Real-Time Processing Within MATLAB

Although SigLab is optimized for dynamic system measurements, it may also be used to continuously acquire data for real-time processing by MATLAB®. After describing the relevant Siglab hardware and software features, several real-time applications examples are given.

Overview

"Real-Time," what does it mean?

The term "real-time" unfortunately lacks a precise and universal meaning. In this note, real-time indicates that the data *acquisition* process in SigLab™ will not over run the data *processing* operations performed using MATLAB® on the host PC. This means that data can be processed at least as fast as it is acquired with no gaps in the input data stream. The real-time throughput is a function of the acquisition sampling rate, data transfer overhead, number of channels, processing complexity and etc.

Processing boundaries

The key to getting reasonable performance in the MATLAB environment is vectorization, that is handling data in blocks rather than individual samples. The SigLab-MATLAB combination is suitable for operations such as filtering, detection of signals, and graphical displays of results. The processed results or unprocessed data streams can also be saved continuously to a file on the personal computer's hard disk.

The techniques to be described in this note *are not suitable* for real-time control applications which involve sending the processed results back to SigLab's output subsystem. In those applications, processing is usually performed on a sample-by-sample basis as opposed to the vectorized block-oriented operations.

Key benefits of the SigLab/MATLAB combination

Using SigLab for data acquisition provides numerous benefits over conventional data acquisition cards. These include:

- flexible analog signal conditioning
- user selectable sampling rates with superb alias protection
- both low and band passes alias filtering
- easy to use software interface to MATLAB.

The MATLAB environment also has several advantages over traditional languages for signal processing:

- efficient core processing routines (filter, FFT and vector arithmetic operations)
- extensive graphic plotting capability
- tools to create GUI based applications

These benefits allow sophisticated real-time signal processing with relative ease.

Mileage may vary!

The real-time performance depends directly on the speed of the computer. The results to be presented were obtained on a 90 MHz Pentium machine. Other relevant parameters were:

- 512K cache
- 16 MB memory
- Adaptec AH2940 PCI SCSI adapter
- Matrox MGA PCI /2 graphics adapter
- 1GB SCSI hard drive

This machine represents a typical business configuration of 1995. However, PCs get faster every day, so real-time performances improve as well.

The Signal Path From SigLab BNC to PC CRT

This section describes the long and arduous journey between the analog electrical signal present on SigLab's input BNC and the color dots on the CRT monitor or to the PC's hard disk drive.

Internal to SigLab

Figure 1 shows the signal flow within SigLab. The signal conditioning blocks allows a choice of one of the following: ac/dc coupling, dc offset and full scale range. This block also contains a fixed cutoff, fourth order, analog low pass filter for analog to digital converter (ADC) alias protection.

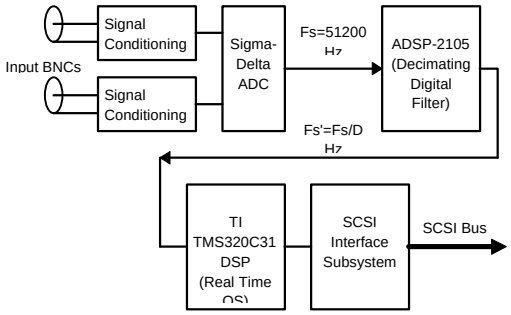

Figure 1 - SigLab internal signal flow.

The Sigma-Delta ADC

The ADC is a sigma-delta design running at a fixed conversion rate of 51200 Hz. Sampling rates below 51200 Hz are provided by the 2105 DSP chip that implements multi-stage decimating digital filters. The input subsystem is under the control of the TMS320C31 processor. This processor provides DMA support for the samples generated by the input subsystem, as well as, the control of the SCSI interface. Apart from some format and scale changes no signal processing is being done in the C31 when SigLab is being used as a data acquisition device. When data requests are made by the host PC, the C31 initiates the SCSI transactions to return blocks of data over the SCSI bus.

The decimating digital filter

In order to get the most out of SigLab's internal DSP processing, a closer look at what goes on inside the 2105 is necessary.

Thirteen sampling rate frequencies are available. These rates are obtained from a combination of digital filtering the ADC output (fixed at 51200 samples per second) and decimation of the filtered data stream.

Table 1 summarizes the sampling rates, decimations, and bandwidths available. As previously mentioned, a key benefit of using SigLab for data acquisition is its alias protection. The bandwidth column represents the available alias free frequency range for a given sampling rate. If ideal brick wall filters could be used, then the numbers in the bandwidth column would be equal to half the sampling rates (the Nyquist frequency), but such filters cannot be implemented on real hardware.

Decimation	Sampling Rate	Bandwidth
1	51200	20000
2	25600	10000
4	12800	5000
10	5120	2000
20	2560	1000
40	1280	500
100	512	200
200	256	100
400	128	50
1000	51.2	20
2000	25.6	10
4000	12.8	5
10000	5.12	2

Table 1 - Available Sampling Rates and Corresponding Bandwidths

SigLab's digital filters have impressive characteristics that guarantee a bandwidth equal to 78% of the Nyquist rate where aliases are suppressed by >90 dB.

$Fs = 51200$

$Nyquist = Fs / 2 = 25600$

$Bandwidth = 20000$

$Efficiency = Bandwidth / Nyquist = 78.125\%$

These filters have a typical pass band ripple of less than 0.01 dB with a stop band being

>90 dB down from the pass band, thereby determining the level of alias suppression. Table 1 shows SigLab's sampling rates and bandwidths are always related by a factor of 2.56. Hence the acquisition bandwidth is often specified as a SigLab setup parameter since it typically has a more meaningful physical connotation (e.g., communications bandwidth, telephone bandwidth, voice bandwidth).

Baseband acquisition

Figure 2 shows the digital signal processing used to select a 5120 Hz sampling rate that therefore corresponds to a 2000 Hz bandwidth. In baseband mode, the acquisition center frequency (Fc) of the digital filtering operation is by definition zero. Therefore, the cosine and sine terms are one and zero respectively. The block diagram then effectively reduces to the upper-two cascaded, decimating low pass filters. In this mode the acquisition system functions just like a "normal" ADC with a sampling rate of 5120 Hz; but, it is protected against aliases over the dc to 2000 Hz band. Signals above the 2000 Hz band will be attenuated by the digital filters. The transition band between 2000 and 2560 Hz (Nyquist rate for Fs=5120 Hz) can have some aliased components, but these are of little consequence.

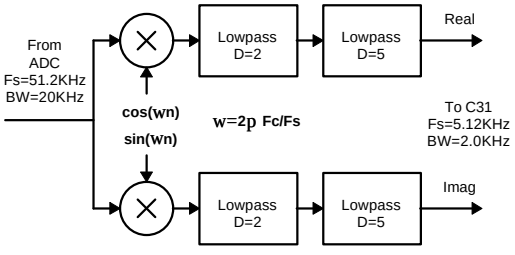

Figure 2 - Digital Filtering Example for a 5120 Hz Sampling Rate.

Band translated acquisition

When the selected acquisition center frequency (Fc) is non zero, the full block diagram is needed to describe the filtering operation. For this example, lets assume Fc=10000 Hz. The filtering process then becomes a digital band pass operator with an alias free acquisition range of 8000 to 12000 Hz. The acquisition bandwidth now doubles due to a complex time-domain data stream at a sampling rate of 5120 Hz (referred to as "real" and "imag" in Figure 2). This mode of acquisition is very useful when a narrow band signal is to be processed. For instance, assume there was a set of signals to be detected in the range of 8000 to 12000 Hz. It is far more efficient to process the 5120 complex samples per second (therefore 10240 total samples per second) than the 51200 real samples per second required to do baseband acquisition of this signal. An example of using this acquisition mode will be given.

Over the SCSI bus...

Figure 3 shows the primary hardware components involved in moving SigLab data to the monitor's screen. SigLab is a SCSI device, and therefore a SCSI adapter must be present in the host PC. Desktop machines typically have an ISA or PCI bus-based card. Some notebook machines have an integrated SCSI adapter while others use a PCMCIA card.

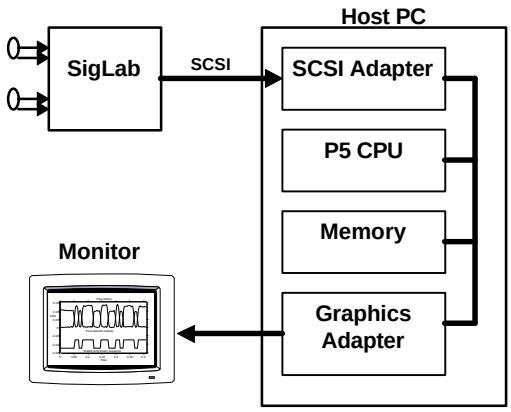

Figure 3 - SigLab to PC Host Interface

The SCSI bus is very efficient in transferring blocks of data, which provides yet another motivation for using block-oriented processing where performance is an issue.

```
% code fragment from rtp_ex1.m, the acquisition and display loop
          % request some data
          ReqID=siglab('DataReq',block_size,chan,...
                       'TimeI','First',0,'NoWait');

          HRTP_(run_flag)=1;  % set the global run flag here
          while HRTP_(run_flag) ==1
               rdy=siglab('DataRdy',ReqID);  % check for data
                                             % and interpret ready status
               if rdy     == 0
                   rflg     = 0;  % no data yet
                   cont_flg = 1;  drawnow;
               elseif rdy >= 1
                   rflg=1;        % there is data waiting
                   if cont_flg==0 & do_check==1
                      % if we don't have to wait, assume data discontinuous
                      set(HRTP_(cdisp),'string','DisCon');
                      do_check=0; drawnow;
                   end;
               else
                   rflg=0; disp(['data request error']);
               end;
               if rflg==1
                 % 'processing' section
                 [y_vec,novld]=siglab('DataGet',ReqID);  % get data into y_vec array
                 % immediately launch another request for data
                 ReqID=siglab('DataReq',block_size,chan,...
                              'TimeI','First',0,'NoWait');
                 set(HRTP_(pl1),'ydata',y_vec);     % plot it
                 drawnow;
                 cont_flg = 0;
               end;
          end; % acquisition, processing and display loop from rtp_ex1.m
```

Listing 1 - Acquisition and Display Loop

Into the Windows/MATLAB environment

Fortunately, very little code is needed to get SigLab data into the MATLAB environment. The underlying high-level software provided makes it easy.

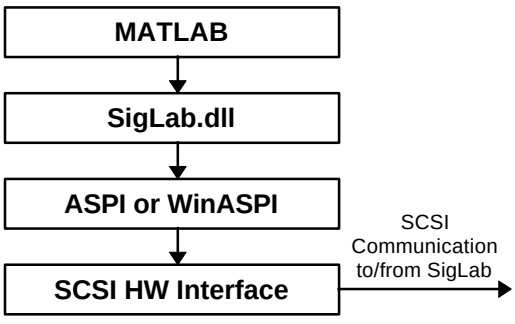

Figure 4 - Conceptual relationship between the major software components.

Figure 4 outlines the major software components involved with controlling SigLab from within the MATLAB environment. The key module is called *siglab.dll*. This software is the primary interface between MATLAB and SigLab. Calls are made to *siglab.dll* just as if it were a "normal" result in SCSI commands with commands and data being passed to and from SigLab.

The *siglab.dll* controls the input, processing, and output subsystems of the 20-22. Input control includes:

- gain, coupling, dc offset
- bandwidth (sampling rate)
- block size
- number of active channels
- triggering

The *siglab.dll* is also used to request data, test for the existence of data, and command a transfer of data into the MATLAB environment.

Once the data is in the MATLAB environment, it is trivial to plot it in virtually every way imaginable.

Real Time Processing Examples

The following section will describe several real time processing tasks. Each example will include a partial listing of the processing code required.

Time domain acquisition and display

A common real-time operation is simply acquiring and displaying data on a continuous basis.

The aforementioned Pentium system can acquire and display a single channel of data at a 12800 Hz sampling rate (5000 Hz bandwidth) if a block size between 1000 and 2000 is used. If smaller or larger block sizes are used, the efficiency drops and a lower sampling rate must be used for the data to remain continuous.

The example application *rtp_ex1.m* creates a figure window with a few simple GUI controls and acquires and plots time domain data from SigLab channel one. Listing 1 shows the acquisition and display loop. When this loop is executing, SigLab has already been configured to acquire data at a specified bandwidth and sampling rate with no triggering ("free-run").

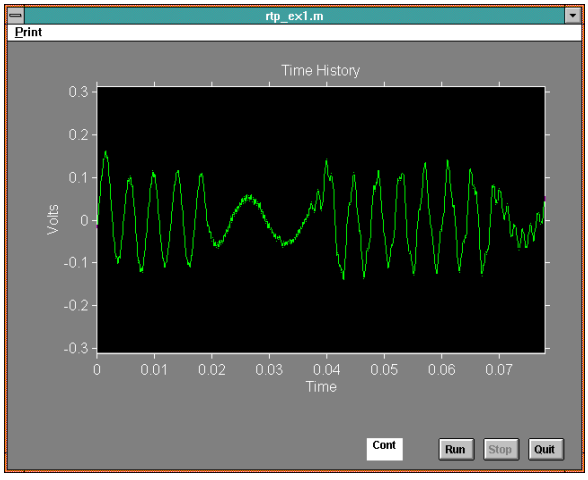

Figure 5 - Acquisition and Display of a Time History.

Figure 5 shows the axis and GUI controls. The Run, Stop buttons control the acquisition and the Quit button is used to close the application.

The text box (Cont) to the left of the Run button indicates whether or not the processing is continuous (gap free). If the MATLAB M-file code does not have to wait for data, it is a safe bet that some has been dropped and the processing cannot keep up with the rate at which data is being generated. Under these conditions the text box will be set to show the “DisCon” message. Although this check is not foolproof, it is a useful indicator of data continuity.

If CPU cycles are required for other operations, the data will likely become discontinuous at high rates. For example, if the user clicks on the figure window and drags the figure across the screen, the DisCon message will appear.

As might be expected, the plotting operation is a significant consumer of time. In fact, if data values exceed the y axis limits, the plotting time increases since the trace must be clipped to keep the graph within the axis boundaries. This can also cause the DisCon message to appear.

Time domain filtering / decimation and display

The previous example simply acquired and displayed data. The goal for the following example is to realize an effective increase in digital resolution by applying a digital low pass decimating filter to the data stream.

The idea of using a filtering operation to increase the effective resolution is a common one. It stems from the fact that the quantization process can be modeled as a uniformly distributed, uncorrelated noise added to the desired signal. The filtering operation reduces the noise while leaving the desired signal. This is tantamount to reducing the quantization noise by increasing the number of bits representing the signal.

The combination of filtering and decimation is a common signal processing operation. The filtering (and decimation) operation is, in fact, one of the key techniques used in the

```
function [y_out, s_out] = decfilt(kernel,dfac,xin,s_in)
% Dick Benson, DSP Technology
% Applies a decimating FIR filter with coefficients=kernel to
% vector xin (single channel).
% Length of kernel must be an integer multiple of dfac.
% Produces filtered & decimated output in y_out along with filter state in s_out.
     lk = length(kernel);
     lx = length(xin);
     if nargin == 3 | s_in==[]
        % assume initialization if no previous state is passed by s_in
        if rem(lk,dfac) ==0
           s_in = zeros((lk/dfac)-1,dfac);
        else
           error('Kernel length must be a multiple of the decimation factor.');
        end;
     end;
     if rem(lx,dfac) >0
         error('Xin length must be an integer multiple of the decimation factor.');
     end;
     y_out  = zeros(lx/dfac,1);
     for k=1:dfac
         [y,s_in(:,k)] = filter(kernel(dfac+1-k:dfac:lk),1,xin(k+0:dfac:lx),s_in(:,k));
         y_out=y_out+y;
     end;
     s_out = s_in;
end;
```

Listing 2 - Decimating FIR Filter Code.

sigma delta converter designs, a confirmation that bandwidth can be traded for resolution.

The following results were obtained on the Pentium system:

1. acquire blocks of data @ Fs=12,800 Hz
2. process (on a block by block basis) sampled data with an 80th order FIR low pass filter integrated with a decimate by 16 operation
3. plot the filtered and decimated results

To demonstrate the effect of this filtering, two time history snapshots were recorded, one without filtering and one with the filtering.

The code to implement a FIR decimating filter is shown in Listing 2. The integration of the filtering and decimating operations is an important point in the code design. It is computationally inefficient to compute results which will subsequently be discarded by the decimation operation.

This code combines the filtering and decimation operations and does not compute filter outputs that would be discarded by the decimation operation. It is also optimized to operate on blocks of data (vectors) since MATLAB's processing is significantly more efficient for vectorized operations.

The processing section of the acquisition and display loop is shown in Listing 3. Again, it is critical to recognize that the data comes from SigLab into the MATLAB environment, in blocks, not on a sample by sample basis. This drastically reduces the overhead of the data transfer, and, allows MATLAB to operate on the data (filter and display) using vectorized methods.

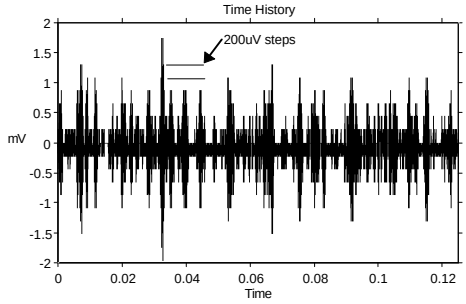

Figure 6 - Unfiltered SigLab ADC Showing Quantization Noise.

First, Figure 6 shows data from SigLab acquired at a 12,800 Hz rate, then simply decimated, without filtering, by a factor of 16. The block size was set to 1600, and therefore 100 samples are repetitively produced and plotted after the decimation operation. The quantization steps of the ADC are apparent. To provide signal headroom, the SigLab front end design does not use the full range of the ADC. The full

```
.............if rflg==1
                % data is available get it into y_vec array
                [y_vec,novld]=siglab('DataGet',ReqID);
                % then immediately launch another request
                ReqID=siglab('DataReq',block_size,chan,...
                             'TimeI','First',0,'NoWait');
                % filter/decimate the data
                [y_vec,Zstate] = decfilt(kernel,decfac,y_vec,Zstate);
                % plot filter's output
                set(HRTP_(pl1),'ydata',y_vec); drawnow;
                cont_flg = 0;
             end;
          end; % acquisition, processing and display loop
```

Listing 3 - Acquisition, Filtering and Plotting Code Fragment From rtp_ex2.m

scale swing is defined to be on the order of 50,000 ADC counts. With SigLab set to the 5 volt range (10 Vpp)

$$Vquant = \frac{10V}{50000} = 200uV$$

This quantization can clearly be seen in Figure 6. The effect of this quantization is often modeled as a noise process added to the desired signal.

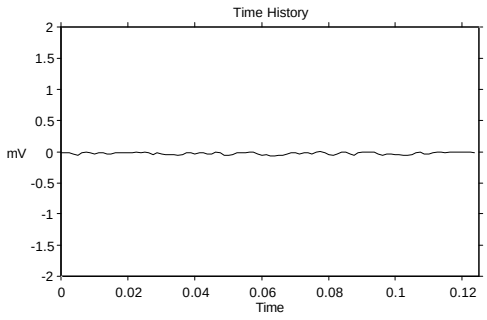

Figure 7 - Filtering and Decimation of SigLab Data Attaining a Lower Quantization Noise Level at a Reduced Sampling Rate.

Figure 7 shows how the inclusion of a digital low pass filter can improve the digital resolution of the signal by reducing the signal bandwidth and hence the quantization “noise”. In this example, reducing the signal bandwidth by a factor of 16 produces a reduction of 12 dB in quantization noise which is equivalent to adding 2 more bits to the ADC. The sampling is also reduced by the decimation process. The high frequency information has been attenuated by the low pass filter’s stop band and there is little reason (from a signal processing perspective) to maintain the high sampling rate.

The reduction in quantization noise is apparent in Figure 7. The frequency response of the FIR low pass filter used for this example is shown in Figure 8.

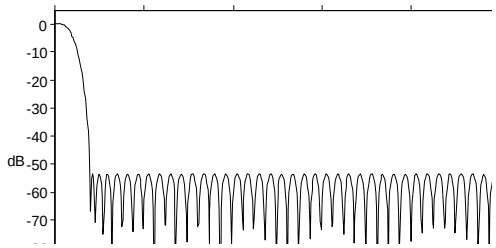

Figure 8 - Filter frequency response for decimating low pass filter.

The filter coefficients were determined by using MATLAB's Remez function. The frequency response was computed using MATLAB's FFT function.

A radio teletype signal demodulation

For this example a radio teletype (RTTY) signal from a short wave communications receiver is used as the input to SigLab. The objective is to determine the characteristics of this signal and design an appropriate detector using SigLab and MATLAB.

The Virtual Spectrum Analyzer (vsa) provides a basic understanding the characteristics of the RTTY signal.

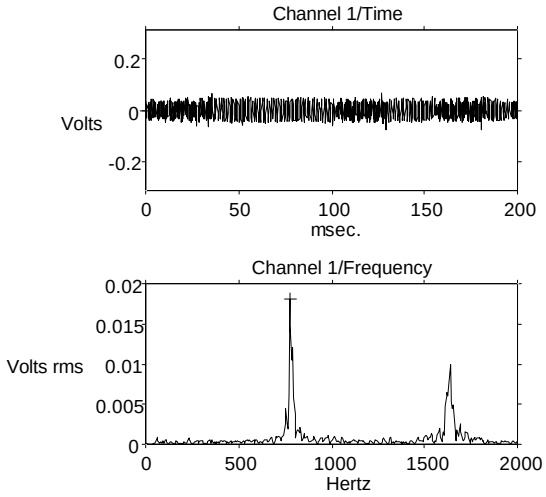

Figure 9 - Spectrum of RTTY Signal

Figure 9 shows the time history and spectrum displays of the vsa analyzing a RTTY signal. The spectrum plot shows that the signal consists primarily of two tones at the frequencies 775 and 1625 Hz. At any given time a RTTY signal contains only one of the two tones. Both are simultaneously visible in the spectrum plot since the time history is long enough to capture data transitions in the RTTY signal. The frequency shift between these tones is a healthy 850 Hz. Using a separate tone detector for each frequency is one approach to demodulating this analog signal into a bit stream.

Figure 10 illustrates one possible implementation of a "tone detector". The first step is a complex modulation (multiplication of the input data by $e^{jw_C n}$) at the desired detection frequency followed by decimating digital low pass filters where $w_C = \frac{2 \cdot p \cdot F_C}{F_S}$ and F_C is the desired detector center frequency in Hz.

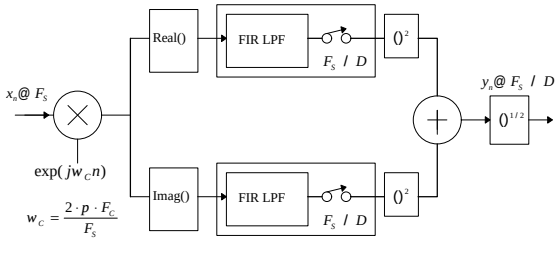

Figure 10 - A Tone Detector.

The complex modulation operation produces sum and difference frequencies in the resulting real and imaginary data streams. These real and imaginary data streams are then processed by two separate low pass filters which attenuate the sum frequencies while passing the difference frequencies. At this point the signal processing involved is virtually identical to that used in SigLab's acquisition processing except, the tone detector can accept a complex input due to the implementation of the complex modulator. Complex inputs arise by using SigLab in a band translated acquisition mode.

The filter outputs are squared, added, and square rooted performing the signal detection. Listing 4 shows the MATLAB code used to implement the tone detector.

```
function [yout,rs_out,is_out]=demod(kernel,dfac,xin,Fs,Fc,rs_in,is_in))
% Complex demodulator / Detector for MATLAB real-time processing
% Accepts both real and complex input time data (xin)
      lxin        = length(xin);
      twist       = xin.*exp(j*2*pi*(round(lxin*Fc/Fs)/lxin)*(0:(lxin-1))');
      [yr,rs_out] = decfilt(kernel,dfac,real(twist),rs_in);
      [yi,is_out] = decfilt(kernel,dfac,imag(twist),is_in);
      yout        = sqrt(yr.*yr+yi.*yi);
```

Listing 4 - Tone Detector Code

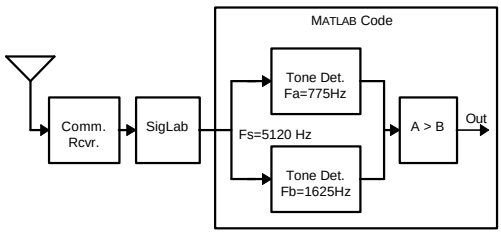

Figure 11 - Overall Block Diagram of the RTTY Demodulation Experiment.

SigLab's bandwidth is set to 2000 Hz which corresponds to a sampling rate of 5120 Hz. The FIR filter used was identical to that shown in Figure 8. Therefore, four 80th order FIR decimating filters are executing in MATLAB at the 5120 Hz sampling rate.

Figure 12 shows three plots of the RTTY detector in operation. The upper two overlaid lines correspond to the two tone detector outputs. It is interesting to note that the amplitudes of the outputs are clearly not equal. This is due mainly to frequency selective fading of the communications channel. The amplitude inequality can actually become much worse than this due to changes in the ionosphere and multi-path reflections.

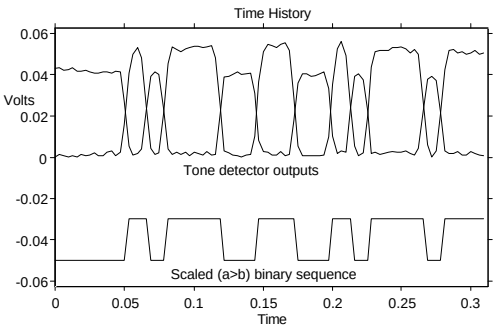

Figure 12 - RTTY "eye" Diagram and Digital Bit Stream.

The lower line on the plot is the result of comparing the two detector outputs. The comparison operator produces a binary result which is simply scaled and offset to fit nicely on this axis. This binary sequence could be fed to a RTTY decoding algorithm to produce text if desired.

Using frequency translation for narrow band signal processing.

As shown in Figure 2, SigLab has the ability to process the (real) sigma-delta ADC signal with what is effectively a complex band pass filter before the results are sent to and processed by MATLAB. Up to this point, base-band acquisition has been used exclusively, but numerous signal processing tasks can benefit from SigLab's frequency translation ability.

This band translation feature allows the analysis of signals over SigLab's entire 20 kHz analysis range without processing samples at the 51200 Hz rate. Of course, the entire 20 kHz band cannot be analyzed simultaneously, but bands can be studied a section at a time

Figure 13 shows the spectrum of an RTTY signal which has only a 160 Hz frequency shift. The frequency shift is difficult to see in the spectrum plot due to the complicated side band structure created by the data being transmitted. The two tones of interest actually lie at 1420 Hz and 1580 Hz.

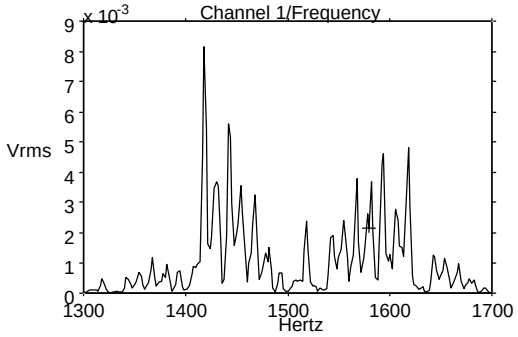

Figure 13 - Spectrum of a Narrow band RTTY Signal.

Without the band translation capability, SigLab's 2000 Hz bandwidth acquisition range would be used. Therefore, subsequent processing would be at a sampling rate of 5120 Hz.

Using SigLab's band translation capability, a center frequency of 1500 Hz can be selected, with a ±200 Hz bandwidth which then includes the two desired detection frequencies plus the associated side bands

clearly visible in Figure 13. This complex time history generated by SigLab can be processed directly since the implementation of the tone detector (Listing 4) is capable of processing either real or complex input sequences. The sampling rate is now $Fs = 2.56 \cdot 200 = 512$ *complex* samples per second. This is *one fifth*, not one tenth, the data rate that would have been required if base band acquisition were used. Remember, the samples are *complex* numbers therefore there are twice as many numbers per unit time to be processed.

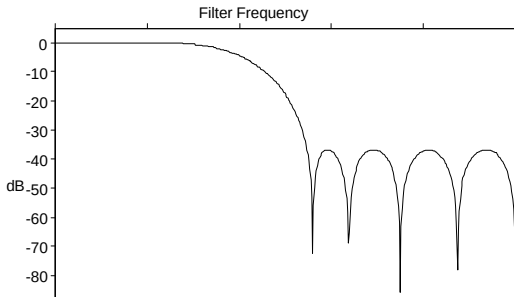

Figure 14 - Lowpass Filter Used with Frequency Translated Data.

Since much of the required signal processing (band translation) is taking place internally to SigLab, less extensive filtering is required in the MATLAB code in spite of the tone spacing being reduced from 850 Hz to 160 Hz. Figure 14 shows the characteristics of the 16th order filter used in conjunction with a decimation factor of 2.

The results of using this filtering and detection scheme on a narrow band RTTY signal are shown in Figure 15. The code identical to that of Listing 5 except different acquisition setup and filtering parameters were used (example M-file: *rtp_ex4.m*).

There are two advantages to using a smaller frequency shift. First and foremost, the signal consumes less valuable spectrum space. Secondly, frequency selective fading is reduced over the previous example with the 850 Hz shift. Everything, however, has a price since a more sophisticated receiver is required.

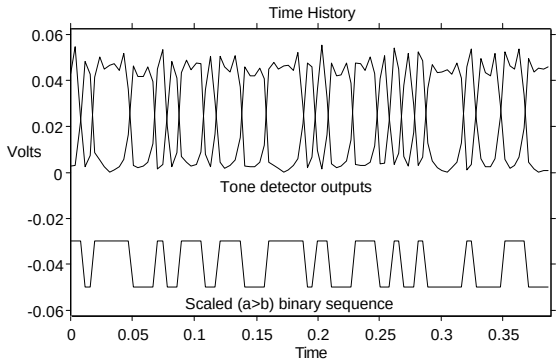

Figure 15 - Narrow Band RTTY Detection.

Direct acquisition to disk

The final example addresses the task of direct data acquisition to a disk file. The excellent tools provided in MATLAB make this an easier job. To accomplish the acquisition to disk, all that is required is the addition of a few statements to the first example of this note.

One megasample records were successfully written (gap free) at a 25600 Hz sampling

```
 if rflg==1
       [y_vec,novld]=siglab('DataGet',ReqID); % get data into y_vec array
       % immediately launch another request
       ReqID=siglab('DataReq',block_size,chan,...
                    'TimeI','First',0,'NoWait');
       [yd_vec1,re_state1,im_state1]=
               demod(kernel,decfac,y_vec,Fs,HRTP_(fc1),re_state1,im_state1);
       [yd_vec2,re_state2,im_state2]=
               demod(kernel,decfac,y_vec,Fs,HRTP_(fc2),re_state2,im_state2);
       set(HRTP_(pl1),'ydata',yd_vec1); % plot tone detector output
       set(HRTP_(pl2),'ydata',yd_vec2); % plot tone detector output
       set(HRTP_(pl3),'ydata', -0.05 +  0.02*(yd_vec1 > yd_vec2) );
       drawnow;
       cont_flg = 0;
     end;
end; % acquisition, processing and display loop rtp_ex3.m
```

Listing 5 - RTTY Detection and Plotting Code.

```
.... code fragment from rtp_ex5.m
          ReqID=siglab('DataReq',block_size,chan,...
                       'TimeI','First',0,'NoWait');
          HRTP_(run_flag)=1;  % set the global run flag here
          % loop until run flag is cleared by Stop action
          scount = 0;  % count the samples acquired
          set(HRTP_(sdisp),'string',int2str(scount));
          fid=fopen('h:\bigfile.bin','w');%**** open a file *****
          while HRTP_(run_flag) ==1
               rdy=siglab('DataRdy',ReqID);% check for data and interpret ready status
               if rdy     == 0
                   rflg     = 0;  % no data yet
                   cont_flg = 1;  drawnow;
               elseif rdy >= 1
                   rflg=1;        % there is data waiting
                   if cont_flg==0 & do_check==1
                      % if we didn't have to wait, assume data discontinuous
                      set(HRTP_(cdisp),'string','DisCon');
                      do_check=0; drawnow;
                   end;
               else
                   rflg=0; disp(['data request error']);
               end;
               if rflg==1
                 [y_vec,novld]=siglab('DataGet',ReqID); % get data into y_vec array
                 % immediately launch another request
                 ReqID=siglab('DataReq',block_size,chan,...
                              'TimeI','First',0,'NoWait');
                 fwrite(fid,y_vec,'float');% *** write 32bit format floats ***
                 scount=scount+block_size;
                 set(HRTP_(sdisp),'string',int2str(scount));
             %   set(HRTP_(pl1),'ydata',y_vec); %for max perfomance,skip plotting data
                 drawnow;
                 cont_flg = 0;
               end;
          end; % acquisition, processing and display loop
          fclose(fid);%*** close the file ***
...% end of acquisition code fragment from rtp_ex5
```

Listing 6 - Writing Data Directly to a Disk File

rate (10 kHz bandwidth) if plotting of the data to the screen was disabled. The system could both write and plot at the 12800 Hz sampling rate (5 kHz bandwidth). It should be noted that the Pentium computer used for these tests had a SCSI hard drive. Consequently the traffic on the SCSI bus was very heavy: from SigLab, to PC, and then back from PC to hard drive. Therefore, performance might even be better on a system that did not use a SCSI disk.

The relevant code fragment is shown in Listing 6. The areas concerning the creation and writing of a file have comments using a **bold** font.

The performance of this code is clearly a function of the host computer's disk storage system. To test the limits of this code, a disk partition, normally used for a scratch storage area, was defragmented using the DOS *defrag.exe* program. The file (h:\bigfile.bin) was subsequently opened and written by the example code into this optimized partition. It is important to have a large continuous area into which the file can be written. Audible disk drive activity usually indicates that multi-track seek operations are taking place, causing data to be missed at high data rates. This situation will be indicated by the DisCon text object.

Reading the file is also a simple task to implement in MATLAB. Listing 7 contains the code required to read the entire file into MATLAB. However, it is easier to write code in Listing 7 simply reads the entire file into MATLAB at one time. Files over 100K samples start to get unwieldy and can cause much disk thrashing when Windows starts

```
elseif strcmp(Action,'rd_file')
       % read and plot the data in file h:\bigfile.bin
       fid=fopen('h:\bigfile.bin','r');
       data=fread(fid,'float');
       fclose(fid);
       set(HRTP_(ax1),'XLim',[0,length(data)],...
                      'YLim',[min(data),max(data)]);
       set(get(HRTP_(ax1),'xlabel'),'visible','on','string','Samples');
       set(HRTP_(pl1),'erasemode','background');
       set(HRTP_(pl1),'xdata',(1:length(data)),'ydata',data); % plot it
```

Listing 7 - Reading and Plotting Data From Disk File

to use virtual memory. Remember, MATLAB arrays consume 8 bytes of memory for each element.

A screen capture of this example program (*rtp_ex5.m*) is shown in Figure 16. The lower center text objects read out the number of samples that have been stored to the disk file after the Run button is pushed. As in the other examples, the "Cont" field indicates if the operation was continuous or not. The plot was created by selecting the menu object labeled "Read and Plot" at the top of the figure. It invokes the code in Listing 7, which plots the entire data record stored in the file h:\bigfile.bin. This particular data is speech from an AM broadcast band station.

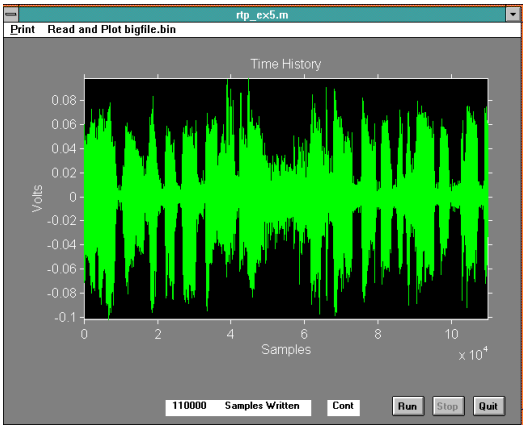

Figure 16 - Example program rtp_ex5.m Screen Capture.

To implement a more refined file I/O scheme, use the tools provided in the *vcapfile.m* utility provided with SigLab. Using these tools, very large files can be written and read back in selectable blocks.

Conclusion

Several examples of real-time processing using the SigLab-MATLAB combination have been given. The performance is impressive considering the operations are taking place on an inexpensive general purpose computing platform (a Pentium class PC), using a high level language, and performing 64 bit floating point arithmetic.

The full source code for all the examples is available by request from Spectral Dynamics, Inc. It will be provided free of charge to SigLab users. The code serves as a starting point for workers wishing to develop their own real-time processing in the MATLAB environment.

For more information contact:
Spectral Dynamics
1010 Timothy Drive
San Jose, CA 95133-1042
Phone: (408) 918-2577
Fax: (408) 918-2580
Email: siglabsupport@sd-star.com
www.spectraldynamics.com