# Lab 2

## Spectrum Analysis

In the study of communication circuits and systems, frequency domain analysis techniques play a key role. The communication systems engineer deals with signal bandwidths and signal locations in the frequency domain. The tools for this analysis are Fourier series and Fourier transforms. The types of signals encountered can range from simple deterministic signals to very complex random signals. In this experiment we mostly consider deterministic signals which are also periodic. The Keysight 33600A will be used at the end of this experiment to generate more random like waveforms.

## Background

### • Power Spectral Density of Periodic Signals

In this background section we would like to connect the Fourier theory for periodic signals to the measurements made by a spectrum analyzer. We are not only interested in the location of spectral lines, but also the amplitudes in dBm. Formally, the frequency domain representation of periodic signals can be obtained using the complex exponential Fourier series. If $x(t)$ is periodic with period $T_s$, that is $x(t) = x(t + kT_s)$ for any integer $k$, then

$$x(t) = \sum_{n=-\infty}^{\infty} X_n e^{j2\pi n f_s t} \tag{1}$$

where

$$X_n = \frac{1}{T_s} \int_{T_0} x(t) e^{-j2\pi n f_s t} dt \tag{2}$$

and $f_s = 1/T_s$. By Fourier transforming (1) term–by–term using $\mathcal{F}\{e^{j2\pi f_0 t}\} = \delta(f - f_0)$, we can write

$$X(f) = \sum_{n=-\infty}^{\infty} X_n \delta(f - n f_s). \tag{3}$$

The Fourier transform of $x(t)$ in terms of the Fourier series coefficients is good starting point, but there is a more convenient way of obtaining the $X_n$ coefficients via the Fourier transform of one period of $x(t)$.

An alternate approach is to obtaining the Fourier transform of $x(t)$ is to write

$$x(t) = \sum_{m=-\infty}^{\infty} p(t - mT_s), \tag{4}$$

where $p(t)$ is one period of $x(t)$, i.e.,

$$p(t) = \begin{cases} x(t), & |t| \leq T_o/2 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Using the Fourier transform pair

$$\sum_{m=-\infty}^{\infty} p(t - mT_s) \Leftrightarrow f_s \sum_{n=-\infty}^{\infty} P(nf_s)\delta(f - nf_s) \tag{6}$$

where

$$P(f) = \mathcal{F}\{p(t)\}, \tag{7}$$

we have

$$X(f) = \sum_{n=-\infty}^{\infty} f_s P(nf_s)\, \delta(f - nf_s). \tag{8}$$

If we now equate (3) and (8) we see that the Fourier coefficients are simply given by

$$X_n = f_s\, P(nf_s), \tag{9}$$

which will be very useful in calculating the power spectrum of the periodic signals considered in this lab. Since the spectrum analyzer measures the power spectral density (PSD) of a signal $x(t)$, we now have to connect the Fourier transform results with the PSD theory.

For power signal $x(t)$, the PSD, $S_x(f)$, gives the distribution of power in $x(t)$ versus frequency. Since $S_x(f)$ is a density, it has units of W/Hz. Formally the power spectral density is defined as

$$S_x(f) = \mathcal{F}\{R_x(\tau)\} = \int_{-\infty}^{\infty} R_x(\tau) e^{-j2\pi f\tau} d\tau, \tag{10}$$

where $R_x(\tau)$ is the autocorrelation function of the signal $x(t)$. The autocorrelation function is computed as a time average for deterministic signals, and a statistical average for random signals. The time average
definition of $R_x(\tau)$, when $x(t)$ is periodic with period $T_s$, is

$$R_x(\tau) = \frac{1}{T_s} \int_{T_s} x(t)x(t + \tau) dt \tag{11}$$

Since $x(t)$ is periodic, it follows that $R_x(\tau)$ is also periodic, which implies that $S_x(f)$ will contain impulses. For $x(t)$ real and periodic it is a simple matter to show that $R_x(\tau)$ and $S_x(f)$ can be written in terms of the Fourier coefficients corresponding to $x(t)$, that is

$$R_x(\tau) = \sum_{n=-\infty}^{\infty} |X_n|^2 e^{j2\pi n f_s \tau} = \sum_{n=-\infty}^{\infty} |f_s P(n f_s)|^2 e^{j2\pi n f_s \tau} \tag{12}$$

and

$$S_x(f) = \sum_{n=-\infty}^{\infty} |X_n|^2 \delta(f - n f_s) = \sum_{n=-\infty}^{\infty} |f_s P(n f_s)|^2 \delta(f - n f_s) \tag{13}$$

Note that when the PSD consists of spectral lines, the units associated with the spectral lines is actually power in W, not W/Hz. We can now relate the PSD theory to the actual PSD measurement in a $Z_0 = R_0 = 50$ ohm environment. Here we assume $x(t)$ is the voltage of a $Z_0 = 50$ ohm source (Keysight 33600 function generator), and the load is the spectrum analyzer, which has input impedance $Z_0 = 50$ ohms. The theoretical power of each spectral line is

$$\frac{|X_n|^2}{50} = \frac{|f_0 P(n f_s)|^2}{50} \text{ Watts (per one-sided spectral line)} \tag{14}$$

at $f = n f_s$ Hz. Be careful here making sure $x(t)$ is indeed the voltage waveform across the 50 ohm load driven from a 50 ohm source. This voltage is one half the open circuit voltage. Note: The Keysight 33600 generators, when in 50 ohm source mode, actually display the voltage across a 50 ohm load impedance, so there is no additional scaling factor needed. Relative to the open circuit voltage a factor of 4 is needed.  Converting to dBm would be the next step to relate theory to measurement on the spectrum analyzer. One additional note however, is that the theoretical PSD we have described is two–sided, while the spectrum analyzer displays only the one–sided spectrum. For the case of real signals we know that the PSD is an even function of frequency.  The power values of the spectrum need to be doubled when relating to the single–sided display, or in dBm, we add 3 dB to the theoretical value, i.e.,

$$P_{\text{SA Theory, dBm}} = 10 \log \left[ 2 \cdot \frac{|f_s P(n f_s)|^2}{50} \right] + 30 \text{ dBm} \tag{15}$$

where $P(f) = \mathcal{F}\{p(t)\}$, the amplitude of $p(t)$ is the voltage across the input, and $f_0 = 1/T_0$ is the fundamental frequency of the generated waveform.

## Pseudo–Noise Sequence Generators

In the testing and evaluation of digital communication systems a source of *random like* binary data is required. A maximal–length sequence generator or pseudo–noise (PN) or *pseudo–random bit sequence* (PRBS) on the Keysight 33600, is a code often used for this purpose. A generator of this type produces a sequences of binary digits that are periodic. For a listing of the properties of maximal–length sequences see the Ziemer and Peterson, *Digital Communications and Spread Spectrum Systems*, Macmillian Publishing Co., New York, 1985. Typical uses of PN sequences are:

- *Encipherment*: A message written in binary digits may be added modulo–2 (exclusive–OR) to a PN sequence acting as a key. The decipherment process consists of adding the same PN sequence, synchronized with the first, modulo–2 to the encoded message. Such a technique is a form of scrambling.
- *Randomizer*: A PN sequence can also be used for breaking up long sequences of 1's or 0's which may bias a communications channel in such a way that performance of the communications system is degraded. The PN sequence is referred to as a *randomizer*. Note that the scrambler mentioned above and the randomizer perform the same function, but for different purposes.
- Testing the performance of a data communication system.
- Code generator.
- Prescribed Sequence Generator, such as needed for word or frame synchronization.

Practical implementation of a PN code generator can be accomplished using an $N$–stage shift register with appropriate feedback connections. An $N$–stage shift register is a device consisting of $N$ consecutive binary storage positions, typically D–type flip–flops, which shift their contents to the next element down the line each clock cycle. With the ready availability of programmable logic devices, it is also possible to implement the generator as a software abstraction, e.g., assembly, C/C++, or Verilog/VHDL code (more on this later). Without feedback the shift register would be empty after $N$ clock cycles. A general feedback configuration is shown in Figure [N_stage_shift] with the input to the first stage being a logical function of the outputs of all $N$ stages. The resulting output sequence will always be periodic with some period $M$. The maximum sequence period is

$$M = 2^N - 1 \tag{16}$$

The feedback arrangement which achieves the maximum sequence period results in a *maximum length sequence* or *m–sequence*. The block diagram and output sequence for a three stage m–sequence generator are shown in Figure [ThreeStage]. The logic levels here are assumed to be bipolar ($\pm A$). In practice the actual logic levels will correspond to the device family levels. Possible exclusive–OR feedback connections for m–sequences with $N = 1$ through $N = 15$ are given in Table 1.

The power spectrum $S_x(f)$ of the $m$–sequence generator output can be found from the autocorrelation function $R_x(\tau)$. By definition we can write

$$R_x(\tau) = \frac{1}{MT} \int_{-MT/2}^{MT/2} x(t)x(t+\tau)\, dt \tag{17}$$

where $T$ is the clock period. For long sequences the above integral is difficult to evaluate.

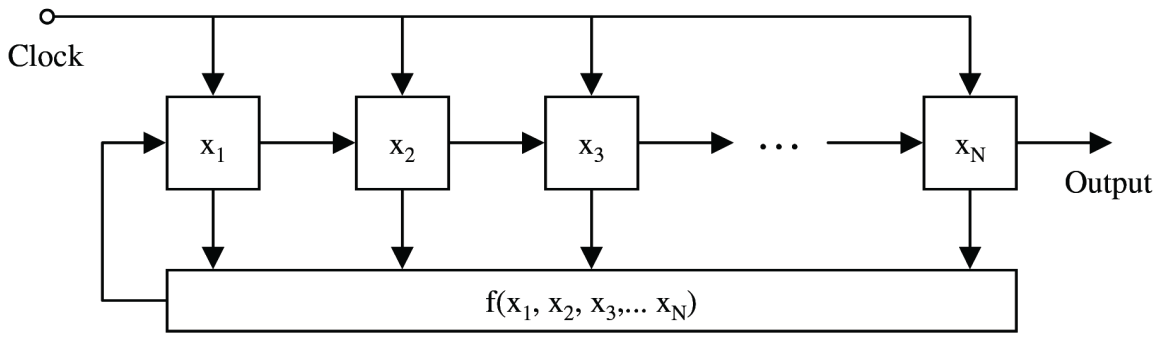



Figure 1: $N$–stage linear feedback shift register (LFSR).

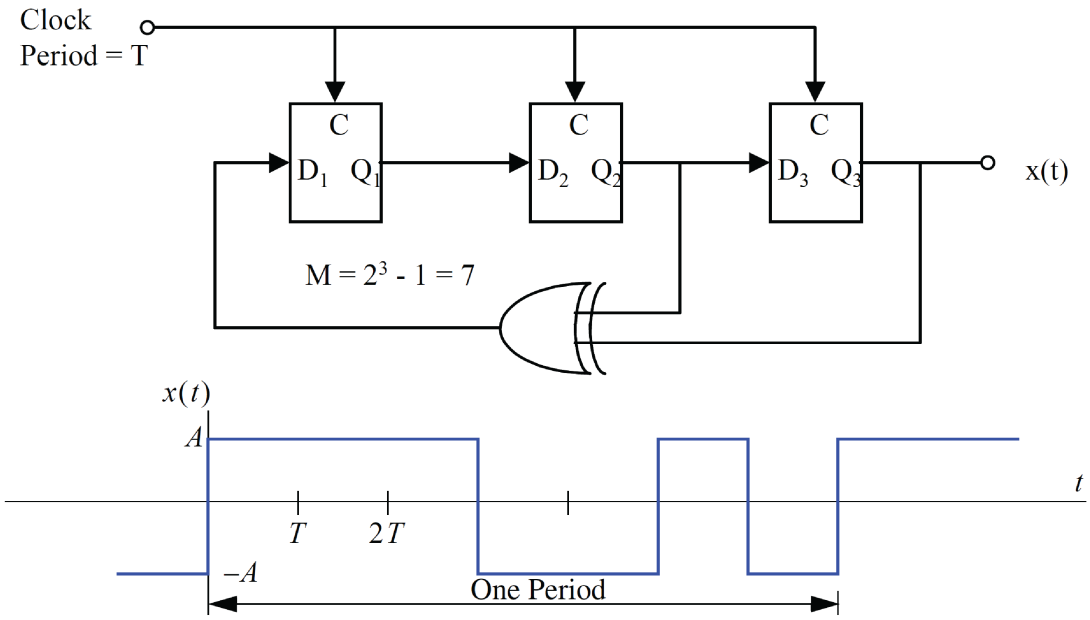



Figure 2: Three stage, length $M = 7$ $m$–Sequence Generator}

When writing and $m$–sequence generator from scratch, it is important to know the shift register (SR) tap connections that yield the maximal length. Table 1 provides these connections for 2–15 stages.

Table 1: $m$–Sequence Generator Feedback Connections for 2 – 15 stages.

| Stages | Connections | Stages | Connections |
|---|---|---|---|
| 2 | $Q_1 \oplus Q_2$ | 9 | $Q_5 \oplus Q_9$ |
| 3 | $Q_2 \oplus Q_3$ | 10 | $Q_7 \oplus Q_{10}$ |
| 4 | $Q_3 \oplus Q_4$ | 11 | $Q_9 \oplus Q_{11}$ |
| 5 | $Q_3 \oplus Q_5$ | 12 | $Q_2 \oplus Q_{10} \oplus Q_{11} \oplus Q_{12}$ |
| 6 | $Q_5 \oplus Q_6$ | 13 | $Q_1 \oplus Q_{11} \oplus Q_{12} \oplus Q_{13}$ |
| 7 | $Q_6 \oplus Q_7$ | 14 | $Q_2 \oplus Q_{12} \oplus Q_{13} \oplus Q_{14}$ |
| 8 | $Q_2 \oplus Q_3 \oplus Q_4 \oplus Q_8$ | 15 | $Q_{14} \oplus Q_{15}$ |

An example of how to compute $R_x(\tau)$ for $M = 7$ ($N = 3$) is given below. For this example a $0$ is used in place of a $-1$ for the bit $0$, but multiplication is defined as $0 \times 1 = 1 \times 0 = -1$ and $0 \times 0 = 1 \times 1 = 1$, which is a bipolar AND operation. We will find the autocorrelation function at discrete times $\tau = kT$ using the normalized correlation

$$R_x(kT) = \frac{1}{M} \sum_{i=1}^{M} \rho_i \tag{18}$$

where $\rho_i$ is the product of a bit in the sequence with a corresponding bit in a shifted version of the sequence. The calculation of $R_x(kT)$ for $k = 0, 1, \text{and } 2$ is shown in Figure 3. The continuous autocorrelation function follows by connecting the discrete time points together as shown in Figure 3.

| | | ← one period → | |
|---|---|---|---|
| Sequence | 0 1 1 1 | 0 0 1 0 1 1 1 | 0 0 1 |
| Sequence Repeated | 0 1 1 1 | 0 0 1 0 1 1 1 | 0 0 1 |
| Correlation | | 1 1 1 1 1 1 1 | |
| | | $R_x(0) = 1$ | |
| Sequence | 0 1 1 1 | 0 0 1 0 1 1 1 | 0 0 1 |
| Seq. Shifted Once | 1 0 1 1 | 1 0 0 1 0 1 1 | 1 0 0 |
| Correlation | | −1 1 −1 −1 −1 1 1 | |
| | | $R_x(T) = -1/7$ | |
| Sequence | 0 1 1 1 | 0 0 1 0 1 1 1 | 0 0 1 |
| Seq. Shifted Twice | 1 1 0 1 | 1 1 0 0 1 0 1 | 1 1 0 |
| Correlation | | −1 −1 −1 1 1 −1 1 | |
| | | $R_x(2T) = -1/7$ | |
| Sequence | 0 1 1 1 | 0 0 1 0 1 1 1 | 0 0 1 |
| Seq. Shifted Three | 1 1 0 0 | 1 0 1 1 1 0 0 | 1 0 1 |
| Correlation | | −1 1 1 −1 1 −1 −1 | |
| | | $R_x(3T) = -1/7$ | |

Figure 3: Development of the autocorrelation function for an $M = 7$ $m$–sequence.

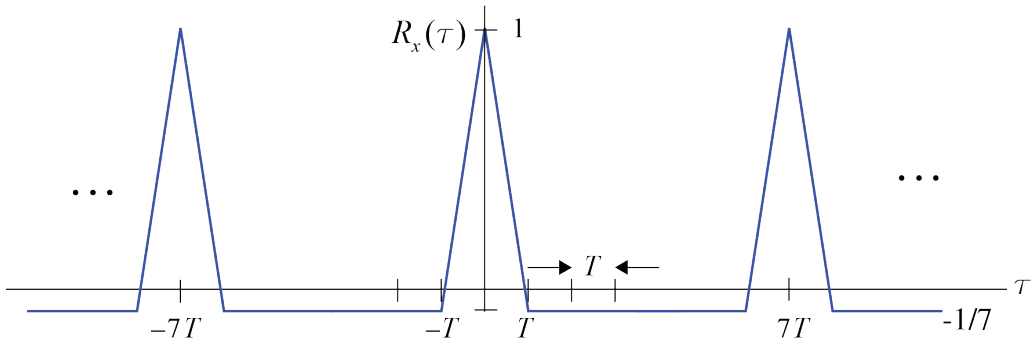



Figure 4: $M = 7$ $m$–sequence autocorrelation function.

## Logic Level Shifting

The analysis of the $m$–sequence generator in the previous section assumed logic levels of $\pm 1$, or more generally $\pm A$. In practice the logic levels produced by hardware will be different, say voltage levels of $A_0$ and $A_1$ for logic 0 and 1 respectively. At the waveform level we can model this by writing

$$y(t) = ax(t) + b, \tag{19}$$

where $a$ and $b$ are level shifting constants. Assuming $x(t)$ has levels of $\pm A$ and $y(t)$ has levels of $(A_0, A_1)$, it can be shown that

$$a = \frac{A_1 - A_0}{2A} \tag{20}$$

$$b = \frac{A_1 + A_0}{2} \tag{21}$$

The autocorrelation function and power spectral density under the transformation can also be found. Starting with the autocorrelation function

$$\begin{aligned} R_y(\tau) &= \langle y(t+\tau)y(t)\rangle \\ &= \big\langle (ax(t+\tau)+b)(ax(t)+b)\big\rangle \\ &= a^2 R_x(\tau) + 2b\langle x(t)\rangle + b^2 \end{aligned} \tag{22}$$

Notice that the last two terms are just constants, so they will only contribute to the spectral line at $f = 0$ or dc. The power spectral density will simply be

$$S_y(f) = a^2 S_x(f) + K\delta(f), \tag{23}$$

where $K = b^2 + 2b\langle x(t)\rangle$. Since the spectrum analyzer does not display the dc value, we are not too concerned with the exact value for $K$.

## Random Binary Sequence

The extreme of the $m$–sequence generator is a purely random sequence. To describe such a waveform requires the concept of a *random process*. Without going into too much detail, we can write

$$x(t) = \sum_{k=-\infty}^{\infty} a_k p(t - kT - \Delta), \tag{24}$$

where $a_k$ is a sequence of independent random variables, corresponding to an ideal *coin–flip* sequence, which takes on values of $\pm A$. The function $p(t)$ is a pulse–type waveform, e.g., rectangular of duration $T$, which is the separation between pulses, and $\Delta$ is a random variable independent of $a_k$ and uniform over the interval $(-T/2, T/2)$. Note that $X(t)$ versus $x(t)$ is used to denote the fact that we are describing a random process.

The autocorrelation function of this waveform is

$$R_X(\tau) = \sum_{k=-\infty}^{\infty} R_m r(\tau - mT), \tag{25}$$

where

$$r(\tau) = \frac{1}{T}\int_{-\infty}^{\infty} p(t+\tau)p(t)\,dt = \frac{1}{T}\big[p(\tau) * p(-\tau)\big] \tag{26}$$

is the pulse shape autocorrelation function, and

$$R_m = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{k=-N}^{N} a_k a_{k+m} \tag{27}$$

is the random coin–toss sequence autocorrelation sequence. For a true random coin–toss sequence, the flips are equally likely and independent of each other, so

$$R_m = \begin{cases} A^2, & m = 0 \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

This simplifies $R_X(\tau)$ to

$$R_X(\tau) = A^2 r(\tau). \tag{29}$$

Taking the Fourier transform we obtain the power spectrum

$$S_x(f) = A^2 \mathcal{F}\{r(\tau)\} = A^2 \mathcal{F}\left\{ \frac{1}{T} p(-\tau) * p(\tau) \right\} = A^2 \frac{|P(f)|^2}{T}, \tag{30}$$

where $P(f) = \mathcal{F}\{p(t)\}$. Due to the fact that the sequence is purely random, the power spectrum is now a continuous function of frequency, at least in this case.

For a rectangular pulse shape, which was inherent in the $m$–sequence discussion,

$$p(t) = \Pi\left( \frac{t - T/2}{T} \right) = \begin{cases} 1, & 0 \le t \le T \\ 0, & \text{otherwise} \end{cases} \tag{31}$$

and the power spectrum is thus

$$S_x(f) = A^2 T \text{sinc}^2(fT) \overset{\text{also}}{=} S_{\text{NRZ}}(f). \tag{32}$$

The NRZ notation is explained in the next section on *line codes*.

As a final note, as in the case of the $m$–sequence generator discussion, if the logic levels are shifted away from $\pm A$ we simply apply the transformation results of (23).

The Keysight 33600 does not produce pure random bit stream with its PRBS, but you can approximate it by increasing the number of shift register stages to 32. With 32 stages the code period is (using Python):

```
# Sequence period for 33600A PRBS Data = PN32
2**32 = 4294967296
```

Which is a very large period, very much like a truly random bit sequence.

## Baseband Digital Data Transmission and Line Coding

Baseband digital data transmission (see Z&T Chapter 4) refers to sending digital data over a channel that is centered in the frequency domain about DC. The physical channel here is often a cable (wire) such as twisted–pair ethernet, RS232, etc. Baseband data transmission also occurs in reading/writing to magnetic storage or free–space optical used in infrared remote controls, where the light is intensity modulated by the baseband data.

As mentioned earlier, an $m$–sequence can be used as a test data source in place of purely random data that may actually be sent over the channel. The format of the data may not always involve a rectangular pulse shape. The subject of *line coding* deals with the selection of pulse shapes and/or coding schemes, that alter the power spectral density from what might be obtained from the simple rectangular pulse shape. A portion of Z&T Chapter 4 is devoted to line codes and their power spectra. Figure 5 provides an example of six different line codes. In this lab we are only interested in *NRZ change* also known as NRZ (non–return–to–zero) and *split–phase* also known as *Manchester* coding.

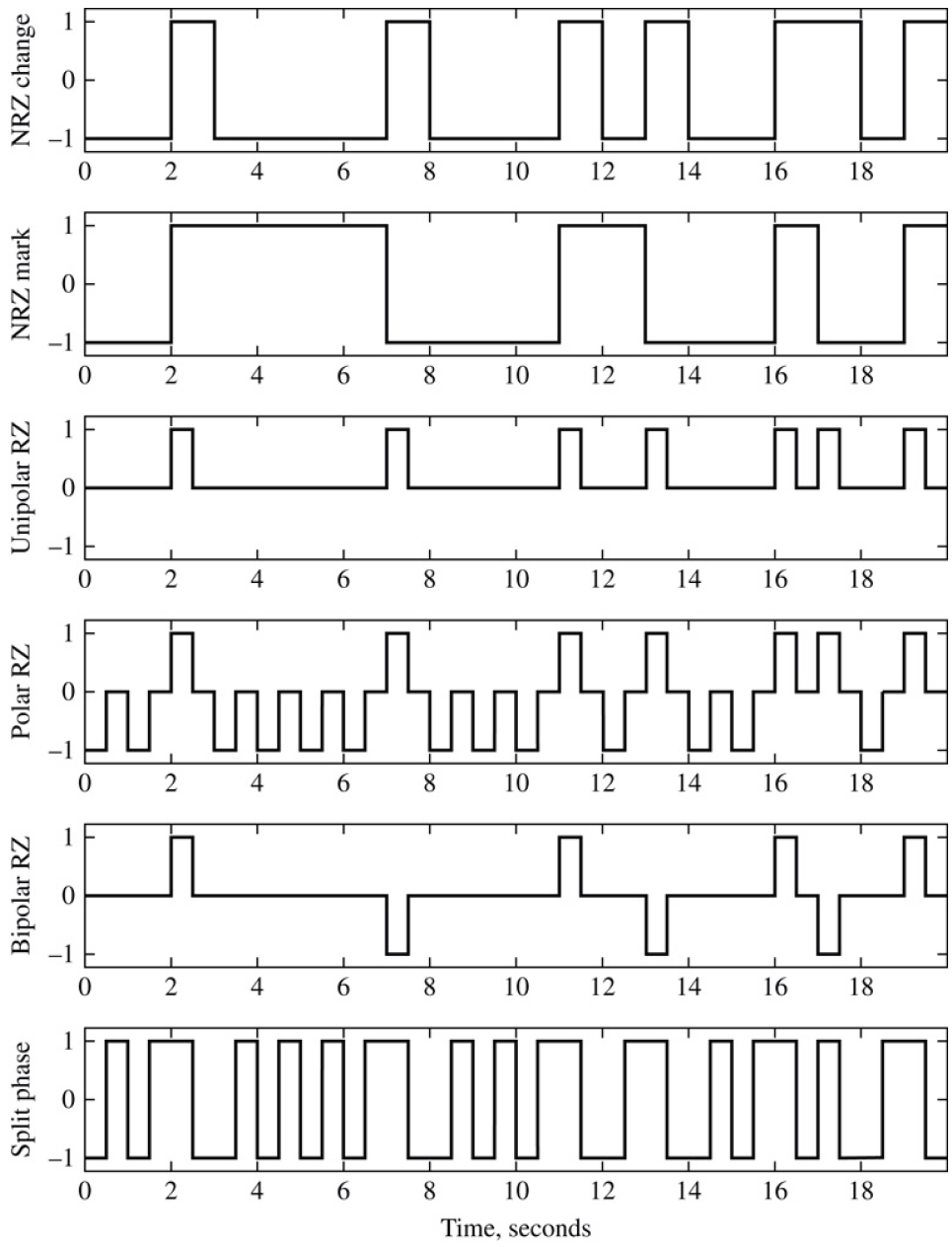



Figure 5: Binary data formats of line codes taken from Z&T p. 217.

The natural waveform produced by the $m$–sequence generator, discussed earlier, is the NRZ format, as well as the PSD of [1]. To obtain Manchester from NRZ we can multiply by a square–wave clock with period equal to the data bit duration, $T$. The corresponding pulse shape for Manchester is (Z&T Example 4.2)

$$p(t) = \Pi\left(\frac{t + T/4}{T/2}\right) - \Pi\left(\frac{t - T/4}{T/2}\right). \tag{33}$$

We find the PSD by first finding $P(f)$

$$P(f) = jT\text{sinc}\left(\frac{T}{2}f\right)\sin\left(\frac{\pi T}{2}f\right). \tag{34}$$

Since $S_r(f) = |P(f)|^2/T$, we can write that

$$S_{\text{MAN}}(f) = A^2 T \text{sinc}^2\left(\frac{T}{2}f\right)\sin^2\left(\frac{\pi T}{2}f\right) \tag{35}$$

Since the 33600A does not natively contain Manchester line coding, you will be creating and then loading one period of an $m$–sequence as an *arbitrary waveform* (ARB) via a CSV file created in the Jupyter notebook sample. A companion NRZ waveform will also be created for comparison purposes.

## Preliminary Analysis

### Problem 1

Find the power spectral density of a periodic pulse train signal, period $T_s$, shown in Figure 6.

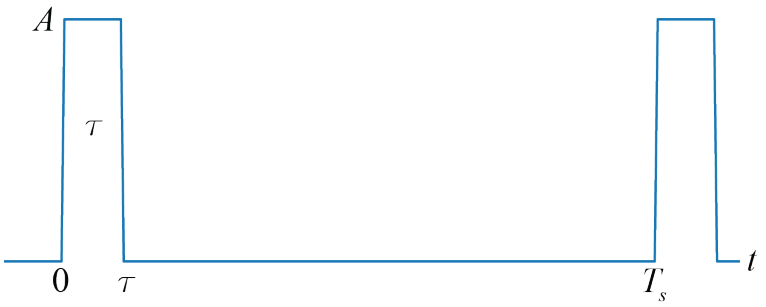



Figure 6: Pulse train signal having pulse period $T_s$, pulse width $\tau$, and having peak amplitude of $A$.

$$x(t) = \sum_{m=-\infty}^{\infty} A\,\Pi\left(\frac{t - mT_s - \tau/2}{\tau}\right),\ 0 < \tau < T_s \tag{36}$$

Assume that the waveform is delivered to a $R_0 = 50$ ohm load. The recommended approach is to utilize equations (6), (9), and finally (13). The (14) gets the scaling relative to an $R_0$ load, i.e.,

$$p(t) \to P(f) \to |X_n|^2 = |f_s P(nf_s)|^2 \text{ in } S_x(f) = \sum_{n=-\infty}^{\infty} |X_n|^2\,\delta(f - nf_s) \tag{37}$$

In the laboratory exercises portion of this lab you will be comparing theory to measurement. The Jupyter notebook sample contains the helpful function `line_spectra_dBm()` for creating single–sided spectral plots in dBm.

```
def line_spectra_dBm(fn,Xn,floor_dBm = -60,R0 = 50):
    """
    Plot the one-sided line spectra from Fourier coefficients in dBm.

    Inputs
    ------
    fn = harmonic frequencies correspinding to Xn's
```

```
    Xn = pulse train FS coefficients
    floor_dBm = spectrum floor in dBm
    R0 = impedance (default 50 ohms)

    Returns
    -------
    Sx_dBm = One-sided bandpass lines as dBm levels

    Mark Wickert February 2019
    """
    Xn_dBm = abs(Xn*sqrt(1000/(2*R0)))
    Xn_dBm[0] = abs(Xn[0])*sqrt(1000/(R0))

    ss.line_spectra(fn,Xn_dBm,mode='magdB',sides=1,floor_dB=floor_dBm)
    ylabel(r'Power (dBm)')
    xlabel(r'Frequency (Hz)')
    title(r'Line Spectra PSD')
    Sx_dBm = 20*log10(2*Xn_dBm)
    Sx_dBm[0] -= 6.02
    return Sx_dBm
```

Note under the hood `line_spectra_dBm()` uses scikit-dsp-comm function `sigsys.line_spectra()` with some scaling so that amplitude spectra in dB in converted to dBm into an `R0` ohm load.

Example

```
n = arange(0,50)
Ts = 1e-6
tau = 100e-9
A = 0.1
Xn = A*tau/T0*sinc(n*tau/T0)*exp(-1j*2*pi*n*tau/2/T0)
```

```
Xn_dBm = line_spectra_dBm(n/Ts/1e6,Xn,floor_dBm = -60)
title(r'Theory 1 MHz Pulse Train, $\tau = 100$ns, $A = 100$mV')
xlabel(r'Frequency (MHz)');
xlim([0,50]);
```

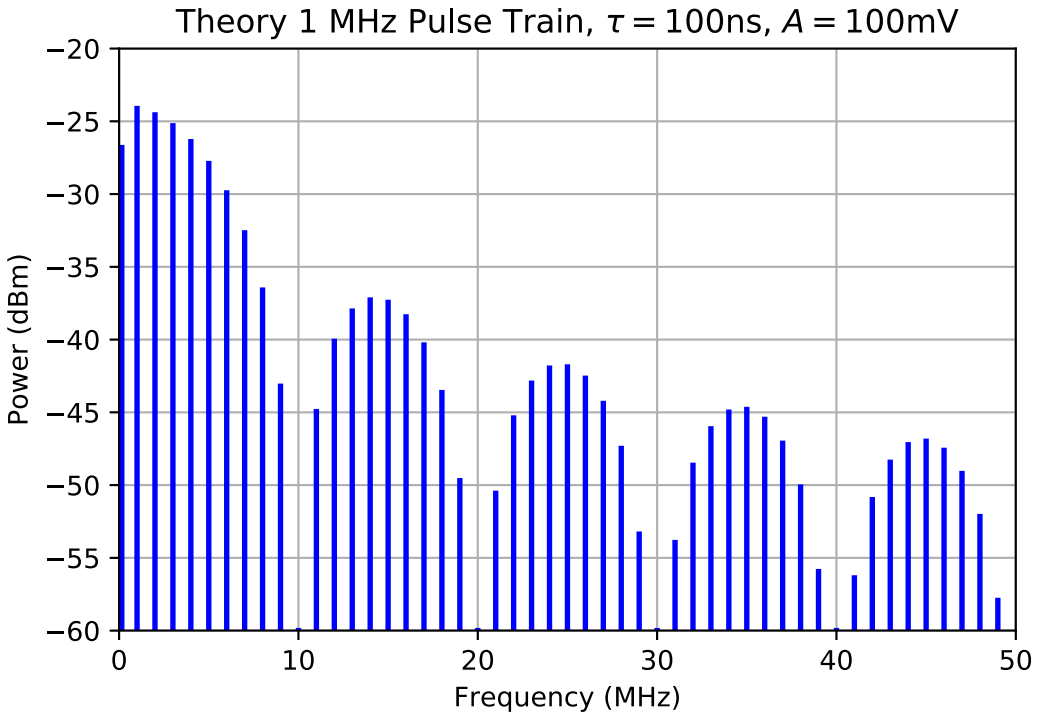



Figure 7: Spectra output from `line_spectra_dBm()` for a 1 MHz pulse train.

## Problem 2

Find the power spectral density of a coherent co–sinusoidal pulse train signal, period $T_s$, shown in Figure 7.

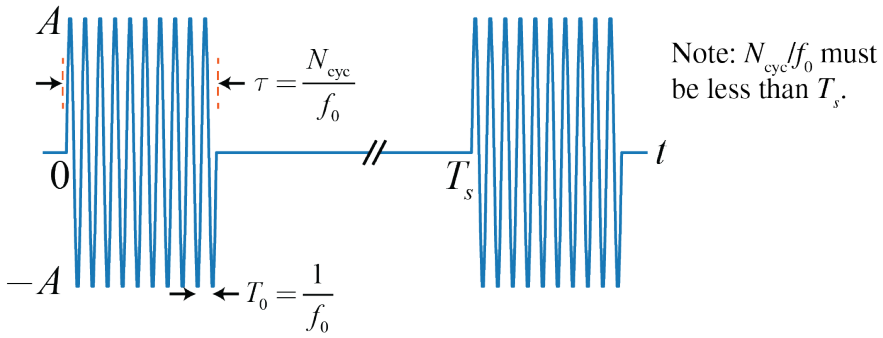



Figure 8: Coherent co–sinusoidal pulse train signal having pulse period $T_s$, exactly $N_{\text{cyc}}$ sinusoid cycles per period, and having peak amplitude of $A$.

$$x(t) = \sum_{m=-\infty}^{\infty} A\,\Pi\left(\frac{t - mT_s - N_{\text{cyc}}/(2f_0)}{N_{\text{cyc}}/f_0}\right)\sin\left[2\pi f_0(t - mT_s) + \theta\right],\ 0 < N_{\text{cyc}}/f_0 < T_s \quad (38)$$

The waveform shown above is the coherent version of the co–sinusoidal signal. A non–coherent version is also possible by simply multiplying a pulse train signal by $\cos(2\pi f_0 t + \theta)$. With this signal configuration the number of sinusoid cycles per pulse period $T_s$ is not necessarily an integer and the phase of each burst may not be the same from cycle–to–cycle.

Coherent Co–Sinusoidal Case

Here the task is to analyze the coherent case using the same approach as outlined in Problem 1. This is the waveform generated by the 33600A when in *burst mode*. More details on setting up the generator later. To get started your initial task is to show that

$$P(f) = \frac{AN_{\text{cyc}}}{2f_0}\Big\{\text{sinc}\big[(f_0 - f)\tau\big]e^{j[\pi(f_0-f)\tau+\theta-\pi/2]} - \text{sinc}\big[(f_0 + f)\tau\big]e^{-j[\pi(f_0+f)\tau+\theta+\pi/2]}\Big\}, \quad (39)$$

In (39) it is convenient to define

$$\beta = N_{\text{cyc}}\left(1 - \frac{n}{T_s f_0}\right) \quad (40)$$

$$\alpha = N_{\text{cyc}}\left(1 + \frac{n}{T_s f_0}\right) \quad (41)$$

as substitutions for $(f_0 \mp f)\tau$, respectively, when $f \to nf_s$ and $\tau \to N_{\text{cyc}}/f_0$. It then follows that

$$X_n = f_s P(nf_s) = \frac{AN_{\text{cyc}}}{2T_s f_0}\big\{\text{sinc}(\beta)e^{j(\pi\beta+\theta-\pi/2)} - \text{sinc}(\alpha)e^{-j(\pi\alpha+\theta+\pi/2)}\big\} \quad (42)$$

Getting $S_x(f)$ is now easy to obtain. Working through the *show that* is tedious work.

In the laboratory exercises portion of this lab you will be comparing theory to measurement. The Jupyter notebook sample contains the helpful function `coherent_cosinusoid()` for calculating the Fourier coefficients. The function `line_spectra_dBm()` can again be used for plotting.

```
def coherent_cosinusoid(f0,A,Ncyc,Ts,Nstop,Nstart=0,theta=0):
    """
    Coherent cosinusoid pulse train Xn Fourier coefficients

    Inputs
    ------
        f0 = carrier frequency in Hz
         A = peak amplitude (2*A = p-p)
      Ncyc = cycle per period
        Ts = pulse train period
     Nstop = the coefficients stop index (actually one less)
    Nstart = the coefficient start index (default is 0)
     theta = starting phase (default = 0)

    Returns
    -------
    Xn = Fourier coefficients over the specified coefficient range

    Note: Ncyc*1/f0 must be less than Ts

    Mark Wickert February 2019
```

```
    """
    Xn = zeros(Nstop-Nstart,dtype = complex128)
    for n in range(Nstart,Nstop):
        beta = Ncyc*(1 - n/(Ts*f0))
        alpha = Ncyc*(1 + n/(Ts*f0))
        Xn[n-Nstart] = A*Ncyc/(2*Ts*f0)*(sinc(beta)*exp(1j*(pi*beta + theta)) \
              - sinc(alpha)*exp(-1j*(pi*alpha + theta)))
    return Xn
```

Example

```
f0 = 10.0e6
A = 0.1/2
Ncyc = 10
Ts = 10e-6
Nstart = 0
Nstop = 401
n = arange(Nstart,Nstop)
Xn_ccs = coherent_cosinusoid(f0,A,Ncyc,Ts,Nstop,Nstart)
Xn_dBm = line_spectra_dBm(n/Ts/1e6,Xn_ccs,floor_dBm = -90)
title(r'Theory 1 MHz Pulse Train, $\tau = 100$ns, $A = 100$mV')
xlabel(r'Frequency (MHz)');
ylim([-90,-10])
xlim([0,20]);
```

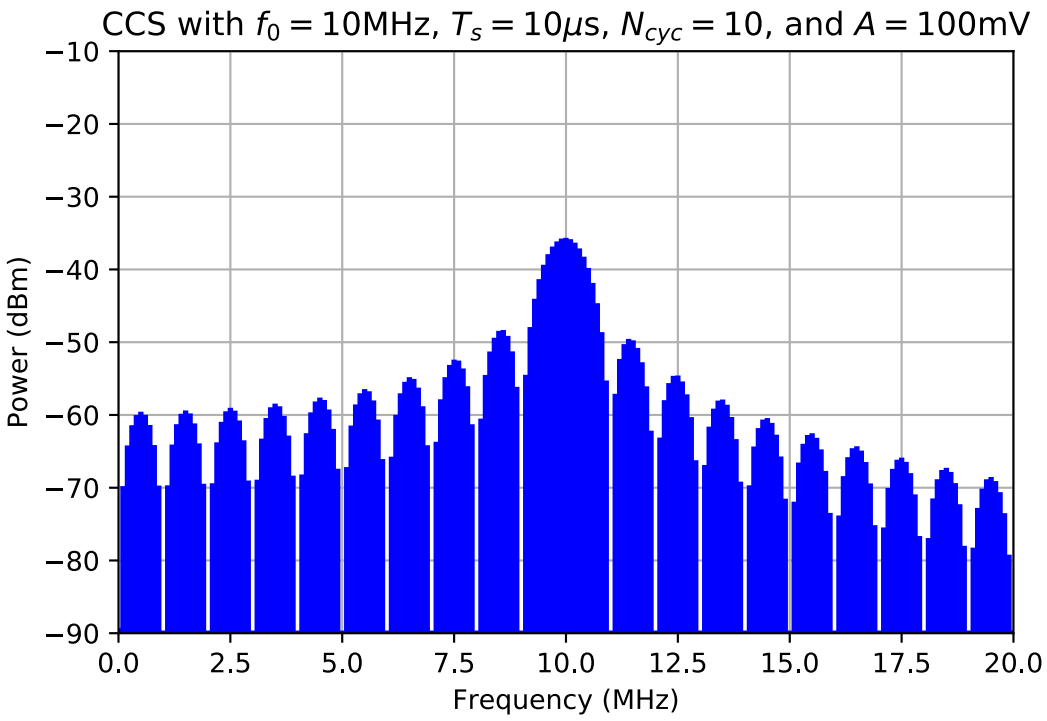


Figure 9: Theoretical spectra output from `line_spectra_dBm()` via $X_n$ obtained from `coherent_cosinusoid()` for a $T_s = 10\mu$s pulse train with $N_{\text{cyc}} = 10$ and $f_0 = 10$MHz.

Coherent and Non–Coherent Case – Alternate Approach

For the periodic pulse train of Problem 1, you can easily find the Fourier transform of the co–sinusoidal signal by multiplying the pulse train by $\cos(2\pi t + \theta)$ in the time domain, hence convolving in the frequency domain as a result of the multiplication theorem

$$X_{\text{cs}}(f) = \mathcal{F}\left\{ \sum_{m=-\infty}^{\infty} A\,\Pi\left( \frac{t - mT_s - \tau/2}{\tau} \right) \right\} * \mathcal{F}\left\{ \cos(2\pi f_0 t + \theta) \right\} \tag{43}$$

The details quickly lead to

$$X_{\text{cs}}(f) = \frac{A}{2} \sum_{n=-\infty}^{\infty} X_n e^{j\theta}\, \delta(f - nf_s - f_0) + \frac{A}{2} \sum_{n=-\infty}^{\infty} X_n e^{-j\theta}\, \delta(f - nf_s + f_0) \tag{44}$$

where $X_n = f_s P(nf_s)$ for the standard pulse train. Since this is in the form of a line spectra, we can identify coefficients $X_{n,cs}$ from (44) directly and form the power spectrum $S_{x_{cs}}(f)$ directly from the average power in each spectral line. The $X_{n,cs}$ are not necessarily harmonically related to $f_s$ as $f_0$ are not necessarily harmonically related. Note that the positive frequency spectrum will be composed terms from both up–shifted and down–shifted pulse train spectra, unless $f_0 \gg nf_s$. For the coherent case spectral line combine according to the phasor additional formula.

The Jupyter notebook sample contains the function `line_spectra_CS_dBm()` for plotting the single–sided line spectra. The user has to supply pulse train coefficients and harmonic frequencies in arrays.

```
def line_spectra_CS_dBm(f0,fn,Xn,floor_dBm = -60,R0 = 50,lm_tol=0.001):
    """
    Plot the one-sided line spectra from Fourier coefficients in dBm for
    bandpass signals, e.g. when a carrier f0 is present. Carrier coherency is
    not required. Overlapping spectral lines folding up from f < 0 will be
    coherently combined if coherence withing line match tolerance lm_tol.

    The dBm values are also returned in an ndarray.

    Inputs
    ------
    f0 = carrier frequency
    fn = harmonic frequencies correspinding to Xn's
    Xn = pulse train FS coefficients
    floor_dBm = spectrum floor in dBm
    R0 = impedance (default 50 ohms)
    lm_tol = line spectra combining match tolerance (default 0.001 in fn units)

    Returns
```

```
    -------
    fn_BPu = Unique bandpass line frequencies
    Sx_dBm_BPu = One-sided bandpass lines as dBm levels (combined if possible)
    M_BPu = Multiplicity of line frequencies in fn_BPu (should be 1 or 2)

    Mark Wickert February 2019
    """
    Xn_dBm = abs(Xn*sqrt(1000/(2*R0)))
    #Xn_dBm[0] = abs(Xn[0])*sqrt(1000/(R0))
    fn_BP = hstack((fn[1:]+f0,array([f0]),abs(fn[1:]-f0)))
    Xn_dBm_BP = hstack((Xn_dBm[1:],array([Xn_dBm[0]]),Xn_dBm[1:]))/2
    # Find unique line frequencies as line overlap occurs near DC from
    # negative frequency spectrum. Use phasor addition to combine
    # the overlapping/like spectral lines. lm_tol sets matching tolerance.
    fn_BPu, M_BPu = signal.unique_roots(fn_BP,tol=lm_tol, rtype='avg')
    Xn_dBm_BPu = zeros(len(fn_BPu),dtype = complex128)
    for k in range(len(fn_BPu)):
        idx_fn = np.nonzero(np.ravel(abs(fn_BP - fn_BPu[k])<=lm_tol))[0]
        Xn_dBm_BPu[k] = sum(Xn_dBm_BP[idx_fn])

    ss.line_spectra(fn_BPu,Xn_dBm_BPu,mode='magdB',sides=1,floor_dB=floor_dBm)
    ylabel(r'Power (dBm)')
    xlabel(r'Frequency (Hz)')
    title(r'Line Spectra PSD')
    Sx_dBm_BPu = 20*log10(2*Xn_dBm_BPu)
    if fn_BPu[0] == 0:
        Sx_dBm_BPu[0] -= 6.02
    return fn_BPu, Sx_dBm_BPu, M_BPu
```

In order to combine terms from the negative side lapping into the positive frequency side, the Scipy function `signal.unique_roots()` is used to detect this condition and then complex add the coefficients, $X_n$ of the underlying pulse train. Note under the hood `line_spectra_cs_dBm()` uses scikit-dsp-comm function `sigsys.line_spectra()` with some scaling so that amplitude spectra in dB in converted to dBm into an `R0` ohm load.

Example

```
n = arange(0,200)
f0 = 10e6
Ts = 10e-6
tau = 10/f0 # Ncyc/fo
A = 0.1/2 # For the 33600 the sinusoial burst is typically Vp-p => Vpeak = Vp-p/2
Xn_cs = A*tau/Ts*sinc(n*tau/Ts)*exp(-1j*2*pi*n*tau/2/Ts) # Pulse train Xn's
```

```
fn_u, Sx_dBm_u, M_u = line_spectra_CS_dBm(f0/1e6,n/Ts/1e6,Xn_cs,floor_dBm = -90)
# fn
#
#
title(r'CCS with $f_0=10$MHz, $T_s=10\mu$s, $N_{cyc}=10$, and $A = 100$mV')
xlabel(r'Frequency (MHz)');
ylim([-90,-10]);
xlim([0,20]);
```

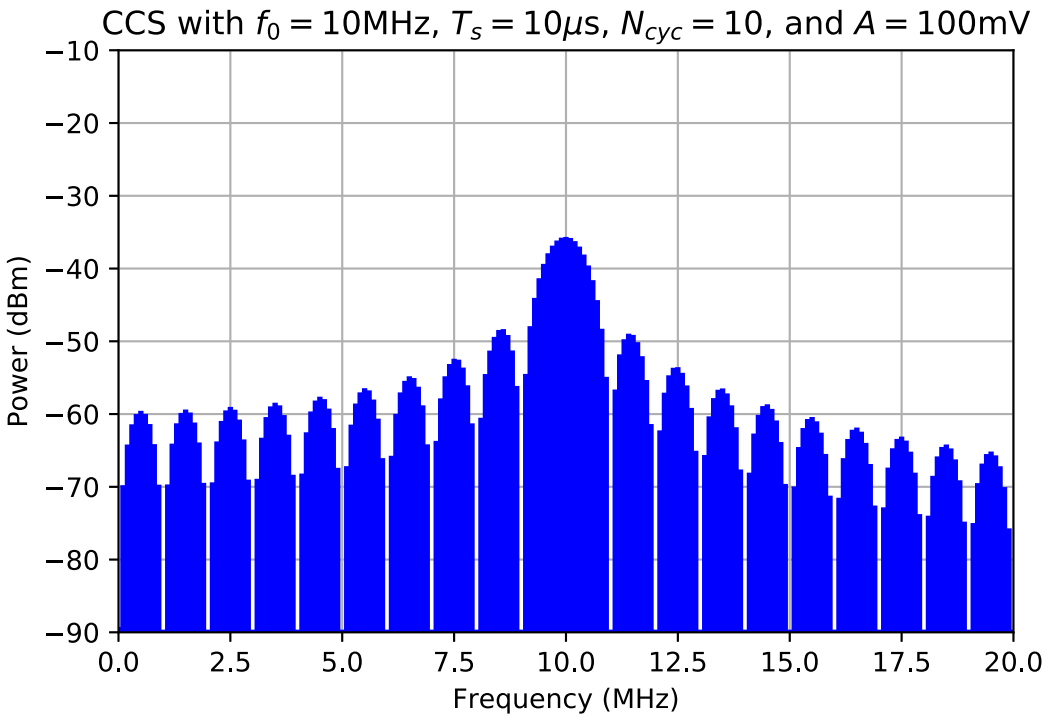



Figure 10: Theoretical spectra output from `line_spectra_cs_dBm()` for a $T_s = 10\mu$s pulse train with $N_{\text{cyc}} = 10$ and $f_0 = 10$MHz.

## Problem 3

Find the power spectral density corresponding to the periodic autocorrelation function shown in Figure 11.

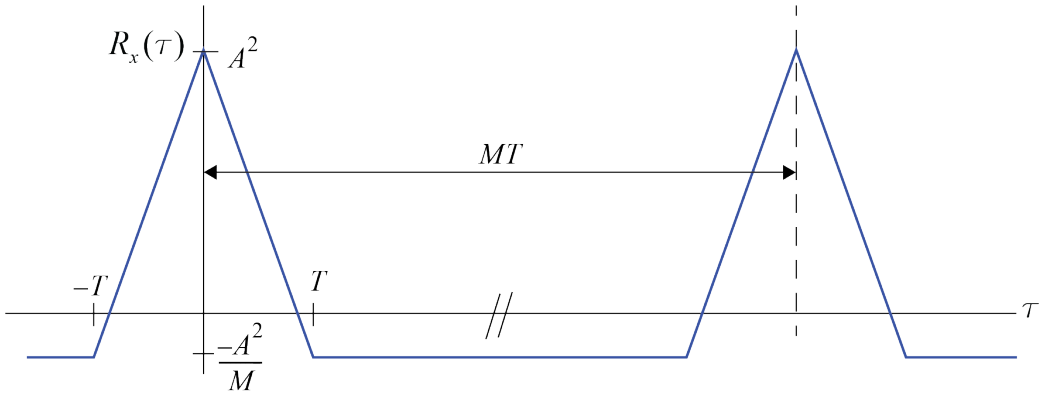



Figure 11: Periodic autocorrelation function for an m–sequence generator having period MT and $\pm A$ logic levels.

Here you take a slightly different approach to finding $S_x(f)$. Starting from the autocorrelation function is a straight shot by simply Fourier transforming $R_x(\tau)$ using (6). This may also be a homework problem assigned in Set #3 of ECE 4625. More hints will be provided for the homework problem version. Note in this problem the logic levels that rise to the above $R_x(\tau)$ are $\pm A$.

Hints:

- Using the Fourier transform pair of (6) we can obtain $S_x(f)$ by first letting $p(t)$ represent one period of $R_x(\tau)$, i.e.,

$$p(\tau) = A^2\left(1+\frac{1}{M}\right)\Lambda\left(\frac{\tau}{T}\right) - \frac{A^2}{M}\Pi\left(\frac{\tau}{MT}\right). \tag{45}$$

- Fourier transforming results in

$$P(f) = A^2T\left(1+\frac{1}{M}\right)\text{sinc}^2(fT) - A^2T\text{sinc}(fMT) \tag{46}$$

- For general logic levels, where a logic high has amplitude $A_1$ and a logical low has amplitude $A_0$, we just need to utilize the results presented the Logic Level Shifting subsection. In particular the non–DC term is scaled from the above by $a^2 = [(A_1 - A_0)/(2A)]^2$.
- Finally, you can plug into the right side of (6) except now you have directly found $S_x(f)$ By taking the Fourier transform of the autocorrelation function $R_x(\tau)$.

## Problem 4

NRZ and Manchester are two popular line coding pulse shapes used in wired and wireless digital communications. For random bit streams these schemes have power spectral density as given earlier in equations (32) and (35) respectively.

Create an overlay plot of $P_{\mathrm{NRZ}}(f)$ and $P_{\mathrm{MAN}}(f)$ in dB, for $0 \leq f \leq 4R_b$. Directly use the analytical expressions of (32) and (35) with $f/R_b$ as the normalized frequency axis. For amplitude scaling set the pulse energy equal to unity. Also note that $T = T_b = 1/R_b$, the bit rate, and for a PSD in dB you use $10\log_{10}(\ )$.

# Laboratory Exercises

## Periodic Pulse Train

### Part a

Using one channel of the Keysight 33600A generator record time domain and frequency domain data using the Agilent MSO–X–6004A oscilloscope and the Keysight N9914A FieldFox spectrum analyzer respectively, for several different duty cycles $\tau/T_s$. As a matter of convenience select $T_s$ as an integer number of graticule lines on the oscilloscope screen.

On the front panel of the 33600A choose Waveforms and then choosing `Pulse` puts the generator in pulse train mode. Set the amplitude to 100 mV p–p into 50 ohms. From there you will see all of the relevant settings. Most important at this time are: frequency, amplitude, pulse width,lead edge, and trail edge. Figure 12 describes how to configure the 33600A.

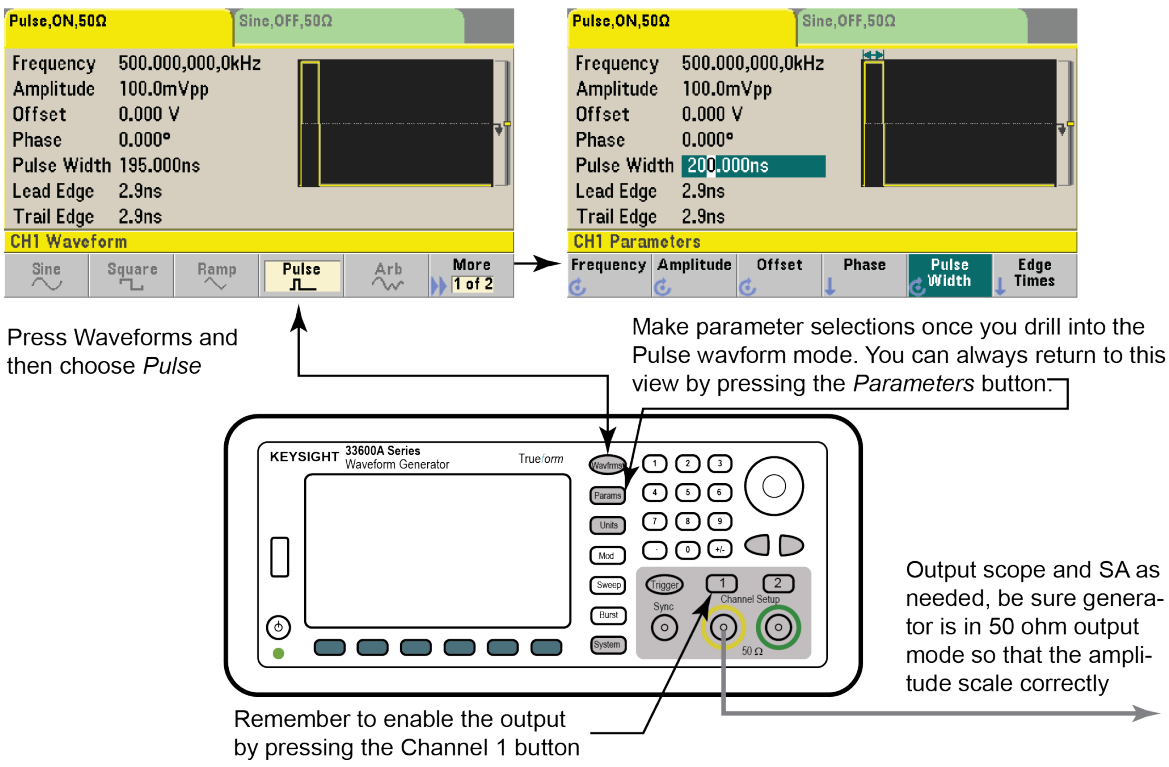



Figure 12: 33600A settings required to generate a pulse train.

For the initial configuration set the pulse repetition frequency, $f_s$, to 500 kHz, making the pulse train period $T_s = 1/f_s = 2\mu$s. Then choose a pulse width $\tau = 200$ ns. What is the duty factor/cycle of tis waveform?

Collect frequency domain data from the SA to include:

1. The spacing between spectral lines.
2. The location of *zeros* in the spectrum envelope.
3. The number of spectral lines between zeros.
4. Any additional data that seems appropriate for comparing the experimental power spectral density with theoretical calculations.
5. Repeat just (1) and (2) with $f_s = 100$ kHz and $\tau = 400$ ns.

For theory comparison in the Jupyter notebook, export a `.csv` file for the initial 500 kHz settings and also the 100 kHz settings of (5). An example of how to do this is included in the sample notebook. Specifically the sample Jupyter notebook provides a nice example of doing this with $f_s = 1$ MHz. Compare the spectral shape and the frequency and amplitude in dBm of the first few spectral lines. The numpy function `scipy.signal.find_peaks()` is useful for extracting peak location and amplitude from the measured data.

Import Data

```
# Skip the first 32 rows, then skip the last row that contains 'END'
f_SA, Sx_1M_10duty_100mV =
loadtxt('PULSE_1M_10duty_100mV.csv',delimiter=',',skiprows=32,
                usecols=(0,1),comments='END',unpack=True)
```

Find Peaks and Corresponding Amplitudes

```
peak_idx = signal.find_peaks(Sx_1M_10duty_100mV,height=(-30,)) # peak threshold =
-30 dBm
for k, k_idx in enumerate(peak_idx[0]):
    print('Peak at %6.4f MHz of height %6.4f dBm' % \
          (f_SA[k_idx]/1e6,Sx_1M_10duty_100mV[k_idx]))
```

```
Peak at 1.0098 MHz of height -24.1698 dBm
Peak at 2.0096 MHz of height -24.6501 dBm
Peak at 3.0094 MHz of height -25.4173 dBm
Peak at 4.0092 MHz of height -26.5120 dBm
Peak at 5.0090 MHz of height -28.0286 dBm
```

The sample `.csv` is included in the Jupyter notebook ZIP package.

### Part b

The pulse rise and fall time can be altered on the 33600A by changing the *Edge Time*, which has a default value of 4 ns. Try increasing this up to 25% of the pulse width (in ns), you should now have a trapezoidal shaped pulse. Comment on the change in spectral content at high frequencies as the rise and fall times increase. Is this the expected result? For your report either import results into Jupyter and plot the `.csv` data or obtain a screenshot directly from the analyzer.

## Periodic Cosinusoidal Pulse Train

### Part a

A periodic co–sinusoidal pulse train can be generated using the Keysight 33600A in *Burst Mode*. The setup steps are described in Figure 13.

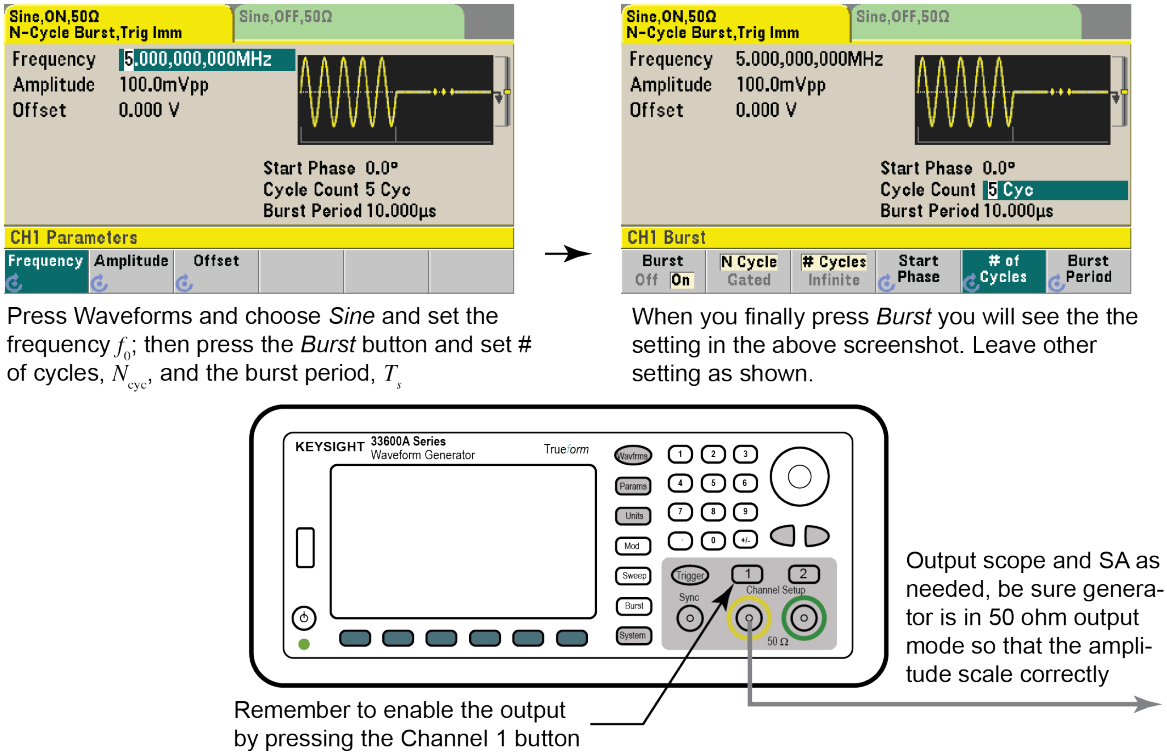



Figure 13: 33600A 33600A settings required to generate a co–sinusoidal pulse train.

To get started press the Waveform and choose *Sine*. Set the frequency to $f_0 = 5$ MHz. Then press the *Burt* button and set the number of cycles per burst, $N_{\text{cyc}} = 5$, and the burst period to $T_s = 10\mu$s.

1. Look at the waveform on the scope to verify that exactly five cycles are contained in the burst.
2. Also verify from the scope that the burst period is indeed $10\mu$s.
3. Study the spectrum and see that the spectral nulls are located as expected
4. Study the spectrum and see that the minimum line spacing is as expected.
5. Repeat steps 1–4 with $N_{\text{cyc}} = 2$.

6. Repeat steps 1–4 with $N_{\text{cyc}} = 10$.

Import one of the three $N_{\text{cyc}}$ cases into the Jupyter notebook via a `.csv` export from the N9914A. Compare the theoretical expectations to the measured using the Python functions found in the sample notebook. Use a frequency span of 1 kHz to 10 MHz. To plot the theoretical power spectrum use the function pair

```
Xn_ccs = coherent_cosinusoid(f0,A,Ncyc,Ts,Nstop,Nstart)
Xn_dBm = line_spectra_dBm(n/Ts/1e6,Xn_ccs,floor_dBm = -90)
```

as described in the *Preliminary Analysis* section of this document.

### Part b

Return $N_{\text{cyc}} = 5$ and experiment with changes to $f_0$. Specifically move $f_0$ down to 4 MHz and then up to 6 MHz. Compare the experimental spectrum captured via a `.csv` with the theoretical expectations for each $f_0$ value.

1. Note in particular that the spectrum peak shifts accordingly.
2. Check to see that the pulse train characteristics of spectral nulls and minimum line spacing remain invariant under changes in $f_0$.

### Part c

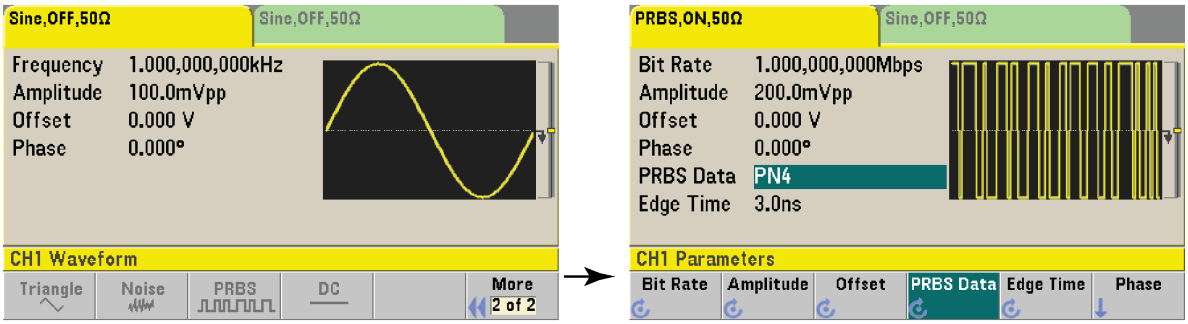
Using the Python function `line_spectra_CS_dBm()` driven by pulse train Fourier coefficients, $X_n$, chose a value for $f_0$ that makes the co–sinusoidal signal *non–coherent*. See if you can observe the spectral line splitting apart. I want you see if a value of $f_0$ will make the spectral lines lapping up from the negative frequency axis to the positive frequency axis can be made to interleave the native positive frequency axis spectral lines. Your Jupyter notebook will ultimately contain a spectrum exhibiting this behavior. Note the 33600A only produces coherent co–sinusoidal signals, as far as I have been able to observe.

## Pseudo–Noise (PN) Sequence Generators (PRBS)

In this portion of the experiment you will verify some of the properties of PN sequence generators, specifically m–sequences. You will also consider random sequences and the impact of line coding, specifically non–return–to zero (NRZ) versus Manchester. The $m$–sequence generator will be implemented using the capabilities of the 33600A. To implement line coding the arbitrary waveform capability (ARB) will be utilized.

## Part a

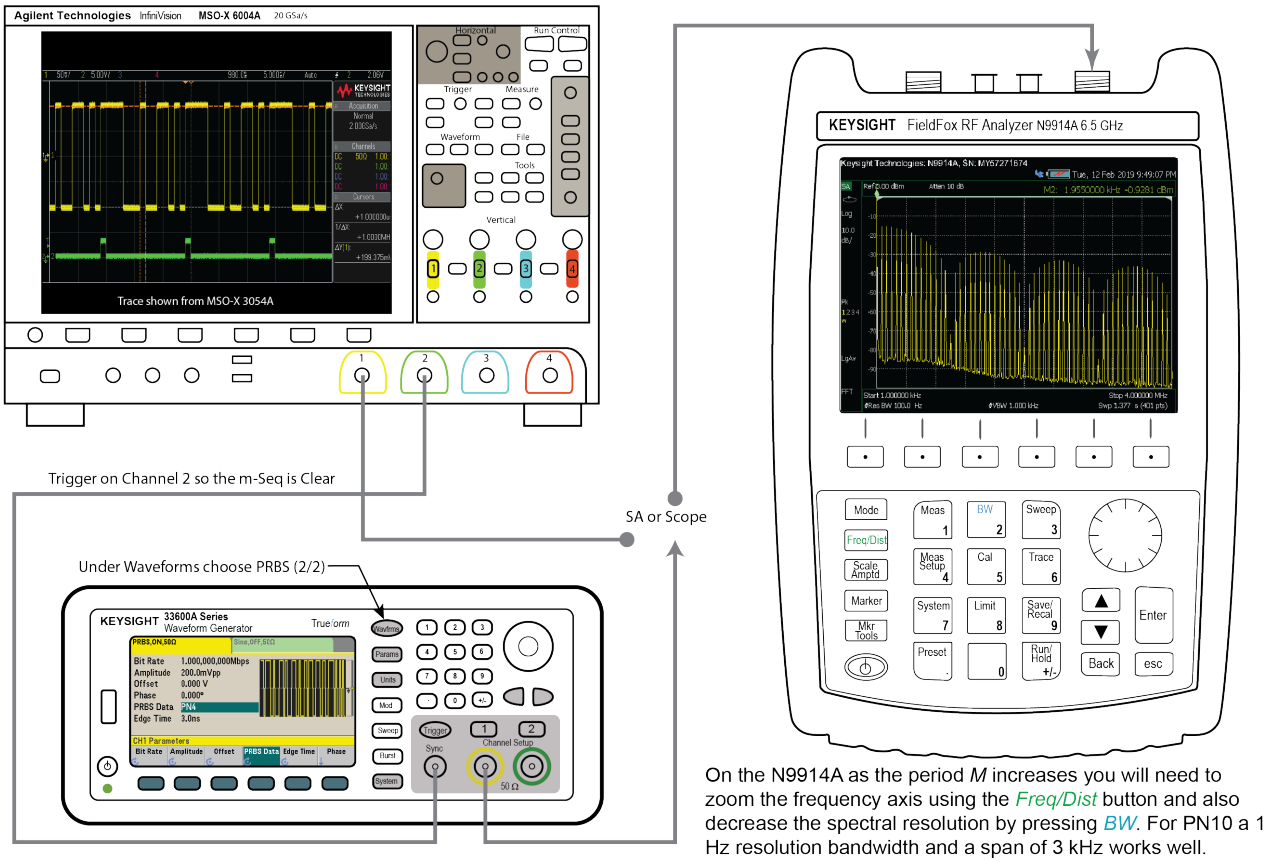
Configure the 33600A as shown in Figure 14. This will a four stage $m$–sequence which has period $M = 2^4 - 1 = 15$ bits. The rate is set to 1 Mbps and the amplitude is 200 mVpp.




Press Waveforms and choose *More 2 of 2* in the lower right to reveal the *PRBS* selection option. Once inside the PRBS selection option chose a Bit Rate of 1 Mbps, and amplitude of 200 mVpp, and PRBS sequence generator PN4, which generates a 4-stage maximal length sequence.

Figure 14: 33600A PRBS settings to produce a PN4 ($M = 2^4 - 1 = 15$) bit stream having bit rate of 1 Mbps and amplitude 200mVpp .

When viewing the output of pseudo random sequence generators on the scope, e.g., the MSO–X 6004A, use the generator trigger waveform to ensure a stable data stream can be observed. Figure 15 shows how to set this up for observing both the time domain waveform and the power spectrum on the SA.




On the N9914A as the period *M* increases you will need to zoom the frequency axis using the *Freq/Dist* button and also decrease the spectral resolution by pressing *BW*. For PN10 a 1 Hz resolution bandwidth and a span of 3 kHz works well.

Figure 15: 33600A PRBS settings and measurement interconnects for the case PN4.

A property of $m$–sequences is that once per period a sequence of *ones* match the shift register length must be present. In the miniature scope waveform of Figure 16 we see a block of 4–ones repeat three times across the trace, perfectly aligned with trigger waveform (the green trace). Note be sure to set the scope to trigger on the *Trigger* waveform output from the 33600A. Next you will look at PN5 and PN10 waveforms and will have great difficulty syncing the scope without the use of the trigger signal.

To complete *Part a* change the generator to *PN5* and observe:

1. On the scope the occurrence of 5–ones in a row once over the sequence period
2. On the spectrum analyzer verify the location of the spectral nulls
3. Verify the minimum line spacing is as expected for PN5 which has period $M = 2^5 - 1 = 31$.

## Part b

Change the generator to *PN10* and repeat the measurements of *Part a*.

## Part c: Almost Random Sequence Generator

Change the generator to *PN32*, which is the largest shift register length available on the 33600A. Observe the spectrum on the N9914A. As you zoom in to a single spectral lobe and then reduce the resolution bandwidth see if you can resolve the spectral lines. What is the period of the *PN32* generator in seconds?

## Part d: Arbitrary Waveform and Line Coding

For this experiment you will utilize the arbitrary waveform (ARB) capability of the 33600A to playback two different 63–bit m–sequence waveforms. In general a PN sequence or PRBS, as referred to by the 33600A, has mathematical form

$$x(t) = \sum_{n=-\infty}^{\infty} a_n p(t - nT_b) \tag{49}$$

where $T_b$ is the bit period and $R_b = 1/T_b$ is the serial bit rate of bit stream. The pulse shapes of interest here are NRZ and MAN as defined earlier. The $\{a_n\}$ form a sequence of $\pm 1$ bits that repeat with period $M$.

Creating ARB waveforms is easy and is described in the Jupyter notebook sample and also in a Keysight ARB creation application note. The files `NRZ.csv` and `MAN.csv` are contained in the Jupyter notebook sample ZIP file. These files correspond to a $M = 63$ $m$–sequence waveform like (62) at 100 samples per bit, e.g.,

```
# Desire Rb = 1 Mbps with 100x oversampling
fs = 100e6 # this parameter is entered on the 33600A
# Samples per bit
Ns = 100
```

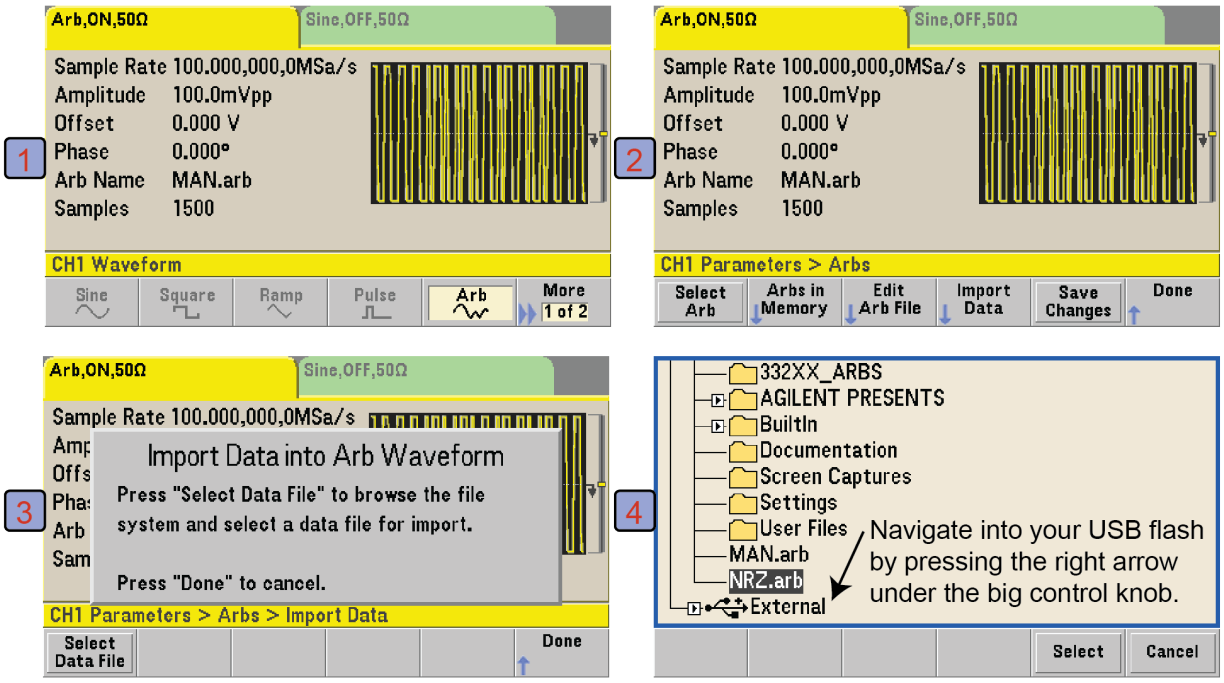
```
# One sample per bit M = 63 bit long m-sequence
N_STAGE = 4
# Use the PN_gen (M-seq) function ss over the bitwise_PN
PN_63 = ss.PN_gen(2**N_STAGE - 1, N_STAGE)
# Upsample and make rectangular pulse shaped bits bipolar
x_NRZ = signal.lfilter(ones(Ns),1,ss.upsample(2*PN_63-1,Ns))

# Create an Ns period squarewave clock to convert NRZ to Manchester
n = arange(len(x_NRZ))
x_CLK = sign(cos(2*pi*1e6/fs*n))
# Multiply the clock by the NRZ waveform
x_MAN = x_NRZ*x_CLK

savetxt('NRZ.csv', x_NRZ, delimiter=',')
savetxt('MAN.csv', x_MAN, delimiter=',')
```

Loading New ARB `.csv` Files into the 33600A

When ARB file is first created in `.csv` form it has to be loaded, and in a sense compiled by the 33600A into a `.arb` file that will allow *waveform *playback* in a repeating loop. Figure 16 shows the steps required to do this. Note: You will need a flash drive to move the `.csv` from your computer to the 33600A.




(1) Press *Waveforms* and choose *Arb*, then (2) press *Import Data*, which will drill further into the configuration. Choose (3) *Select Data File* and (4) navigate to your .csv file. Press Select for your file and then when back up to (3) press Done. Now you are ready to move on to loading an ARB file.

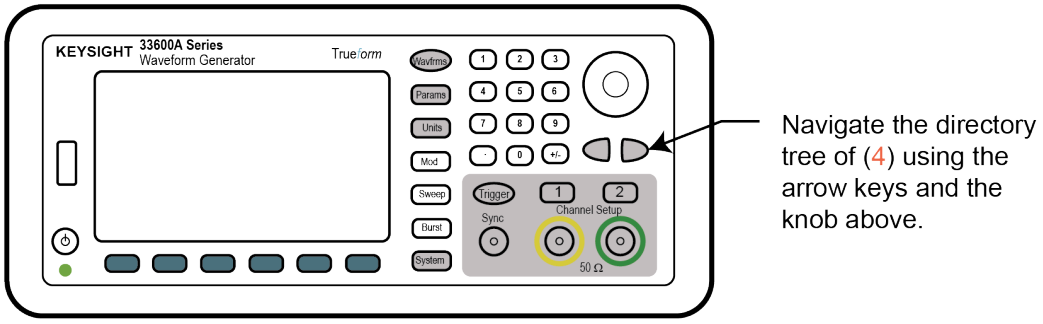
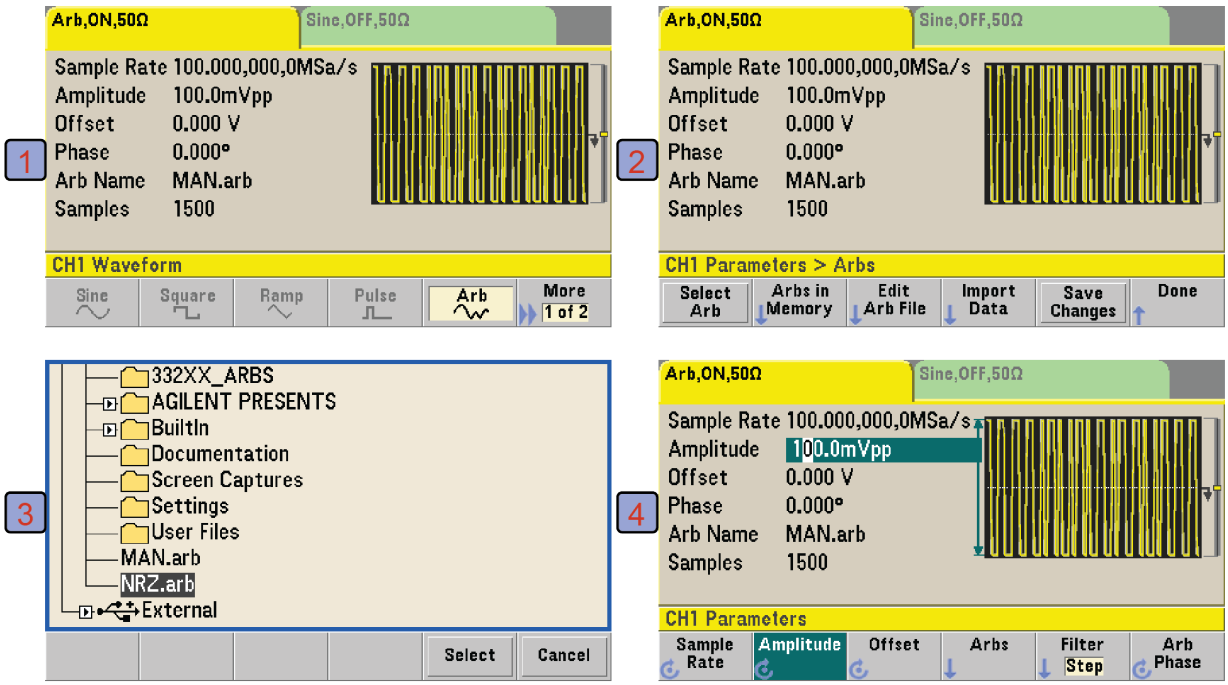



Figure 16: Copying a `.csv` file from USB flash to the 33600A.

Running an Existing ARB File into the 33600A

Figure 17 shows the steps for running an existing/recently loaded `.ARB` file.




(1) Press *Waveforms* and choose *Arb*, then (2) press *Select ARB,* which will drill into (3) *Select Data File.* Select your ARB file. Press (3) *Select* for your file and now in (4) you can move on to setting the parameters of your ARB file: Sample Rate (100 MSa/s = MHz) and the amplitude 100 mVpp.

Figure 17: Running an existing `.ARB` file on the 33600A using the *ARB* waveform mode.

Completing NRZ/MAN experiment by loading and running `.arb` files and making some observations.

1. Load the NRZ ARB file and after setting the sampling rate to 100 MHz to insure $R_b = 1$ Mbps and amplitude 100 mVpp, take a look at the waveform on the scope and on the spectrum analyzer. From the scope see that the pattern corresponds to $M = 63$. From the power spectrum note the location of the spectral nulls and line spacing. The general shape should match the Fourier transform results from the preliminary analysis plots of $P_{\text{NRZ}}(f)$.
2. Load the MAN ARB file and after setting the sampling rate to 100 MHz to insure $R_b = 1$ Mbps and amplitude 100 mVpp, take a look at the waveform on the scope and on the spectrum analyzer. From the scope see that the pattern corresponds to $M = 63$. From the power spectrum note the location of the spectral nulls and line spacing. The general shape should match the Fourier transform results from the preliminary analysis plots of $P_{\text{MAN}}(f)$.

# References


1. Rodger Ziemer and William Tranter, Principles of Communications, 8th edition, Wiley, 2014.
2. https://commonmark.org/.


# Appendix

## Capturing Screenshots from the Keysight 33600A

TBD

## Deleting ARB and Screen Shots from the Keysight 33600A

TBD