detection -Hypothesis Testing Framework

- Ideally we would like the **probability** of **false alarm** to be as **low as possible**, and at the same time, their **probability of detection** as **high as possible**.
- In a **real-world** situation, this is **not achievable**, because these two **parameters** are **constraining each other**. (See the Receiver Operating Characteristic curve in the next slide).
- The **detection problem** is a **trade-off**, which depends on **how** the **Type I** and **Type II errors** should be **balanced**.

8.2.1 Signal Detection -Hypothesis Testing Framework

Receiver Operating Characteristic (ROC)

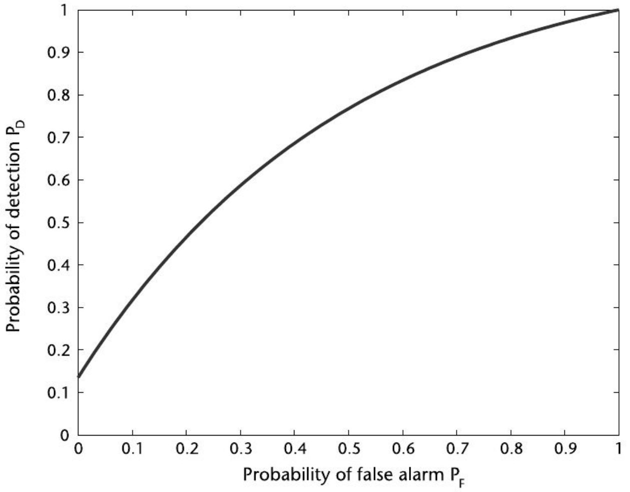
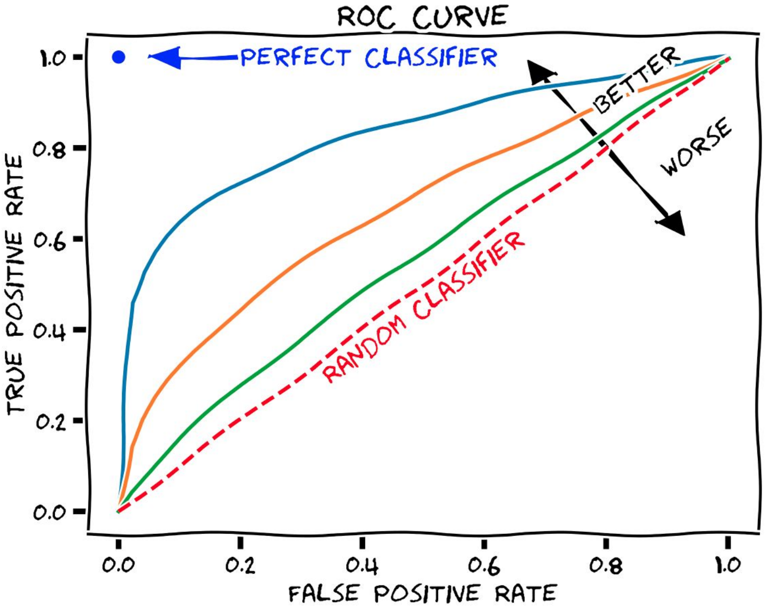


Figure 8.6 A typical receiver operating characteristic, where the x-axis is the probability of false alarm (P_F), and the y-axis is the probability of detection (P_D).



8.2.1 Signal Detection -Hypothesis Testing Framework

In conclusion:

The **detection becomes a thresholding problem** for our correlator.

The **objective** becomes **determining a reference** or criteria **for validating a peak**, which **can be** radically **different over time**.

Operate regardless of **the input** scaling.

8.2.1 Signal Detection -Hypothesis Testing Framework

Normalizing

A common technique to aid with this thresholding process is to **self-normalize** the **received signal** between $\in [0, 1]$.

A simple way to accomplish this operation is to **scale our cross-correlation** metric **$C_{y,x}$ by the mean energy of the input signal $\mathbf{x}$** by **implementing a moving average filter**.

The **moving averaging** would be **modeled as**:

$$u_{ma}[n] = \sum_{i=0}^{N} u[n-i]$$

Where N is the length of the preamble or sequence of interest

8.2.1 Signal Detection -Hypothesis Testing Framework

Finally we can define our detector as:

$$\mathcal{H}_0 : \frac{y[n]}{u_{ma}[n]} < T \text{ no signals,}$$
$$\mathcal{H}_1 : \frac{y[n]}{u_{ma}[n]} \geq T \text{ signals exist,}$$

Where:

- T is our threshold value.
- $y[n]$ our cross correlation
- $u_{ma}[n]$ our moving average

8.2.1 Signal Detection

Let’s look at some MATLAB code on *findSignalStartTemplate.m*

8.2.2 Alternative Sequences

Besides Barker sequences, there are **other sequences** that have similar properties of **minimal cross correlation except at specific instances**.

8.2.2 Alternative Sequences -Zadoff-Chu

- Used for **LTE synchronization** and **channel sounding** operations
- **Constant amplitude**
- **Zero circular autocorrelation**
- **Low correlation** between **different sequences**
- **Limited correlation** between **themselves** (useful in a **multiaccess environment** where many users can transmit signals.)

8.2.2 Alternative Sequences -Zadoff-Chu

- The sequence numbers are **generated as**

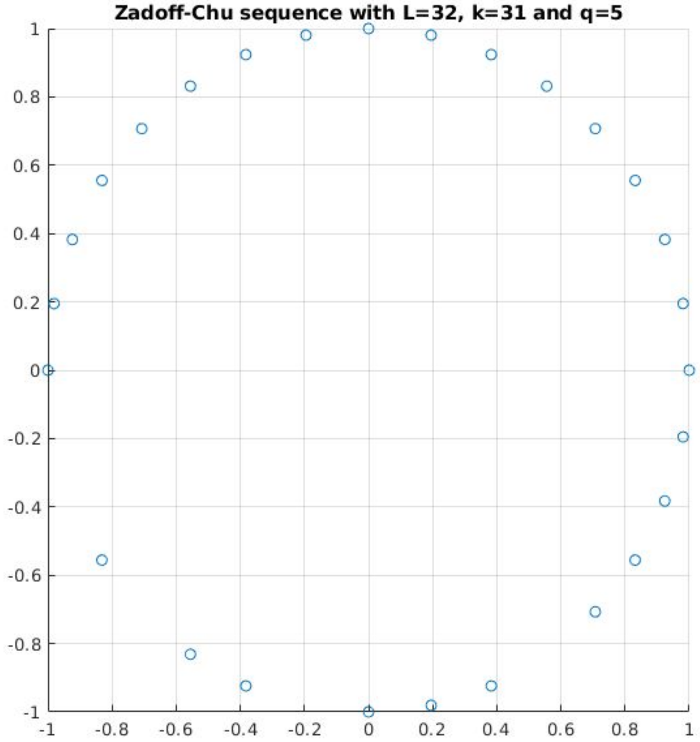
$$s_n = exp\left(-j\frac{\pi\, k\, n\, (n+1+2q)}{L} \right)$$

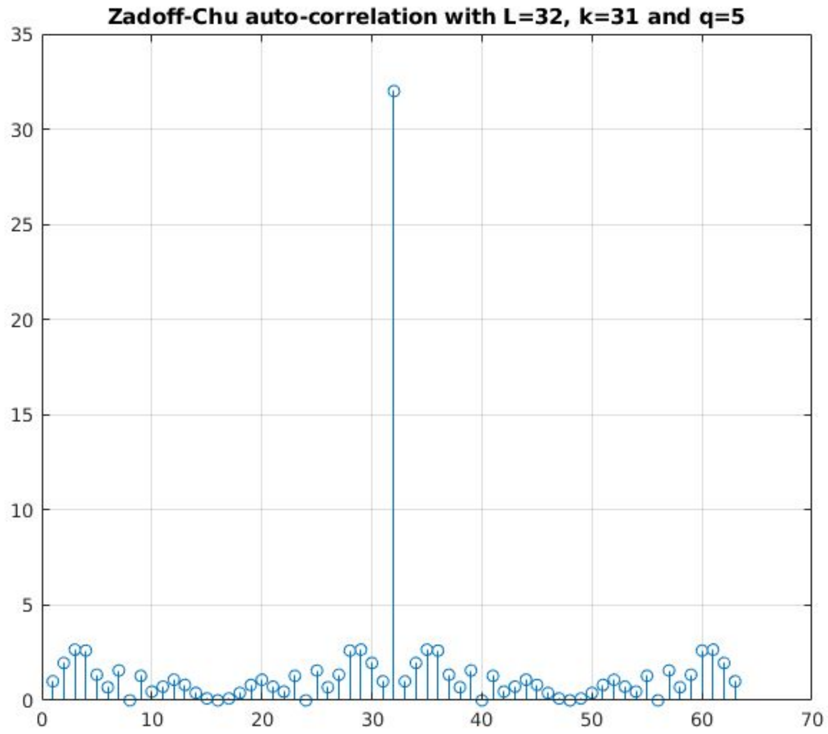
Where:

- L is the sequence length,
- n is the sequence index,
- q an integer,
- k is a coprime number with L

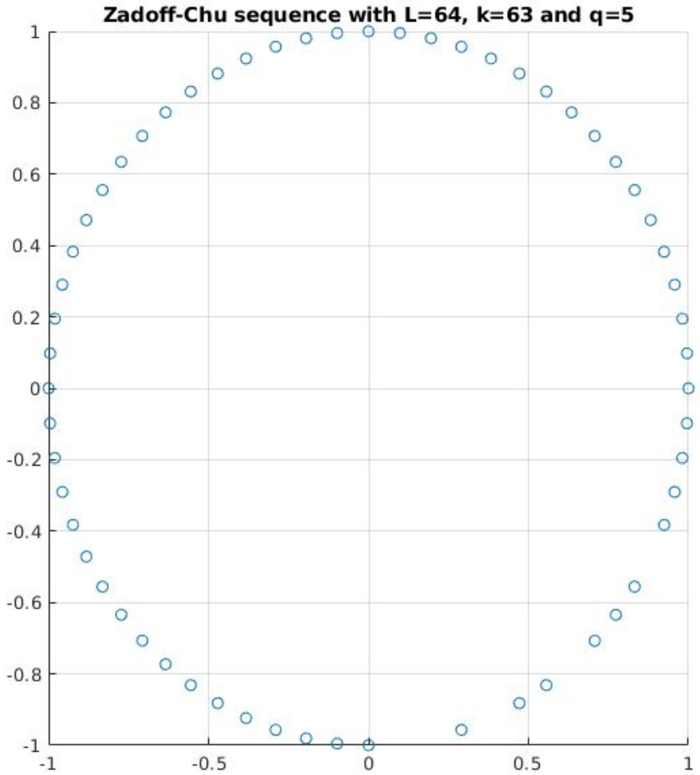
Check function called *Zadoff_Chu in* chapter 8

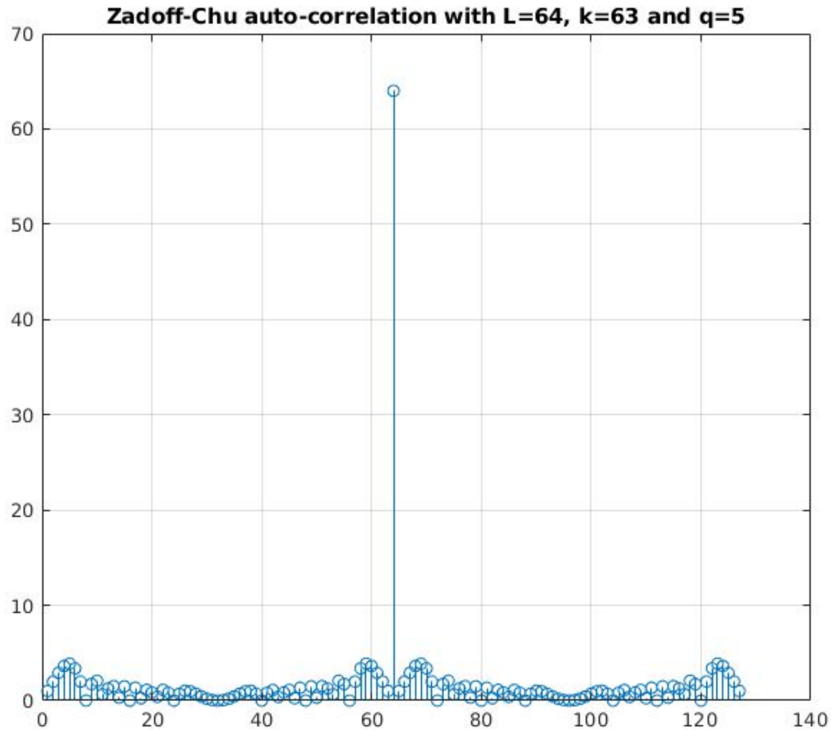
8.2.2 Alternative Sequences -Zadoff-Chu





8.2.2 Alternative Sequences -Zadoff-Chu





8.2.2 Alternative Sequences -Golay complementary sequences

- Used for **channel estimation and synchronization** within the preamble of **IEEE 802.11ad packets**.
- They are sequences of **bipolar symbols with minimal autocorrelation** properties.
- These sequences come in **complementary pairs** that are typically denoted as **Ga_n** and **Gb_n**, where n is the sequence length.
- **IEEE 802.11ad** uses pairs **Ga_{32}**, **Ga_{64}**, and **Gb_{64}**
- G_a and G_b **autocorrelation** can be performed in **parallel** in **hardware**.
- Depending of a **packet type**, a **correlator bank** can be used to identify that specific structure, conditioning the processing receiver to a **specific decoder path**.

8.2.2 Alternative Sequences -Golay complementary sequences

Some **mathematical properties**:

$$R_{Ga}[n] + RGb[n] = 2N$$

$$R_{Ga}[0] + RGb[0] = 0$$

Where R is the autocorrelation of $\mathrm{G_a}$ and $\mathrm{G_b}$ and N is the number of samples.

8.2.2 Alternative Sequences -Golay complementary sequences

Some **mathematical properties**:

A complementary pair G_a, G_b may be encoded as polynomials

$$G_a[z] = a[0] + a[1]z + \ldots + a[N-1]z^{N-1}$$

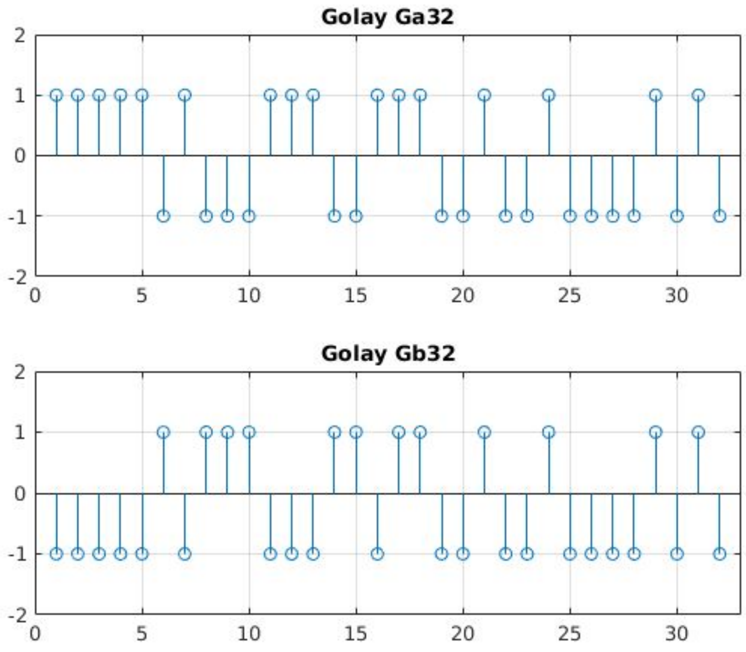
and similarly for $G_b(z)$.

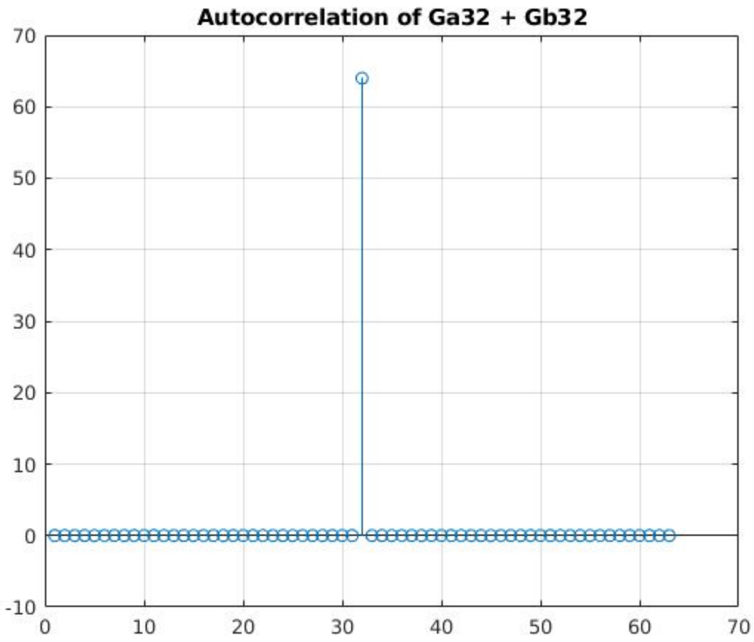
The complementarity property of the sequences is equivalent to the condition

$$|A[z]|^2 + |B[z]|^2 = 2N$$

Check Matlab function called *wlanGolaySequence*

8.2.2 Alternative Sequences -Golay complementary sequences





8.3 Putting the Pieces Together

We have **all** the necessary **pieces to build a wireless receiver** that can **handle carrier offsets**, **timing mismatches**, and **random packet delay**.

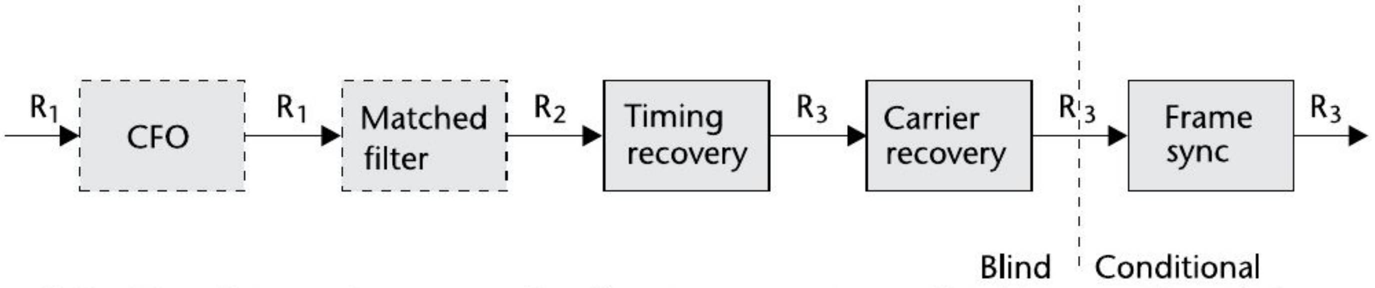


Figure 8.9 Complete receiver processing flow to recover transmitted frames. The relative sample rates are defined by R_n.

8.4 Channel Coding

The primary **purpose of channel coding** is to **increase efficiency**, the better the system can correct the inevitable errors introduced by wireless transmission the more efficient it will be. Specific benefits include:

- Reduced **error rates** and retransmission
- Increased **capacity**
- Increased **throughput**
- Reduced **power usage**

8.4.1 Repetition Coding

One of key building blocks of any communication system is the **forward error correction** (FEC), where **redundant data is added** to the transmitted stream to make it more robust to channel errors.

Each transmitted bit is **repeated R** times.

```
>> R = 4;
>> u = [1 1 0 1 0 0 1];
>> t = repmat(u,R,1)

t =

     1     1     0     1     0     0     1
     1     1     0     1     0     0     1
     1     1     0     1     0     0     1
     1     1     0     1     0     0     1
```

MATLAB has the ***repmat*** function, which takes an input vector u and a repetition factor R.

8.4.2 Interleaving

A **repetition code** is **not robust when** a **large quantity** of **data** is **corrupted** in **contiguous blocks**.

Example: Suppose we have this data

data = 101101

By using a repetition code of 4, we expect

111100001111111100001111

But we get

111100 − − − − − − − − −1100001111

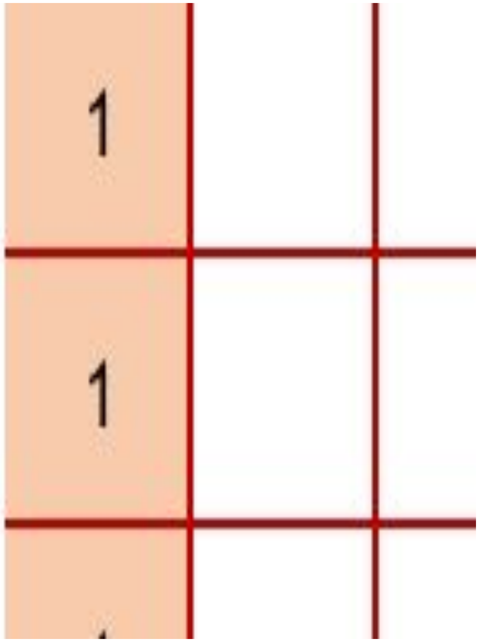
8.4.2 Interleaving

Interleaving is an approach where **binary data is reordered** such that the **correlation** existing **between the individual bits** within a specific sequence is significantly **reduced**.

Since errors usually occur across a consecutive series of bits, interleaving a bit sequence prior to transmission and deinterleaving the intercepted sequence at the receiver **allows** for the **dispersion of bit errors across the entire sequence**, thus minimizing its impact on the transmitted message.

A simple interleaver will **mix** up the **repeated bits** to **make** the **redundancy** in the data even **more robust to error**. It reorders the duplicated bits among each other to ensure that **at least one redundant copy** of each **will arrive** even if a series of bits are lost.

8.4.2.1 Block Interleaving



8.4.2.1 Convolutional Interleaving

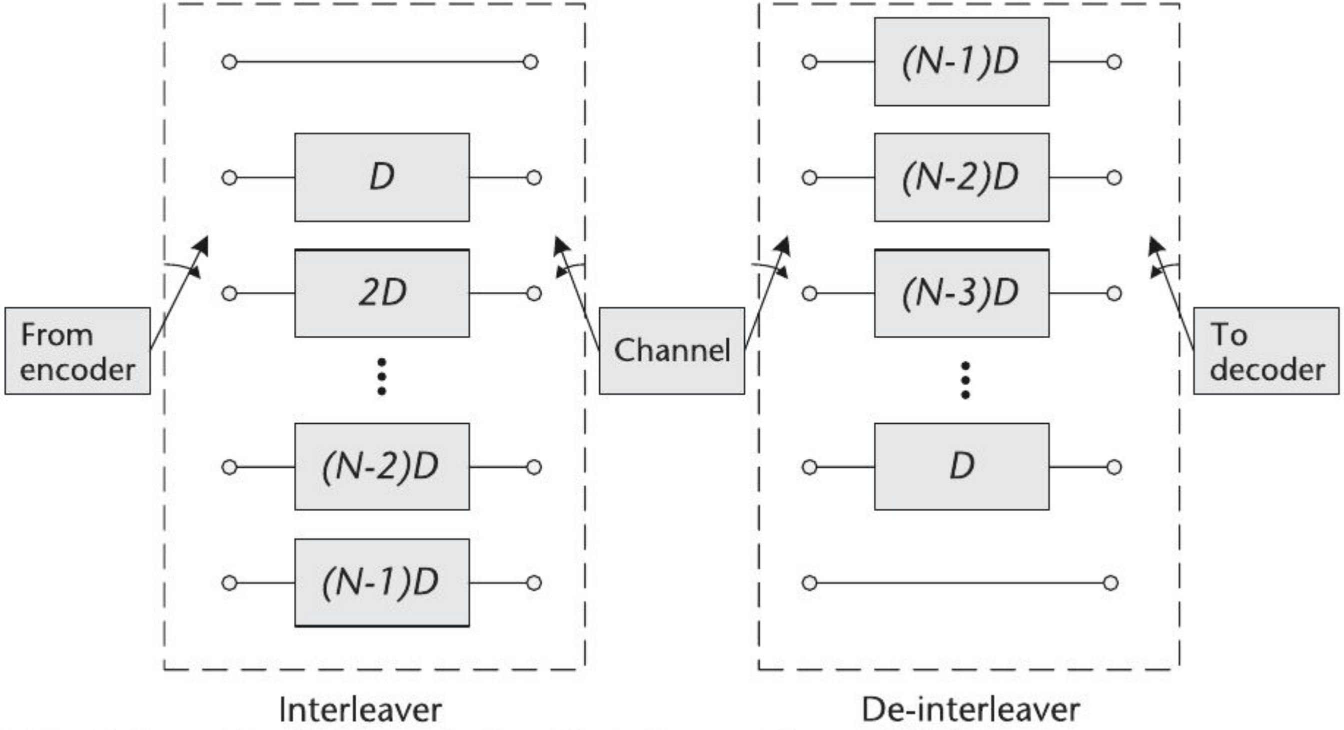
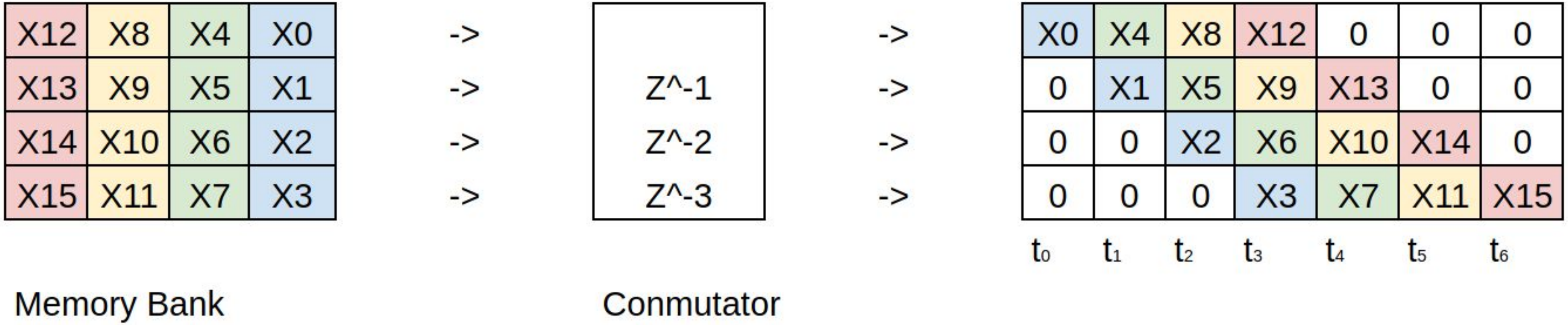


Figure 8.11 Schematic of a convolutional interleaver. (From [13].)

8.4.2.1 Convolutional Interleaving



8.4.3 Encoding

Besides interleaving multiple copies of data, we can instead **encode** the **data** into **alternative sequences** that **introduce redundancy**.

Many encoding schemes can **introduce redundancy without increases in data**.

For example, in the case of repetitive coding that duplicates every bit with R = 2, this number is usually inverted in FEC discussions as a rate of 1/2 , a **convolutional encoding** scheme can introduce **rates closer to 1**.

Channel encoding is a **mathematically complex** area in information theory.

8.4.3 Encoding

Encoders can typically be categorized into **two basic types**:

Block encoders:

- They work on **specific predefined groups** or blocks of bits

Convolutional type encoders:

- They work on **streams of data** of indeterminate size but can be made to work on **blocks of data if necessary**.

8.4.3 Block Code -Reed-Solomon (RS)

- Linear-block-code **developed** in the **1960s**.
- RS codes work by **inserting symbols** into a given frame or block of data, which are then **used to correct symbol errors** that occur.
- If we define **M** as the length of a given frame, sometimes called the message **length**, and define **E** as the **encoded frame** then we can **correct up to** $\lfloor \frac{E-M}{2} \rfloor$ messages.
- The symbols you can encode with a RS can be **integers between** $[0, 2^N - 1]$ where **N** is the **exponent** of our finite **Galois field** $GF(2^N)$

8.4.3 Block Code -Reed-Solomon (RS)

```
K>> m = 3;            % Number of bits per symbol
K>> n = 2^m - 1;      % Codeword length
K>> k = 3;            % Message length
K>> msg = gf([2 7 3; 4 0 6],m)

msg = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)

Array elements =

   2   7   3
   4   0   6

K>> code = rsenc(msg,n,k)

code = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)

Array elements =

   2   7   3   3   6   7   6
   4   0   6   4   2   2   0
```

MATLAB example of a Reed-Solomon encoder

8.4.3 Block Code -Bose Chaudhuri Hocquenghem (BCH)

- Relies on the concept of **Galois fields**.
- Better at correcting **errors that do not occur in groups**.
- **Correct more errors** than RS for the same amount of parity bits.
- Require **more computational power** to decode than RS.

8.4.3 Block Code -Bose Chaudhuri Hocquenghem (BCH)

```
>> M = 4;
>> n = 2^M-1;    % Codeword length
>> k = 5;        % Message length
>> nwords = 1; % Number of words to encode
>>
>> msgTx = gf(randi([0 1],nwords,k))

msgTx = GF(2) array.

Array elements =

   0   1   1   0   0

>> enc = bchenc(msgTx,n,k)

enc = GF(2) array.

Array elements =

   0   1   1   0   0   1   0   0   0   1   1   1   1   0   1
```

MATLAB example of a BCH encoder

8.4.3 Block Code -Low Density Parity Check Codes (LDPC)

- **Complex** in hardware.
- Can **approach** the theoretical **Shannon limit** for certain redundancy rates unlike RS and BCH.
- When utilizing LDPC the implementor must **select a parity matrix** which the **encoder** and **decoder** will utilize, and the characteristics of **this matrix** will **determine performance** of the code.
- You **need to encode a significant amount of bits** compared to the other codes to utilize LPDC efficiently in many cases.

8.4.3 Convolutional Code

- **Redundancy** is **introduced** by the **succession of information** passed through the encoder/decoder.
- Creates **dependency** on **consecutive symbols or bits**.

8.4.3 Convolutional Code

- Consider an encoding scheme with R = 2 with a recursive encoder

$$y_{n,1} = (x_n + x_{n-2} + x_{n-3}) + x_{n-1} + x_{n-3}$$
$$y_{n,2} = x_{n-2} + x_{n-3}$$

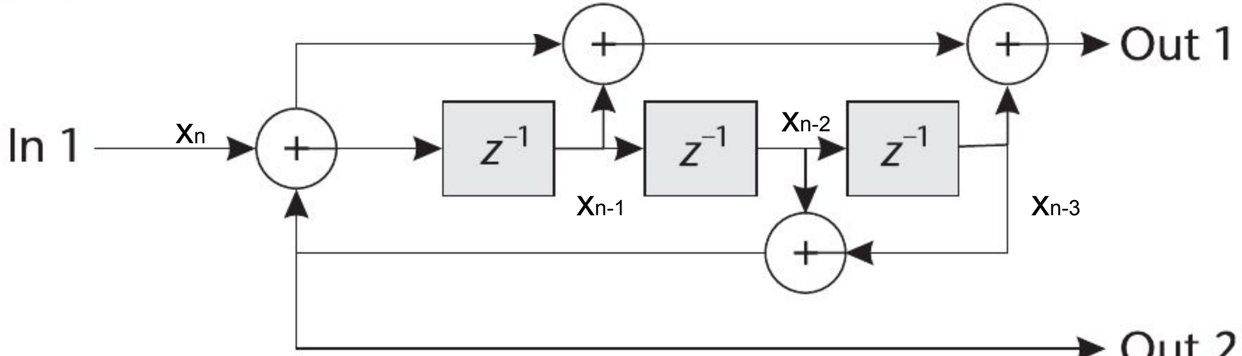


Figure 8.12 Example $R = 2$ convolutional encoder utilized in 3GPP LTE.

8.4.3 Convolutional Code -Viterbi Algorithm

The concept of the Viterbi/trellis decoder is to trace back through previous decisions made and utilize them to best determine the most likely current bit or sample.

Convolutional Codes: https://www.youtube.com/watch?v=kRIfpmiMCpU

Viterbi Algorithm: https://www.youtube.com/watch?v=dKIf6mQUfnY

8.4.3 Convolutional Code -Viterbi Algorithm

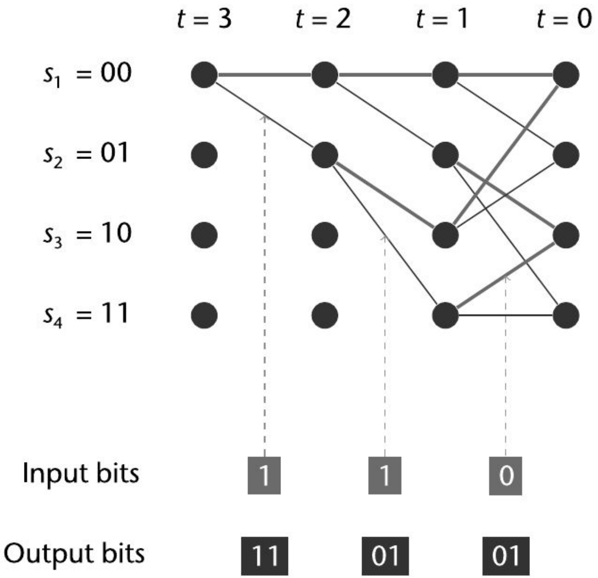


Figure 8.13 Viterbi/trellis decoder lattice diagram.

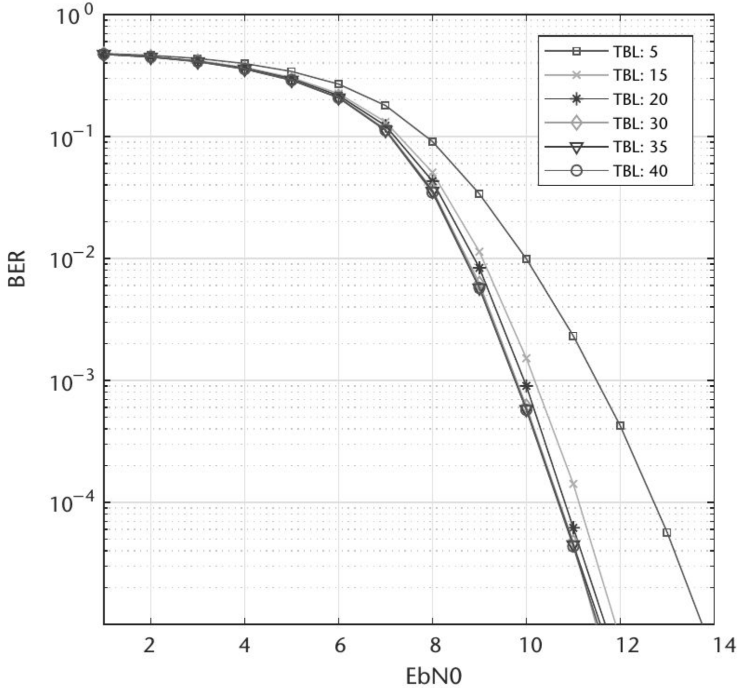


Figure 8.14 BER results of Viterbi decoder for 16-QAM with increasing traceback length.

8.4.3 Convolutional Code -Turbo Codes

- Introduced in the early **90's**.
- Heavily utilized by **third** and **fourth generation cellular standards** as their primary FEC scheme.
- Can **operate near the Shannon limit** for performance but are **less computationally intensive than LDPC with less correction performance**.
- Turbo inherently utilizes the **Viterbi algorithm** internally for decoding with some additional **interleaving**, as well as using a set of **decoders**, and performs **likelihood estimation** between them.
- Very **powerful coding technique** as long as **you have the resources** on your hardware to implement the decoder at the necessary speeds.

8.4.3 Encoding

Modern standards like LTE and IEEE 802.11, will utilize adaptive **Modulation and Coding Schemes (MCSs)**.

- Reduce coding redundancy
- Increase modulation order

8.4.3 Encoding

802.11ac - VHT MCS, SNR and RSSI

VHT MCS	Modulation	Coding	20MHz				40MHz				80MHz				160MHz			
			Data Rate		Min. SNR	RSSI	Data Rate		Min. SNR	RSSI	Data Rate		Min. SNR	RSSI	Data Rate		Min. SNR	RSSI
			800ns	400ns			800ns	400ns			800ns	400ns			800ns	400ns		
1 Spatial Stream																		
0	BPSK	1/2	6.5	7.2	2	-82	13.5	15	5	-79	29.3	32.5	8	-76	58.5	65	11	-73
1	QPSK	1/2	13	14.4	5	-79	27	30	8	-76	58.5	65	11	-73	117	130	14	-70
2	QPSK	3/4	19.5	21.7	9	-77	40.5	45	12	-74	87.8	97.5	15	-71	175.5	195	18	-68
3	16-QAM	1/2	26	28.9	11	-74	54	60	14	-71	117	130	17	-68	234	260	20	-65
4	16-QAM	3/4	39	43.3	15	-70	81	90	18	-67	175.5	195	21	-64	351	390	24	-61
5	64-QAM	2/3	52	57.8	18	-66	108	120	21	-63	234	260	24	-60	468	520	27	-57
6	64-QAM	3/4	58.5	65	20	-65	121.5	135	23	-62	263.3	292.5	26	-59	526.5	585	29	-56
7	64-QAM	5/6	65	72.2	25	-64	135	150	28	-61	292.5	325	31	-58	585	650	34	-55
8	256-QAM	3/4	78	86.7	29	-59	162	180	32	-56	351	390	35	-53	702	780	38	-50
9	256-QAM	5/6			31	-57	180	200	34	-54	390	433.3	37	-51	780	866.7	40	-48

8.4.4 BER Calculator

BER is a commonly used **metric** for the **evaluation and comparison of digital communication systems**.

The **ratio** of **bit errors to the total number of bits received** can provide us with an approximate BER calculation.