# Lab 8: Overlap-Save Method for filtering using FFT

EE 352 DSP Laboratory (Spring 2019)

**Lab Goals**

To understand implementation of real-time block processing on DSP. The following tasks will be carried out here,

- Understand the DMA operation in the DSP.
- Implement the real time linear convolution using overlap-save method.

## 1 Introduction

In previous lab experiments, the processing of data was done on a sample-by-sample basis. But in signal processing, we encounter various applications which require you to process the data in blocks. Naturally this requires buffering the data. Though in the former case also, we required a buffer, the processing happened on a sample by sample basis. But in block based filtering, entire buffer is filled up first and then the processing is done on the entire block at once. This brings up an another issue, what happens to the data that is continuously streaming in when the buffer is being processed? This could lead to loss of data if the processing of the buffer is not complete before the arrival of next data sample. To avoid this data loss, we use two buffers. One buffer (buffer 'ping') gets filled up while the data in the other buffer (buffer 'pong') is being processed. Once the processing of buffer 'pong' is complete, we process the data stored in buffer 'ping' while storing the next incoming data in buffer 'pong'. This process continues indefinitely.

Direct memory access (DMA) is a feature that allows us to perform the above operation. The DMA controller is used to move data among internal memory, external memory, and peripherals without intervention from the CPU and in the background of CPU operation. The CPU has to just initiate the DMA controller and can then continue with its job of processing. The C5515/35 DSP includes four DMA controllers with four DMA channels each for a total of 16 DMA channels. We now briefly describe, with the help of an example, Ping-Pong buffering using DMA.

## 2 Ping-Pong buffering using DMA

You are provided a `DMA_checkup` project along with this handout which illustrates how DMA is implemented on the DSP. This example uses *Direct Memory Access* (DMA) feature for efficient transfer of data between memory and the codec. We use only one DMA controller (DMA0) for this example. Also, each DMA has 4 channels, out of which channels 0 and 1 are used as input channels and 2 and 3 are used as output channels. You can quickly go through the files `dma_registers.h`, `dma_setup.h`, `dma_setup.c` to see how different DMA channels have been configured.

The code takes in data through the codec and stores it in `DMA_InpX` buffers (first half part of this buffer is Ping buffer and latter half is Pong buffer) using DMA. Generally, the data in these buffers will be processed alternately and the result will be placed in output (`DMA_OutX`) buffers. For illustration, we are just setting output equal to the input.

### Program Flow

- In `main( )` function, we initialize c5515 board using `InitSystem()`. Then we configure AIC3204 codec. We set I2S bus in slave mode using `I2S0_setup()` and finally we configure DMA0 and its all 4 channels.
- `DMA_ISR(void)` is an Interrupt Service Routine defined in `dma_setup.c` file which sets Ping-Pong flags. The `DMA_ISR( )` is called when a buffer has been filled. It modifies a state variable (flags) named `PingPongFlagInX` and `PingPongFlagOutX` to 1 or 0 indicating whether the filled buffer is a PING or a PONG buffer. By checking the values of these flags, we use the corresponding data in `while(1)` loop inside `main( )` function.

### Important Notes:

- Keep `memory model` in run time options `small`.
- Select `--ptrdiff_size` in `run time options` to be equal to 16.
- In `File search path`, add `usbstk5515bsl.lib` provided to you provided along with this handout (and not from the usual location).
- Don't delete ***libc.a****("C:/Program Files (x86)/Texas Instruments/ccsv4/tools/compiler/c5500/lib/libc.a")* which is already added in the uploaded project and don't add ***rts55x.lib*** like you do normally.
- Enable real time auto refresh mode to observe the output in real time. You may refer the appendix for the steps.

## 2.1 Learning Checkpoint

**All the following tasks need to be shown to your TA to get full credit for this lab session.**

To verify whether your DMA setup is working correctly, perform the following steps:

- You will be using the `DMA_checkup` project for this checkpoint.
- As explained earlier, inside the infinite `while` loop, the program is just reading the values from the DMA buffers depending upon which of the two buffers (PING and PONG) is filled into the array `BufferL`. Later, these values are output into DMA output buffers.
- Run this project. **Follow the 'Important Notes' mentioned earlier.**
- Now input a sine wave of frequency 3kHz to 5kHz from the function generator and observe the output on the DSO. If it is showing a sinusoid with the same frequency, then you are all set to move ahead. Skip the remaining steps in this checkpoint.
- Graph the array `BufferL` into which the values are being read. Find out the size and the data-type of the array yourself.
- If the plot is showing a sinusoid, check its amplitude. If it is very low (say, of the order of a few hundreds) then in the statements writing the values to the DMA output buffers, apply a gain to the `BufferL` values such that they occupy the whole range (or at least considerable part of the whole range) of the data-type.
- If even this doesn't work, approach one of the TAs/RAs to help you with the debugging.

# 3 Convolution via Frequency Domain

## 3.1 Learning checkpoint

**All the following tasks need to be shown to your TA to get full credit for this lab session.**

- Let, $x[n] = \{4, 5, 4, 6, 7, 8, 9, 1\}$ and $h[n] = \{8, 9, 0\}$ be two sequences.
- Perform linear convolution of $x[n]$ with $h[n]$ in time domain manually and note down the result.
- Perform 8-point circular convolution of $x[n]$ with $h[n]$ in time domain manually and note down the result.
- Compare the two results. How many samples of circular convolution output match with the linear convolution output? Comment why some samples are corrupted.

Suppose, a long duration input sequence x[n] is to be convolved with a finite duration impulse response h[n]. The linear convolution operation in time domain is slow because of the large number of multiplications and additions that must be performed to compute each output sample.

We can perform the similar convolution operation by going into the frequency domain (Convolution Theorem). Since, the filtering performed via FFT involves operation on a fixed size data block, the input sequence needs to be divided into fixed size blocks and then process one block at a time as IFFT(FFT($x[n]$) $\times$ FFT($h[n]$)). Since this is block based processing, we can use DMA so that when one input data block (in ping buffer) is being processed, the next incoming data in stored in pong buffer and vice versa.

The output sequence obtained using the block based frequency domain method is actually the circular convolution of the two blocks (as per the convolution theorem). We have seen in checkpoint 3.1 that the circular convolution and linear convolution results are not exactly same. Overlap-save and overlap-add are two techniques which are used to evaluate linear convolution by efficiently manipulating the samples in each output block.

## 3.2 Overview of Overlap-Save method

If the length of the filter is $L$ and the input data size is $M$, then the convolved output will have $N = L+M-1$ values. So, if we have a real-time data coming continuously and would like to use a frequency domain approach to filtering, then we need to implement product of FFTs. For efficient FFT operation, the following procedure must be followed.

- Let's say we are allowed to use FFT blocks of size $N$ which is a power of 2.
- Let $L$ be the length of the given filter.
- Now, to make full use of the available size of the FFT block, we can set the length of the input blocks to $M$ such that, $N = L + M - 1$ or $M = N - (L - 1)$. Text
- The critical step is to choose these $M$ values in the input blocks from the incoming data stream. This is performed as follows.
  1. If this is the first input block, choose the first $L - 1$ values to be zeros. Else, choose them to be the last $L - 1$ values from the previous input block.
  2. Choose the remaining $N - (L - 1)$ values as the latest received values from the input data stream.
- If the $i^{th}$ input block is represented by $x_i[n]; 0 \leq n \leq N - 1$ and filter coefficients are represented as $h[n]; 0 \leq n \leq L - 1$, $N$ point FFTs of $x_i[n]$ and $h[n]$ are computed (by appending $h[n]$ by $N - L$ zeros) and their product is obtained.
- Now, calculate the $i^{th}$ output block as IFFT(FFT($x_i[n]$) $\times$ FFT($h[n]$)).
- The first $L-1$ values of the resulting output are discarded and remaining output values are concatenated at each stage to form the correct output.
- This is also illustrated in Figure 1.

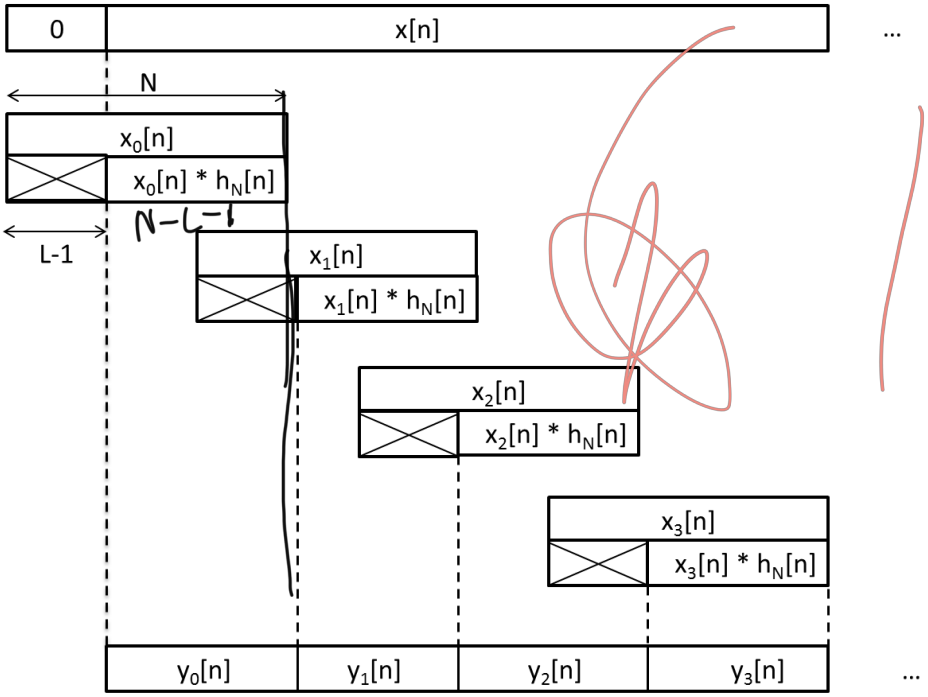


Figure 1: Illustration of overlap-save method

# 4 What is the benefit of using DMA? (Timing Considerations)

Suppose our filter length is 128 and the FFT size we used is 1024, i.e., $L = 128$ and $N = 1024$, then $M = N - L + 1 = 897$. So, every time we collect 897 new samples of data in a block and process them. Each time, we retain the last 127 values of the previous input data block in the current input data block. Also we append 896 zeros to the filter coefficients.

Let the input block be $x[n]$ and impulse response be $h[n]$, now

$X[k] = FFT(x[n])$ and $H[k] = FFT(h[n])$ are computed (1024 point FFT).

$y[n] = IFFT(X[k] \times H[k])$ is computed and the first 127 values are discarded each time and remaining samples are sent as output.

### Timing Considerations

Let the input be $x[n]$ of length 800 samples and impulse response be $h[n]$ of length 128. If we perform convolution operation sample by sample using circular buffering in assembly, we find that, for this case, we need approx 150 clocks cycles to obtain one output sample, hence total 120000 clock cycles are needed to obtain all the output samples.

If we perform the same in the frequency domain IFFT(FFT(x[n])FFT(h[n])) using N = 1024, we require total 41401 clock cycles, as shown in Table 2. We observe that the convolution output is obtained using much less number of clock cycles.

Some more timing considerations:

- The DSP is working at a frequency of 100 MHz (clock period = 10ns)
- Let the sampling frequency is to be 48kHz, i.e every input sample arrives every 20 us approx.

| Complex FFT | Cycles |
|---|---|
| 8 pt | 130 |
| 16 pt | 170 |
| 32 pt | 321 |
| 64 pt | 436 |
| 128 pt | 912 |
| 256 pt | 1668 |
| 512 pt | 3740 |
| 1024 pt | 7315 |

Table 1: Clock cycles

| Operation | Clock Cycles |
|---|---|
| FFT(x[n]) | 7315 |
| FFT(h[n]) | 7315 |
| IFFT(FFT(x[n])FFT(h[n])) | 7315 |
| Complex Multiplications | 19*1024 |
| Total: | 41401 |

Table 2: Frequency Domain Timing Considerations

Consider a continuous stream of input samples. If we choose to perform block based filtering, we find that each block takes $41401 * 10ns = 414us$ approx. Since the input arrives every 20 us approx, 21 samples will be dropped every time the block is processed.

We can solve the issue using DMA ping-pong buffering as explained in section 1 and 2.

### 4.1 Learning checkpoint

**All the following tasks need to be shown to your TA to get full credit for this lab session.**

In this checkpoint, you will implement the overlap-save method. Use the `lab8_filter` project provided with this handout for this checkpoint. It contains the impulse response of length 12.

- You may declare any extra buffers (arrays, essentially), if needed.
- You have to use a 256 length FFT (as mentioned in the file `audio_setup.h`).
- Decide the value of the constant `PINGPONGSIZE` declared in the same file.
- Write a code to implement overlap-save method.
- To verify the working of your code, define the `COEFFS` array (inside the file `lowpass.h`) to be an impulse, i.e., set first coefficient to say, 3000, and all others to 0.
- Now, provide sine-wave as input using function generator. If you observe sine wave at the output, then your code is correct. Otherwise, debug your code by single-stepping.
- Now, use the `COEFFS` array which was given in the skeleton code.
- How will you determine the frequency response of the filter? (Hint: You have the signal generator at your disposal).
- Determine the cut-off frequency of the filter in discrete-time domain.

# Appendices

Real-time auto refresh mode The graph window in the CCS can be set in the auto refresh mode using `Silicon Real Time Mode`. Following steps indicate the real time refresh mode settings.

1. Load the program.
2. Go to `Target` ⇒ `Advanced` and select `Enable Silicon Real Time Mode` as shown in figure 2.

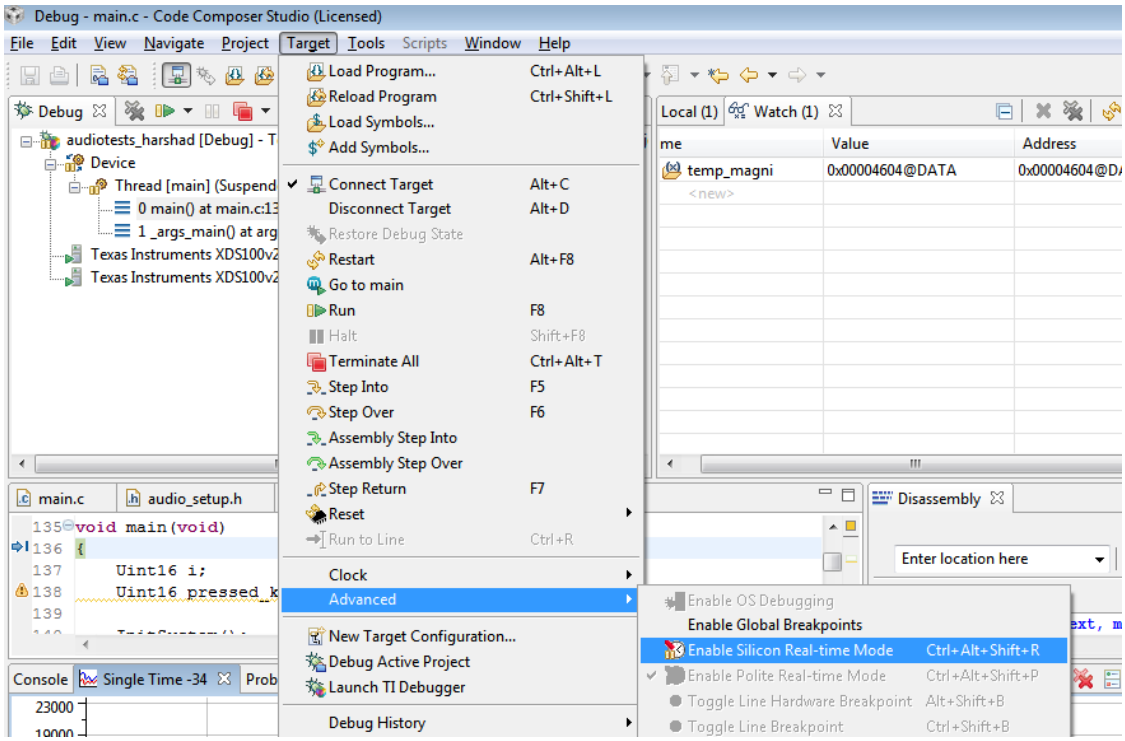


Figure 2: Selecting the Silicon-real time mode.

3. Go to `tools` ⇒ `Graphs` and select and Enable Continuous refresh in graph window.

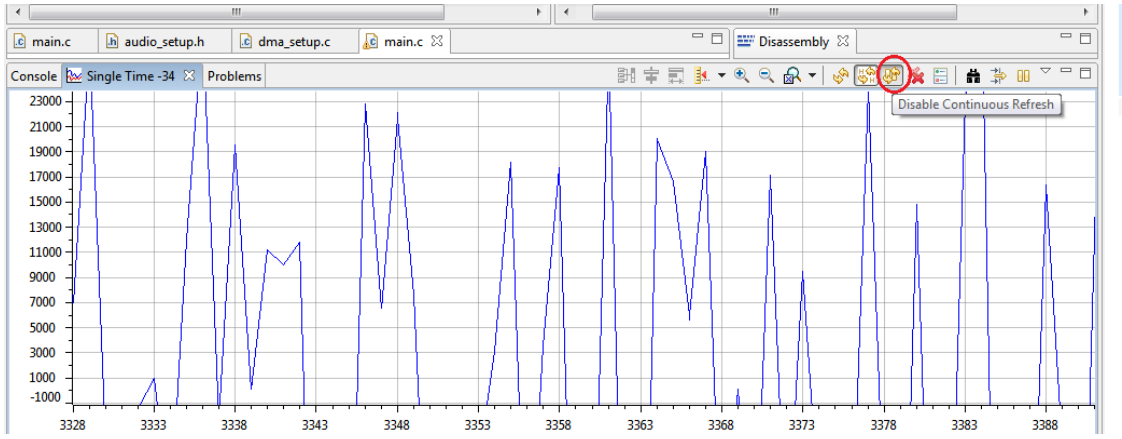


Figure 3: The continuously refreshing graph

With these settings, you should be able to view the continuously refreshing graph in the graph window as shown in figure 3.