ECE 4670 Spring 2014 Lab 2
Spectrum Analysis

1 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. A microcontoller will be used at the end of this experiment to generate more random like waveforms.

1.1 Background

1.1.1 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 no 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_o, that is $x(t) = x(t + kT_0)$ for any integer k, then

$$x(t) = \sum_{n=-\infty}^{\infty} X_n e^{j2\pi n f_0 t} \tag{1}$$

where

$$X_n = \frac{1}{T_0} \int_{T_0} x(t) e^{-j2\pi n f_0 t} dt \tag{2}$$

and $f_o = 1/T_o$. By Fourier transforming (1) term-by-term using $\mathcal{F}\{e^{j2\pi f_0 t}\} \Leftrightarrow X(f - f_0)$, we can write

$$X(f) = \sum_{n=-\infty}^{\infty} X_n \delta(f - n f_0). \tag{3}$$

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

$$x(t) = \sum_{m=-\infty}^{\infty} p(t - mT_0), \tag{4}$$

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

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

Using the Fourier transform pair

$$\sum_{m=-\infty}^{\infty} p(t - mT_0) \Leftrightarrow \sum_{n=-\infty}^{\infty} f_0 P(nf_0)\delta(f - nf_0) \tag{6}$$

where

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

we have

$$X(f) = \sum_{n=-\infty}^{\infty} f_0 P(nf_0)\, \delta(f - nf_0). \tag{8}$$

If we now equate (3) and (8) we see that

$$X_n = f_0\, P(nf_0), \tag{9}$$

which will be 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_0 is,

$$R_x(\tau) = \frac{1}{T_0}\int_{T_0} x(t)x(t + \tau)dt \tag{11}$$

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

$$R_x(\tau) = \sum_{n=-\infty}^{\infty} |X_n|^2 e^{j2\pi nf_0\tau} = \sum_{n=-\infty}^{\infty} |f_0 P(nf_0)|^2 e^{j2\pi nf_0\tau} \tag{12}$$

and

$$S_x(f) = \sum_{n=-\infty}^{\infty} |X_n|^2 \delta(f - nf_0) = \sum_{n=-\infty}^{\infty} |f_0 P(nf_0)|^2 \delta(f - nf_0) \tag{13}$$

Note that when the PSD consists of spectral lines, the units associated with the spectral lines is actually power, at least when the impedance is included in the analysis. We now relate the PSD theory to the actual PSD measurement in a $Z_0 = 50$ ohms environment. Assuming $x(t)$ is the open circuit voltage of a $Z_0 = 50$ ohm source (lab function generator), and the load is

the spectrum analyzer, which has load impedance $Z_0 = 50$ ohms, the theoretical power of each spectral line is

$$\frac{|X_n|^2}{4 \cdot 50} = \frac{|f_0 P(nf_0)|^2}{4 \cdot 50} \text{ Watts} \tag{14}$$

at $f = nf_0$ Hz, where the 4 is due to the voltage divider action of the source and load Z_0 values. 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 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 (14) need to be doubled when relating to the single-sided display, or in dBm, we add 3 dB to the theoretical value, i.e.,

$$P_{\text{SA Theory, dBm}} = 10 \log \left[2 \cdot \frac{|f_0 P(nf_0)|^2}{4 \cdot 50} \right] + 30 \text{ dBm} \tag{15}$$

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

1.1.2 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) code is often used for this purpose. A generator of this type produces a sequences of binary digits that is periodic. For a listing of the properties of maximal-length sequences see Ziemer and Peterson[1]. 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

[1] R. E. Ziemer and R. L. Peterson, *Digital Communications and Spread Spectrum Systems*, Macmillian Publishing Co., New York, 1985

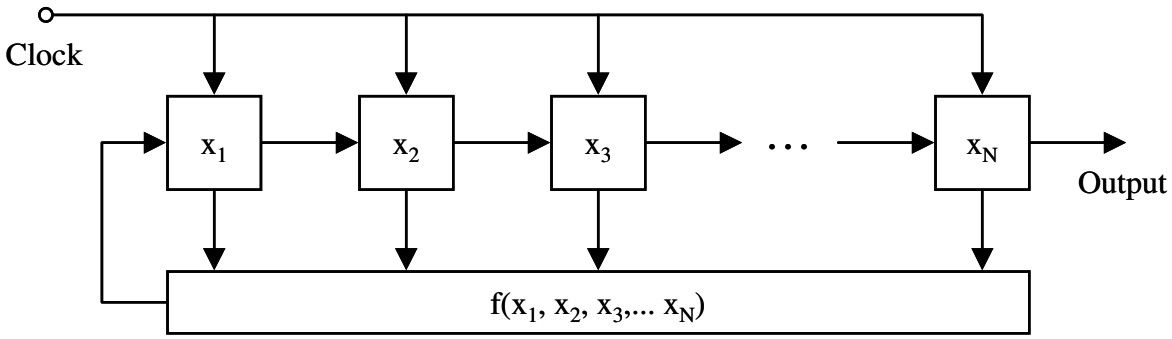


Figure 1: N-stage shift register with logical feedback

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 1 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 2. 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 auto-correlation 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.

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 ρ_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 $k = 0, 1$, and 2 is shown in Figure 3. The continuous autocorrelation function follows by connecting the discrete time points together as shown in Figure 4.

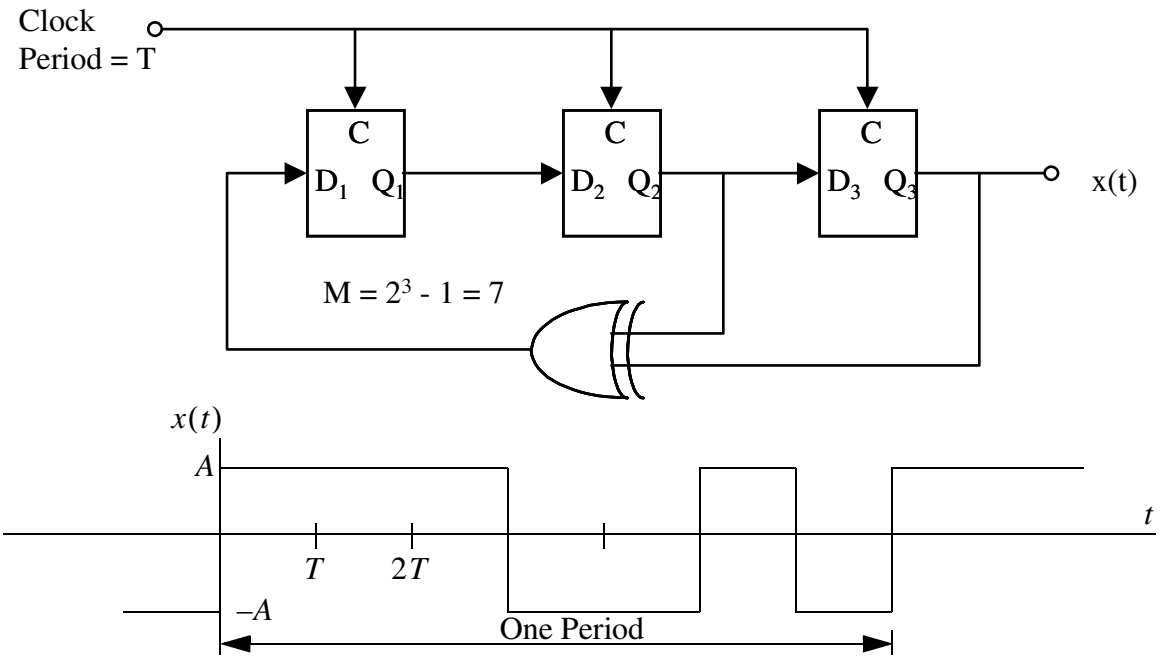


Figure 2: Three stage, length $M = 7$ m-Sequence Generator

Table 1: m-Sequence Generator Feedback Connections

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

		← 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	
		$R_x(0) = -1/7$	
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 1	
		$R_x(0) = -1/7$	
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	
		$R_x(0) = -1/7$	

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

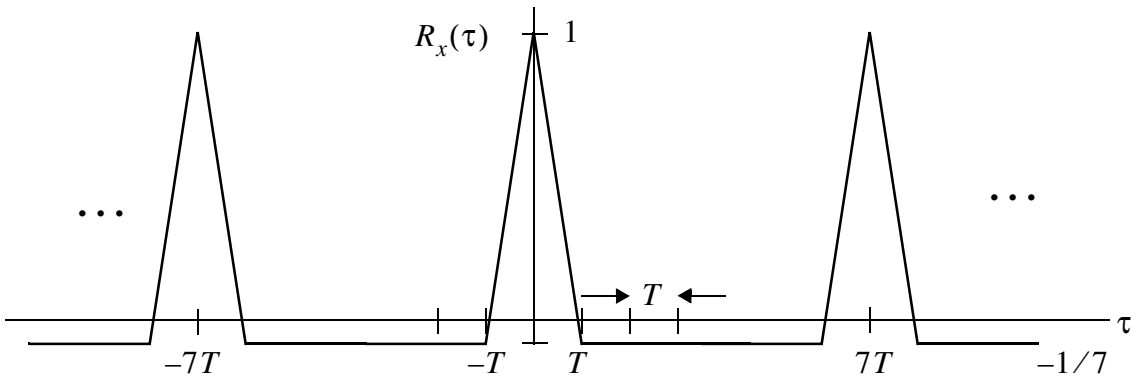


Figure 4: $M = 7$ m-sequence autocorrelation function

1.1.3 Logic Level Shifting

The analysis of the m-sequence generator in the previous section assumed logic levels of ± 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.

1.1.4 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 Δ 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

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

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).

1.1.5 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.

The natural waveform produced by the m-sequence generator, discussed earlier, is the NRZ format, as well as the PSD of (32). 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 $P(f)$

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

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

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

1.1.6 MATLAB Tools

Beyond the ADS package, there is a set of m-code files to support the simulation of the periodic pulse train, including a cosinusoidal pulse train, m-sequence, and random binary signals. For the case m-sequences and random sequences the line code can be NRZ or Manchester. The waveform generation functions are:

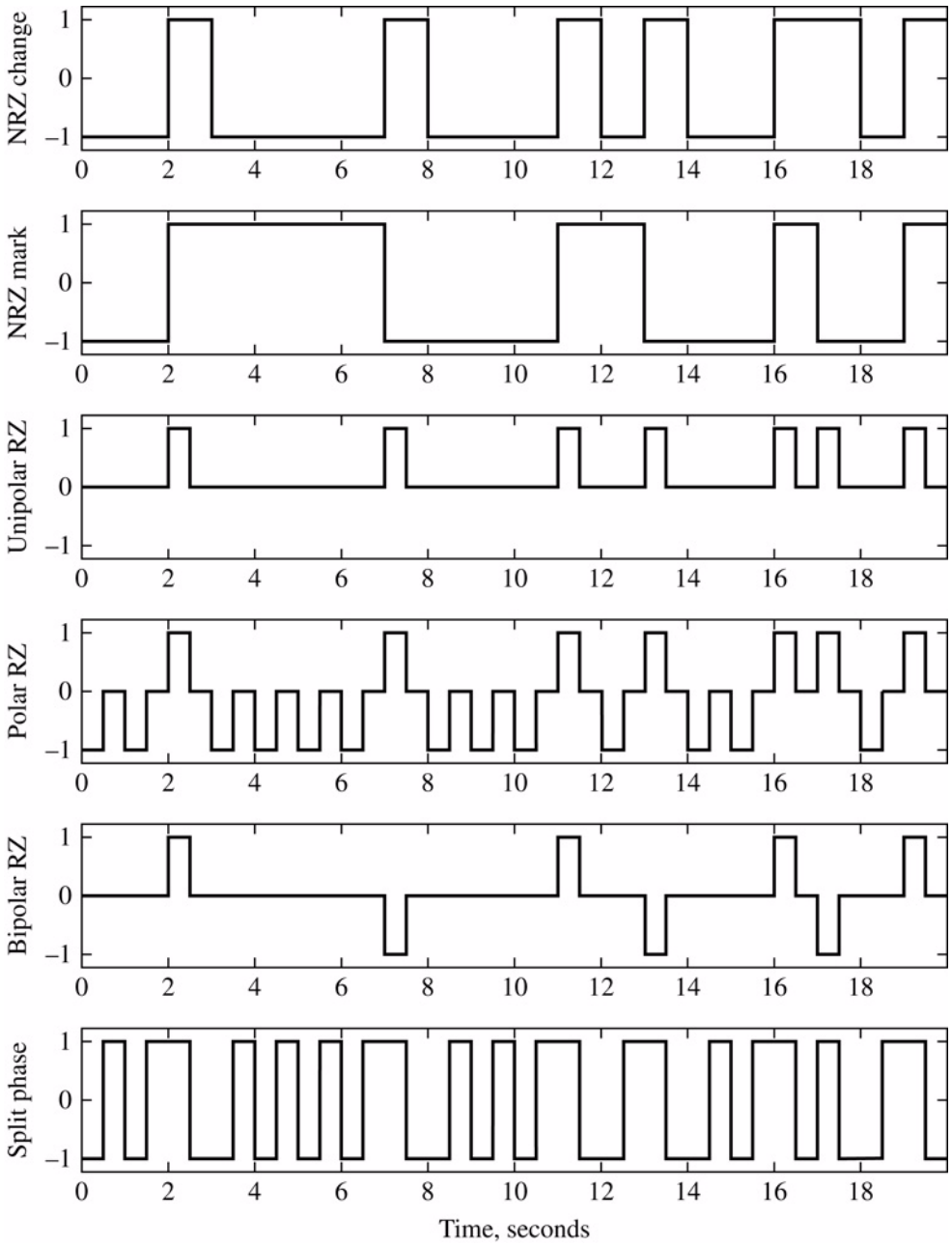


Figure 5: Binary data formats of linecodes taken from Z &T p. 212.

- `[x,t] = pulse_train(Nsim,Ns,Ntau)`
- `[x,t] = seq_gen(Nsim,Ns,SR_len,pulse,levels)`, which calls `x = m_seq(N)` to obtain one period of an m-sequence having shift register length N; if $N = 0$ a random stream is created and `pulse` may be `'NRZ'` or `'Manchester'`

The waveforms can be plotted using `plot()`, and the autocorrelation function and power spectral density can be estimated and plotted using the functions:

- `[R,tau] = simpleAcorr(x,Ns,tLags)` (plot generated if no arguments returned)
- `[Px,F] = simpleSA(x,N,fs,min_dB,max_dB,color)` (plot generated if no arguments returned)

A few simple examples of function usage follow. We begin with a pulse train with 10% duty cycle, i.e., $\tau/T = 0.1$.

```
>> % 1000 periods, 100 samples per period, so tau <=> 10 samples
>> [x,t] = pulse_train(1000,100,10);
>> subplot(311)
>> plot(t,x)
>> axis([0 2.5 -.1 1.1])
>> grid
>> subplot(312)
>> simpleAcorr(x,100,2.5);
>> plot(tau,R)
>> axis([-2.5 2.5 -.01 0.1])
>> grid
>> subplot(313)
>> simpleSA(x,2^10,100,-60, 10);
>> print -tiff -depsc matlab1.eps
```

Time, autocorrelation, and spectral density domain plots for a pulse train with $T_0 = 1$ are given in Figure 6. The natural scaling produced by the function `pulse_train()` has the period equal unity. When plotting the units in any of the three plots may of course be changed to match the physical units of interest. Also note that the PSD plot is not scaled in dBm as it stands at present. Next we consider an m-sequence with $N = 3$, so the sequence period is $M = 2^3 - 1 = 7$. Time, autocorrelation, and spectral density domain plots for this short m-sequence are given in Figure 8. The time axis scaling is 1s per bit, but can be scaled in the plotting to any value.

```
>> % 1000 bits, 20 samples per period, N = 3, NRZ line coding, logic levels -1, +1
>> [x,t] = seq_gen(1000,20,3,'NRZ',[-1 1]);
>> subplot(311)
>> plot(t,x)
>> axis([0 20 -1.1 1.1])
>> grid
>> subplot(312)
>> simpleAcorr(x,20,10);
>> axis([-2.5 2.5 -.4 1.1])
>> subplot(313)
>> simpleSA(x,2^10,20,-60, 20);
>> axis([0 5 -60 25])
```

As a final example we consider a random bit stream with Manchester line coding. Time, autocorrelation, and spectral density domain plots for a random sequence are given in Figure 8. The time axis scaling is for $T = 1$s, but can be scaled in the plotting to any value desired. Of special note is the fact that since a purely random sequence has no periodicities, the autocorrelation function is aperiodic and the power spectrum is continuous, i.e., no spectral lines.

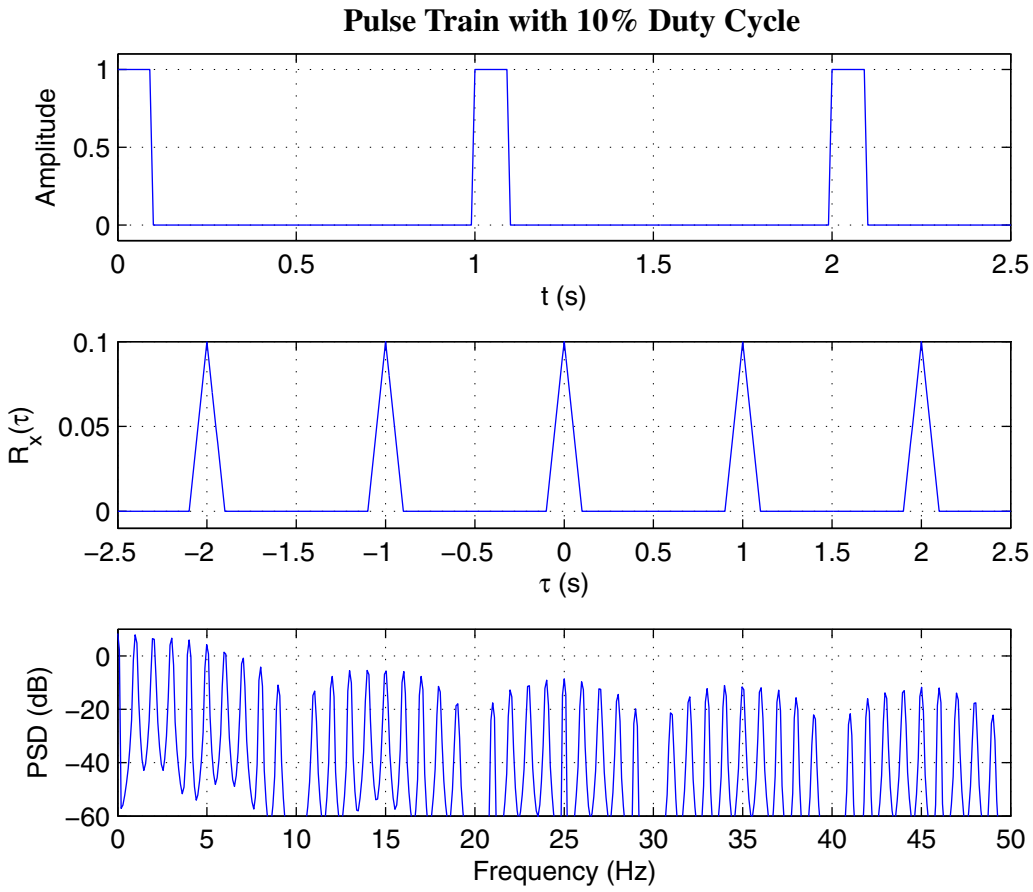


Figure 6: The waveform, autocorrelation, and PSD of a pulse train waveform having $T_0 = 1$, $\tau = 0.1$.

```
>> % 1000 bits, 20 samples per period, N = 3, Manchester line coding, logic levels 0, +1
>> [x,t] = seq_gen(10000,20,0,'Manchester',[0 1]);
>> plot(t,x)
>> grid
>> subplot(312)
>> simpleAcorr(x,20,10);
>> axis([-5 5 0 0.6])
>> subplot(313)
>> simpleSA(x,2^11,20,-60, 20);
>> simpleAcorr(x,20,20);
```

Full listings of the functions described here can be found in Appendix A.

1.2 Preliminary Analysis

1. Find the power spectral density of a periodic pulse train signal

$$x(t) = \sum_{m=-\infty}^{\infty} A\,\Pi\left(\frac{t - mT_0}{\tau}\right),\ 0 < \tau < T_0 \tag{36}$$

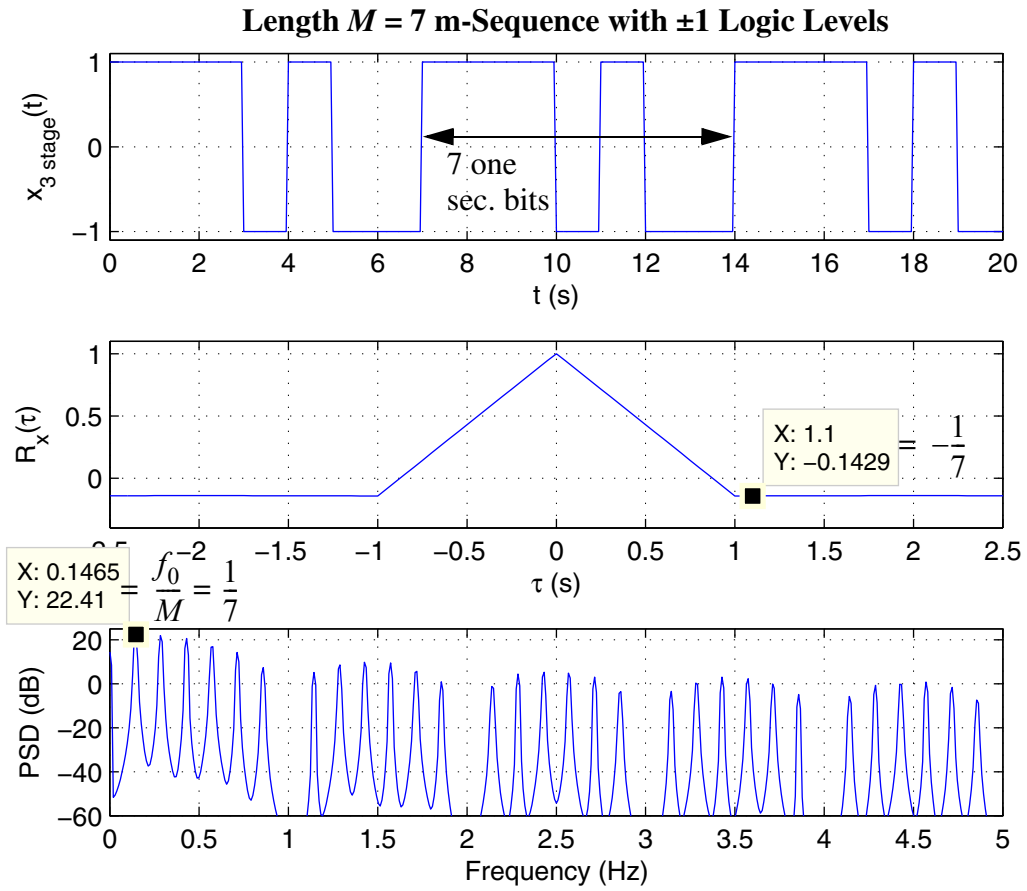


Figure 7: The waveform, autocorrelation, and PSD of an $N = 3$ m-sequence having ± 1 logic levels.

2. Find the power spectral density of a periodic cosinusoidal pulse train

$$x(t) = \sum_{m=-\infty}^{\infty} A\,\Pi\left(\frac{t - mT_0}{\tau}\right)\cos 2\pi f_c t,\ 0 < \tau < T_0 \tag{37}$$

   Note T_0 is the pulse train period and f_c is the carrier frequency. It may be that $T_0 = L/f_c$ with L an integer, that is the we have a *coherent pulse train* where where the number of carrier cycles in each pulse is integer related to the pulse width.

3. Find the power spectral density corresponding to the periodic autocorrelation function shown in Figure 9. For a detailed analysis of the PN spectrum see the solutions to *ECE 4625/5625 Set 2, Problem 1*. The hints below will also be helpful.

4. For the purely random bit stream waveform find the power spectral density for both NRZ and Manchester line coding at bit rate $R_b = 1/T = 100$ kbps.

Hints:

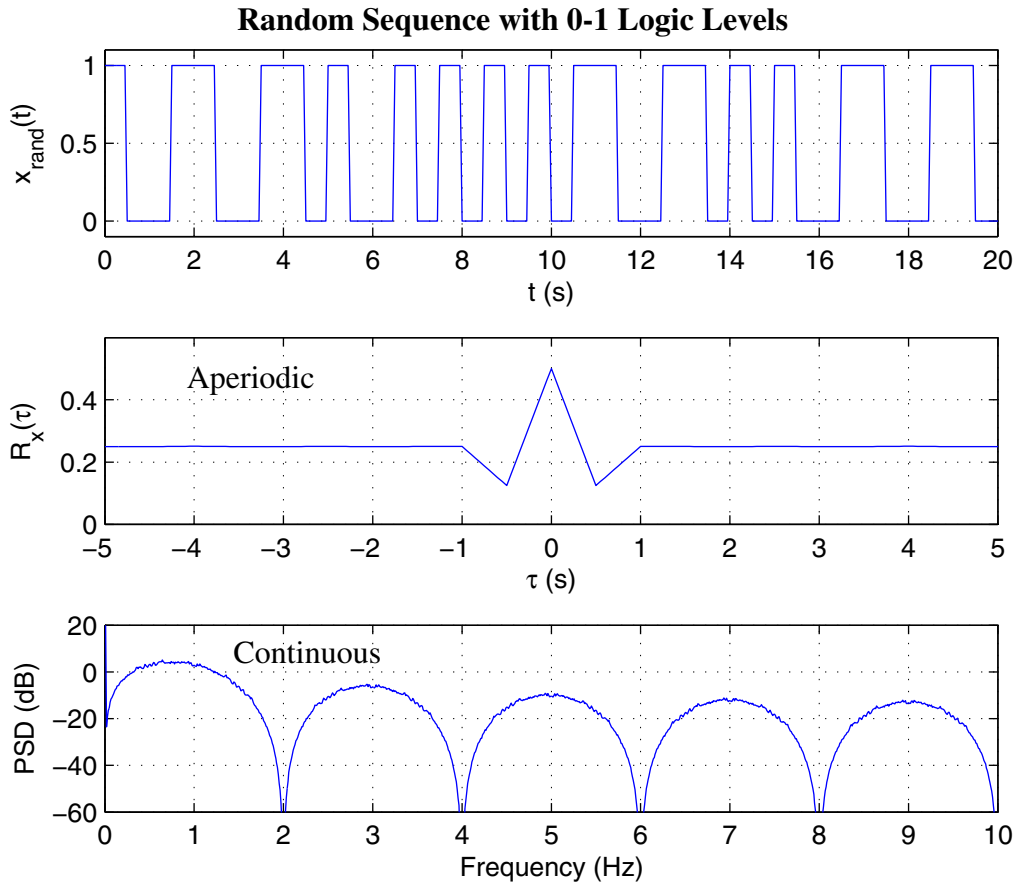


Figure 8: The waveform, autocorrelation, and PSD of a random sequence having [0,1] logic levels.

- In working (1–3) consider Section 2.5.6 of Z&T (pp. 50–51). Also recognize that when a rectangle is convolved with itself you get a triangle.
- In working (2) make use of the autocorrelation property which states that for

$$y(t) = x(t)\cos(2\pi f_0 t), \tag{38}$$

the autocorrelation of $y(t)$ is related to $x(t)$ via

$$R_y(\tau) = \frac{1}{2}R_x(\tau)\cos(2\pi f_0 \tau). \tag{39}$$

The power spectral density relationship follows from the modulation theorem for Fourier transforms.

- In working (3) also note that the plot in Figure 9 is valid for any sequence period, M. 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.,

$$p(\tau) = A^2\left(1+\frac{1}{M}\right)\Lambda\left(\frac{\tau}{T}\right) - \frac{A^2}{M}\Pi\left(\frac{\tau}{MT}\right). \tag{40}$$

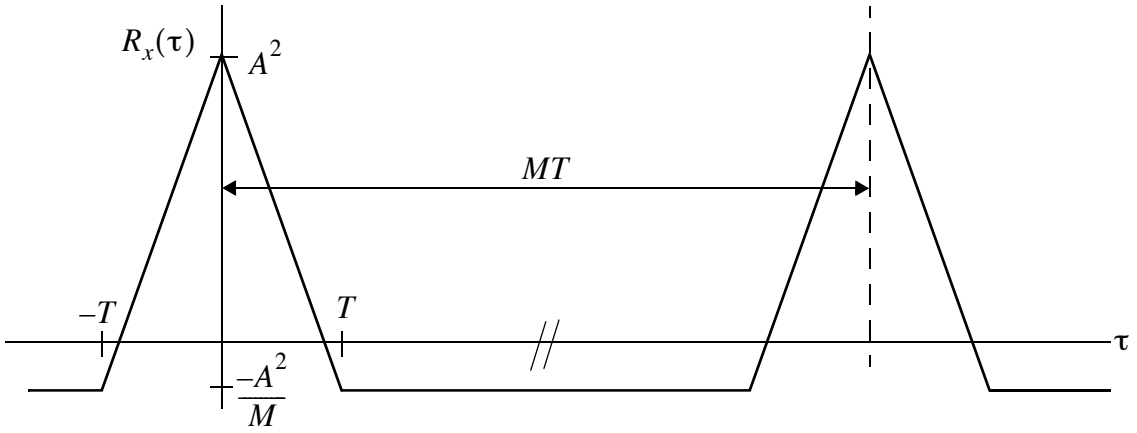


Figure 9: Periodic autocorrelation function for an m-sequence generator having period MT and $\pm A$ logic levels.

Fourier transforming results in

$$P(f) = A^2 T \left(1 + \frac{1}{M}\right) \text{sinc}^2(fT) - A^2 T \,\text{sinc}(fMT) \tag{41}$$

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 $a^2 = [(A_1 - A_0)/(2A)]^2$.

- In working (4) realize that the random bit stream waveform autocorrelation function is now aperiodic, so the power spectrum will be a continuous function of frequency. See the random binary sequence subsection.

1.3 Periodic Pulse Train

1.3.1 Laboratory Exercises

Using an Agilent 33250A generator record time domain and frequency domain data using the oscilloscope and spectrum analyzer respectively, for several different duty cycles τ/T_s. As a matter of convenience select T_s as an integer number of graticule lines on the oscilloscope screen. Pressing the Pulse button on the front panel of the 33250A puts the generator in pulse train mode. A good starting point is a low amplitude of 0 v and a high amplitude of 200 mv (200 mv p-p). The frequency F_s can be 10 kHz, making $T_s = 1/f_s = 100\,\mu$s. Then choose pulse width $\tau = 10\,\mu$s.

The frequency domain data should include:

1. The spacing between spectral lines.
2. The location of *zeros* in the spectrum envelope.
3. The number of spectral lines between zeros.

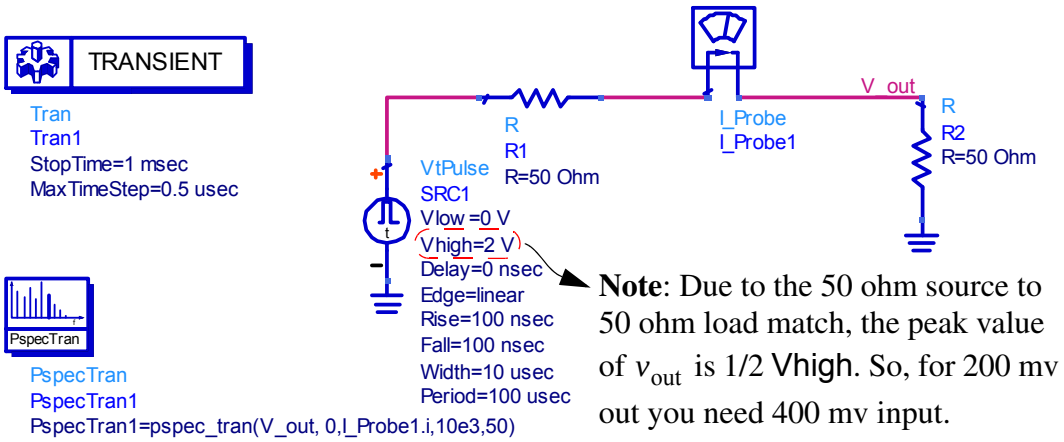


Figure 10: ADS simulation schematic for a pulse train waveform in a 50 ohm environment.

4. Any additional data that seems appropriate for comparing the experimental power spectral density with theoretical calculations. There is an ADS file package on the course Web Site (`ece4670_Lab2_prj.rar`) that contains a simulation of the pulse train and produces line spectra with dBm amplitudes. MATLAB tools are also available, discussed earlier. The schematic is shown in Figure 10 and sample output is shown in Figure 11. The key to obtaining the power spectral density plots in ADS, in the use of the schematic block `pspec_tran()`. To understand how to set this function up for different pulse periods consult Figure 12

   Use ADS simulation results and the theoretical results to compare with your measured results. Are t hey in agreement in both frequency spacing and dBm amplitudes?

5. The pulse rise and fall time can be altered on the 33250A by changing the Edge Time, which has a minimum/default value of 5 ns. Try increasing this up to 25% of the pulse width, 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? Within the ADS simulation the pulse rise and fall times can be modified accordingly.

1.4 Periodic Cosinusoidal Pulse Train

A periodic cosinusoidal pulse train can easily be generated using the Agilent 33250A in Burst Mode.

1.4.1 Laboratory Exercises

1. The waveform should be of the form shown in Figure 13. Start with a 200 mv p-p amplitude sinewave and $f_c = 1/T_c = 500$ kHz. The pulse repetition rate should be $f_0 = 1/T_0 = 10$ kHz, or for 33250A you set burst period = 100 μs. On the 33250A the value for pulse width τ is set in terms of the number of cycles per burst, that is $\tau = MT_0$ provided $MT_0 < T_s$. Start with #cycles = 10 which makes $\tau = 20\,\mu$s.

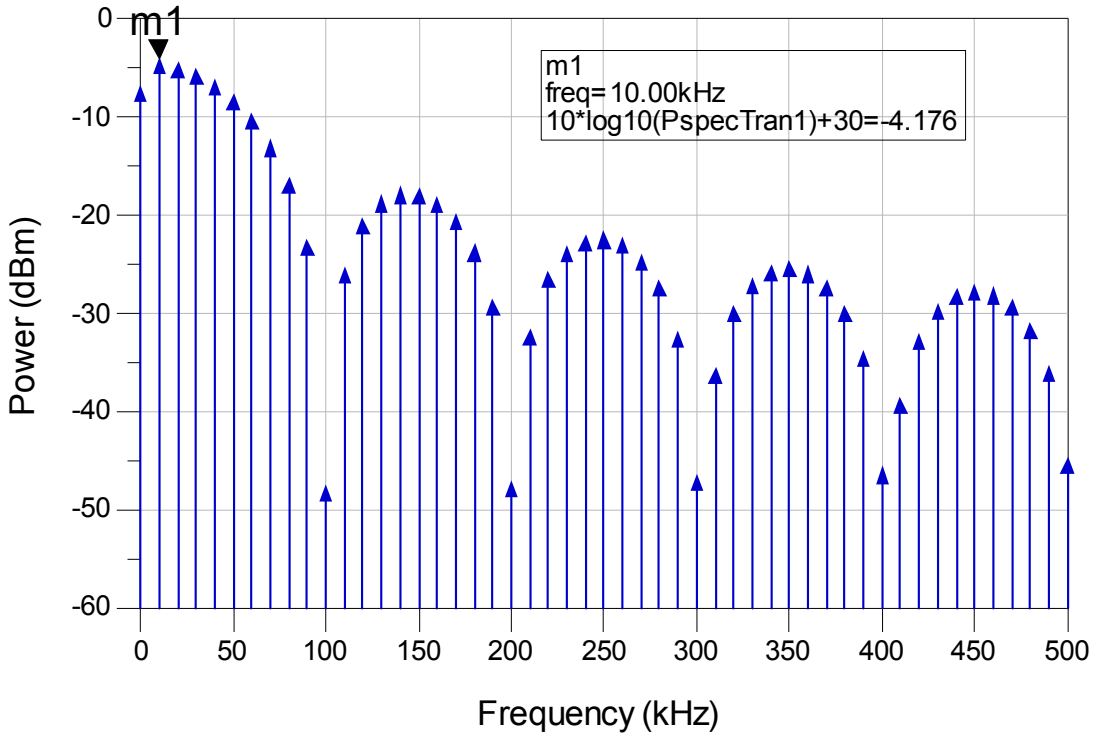


Figure 11: Power spectrum simulation results for the simulation of Figure10.

2. Take frequency domain data from the spectrum analyzer similar to that of part 1, above, so that the experimental power spectral density can be compared with theoretical calculations and simulation results. An ADS simulation of the cosinusoidal pulse train is also available, as well as MATLAB tools. The simulation block diagram is shown in Figure 14. Sample results are shown in Figure 15. Use this simulation to compare with your measurements and your theoretical expectations.

3. Observe the spectra for different values of τ/T_s with f_o fixed. Comment on your observations.

4. Now vary f_c about 500 kHz and observe the spectra. Comment on your observations. Are the results as expected?

1.5 Pseudo-Noise (PN) Sequence Generators

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 NRZ versus Manchester. The m-sequence generator will be implemented in a microcontroller, so the actual waveform generation will be under software control. The specific microcontroller is an *mbed NXP LPC1768* (`mbed.org`), from NXP semiconductor. The mbed is programmed using C/C++ with *cloud-based* development tools. To proceed with this part of

pspec_tran()

Returns transient power spectrum

Syntax
y = pspec_tran(vPlus, vMinus, iOut, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Range	Type	Default	Required
vPlus	voltage at the positive terminal	(-∞, ∞)	real, complex		yes
vMinus	voltage at the negative terminal	(-∞, ∞)	real, complex		yes
iOut	current through a branch measured for power calculation	(-∞, ∞)	real, complex		yes
fundFreq	fundamental frequency	[0, ∞)	real		yes
numHarm	number of harmonics of fundamental frequency	[0, ∞)	integer		yes
windowType	type of window to be applied to the data	[0, 9]†	integer, string	0	no
windowConst	window constant ‡ †	[0, ∞)	integer, real	0	no
† The window types and their default constants are: 0 = None 1 = Hamming 0.54 2 = Hanning 0.50 3 = Gaussian 0.75 4 = Kaiser 7.865 5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.) 6 = Blackman 7 = Blackman-Harris 8 = 8510-Minimum 0 9 = 8510-Maximum 13					

Examples

```
a = pspec_tran(v1, v2, I_Probe1.i, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec_tran(), vspec_tran()

Notes/Equations

This measurement gives a power spectrum, delivered to any part of the circuit. The measurement gives a set of RMS power values at each frequency. The *fundFreq* argument is the fundamental frequency that determines the portion of the time-domain waveform to be converted to frequency domain (typically one full period corresponding to the lowest frequency in the waveform). The *numHarm* argument is the number of harmonics of the fundamental frequency to be included in the power spectrum.

Figure 12: Help file information for the block `pspec_tran()`.

the experiment is going to require some introduction to the mbed hardware and the development environment. You will find that the coding will be very straightforward, as most of the work in using this processor has been done in the development of underlying C++ class libraries. Explicit knowledge of C++ is not really required either. For our purposes, just ANSI C coding knowledge is required.

The mbed is based on an ARM Cortex-M3 MCU in a DIP fashioned PCB. The pin-out of the DIP PCB, showing the various I/O peripherals, is shown in Figure 16.

Background material on the mbed can be found in Appendix B. The instructor will go through some of the details in class and you will also be given a handout from an ARM Microcontrollers book[2] to assist you in getting started.

[2]Bert van Dam, *ARM Microcontollers part1*, Elektor International Media. ISBN: 978-0-905705-94-1.

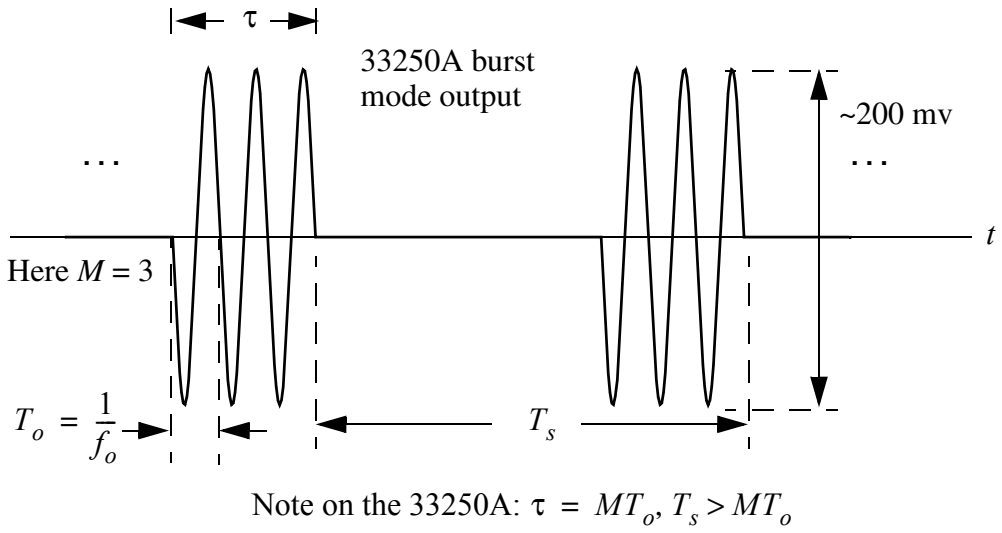


Figure 13: Coherent Cosinusoidal Pulse Train Waveform

The code listing for a supplied PN-code or *m*-sequence generator is given below. The file can be found on the Web site as `PN_seq.cpp`. As a warm-up, your first encounter with the mbed IDE will be to write a fashing LEDs program. You will then move on to creating a project in which to place the code contained in `PN_seq.cpp`.

```
1  // A PN Sequence Generator on the mbed
2  // PC_seq.cpp
3  // Use a ticker to create a periodic event which is like an ISR
4  //
5  // Mark Wickert, February 2011
6
7  #include "mbed.h"
8
9  // Function Prototypes
10 int rand_int(void);
11
12 Ticker pn_timer;        // Ticker object (Timer interrupt) for gen_PN event
13 DigitalIn DIP_sw1(p10); // NRZ or Manchester input switch
14 DigitalIn DIP_sw4(p11); // 'Random bits or m-seq bits switch
15 DigitalOut my_pin5(p5); // Output bit sequence
16 DigitalOut my_pin6(p6); // Ouput synch pulse when using m-seq
17 DigitalOut led1(LED1);  // Initialize complete LED
18 DigitalOut led2(LED2);  // Bit sequence LED
19 DigitalOut led3(LED3);  // Synch pulse LED
20 Serial pc(USBTX, USBRX); // tx, rx
21
22 unsigned int sr = 0x1;
23 unsigned int clk_state = 0x1;
24 unsigned int bit = 0x0;
25 unsigned int synch_bit = 0x0;
26 unsigned int fb, mask1, mask2, synch;
```

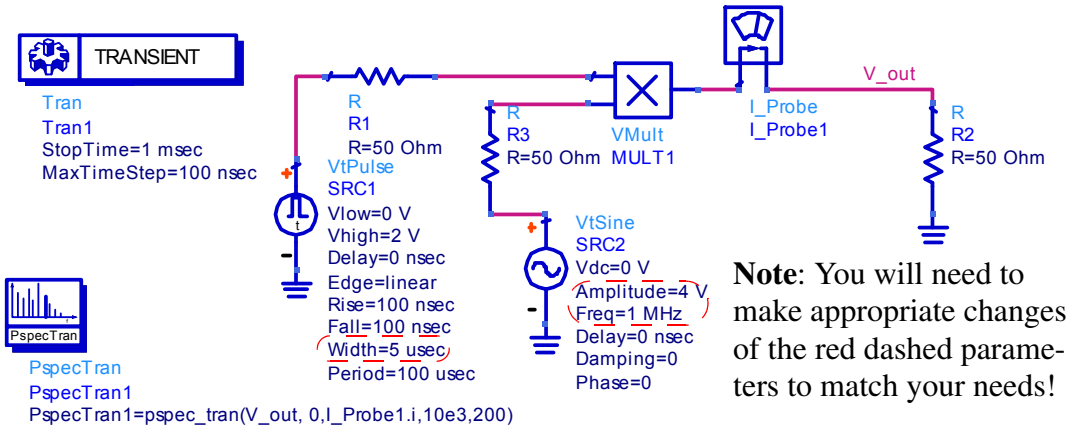


Figure 14: ADS simulation schematic for a pulse train waveform in a 50 ohm environment.

```
unsigned int tap1 = 5; // 5  9  10
unsigned int tap2 = 3;  // 3  5  3

//Random number generator
int rand_int(void)
{
    static int a = 100001;

    a = (a*125) % 2796203;
    return a;
}

void gen_PN() {
    my_pin5 = bit;
    my_pin6 = synch_bit;
    led2 = bit;
    led3 = synch_bit;
    if (clk_state == 0x1)
    {
    // Advance m-sequence generator by one bit
    // XOR tap1 and tap2 SR values and feedback to input
        fb = ((sr & mask1)>> tap1) ^ ((sr & mask2) >> tap2);
        sr = (sr << 1) + fb;
        bit = sr & 0x1;
        // Use random number generator in place of m-sequence bits
        if (DIP_sw4)
        {
            bit = rand_int() & 0x1;
        }
        clk_state = 0x0;
        // See if all 1's condition exits in SR, if so output a synch pulse
        if ((sr & synch) == synch) {
            synch_bit = 0x1;
        }
        else
```

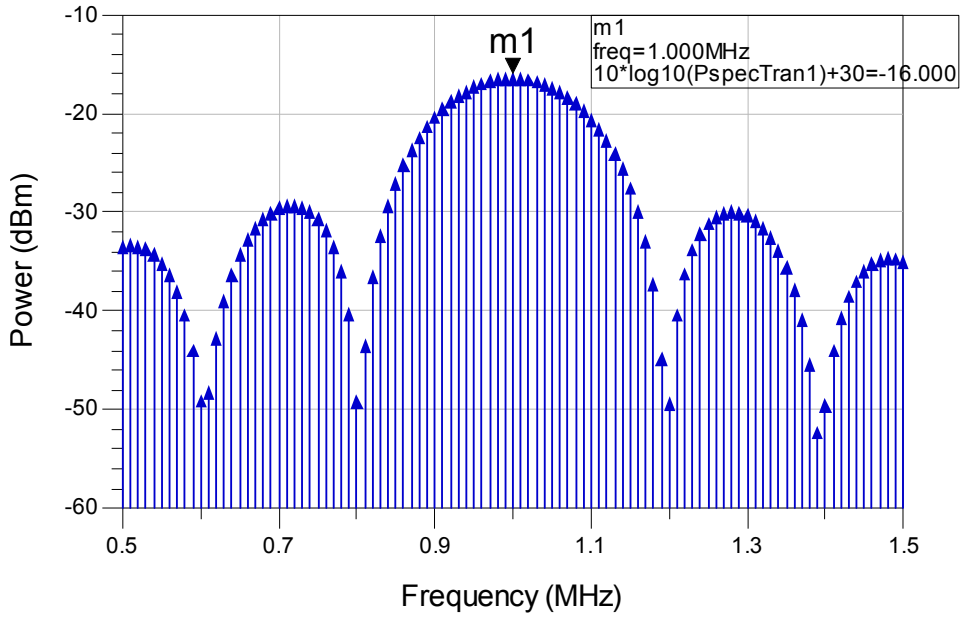


Figure 15: Power spectrum simulation results for the simulation of Figure14.

```
        {
            synch_bit = 0x0;
        }
    }
    else
    {
        if (DIP_sw1) bit = !bit;
        clk_state = 0x1;
    }
}

int main() {
    led1 = 0;
    pc.printf("Test the PN Code Generator");
    pc.printf("Tap1 = %d, Tap2 = %d",tap1, tap2);
    // tap number is one less than length since we start at position 1
    tap1 -= 1;
    tap2 -= 1;
    mask1 = 0x1 << (tap1);
    mask2 = 0x1 << (tap2);
    // For synch detect all 1's condition in SR
    // We know the m-sequence period is 2^N - 1
    synch = (0x1 << (tap1+1)) - 1; // SR length is tap1 + 1
    pc.printf("mask1 = %d,  mask2 = %d, synch = %d\n",mask1, mask2, synch);
    led1 = 1;
    // Initialize the ticker object by setting setting the sampling period for calling
    // the gen_PN function. This needs to be the last call before we enter  the infinite
    // while loop.
    pn_timer.attach_us(&gen_PN, 5.0);
```

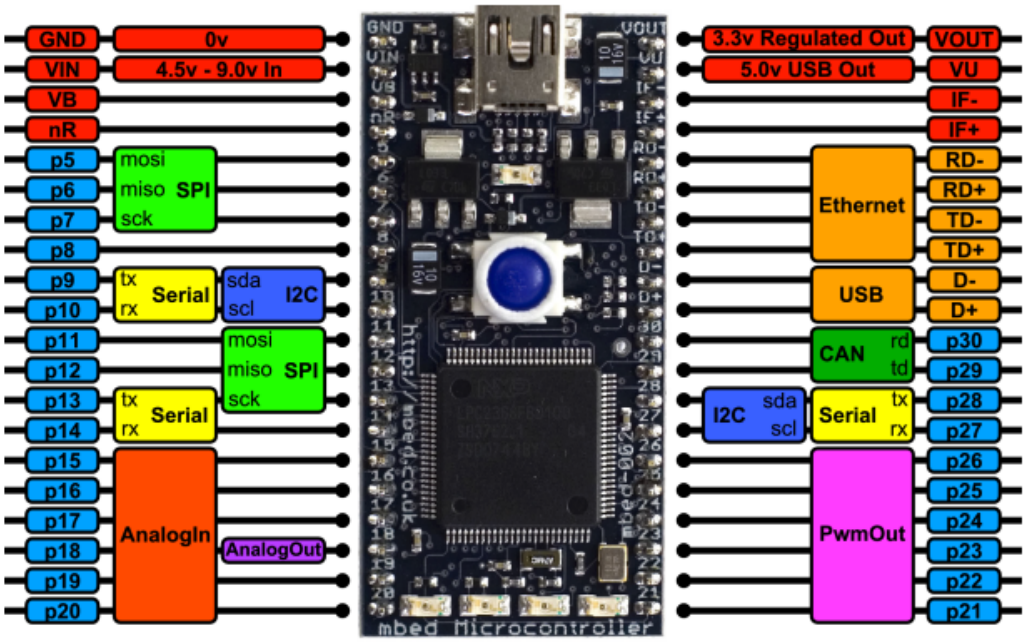


Figure 16: Block diagram/pin-out of the mbed.

```
91      while (1) {
92          //led1 = !led1;
93          //wait(0.1);
94      }
95  }
```

1.5.1 Code Description

The code consists global variable and object declarations, two custom functions, and the `main()` function. The heart of the actual *m*-sequence generator lies inside the function `gen_PN`, lines 39–71. Unsigned integers, of length 32 bits, are used to hold the contents of the PN shift register. At each clock cycle the contents of variable `sr` is shifter to the left by one bit (line 49). The XOR feedback value is shifted into the LSB position. Each of two XOR feedback values are obtained via bit masking a single bit from `sr`, right shifting the masked bit into the LSB position, and then XORing the values with each other. The result is `fb` (line 48).

The process repeats upon each call to `gen_PN`. Calls to `gen_PN` occur when a timer based interrupt fires via the *call-back* function declared (line 90) using the Ticker object (line 12). The period between call backs is set by the period parameter in line 90.

1.5.2 Laboratory Exercises

1. As a warm-up exercise work through the blinking LED example found in Appendix B or the handout from the ARM Microcontrollers book. Create a new project in the integrated development environment (IDE) similar to:

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
        while(1) {
                myled = 1;
                wait(0.2);
                myled = 0;
                wait(0.2);
        }
}
```

Download and run the code on the mbed. Demonstrate the running code to the instructor.

2. Before jumping into the *m*-sequence generator you will next build a clock generator that has an LED monitor as well as digital output to an mbed pin. The design requirements are to output the clock waveform on pin 8 (p8) at a rate of 10 kHz. The clock waveform should also flash LED4. Create a new project in the IDE that contains the code shown below:

```
#include "mbed.h"

DigitalOut myled(LED4);
DigitalOut myclock(p8);
Ticker clk_timer;        // Ticker object (Timer interrupt) for waveform generation
Serial pc(USBTX, USBRX); // enable tx, rx on PC serial port via a termal program

float period = 100.0; // start with interrupt period being 100.0 us;
                      // will need to change to make 10 kHz clock
unsigned int sq_state = 1;

void gen_sqwav()
{
        // write code to toggle LED 4; make use of the global sq_state
        // write code to toggle output pin8; make use of the global sq_state
}


int main() {
        // write message to pc terminal
        pc.printf("Squarewave with Ticker period of %4.2f us \n", period);
        // Configure the Ticker object just before entering the infinite
        // while loop for event processing
        clk_timer.attach_us(&gen_sqwav, period); // function called is gen_sqwav()
        while(1) {
                // idle loop is doing nothing but waiting to service interrupts
        }
}

```

In order to create a signal generator with the mbed, we need to generate a periodic event where we can write code that will toggle output pins. The `Ticker` object can do just that. It allows us to set up a timer-based interrupt service routine, i.e., one that will fire at a periodic interval. Configure the code so that both `LED4` and `p8` toggle between 0 and 1. Set the proper value for the period, in μs, so that the clock frequency is indeed 10 kHz. Observe the waveform on the scope and record the amplitude levels and the fundamental frequency.

The mbed has been mounted on a small breadboard for you. You need to place a ground wire from the mbed GND pin to the breadboard ground bus (blue) and a second wire from the mbed regulated 3.3 power pin VOUT to the the red power bus on the opposite side of the breadboard. See Figure 16 to orient yourself with the mbed pinout. You will need to attach a scope lead to PIN8 with a ground wire on the breadboard ground bus as well (see also Figure 17). Do not probe directly on the mbed PCB to avoid the chance of accidentally shorting pins.

In this program note that the `pc.prinf()` statement writes information to the mbed virtual comm port. The use of the serial port app `CoolTerm` is described in Appendic C of this document. Check it out as it will be a helpful debugging aid in Problem 3 and beyond.

3. Create a new project in the IDE that contains the code `PN_seq.cpp` downloaded from the Web Site. Wire-up the breadboard as shown in Figure 17. Configure the generator for $N = 3$ (code period $2^3 - 1 = 7$) and verify on the scope that the proper sequence is being generated. You will want to observe the output (pin 5) on scope Channel 1 and the synch (pin 6) on Channel 2. Synch the scope to the Channel 2 input. The Dip switches need to be set to the m-sequence mode and the pulse shape needs to be NRZ. Take care so as to not short any pins on the mbed. You will need to study the code to learn how the digital inputs are being used to control the sequence generator functionality. What is the bit rate of the m-sequence?

4. Using the spectrum analyzer, interfaced with the high impedance probes, connect to PIN5 and verify the spectral properties of the $N = 3$ m-sequence. You are in particular looking at the shape of the spectrum, the location of the lowest frequency spectral line above DC, spacing between spectral lines, and the location of spectral nulls. All of this should agree quite well with theory.

5. Compare your results using the ADS simulation of the `ADS_PN_Gen`, which has typical power spectrum results as shown in Figure 19 or with an equivalent MATLAB simulation. Note the ADS simulation example was created with $N = 4$.

6. Change the DIP switch setting to produce a Manchester encoded bit stream. Examine the waveform in the time domain (scope) and frequency domain (spectrum analyzer) and compare your observations with theory and simulation.

7. Repeat 3–6 above, with the generator now producing an $N = 10$ m-sequence.

8. As the final experiment, change the DIP switch settings to produce an NRZ waveform using a *random* sequence. Note that the random sequence is not truly random, but the period is much larger than the $N = 10$ sequence. Observe the spectral properties of this signal. Can find

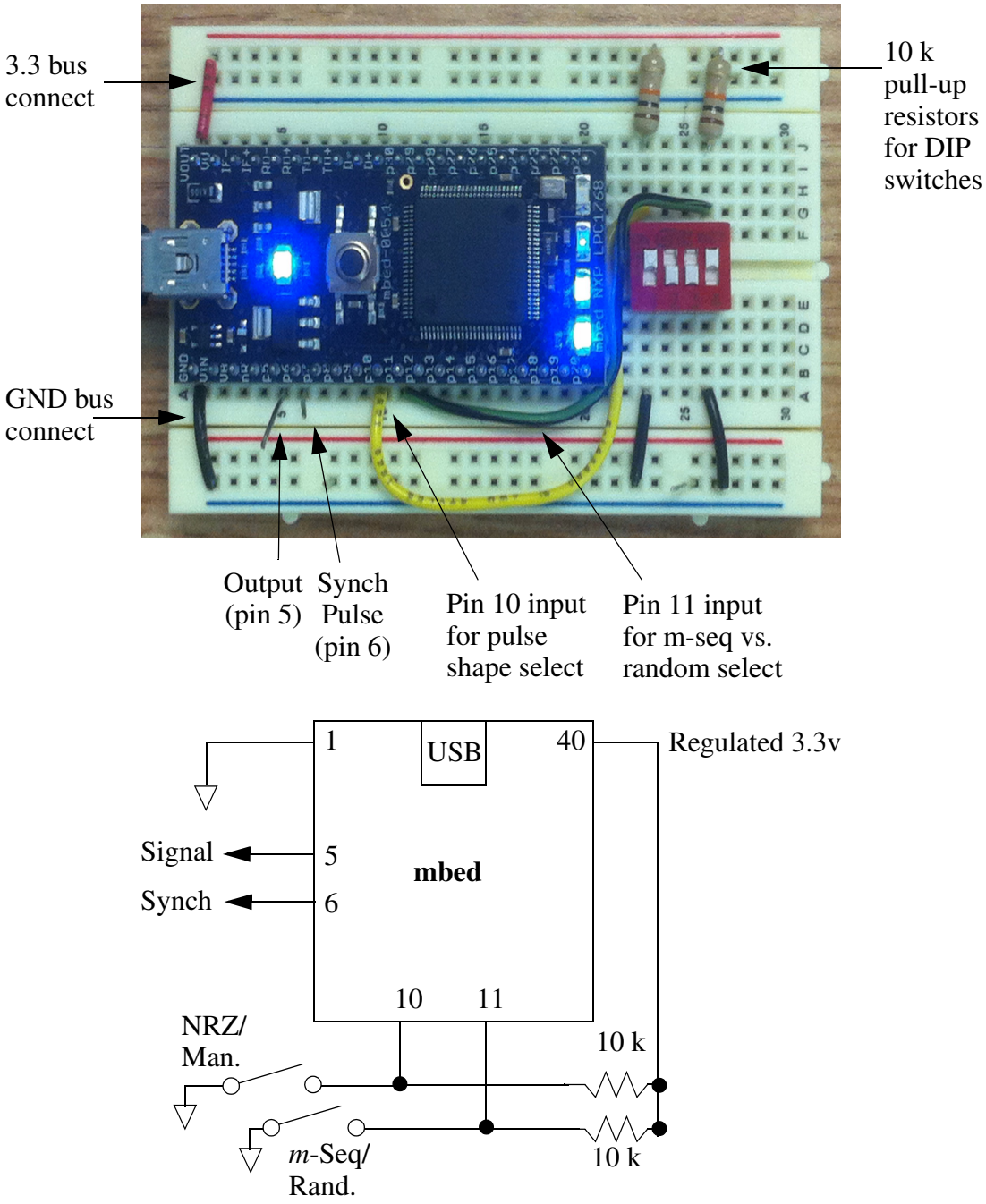


Figure 17: Breadboard configuration for mbed *m*-sequence generator.

discrete spectral lines when you zoom way in with the analyzer? Do you think the spectrum is really continuous in frequency? Explain? Change the generator to the Manchester mode just to confirm your expectations.

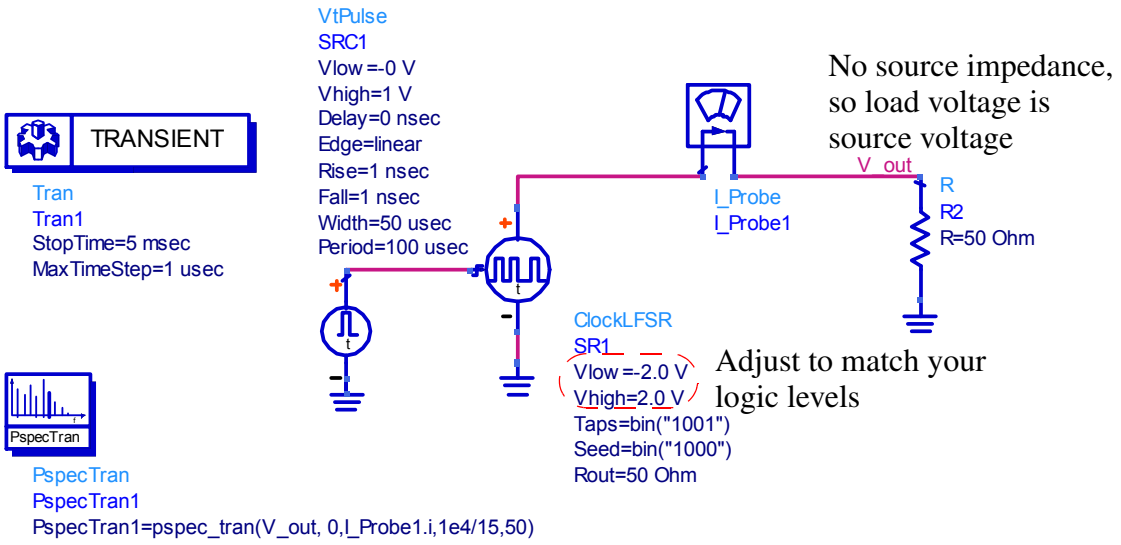


Figure 18: ADS simulation schematic for an *m*-sequence waveform in a 50 ohm environment.

1.6 Appendix A

Code listing for the MATLAB functions.

1.6.1 Pulse Train

```
function [x,t] = pulse_train(Nsim,Ns,Ntau)
% [x,t] = pulse_train(Nsim,Ns,Ntau)
%
% Nsim = number of pulse train periods to simulate
%   Ns = number of samples per period
% Ntau = number of samples pulse train is at level 1
%
%    x = signal vector
%    t = corresponding time vector assuming a period of 1
%
% Mark Wickert, February 2011

n = 0:Nsim-1;
x = [ones(1,Nsim); zeros(Ns-1,Nsim)];
x = reshape(x,1,Ns*Nsim);
x = filter(ones(1,Ntau),1,x);
t = [0:Ns*Nsim - 1]/Ns;
```

1.6.2 Sequence Generator

```
function [x,t] = seq_gen(Nsim,Ns,SR_len,pulse,levels)
% [x,t] = seq_gen(Nsim,Ns,SR_len,pulse,levels)
%
```

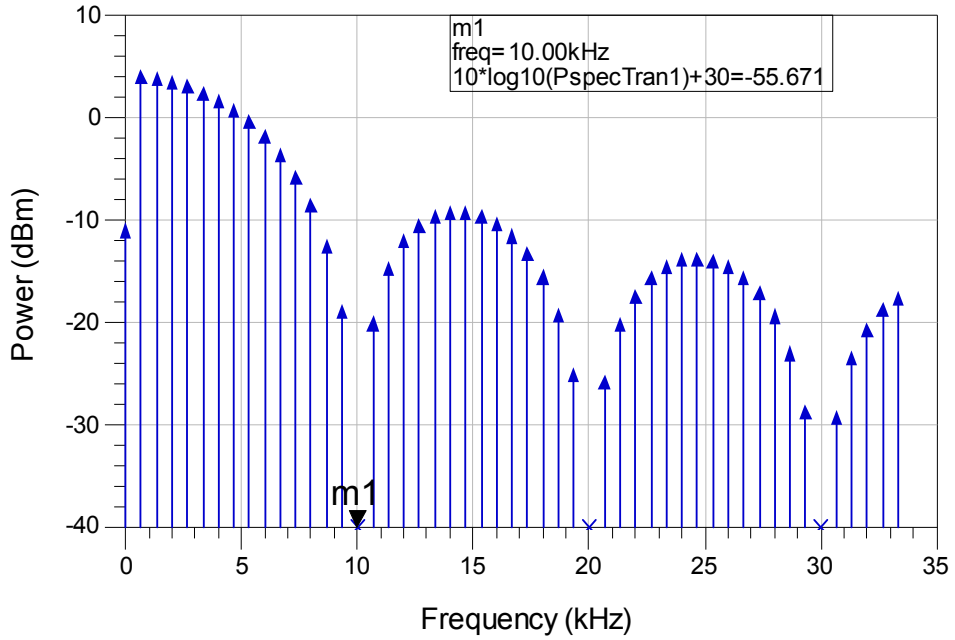


Figure 19: Power spectrum simulation results for the simulation of Figure18.

```
%   Nsim = number of pulse train periods to simulate
%     Ns = number of samples per period
% SR_len = shift register length; 0 gives a  random sequence
%  pulse = line coding pulse type: 'NRZ' or 'Manchester'
% levels = logic level as a two element vector: [low high]
%
%    x = signal vector
%    t = corresponding time vector assuming a period of 1
%
% Mark Wickert February 2011

n = 0:Nsim-1;
if SR_len == 0
    data = 2*randi([0,1],1,Nsim)- 1;
else
    PN = m_seq(SR_len);
    data = 2*PN(mod(n,2^SR_len - 1)+1)-1;
end

x = [data; zeros(Ns-1,Nsim)];
x = reshape(x,1,Ns*Nsim);

switch pulse
    case 'NRZ'
        p = ones(1,Ns);
    case 'Manchester'
        p = [ones(1,Ns/2) -ones(1,Ns/2)];

```

```
    otherwise
        error('Invalid pulse type, must be NRZ or Manchester')
end

x = filter(p,1,x)';
% Level shift +1/-1 signal to levels(1)/levels(2) values
x = (levels(2)-levels(1))/2*x + (levels(2) + levels(1))/2;
t = [0:Ns*Nsim - 1]/Ns;
```

1.6.3 m-Sequence Generator

```
function c = m_seq(m)
%function c = m_seq(m)
%
% Generate an m-sequence vector using an all-ones initialization
%
% Mark Wickert, April 2005

sr = ones(1,m);

Q = 2^m - 1;
c = zeros(1,Q);

switch m
    case 2
        taps = [1 1 1];
    case 3
        taps = [1 0 1 1];
    case 4
        taps = [1 0 0 1 1];
    case 5
        taps = [1 0 0 1 0 1];
    case 6
        taps = [1 0 0 0 0 1 1];
    case 7
        taps = [1 0 0 0 1 0 0 1];
    case 8
        taps = [1 0 0 0 1 1 1 0 1];
    case 9
        taps = [1 0 0 0 0 1 0 0 0 1];
    case 10
        taps = [1 0 0 0 0 0 0 1 0 0 1];
    case 11
        taps = [1 0 0 0 0 0 0 0 0 1 0 1];
    case 12
        taps = [1 0 0 0 0 0 1 0 1 0 0 1 1];
    case 16
        taps = [1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1];
    otherwise
        disp('Invalid length specified')
end

```

```
for n=1:Q,
   tap_xor = 0;
   c(n) = sr(m);
   for k=2:m,
       if taps(k) == 1,
           tap_xor = xor(tap_xor,xor(sr(m),sr(m-k+1)));
       end
   end
   sr(2:end) = sr(1:end-1);
   sr(1) = tap_xor;
end
```

1.6.4 Autocorrelation Function Estimator

```
function [R,tau] = simpleAcorr(x,Ns,tLags)
% [R,tau] = simpleAcorr(x,Ns,tLags)
%
%     x = input signal vector.
%    Ns = number of samples per period or samples per bit.
% tlags = number of time lags in seconds either side of zero to be
%         calculated for the plot.
%
%     R = autocorrelation function vector.
%   tau = time lag vector (zero is in the center).
%
% Mark Wickert February 2011

lags = ceil(tLags*Ns);
[R,lags] = xcorr(x,lags,'unbiased');
tau = lags/Ns;

if nargout == 2
    return
end

plot(tau,R)
grid
xlabel('Lag Time (T_0 units)')
ylabel('Autocorrelation function')
```

1.6.5 Power Spectral Density Estimator

```
function [Px,F] = simpleSA(x,N,fs,min_dB,max_dB,color)
% [Px,F] = simpleSA(x,N,fs,min_dB,max_dB,color)
% Plot the estimated power spectrum of real and complex baseband signals
% using the toolbox function psd(). This replaces the use of psd().
%
%      x = complex baseband data record, at least N samples in length
%      N = FFT length
%     fs = Sampling frequency in Hz
% min_dB = floor of spectral plot in dB
```

```
% max_dB = ceiling of spectral plot in dB
%  color = line color and type as a string, e.g., 'r' or 'r--
%
%%%%%%%%%% Optional output variables ////////////////////////////////////
% Px = power spectrum estimate values on two-sided frequency axis
%  F = the corresponding frequency axis values
%
% Mark Wickert, October 2008

% while length(x) < N
%     N = N/2;
% end

warning off
if isreal(x)
    [Px,F] = psd(x,N,fs);
else
    [Px,F] = psd(x,[],[],N,fs);
    N = fix(length(F)/2);
    F = [F(end-N+1:end)-fs; F(1:N)];
    Px = [Px(end-N+1:end); Px(1:N)];
end
warning on

if nargout == 2
    return
end

if nargin == 3 || nargin == 5,
    plot(F,10*log10(Px))
else
    plot(F,10*log10(Px),color)
end

if nargin >= 5
    axis([F(1) F(end) min_dB max_dB]);
end

xlabel('Frequency (Hz)')
ylabel('Power Spectrum in dB')
grid on
```

1.7 Appendix B

Introduction to programming an mbed microcontroller.

1.7.1 Getting Logged In/Setting up an Account

Plug the mbed into a USB port on the Lab PC. The mbed will mount like a flash drive. When the drive folder opens double-click the mbed.htm file and the Web browser will on to the page shown in Figure 20. If you do not have an account set up you will need to do so. Each lab team will share

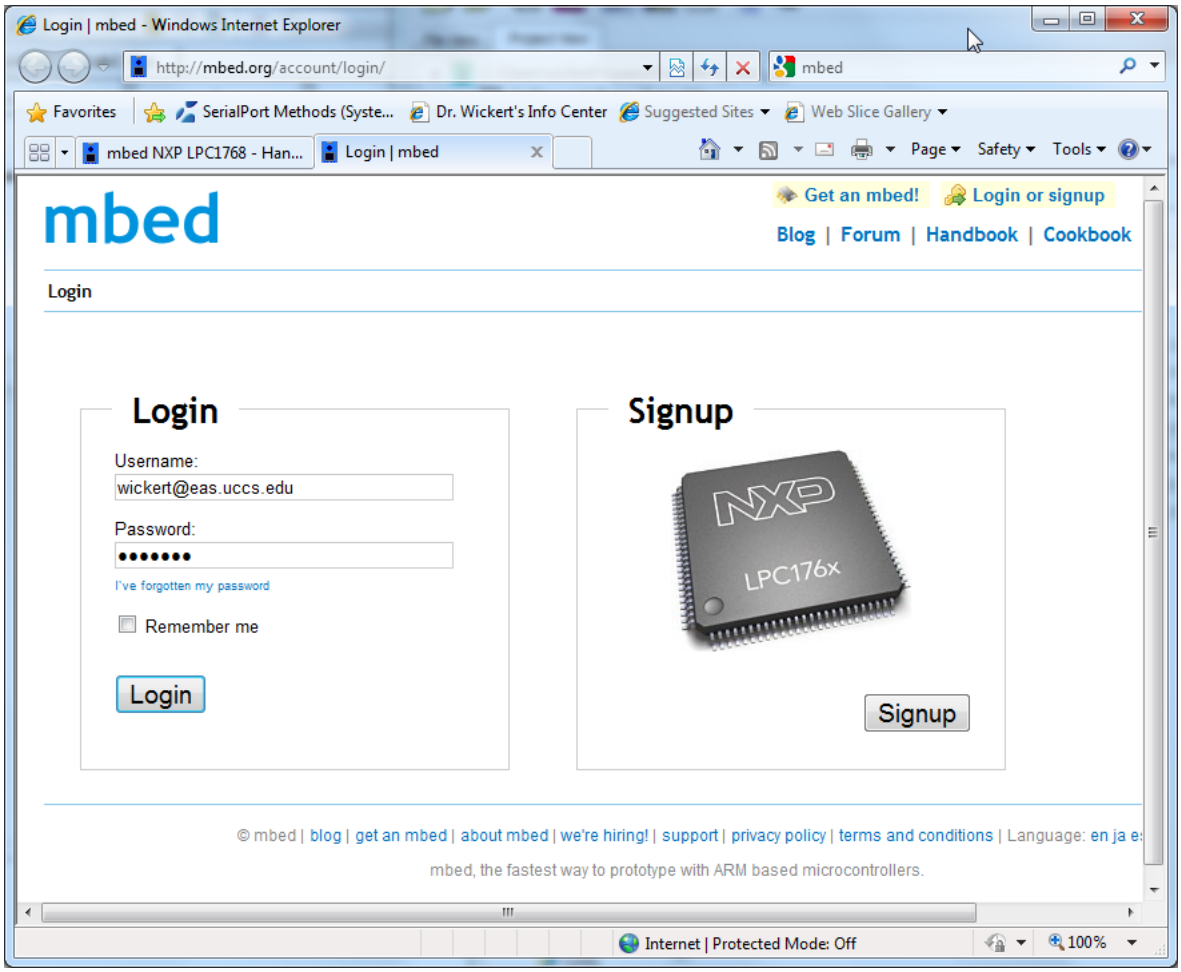


Figure 20: mbed login page.

an account.

1.7.2 Getting Started Page and Launching the Compiler

The next screen you see is the getting started page, similar to Figure 21. To launch the cloud-based complier and integrated development environment (IDE) you need to click the link in the upper right entitled Compiler. It is best to open this in a new tab, so you can still refer to the getting started page. With the IDE open you can now follow the instructions to build your first program as shown in Figure 22.

1.7.3 The mbed Library

With the compiler open, you will see that under each project there is an item that can be expended to see the contents of the mbed library. This is also available in a web page. Specific mbed library objects of interest to us are `DigitalOutput`, `DigitalInput`, `Ticker`, and `Serial`. A brief

Figure 21: The getting started page.

description of the DigitalOutput object is shown Figure 23. A brief description of the DigitalIntput object is shown Figure 24. A brief description of the Ticker object is shown Figure 25. A brief description of the Serial object, which allows communication with the PC host, is shown Figure 26.

1.8 Appendix C

Connecting to the mbed using the terminal program *CoolTerm*. Mbed code in Section 1.5.2 Problem 2 and 3 made use of the the mbed's ability to connect to the PC via a virtual serial port embedded in the USB connection.

- CoolTerm is freeware available at `http://freeware.the-meiers.org/`
- No installation is required other than to drag the app `.exe` file and a support folder into a convenient location on the PC

Consider the example mbed program shown below:

```
// Count the time to toggle a LED

#include "mbed.h"

Timer timer;
DigitalOut led(LED1);
Serial pc(USBTX, USBRX); // tx, rx

int begin, end;

int main() {
    pc.printf("Hello Mbed World!\n");
    timer.start();
    begin = timer.read_us();
    led = !led;
    end = timer.read_us();
    pc.printf("Toggle the led takes %d us\n", end - begin);
    begin = timer.read_us();
    wait_us(10);
    led = !led;
    end = timer.read_us();
    pc.printf("Toggle the led takes %d us\n", end - begin);
}
```

The lines `pc.printf()` use a special version of C `printf()` to send character strings (formated or not) to the PC terminal program.

To use CoolTerm you need to know which comm port the mbed is connected to. The Windows Device Manager can show you this (right-click over the Windows icon in the low left corner of the desktop) see Figure 27a.

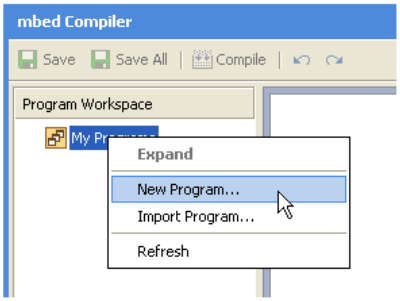
With comm port identified, now you start CoolTerm and click the Options toolbar button to select the desired comm port as shown in Figure 27b. The baud rate should be set to 9600. Close the options dialog and click the Connect button. Now when you run an mbed program that talks to the PC you will see the character strings as shown for the sample program in Figure 27c.

Creating a program

From the mbed microcontroller Handbook.

The mbed Compiler is an online application used to create your own programs for the mbed microcontroller. It translates ***program source code*** that you write in to a ***program binary*** that the Microcontroller can execute.

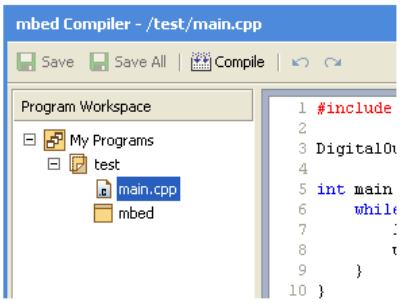
The following instructions demonstrate how to generate the Program Binary used in the Microcontroller Getting Started example.



Instructions

1. Open the mbed Compiler

Open the online compiler using the link in the site menu (top-right of the page). This will open the Compiler in a new tab or window, so you can move back and forth to continue to read these instructions.

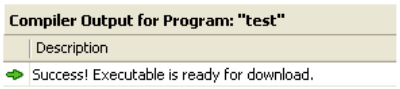


2. Create a New Program

To create a new program in your personal Program Workspace:

- Right-click (Mac users, Command-Click) on "My Programs", and select "New Program..."
- Enter the name of the new program (e.g. "test"), and click "OK"

Your new program folder will be created under "My Programs".

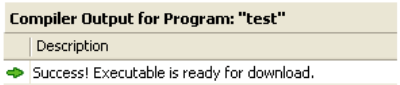


3. View the default program source code

Click on the "main.cpp" file in your new program to open it in the file editor window. This is the main source code file in your program, and by default contains a simple program already. The code should look like:

3. View the default program source code

Click on the "main.cpp" file in your new program to open it in the file editor window. This is the main source code file in your program, and by default contains a simple program already. The code should look like:



HelloWorld - main.cpp » Import this program

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

The other item in the program folder is the "mbed" library - this provides all the useful functions to start up and control the mbed Microcontroller, such as the DigitalOut interface used in this example.

4. Compile and Download the Program

To compile the program, click the ***Compile*** button in the toolbar. This will compile all the program source code files within the program folder to create a binary program.

- After a successful compile, you will get a "Success!" message in the compiler output and the download dialog will pop up. Save it to the location of the mbed Microcontroller drive, and then hit reset on the microcontroller to start it running!
- If there are errors, they will show up in the "Compiler Output" window, and will need to be fixed!

Figure 22: Building your first program.

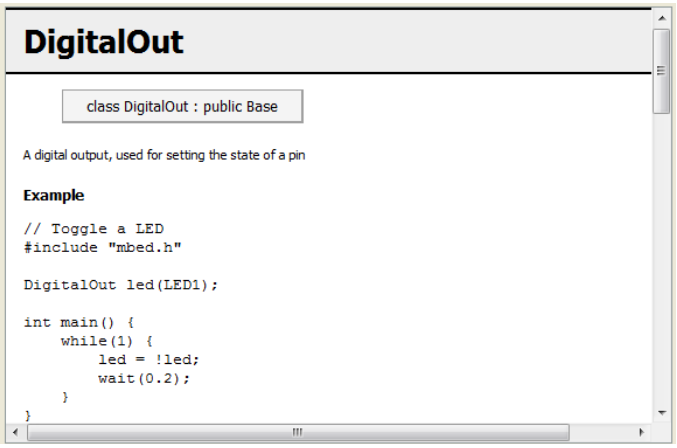


Figure 23: The `DigitalOutput` object.

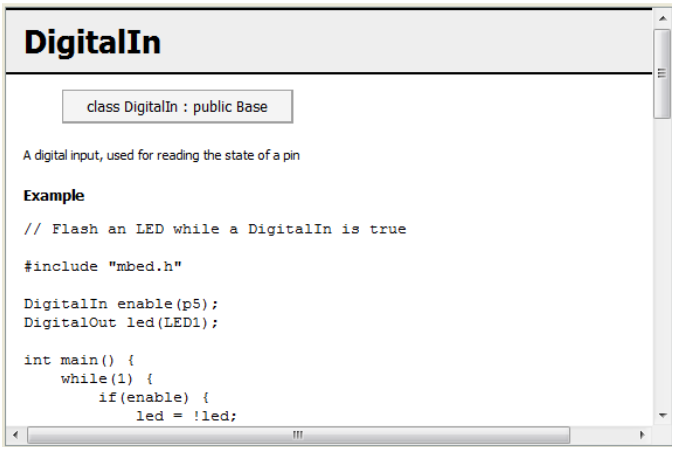


Figure 24: The `DigitalOutput` object.

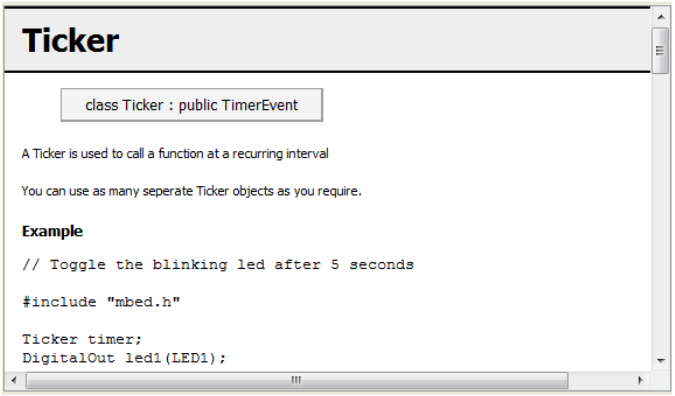


Figure 25: The `Ticker` object.

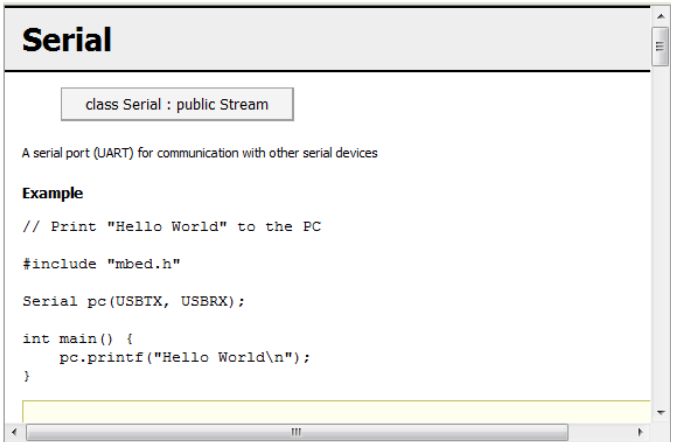
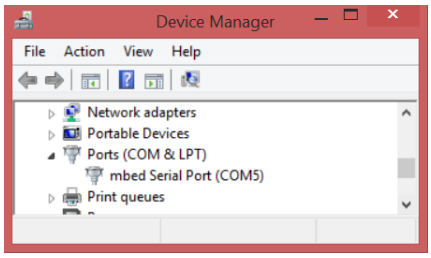
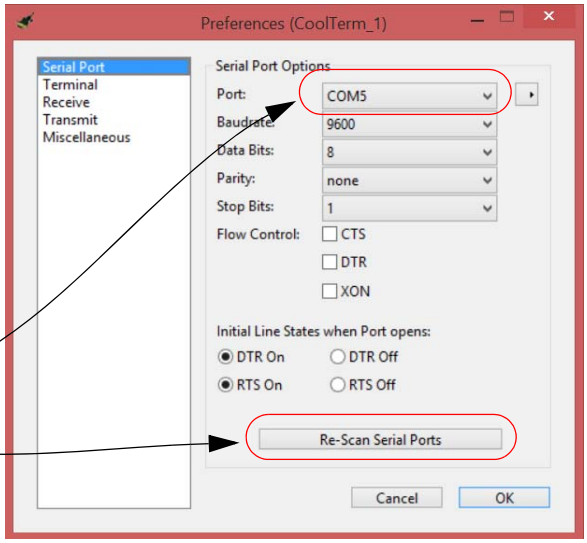


Figure 26: The `Serial` object.



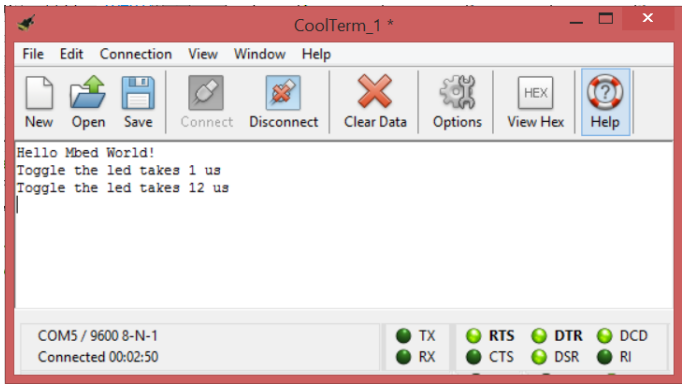
(a) Using device manager to find the comm port



Choose your port

Rescan if your port does not show

(b) CoolTerm Options



(c) Running Timer_Test.c

Figure 27: Setting up CoolTerm to connect to the mbed.