Modified FFT Algorithm

March 29, 2019

In this document I will outline a modification of the FFT algorithm that can achieve the desired filtering in the quantum emulation device. The five qubit quantum emulation device represents a quantum state with an input signal having energy at 32 different frequencies. The frequencies span from 1kHz to 32kHz at multiples of 1kHZ. When gate operations are performed on the device, this signal is transformed and filtered into a signal representing the output state of the system. The filtering operations performed depend on the qubit being targeted for the gate operation. This document describes a digital filtering algorithm that greatly reduces the complexity of all filtering operations. It relies on the similarity of the desired filtering structure to the radix-2 Decimation in Frequency FFT algorithm.

Complexity Comparison FFT vs. FIR Filter

We assume the following approximate parameters derived from our system: Assume the time-domain sequence length is $N = 512$ samples. Also assume we have 32 distinct frequencies, and that we always filter exactly half of them per filtering operation.
We compare the following three options

1. Full FFT and IFFT
2. Partial FFT and IFFT (to be described below)
3. Filter bank of $N_f = 16$ separate bandpass filters, order assumed to be $L = 150$.

We pick a moderate value for the filter order because there may be more efficient ways to do the filtering (e.g. a comb filter).

1. Using a radix-2 algorithm, the first method will require approximately $2N \log_2(N) = 4096$ multiply-adds with an equal portion coming from the FFT and the IFFT.
2. The second method will require (as will be shown) $N \log_2(N) = 2048$ multiply-adds.
3. The third method will require approximately $L \cdot N_f \cdot N = 614400$ multiply-add operations. This comes from needing L multiply adds per sample, times the number of samples in the sequence, times the number of filters applied.

Reduced FFT Model

We assume a system with a sampling frequency f_s and a time domain sequence $x[n]$ of length N. To enable the use of a radix-2 algorithm, we assume $N = 2^k$ for some positive integer k. The Discrete Fourier Transform of $x[n]$ is defined as

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

Notice that $x[n]$ (i.e. x with a serif font and with index n) will denote time-domain sequences while $\mathsf{x}[k]$ (sans serif and with index k) will denote frequency domain sequences.

Desired Filtering Operations

We assume that we have captured precisely an integer number of periods of the lowest frequency of $x[n]$ which allows us to disregard issues of spectral leakage. Recall that the resolution of the DFT is defined as

$$\Delta f = \frac{f_s}{N} \tag{2}$$

indicating that neighboring frequency bins $\mathsf{x}[k]$ and $\mathsf{x}[k+1]$ for some k are spaced by Δf. We assume that our frequency bins are perfectly aligned with the desired frequencies (e.g. $\Delta f/N = 1000$ which ensures each frequency bin is a multiple of 1 kHz).

Keeping things general, suppose we want to perform the following filtering operations:

1. Nulling every other frequency of $\mathsf{x}[k]$ while passing the rest.
2. Nulling two adjacent frequencies $\mathsf{x}[k]$ and $\mathsf{x}[k+1]$ while passing the next two $\mathsf{x}[k+2]$ and $\mathsf{x}[k+3]$.
3. Nulling four adjacent frequencies while passing the next four
4. All further generalizations up to nulling $N/2$ adjacent frequencies and passing the other $N/2$ frequencies.

In general, for some nonnegative integer $\ell \leq \log_2(N/2)$, we want to null out 2^ℓ bins and pass the next 2^ℓ bins in a repeating fashion over the length of the frequency domain sequence.

$\ell = 0$ Case

When $\ell = 0$, the case corresponds to that of filtering out every other frequency. If we gather the portion of the sequence that we wish to filter into a set $\mathcal{X}_{\mathsf{stop}}$ and the other portion, which we wish to pass, into another set $\mathcal{X}_{\mathsf{pass}}$, we will have either

$$\begin{aligned} \mathcal{X}_{\mathsf{stop}} &= \{\ldots \mathsf{x}[-2], \mathsf{x}[0], \mathsf{x}[2], \ldots\} \\ \mathcal{X}_{\mathsf{pass}} &= \{\ldots \mathsf{x}[-3], \mathsf{x}[-1], \mathsf{x}[1], \ldots\} \end{aligned} \tag{3}$$

or

$$\begin{aligned} \mathcal{X}_{\mathsf{stop}} &= \{\ldots \mathsf{x}[-3], \mathsf{x}[-1], \mathsf{x}[1], \ldots\} \\ \mathcal{X}_{\mathsf{pass}} &= \{\ldots \mathsf{x}[-2], \mathsf{x}[0], \mathsf{x}[2], \ldots\} \end{aligned} \tag{4}$$

That is, either the even-indexed frequencies should be nulled, or the odd indexed frequencies should be nulled. We assume for the purpose of demonstration that the latter case holds. To perform the complementary filtering operation requires only a slight modification of what's to follow.

In this case, we'd like to use the FFT to null out odd indexed frequencies and pass the even-indexed frequencies. Denote the even indexed frequencies by the sequence $\mathsf{x}[2k]$ and the odd by $\mathsf{x}[2k+1]$. The naive method would be to fully calculate $\mathsf{x}[k]$, the FFT of $x[n]$, and manually enforce $\mathsf{x}[2k+1] = 0$, but this would waste $N\log(N)/2$ operations, as the computation of $\mathsf{x}[2k+1]$ is unnecessary. It is computed and immediately nulled out.

The proposed alternative is to only calculate the $\mathsf{x}[2k]$ and interleave this sequence with zeros, and finally perform the IFFT of the resultant frequency domain sequence normally. First we demonstrate how to calculate $\mathsf{x}[2k]$ only.
We can start by simply substituting the index $2k$ for k in the DFT definition. If the DFT of $x[n]$ is

$$\mathsf{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk} \tag{5}$$

then the sequence $\mathsf{x}[2k]$ is simply given by

$$\mathsf{x}[2k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}n2k} \tag{6}$$

We can make this modified summation adhere to the usual DFT definition by making the following manipulation

$$\mathsf{x}[2k] = \sum_{n=0}^{N/2-1} (x[n] + x[n+\frac{N}{2}]) e^{-j\frac{2\pi}{N/2}nk} \tag{7}$$

Equation 6 and Equation 7 are equal, but in the latter we have expressed $\mathsf{x}[2k]$ as the $N/2$ point DFT of the time-domain sequence $x[n] + x[n+\frac{N}{2}]$.

By computing Equation 7 only, which can be computed from $x[n]$, we save half the number of operations of computing $\mathsf{x}[k]$ entirely. The time domain sequence $x[n] + x[n+\frac{N}{2}]$ can be fed into any traditional $N/2$ point FFT solver and the resulting sequence will be $\mathsf{x}[2k]$. In fact, any radix-2 decimation in frequency algorithm calculates precisely this summation in the computation of $\mathsf{x}[k]$. We have just performed the first step explicitly to avoid having the solver also compute the sequence for $\mathsf{x}[2k+1]$

Once the sequence $\mathsf{x}[2k]$ is computed, we can form a length N sequence $\tilde{\mathsf{x}}[k]$ by simply interleaving zeros in the place of $\mathsf{x}[2k+1]$. Performing the usual IFFT on this modified $\tilde{\mathsf{x}}[k]$ will give the time domain sequence of the desired output signal. That is, it will be filtered properly. Furthermore, since we are interleaving zeros, the IFFT converting back into the time domain will also perform $\frac{N}{2}\log(N)$ multiplications by zero, which should be avoided via some type of compiler optimization.

In summary, if we compute the FFT using this modification, we can save up to $N\log(N)$ multiply add operations, half of the savings coming from the FFT, and half coming from the avoiding IFFT multiplications by zero.

$\ell = 1$ Case

For $\ell = 1$ things are a little bit more complicated, but follow as a generalization of the above. If we recreate our sets $\mathcal{X}_{\mathsf{stop}}$ and $\mathcal{X}_{\mathsf{pass}}$ we will have either

$$\begin{aligned} \mathcal{X}_{\mathsf{stop}} &= \{\ldots \mathsf{x}[0], \mathsf{x}[1], \mathsf{x}[4], \mathsf{x}[5] \ldots\} \\ \mathcal{X}_{\mathsf{pass}} &= \{\ldots \mathsf{x}[-2], \mathsf{x}[-1], \mathsf{x}[2], \mathsf{x}[3] \ldots\} \end{aligned} \tag{8}$$

or

$$\begin{aligned} \mathcal{X}_{\mathsf{stop}} &= \{\ldots \mathsf{x}[-2], \mathsf{x}[-1], \mathsf{x}[2], \mathsf{x}[3] \ldots\} \\ \mathcal{X}_{\mathsf{pass}} &= \{\ldots \mathsf{x}[0], \mathsf{x}[1], \mathsf{x}[4], \mathsf{x}[5] \ldots\} \end{aligned} \tag{9}$$

If we assume the case of the two sets given as (8), then our task is to compute the second sequence, $\mathcal{X}_{\mathsf{pass}}$, only and interleave it with zeros in the place of members of $\mathcal{X}_{\mathsf{stop}}$. To define $\mathcal{X}_{\mathsf{pass}}$ terms of time-domain sequences, first reconsider the odd/even frequency domain sequences $\mathsf{x}[2k] = \{\ldots \mathsf{x}[-2], \mathsf{x}[0], \mathsf{x}[2] \ldots\}$ and $\mathsf{x}[2k+1] = \{\ldots \mathsf{x}[-3], \mathsf{x}[-1], \mathsf{x}[1] \ldots\}$. We can further index these two sequences into their respective odd and even-indexed members. For example, splitting $\mathsf{x}[2k]$ into even and odd indices respectively gives

$$\begin{aligned} \mathsf{x}[4k] &= \{\ldots \mathsf{x}[-4], \mathsf{x}[0], \mathsf{x}[4] \ldots\} \\ \mathsf{x}[4k+2] &= \{\ldots \mathsf{x}[-2], \mathsf{x}[2], \mathsf{x}[6] \ldots\} \end{aligned} \tag{10}$$

Correspondingly, $\mathsf{x}[2k+1]$ can be segmented into its even and odd indexed members (given respectively below)

$$\begin{aligned} \mathsf{x}[4k+1] &= \{\ldots \mathsf{x}[-3], \mathsf{x}[1], \mathsf{x}[5] \ldots\} \\ \mathsf{x}[4k+3] &= \{\ldots \mathsf{x}[-5], \mathsf{x}[-1], \mathsf{x}[3] \ldots\} \end{aligned} \tag{11}$$

Comparing with the desired sequences in 8, it's clear that we wish to compute only $\mathsf{x}[4k+2]$ and $\mathsf{x}[4k+3]$ and not the other two sequences since they will be zeroed out.

How can we achieve this computation? First define $y_1[n]$ to be

$$y_1[n] = x[n] + x[n + \frac{N}{2}] \tag{12}$$

which is the time-domain sequence whose DFT gives $\mathsf{x}[2k]$. We express the sequence $\mathsf{x}[4k+2]$ as a partial FFT of $y_1[n]$ just as we expressed $\mathsf{x}[2k]$ as a partial FFT of $x[n]$ earlier. However, this time note that $\mathsf{x}[4k+2]$ constitutes the odd-indexed values of $\mathsf{x}[2k]$, not the even ones. Performing the calculation carefully for odd-indexed sequences, we obtain the result below. Notice the crucial differences with the expression for even indexed sequences.

$$\mathsf{x}[4k+2] = \sum_{n=0}^{N/4-1} ((y_1[n] - y_1[n+\frac{N}{4}]) \cdot e^{-j\frac{2\pi}{N/2}n}) e^{-j\frac{2\pi}{N/4}nk} \tag{13}$$

The second sequence required to compute $\mathcal{X}_{\mathsf{pass}}$ as defined in (8) is $\mathsf{x}[4k+3]$. This sequence is formed as the odd-indexed sequences of $\mathsf{x}[2k+1]$. Thus we first define $\mathsf{x}[2k+1]$ in its full form. We have

$$\mathsf{x}[2k+1] = \sum_{n=0}^{N/2-1} ((x[n] - x[n+\frac{N}{2}]) \cdot e^{j\frac{2\pi}{N}n}) e^{-j\frac{2\pi}{N/2}kn} \tag{14}$$

Letting $y_2[n] = (x[n] - x[n+\frac{N}{2}]) \cdot e^{j\frac{2\pi}{N}n}$, the sequence $\mathsf{x}[4k+3]$ is the DFT of the odd-indexed members of $y_2[n]$ i.e.

$$\mathsf{x}[4k+3] = \sum_{n=0}^{N/4-1} ((y_2[n] - y_2[n+\frac{N}{4}]) \cdot e^{j\frac{2\pi}{N/2}n}) e^{-j\frac{2\pi}{N/4}nk} \tag{15}$$

Given the two sequences $\mathsf{x}[4k+2]$ and $\mathsf{x}[4k+3]$ computed as above, we can rebuild a new frequency domain sequence $\tilde{\mathsf{x}}[k]$ with interleaved zeros in place of $\mathsf{x}[4k]$ and $\mathsf{x}[4k+1]$ and again perform the IFFT.

Cases $\ell > 1$

The cases beyond this follow similarly. For example, $\ell = 2$ will require computation of four of the eight possible $\mathsf{x}[8k+j]$ $(0 \leq j \leq 7)$ sequences.

Implementation Details

As previously mentioned, the radix-2 decimation in frequency FFT calculates the full FFT by recursively computing the above sequences. Though the radix-2 FFT lends itself to this recursive implementation, it would be equally effective, and more readable in the beginning, to write a function that takes in the qubit number and computes the appropriate sequences non-recursively. Standard FFT algorithms can still be used to compute the partial FFT sequences. For example, projections on qubit 0 will always entail a computation of $\mathsf{x}[2k]$ and hence, the code can first compute $x[n] + x[n+\frac{N}{2}]$ and feed this into a traditional FFT algorithm. If the target qubit $q = 1$, the code can first compute $x[n] + x[n+\frac{N}{2}]$ and then compute a second sequence based on that, as outlined above. Then this second sequence which will have 1/4 the original length can be used with an FFT solver to compute, e.g. $\mathsf{x}[4k]$.