Lab 1: Block Filtering

Overlap-Add and Overlap-Save Algorithms

Introduction and Background

For this lab, you will need to understand the theory behind the Overlap-Add and Overlap-Save algorithms in order to get them successfully up and running. The functions you will implement already do exist in Simulink; otherwise you wouldn't have the real-time filtering capabilities you've been enjoying so far! However, for this lab you must implement them from scratch to learn all the nuances.

In a real-time signal processing scenario, an input signal is constantly fed into the DSP to be processed, and the associated output is made ready without "much delay" (otherwise it wouldn't be called "real-time"). So if we had a long input signal that continues in time indefinitely, we don't want to wait until the entire input signal has terminated before filtering it; this would be equivalent to doing something "off-line" after all the input is available. Instead, we would like to filter the blocks of input as they come in. Furthermore, we would like to filter these blocks quickly, which means the use of the Fast Fourier Transform (FFT) algorithm is necessary to implement the associated processes in the frequency domain. There are two methods for segmenting a long (possibly infinitely long) input signal into shorter blocks, and processing them quickly using the FFT. They are called the Overlap-Add method and the Overlap-Save method. The former method is easiest to explain.

Overlap-Add

The Overlap-Add method is based on the observation that when we consider two discrete-time signals, say $x_k(n)$ and $h(n)$, with support L and support M, respectively (note: the support is the length of the smallest consecutive stretch of points that contains all non-zero signal elements), the resulting convolution $y_k(n) = x_k(n) * h(n)$, has a support of L+M-1. For example, say the support for $x_k(n)$ is $n = 0, \ldots, L\text{-}1$ and the support for $h(n)$ is $n = 0, \ldots, M\text{-}1$, then the support for $y_k(n)$ is at $n = 0, \ldots, L+M\text{-}2$. The questions at the end of this lab will elucidate this concept.

Using this idea suppose our input stream $x(n)$ is an infinite sequence starting at time $n = 0$. Divide $x(n)$ into L-length blocks and convolve each L-block with $h(n)$ (using *linear* convolution). Then sum all the convolution outcomes along the L-boundaries (we elaborate more soon). This works because of the additive property of convolution which states $(x_1(n) + x_2(n)) * h(n) = x_1(n) * h(n) + x_2(n) * h(n)$, and is visually depicted in Figure 1.

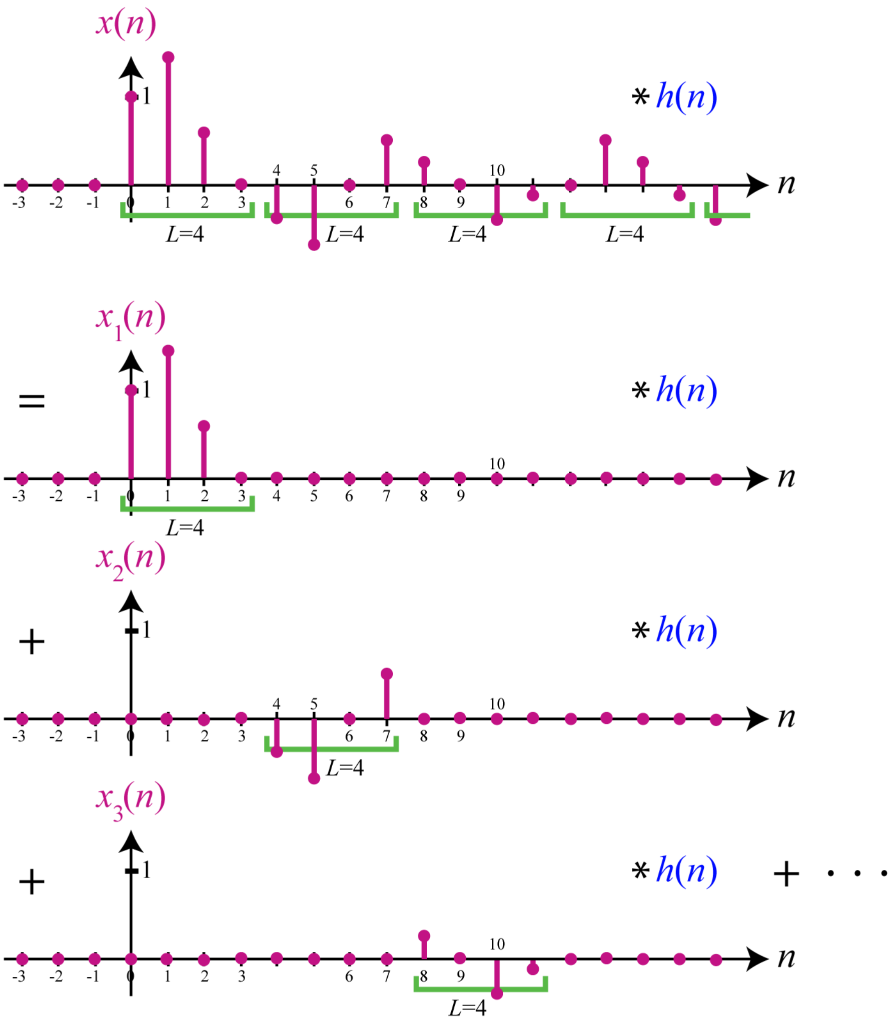

Figure 1: Basic Idea Behind Overlap-Add Method. The kth L-block of $x(n)$ is denoted $x_k(n)$.

In Figure 1, the operation of convolving a very long $x(n)$ with $h(n)$ is equivalent to the operation of convolving each L-block of $x(n)$, denoted $x_k(n)$ for the kth block, with $h(n)$ and then conducting addition judiciously to deal with the "tail" region from each block convolution as we discuss next. An important aspect is that *after* convolving each block with $h(n)$, the resulting intermediate signal is L+M-1 samples in length as discussed earlier, and thus the extra M-1 samples at the end due to convolution expanding the support (called the "tail") must be added to the first M-1 samples of the *next* convolved block. This is illustrated in Figure 2, where the right hand side (RHS) of the equality shows the result (graphically) after convolution. Specifically, the kth output block is given by: $y_k(n) = x_k(n) * h(n)$ and the last M-1 samples of $y_k(n)$ must be added to the first M-1 samples of $y_{k+1}(n)$ to produce the appropriate output signal $y(n) = x(n) * h(n)$.

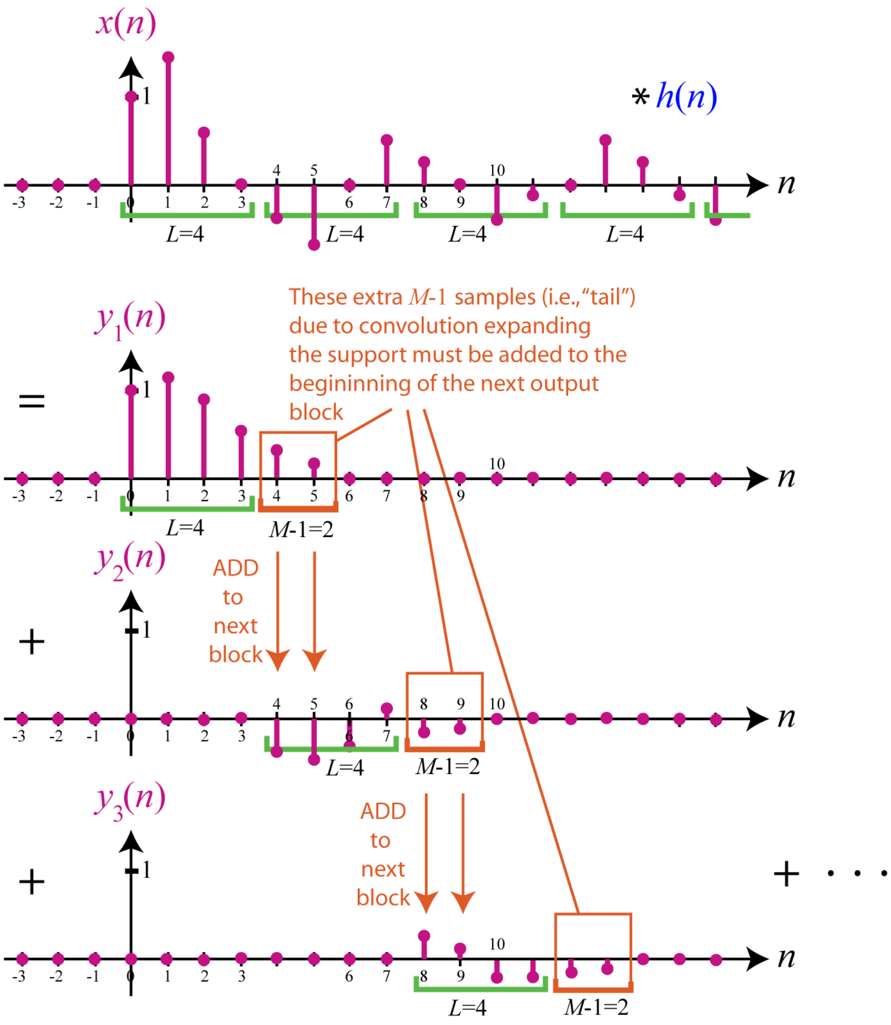

Figure 2: "Tail" Resulting from Convolution must be Added to Next Convolved Block; Note: $y_k(n) = x_k(n) * h(n)$.

One main challenge in a real-time processing scenario is that the timing of completing a block convolution needs to be appropriately synchronized with the overall output speed so that that tail region may be added to the next block at the right time. If the process of convolving each block is slower than outputting the samples of blocks already convolved, then the tail region will not have the opportunity to be added to the next block (because the next block is still in the process of going through convolution), resulting in the erroneous output of the samples 0 to M-2 and L to L+M-2 for each block. One way to deal with this is to slow down the rate of output, which may fail to meet the timing requirements for a given application. Another, more attractive approach, is to speed up the process of convolution. This can be achieved through the use of the FFT for convolution. A reminder of how to use the FFT algorithm to filter a block of input to perform convolution is summarized here (note: this is not the entire Overlap-Add algorithm as the output blocks must be combined at the end of this):

1. Zero-pad the filter $h(n)$ with L-1 zeros to make it length (L+M-1).
2. Compute the (L+M-1)-FFT of the zero-padded $h(n)$ result of Step 1 and save.

3. Zero-pad each L-block input segment $x_k(n)$ with M-1 zeros it make it length (L+M-1).
4. Compute the (L+M-1)-FFT of the zero-padded result of Step 3 and save.
5. Multiply sample-by-sample the two FFT results from Steps 2 and 4.
6. Take the inverse (L+M-1)-FFT (IFFT) of the resulting product from Step 5 to produce $y_k(n)$.

After each block $y_k(n)$ is computed as above, it is added to the next block as shown in Figure 2 and more simply in Figure 3. The throughput is L samples per block processed.

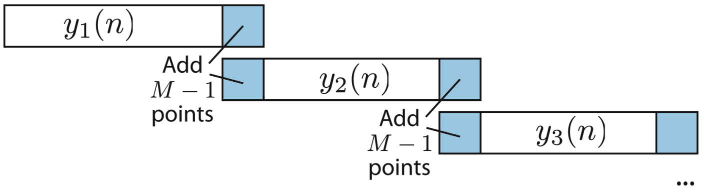

Figure 3: Illustration of Combining filtered output blocks for Overlap-Add Method.

Overlap-Save

The Overlap-Save method is a bit more difficult to explain than the Overlap-Add method as it is based, in part, on the concept of *circular* convolution which in this context results in time-domain aliasing. We describe an Overlap-Save method with the same throughput of L samples per block processed as discussed for Overlap-Add.

Textbooks often give complicated formulas when explaining circular convolution. Here is an easier way to consider circular convolution. Let's consider $x_k(n)$ and $h(n)$ with support regions $n = 0, \ldots, N$-1 and $n = 0, \ldots, M$-1, respectively; note the difference from the previous section: the support length for $x_k(n)$ is now N for Overlap-Save, but it was L for Overlap-Add. Let $y_{L,k}(n)$ be the result of normal convolution (often called *linear* convolution) of $x_k(n)$ and $h(n)$. Then the N-circular convolution of $x_k(n)$ and $h(n)$ can be described in terms of $y_{L,k}(n)$ via the diagram in Figure 4 for $N = 4$ and $M = 3$. Consider two stages. The first stage takes a *periodic extension* of the linear convolution result $y_{L,k}(n)$ where the period used for the extension is N. The second stage *windows* the result such that we consider only N points at $n = 0, \ldots, N$-1 leaving all other points with zero-values. The periodic extension is created by adding all shifted versions (where the shifts are restricted to be integer multiples of N) of the linear convolution result $y_{L,k}(n)$. Because $x_k(n)$ is of length N and $h(n)$ of length M, the length of $y_{L,k}(n)$ will be N+M-1. Thus, taking a periodic extension of $y_{L,k}(n)$ with period N will result in overlap. Since the overall goals is to conduct a linear convolution (but using the FFT), this overlap is essentially "corruption" in the beginning of the windowed sequence as shown in Figure 4. In general the points at $n = 0, 1, 2, \ldots, M$-2 will be corrupt from what we call time-domain "self-aliasing". However, the remaining points at $n = M$-1, M-2, M-3, …, N-1 will still be equal to the desired $y_{L,k}(n)$.

The idea in the Overlap-Save method is to exploit the process of N-circular convolution, which can more efficiently be implemented via an FFT in contrast to normal linear convolution. Since the output from linear convolution is N+M-1 in length, the N-circular convolution will corrupt the first M-1 samples, leaving the last N-M+1 samples of the circular convolution result pristine (we do not worry about a "tail" since for circular convolution the window only takes N points effectively discarding the last

M-1 samples from a normal linear convolution). Since the first *M*-1 samples are always corrupted and undesirable, the Overlap-Save method selects *N*-blocks that overlap by *M*-1 points such that the previous block output can provide the information of the corrupted points in the current block output. For the very first block (which has no “previous” block to compensate for corruption), the first *M*-1 samples are set to 0 effectively shifting the input to the right by *M* and avoiding self-aliasing in that case.

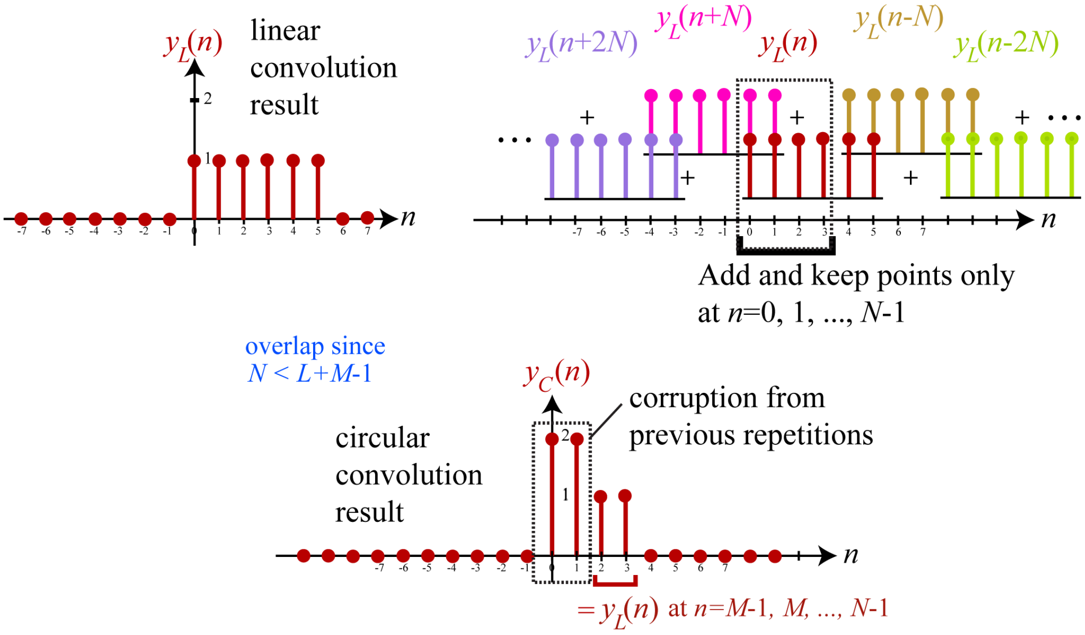

Figure 4: Illustration of Circular Convolution in Contrast to Normal Convolution. Please note for simplicity, the block subscript *k* used in the body of the text is dropped in this diagram.

Finally after each output block is generated it is combined into the convolved output stream *y*(*n*) as illustrated in Figure 5. Figure 5 demonstrates the relationship between input blocks and output blocks for the Overlap-Save method. The *N*-length blocks need to overlap effectively by *M*-1 points in order to compensate for the corruption due to using the FFT for convolution (resulting in corruption in the circularly convolved output block).

The following algorithm will further help you implement the Overlap-Save method. We deal with input blocks of length *N*. To have an equivalent throughput to the Overlap-Add method (which helps for comparison reasons), we let *N*=*L*+*M*-1 where *L* is the same parameter as specified in the Overlap-Add method and *M* is the length of *h*(*n*) as for both the Overlap-Add and Overlap-Save methods:

1. Zero pad *h*(*n*) with *L*-1 zeros such that it is of length *N* and has support *n*=0, 1, …, *N*-1.
2. Take the *N*-FFT of *h*(*n*) and save for repeated access in the future.
3. If the current block being processed is the *first* block, then the first *M*-1 samples are zeros; fill the remaining *L* samples with the signal *x*(*n*), starting at *n*=0. *Save* the last *M*-1 samples of this block for processing of the next block.
   If the current block being processed is *not* the first block, then take the last *M*-1 samples from the previous block (previously *save*d) as the first *M*-1 samples of the current block (this creates an overlap), and fill the remaining *L* samples with the next new samples of *x*(*n*). *Save* the last *M*-1 samples of this block for processing of the next block.

Figure 5 illustrates this overlap process.

4. Take the *N*-FFT of each *N*-length input block $x_k(n)$ and save.
5. Multiply sample-by-sample the FFT result from Step 2 with the FFT result from Step 4.
6. Take the inverse *N*-FFT of the result from Step 5.
7. Discard the first *M*-1 samples from Step 5 that are corrupted, keeping only the last *L* samples.
8. Concatenate the kept *L* samples of each block to produce the overall output as illustrated in Figure 5.

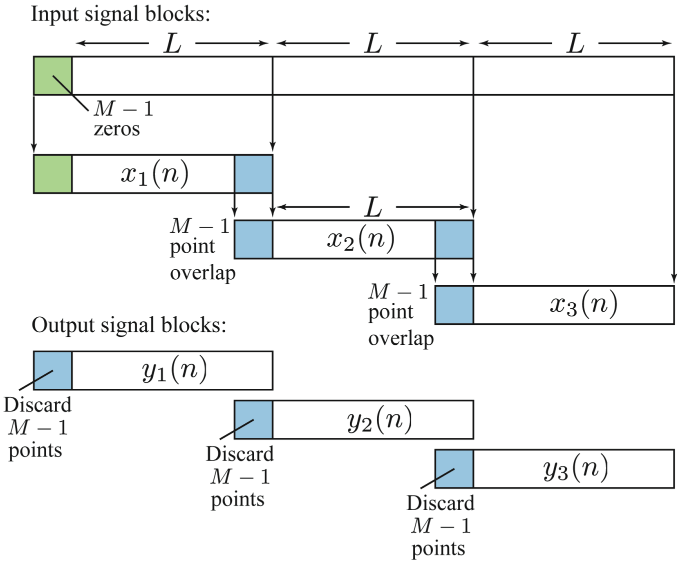

Figure 5: Illustration of Breaking up Input Blocks and Combining Filtered Output Blocks for Overlap-Save Method.

Notice that for the Overlap-Save method, no zero-padding is required for any of the input blocks (but it may be required one-time for the FFT of *h*(*n*)), and no addition operation is required either as in the Overlap-Add method. Depending on the relative values of the parameters *L*, *M* and *N*, one method may be more computationally and/or memory efficient than the other.

Design and Implementation

Overlap-Save Algorithm

First, you will construct a block diagram for an Overlap-Save algorithm using elementary Simulink blocks. For this model, we will walk you through it step by step. The model will implement an FIR filter kernel of length *M* = 113. The algorithm will use an FFT and inverse FFT of length *N* = *L* + *M* -1 = 512. Thus, the input blocks will be of length *N* = 512 and the throughput will be *L* = 400 output samples per processed block.

Designing the Filter Kernel

1. Before the Simulink model can be built, the FIR filter needs to be designed. Go to the command window and type fdatool to bring up MATLAB’s Filter Design & Analysis Tool.

2. Select Lowpass under Response Type.
3. Choose an Equiripple FIR under Design Method.
4. Specify the filter order as 112 (this will result in a kernel of length $M = 113$).
5. Under Frequency Specifications, set Units to Hz, Fs to 8000, Fpass to 400, and Fstop to 800.
6. Click Design Filter. The Magnitude response of the filter is displayed.
7. Go to File → Export. Choose Export To Workspace and Export As Coefficients. Under Variable names, name the Numerator h. Press Export. This exports the filter coefficients to the MATLAB Workspace as a 1×113 vector named h. You can verify the presence of this variable by going to the command window and typing "whos".

Building the Simulink Model

1. In a new Simulink model, set the Amplitude of a Sine Wave Source to 1 and the Frequency to 100 Hz. Also, set the Sample time to 1/8000 (this will imitate a sampling rate on the C6437 board).
2. Connect your input to a Buffer block that you can find in Signal Processing Blockset → Signal Management → Buffers. Set the Output buffer size to 400. The buffer divides the input signal into data block segments of length L. The output of the buffer is a frame-based signal (as opposed to a sample-based signal) such that each segment (or "frame") of 400 samples is processed as one chunk, as required by the Overlap-Save process.
3. Add a Delay Line block to the diagram from Signal Processing Blockset → Signal Management → Buffers. Set the Delay line size to 400 and connect the output of the Buffer to the Delay Line input. Effectively, the Delay Line delays its input by one data block ("frame") of length L.
4. The Overlap-Save algorithm calls for the last M-1 points from the previous data block to be saved and appended to the beginning of the next data block. The Delay Line inserted in step 3 above allows us to access the previous data block. In order to extract the necessary M-1 points, insert a Submatrix block from Signal Processing Blockset → Signal Management →Indexing that you connect to the output of the Delay line. Set the Row span to Range of rows, the Starting row to Index, the Starting row index to 289, the Ending row to Last, and the Column span to All columns. Here's what this block does: the data blocks outputted from the Delay Line are 400×1 column vectors; we want the last M-1 points. The Submatrix block selects elements 289 through 400 of these input vectors and outputs 112×1 column vectors.
5. The next step is to take the M-1 saved points from step 4 and append them to the beginning of the current data block. To do this, we can use a matrix concatenate block from Simulink→ Math Operations. Insert this block into the model and set Number of inputs to 2, Mode to Multidimensional Array, and Concatenate Dimension to 1. Connect the output of the Submatrix block from step 4 to the first (top) input of the Matrix concatenate block, and connect the output of the Buffer block from step 2 to the second (bottom) input of the Matrix concatenate block. These connections cause the 112×1 vectors from the Submatrix block (the M-1 saved data points) and the 400×1 vectors from the Buffer (the current data block) to be combined into 512×1 vectors that are suitable for FFT calculation.
6. Add an FFT block to the mode. Connect the output of the Matrix Concatenate block to the input of the FFT block. This will compute the 512-point FFT of the overlapped data blocks. Notice that $N = L+M-1 = 512$ is chosen to be a power of two; this is necessary because Simulink's FFT block uses a radix-2 FFT algorithm.

7. The next step in the Overlap-Save algorithm is to multiply the FFT computed in step 7 by the FFT of the filter kernel. Before we can do this, we need to import the filter coefficients into the Simulink model. Add a From Workspace block to the model from Simulink → Sources. Set Data to the name of the filter kernel you exported to the MATLAB Workspace and Sample Time to 400/8000. Note that this sample time causes the filter coefficients to be read at the same rate that data blocks are outputted from the Buffer block of step 3.
8. In order to compute the FFT of the filter kernel, we need to extend it so it has a length of 512. To do this, add a Pad block from Signal Processing Blockset → Signal Operations. Set Pad over to Columns, Pad value to 0, and Column size to 512. Connect the output of the From Workspace block in step 7 to the input of the Pad block. The Pad block simply appends enough zeros to the end of the filter kernel to make it a 512×1 vector.
9. Add another FFT block to the diagram and connect the output of the Pad block from the previous step to the input of this FFT block. Clearly, this just computes the 512-point FFT of the filter kernel.
10. Now we are ready to perform the frequency multiplication necessary for FFT convolution. Connect the Output of the two FFT blocks to the two inputs of a Product block.
11. Once frequency multiplication has occurred, the inverse FFT needs to be computed. Insert an IFFT block from Signal Processing Blockset → Transforms. Under the IFFT block parameters, select the check box labeled "Input is conjugate symmetric." This tells Simulink that the output should be real-valued; that is, any small imaginary parts in the output due to rounding errors will be ignored. Connect the output of the Product block to the input of the IFFT block.
12. The last major step in the Overlap-Save algorithm is to discard all of the points that have aliasing. Namely, the first M-1 points of the data blocks resulting from the inverse FFT operation need to be thrown out. To do this, insert another Submatrix block into the model. Set Row span to Range of rows, Starting row to Index, Starting row index to 113, Ending row to Last, Column span to All columns. Connect the output of the IFFT block to the input of the Submatrix block. Note that by discarding M-1 points, the data blocks are reduced in size back to 400 points, the size of the original data blocks from the input signal.
13. Add an Unbuffer block from Signal Processing Blockset → Signal Management → Buffers and connect its input to the output of the Submatrix block from step 12. As its name suggests, the Unbuffer block takes the frame-based signal of 400×1 vectors from the Submatrix block and converts it into a sample-based signal (the original format of the input).
14. To view the filtered signal, add a Scope block to the model from Simulink → Sinks and connect its input to the output of the buffer block.
15. The Overlap-Save filter is now complete. Save your model.

Overlap-Add Algorithm

Now that you have walked through the Overlap-Save algorithm step-by-step, it is time to try designing an algorithm for yourself. Your task in this section is to design from scratch a filter that uses the Overlap-Add algorithm. You should be able to do this using the same type of blocks (Buffer, Delay Line, Submatrix, etc.) that were used in the Overlap-Save filter. The guidelines are as follows:

1. Use the same lowpass filter as was used for the Overlap-Save algorithm. However, change the order of the filter to 500 such that the filter kernel has a length of M = 501. Also, change the Fstop parameter from 800 Hz to 500 Hz. Essentially, we are using a longer, more expensive filter kernel to produce a lowpass frequency response that drops of much more sharply than the previous filter. Even with a filter order of 500, FFT convolution can filter the output in a reasonable amount of time.
2. Divide the input to the filter into data blocks of length L = 1548. Note that this makes the size of your FFT and inverse FFT calculations $N = L+M-1 = 2048$.

Here are some hints on how to proceed with your design:

1. It is probably best to use Pad blocks from Signal Processing Blockset → Signal Operations to append zeros to the end of data blocks.
2. When using the Delay Line block, make sure that the block parameter named Delay line size is set to be the same size as the input vector to the block. For example, if you input 50×1 data blocks into the Delay Line and want a delay of one data block, set the Delay line size to 50.
3. If you use a From Workspace block to import the filter coefficients to the Simulink model in a similar way to the Overlap-Save algorithm, make sure to set the sample time to 1548/8000. This will match the rate at which the length L data blocks are received by the filter at its input.

Perform the same tests as for the Overlap-Save algorithm. Note and explain any obtained result.