$$\mathrm{Pr}_e = \sum_{j=\left\lceil \frac{N_s}{2} \right\rceil}^{N_s} \binom{N_s}{j} \mathrm{Pr}_{e0}{}^{j} \left(1-\mathrm{Pr}_{e0}\right)^{N_s-j} \tag{8–52}$$

where Pr_{e0} is the probability of error on the single decision, that is, the probability of error for an isolated pulse receiver as derived in Sections 8.1.1–8.1.5 for the different modulation formats.

Performance comparison between soft and hard detection is a typical problem of digital communications. A detailed analysis can be found in (Proakis, 1995) for different coding schemes (remember that the presence of N_s pulses per bit derives from the application of a code repetition coder before modulation). It can be shown, in general, that for a variety of code families, soft decision outperforms hard decision when considering propagation over an AWGN channel. This result will be shown on simulation data in Checkpoint 8–1.

The above performance loss in hard decision versus soft decision refers to the Gaussian noise case. In the presence of non-Gaussian noise, as with interference noise, for example, an opposite trend might be observed. With reference to the IR-UWB case, in particular (Weeks et al., 1999), showed that hard decision can perform better than soft decision when several UWB interfering signals are present at the receiver. Performance of hard decision in this case is mainly affected by the total number of interferers, that is the symbol error rate decreases due to the presence of collisions when deciding for a single pulse, while performance of the soft decision scheme is mainly affected by the average interfering power that is collected at the receiver during an entire symbol period. In the presence of great variations in the interfering signal powers, as occurs for example in the case of a few close interferers and several distant interferers, hard decision can perform better than soft decision.

CHECKPOINT 8–1

In this checkpoint, we will simulate the behavior of the receivers described in Section 8.1. New MATLAB functions are required for simulating both the propagation of the transmitted signal over the AWGN channel and detection at the receiver. Performance is evaluated in terms of measured average error probability at the output of the detector.

Regarding the transmitter, PPM and PAM signals can be generated by using the MATLAB functions introduced in Chapter 2, in particular Functions 2.6 and 2.9 which generate PPM-TH and PAM-DS signals, respectively.

To simulate propagation of these signals over the AWGN channel, three MATLAB functions are defined: Function 8.1 for emulating the attenuation of the signal energy with distance and Functions 8.2 and 8.3 for introducing thermal noise at the receiver input. We assume perfect synchronization between transmitter and receiver and $\tau = 0$, that is, no additional MATLAB functions are required for simulating the effect of channel delay.

Function 8.1 (see Appendix 8.A) attenuates the input signal `tx` according to the rule of Eq. (8–3), that is, the channel gain is evaluated as a function of the distance `d` between transmitter and receiver, and two constant terms `gamma` and `c0` representing the power

decay with distance and the reference attenuation at 1 meter. Function 8.1 returns vector `rx`, representing the attenuated signal and the value of the channel gain `attn`.

The command line for executing Function 8.1 is:

```
[rx,attn] = cp0801_pathloss(tx,c0,d,gamma);
```

Function 8.2 (see Appendix 8.A) generates a white Gaussian noise signal and adds it to the input signal given via vector `input`. The noise level is determined according to a specified E_b/N_0 ratio, where E_b is the energy per bit of the useful signal at the receiver. Function 8.2 can manage multiple E_b/N_0 values, which must be given as input to the function in vector `ebno`. Values in `ebno` are in logarithmic units. In the presence of multiple E_b/N_0 values, Function 8.2 generates multiple outputs. For a given E_b/N_0 value, Function 8.2 measures the E_b value from the input signal and determines the corresponding N_0 value. Measuring the E_b value, in particular, requires knowledge of the number of bits that are conveyed by the input signal. This value is given as input by `numbits`. Function 8.2 returns the array `output`, which contains all signals resulting from the introduction of Gaussian noise over the input signal. Each row of the array output corresponds to one E_b/N_0 value. Function 8.2 also returns the array `noise`, which contains different realizations of the Gaussian noise, one realization per row.

The command line for executing Function 8.2 is:

```
[output,noise] = cp0801_Gnoise1(input,ebno,numbits);
```

The above command stores the array `output` in memory, which contains all signals generated by Function 8.2 for the different E_b/N_0 values contained in the input vector `ebno`.

We can use Functions 8.1 and 8.2 for evaluating the effect of propagation of 2PPM-TH signals over AWGN channels. First, the transmitted signal is generated by executing Function 2.6 (see Checkpoint 2–1) with the following parameters: `Pow=-30; fc=50e9; numbits=2; Ts=3e-9; Ns=1; Tc=1e-9; Nh=3; Np=2; Tm=0.5e-9; tau=0.25e-9; dPPM=0.5e-9; G=1`. The command line for generating the 2PPM-TH signal is:

```
[bits,THcode,stx0,ref]=cp0201_transmitter_2PPM_TH;
```

The above command stores vector `bits` in memory, which contains the binary sequence generated by the source and vector `stx0` representing the transmitted signal, and produces the graphical output of Figure 8–14.

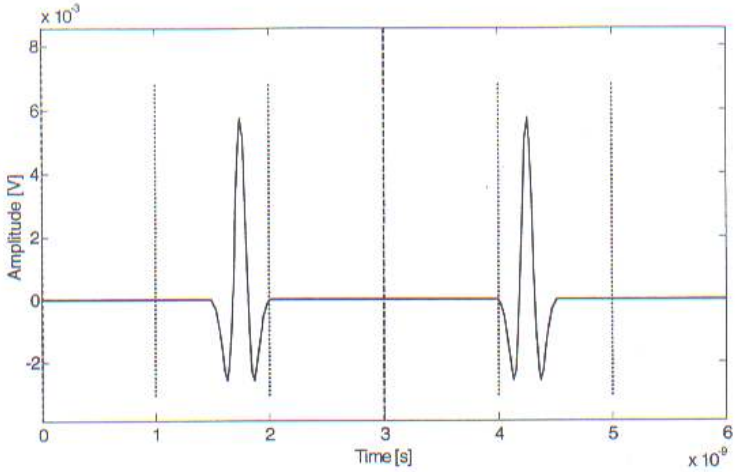

Figure 8–14 Transmitted 2PPM-TH signal `stx0`.

Figure 8–14 shows that the transmitted signal `stx0` consists of two pulses, located in the central slot of the corresponding frame. The peak value of each pulse is on the order of $6 \cdot 10^{-3}$ V.

Function 8.1 can now be used for simulating the attenuation suffered by signal `stx0` during propagation. In particular, we assume the receiver to be 10 meters away from the transmitter in a free space scenario with 30 dB of power loss at a reference distance of 1 meter. The following quantities are thus defined:

```
d = 10;
gamma = 2;
c0 = 10^(-30/20);
```

and then the following command is executed:

```
[srx0,attn] = cp0801_pathloss(stx0,c0,d,gamma);
```

The above command stores vector `srx0` in memory, which represents the attenuated version of signal `stx0`. Figure 8–15 shows the envelope of signal `srx0`.

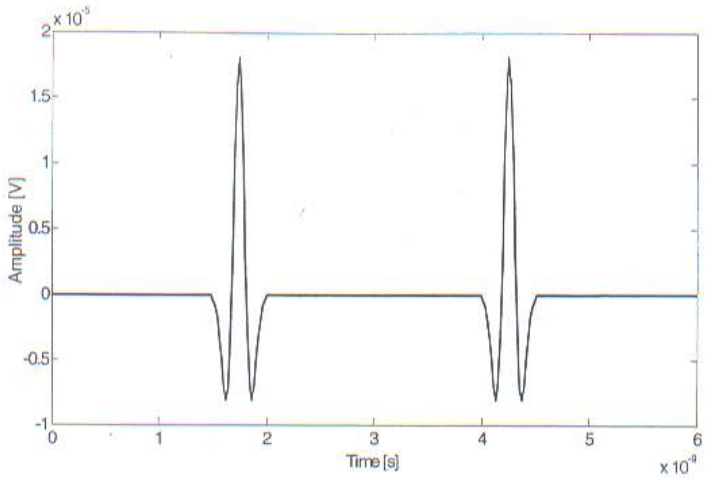

Figure 8–15 Signal `srx0`, that is, signal `stx0` after 10 meters propagation over the free space.

The peak value of the two pulses in Figure 8–15 are on the order of $2 \cdot 10^{-5}$ V. The channel gain α is thus about 0.0033.

Thermal noise can be introduced in signal `srx0` by using Function 8.2. Four different E_b/N_0 values are considered: 0 dB, 10 dB, 20 dB, and 30 dB. The following quantities are thus defined:

```
numbits = length(bits);
ebno = [0 10 20 30];
```

and the following commands are executed:

```
[rx0,noise] = cp0801_Gnoise1(srx0,ebno,numbits);
rx1 = rx0(1,:);
rx2 = rx0(2,:);
rx3 = rx0(3,:);
rx4 = rx0(4,:);
```

The above commands store four different signals in memory, `rx1`, `rx2`, `rx3` and `rx4`, which correspond to the introduction of increasing levels of thermal noise at the receiver. The graphical representation of these signals is presented in Figures 8–16, 8–17, 8–18, and 8–19.

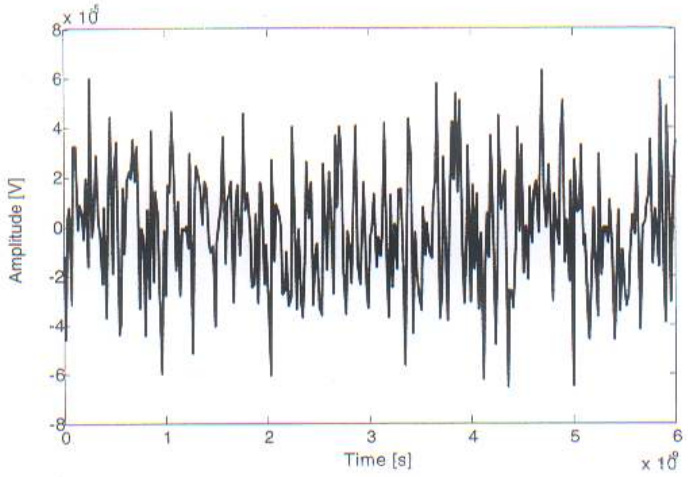

Figure 8–16 Signal rx1 (E_b/N_0 = 0 dB).

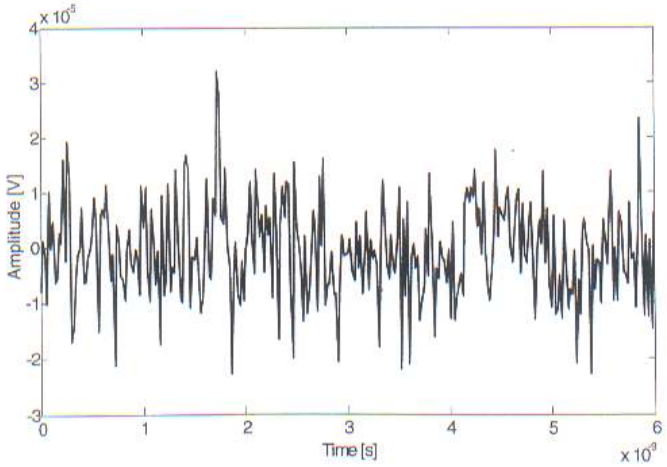

Figure 8–17 Signal rx2 (E_b/N_0 = 10 dB).

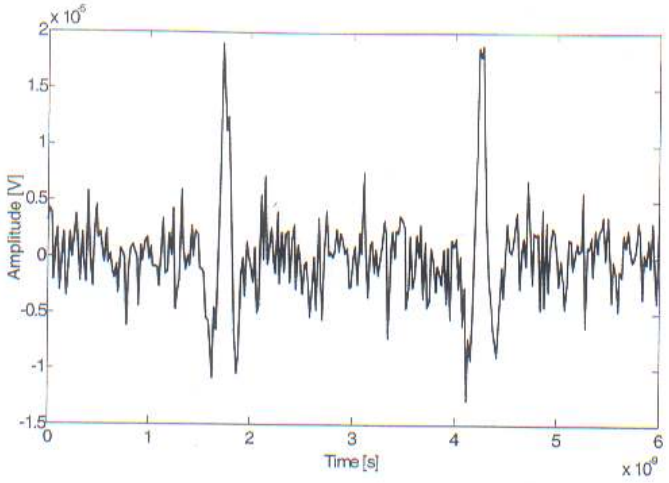

Figure 8–18 Signal rx3 (E_b/N_0 = 20 dB).

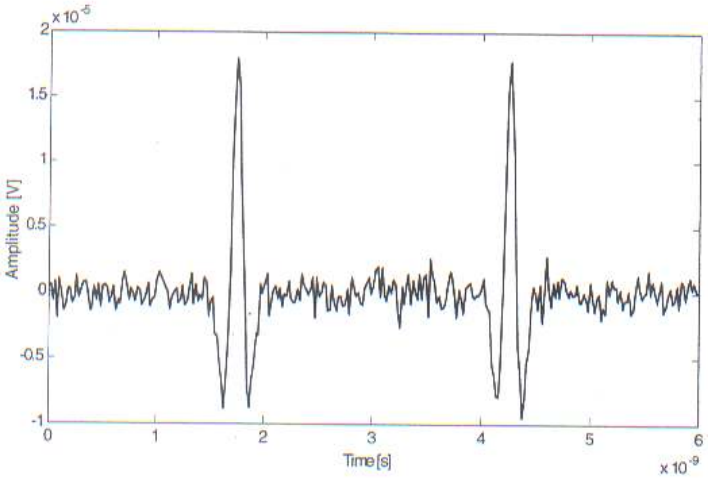

Figure 8–19 Signal rx4 (E_b/N_0 = 30 dB).

Figures 8–16 to 8–19 show the effect over the useful signal waveform of the presence of thermal noise at the receiver. As expected, distortion is more evident for low E_b/N_0 values.

The above simulation, which has focused on binary PPM, can be obtained in a similar way for PAM signals.

Function 8.3 (see Appendix 8.A) is a second MATLAB function that can be used for generating Gaussian noise. Function 8.3 is similar to Function 8.2, except for the fact that the noise level is evaluated according to a target E_x/N_0 value rather than a target E_b/N_0 value, where E_x is the received energy within a single pulse. The E_x value is determined within the function by dividing the total measured energy by the number of transmitted pulses. The number of pulses, which is eventually higher than the number of transmitted bits (in the presence of code repetition coding), must be given as input via the parameter `numpulses`. The command line for executing Function 8.3 is:

```
[output,noise] = cp0801_Gnoise2(input,exno,numpulses);
```

Once again, the array `output` contains signals corresponding to different input values of E_x/N_0, one signal per row.

We can now introduce the MATLAB functions used for simulating receiver activity.

According to the analysis of Section 8.1, the optimum receiver for the AWGN channel consists of a signal correlator followed by a ML detector. As shown in Figure 8–3, the correlator cross-correlates the signal at the receiver input with multiple waveforms to obtain the correlation metrics. The detector then applies a decision rule by observing the correlation metrics. We simulate the above process in the specific cases of binary orthogonal PPM-TH, and of binary antipodal PAM-DS. Functions 8.4 and 8.5 are introduced for the PPM-TH case, while Functions 8.6 and 8.7 serve for the PAM-DS case.

Function 8.4 (see Appendix 8.A) evaluates the correlation mask m(t) of the correlator of Figure 8–6. To determine the correlation mask, the receiver must know the exact position of all pulses within each frame of the transmitted signal. When a PPM-TH signal is generated via Function 2.6 (see Appendix 2.A), this information is available in the output vector `ref`. Vector `ref`, in fact, is a copy of the transmitted signal `Stx`, except for the absence of PPM modulation. Function 8.4 receives as input vector `ref` together with the information on sampling frequency `fc`, number of transmitted pulses `numpulses`, and value of the PPM shift `dPPM` used for generating the transmitted signal. Function 8.4 returns vector `mask`, which represents the waveform of the correlation mask and is executed as follows:

```
[mask] = cp0801_PPMcorrmask(ref,fc,numpulses,dPPM);
```

Note that vector `mask` represents the correlation mask for the whole transmitted signal. To operate the decision over a single bit period, the code that implements the receiver divides the above vector in different parts, each part corresponding to the correlation mask for a single bit period.

Function 8.5 (see Appendix 8.A) simulates the activity of both correlator and detector in the receiver scheme of Figure 8–6, and evaluates receiver performance in terms of average probability of error on the bit P_b as measured after decision. Multiple signals corresponding to the same binary stream can be analyzed all together, that is, the P_b for different values of noise level can be evaluated with a single run of the code.

The following inputs are required: array `R` representing the different waveforms to be considered (one waveform per row), vector `mask` representing the correlation mask (the same for all signals), the value of the sampling frequency `fc`, the original binary stream generated by the transmitter `bits`, the number of pulses per bit `Ns`, and the average pulse

repetition period in seconds `Ts`. Function 8.5 returns two outputs: the array `RXbits` containing the different binary streams at the receiver output (one stream on each row) and vector `BER` containing the measured Pr_b values. The command line for executing Function 8.5 is:

```
[RXbits,BER] = cp0801_PPMreceiver(R, mask, fc,...
   bits, numbit, Ns, Ts);
```

Function 8.6 (see Appendix 8.A) evaluates the correlation mask m(t) in the case of PAM-DS signals (see Figure 8–10). The algorithm implemented in Function 8.6 is similar to Function 8.4. It requires the following inputs: the reference vector `ref`, which is provided by Function 2.9 when generating the PAM-DS signal; the value of the sampling frequency `fc`; and the number of transmitted pulses `numpulses`. Function 8.6 returns vector `mask`, which represents the waveform of the correlation mask and is executed as follows:

```
[mask] = cp0801_PAMcorrmask(ref,fc,numpulses);
```

Once again, note that vector `mask` represents the correlation mask for the whole transmitted signal. To operate the decision over a single bit period, the receiver divides the above vector into different parts, each part corresponding to the correlation mask for a single bit period.

Function 8.7 (see Appendix 8.A) simulates the activity of both correlator and detector for the scheme in Figure 8–10, and evaluates receiver performance in terms of measured Pr_b at the receiver output. Function 8.7 is similar to Function 8.5, except for the decision rule. In the PAM case, in fact, the detector decides for a 0 bit in the presence of negative values at the correlation output, and for a 1 bit for positive values at the correlation output. The command line for executing Function 8.7 is:

```
[RXbits,BER] = ...
   cp0801_PAMreceiver(R,mask,fc,bits,numbit,Ns,Ts);
```

Functions 8.5 to 8.7 can be used to evaluate by simulation the performance of the receiver structures described in Section 8.1. We start by considering the case of binary orthogonal PPM-TH. Two UWB signals with different N_s values are generated: signal `s_ppm1` with `Ns=1` and signal `s_ppm2` with `Ns=3`. The remaining parameters of Function 2.6 are the same for both signals: `Pow=-30; fc=50e9; numbits=10000; Ts=3e-9; Tc=1e-9; Nh=3; Np=30000; Tm=0.5e-9; tau=0.25e-9; dPPM=0.5e-9; G=0.`

The output parameters for the two signals are obtained as follows:

```
[bits1,THcode,s_ppm1,ref1]=cp0201_transmitter_2PPM_TH;
[bits2,THcode,s_ppm2,ref2]=cp0201_transmitter_2PPM_TH;
```

Note that in the case of signal `s_ppm2`, the transmitter generates three pulses for each bit. The number of transmitted bits is thus the same as signal `s_ppm1`, but the number of transmitted pulses is multiplied by three.

Given `s_ppm1` and `s_ppm2`, we can determine the corresponding correlator masks, `mppm1` for signal `s_ppm1` and `mppm2` for `s_ppm2`:

```
fc = 50e9;
dPPM = 0.5e-9;
Ts = 3e-9;
numbit = 10000;
numpulses1 = 10000;
[mppm1] = cp0801_PPMcorrmask(ref1,fc,numpulses1,dPPM);
numpulses2 = 30000;
[mppm2] = cp0801_PPMcorrmask(ref2,fc,numpulses2,dPPM);
```

We can now simulate the effect of propagation over the AWGN channel. Our aim is to evaluate performance for the different signal formats as a function of the signal-to-noise ratio at the receiver. In particular, we are interested in evaluating Pr_b vs. E_x/N_0, where E_x is the energy received within a single pulse. Note that both Functions 8.2 and 8.3 introduce thermal noise by measuring the energy of the received signal, that is, either E_b or E_x, and by then generating a noise signal with the corresponding N_0 value. As a consequence, in this particular analysis, channel gain does not affect receiver performance and we can introduce thermal noise directly in the transmitted signal. To evaluate Pr_b when E_x/N_0 is in the range between 0 dB and 8 dB, we define the following vector `exno`:

```
exno = [ 0 2 4 6 8 ];
```

Then, we generate noise at the receiver by executing Function 8.3:

```
[rx_ppm1,noise] = cp0801_Gnoise2(s_ppm1, exno,...
   numpulses1);
[rx_ppm2,noise] = cp0801_Gnoise2(s_ppm2, exno,...
   numpulses2);
```

Both arrays `rx_ppm1` and `rx_ppm2` are composed of five rows. Each row is associated with one E_x/N_0 value.

The final step of the simulation requires the introduction of Function 8.5. In the case of signal `s_ppm1`, the parameter `HDSD` does not affect performance since hard decision is equivalent to soft decision given that one pulse is transmitted per bit. The code for evaluating Pr_b in the case of signal `s_ppm1` is:

```
[RXbits1, BER1] = cp0801_PPMreceiver(rx_ppm1,...
   mppm1, fc, bits1, numbit, 1, Ts);
```

The above command stores the following in memory: the five estimated binary streams `RXbits1` and vector `BER` of the corresponding Pr_b values. Figure 8–20 illustrates the result of the simulation by representing Pr_b for different values of E_x/N_0. As expected, Pr_b decreases with E_x/N_0 in dB. Moreover, since the PPM shift of signal `s_ppm1` is equal to the time duration of the pulse, the result in Figure 8–20 corresponds to the case of binary transmission with orthogonal signaling. By comparing Figure 8–20 with Figure 8–7, we verify the agreement of the simulation results with the values derived analytically.

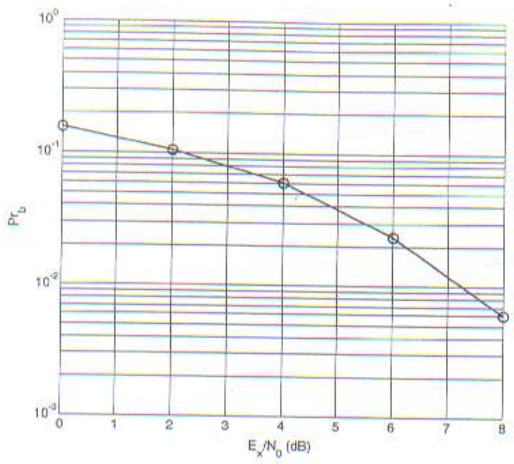

Figure 8–20 Pr_b vs. E_x/N_0 for signal s_ppm1.

In the case of signal `s_ppm2`, we first execute Function 8.5 with `HDSD = 1` and then with `HDSD = 2`. In both cases, the command line is the following:

```
[RXbits2,BER2] = cp0801_PPMreceiver(rx_ppm2, mppm2,...
   fc, bits2, numbit, 3, Ts);
```

Figure 8–21 compares Pr_b for signal `s_ppm1` with Pr_b for signal `s_ppm2`. Regarding `s_ppm2`, both hard decision and soft decision cases are represented. As expected, the probability of error on the bit decreases with increasing redundancy represented by the N_s parameter of the code repetition coder. Moreover, the result in Figure 8–21 confirms that in the presence of Gaussian noise, soft decision performs better than hard decision. Note that for $E_x/N_0 > 6$ dB and $N_s = 3$, the measured Pr_b values for hard and soft decision are not represented since the simulation for the given number of simulated bits returns a 0 value.

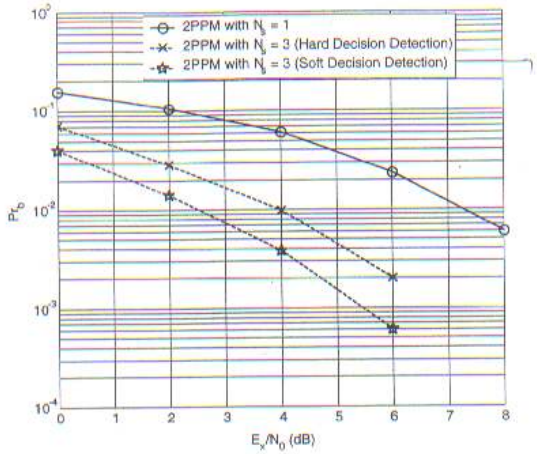

Figure 8–21 Pr_b vs. E_s/N_0 for a 2PPM-TH-UWB signal with one pulse per bit (solid line), three pulses per bit and hard decision detection (crosses), and three pulses per bit and soft decision detection (stars).

The same analysis can be repeated for the PAM-DS case, with the only difference that Functions 8.6 and 8.7 are executed in place of Functions 8.4 and 8.5. Figure 8–22 compares the average Pr_b of 2PAM with 2PPM in the presence of only one pulse per bit, or $N_s = 1$. The result in Figure 8–22 confirms that for the same transmitted signal energy, the use of antipodal signals results in improved performance when using PAM. Simulation results also confirm a gain of 3 dB in transmitted power.

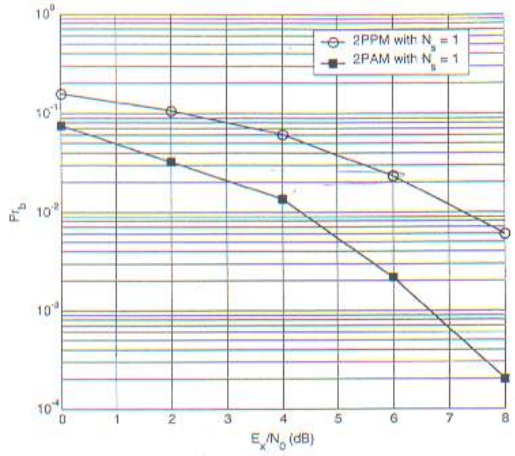

Figure 8–22 Pr_b vs. E_x/N_0 for a 2PPM-TH-UWB signal (white circles), and for a 2PAM-DS-UWB signal (black squares). In both cases, one pulse is transmitted for each bit.

Finally, Figure 8–23 compares the Pr_b versus E_x/N_0 curve for a 2PAM signal with different N_s values and hard versus soft decoding. Once again we observe that Pr_b drastically decreases when increasing redundancy of the code repetition coder, and that soft decision outperforms hard decision.

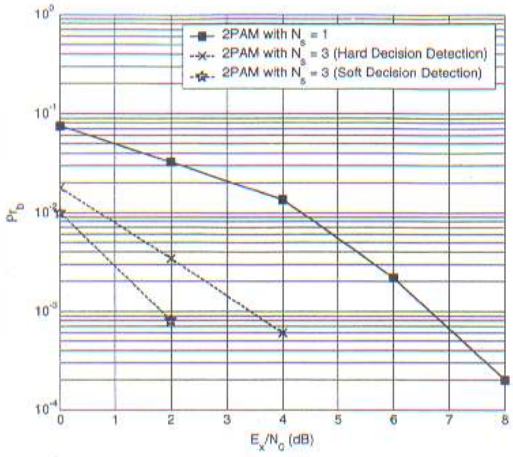

Figure 8–23 Pr_b vs. E_x/N_0 for a 2PAM-DS-UWB signal with one pulse per bit ($N_s = 1$, solid line), three pulses per bit ($N_s = 3$) with hard decision detection (crosses), and three pulses per bit ($N_s = 3$) with soft decision detection (stars).

CHECKPOINT 8–1

8.2 PROPAGATION OVER A MULTI-PATH-AFFECTED UWB RADIO CHANNEL

Propagation over the AWGN channel was analyzed in Section 8.1. As previously discussed, the AWGN channel is characterized by two parameters, channel gain and channel delay, and by AWGN, typically thermal noise, at the receiver. In this case, the structure of the optimum receiver is relatively simple. Basically, the receiver has the task of selecting the one waveform out of M that best matches the received signal.

The presence of multiple paths between transmitter and receiver introduces complexity in both channel model and receiver structure. The channel exhibits time-varying properties that must be taken into account in the channel model. Due to distortion, the received signal often has little resemblance with the transmitted waveform. This is particularly true for indoor transmissions, where propagation is perturbed by a number of