ESD5 – Fall 2024
Workshop
Simulating Communication Systems with Simulink

Department of Electronic Systems
Aalborg University

November 11, 2024

---

1 A quick introduction to Simulink

What is Simulink?: Simulink, developed by MathWorks, is a graphical programming environment for modeling, simulating, and analyzing multi-domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries.

A more intuitive definition: Simulink is a product from MathWorks and is inserted on the Matlab environment, but it has its own "product" family. The main idea behind Simulink is that Engineers often start a project by sketching some block diagram of the desired system to be built, where each block usually has inputs and outputs and performs a specialized function. Based on this view, Simulink equips us with a library of often-used blocks in diverse areas of Engineering, such as communications and control. This way, it is possible to quickly test a system that one wants to build by using such blocks and connecting them in the desired way while minimizing the quantity of coding needed – since each block already implements a pre-designed function. Note that some blocks can also model system phenomena, such as the effects of wireless channels and possible hardware impairments one may face in practice. After testing a system through Simulink, the product provides us with several ways to ease the implementation of the practical system, for example, providing tools to convert the Simulink blocks into other languages such as VHDL, C, Python. You can also create blocks, play with parameters, and change the standard blocks provided by the software. With that and having a notion of how the system should work, an Engineer can go with greater security and awareness for the actual implementation of the system in practice.

Goal: This workshop aims to provide a crash course on how to work with Simulink and use these notions to evaluate the operation of an end-to-end physical layer design for transmitting digital signals, which was introduced mainly in Lectures 11 and 12.

Simulink training: For complete basic training on Simulink, we refer the interested student to `https://se.mathworks.com/learn/tutorials/simulink-onramp.html` (expected 3 hours, it has several examples, including the modeling of physical phenomena – velocity, acceleration, and other differential equations). During the workshop, we will introduce some of the concepts presented in the course provided by MathWorks. Simulink could be a useful tool when developing your Semester Projects (the current and the next ones). Simulink is widely used in the industry; see `https://enlyft.com/tech/products/mathworks`.

AAU provides student licenses: Please see `https://se.mathworks.com/academia/tah-portal/aalborg-universitet-30329468.html` and `https://www.ekstranet.its.aau.dk/software/mathworks/#387051`. Remember to install Simulink when running the installer.

Other requirements: When installing Matlab and Simulink remember to check the *Digital Signal Processing (DSP) Toolboxes* and *Communications Toolbox*. See the figures below. If possible, check all the boxes so we do not have any dependency problems. PS: Note the presence of an antenna toolbox and 5G – the latter related to the next generation of cellular networks (the one you use in our phones).

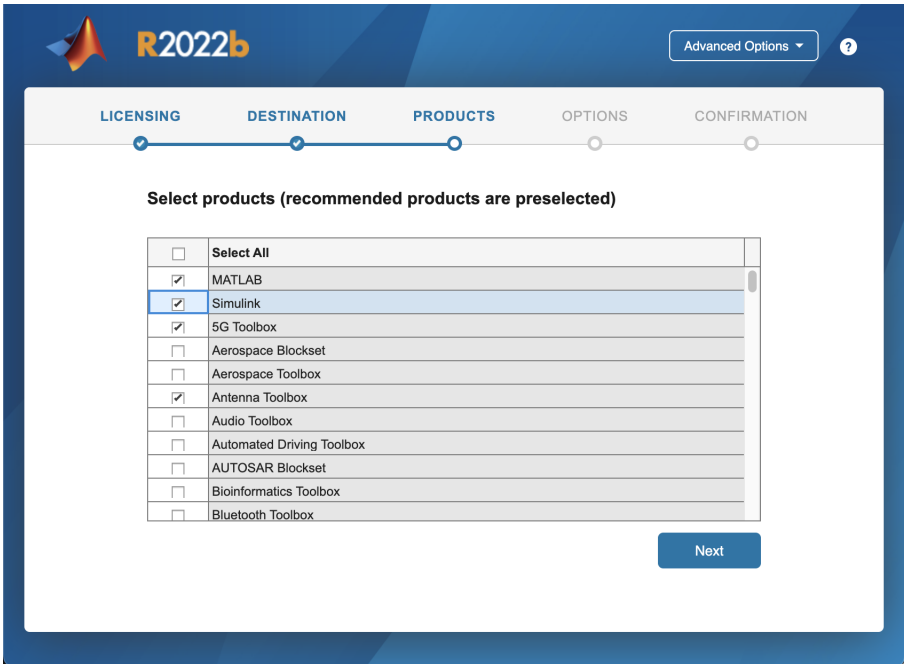

Figure 1: MathWorks installer (1).

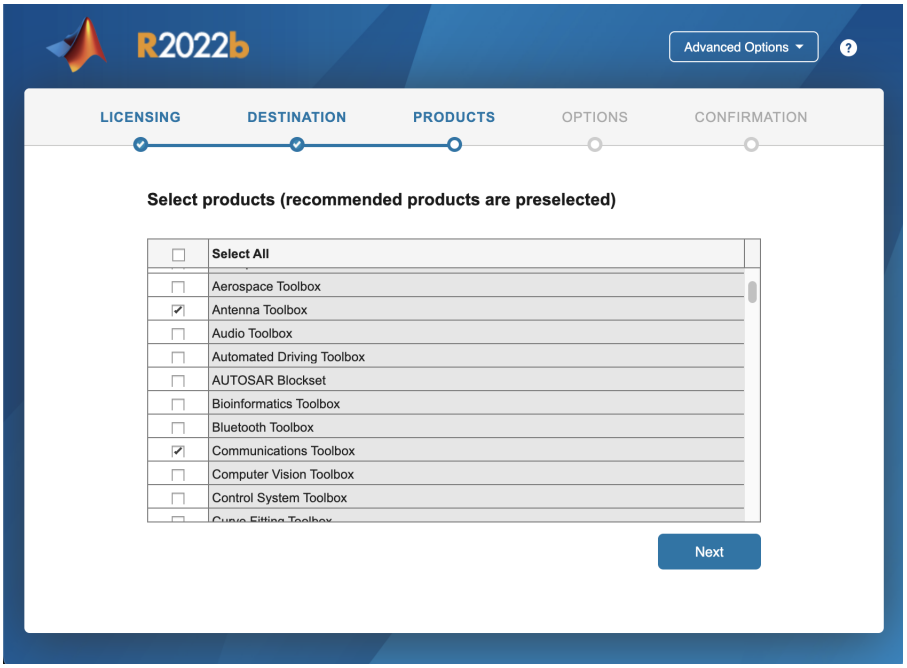

Figure 2: MathWorks installer (2).

2 A basic Simulink project

To start using Simulink, we will implement a simple finite impulse response (FIR) filter. Consequently, we focus on discrete-time systems,[1] where we evaluate the dynamics of the system of interest at specific time points. For example, for a first-order system, its difference equation can be given as

$$x[n] = x[n-1] + r[n]\Delta t, \tag{1}$$

where the current state of the system is $x[n]$, the previous state is $x[n-1]$, $r[n]$ is the first difference between states that can depend on some input at time t or, in its simplest version, can be a constant, and Δt is the time step. The FIR example will help us navigate the Simulink environment and learn some of its basic blocks. These basic blocks will be useful in the next section, where you will receive a digital communication scheme already sketched but with the need to adjust some parameters and visualize what is happening along the way.

A causal discrete-time FIR: Let's start by recapping the definition of a causal discrete-time FIR of order N, which can be written as [1]:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots + b_N x[n-N] \tag{2}$$

$$= \sum_{i=0}^{N} b_i x[n-i], \tag{3}$$

where $x[n]$ is the input signal, $y[n]$ is the output signal, N is the order of the filter, and b_i is the value of the impulse response at the i-th instant for $i \in \{1, \ldots, N\}$. The above computation is also known as a *discrete convolution*, where the terms $x[n-i]$ are called *taps*. One way to visualize the expression above is through the following block diagram:

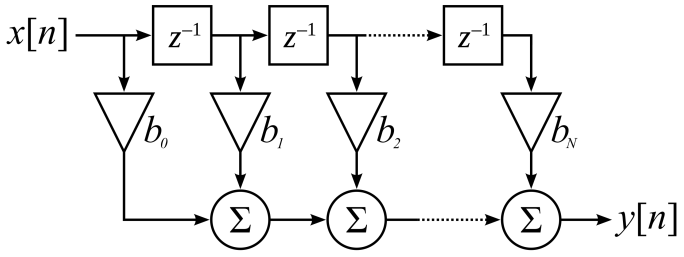

Figure 3: Causal, discrete-time FIR filter of order N. Source: `https://en.wikipedia.org/wiki/Finite_impulse_response#/media/File:FIR_Filter.svg`.

[1] Simulink can also model continuous-time systems ($\Delta t \to 0$) by using differential equations. For example, consider the case $\frac{dx}{dt} = r(t)$. When Simulink is asked to evaluate, say, the current state at a continuous time t_2, that is, $x(t_2)$, it integrates the stored relationship $\frac{dx}{dt} = r(t)$ as $x(t_2) = \int_{t_1}^{t_2} r(t)dt$.

In the figure, each inverse Z-transform inserts a delay to the input signal and outputs a tap $x[n-i]$. This structure is known as the *tapped delay line*. When the input signal is an impulse, the impulse response is given by

$$h[n] = \sum_{i=0}^{N} b_i \delta[n-i] = \begin{cases} b_i, & i \in \{1, \ldots, N\} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Moving average: The most simple, useful FIR filter is the moving average, which is defined by following the filter coefficients:

$$b_i = \frac{1}{N+1}. \tag{5}$$

Then, an order $N = 1$ moving average is:

$$y[n] = \frac{1}{2}x[n] + \frac{1}{2}x[n-1]. \tag{6}$$

In other words, the first-order moving average takes the average of the current state and the previous one, which is helpful when dealing with higher-frequency noise, as shown in the example below.

Implementing a moving-average filter on Simulink: First, we must understand how to implement a simple discrete system on Simulink. Then, we will work together to build and analyze the moving average filter. The analysis and the tools that we are going to make here are going to be crucial in the next section.

The figures below indicate creating a Simulink model. Different templates are available, but we will begin with a simple, blank template. Moreover, creating a black model also indicates the *library browser*, where we can find the massive amount of blocks pre-designed by Simulink. In the meantime, we are going to focus on the basic blocks available in the Simulink folder (in blue).

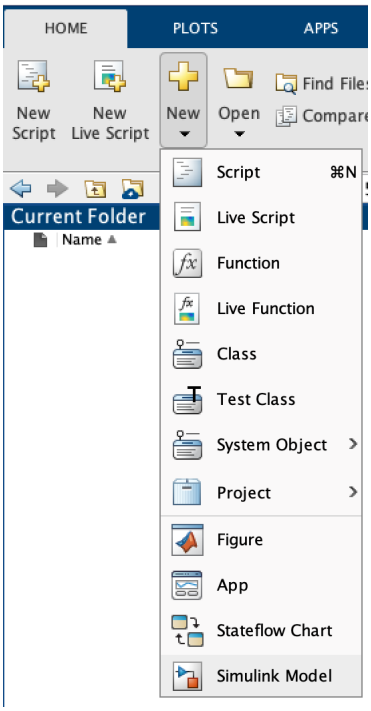

Figure 4: Creating blank Simulink model.

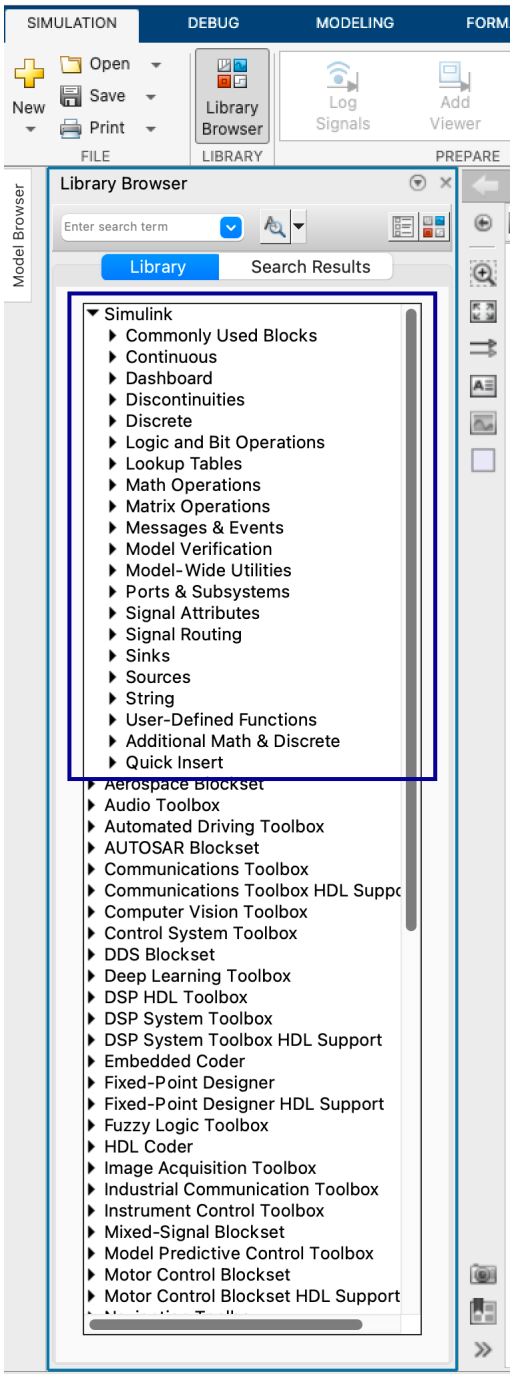

Figure 5: Library of blocks with focus on the basic ones in blue. You can go around and familiarize yourself with them. The names are very suggestive, and to put a block on the drawing background, you need to drag and release. Double-click on blocks to find their parameters. You will notice that the UX design is intuitive most of the time.

We are going to build the following diagram.

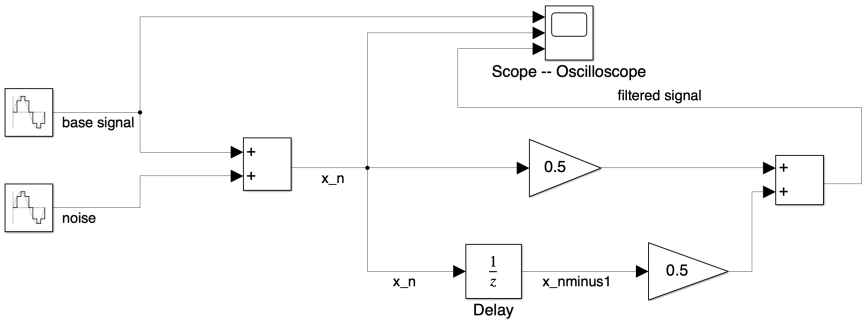

Figure 6: First-order moving average filter applied over a noisy sine.

Steps to build the diagram:

1) Introduce two Sine sources: base signal and noise. You can find it on SOURCES. Double-click on them to change their parameters and configure them as:

   - Base signal: Amplitude = 10, Frequency = 1 rad/sec, and Sample Time = 0.2. Sample time defines Δt in (1), automatically discretizing the Sine function for us. This is the parameter that makes our system discrete instead of continuous.
   - Noise signal: Amplitude = 2.5, Frequency = 18 rad/sec, and Sample Time = 0.2.

2) Add the two signals using an Add block, which can be found on MATH OPERATIONS.

3) Now, it is time to construct the first-order moving-average filter. We add a Delay Block available on DISCRETE, which implements a tap $x[n-1]$. Then, add 2 Gain Blocks, available on MATH OPERATIONS. Link the signals according to the figure and add them using an Add block (MATH OPERATIONS).

4) So, we have implemented the moving average filter with an input consisting of a discretized noisy sine. But we cannot see what is happening. How do we look at what is happening? Simulink implements several display tools to evaluate our system. One of the coolest is the Scope, found on SINK. Scope acts as a **virtual oscilloscope**. Fig. 7 shows what is expected to be observed on the Scope.

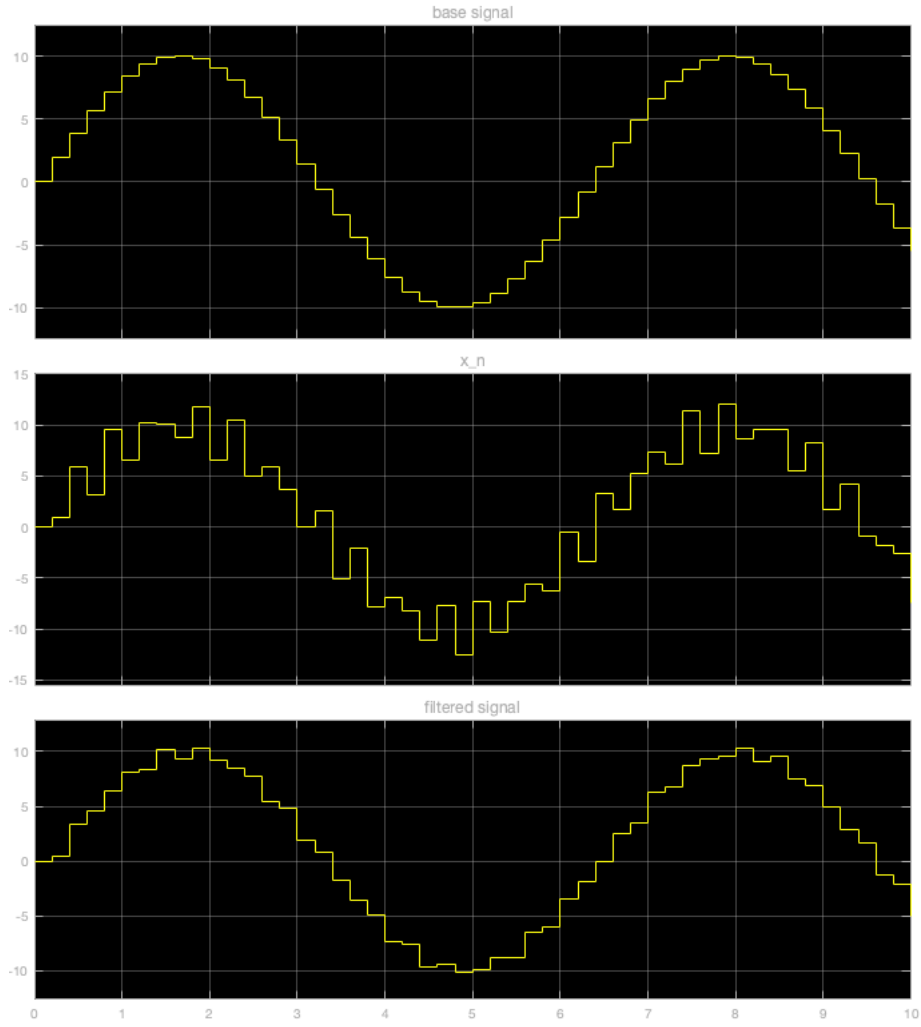

Figure 7: Expected output result of the first-order moving average filter observed from the Scope with a simulation time (Stop Time) of 10.

Exercises

(a) Play with the parameters of the noise signal. What happens if the noise signal's amplitude and frequency are close to the base signal? What about the contrary?

(b) Back to the original parameters, compare the performance of a first-order moving average and a second-order moving average.

3 The physical layer of a digital communication system

Now that you are a little bit more familiar with Simulink, we will analyze a binary PAM communication system, that is, with modulation order $M = 2$. We give the system implemented to you, but you will carry out the analysis. We propose implementing a simpler version of Problem 3 from Problem Set 7 (Lecture 12). It is simpler because we will assume binary PAM, so the symbols transmitted are "`0`" and "`1`". We aim to simulate the physical layer of a 2-PAM digital communication system. We introduce the mathematical notation of the 2-PAM below.

Consider a 2-PAM modulation. Let T_b be the period of one bit and $R_b = \frac{1}{T_b}$ the bit rate. For $m \in \{0, 1\}$, the signal waveform for the m-th bit can be written as:

$$s_m(t) = A_m g_T(t) \tag{7}$$

where A_m is the amplitude of the m-th symbol and $g_T(t)$ is a the pulse shape. We use an antipodal amplitude modulation, which assigns the amplitude $A_0 = -1$ to the bit "`0`"; conversely, the amplitude $A_1 = 1$ is assigned to bit "`1`". We assume a cosine pulse shape:

$$g_T(t) = \cos(2\pi R_b t). \tag{8}$$

Then, the transmitted signal is:

$$u_m(t) = s_m(t)\cos(2\pi f_c t), \tag{9}$$

where $f_c = \frac{1}{T_c}$ is the carrier frequency. As shown in the solutions for Problem Set 7, the bandpass basis function for the 2-PAM is:

$$\psi(t) = \sqrt{\frac{2}{\mathcal{E}_g}} g_T(t)\cos(2\pi f_c t), \tag{10}$$

where $\mathcal{E}_g$ is the energy of $g_T(t)$. The energy of $g_T(t)$ can be find as:

$$\mathcal{E}_g = \int_0^{T_b} g_T^2(t)dt = \int_0^{T_b} \cos^2\left(\frac{2\pi}{T_b}t\right) = \frac{T_b}{2}. \tag{11}$$

The transmitted signal goes through an additive white Gaussian noise channel. The receiver receives:

$$r_m(t) = u_m(t) + n(t), \tag{12}$$

where $n(t)$ is the noise. We define the signal-to-noise ratio (SNR) as how much the noise power is less than the power of the desired signal on average. We assume perfect carrier-phase recovery, which means the receiver knows the carrier signal and can synchronize perfectly with the transmitter (see Fig. 8). Then, by cross-correlating the received signal with the bandpass basis function and integrating it, we have:

$$\int_0^T r(t)\psi(t)dt = A_m\sqrt{\frac{E_g}{2}} + n = A_m\sqrt{\frac{T_b}{4}} + n, \tag{13}$$

where n is dependent on the noise realization at a certain time. Finally, the detector assigns "`0`" if the sampled signal from the integral is negative and "`1`" otherwise.

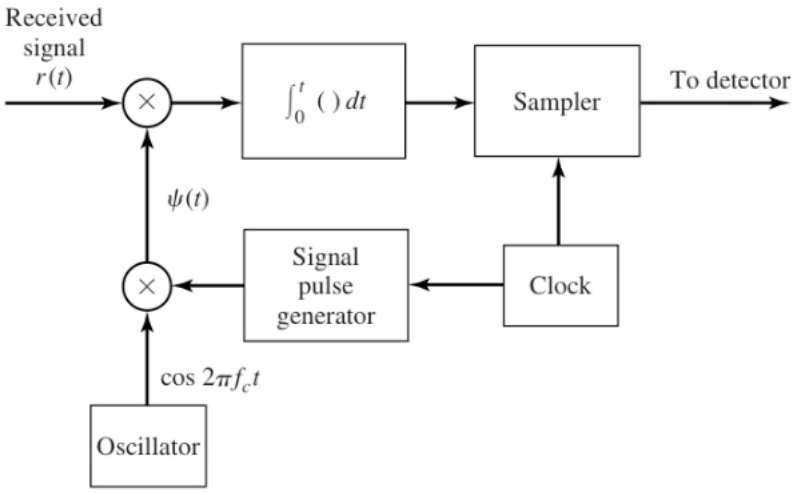

Figure 8: Demodulation of bandpass digital PAM signal from [2].

Our goal today is to implement the system described above when assuming a Bernoulli source generator, as shown in the figure below.

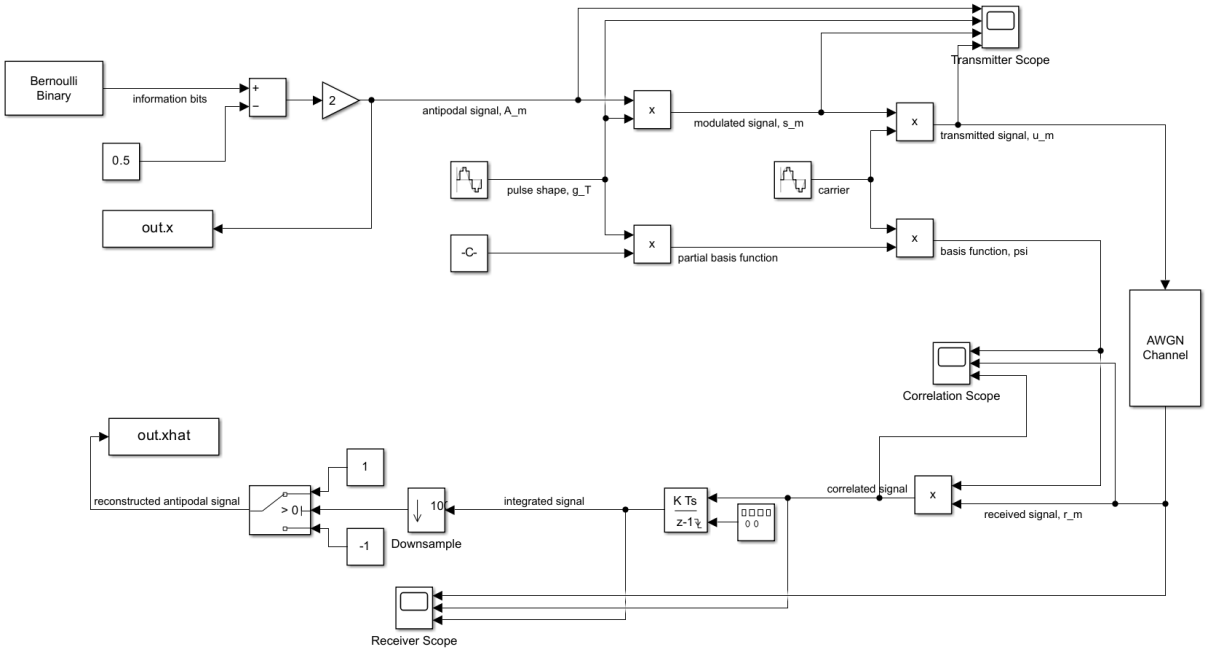

Figure 9: Implemented system.

Exercises

You received the file with the system as above but without the Scopes. The system is configured as follows: T_b = bit_interval = 0.01 s, T_c = carrier_interval = 0.001 s, and ΔT = sampling_interval = 0.0001 s. Please consider the following:

(a) Add the scopes and try to interpret/analyze the provided system by relating each signal and step with the mathematical description introduced above. Set STOP TIME = 0.1 s, meaning that you will analyze 10 bits. What is the current SNR?

(b) The OUT.X and OUT.XHAT outputs the input and output sequences to the Matlab Workspace so that you can work with the signals as vectors. Set STOP TIME = 1 s. To compute the Probability of Error, P_e, for a given SNR, use the following formula written in Matlab code:

$$P_e = 1 - \frac{\text{sum}(\text{out.x}(1:\text{end}-1) == \text{out.xhat}(2:\text{end}))}{\text{length}(\text{out.x}(1:\text{end}-1))}. \tag{14}$$

Change the SNR to the following values: -20, -15, -10, -5, 0, 5, 10, 15, and 20 dB. Run the model with STOP TIME = 1 s at each change, compute P_e as above, and store its

value in a vector. Finally, plot a figure where P_e is the y-axis, and the SNR value is in the x-axis. This probability of error plot is a very common way to analyze receivers and modulations. What trend did you notice?

(c) Instead of generating random bits, there is a way to put a signal of your preference by using a block available on SOURCE, for example, an image. Try to do so, and after the communication process, you can reconstruct the message. Change the SNR values to see how the communication affects the recovered image quality. OBS: To simplify, search for an image of black and white pixels.

References

[1] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals & Systems (2nd Ed.)* USA: Prentice-Hall, Inc., 1996. ISBN: 0138147574.

[2] John G. Proakis and Masoud Salehi. *Communication Systems Engineering*. Second. Upper Saddle River, NJ, USA: Prentice-Hall, 2001. ISBN: 0130617938.