

MIT 6.02 DRAFT Lecture Notes
Spring 2012 (Last update: March 17, 2012)
Comments, questions or bug reports?
Please contact {verghese, hari} at mit.edu

CHAPTER 11

LTI Models and Convolution

This chapter will help us understand what *else* besides noise (which we studied in Chapter 9) perturbs or distorts a signal transmitted over a communication channel, such as a voltage waveform on a wire, a radio wave through the atmosphere, or a pressure wave in an acoustic medium. The most significant feature of the distortion introduced by a channel is that the output of the channel typically does not instantaneously respond to or follow the input. The physical reason is ultimately some sort of inertia effect in the channel, requiring the input to supply energy in order to generate a response, and therefore requiring some time to respond, because the input power is limited. Thus, a step change in the signal at the input of the channel typically causes a channel output that rises more gradually to its final value, and perhaps with a transient that oscillates or “rings” around its final value before settling. A succession of alternating steps at the input, as would be produced by on-off signaling at the transmitter, may therefore produce an output that displays **inter-symbol interference (ISI)**: the response to the portion of the input signal associated with a particular bit slot spills over into other bit slots at the output, making the output waveform only a feeble representation of the input, and thereby complicating the determination of what the input message was.

To understand channel response and ISI better, we will use **linear, time-invariant (LTI)** models of the channel, which we introduced in the previous chapter. Such models provide very good approximations of channel behavior in a range of applications (they are also widely used in other areas of engineering and science). In an LTI model, the response (i.e., output) of the channel to *any* input depends only on one function, $h[\cdot]$, called the **unit sample response** function. Given any input signal sequence, $x[\cdot]$, the output $y[\cdot]$ of an LTI channel can be computed by combining $h[\cdot]$ and $x[\cdot]$ through an operation known as **convolution**.

Knowledge of $h[\cdot]$ will give us guidance on choosing the number of samples to associate with a bit slot in order to overcome ISI, and will thereby determine the maximum bit rate associated with the channel. In simple on-off signaling, the number of samples that need to be allotted to a bit slot in order to mitigate the effects of ISI will be approximately the number of samples required for the unit sample response $h[\cdot]$ to essentially settle to 0. In this connection, we will also introduce a tool called an **eye diagram**, which allows

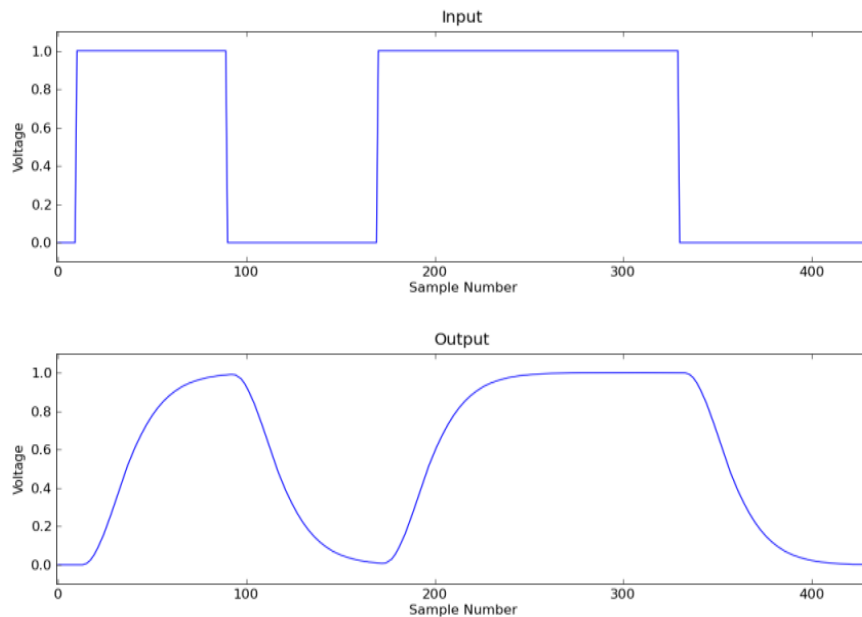


Figure 11-1: On-off signaling at the channel input produces a channel output that takes a non-zero time to rise or fall, and to settle at 1 or 0.

a communication engineer to determine whether the number of samples per bit is large enough to permit reasonable communication quality in the face of ISI.

■ 11.1 Distortions on a Channel

Even though communication technologies come in enormous variety, they generally all exhibit similar types of distorting behavior in response to inputs. To gain some intuition on the basic nature of the problem, we first look at some simple examples. Consider a transmitter that does on-off signaling, sending voltage samples that are either set to $V_0 = 0$ volts or to $V_1 = 1$ volt for all the samples in a bit period. Let us assume an LTI channel, so that

1. the superposition property applies, and
2. if the response to a unit step $u[n]$ at the input is unit-step response $s[n]$ at the output, then the response to a shifted unit step $u[n - D]$ at the input is the identically shifted unit-step response $s[n - D]$, for any (integer) D .

Let us also assume that the channel and receiver gain are such that the unit step response $s[n]$ eventually settles to 1.

In this setting, the output waveform will typically have two notable deviations from the input waveform:

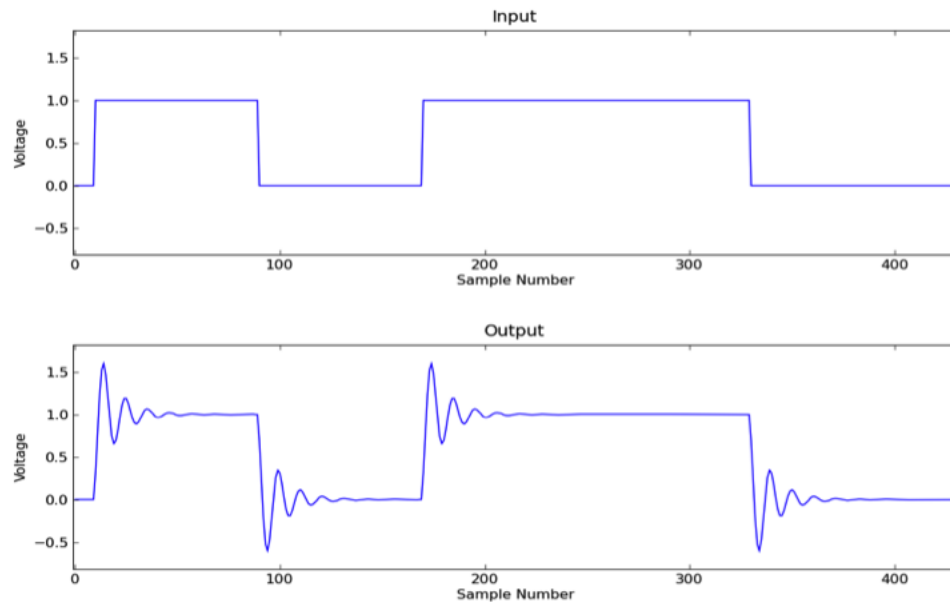


Figure 11-2: A channel showing “ringing”.

1. **A slower rise and fall.** Ideally, the voltage samples at the receiver should be identical to the voltage samples at the transmitter. Instead, as shown in Figure 11-1, one usually finds that the nearly instantaneous transition from V_0 volts to V_1 volts at the transmitter results in an output voltage at the receiver that takes longer to rise from V_0 volts to V_1 volts. Similarly, when there is a nearly instantaneous transition from V_1 volts to V_0 volts at the transmitter, the voltage at the receiver takes longer to fall. It is important to note that if the time between transitions at the transmitter is shorter than the rise and fall times at the receiver, the receiver will struggle (and/or fail!) to correctly infer the value of the transmitted bits using the voltage samples from the output.
2. **Oscillatory settling, or “ringing”.** In some cases, voltage samples at the receiver will oscillate before settling to a steady value. In cables, for example, this effect can be due to a “sloshing” back and forth of the energy stored in electric and magnetic fields, or it can be the result of signal reflections at discontinuities. Over radio and acoustic channels, this behavior arises usually from signal reflections. We will not try to determine the physical source of ringing on a channel, but will instead observe that it happens and deal with it. Figure 11-2 shows an example of ringing.

Figure 11-3 shows an example of non-ideal channel distortions. In the example, the transmitter converted the bit sequence ...0101110... to voltage samples using ten 1 volt samples to represent a “1” and ten 0 volt samples to represent a “0” (with sample values of 0 before and after). In the example, the settling time at the receiver is longer than the reciprocal of the bit period, and therefore bit sequences with frequent transitions, like 010,

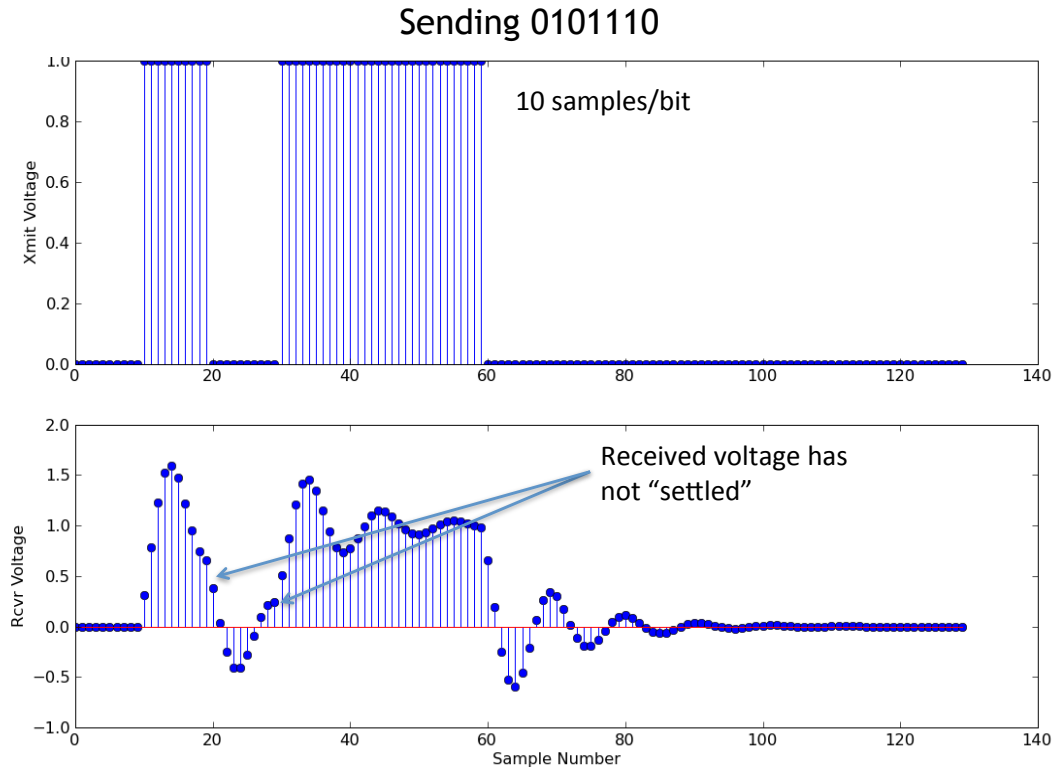


Figure 11-3: The effects of rise/fall time and ringing on the received signal.

are not received faithfully. In Figure 11-3, at sample number 21, the output voltage is still ringing in response to the rising input transition at sample number 10, and is also responding to the input falling transition at sample number 20. The result is that the receiver may misidentify the value of one of the transmitted bits. Note also that the receiver will certainly correctly determine that the fifth and sixth bits have the value '1', as there is no transition between the fourth and fifth, or fifth and sixth, bit.

As this example demonstrates, the slow settling of the channel output implies that the receiver is more likely to wrongly identify a bit that differs in value from its immediate predecessors. This example should also provide the intuition that if the number of samples per bit is large enough, then it becomes easier for the receiver to correctly identify bits because each sequence of samples has enough time to settle to the correct value (in the absence of noise, which is of course a random phenomenon that can still confound the receiver).

There is a formal name given to the impact of rise/fall times and settling times that are long compared to a bit slot: we say that the channel output displays **inter-symbol interference**, or **ISI**. ISI is a fancy way of saying that *the received samples corresponding to the current bit depend on the values of samples corresponding to preceding bits*. Figure 11-4 shows four examples: two for channels with a fast rise/fall compared to the duration of the bit

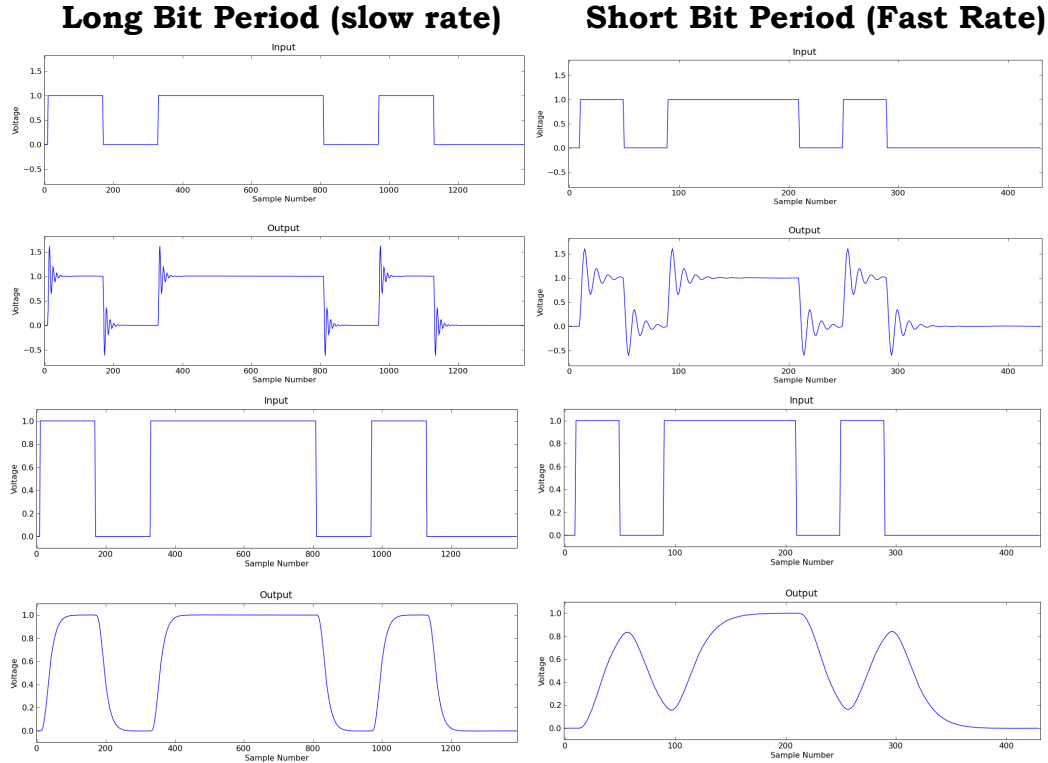


Figure 11-4: Examples of ISI.

slot, and two for channels with a slower rise/fall.

We now turn to a more detailed study of LTI models, which will allow us to understand channel distortion and ISI more fundamentally. The analysis tools we develop, in this chapter and the next two, will also be very valuable in the context of signal processing, filtering, modulation and demodulation, topics that are addressed in the next few chapters.

■ 11.2 Convolution for LTI Models

Consider a discrete-time (DT) linear and time-invariant (LTI) system or channel model that maps an input signal $x[.]$ to an output signal $y[.]$ (Figure 10-4), which shows what the input and output values are at some arbitrary integer time instant n . We will also use other notation on occasion to denote an entire time signal such as $x[.]$, either simply writing x , or sometimes writing $x[n]$ (but with the typically unstated convention that n ranges over all integers!).

A discrete-time (DT) LTI system is completely characterized by its response to a *unit sample function* (or unit pulse function, or unit “impulse” function) $\delta[n]$ at the input. Recall that $\delta[n]$ takes the value 1 where its argument $n = 0$, and the value 0 for all other values of the argument. An alternative notation for this signal that is sometimes useful for clarity is

$\delta_0[\cdot]$, where the subscript indicates the time instant for which the function takes the value 1; thus $\delta[n - k]$, when described as a function of n , could also be written as the signal $\delta_k[\cdot]$.

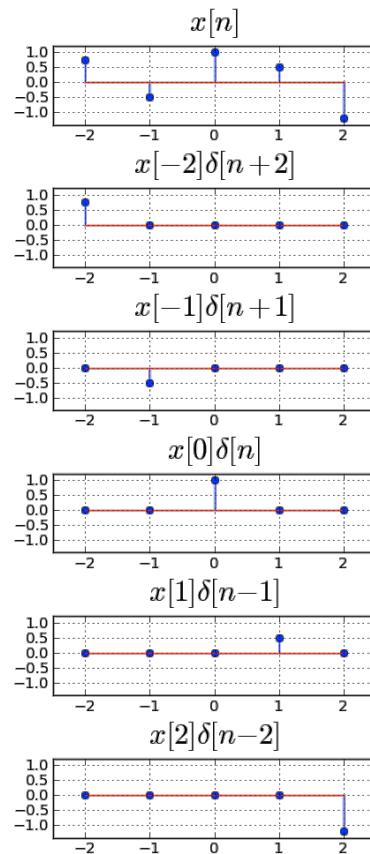


Figure 11-5: A discrete-time signal can be decomposed into a sum of time-shifted, scaled unit-sample functions: $x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - k]$.

The **unit sample response** $h[n]$, with n taking all integer values, is simply the sequence of values that $y[n]$ takes when we set $x[n] = \delta[n]$, i.e., $x[0] = 1$ and $x[k] = 0$ for $k \neq 0$. The response $h[n]$ to the elementary input $\delta[n]$ can be used to characterize the response of an LTI system to *any* input, for the following two reasons:

- An arbitrary signal $x[\cdot]$ can be written as a sum of scaled (or weighted) and shifted unit sample functions, as shown in Figure 11-5. This is expressed in two ways below:

$$\begin{aligned} x[\cdot] &= \cdots + x[-1]\delta_{-1}[\cdot] + x[0]\delta_0[\cdot] + \cdots + x[k]\delta_k[\cdot] + \cdots \\ x[n] &= \cdots + x[-1]\delta[n + 1] + x[0]\delta[n] + \cdots + x[k]\delta[n - k] + \cdots \end{aligned} \quad (11.1)$$

- The response of an LTI system to an input that is the scaled and shifted combination of other inputs is the same scaled combination—or **superposition**—of the correspondingly shifted *responses* to these other inputs, as shown in Figure 11-6.

Since the response at time n to the input signal $\delta[n]$ is $h[n]$, it follows from the two obser-

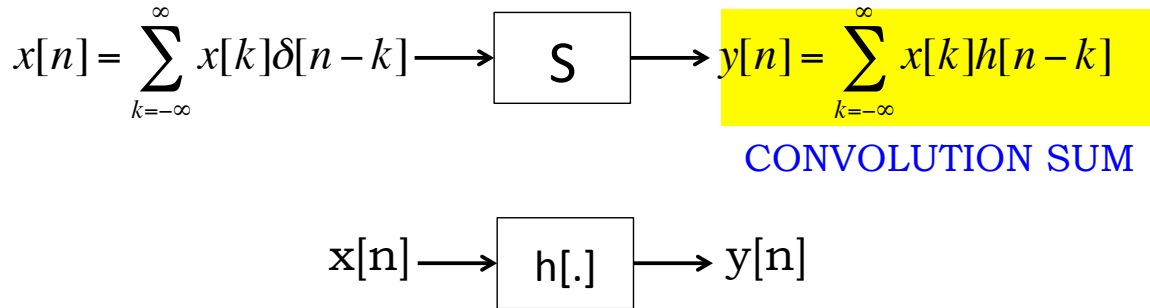


Figure 11-6: Illustrating superposition: If S is an LTI system, then we can use the unit sample response h to predict the response to any waveform x by writing x as a sum of shifted, scaled unit sample functions, and writing the output as a sum of shifted, scaled, unit sample responses, with the same scale factors.

variations above that the response at time n to the input $x[.]$ is

$$\begin{aligned} y[n] &= \cdots + x[-1]h[n+1] + x[0]h[n] + \cdots + x[k]h[n-k] + \cdots \\ &= \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \end{aligned} \quad (11.2)$$

This operation on the time functions or signals $x[.]$ and $h[.]$ to generate a signal $y[.]$ is called **convolution**. The standard symbol for the operation of convolution is $*$, and we use it to write the prescription in Equation (11.2) as $y[n] = (x * h)[n]$. We will also simply write $y = x * h$ when that suffices.¹

A simple change of variables in Equation (11.2), setting $n - k = m$, shows that we can also write

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m] = (h * x)[n]. \quad (11.3)$$

The preceding calculation establishes that convolution is *commutative*, i.e.,

$$x * h = h * x.$$

We will mention other properties of convolution later, in connection with series and parallel combinations (or compositions) of LTI systems.

Example 1 Suppose $h[n] = (0.5)^n u[n]$, where $u[n]$ denotes the unit step function defined previously (taking the value 1 where its argument n is non-negative, and the value 0 when

¹A common (but illogical, confusing and misleading!) notation for convolution in much or most of the engineering literature is to write $y[n] = x[n] * h[n]$. The index n here is doing triple duty: in $y[n]$ it marks the time instant at which the result of the convolution is desired; in $x[n]$ and $h[n]$ it is supposed to denote the entire signals $x[.]$ and $h[.]$ respectively; and finally its use in $x[n]$ and $h[n]$ is supposed to convey the time instant at which the result of the convolution is desired. The defect of this notation is made apparent if one substitutes a number for n , so for example $y[0] = x[0] * h[0]$ —where does one go next with the right hand side? The notation $y[0] = (x * h)[0]$ has no such difficulty. Similarly, the defective notation might encourage one to “deduce” from $y[n] = x[n] * h[n]$ that, for instance, $y[n-3] = x[n-3] * h[n-3]$, but there is no natural interpretation of the right hand side that can convert this into a correct statement regarding convolution.

the argument is strictly negative). If $x[n] = 3\delta[n] - \delta[n - 1]$, then

$$y[n] = 3(0.5)^n u[n] - (0.5)^{n-1} u[n - 1].$$

From this we deduce, for instance, that $y[n] = 0$ for $n < 0$, and $y[0] = 3$, $y[1] = 0.5$, $y[2] = (0.5)^2$, and in fact $y[n] = (0.5)^n$ for all $n > 0$.

The above example illustrates that if $h[n] = 0$ for $n < 0$, then the system output cannot take nonzero values before the input takes nonzero values. Conversely, if the output never takes nonzero values before the input does, then it must be the case that $h[n] = 0$ for $n < 0$. In other words, this condition is necessary and sufficient for **causality** of the system.

The summation in Equation (11.2) that defines convolution involves an infinite number of terms in general, and therefore requires some conditions in order to be well-defined. One case in which there is no problem defining convolution is when the system is causal and the input is zero for all times less than some finite start time s_x , i.e., when the input is *right-sided*. In that case, the infinite sum

$$\sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

reduces to the finite sum

$$\sum_{k=s_x}^n x[k]h[n-k],$$

because $x[k] = 0$ for $k < s_x$ and $h[n-k] = 0$ for $k > n$.

The same reduction to a finite sum occurs if $h[n]$ is just right-sided rather than causal, i.e., is 0 for all times less than some finite start time s_h , where s_h can be negative (if it isn't, then we're back to the case of a causal system). In that case the preceding sum will run from s_x to $n - s_h^2$. Yet another case in this vein involves an input signal or unit sample response that is nonzero over only a finite interval of time, in which case it almost doesn't matter what the characteristics of the other function are, because the convolution yet again reduces to running over the terms in a finite time-window.

When there actually are an infinite number of nonzero terms in the convolution sum, the situation is more subtle. You may recall from discussion of infinite series in your calculus course that such a sum is well-defined—independently of the order in which the terms are added—precisely when the sum of *absolute values* (or magnitudes) of the terms in the infinite series is finite. In this case we say that the series is *absolutely summable*. In the case of the convolution sum, what this requires is the following condition:

$$\sum_{m=-\infty}^{\infty} |h[m]| \cdot |x[n-m]| < \infty \quad (11.4)$$

An important set of conditions under which this constraint is satisfied is when

1. the magnitude or absolute value of the input at each instant is bounded for all time

²The infinite sum also reduces to a finite sum when both $x[\cdot]$ and $h[\cdot]$ are *left-sided*, i.e., are each zero for times *greater* than some finite time; this case is not of much interest in our context.

by some fixed (finite) number, i.e.,

$$|x[n]| \leq \mu < \infty \quad \text{for all } n ,$$

and

2. the unit sample response $h[n]$ is absolutely summable:

$$\sum_{n=-\infty}^{\infty} |h[n]| = \alpha < \infty . \quad (11.5)$$

With this, it follows that

$$\sum_{m=-\infty}^{\infty} |h[m]| \cdot |x[n-m]| \leq \mu \alpha < \infty ,$$

so it's clear that the convolution sum is well-defined in this case.

Furthermore, taking the absolute value of the output $y[n]$ in Equation (11.3) shows that

$$\begin{aligned} |y[n]| &= \left| \sum_{m=-\infty}^{\infty} h[m]x[n-m] \right| \leq \mu \left| \sum_{m=-\infty}^{\infty} h[m] \right| \\ &\leq \mu \sum_{m=-\infty}^{\infty} |h[m]| = \mu \alpha . \end{aligned} \quad (11.6)$$

Thus absolute summability of the unit sample response suffices to ensure that, with a bounded input, we not only have a well-defined convolution sum but that the output is bounded too.

It turns out the converse is true also: absolute summability of the unit sample response is *necessary* to ensure that a bounded input yields a bounded output. One way to see this is to pick $x[n] = \text{sgn}\{h[-n]\}$ for $|n| \leq N$ and $x[n] = 0$ otherwise, where the function $\text{sgn}\{\cdot\}$ is takes the sign of its argument, i.e., is $+1$ or -1 when its argument is respectively positive or negative. With this choice, the convolution sum shows that

$$y[0] = \sum_{n=-N}^N |h[n]|$$

If $h[\cdot]$ is not absolutely summable, then $y[0]$ is unbounded as $N \rightarrow \infty$.

The above facts motivate the name that's given to an LTI system with absolutely summable unit sample response $h[n]$, i.e., satisfying Equation (11.5): the system is termed **bounded-input bounded-output (BIBO) stable**. As an illustration, the system in Example 1 above is evidently BIBO stable, because $\sum_n |h[n]| = 1/(1 - 0.5) = 2$.

Note that because convolution is commutative, the roles of x and h can be interchanged. It follows that convolution is well-defined if the input $x[\cdot]$ is absolutely summable and the unit sample response $h[\cdot]$ is bounded, rather than the other way around; and again, the result of this convolution is bounded.

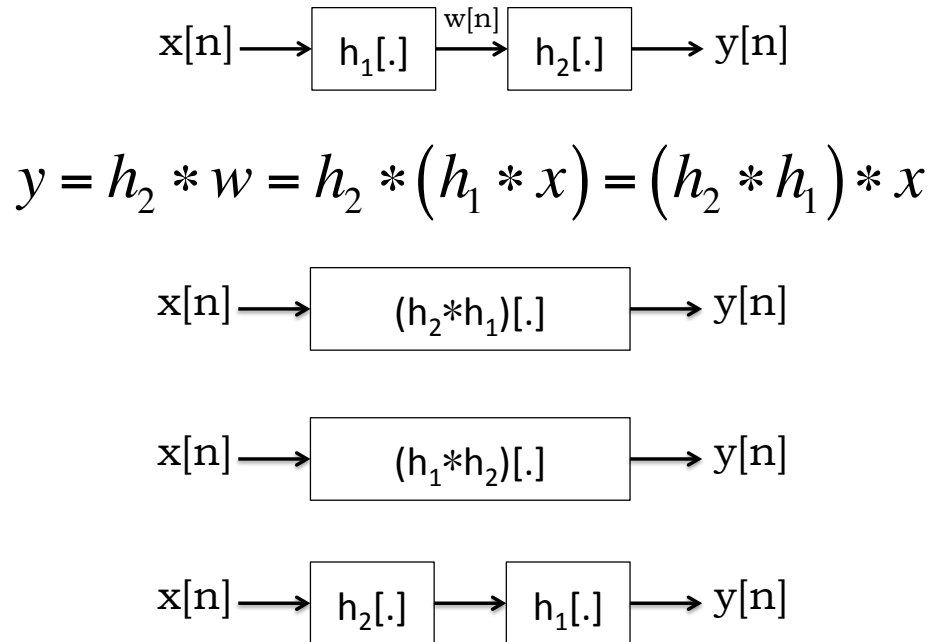


Figure 11-7: LTI systems in series.

■ 11.2.1 Series and Parallel Composition of LTI Systems

We have already noted that convolution is *commutative*, i.e., $x * h = h * x$. It turns out that it is also *associative*, i.e.,

$$(h_2 * h_1) * x = h_2 * (h_1 * x),$$

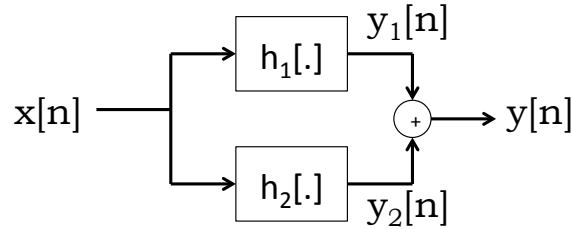
provided each of the involved convolutions is well-behaved. Thus the convolutions—each of which involves two functions—can be done in either sequence. The direct proof is essentially by tedious expansion of each side of the above equation, and we omit it.

These two algebraic properties have immediate implications for the analysis of systems composed of *series* or *cascade* interconnections of LTI subsystems, as in Figure 11-7. The figure shows three LTI systems that are equivalent, in terms of their input-output properties, to the system represented at the top. The proof of equivalence simply involves invoking associativity and commutativity.

A third property of convolution, which is very easy to prove from the definition of convolution, is that it is *distributive* over addition, i.e.,

$$(h_1 + h_2) * x = (h_1 * x) + (h_2 * x),$$

provided each of the individual convolutions on the right of the equation is well-defined. Recall that the addition of two time-functions, as with $h_1 + h_2$ in the preceding equation, is done pointwise, component by component. Once more, there is an immediate application to an interconnection of LTI subsystems, in this case a *parallel* interconnection, as in Figure 11-8.



$$y = y_1 + y_2 = (h_1 * x) + (h_2 * x) = (h_1 + h_2) * x$$

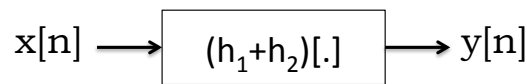


Figure 11-8: LTI systems in parallel.

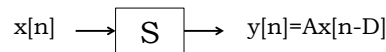


Figure 11-9: Scale-and-delay LTI system.

Example 2 (Scale-&-Delay System) Consider the system \mathcal{S} in Figure 11-9 that scales its DT input by A and delays it by $D > 0$ units of time (or, if D is negative, advances it by $|D|$). This system is linear and time-invariant (as is seen quite directly by applying the definitions from Chapter 10). It is therefore characterized by its unit sample response, which is

$$h[n] = A\delta[n - D].$$

We already know from the definition of the system that if the input at time n is $x[n]$, the output is $y[n] = Ax[n - D]$, but let us check that the general expression in Equation (11.2) gives us the same answer:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n - k] = \sum_{k=-\infty}^{\infty} x[k]A\delta[n - k - D].$$

As the summation runs over k , we look for the unique value of k where the argument of the unit sample function goes to zero, because this is the only value of k for which the unit sample function is nonzero (and in fact equal to 1). Thus $k = n - D$, so $y[n] = Ax[n - D]$, as expected.

A general unit sample response $h[.]$ can be represented as a sum—or equivalently, a parallel combination—of scale-&-delay systems:

$$h[n] = \cdots + h[-1]\delta[n + 1] + h[0]\delta[n] + \cdots + h[k]\delta[n - k] + \cdots . \quad (11.7)$$

An input signal $x[n]$ to this system gets scaled and delayed by each of these terms, with the

results added to form the output. This way of looking at the LTI system response yields the expression

$$\begin{aligned} y[n] &= \cdots + h[-1]x[n+1] + h[0]x[n] + \cdots + h[m]x[n-m] + \cdots \\ &= \sum_{m=-\infty}^{\infty} h[m]x[n-m]. \end{aligned}$$

This is the alternate form of convolution sum we obtained in Equation (11.3).

■ 11.2.2 Flip-Slide-Dotting Away: Implementing Convolution

The above descriptions of convolution explain why we end up with the expressions in Equations (11.2) and (11.3) to describe the output of an LTI system in terms of its input and unit sample response. We will now describe a graphical construction that helps to visualize and implement these computations, and that is often the simplest way to think about the effects of convolution.

Let's examine the expression in Equation (11.2), but the same kind of reasoning works for Equation (11.3). Our task is to implement the computation in the summation below:

$$y[n_0] = \sum_{k=-\infty}^{\infty} x[k]h[n_0 - k]. \quad (11.8)$$

We've written n_0 rather than the n we used before just to emphasize that this computation involves summing over the dummy index k , with the other number being just a parameter, fixed throughout the computation.

We first plot the time functions $x[k]$ and $h[k]$ on the k axis (with k increasing to the right, as usual). How do we get $h[n_0 - k]$ from this? First note that $h[-k]$ is obtained by reversing $h[k]$ in time, i.e., a **flip** of the function across the time origin. To get $h[n_0 - k]$, we now **slide** this reversed time function, $h[-k]$, to the *right* by n_0 steps if $n_0 \geq 0$, or to the *left* by $|n_0|$ steps if $n_0 < 0$. To confirm that this prescription is correct, note that $h[n_0 - k]$ should take the value $h[0]$ at $k = n_0$.

With these two steps done, all that remains is to compute the sum in Equation (11.8). This sum takes the same form as the familiar **dot product** of two vectors, one of which has $x[k]$ as its k th component, and the other of which has $h[n_0 - k]$ as its k th component. The only twist here is that the vectors could be infinitely long. So what this steps boils down to is taking an instant-by-instant product of the time function $x[k]$ and the time function $h[n_0 - k]$ that your preparatory "flip and slide" step has produced, then summing all the products.

At the end of all this (and it perhaps sounds more elaborate than it is, till you get a little practice), what you have computed is the value of the convolution for the *single* value n_0 . To compute the convolution for another value of the argument, say n_1 , you repeat the process, but sliding by n_1 instead of n_0 .

To implement the computation in Equation (11.3), you do the same thing, except that now it's $h[m]$ that stays as it is, while $x[m]$ gets flipped and slid by n to produce $x[n - m]$, after which you take the dot product. Either way, the result is evidently the same.

Example 1 revisited Suppose again that $h[m] = (0.5)^m u[m]$ and $x[m] = 3\delta[m] - \delta[m - 1]$. Then

$$x[-m] = -\delta[-m - 1] + 3\delta[-m] ,$$

which is nonzero only at $m = -1$ and $m = 0$. (Sketch this!) As a consequence, sliding $x[-m]$ to the left, to get $x[n - m]$ when $n < 0$, will mean that the nonzero values of $x[n - m]$ have *no overlap* with the nonzero values of $h[m]$, so the dot product will yield 0. This establishes that $y[n] = (x * h)[n] = 0$ for $n < 0$, in this example.

For $n = 0$, the only overlap of nonzero values in $h[m]$ and $x[n - m]$ is at $m = 0$, and we get the dot product to be $(0.5)^0 \times 3 = 3$, so $y[0] = 3$.

For $n > 0$, the only overlap of nonzero values in $h[m]$ and $x[n - m]$ is at $m = n - 1$ and $m = n$, and the dot product evaluates to

$$y[n] = -(0.5)^{n-1} + 3(0.5)^n = (0.5)^{n-1}(-1 + 1.5) = (0.5)^n .$$

So we have completely recovered the answer we obtained in Example 1. For this example, our earlier approach—which involved directly thinking about superposition of scaled and shifted unit sample responses—was at least as easy as the graphical approach here, but in other situations the graphical construction can yield more rapid or direct insights.

■ 11.2.3 Deconvolution

We've seen in the previous chapter how having an LTI model for a channel allows us to predict or analyze the distorted output $y[n]$ of the channel, in response to a superposition of alternating positive and negative steps at the input $x[n]$, corresponding to a rectangular-wave baseband signal. That analysis was carried out in terms of the unit step response, $s[n]$, of the channel.

We now briefly explore one plausible approach to *undoing* the distortion of the channel, assuming we have a good LTI model of the channel. This discussion is most naturally phrased in terms of the unit sample response of the channel rather than the unit step response. The idea is to process the received baseband signal $y[n]$ through an LTI system, or LTI *filter*, that is designed to cancel the effect of the channel.

Consider a simple channel that we model as LTI with unit sample function

$$h_1[n] = \delta[n] + 0.8\delta[n - 1] .$$

This is evidently a causal model, and we might think of the channel as one that transmits perfectly and instantaneously along some direct path, and also with a one-step delay and some attenuation along some echo path.

Suppose our receiver filter is to be designed as a causal LTI system with unit sample response

$$h_2[n] = h_2[0]\delta[n] + h_2[1]\delta[n - 1] + \cdots + h_2[k]\delta[n - k] + \cdots . \quad (11.9)$$

Its input is $y[n]$, and let us label its output as $z[n]$. What conditions must $h_2[n]$ satisfy if we are to ensure that $z[n] = x[n]$ for all inputs $x[n]$, i.e., if we are to undo the channel distortion?

An obvious place to start is with the case where $x[n] = \delta[n]$. If $x[n]$ is the unit sample function, then $y[n]$ is the unit sample response of the channel, namely $h_1[n]$, and $z[n]$ will

then be given by $z[n] = (h_2 * h_1)[n]$. In order to have this be the input that went in, namely $x[n] = \delta[n]$, we need

$$(h_2 * h_1)[n] = \delta[n]. \quad (11.10)$$

And if we satisfy this condition, then we will actually have $z[n] = x[n]$ for arbitrary $x[n]$, because

$$z = h_2 * (h_1 * x) = (h_2 * h_1) * x = \delta_0 * x = x,$$

where $\delta_0[\cdot]$ is our alternative notation for the unit sample function $\delta[n]$. The last equality above is a consequence of the fact that convolving any signal with the unit sample function yields that signal back again; this is in fact what Equation (11.1) expresses.

The condition in Equation (11.10) ensures that the convolution carried out by the channel is inverted or undone, in some sense, by the filter. We might say that the filter **deconvolves** the output of the system to get the input (but keep in mind that it does this by a further convolution!). In view of Equation (11.10), the function $h_2[\cdot]$ is also termed the *convolutional inverse* of $h_1[\cdot]$, and vice versa.

So how do we find $h_2[n]$ to satisfy Equation (11.10)? It's **not** by a simple division of any kind (though when we get to doing our analysis in the frequency domain shortly, it will indeed be as simple as division). However, applying the “flip–slide–dot product” mantra for computing a convolution, we find the following equations for the unknown coefficients $h_2[k]$:

$$\begin{aligned} 1 \cdot h_2[0] &= 1 \\ 0.8 \cdot h_2[0] + 1 \cdot h_2[1] &= 0 \\ 0.8 \cdot h_2[1] + 1 \cdot h_2[2] &= 0 \\ &\dots \\ 0.8 \cdot h_2[k-1] + 1 \cdot h_2[k] &= 0 \\ &\dots, \end{aligned}$$

from which we get $h_2[0] = 1$, $h_2[1] = -0.8$, $h_2[2] = -0.8h_2[1] = (-0.8)^2$, and in general $h_2[k] = (-0.8)^k u[k]$.

Deconvolution as above would work fine if our channel model was accurate, and if there were no noise in the channel. Even assuming the model is sufficiently accurate, note that any noise process $w[\cdot]$ that adds in at the output of the channel will end up adding $v[n] = (h_2 * w)[n]$ to the noise-free output, which is $z[n] = x[n]$. This added noise can completely overwhelm the solution. For instance, if both $x[n]$ and $w[n]$ are unit samples, then the output of the receiver's deconvolution filter has a noise-free component of $\delta[n]$ and an additive noise component of $(-0.8)^n u[n]$ that dwarfs the noise-free part. After we've understood how to think about LTI systems in the frequency domain, it will become much clearer why such deconvolution can be so sensitive to noise.

■ 11.3 Relating the Unit Step Response to the Unit Sample Response

Since

$$\delta[n] = u[n] - u[n - 1]$$

it follows that for an LTI system the unit sample response $h[n]$ and the unit step response $s[n]$ are simply related:

$$h[n] = s[n] - s[n - 1].$$

This relation is a consequence of applying superposition and invoking time-invariance. Hence, for a causal system that has $h[k] = 0$ and $s[k] = 0$ for $k < 0$, we can invert this relationship to write

$$s[n] = \sum_0^n h[k]. \quad (11.11)$$

We can therefore very simply determine the unit step response from the unit sample response. It also follows from Equation (11.11) that the time it takes for the unit step response $s[n]$ to settle to its final value is precisely the time it takes for the unit sample response $h[n]$ to settle back down to 0 and stay there.

When the input to an LTI system is a sum of scaled and delayed unit steps, there is no need to invoke the full machinery of convolution to determine the system output. Instead, knowing the unit step response $s[n]$, we can again simply apply superposition and invoke time-invariance.

We describe next a tool for examining the channel response under this ISI, and for setting parameters at the transmitter and receiver.

■ 11.4 Eye Diagrams

On the face of it, ISI is a complicated effect because the magnitude of bit interference and the number of interfering bits depend both on the channel properties and on how bits are represented on the channel. Figure 11-10 shows an example of what the receiver sees (bottom) in response to what the transmitter sent (top) over a channel with ISI but no noise. Eye diagrams (or “eye patterns”) are a useful graphical tool in the toolkit of a communications system designer or engineer to understand ISI. We will use this tool to determine whether the number of samples per bit is large enough to enable the receiver to determine “0”s and “1”s reliably from the demodulated (and filtered) sequence of received voltage samples.

To produce an eye diagram, one begins with the channel output that results from a long stretch of on-off signaling, as in the bottom part of Figure 11-10, then essentially slices this up into smaller segments, say 3 bit-slots long, and overlays all the resulting segments. The result spans the range of waveform variations one is likely to see over any 3 bit-slots at the output. A more detailed prescription follows.

Take all the received samples and put them in an array of *lists*, where the number of lists in the array is equal to the number of samples in k bit periods. In practice, we want k to be a small positive integer like 3. If there are s samples per bit, the array is of size $k \cdot s$.

Each element of this array is a *list*, and element i of the array is a list of the received

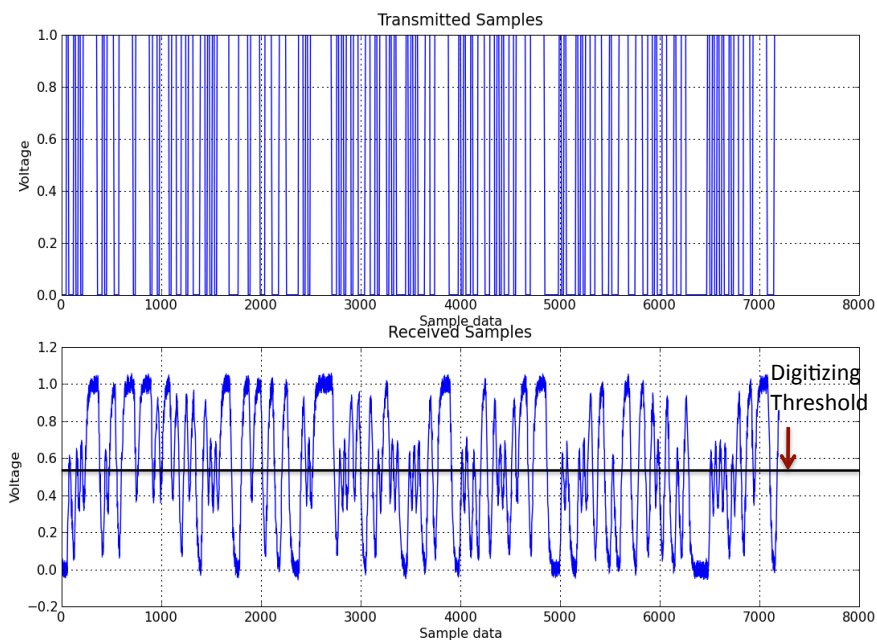


Figure 11-10: Received signals in the presence of ISI. Is the number of samples per bit “just right”? And what threshold should be used to determine the transmitted bit? It’s hard to answer these question from this picture. An eye diagram sheds better light.

samples $y[i], y[i + ks], y[i + 2ks], \dots$. Now suppose there were no ISI at all (and no noise). Then all the samples in the i^{th} list corresponding to a transmitted “0” bit would have the same voltage value, and all the samples in the i^{th} list corresponding to a transmitted “1” would have the same value. Consider the simple case of just a little ISI, where the previous bit interferes with the current bit, and there’s no further impact from the past. Then the samples in the i^{th} list corresponding to a transmitted “0” bit would have two distinct possible values, one value associated with the transmission of a “10” bit sequence, and one value associated with a “00” bit sequence. A similar story applies to the samples in the i^{th} list corresponding to a transmitted “1” bit, for a total of *four* distinct values for the samples in the i^{th} list. If there is more ISI, there will be more distinct values in the i^{th} list of samples. For example, if two previous bits interfere, then there will be eight distinct values for the samples in the i^{th} list. If three bits interfere, then the i^{th} list will have 16 distinct values, and so on.

Without knowing the number of interfering bits, to capture all the possible interactions, we must produce the above array of lists for every possible combination of bit sequences that can ever be observed. If we were to plot this array on a graph, we will see a picture like the one shown in Figure 11-11. This picture is an eye diagram.

In practice, we can’t produce every possible combination of bits, but what we can do is use a long random sequence of bits. We can take the random bit sequence, convert it in to a long sequence of voltage samples, transmit the samples through the channel, collect the

received samples, pack the received samples in to the array of lists described above, and then plot the result. If the sequence is long enough, and the number of interfering bits is small, we should get an accurate approximation of the eye diagram.

But what is “long enough”?

We can answer this question and develop a less *ad hoc* procedure by using the properties of the unit sample response, $h[n]$. The idea is that the sequence $h[0], h[1], \dots, h[n], \dots$ captures the complete noise-free response of the channel. If $h[k] \approx 0$ for $k > \ell$, then we don't have to worry about samples more than ℓ in the past. Now, if the number of samples per bit is s , then the number of *bits* in the past that can affect the present bit is no larger than ℓ/s , where ℓ is the length of the non-zero part of $h[\cdot]$. Hence, it is enough to generate all bit patterns of length $B = \ell/s$, and send them through the channel to produce the eye diagram. In practice, because noise can never be eliminated, one might be a little conservative and pick $B = (\ell/s) + 2$, slightly bigger than what a noise-free calculation would indicate. Because this approach requires 2^B bit patterns to be sent, it might be unreasonable for large values of B ; in those cases, it is likely that s is too small, and one can find whether that is so by sending a random subset of the 2^B possible bit patterns through the channel.

Figure 11-11 shows the *width of the eye*, the place where the diagram has the largest distinction between voltage samples associated with the transmission of a '0' bit and those associated with the transmission of a '1' bit. Another point to note about the diagrams is the “zero crossing”, the place where the upward rising and downward falling curves cross. Typically, as the degree of ISI increases (i.e., the number of samples per bit is reduced), there is a greater degree of “fuzziness” and ambiguity about the location of this zero crossing.

The eye diagram is an important tool, useful for verifying some key design and operational decisions:

1. Is the number of samples per bit large enough? If it is large enough, then at the center of the eye, the voltage samples associated with transmission of a '1' are clearly above the digitization threshold and the voltage samples associated with the transmission of a '0' are clearly below. *In addition*, the eye must be “open” enough that small amounts of noise will not lead to errors in converting bit detection samples to bits. As will become clear later, it is impossible to guarantee that noise will never cause errors, but we can reduce the likelihood of error.
2. Has the value of the digitization threshold been set correctly? The digitization threshold should be set to the voltage value that evenly divides the upper and lower halves of the eye, if 0's and 1's are equally likely. We didn't study this use of eye diagrams, but mention it because it is used in practice for this purpose as well.
3. Is the sampling instant at the receiver within each bit slot appropriately picked? This sampling instant should line up with where the eye is most open, for robust detection of the received bits.

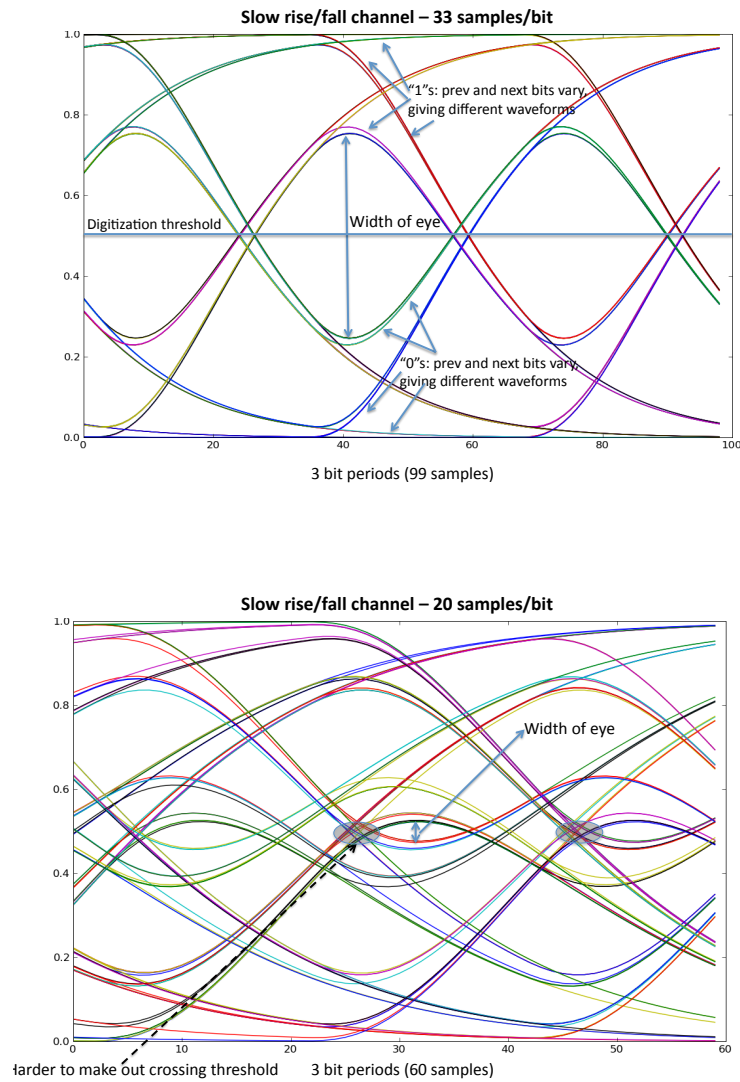


Figure 11-11: Eye diagrams for a channel with a slow rise/fall for 33 (top) and 20 (bottom) samples per bit. Notice how the eye is wider when the number of samples per bit is large, because each step response has time to settle before the response to the next step appears.