Implementation - Filter Bank Multi-Carrier - Cosine Modulated Multitone

David Castro

I. Introduction

The initial purpose of this implementation project was to simulate Cosine Modulated Multitone in a Massive MIMO setting, following the paper of [1], but due to time constraints and an overestimation of personal ability I was not able to achieve anything close to reaching this level in the paper. What was achieved was the implementation of the Cosine Modulated Multitone (CMT), using resources such as a paper that goes through the general layout from a continuous perspective [2]. The main mathematical resources seen in this paper come from and follow similar notation from a textbook *Signal Processing Techniques for Software Radios* [3]. There was no initial intention of purchasing this book, but after researching possible implementations for CMT I found current papers citing this book and older papers explaining CMT in a very difficult to comprehend manner. I would like to note that I try to fill in and explain the mathematical concepts of [3], and that the book was only used for reference, notation, initial understanding.

II. Background on CMT

There is quite a long history behind CMT. Most of my understanding from the history comes from the author and professor Behrouz Farhang-Boroujeny, who has done a lot of research into filter banks and its uses in communications. A paper that gives a large overview over CMT is one that discusses the differences between OFDM and filter bank multicarrier [4]. Cosine Modulated Multitone is a type of filter bank multicarrier. In this paper, it introduces the history behind this technology. Filter Bank Multi-Carrier or CMT was initially derived in 1966 [5]. This led to a paper the following year which developed an alternative filter bank multicarrier known as Staggered Multitone, or Offset QAM filter bank [6]. There was some research done into Staggered Multitone for the following decade, but overall this technology got forgotten. In the 90s and early 2000's this technology was rediscovered in the discrete domain for digital subscriber line (DSL) communications under the name of discrete wavelet multitone (DWMT). In this reinvention in the discrete domain, there has been some talks about possible implementation in standards regarding power line communications [7], but it seems like it has not gotten much attention since. Personally I would like to note that researching this topic has been somewhat intimidating, at least for CMT because I personally could not find any good papers or documents that explain the mathematical concepts in an introductory manner. This reason leads to me to believe why it may not be very popular.

III. Cosine Modulated Multitone

This section aims to introduce CMT and its initial design in the continuous domain. The paper used for this is [2]. This paper does not really go through the initial derivations for CMT, but instead introduces CMT and what we need to satisfy in order to work for satisfying conditions such as no inter-symbol interference (ISI) and no inter-carrier interference (ICI). The importance of eliminating ISI and ICI will be discussed later. For now we can see the the initial implementation for CMT in figure 1 and 2.

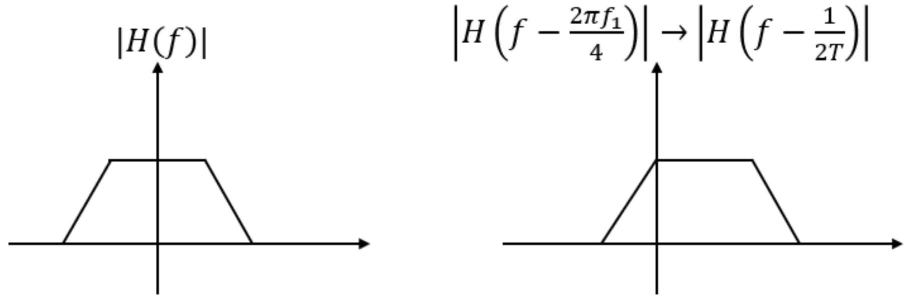

$|H(f)|$
$\left|H\left(f-\frac{2\pi f_1}{4}\right)\right| \rightarrow \left|H\left(f-\frac{1}{2T}\right)\right|$

Fig. 1. Visual of Cosine Modulate Multitone and that it has a Vestigial Side-band Modulation

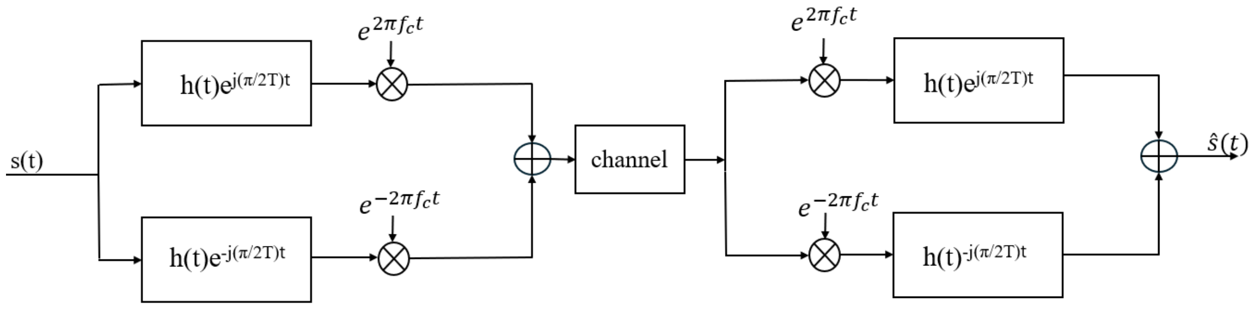


Fig. 2. Initial General CMT Layout

Since s(t), h(t) and the modulated signals are real (the summation is just summing two complex conjugates), we can take simplify the system into one branch, by taking the 'Real'. This result is seen in figure 3.

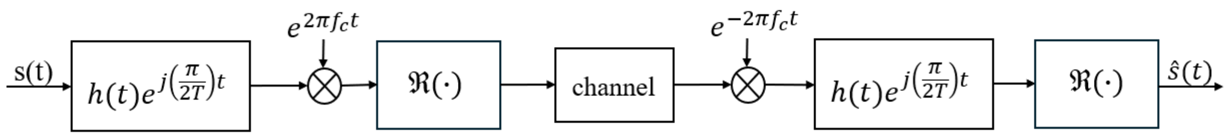


Fig. 3. Simplified CMT Block Diagram From Figure 2

Once we have this single block diagram, we can combine many in order to produce a filter bank which we will now call CMT. Figure 4 and 5 give a visual for how a transmitter and receiver would look like it in the continuous domain.

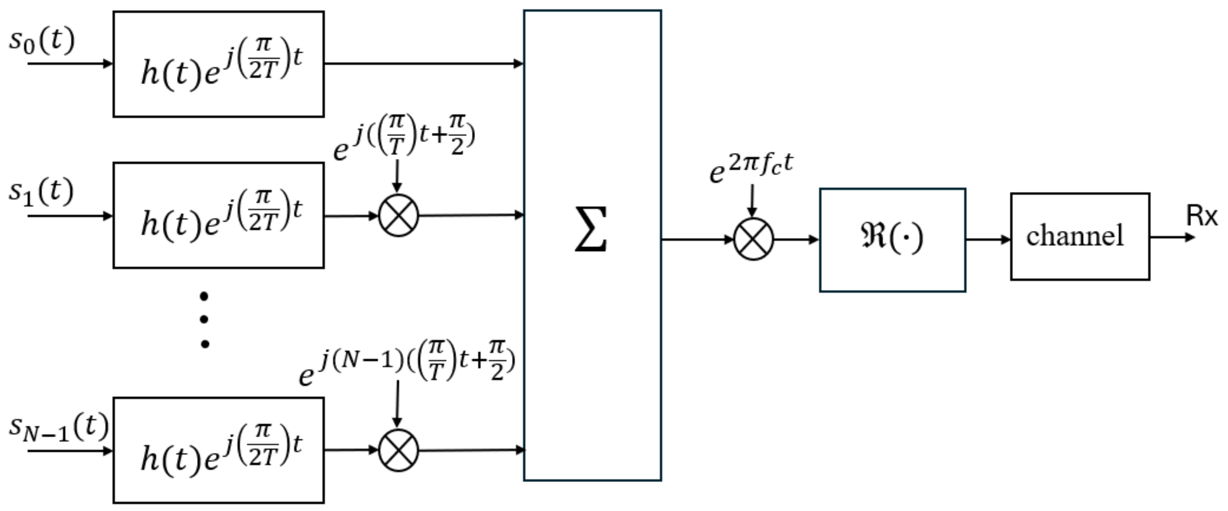

$s_0(t)$
$h(t)e^{j\left(\frac{\pi}{2T}\right)t}$
$s_1(t)$
$e^{j\left(\left(\frac{\pi}{T}\right)t+\frac{\pi}{2}\right)}$
$s_{N-1}(t)$
$e^{j(N-1)\left(\left(\frac{\pi}{T}\right)t+\frac{\pi}{2}\right)}$
Σ
$e^{2\pi f_c t}$
$\Re(\cdot)$
channel
Rx

Fig. 4. CMT Transmitter

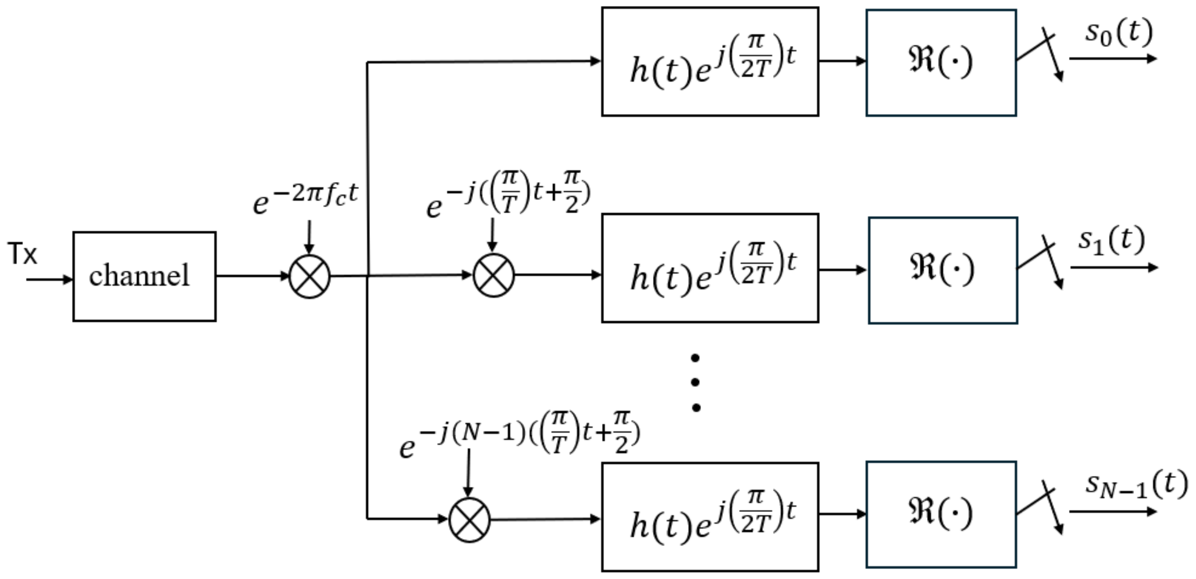

Tx
channel
$e^{-2\pi f_c t}$
$e^{-j\left(\left(\frac{\pi}{T}\right)t+\frac{\pi}{2}\right)}$
$e^{-j(N-1)\left(\left(\frac{\pi}{T}\right)t+\frac{\pi}{2}\right)}$
$h(t)e^{j\left(\frac{\pi}{2T}\right)t}$
$\Re(\cdot)$
$s_0(t)$
$s_1(t)$
$s_{N-1}(t)$

Fig. 5. CMT Receiver

Since we have our continuous implementation of CMT we can now go into implementing this into the discrete domain, but first I would like to talk about ISI and ICI. In [2] they go through the mathematical proof for how to eliminate ISI and ICI. ISI is when there are symbols from different timings that can affect each other. This means a symbol sent earlier affects the symbol/signal sent later. They find that this can be eliminated by having h(t) be a square root nyquist filter with zero crossings at intervals of time 2T. ICI is when branches affect each other at the same time. The main problem usually comes when neighboring branches have an affect on each other. For example a signal sent through branch 1 somehow affects the signal at branch 2 and vice versa. This needs to be eliminated if data is going to be sent parallel. The way ICI is eliminated is by introducing a $pi/2$ phase shift between adjacent branches. The $pi/2$ phase shift in neighboring branches can be seen in figure 4 and 5. I chose to not go throughout these two proofs due to limited time in writing this paper. Either way I believe these proofs are not very interesting as they only really show that the design works and not how it came to be developed. [5] goes through the initial development of this design. In my opinion [5] is very tough to process, but it does show how CMT was first developed.

IV. Transmitter/Synthesis - Implementing in Discrete Domain

In filter banks, when we combine many signals to synthesize one signal which the transmitter is doing, this is sometimes known as synthesis. We have seen the block layout for CMT in the linear domain, but to really take advantage of today's technology, this type of structure should be converted to the digital domain.

Before starting I would like to highlight that derivation follows from [3]. I aim to fill in the empty gaps and provide explanations not given in the literature to create a difference between me and the book.

In order to start, lets see recall our linear CMT design, and model a single branch - figure 6. These first steps will be about simplifying our CMT design first.

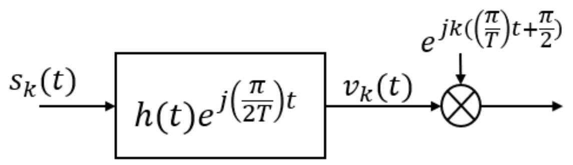

$s_k(t)$
$h(t)e^{j\left(\frac{\pi}{2T}\right)t}$
$v_k(t)$
$e^{jk\left(\left(\frac{\pi}{T}\right)t+\frac{\pi}{2}\right)}$

Fig. 6. Single Branch of CMT Design on Transmitter Side

$$s_k(t) = \sum_n s_k[n]\delta(t - nT) \quad (1)$$

The reasoning for equation (1) is that we have our initial inputs or symbols as $s_k[n]$. We don't want to place them at any time,but instead at a consistent interval Period (T) so that they can be sampled easily at the receiver. This behavior can be seen in figure 7.

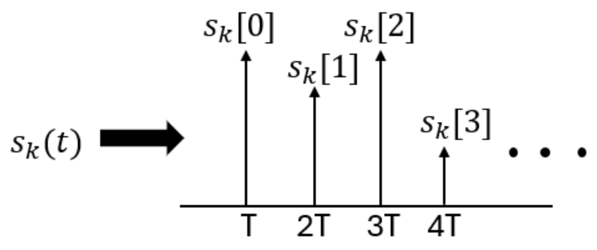


Fig. 7. An Example of Behavior for s(t)

Now we want to find $v_k(t)$:

$$v_k(t) = s_k(t) * h(t)e^{j(\pi/2T)t}$$

$$= \sum_n s_k[n]\delta(t - nT) * h(t)e^{j(\pi/2T)t}$$

$$= \sum_n s_k[n]h(t - nT)e^{j(\pi/2T)(t-nT)}$$

$$= \sum_n s_k[n]h(t - nT)e^{j(\pi/2T)}e^{-j(\pi n/2)}$$

We can rewrite the $e^{-j\pi n/2}$ as +/-j depending on 'n'

$$= \sum_n (-j)^n s_k[n]h(t - nT)e^{j(\pi/2T)}$$

Next we rewrite $s_k(t)$ as $\hat{s}_k(t)$ by bringing the $(-j)^n$ with it

$$\hat{s}_k(t) = \sum_n (-j)^n s_k[n]\delta(t - nT)$$

Currently our output for the block diagram (Fig. 6) is $v_k(t) * e^{jk(\pi t/T+\pi/2)}$

We further simplify this equation by manipulating $e^{jk(\pi t/T+\pi/2)}$

$$e^{jk(\pi t/T+\pi/2)} => e^{jk(\pi t/T)} * e^{jk\pi/2}$$

From here $e^{jk\pi/2}$ can be written as $(j)^k$

From all this simplification we can rewrite our CMT filterbank structure (Fig. 8):

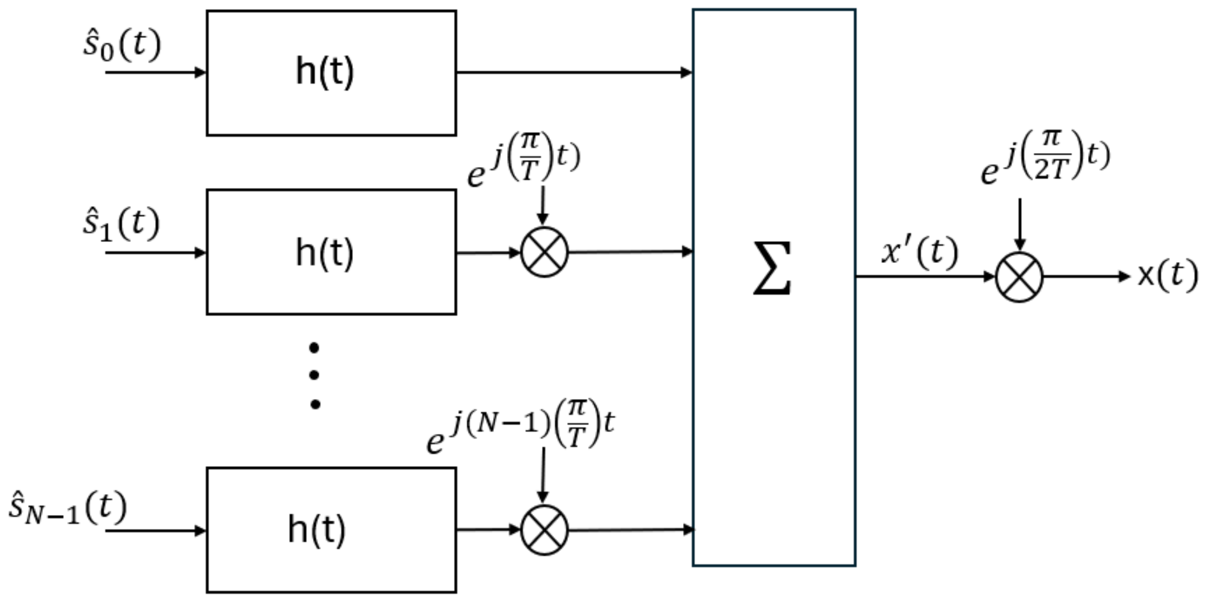

$\hat{s}_0(t)$
h(t)
$\hat{s}_1(t)$
h(t)
$e^{j\left(\frac{\pi}{T}\right)t)}$
$\hat{s}_{N-1}(t)$
h(t)
$e^{j(N-1)\left(\frac{\pi}{T}\right)t}$
Σ
$x'(t)$
$e^{j\left(\frac{\pi}{2T}\right)t)}$
x(t)

Fig. 8. Transmitter Filter Bank After Some Further Modifications

Note* - We do not include the carrier modulation, $e^{2\pi f_c}$, or the real function in this block. We can ignore them as they will not have any affect on the rest of our mathematical analysis.

Now with our simplified CMT analysis filter bank, we can convert to the discrete domain:

First we will take a look at $v_k(t)$

$$v_k(t) = \sum_n (-j)^n s_k[n]h(t-nT)e^{j(\pi/2T)}$$

The first thing we can change is the function $h(t-nT)$.

$$h(t-nT) => h(t) * \delta(t-nT)$$

This time shifted delta function can be thought of as an upsampler. The reason is because each original point of h(t) is now being separated by 'T' time (Fig. 9 for visual):

$$h_{upsampled}(t) = h(t) * \delta(t-nT)$$

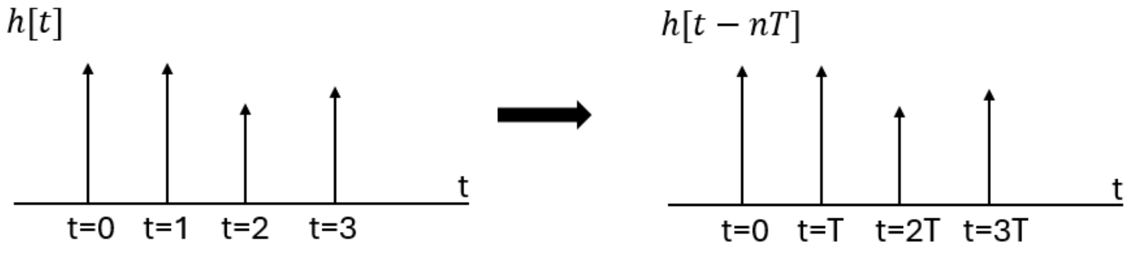


Fig. 9. h(t) having its values separated by ’T’ time

This process in the time domain can be compared to an upsampler in the discrete domain. In essence instead of upsampling by ’T’ time we are upsampling by ’L’ points. Figure 10 shows the block diagram used for notating upsampling, while the second figure, 11, shows how it works in the discrete domain.

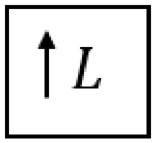


Fig. 10. Upsampler Symbol

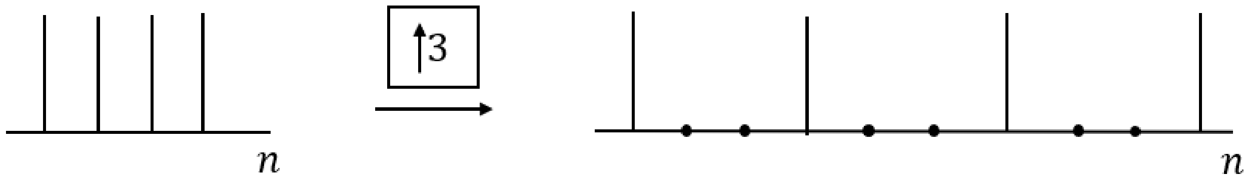


Fig. 11. Upsampling Example with an Upsampler of 3

The second thing to look at is how each modulator gets affected, e.g. $e^{jk(\pi t/T}$

* Somethings to note before going through any math.

- In the continuous domain, the signal $s_k(t)$ had each of its points separated by ’T’ time.

- In the discrete domain, we can accomplish the same thing by upsampling by 'L'.
- Each point in the discrete domain is separated by Ts time. When we upsample, the data rate increases since we aim to keep the same amount of time between the original points but we now have 'L-1' zero points between our original data. This means the time between each point has to decrease, and we end up with a new time between each point in the upsampled data:

$$\frac{T_s}{L} = T_{new}$$

-Now lets take a look at our modulators. For the example $e^{jk(\pi t/T)}$. In the continuous version, the frequency shift (f_k) changes by:

$$f_k = k * f_c \quad k = 0, 1, ...(N-1)$$

$$f_k = \frac{k}{T_k}$$

If we were to convert this to the digital domain, our frequencies become normalized to the sampling frequency $(2\pi = f_s)$

$$f_{k,digital} = \frac{f_k}{f_s}$$

$$= \frac{k/T_k}{1/T_s}$$

but as we recently said, this T_s is not the original T_s. We want T_{new} since we are working in a different frequency normalization, therefore our equation becomes:

$$= \frac{k/T_k}{1/T_{new}}$$

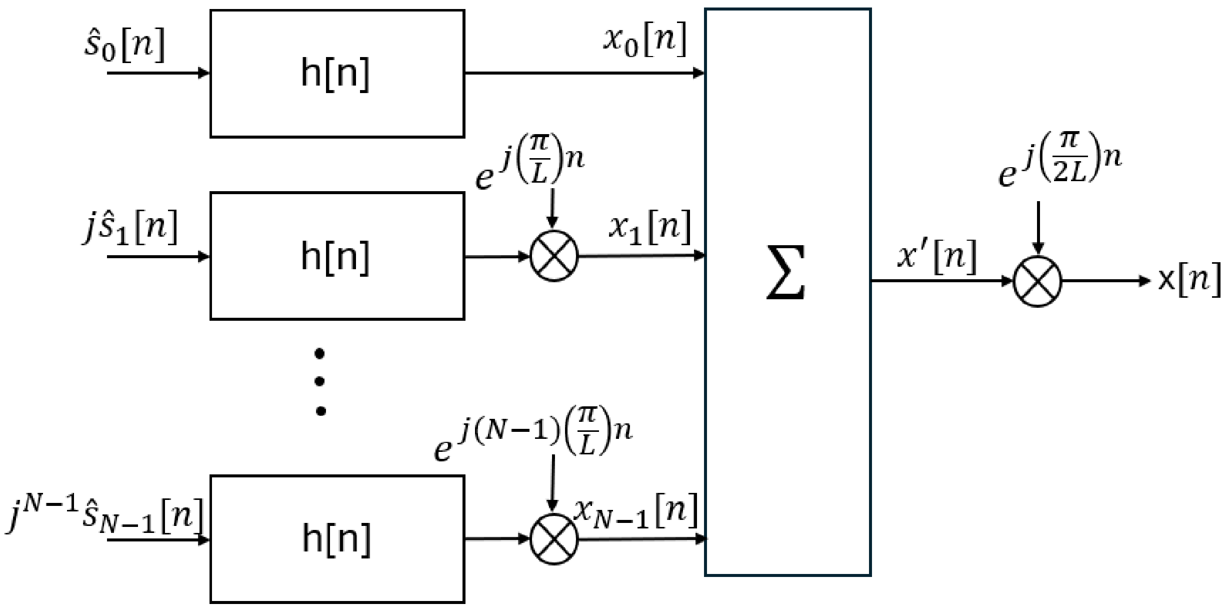
$$= \frac{k/T_k}{L/T_s}$$

We can further simplify since in our CMT filter bank, T_k and T_s are the same values. Since we have T_k and T_s both as 'T' in our system, our final formulation for this is:

$$f_{k,digital} = \frac{k}{L}$$

So in essence by having a standard 'T' throughout the system, we can just replace any 'T' with 'L' in order to convert to our digital domain.

Lastly we can convert any 't' to 'n' as that will be our discrete variable, and now we have our converted digital CMT filter bank:



$\hat{s}_0[n]$
h[n]
$x_0[n]$
$j\hat{s}_1[n]$
h[n]
$e^{j\left(\frac{\pi}{L}\right)n}$
$x_1[n]$
$j^{N-1}\hat{s}_{N-1}[n]$
h[n]
$e^{j(N-1)\left(\frac{\pi}{L}\right)n}$
$x_{N-1}[n]$
Σ
$x'[n]$
$e^{j\left(\frac{\pi}{2L}\right)n}$
x[n]

Fig. 12. Discrete Filter Bank for Transmitter Side

While this figure may seem like the final design for our CMT filter. There is mathematical digital signal processing (dsp) techniques and some algebraic simplification that can be done. The following steps will continue in order to get our final transmitter CMT digital design:

$$x_k[n] = \sum_m j^k(-j)^m s_k[m]h(n-mL)e^{jk(\pi/L)n}$$

we can rewrite our $e^{jk(\pi/L)n} -> (-1)^{mk} * e^{\frac{k\pi}{L}(n-mL)}$

since

$$(-1)^{mk} e^{\frac{jk\pi}{L}n} e^{-jk\pi m}$$

where

$$(-1)^{mk} * e^{\frac{jk\pi m}{L}} = 1 \quad ; m * k = even, odd$$

we can rewrite our $x_k[n]$ as:

$$x_k[n] = \sum_m j^k(-j)^m(-1)^{mk} s_k[m]h_k[n-mL]$$

where, $h_k[n] = h[n]e^{\frac{jk\pi}{L}n}$

How we can interpret this is $h_k[n]$ is $h[n]$ phase shifted or $H_k(z)$ frequency shift by $f = \frac{k}{2L}$

$s_k[n]$ can be rewritten as

$$\tilde{s}[n] = \begin{cases} s_k[n] \quad for\ k\ even \\ (-1)^n s_k[n] \quad for\ k\ odd \end{cases}$$

therefore our new $x_k[n]$ (Fig. 13 for visual):

$$x_k[n] = \sum_m j^k (-j)^m \tilde{s}_k[m] h_k[n - mL]$$

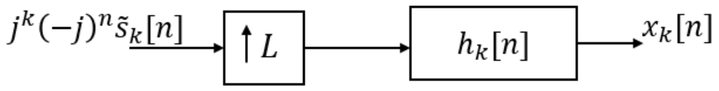


Fig. 13. A single Branch of our Filter Bank on the Transmitter side

We can now also write the z-transform for our model:

$$X_k(z) = j^k \tilde{S}_k(jz^L) H_k(z)$$

The reasoning for writing $S_k(jz^L)$ can be seen from when we take the z-transform of our upsampled $j^k(-j)^n \tilde{s}_k[n]$:

$$S_{k,e}(z) = \sum_n j^k (-j)^n \tilde{s}_k[n] z^{-nL}$$

we are able to say z^{-nL} because after being upsampled, the points will be zero everywhere else besides these points (there are 'L-1' zeros between each 'nL' adjacent point)

$$S_{k,e}(z) = j^k \sum_n (-j)^n \tilde{s}_k[n] z^{-nL}$$

$$S_{k,e}(z) = j^k \sum_n \tilde{s}_k[n] (\frac{1}{-j} z^L)^{-n}$$

$$S_{k,e}(z) = j^k \tilde{S}_k(jz^L)$$

If we recall that we have a modulator at the end of our $x_k[n]$ summation: $e^{\frac{j\pi}{2L}n}$ (Fig. 12 for visual) we can then write the result of our transmitter as:

$$x[n] = \sum_{k=0}^{N-1} x_k[n] e^{\frac{j\pi}{2L}n}$$

It is common in this type of literature to rewrite $W_L = e^{-j\frac{2\pi}{L}}$

In our new notation, $x[n]$ becomes:

$$x[n] = \sum_{k=0}^{N-1} x_k[n] W_{2L}^{-\frac{n}{2}}$$

also for easier writing we are going to say $x'[n] = \sum_{k=0}^{N-1} x_k[n]$

$$x[n] = x'[n] W_{2L}^{-\frac{n}{2}}$$

if we are to take the Z-transform of this:

$$X(z) = \sum_n x'[n] W_{2L}^{-\frac{n}{2}} z^{-n}$$

$$X(z) = \sum_n x'[n] (z W_{2L}^{\frac{1}{2}})^{-n}$$

$$X(z) = X'(z W_{2L}^{1/2})$$

Since

$$X'(z) = \sum_{k=0}^{N-1} X_k(z)$$

and

$$X_k(z) = j^k \tilde{S}_k(jz^L) H_k(z)$$

where

$$H_k(z) = \sum_n h[n] z^{-n} e^{j\frac{k\pi}{L}}$$

$$= \sum_n h[n] z^{-n} W_{2L}^{-kn}$$

we can use these to rewrite X'(z):

$$X'(z) = \sum_{k=0}^{N-1} \sum_n j^k \tilde{S}_k(jz^L) h[n] z^{-n} W_{2L}^{-kn}$$

We can apply polyphase decomposition to h[n]. Before doing that I will give a small primer and example on what polyphase decomposition is.

Polyphase Decomposition Primer:

Polyphase decomposition is used in DSP for breaking up a system (e.g. filter) into many smaller systems. It has benefits when used with upsampling and filterbanks that will be seen later, but for now I will give simple example that I believe showcases the behavior:

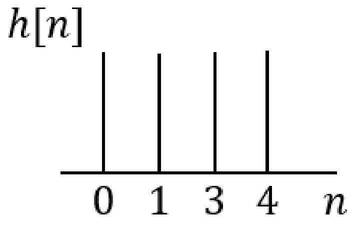


Fig. 14. 4 Tap Impulse Response for polyphase example

Looking at the figure 14 this will be used for example. We can write the Z-transform for this figure as:

$$H(z) = \sum_n h[n]z^{-n}$$

$$H(z) = h_o z^0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3}$$

we are now going to break up this filter into two parts - even and odd:

$$= (h_o z^0 + h_2 z^{-2}) + (h_1 z^{-1} + h_3 z^{-3})$$

$$= (h_o z^0 + h_2 z^{-2}) + z^{-1}(h_1 + h_3 z^{-2})$$

we now broke up this system into two polyphases:

$$E_0(z^2) = h_o z^0 + h_2 z^{-2}$$

$$E_1(z^2) = z^{-1}(h_1 + h_3 z^{-2})$$

$$H(z) = E_0(z^2) + z^{-1}E_1(z^2)$$

This type of of decomposition can be extended to higher order, meaning we can break up the system into 3 parts, 4 parts, etc. It does not even matter if the system is not a multiple of what you are breaking up the polyphase decomposition. In our even and odd example the system was split evenly, but if your system was not, you could just append zeros, and the behavior stays the same.

The two general formulas used for polyphase decomposition are:

$$E_l(z) = \sum_n h[nM + l]z^{-n}$$

$$H(z) = \sum_{l=0}^{M-1} z^{-l} E_l(z^M)$$

For further readings or information, see [8]

Returning back to our transmitter design:

$$X'(z) = \sum_{k=0}^{N-1} \sum_n j^k \tilde{S}_k(jz^L) h[n] z^{-n} W_{2L}^{-kn}$$

we want to apply polyphase decomposition to h[n] by using the formula we defined in the previous section:

$$H(z) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} z^{-l} E_l(z^M) W_{2L}^{-kn}$$

there is a trick we can do here that simplifies this formula. Lets recall $H_k(z)$

$$H_k(z) = \sum_n h[n] z^{-n} W_{2L}^{-kn}$$

the relationship between each $H_k(z)$ can be seen with:

$$H_0(z) = \sum_n h[n] z^{-n}$$

$$H_1(z) = \sum_n h[n] z^{-n} W_{2L}^{-n} = \sum_n h[n](zW_{2L})^{-n} = H_0(zW_{2L})$$

$$H_2(z) = \sum_n h[n] z^{-n} W_{2L}^{-2n} = \sum_n h[n](zW_{2L}^2)^{-n} = H_0(zW_{2L}^2)$$

From this we can see that each $H_k(z)$ is a phase shift of $H_0(z)$

$$H_k(z) = H_0(zW_{2L}^k)$$

We can use this in polyphase decomposition for making sense of what we want to do:

$$H_0(z) = \sum_{l=0}^{M-1} z^{-l} E_l(z^M)$$

$$H_1(z) = H_0(zW_{2L}) = \sum_{l=0}^{M-1} (zW_{2L})^{-l} E_l((zW_{2L})^M)$$

$$\ldots.$$

$$H_k(z) = H_0(zW_{2L}^k) = \sum_{l=0}^{M-1} (zW_{2L}^k)^{-l} E_l((zW_{2L}^k)^M)$$

so now that we have rewritten our $H_k(z)$ equation. We can look at it and wonder, "how can we further simplify this?" The variable we have under control is the polyphase decomposition order we can perform - 'M'.

$$we\,know: \quad W_{2L}^{kM} = e^{-j\frac{2\pi}{2L}kM}$$

The simplest way to simplify this solution would be to set M = 2L so that

$$W_{2L}^{2Lk} = e^{-j\frac{2\pi}{2L}2LK} = 1 \quad for\,any\,k$$

This completes our polyphase design for $H_k(z)$

$$X'(z) = \sum_{k=0}^{N-1} \sum_{l=0}^{2L-1} j^k \tilde{S}_k(jz^L) W_{2L}^{-kl} E_l(z^{2L}) z^{-l}$$

The nice thing about this form is that we can rewrite it in matrices:

$$\begin{bmatrix} \tilde{S}_0(jz^l), & j\tilde{S}_1(jz^l), & ... & j^{N-1}\tilde{S}_{N-1}(jz^l), & 0, & ... & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & W_{2L} & ... & W_2L^{2L-1} \\ ... & ... & ... & ... \\ 1 & W_{2L}^{2L-1} & ... & W_{2L}^{(2L-1)(2L-1)} \end{bmatrix}$$

$$\text{X} \begin{bmatrix} E_0(z^{2L}) & 0 & 0 & ... & 0 \\ 0 & E_1(z^{2L}) & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & E_{2L-1}(z^{2L}) \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \\ ... \\ z^{-(2L-1)} \end{bmatrix}$$

*Note the appended zeros to the input vector is to satisfy the fact that we are using polyphase greater than the amount of inputs. We originally have 'N' inputs, but we broke up our system into '2L' inputs therefore we are saying the inputs for those extra polyphase filters is zero.

We found X'(z), we now want X(z):

$$X(z) = X'(zW_{2L}^{1/2}$$

$$= \sum_{k=0}^{N-1} \sum_{l=0}^{2L-1} j^k \tilde{S}_k(j(zW_{2L}^{1/2})^L) W_{2L}^{-kl} E_l((zW_{2L}^{1/2})^{2L})(zW_{2L}^{1/2})^{-l}$$

$$= \sum_{k=0}^{N-1} \sum_{l=0}^{2L-1} j^k \tilde{S}_k(jz^L W_{2L}^{L/2}) W_{2L}^{-kl} E_l(z^{2L} W_{2L}^{L})(zW_{2L}^{1/2})^{-l}$$

we can do some simplification by:

$$W_{2L}^{L/2} = e^{-j\frac{2\pi}{2L}L/2} = e^{-j\pi/2} = -j$$

$$W_{2L}^{L} = e^{-j\frac{2\pi}{2L}L} = e^{-j*pi} = -1$$

and then going back to X(z):

$$X(z) = \sum_{k=0}^{N-1} \sum_{l=0}^{2L-1} j^k \tilde{S}_k(j*(-j)z^L) W_{2L}^{-kl} E_l(-z^{2L})(zW_{2L}^{1/2})^{-l}$$

$$= \sum_{k=0}^{N-1} \sum_{l=0}^{2L-1} j^k \tilde{S}_k(z^L) W_{2L}^{-kl} E_l(-z^{2L})(zW_{2L}^{1/2})^{-l}$$

We can also write this in matrix formulation:

$$\begin{bmatrix} \tilde{S}_0(z^l), & j\tilde{S}_1(z^l), & ... & j^{N-1}\tilde{S}_{N-1}(z^l), & 0, & ... & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & W_{2L} & ... & W_2 L^{2L-1} \\ ... & ... & ... & ... \\ 1 & W_{2L}^{2L-1} & ... & W_{2L}^{(2L-1)(2L-1)} \end{bmatrix}$$

$$\text{X} \begin{bmatrix} E_0(z^{-2L}) & 0 & 0 & ... & 0 \\ 0 & E_1(z^{-2L}) & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & E_{2L-1}(-z^{2L}) \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1}W_{2L}^{-1/2} \\ ... \\ z^{-(2L-1)W_{2L}^{-\frac{2L-1}{2}}} \end{bmatrix}$$

We can now write our model as:

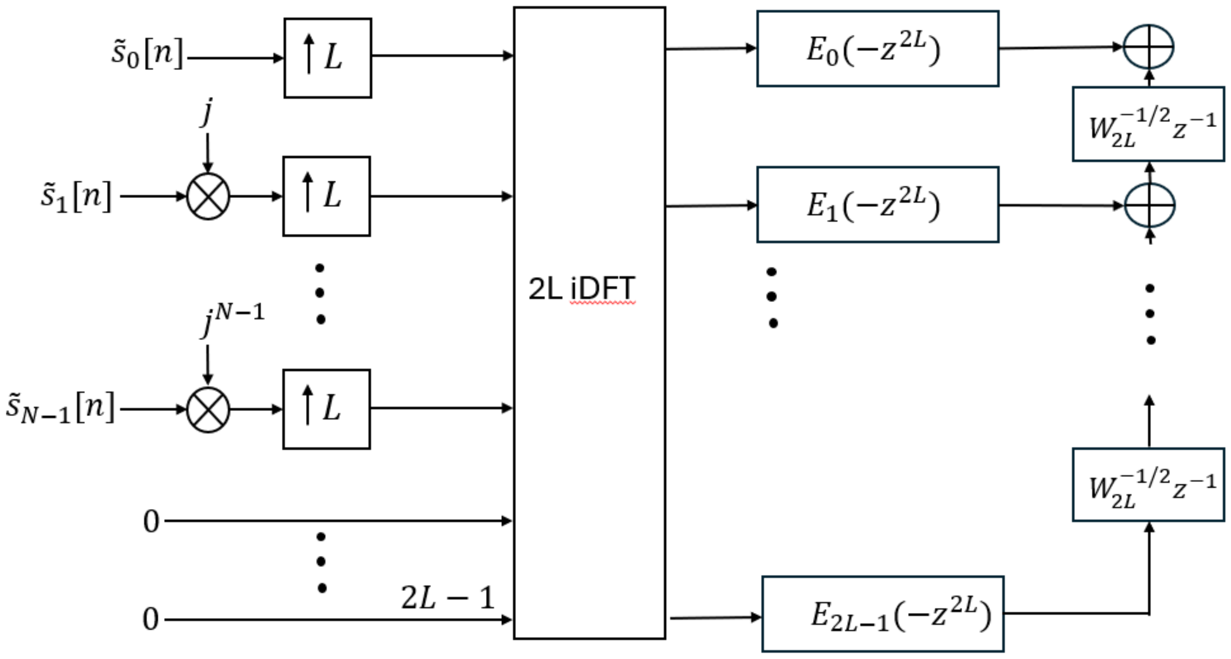


Fig. 15. Transmitter Filter Bank with Polyphase Decomposition

There is technically one further step that can be implemented and that is something known as the noble identity (Proof is not provided but can be found in [8]. This type of swap is usually desired because when you upsample you are technically processing through more data therefore performing the operation of H(z) and then upsampling is seen as a way of saving processing time.

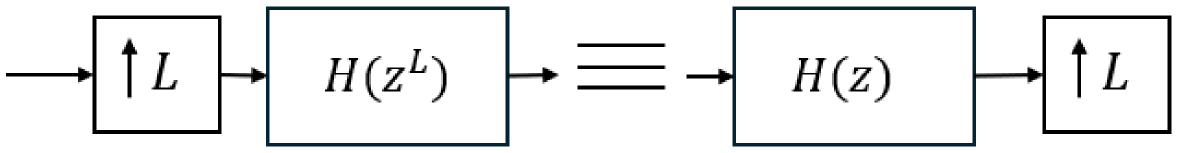


Fig. 16. Noble Identity for Upsampler

Also we are allowed to move/swap the upsampler block with the iDFT because the upsampler holds the property of linearity. The iDFT block is just multipliers and adders. So our new and final model becomes:

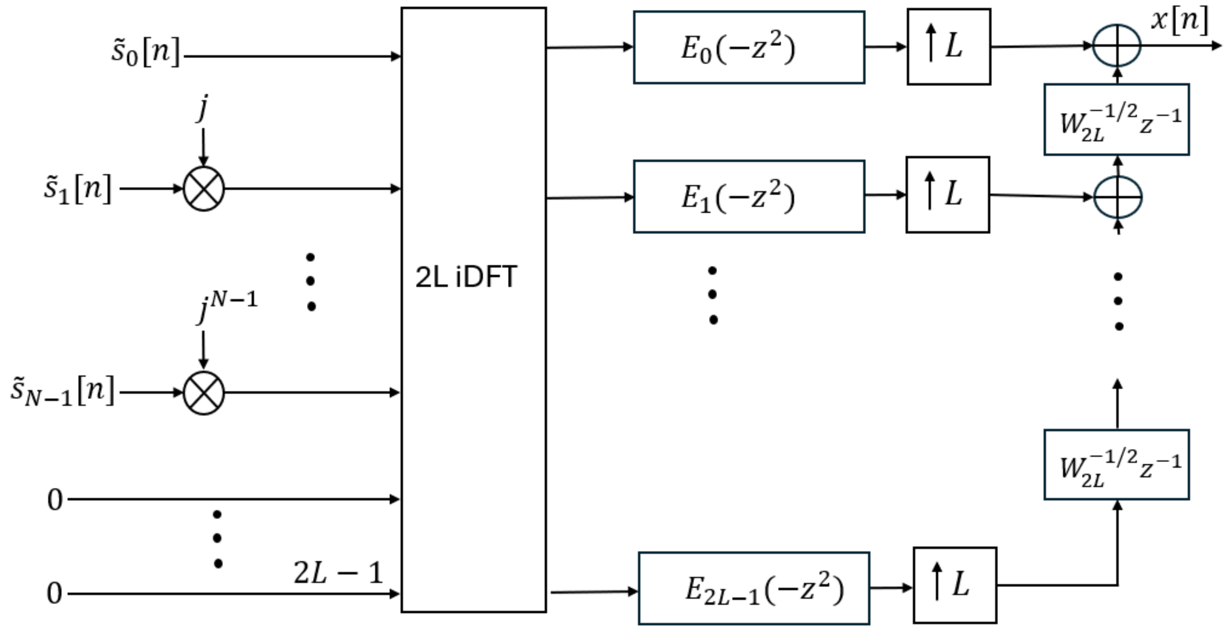


Fig. 17. Transmitter Filter Bank with Polyphase Decomposition and Noble Identity Applied

CMT Analysis Filter Bank:

The analysis filter bank, also known as the receiver filter bank shares the same math and setup as we did on the transmitter/synthesis side. The difference though is that we will be performing the opposite operations. This means instead of upsampling, we will be downsampling

The act of downsampling is also known as decimating and is shown through:

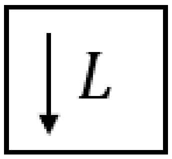


Fig. 18. DownSampler Block

We found on the transmitter side that our output could be modeled as:

$$\begin{bmatrix}\tilde{S}_0(z^l), & j\tilde{S}_1(z^l), & \ldots & j^{N-1}\tilde{S}_{N-1}(z^l), & 0, & \ldots & 0\end{bmatrix}\begin{bmatrix}H_0(z)\\ H_1(z)\\ \ldots\\ H_{2L-1}(z)\end{bmatrix}$$

The same filter banks are used on both sides since we want to extract the information transmitted in each filter. Therefore our received signal $S_k(z)$ can be represented through:

$$\begin{bmatrix}S_0(z)\\ jS_1(z)\\ \ldots\\ j^{N-1}S_{N-1}(z)\end{bmatrix} = \begin{bmatrix}H_0(z)\\ H_1(z)\\ \ldots\\ H_{2L-1}(z)\end{bmatrix} \text{X(z)}$$

We can see our linear system for the k branch as:

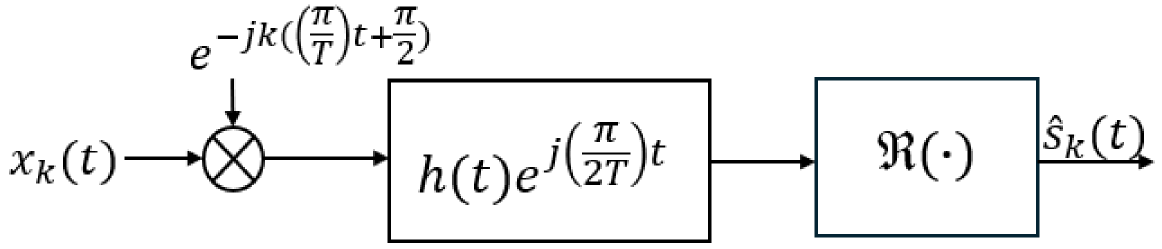

$e^{-jk((\frac{\pi}{T})t+\frac{\pi}{2})}$
$x_k(t)$
$h(t)e^{j(\frac{\pi}{2T})t}$
$\Re(\cdot)$
$\hat{s}_k(t)$

Fig. 19. Single Branch of our Receiver in linear form

And we can also see its discrete equivalence:

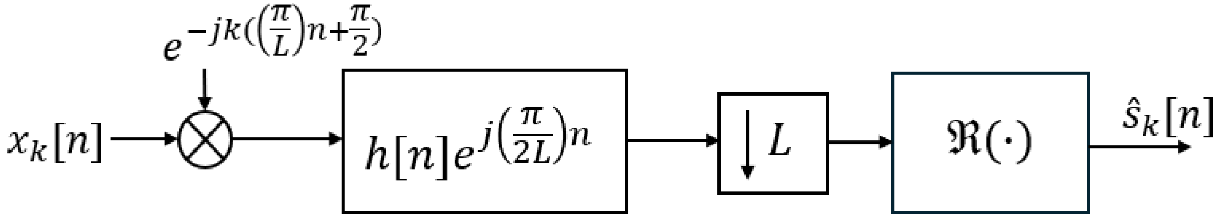

$e^{-jk((\frac{\pi}{L})n+\frac{\pi}{2})}$
$x_k[n]$
$h[n]e^{j(\frac{\pi}{2L})n}$
$\downarrow L$
$\Re(\cdot)$
$\hat{s}_k[n]$

Fig. 20. Discrete Implementation of a Single Branch on the Receiver

The main difference between the transmitter and receiver here that affects the calculations is that there is a $-j$ in the first modulator. This affects the calculations in that instead of multiplying by a $(j)^{N-1}$ we will multiply instead by $(-j)^{N-1}$. I would like to also note that this should technically affect the 2L iDFT block where if you go through the calculations, which are the same except for this $-j$ it causes you to end up with a 2L DFT block instead. To me this does not really make sense because you want your filter bank to match on both sides so that it gets the correct signal at the corresponding receiver. This is where I get some confusion and could not find any information on this. The resources I used that explain CMT say to use a 2L iDFT block on the receiver which makes sense logically, but mathematically it comes to be a 2L DFT block. I am not quite sure, but in implementation I used a 2L iDFT block on the receiver as well.

So we can see our model comes out to be:

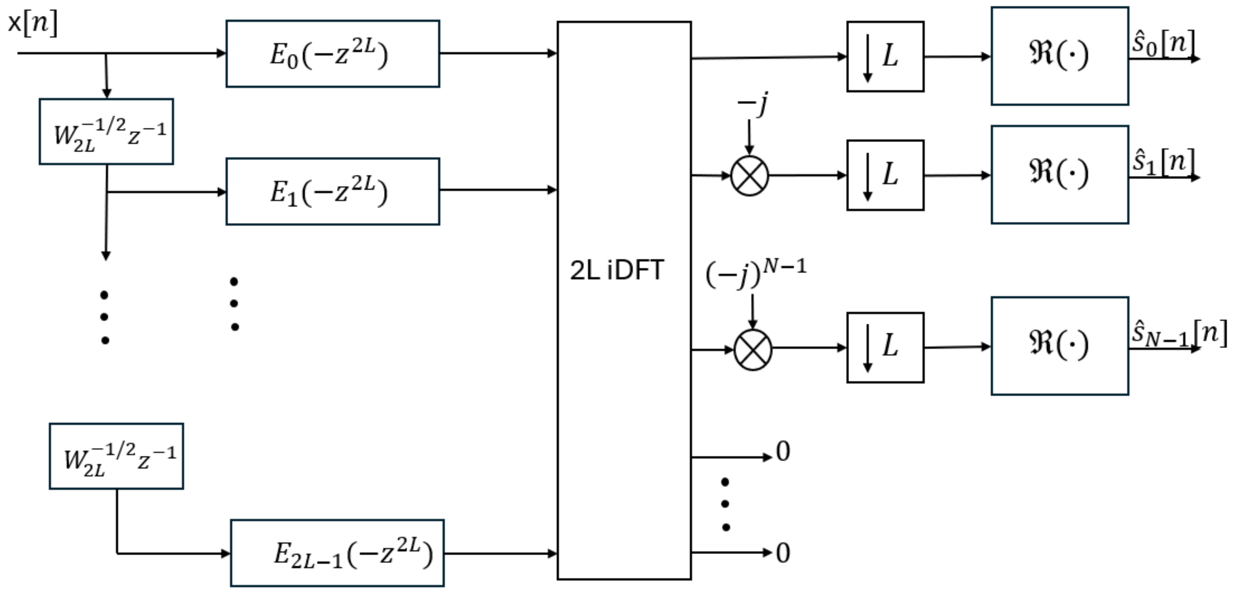


Fig. 21. Receiver Filter Bank with Polyphase Decomposition

We can also use a noble identity for downsampling as similarly done in the transmitter.

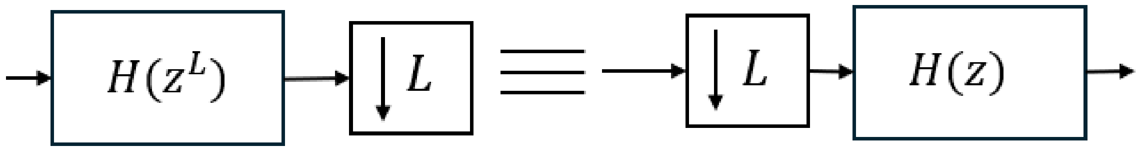


Fig. 22. Noble Identity of Downsampler

Its desired to downsample first since you make your input smaller and then perform the process of H(z) which means you are not performing as many operations.

The downsampling block holds the property of linearity so it can be swapped with 2L iDFT block and $(-j)^k$ multiplier.

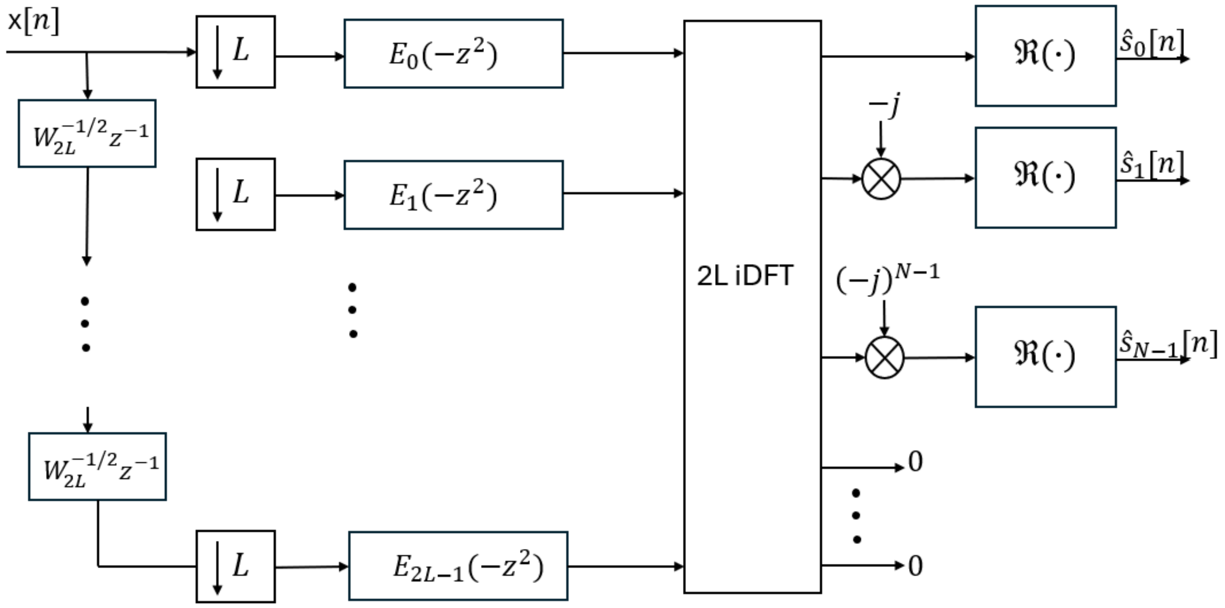


Fig. 23. Receiver Filter Bank with Noble Identity Applied

V. Simulation of CMT

There are many ways to attempt to simulate this design. It can be at a very high level where you just work with the main equations (matrices derived), a medium level by simulating the block structures, or you can run the design at a very low level where you simulate fixed point taps and try to account for quantization. I aimed for simulating by implementing the matrix functions derived. The reason I chose these and not the ones using noble identities was due to lack of time.

The first part that comes with designing the system is selecting your H(z) which is your pulse modulating signal. We know that the design has to be a square root nyquist filter. MATLAB has a built in function for designing a root-raised-cosine-filter. This filter corresponds to taking the square root in the frequency domain of a raised-cosine-filter which is a nyquist filter.

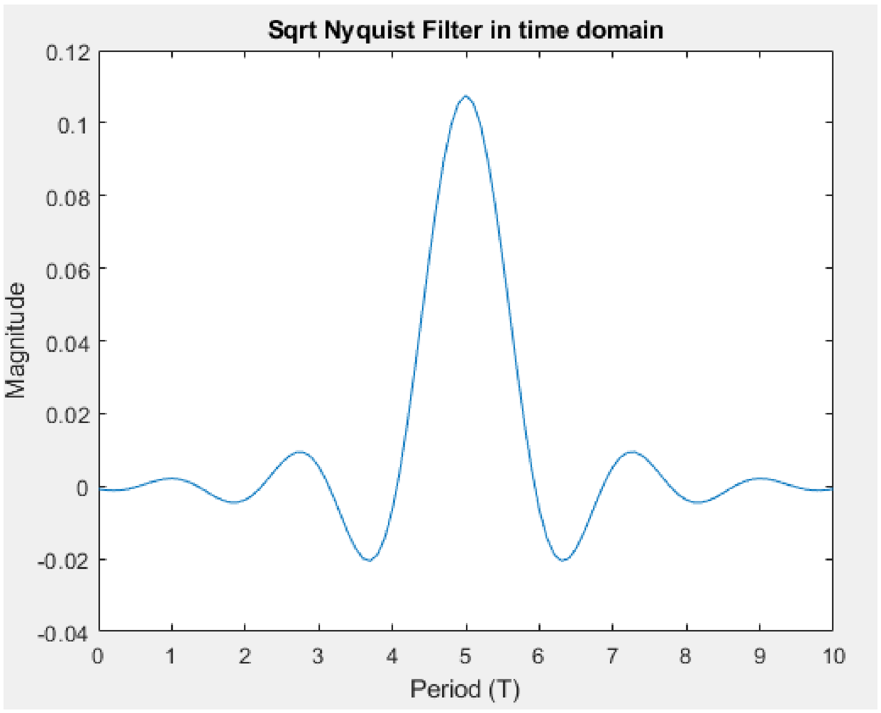


Fig. 24. Square Root Nyquist Filter in the Time Domain

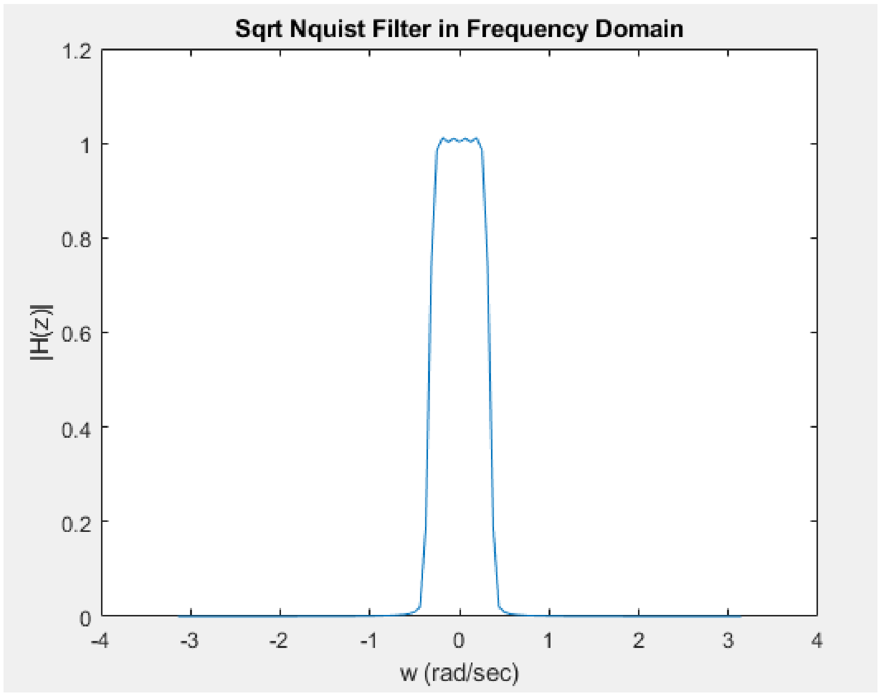


Fig. 25. Square Root Nyquist Filter in Frequency Domain (Normalized its Gain to 1)

Once you have your pulse shaping filter correctly designed. The next step is to obtain your polyphase filters. If your setup is correct, adding up all the polyphase filters should result in your original pulse shaping function.

The next step is to implement your $H_k(z)$ through the matrix multiplication.

$$\begin{bmatrix} H_0(z) \\ H_1(z) \\ \dots \\ H_{2L-1}(z) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & W_{2L} & \dots & W_2L^{2L-1} \\ \dots & \dots & \dots & \dots \\ 1 & W_{2L}^{2L-1} & \dots & W_{2L}^{(2L-1)(2L-1)} \end{bmatrix} \begin{bmatrix} E_0(z^{-2L}) & 0 & 0 & \dots & 0 \\ 0 & E_1(z^{-2L}) & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & E_{2L-1}(-z^{2L}) \end{bmatrix}$$

$$\text{x} \begin{bmatrix} 1 \\ z^{-1}W_{2L}^{-1/2} \\ \dots \\ z^{-(2L-1)}W_{2L}^{-\frac{2L-1}{2}} \end{bmatrix}$$

We can see if the performance comes out correctly if you plot out each $H_k(z)$ and each one is shifted to the right by $\frac{k2\pi}{2L}$

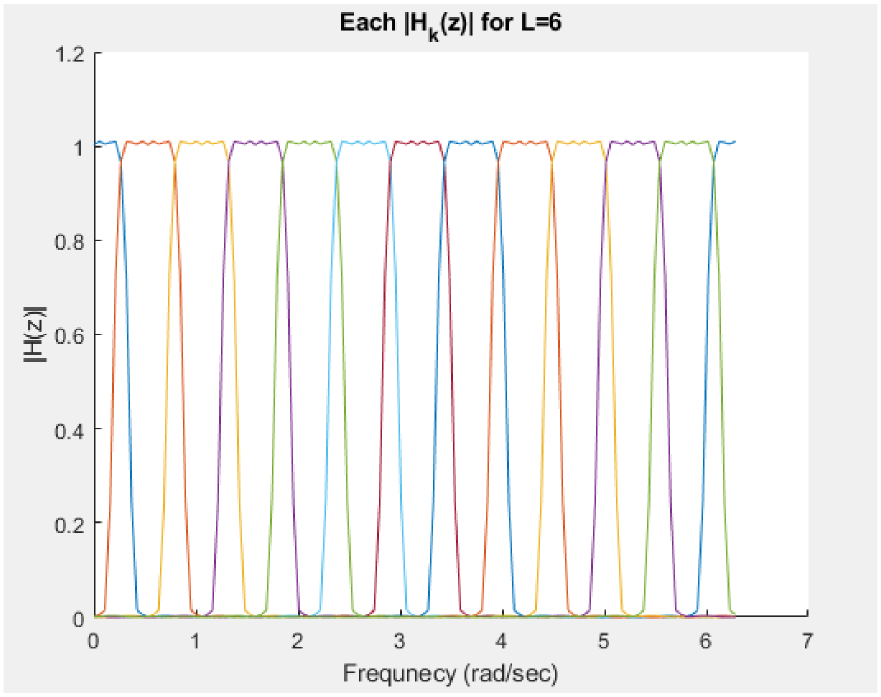


Fig. 26. This is the Magnitude Response of the Filterbank for CMT Transmitter and Receiver

Once you have this design, all that is needed is to simulate your upsampled inputs. These inputs can be any real symbol values you decide to modulate onto your signal. From my understanding, this is very similar to OFDM, with how you are placing your symbol at the center of each of $H_k(z)$ which act as the subcarriers. A visual example of how this works can be seen in the figure below. In this example our symbols are either 0 or 1 (binary values) therefore it will turn OFF or ON a subcarrier. Since this occurs. In this example only our first 4 subcarriers can carry information. We can see here the first, third, and fourth subcarriers carry a one, while the second subcarrier carries a zero. This means at $H_1(z)$, it will see a zero as it only picks up on the second subcarrier.

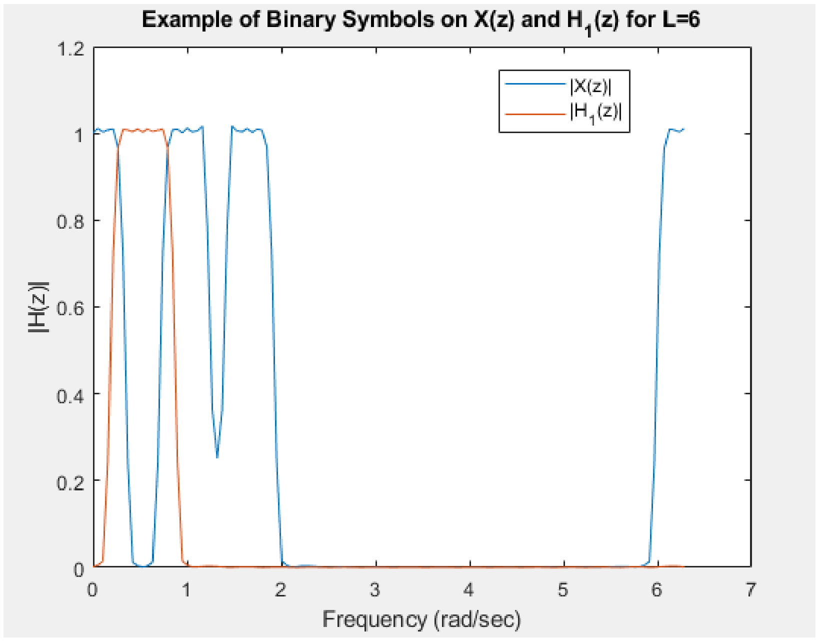


Fig. 27. Binary Symbol Modulation Example Showcasing x[n] and One of the Receiver Filter Banks

On the receiver end, you again apply your $H_k(z)$ filter bank to get your inputs. One thing to account for is applying your $(-j)^k$ multiplier to each received symbol, but once the process was done on the transmitter side, it is very easy to do on the receiver side. Since we upsampled on the transmitter side by L, we also have to downsample by L on the transmitter side and should now be able to sample your symbol.

In the setup we say at the end of each of the receivers to take the ”Real” value of each resulting signals. So theoretically this should work. The reason is because our pulse shaping function, H(z), is a real and even function. So the Fourier transform of a real and even function is always real. This means when you apply convolution or multiply in the frequency domain your result will be real. The problem with this is in MATLAB it does not really work with symmetry well. Your function may be an even function, but the way you have the data points (taps) in MATLAB, it may not actually be an even function for the fast fourier transform (fft). I believe there are ways to fix this, but from my limited testing and research certain things would need to be changed such as manually implementing the square root nyquist filter instead of using a MATLAB function to do so. Since our result does not behave properly, a band-aid fix is to take the ’magnitude’ of the result instead of the ’Real’. Technically they should be the same since theoretically our received signal is suppose to be completely real in the frequency domain, but since we do end up getting a complex value, we can get around this by taking the magnitude.

I have some thoughts on going through the process of simulating CMT. Initially I wanted to do this process by simulating the block diagrams, and had the code done, but the results were not coming out correctly. I believe this occurs due to floating point error in MATLAB. The value for the taps (pulse-shaping function) get quite small so it seems like this causes some very weird behavior where things do not behave properly. I think I could have fixed this process by making the pulse-shaping function not run as long, so the tap values do not become so small, but this may not completely fix the problem. Because of this I believe I did not get to the test the system out like how I wanted too. Implementing CMT in comparison to my past experiences with OFDM was much more frustrating because of this reason since OFDM is just much simpler to implement. There is not this whole setup required in order to get things started.

VI. Conclusion

I did not get very far in this implementation project. There were other properties about CMT that has to deal with designing the pulse-shaping filter so that it can mitigate doppler effects and ISI even more [4]. Also, you can implement a blind equalization algorithm at the receiver [9]. From the readings I have done, there appears to be some benefits that can be obtained over OFDM, but the initial barrier to entry for designing this system is much higher in my opinion. A lot of the challenges though I believe comes with a lot of the text on this subject being quite difficult sometimes to understand. Overall, there is a lot for me to research and implement in order for me to truly understand this system.

References

[1] A. Farhang, N. Marchetti, L. E. Doyle, and B. Farhang-Boroujeny, “Filter bank multicarrier for massive mimo,” in *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, 2014, pp. 1–7.

[2] B. Farhang-Boroujeny and C. Yuen, “Cosine modulated and offset qam filter bank multicarrier techniques: A continuous-time prospect,” *EURASIP J. Adv. Sig. Proc.*, vol. 2010, 12 2010.

[3] P. Vaidyanathan, *Multirate Systems and Filter Banks*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 1993.

[4] B. Farhang-Boroujeny, “Ofdm versus filter bank multicarrier,” *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 92–112, 2011.

[5] R. W. Chang, “Synthesis of band-limited orthogonal signals for multichannel data transmission,” *The Bell System Technical Journal*, vol. 45, no. 10, pp. 1775–1796, December 1966.

[6] B. Saltzberg, “Performance of an efficient parallel data transmission system,” *IEEE Transactions on Communication Technology*, vol. 15, no. 6, pp. 805–811, December 1967.

[7] “Draft standard for broadband over power line networks: Medium access control and physical layer specifications,” *IEEE Unapproved Draft Std P1901/D2.01, Jan 2010*, 2010.

[8] B. Farhang-Boroujeny, *Signal Processing Techniques for Software Radios*, 2nd ed. Salt Lake City, UT: Lulu Publishing, 2010.

[9] ——, “Multicarrier modulation with blind detection capability using cosine modulated filter banks,” *IEEE Transactions on Communications*, vol. 51, no. 12, pp. 2057–2070, 2003.

Codebase that runs through CMT filter 100 times (Real Final part of the Code:

```
% CMT_filterbank_6 - version 6 (ber and mulitiple streams)

%% setting up inputs:
close all; clc; clear;
N = 4; %number of input streams (Note N <= L)
L = 6;
stream_len = 100;
for cnt = 1:1:N
    input = randi([0, 4], 1, stream_len)/4; %generating random input stream
    original_input(cnt,:) = input;
    input = 1j^(cnt-1) .* input; %applying j term as seen in diagram/math
    S_mat(cnt, :) = input;
    S_mat_e(cnt,:) = upsample(S_mat(cnt, :), L);
end

%inserting additional input streams into S_mat_e (zeros) in order to
%satisfy the 2L input streams we need
zeros_append = zeros(2*L - N,  stream_len*L);
S_mat_e = vertcat(S_mat_e, zeros_append);


%% Sqrt Nyquist Filter using Matlab Functions
beta = 0.25; %roll off factor
span = 10; %number of symbols (how many Ts)
sps  = 10; %how many points per symbol
Period = 10^-6; %(time per symbol)
Ts = Period/sps; %(time between each point, inverse of sampling rate)
t = (0:1:(span*sps)) * Ts;
h_design = rcosdesign(beta, span, sps, 'sqrt')/pi;
h = h_design;
figure(1)
subplot(2,1,1)
plot(t/Period,h)
subplot(2,1,2)
plot((abs(fftshift(fft(h)))))

%% transceiver (synthesis)
%synthesis filter bank:
%going to design polyphase filter:

%need to append zeros to h (filter) in order to make it so that it properly
... satisfies polyphase formula
if length(h) > 2*L
    num_zeros = 2*L;
else
    num_zeros = (2*L + 2) - length(h);
end
    h_zero_pad = horzcat(h, zeros(1, num_zeros));
```

```
% this way does E(-z^2L):
%way to think about a filter is by seeing it as taps:
% h0 + h1z^-1 + h2z^-2... -> [1 0 2] == 1 + 2z^-2
for l = 1:1:(2*L) %matlab starts at 1 not 0 therefor shift everything by 1
    %we want E_l(z^2L)
     n = 0:(2*L):(length(h)); %this creates -h(L*n + l) (want negative for E(-z^2L)
     E_l(l,:) = -h_zero_pad(n + l); %break up my impulse response by 2L polphase
filters
     E_l_e(l, :) = upsample(E_l(l,:), 2*L); %upsampling by L to create proper format
end

%each E_l_e sees a delay chain (delay chain increases at E_l_e increases)
E_l_e_delay = zeros(length(E_l_e(:,1)), length(E_l_e(1,:))+2*L);
for l = 1:1:(2*L)
    E_l_e_delay(l,1:length(E_l_e(1,:))+(l-1)) = horzcat(zeros(1,l-1), E_l_e(l,:));
end

%creating the W2L matrices we see in CMT
l = 0:1:(2*L-1);
k = transpose(0:1:(2*L-1));
W2L_matrix = exp(1j * pi/L * k * l); %the 2L point idft
W2L_half_vector = transpose(exp(1j * pi/L * 1/2 * l));

%converting E_l_e to frequency
for l = 1:1:(2*L)
    freq_E_l_e_delay(l,l,:) = fft(E_l_e_delay(l,:));
end

%converting input streams to frequency (S_k(z^L):
% for l = 1:1:(2*L)
%     freq_symbol_k(:,l) = fft(S_mat_e(l,:));
% end
freq_symbol_k = transpose(S_mat_e);



%multiplying IDFT and E_k matrix (frequency):
%new matrix freq-> F* x diag(E(-z^2l)
for col = 1:1:(2*L)
    for row = 1:1:(2*L)
            %matrix multiplcation between iDFT_2L and my E_k(z^2L)
            new_matrix_freq(row,col,:) = W2L_matrix(row,col) .*
freq_E_l_e_delay(col,col, :);
    end
end

%creating my H_k_z filters:
H_k_z_transmitter = zeros(2*L, size(new_matrix_freq,3));
for row = 1:1:(2*L)
    for col = 1:1:(2*L)
        H_k_z_transmitter(row,:) = H_k_z_transmitter(row,:) + ...
        reshape(new_matrix_freq(row, col,:) .* W2L_half_vector(row) ...
        , [1, size(new_matrix_freq(1,1,:),3)]);
    end
```

```
end


%multiplying inputs with the filters:
X_z = freq_symbol_k * H_k_z_transmitter;

%% applying channel
channel = [1 0 1 0 0 0 1];
channel_zero_pad = horzcat(channel, zeros(1, size(X_z,2)-length(channel)));
for k = 1:1:size(X_z, 1)
   X_z_post_channel(k,:) = fft(channel_zero_pad) .* X_z(k,:);
end


%% receiver (analysis filter bank)
%going to design polyphase filter:


%need to append zeros to h (filter) in order to make it so that it properly
... satisfies polyphase formula
if length(h) > 2*L
    num_zeros = 2*L;
else
    num_zeros = (2*L + 2) - length(h);
end
    h_zero_pad = horzcat(h, zeros(1, num_zeros));

% this way does E(-z^2L) and then upsamples the result:
for l = 1:1:(2*L) %matlab starts at 1 not 0 therefor shift everything by 1
    %we want E_l(z^2L)
     n = 0:(2*L):(length(h)); %this creates -h(L*n + l) (want negative for E(-z^2L)
     E_l(l,:) = -h_zero_pad(n + l); %break up my impulse response by 2L polphase
filters
     E_l_e(l, :) = upsample(E_l(l,:), 2*L); %upsampling by L
end

%each E_l_e sees a delay chain (delay chain increases at E_l_e increases)
E_l_e_delay = zeros(length(E_l_e(:,1)), length(E_l_e(1,:))+2*L);
for l = 1:1:(2*L)
    %w = linspace(0,2*pi, size(E_l(1,:),2));
    E_l_e_delay(l,1:length(E_l_e(1,:))+(l-1)) = horzcat(zeros(1,l-1), E_l_e(l,:));
end

l = 0:1:(2*L-1);
k = transpose(0:1:(2*L-1));
W2L_matrix = exp(1j * pi/L * k * l); %the 2L point idft
W2L_half_vector = transpose(exp(1j * pi/L * 1/2 * l));


%for row = 0:1:(2*L-1)
    for col = 1:1:(2*L)
        for row = 1:1:(2*L)
            %matrix multiplcation between iDFT_2L and my E_k(z^2L)
            new_matrix_freq(row,col,:) = W2L_matrix(row,col) .*
freq_E_l_e_delay(col,col, :);
```

```
        end
    end


H_k_z_receiver = zeros(2*L, size(new_matrix_freq,3));
for row = 1:1:(2*L)
    for col = 1:1:(2*L)
        H_k_z_receiver(row,:) = H_k_z_receiver(row,:) + ...
        reshape(new_matrix_freq(row, col,:) .* W2L_half_vector(row) ...
        , [1, size(new_matrix_freq(1,1,:),3)]);
    end
end

out_pos = 0; %just a position var to make things work
%each row of X_z is an input (technichally there are rows with zeros from
%the upsampler but to save computation time I just skip by L inputs)
for in_cnt = 1:L:size(X_z_post_channel, 1) %going through each true input (row of
X_z)
    out_pos = out_pos + 1;
    for cnt = 1:1:(size(H_k_z_receiver,1))
        %multiply each receive filter with one input to get result
        S_k_z(cnt,:) =  (H_k_z_receiver(cnt,:) .* (X_z_post_channel(in_cnt, :)));
    end


    for cnt = 1:1:N
        S_k_n(cnt,:) = (-1j^(cnt-1))*ifft(S_k_z(cnt, :));
        %when you downsample it gets scaled down
        S_k_n_receive(cnt,:) = L*downsample(S_k_n(cnt, :), L);
        S_k_z_receive(cnt,:) = fft(S_k_n_receive(cnt,:));
    end



    %note the downsampled odd and even Sk are phased shifted by pi therefore to
    %line them up so we can sample them properly, we are going to fftshift them
    for cnt = 2:2:N
        S_k_z_receive(cnt,:) = fftshift(S_k_z_receive(cnt,:));
    end



    %the h is suppose to be a real and even filter therfore the final results
    %should be real. The way matlab implements dft makes it difficult for it
    %too properly intrepret if something is even or odd. In order to fix this I
    %wuold have to adjust the fft function or the manually implement the h
    %function myself. due to time constraint, I am going to just take the abs
    %value and consider that the real value since in essence theyre the same in
    %this case
        %get the first value (which will have the information)
        result_result = abs(S_k_z_receive(:, 1));
        quant_val = [-5:0.25:5]; %what to quantize too
        quant_output(:, out_pos) =
interp1(quant_val,quant_val,result_result,'nearest');
end
```

```
%find the ber for each input
for in = 1:1:size(quant_output, 1) %for how many inputs
    ber_k(in,:) = sum(original_input(in,:) ~=
quant_output(in,:))/length(original_input(in,:))
end


figure(2)
hold on;
for k = 1:1:size(H_k_z_transmitter,1)
    plot(abs(H_k_z_transmitter(k,:)))
end
plot(abs(fft(channel_zero_pad)))
```

Code Base 1 that showcases some Graphs:

```
%CMT filter Bank update version 3
clc; clear;

%% setting up inputs:
N = 6; %number of input streams (Note N <= L)
L = 6;
for cnt = 1:1:N
    input = randi([0, 1], 1, 10); %generating random input stream
    S_mat(cnt, :) = input;
    S_mat_e(cnt,:) = upsample(S_mat(cnt, :), L);
end

%% Sqrt Nyquist Filter using Matlab Functions
beta = 0.25; %roll off factor
span = 10; %number of symbols (how many Ts)
sps  = 10; %how many points per symbol
Period = 10^-6; %(time per symbol)
Ts = Period/sps; %(time between each point, inverse of sampling rate)
t = (0:1:(span*sps)) * Ts;
h = rcosdesign(beta, span, sps, 'sqrt')/pi;
figure(1)
plot(t/Period,h)
title('Sqrt Nyquist Filter in time domain')
xlabel('Period (T)')
ylabel('Magnitude')
figure(2)
f_axis = linspace(-pi, pi, length(h));
plot(f_axis, (abs(fftshift(fft(h)))))
title('Sqrt Nquist Filter in Frequency Domain')
xlabel('w (rad/sec)')
ylabel('|H(z)|')



%% transciever (synthesis)
%synthesis filter bank:
%going to design polyphase filter:
L = 6;

%need to append zeros to h (filter) in order to make it so that it properly
... satisfies polyphase formula
if length(h) > 2*L
    num_zeros = 2*L;
else
    num_zeros = (2*L + 2) - length(h);
end
    h_zero_pad = horzcat(h, zeros(1, num_zeros));

% this way does E(-z^2L):
%way to think about a filter is by seeing it as taps:
```

```
% h0 + h1z^-1 + h2z^-2... -> [1 0 2] == 1 + 2z^-2
for l = 1:1:(2*L) %matlab starts at 1 not 0 therefor shift everything by 1
    %we want E_l(z^2L)
     n = 0:(2*L):(length(h)); %this creates -h(L*n + l) (want negative for E(-z^2L)
     E_l(l,:) = -h_zero_pad(n + l); %break up my impulse response by 2L polphase
filters
     E_l_e(l, :) = upsample(E_l(l,:), 2*L); %upsampling by L to create proper format
end

%each E_l_e sees a delay chain (delay chain increases at E_l_e increases)
E_l_e_delay = zeros(length(E_l_e(:,1)), length(E_l_e(1,:))+2*L);
for l = 1:1:(2*L)
    E_l_e_delay(l,1:length(E_l_e(1,:))+(l-1)) = horzcat(zeros(1,l-1), E_l_e(l,:));
end

%sanity check
% figure(2)
% freqz(sum(E_l_e_delay,1))


%creating the W2L matrices we see in CMT
l = 0:1:(2*L-1);
k = transpose(0:1:(2*L-1));
W2L_matrix = exp(1j * pi/L * k * l); %the 2L point idft
W2L_half_vector = transpose(exp(1j * pi/L * 1/2 * l));


%non sanity check part is for converting to frequency
sum_freq_E_l_e = zeros(1, length(E_l_e_delay(1,:)));
for l = 1:1:(2*L)
    freq_E_l_e_delay(l,l,:) = fft(E_l_e_delay(l,:));
    sum_freq_E_l_e = sum_freq_E_l_e + fft(E_l_e_delay(l,:)); %sanity check
end
sum_h_n = ifft(sum_freq_E_l_e); %sanity check
% figure(3)
% freqz(sum_h_n)


%multiplying IDFT and E_k matrix (frequency):
%new matrix freq-> F* x diag(E(-z^2l)
for col = 1:1:(2*L)
    for row = 1:1:(2*L)
            %matrix multiplcation between iDFT_2L and my E_k(z^2L)
            new_matrix_freq(row,col,:) = W2L_matrix(row,col) .*
freq_E_l_e_delay(col,col, :);
    end
end

%sanity check by seeing if filter bank is properly produced:
H_k_z = zeros(2*L, size(new_matrix_freq,3));
for row = 1:1:(2*L)
    for col = 1:1:(2*L)
        H_k_z(row,:) = H_k_z(row,:) + ...
        reshape(new_matrix_freq(row, col,:) .* W2L_half_vector(row) ...
        , [1, size(new_matrix_freq(1,1,:),3)]);
    end
end

figure(4)
hold on
freq_axis = linspace(0,2*pi, length(H_k_z(1,:)));
for cnt = 1:1:length(H_k_z(:,1))
    plot(freq_axis, abs(H_k_z(cnt,:)))
end
hold off
title('Each |H_k(z)| for L=6')
xlabel('Frequnecy (rad/sec)')
ylabel('|H(z)|')


%% reciever (analysis filter bank)
%going to design polyphase filter:
L = 6;

%need to append zeros to h (filter) in order to make it so that it properly
... satisfies polyphase formula
if length(h) > 2*L
    num_zeros = 2*L;
else
    num_zeros = (2*L + 2) - length(h);
end
    h_zero_pad = horzcat(h, zeros(1, num_zeros));

% this way does E(-z^2L) and then upsamples the result:
for l = 1:1:(2*L) %matlab starts at 1 not 0 therefor shift everything by 1
    %we want E_l(z^2L)
     n = 0:(2*L):(length(h)); %this creates -h(L*n + l) (want negative for E(-z^2L)
     E_l(l,:) = -h_zero_pad(n + l); %break up my impulse response by 2L polphase
filters
     E_l_e(l, :) = upsample(E_l(l,:), 2*L); %upsampling by L
end

%each E_l_e sees a delay chain (delay chain increases at E_l_e increases)
E_l_e_delay = zeros(length(E_l_e(:,1)), length(E_l_e(1,:))+2*L);
for l = 1:1:(2*L)
    %w = linspace(0,2*pi, size(E_l(1,:),2));
    E_l_e_delay(l,1:length(E_l_e(1,:))+(l-1)) = horzcat(zeros(1,l-1), E_l_e(l,:));
end

l = 0:1:(2*L-1);
k = transpose(0:1:(2*L-1));
W2L_matrix = exp(-1j * pi/L * k * l); %the 2L point idft
W2L_half_vector = transpose(exp(1j * pi/L * 1/2 * l));


%for row = 0:1:(2*L-1)
    for col = 1:1:(2*L)
        for row = 1:1:(2*L)
            %matrix multiplcation between iDFT_2L and my E_k(z^2L)
            new_matrix_freq(row,col,:) = W2L_matrix(row,col) .*
freq_E_l_e_delay(col,col, :);
        end
    end


H_k_z = zeros(2*L, size(new_matrix_freq,3));
for row = 1:1:(2*L)
    for col = 1:1:(2*L)
        H_k_z(row,:) = H_k_z(row,:) + ...
        reshape(new_matrix_freq(row, col,:) .* W2L_half_vector(row) ...
        , [1, size(new_matrix_freq(1,1,:),3)]);
    end
end

figure(5)
hold on
freq_axis = linspace(0,2*pi, length(H_k_z(1,:)));
for cnt = 1:1:length(H_k_z(:,1))
    plot(freq_axis, abs(H_k_z(cnt,:)))
end
```