Sampling and generating of analog signals

Labviev

Programming examples for sampling and generating of analog signals Programming skeleton for on-line signal processing

September 2013

Dušan Ponikvar

Faculty for Mathematics and Physics, Physics department
Jadranska 19, Ljubljana, Slovenia

The acquisition of analog signals using interface USB-6211 and Labview

Computer interface NI USB-6211 (National Instruments) includes one analog to digital converter (ADC) with the resolution of 16 bits. The ADC has 16 input channels (8 input channels when used in differential mode), and all digital electronic circuitry needed for periodic sampling of analog signals and transfer of samples into the PC computer. The sampled data is first stored into the memory of the USB-6211, and subsequently transferred to PC in packets. All transfer and packet handling is provided by mechanisms implemented partially in the hardware and partially in the device drivers. Once the packets are in the PC they can be used by user software. The mechanisms implemented require several parameters for correct operation; these parameters are passed to the interface when the user program calls functions which are stored in libraries. These functions allow the following modes of operation:

- **Taking a single sample of analog signal(s)**
  The interface takes a single sample of the analog input signal (or signals connected to more than one input channel). The result of sampling is immediately transferred to PC and into the user program which initiated the operation.
- **One-time only periodical sampling of analog signal(s)**
  The interface periodically samples the input analog signal (or signals connected to more than one input channel). The interface acquires a string of samples; the length of the string is determined by the calling function, as is the time interval between successive samples. The string is handled by device driver and stored partially in the interface itself, and partially in the memory of the PC. Once the sampling is finished, the completed string of samples is passed to the user program which initiated the sampling. The sampling can be initiated again by repeated call from the user program.
- **Continuous periodical sampling of analog signal(s)**
  The interface continuously samples the input analog signal (or signals connected to more than one input channel). The interface uses the user-supplied parameters to define the time interval between successive samples. Once it acquires the appropriate numbers of samples the interface passes the string of samples to the user program, but does not stop the sampling; there is no gap in sampling. The sampling continues seamlessly, and new samples are accumulated in a string which will be passed to the PC once it becomes long enough. The length of the string is determined by the calling program.
  The sampling is stopped once the user program requests stopping.

Regardless of the mode the user program must execute five steps to get samples:

1. Initialize the path between the drivers of the interface and itself, including the passing of basic parameters for the interface and the ADC in particular.
2. Pass parameters to the unit within the interface which takes care about the time intervals between successive samples (this step is omitted in the first mode).
3. Start the sampling (this step can be omitted in some cases).
4. Read the (string of) samples into the user program.
5. Break the path between the drivers of the user interface and user program.

Each step requires a separate function call to the library, each call requires parameters.

Taking a single sample of analog signal(s)

An example of a program for taking a single sample of an analog signal is given in Fig. 1. The interface samples the input signal once only, and presents the result in »Result«. The program starts with the leftmost block titled »DAQmx« (step 1), and establishes the communication between the interface driver and our program. The required parameters are:

- The user wants to sample an analog signal, therefore the initialization deals with the ADC and the analog input circuitry; this is defined by selecting the »AI Voltage« from the drop-down menu just below the first block.
- There may be more than one interface connected to the computer, and each interface can have more than one analog input channel. The user needs to state the interface name and the name of the input channel(s) used. Both names are best selected from a drop-down menu connected to the left of the block and also available at the run time of the program. The name of the drop-down menu is »Physical Ch.« in present example, and a typical interface name is »Dev1«, depending on the number of interfaces present and the order of connections. The name is assigned to the interface once it is first plugged into the PC, and the list of all interfaces (plugged in the past and/or present now) is available in »Measurement & Automation« tool, a part of the Labview package. The name of the interface is followed by the name of the channel where the analog signal is connected. For example when the analog signal is connected between input »ai0« and »AGND«, the complete name is »Dev1/ai0«.
- An amplifier is located in front of the ADC, and its gain can be selected by the user; the gain determines the maximum range of voltages that can be measured. The user can select among ranges ±10V, ±5V, ±1V in ±0,2V for the USB-6211. In the example in Fig. 1 the ±10V range is selected, and the lower limit is calculated from the upper by inverting the sign.
- The analog input signals can be connected to the interface USB-6211 in three different ways, see the literature on the interface. The way signals are connected is specified in the drop-down menu connected to the top of the block. Options are: RSE (»Referenced Single Ended«, signal against AGND), NRSE (»Non Referenced Single Ended«, signal against common reference point »AI Sense«), and Differential (the analog signal is connected between two inputs to the ADC, for instance between »ai0« and »ai8«, where the potential of each input

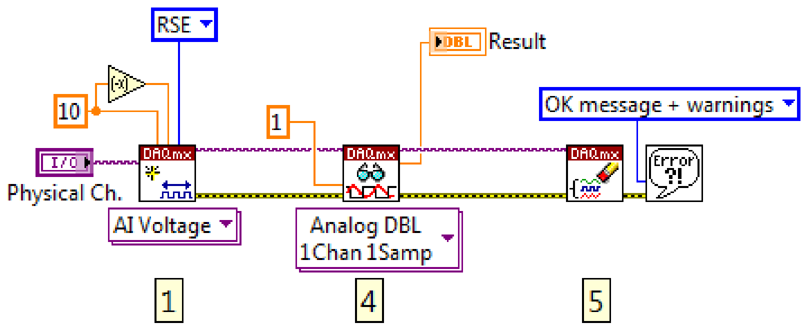

Figure 1: The Labview software for taking a single sample of an analog signal

must be within the range selected; the ADC measures the difference between the two potentials). The example in Fig. 1 selects the option »RSE«, which is the most common.

The successfully executed block returns the identification code of the interface at its upper right corner connection. This code must be used in all subsequent communication with the interface. The error code is returned at lower right corner of the block, and should also be passed to subsequent blocks. If something goes wrong within a block then the execution of subsequent blocks is immaterial, therefore these must be notified not to waste time and resources. The error code propagates through the chain of blocks, and is presented to the user at the end of the chain (if there is an error).

Once the first block is successfully executed the interface is ready to take a sample, and the user program can simply read the result of conversion; there is no need to start the conversion. The next block (step 4) is used for reading. It needs the identification code of the interface and the error message from previous block, so both are wired to its inputs.

- The same block can be used for reading different data from the interface, and the user must supply parameters specifying the format and amount of data to read from the interface. The selection from the drop-down menu just below the block specifies both. In our example we need to measure an analog signal, and this is done by selecting option »Analog« from this menu; the selection opens another sub-drop-down menu. We need to read one single analog channel, and this is done by selecting the option »Single Channel«; this opens another sub-sub-drop-down menu. We need to read one single result, and this is done by selecting the option »Single Sample«; the last sub-drop-down menu is opened, here we select the format for the result, in our example we opt for a real number, double precision.
- Once this block starts executing the ADC is started, and the block waits for the result to be passed from the interface. If all is fine the waiting will be short, but if something goes wrong we do not want the block to keep waiting. To prevent the indefinite waiting the upper limit for the time to wait is set to 1 second by wiring one to the block at the middle connector, left side.

The execution of this block returns the result of the measurement at the middle connector, right side of the block; the result is wired to a numeric indicator named »Result«. There are two additional outputs from the block; the identification code and the error message. Both are passed to the next block used to break the connection between the driver of the interface and user program, releasing the interface for other programs. If there was an error detected, then this error is presented to the user within an error window.

The example deals with a single measurement on a single channel. When more analog signals are to be measured simultaneously, these are connected to more than one input channels. These channels should be listed in the »Physical Ch« as, for instance, »Dev1/ai0, Dev1/ai1, Dev1/ai2« for three analog signals. This also means that we need to read more than one result (three actually), and this should be defined in the drop-down menu for the second block. The option »Multiple Channel« should be selected, the rest remains the same. Such selection returns an array with three elements from the second block, and the numeric indicator must be replaced by an array containing three numeric indicators to properly present results. The last block of the chain remains unchanged.

One-time only periodical sampling of analog signal(s)

An example of a Labview program for one-time periodical sampling of an analog signal is given in Fig. 2. The samples are taken at regular time intervals, it has been selected to take 1000 samples of a single analog signal, and the time interval between successive samples is 100µs. The string of results should be presented to the user in a form of a diagram at the screen of the computer.

The program starts the same way as the program for the previous example (Fig. 1). The leftmost block (step 1) establishes the communication between the driver of the interface card and the user program, and returns the identification code of the interface and (possibly) the error code; both are passed to the second block. For periodic sampling the read (as defined in the previous example) might be periodically repeated in the program. However, such technique does not allow exact periodical sampling, and also cannot provide fast sampling due to the lag in retrieving results from the interface; a better technique is required. The hardware of the interface includes a timer which can define time intervals between successive samples, and the samples are automatically stored in the memory and can be passed to the user program as a string once all samples are acquired. The second block in the chain (step 2) is used to initialize the timer and define sampling interval.

- The block should initialize the timer to act as a trigger for sampling, and this is selected in a drop-down menu just below the block, where the option »Sample Clock« needs to be selected. Since the clock signal for the timer is generated internally in the interface, we do not need to define the name of the clock signal used, and the third connector from the top at the left side of the block is left unconnected.
- We want to acquire a fixed number of samples of the input signal, and this mode should be passed to the block as a parameter; the connector at the top of the block is wired to a drop-down selector, and the »Finite Samples« is selected. The required number of samples is wired to the second connector at the top of the module, and the number of samples per second (10000 in this case) is wired to the second connector from the top at the left side.

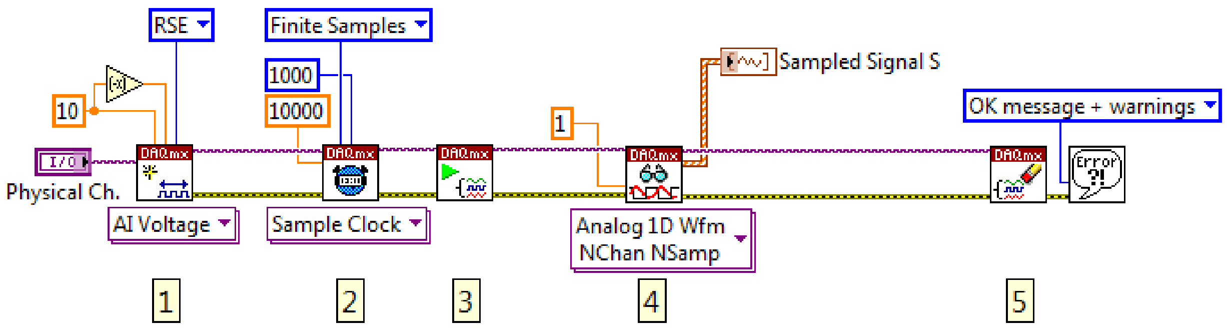

Figure 2: A Labview program for one-time only periodical sampling of an analog signal

There are again two outputs from the block, these are the identification code of the interface and the error message, and are passed to the next block. The third block (step 3) starts the sampling; its outputs are again passed to the next block.

The fourth block (step 4) is used to read the resulting string of results from the interface into the user program.

- This time a string of samples is to be read, and the block for reading should be prepared for this; the drop-down menu just beneath the block specifies what to read and how to present it. Clicking on the menu opens a drop-down menu, and the option »Analog« should be selected. This opens a sub-drop-down menu, where the number of analog signals read can be selected. The option »Single Channel« will do here and this opens another sub-sub-drop-down menu, where we select to read a string of samples by selecting the option »Multiple Samples«. This again opens a new sub menu with possible formatting of the retrieved samples. This can be either a simple one dimensional array of real values (»1D DBL«) or a cluster of various data named »Waveform«. The last includes the one dimensional array, the time interval between samples, and some additional data. This seems simpler to present in a diagram, and is selected in this example.
- The expected string of samples might not be available immediately, since the interface needs at least (number_of_samples x time_interval) to acquire the complete string; the block can wait for the string some time. For this example one tenth of a second would suffice. The maximum waiting time is wired to the middle connector on the left side of the block, where number 1 is wired as a good measure. If the string with samples is not read within one second, the block will break its execution and report an error.

The string of results is formatted as a waveform and is available at the right side of the block. The waveform format is represented by a thick striped line, and can be passed to the graph indicator without any additions. The graph indicator resizes and adjusts the scale to accommodate the waveform automatically.

As expected the end of sampling must be terminated by breaking the connection between the driver of the interface and the user program, and this is the task of the last block in the chain (step 5). In case of an error the message is presented to the user.

If more than one analog signal is to be sampled, then the modifications are similar to the ones described for taking a single sample. The channels used in sampling must be specified within »Physical Ch.«, and the way the resulting string is reported must be changed in the drop-down menu beneath block for step 4. The resulting output from block 4 is an array of waveforms in this case, and this can also be connected to the graph indicator; several lines will be displayed in the graph.

Continuous periodical sampling of an analog signal(s)

Continuous sampling could in theory be performed by repeating steps 3 and 4 from example in Fig. 2. However, this would introduce significant gaps between strings of samples caused by the finite time needed to read results before next acquisition is started. A better technique is needed. Luckily, the modification requires only a change in operating parameters, and the addition of a loop. An example of a Labview program for the continuous periodical sampling of an analog signal is given in Fig. 3. The following list summarizes the modifications:

- The triggering of the ADC should not stop when a string of samples is acquired; therefore the parameter for the operation of the second block in the drop-down menu above the block should be changed from »Finite Samples« to »Continuous Samples«.

- The triggering section of the interface does not need to know about the number of samples in a string, since the sampling is continuous; the corresponding connector of the second block is left open.
- The user program will, however, read strings of samples from the interface once a sufficient number of samples is available for reading, and the block for reading should know how many samples to expect. This number is wired to the second connector from the top on the left side of the fourth block; it is 1000 in the example program. The execution of the block is suspended until this number of samples is available, and then read. However, to stop the indefinite waiting in case of an error the maximum waiting time is wired as before and specified to be 1 second.
- The step 4 can be repeated indefinitely by putting it into a (while) loop. Each repetition will return a new string of samples which is seamlessly connected with the previous string; there is no time gap between two adjacent strings of samples. The execution of the loop and the sampling can be stopped by a »stop« button inserted in the while loop, and is also stopped in case of an error during the sampling.

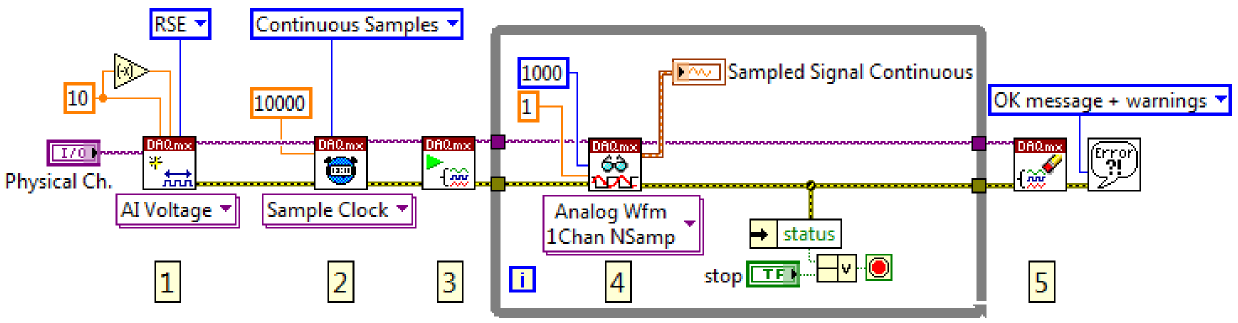

Figure 3: A Labview program for continuous periodical sampling of an analog signal

When sampling continuously the acquired samples are temporarily stored into an intermediate memory, which may be part of the personal computer or the interface itself; the driver reserves a memory space which should be adequate in normal operation. If the user program removes the acquired strings of samples from this memory fast enough, then the program runs as expected. If the user program for some reason does not remove samples from this memory fast enough, then the memory gets filled up, and new strings of samples might not have enough space to be stored; in this case the measurement is corrupted, and the driver stops the acquisition and reports an error. The user program must not let this happen; therefore the results must be read out from the memory fast enough.

The speed of reading is limited by the execution of all programs concurrently running on the computer, and the speed of data processing within the while loop in particular. The last can vary from a simple storing of the data to disk to sophisticated filtering and data processing. The drawing on the screen might be particularly time consuming. Not all computers are capable of running sophisticated on-line data processing. The Task Manager (Windows) provides an insight on the capabilities of the processor used and its current load.

The generation of analog signals using interface USB-6211 and Labview

The interface USB-6211 includes two digital to analog converters (DAC), which can be used to generate analog signals following digitally generated patterns. Both DACs have a resolution of 16 bits. The operation of the two DACs is supported by timing circuits and memory for strings of samples. The transfer of samples into the DACs and overall handling of the DACs is provided partially by the hardware, and partially by the device driver. The driver distributes the parameters for the operation of the two DACs, and the samples. The appropriate functions to reach the driver are stored in libraries, a part of the driver package for the Labview.

Analog signal can be generated in at least three different modes of operation:

- **Generating a constant voltage**
  The user program sends a digital code to the interface, and the DAC generates corresponding DC voltage. The analog output from the DAC is constant and remains unchanged until the user program supplies different code or the interface is turned off.
- **Generating repeatedly an analog signal, the signal samples are sent to the DAC once only**
  The user program sends a string of digital codes and parameters for the operation, and the interface autonomously generates analog voltages that correspond to the codes sent. The time interval between the successively generated voltages is a parameter for the operation. When the generation reaches the end of a string it repeats the string from the beginning.
- **Generating an analog signal continuously; samples are defined by the PC in real time**
  The user program sends a string of digital codes and parameters for the operation, and the interface autonomously generates analog voltages that correspond to the codes sent. The time interval between the successively generated voltages is a parameter for the operation. The user program must supply the successive string of digital codes before the generation of the current string reaches the end, otherwise the DAC runs-out of data; the driver then terminates the execution and reports an error. The next string of samples seamlessly follows the previously generated string; there is no time gap in between.

The generation in last two modes can be terminated by a command from the user program. Regardless of the mode of operation the user program needs to go through the following steps:

1. Initialize the path between the drivers of the interface and itself, including the passing of basic parameters for the interface and the DAC in particular.
2. Pass parameters to the unit within the interface which takes care about the time intervals between successively generated analog voltages (this step is omitted in the first mode).
3. Send digital codes representing the generated voltages.
4. Start of the generation, may be omitted in some modes.
5. Periodically check the status of the interface (this step can be omitted if we are sure the interface is working correctly).
6. Stop the generation (this step can be omitted in most cases, since step 7 stops the generation on its own before it disconnects the driver from the user interface).
7. Break the path between the driver of the interface and the user program.

Each step requires a separate function call to the library, each call requires parameters.

Generating a constant voltage

An example of a Labview program for generating of a constant voltage is given in Fig. 4. The outcome of the program is a fixed voltage at the analog output of the interface with a value corresponding to the code defined in the program.

The program starts with the leftmost block (step 1), Fig. 4. This block is used to connect the driver for the interface and the user program, and performs basic initialization of the DAC.

- The block needs to be told what to connect and initialize. Since an analog signal is to be generated we select the option »AO Voltage« (»Analog Output Voltage«) from the drop-down menu just beneath the block.
- There may be more than one interface connected to the computer, and the block needs to know the one the user is interested in. The name of the interface and the name of the analog output connector should be passed to the block at »Physical Ch.«. The typical name of the interface is »Dev1«, and a typical name of the analog output connector is »ao0« or »ao1« (or both). Both put together may look like »Dev1/ao0«.The allowed names can be found in NI »Measurement & Automation Explorer« software.
- The analog signal at the output of the interface is limited; the limits are symmetrical and should be passed to the block at two connectors at the top of the block. In the example these limits are +10V and -10V, where the lower limit is derived from the upper.

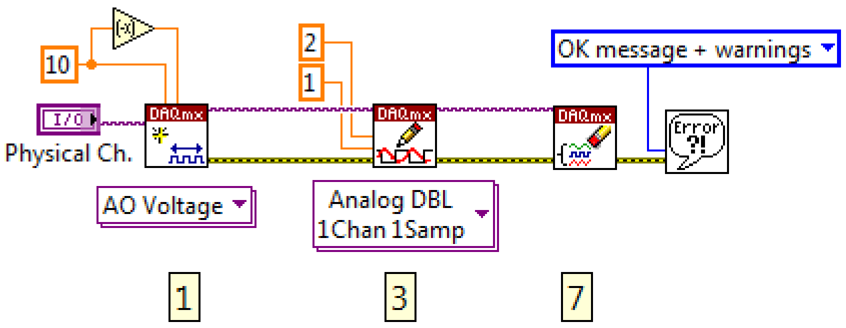

Figure 4: The Labview program to generate a constant voltage

The block returns an identification code of the interface and an error message. Both should be passed to the next block, as in examples for sampling of the input signal.

The next block (step 3) is used to send the code for the analog voltage to the DAC residing in the interface. The code can be is simply a voltage to be generated; code »2« produces an analog voltage of 2V. The block requires the identification code of the interface and error message from the previous block.

- The code wired to the block and sent further to the interface can be given in different formats. In case of generating a single constant voltage the code is a real value scalar, and this format should be selected in the drop-down menu just beneath the block: »Analog DBL, 1Chan 1Samp«. The precision of the code is »double«.
- The code itself is wired to the second connector from the top, left side of the block. The example connects double real value of 2 there.

- The interface may be busy, and the code cannot be transferred to the interface immediately. The block can wait, but we do not want it to wait indefinitely. The maximum waiting time should be specified at the third connector from the top, left side, and is given as 1 second in the example.
- There may be an error when the code gets sent. The error code is available at the connector at the lower right corner of the block, and the identification code of the interface at the upper right corner. Both should be passed to the next block.

When the execution of this block is finished the output of the specified DAC holds the desired voltage. The program can now break the connection between the driver and the user software and terminate; this gets done by the last block in the chain (step 7). This block takes the identification code of the interface and the error code from the previous block, breaks the connection and returns the final error code, which may be written to the screen of the computer.

When two DACs are to be driven simultaneously, the program from Fig. 4 can be used with minor modifications. The first block should be notified that more than one DAC is to be used, and this can be done by listing DAC names in the »Physical Ch.«, like »Dev1/ao0, Dev1/ao1« for two DACs. Secondly, the middle block should be prepared to receive codes for more channels by selecting a suitable option from the drop-down menu beneath the block; the option »Analog DBL, NChan 1Samp« should be selected. Finally, the code connected to the second connector from the top at the left side of the block should be an array of doubles. The length of the array should be the same as the number of DACs used.

If several consecutive analog voltages are to be generated, then the middle block from Fig. 4 can be put into a loop and a new value can be passed to the block within each repetition of the loop. When the generated voltages are to be periodical, a time delay can be inserted into the loop. However, such generation has its disadvantages. The most important is the fact that the minimal time interval between two successive samples is limited by the speed of the computer and the speed of the interface in particular. Additionally, the time interval is influenced by tasks running concurrently on the PC, and is therefore not reliable. When the user needs to generate periodical signals, the mode of generation must be changed and the timing put under the control of the interface, as described in the following two chapters.

Generating repeatedly an analog signal

An example of a Labview program for generating a string of analog voltages is shown in Fig. 5. The string of codes and the required time interval between the successive analog voltages at the output of the DAC is sent to the memory of the interface once only at the beginning of the program, and this string of codes is then automatically repeated by the interface without the intervention of the PC. The codes for the signal can be anything within the range of the DAC, and the length of the string is under control of the user. It is therefore possible to generate any periodical signal without the PC, the computer is used only to initialize the interface and calculate the codes.

The program starts the same way as in the previous example by connecting the driver with the user program. The second step is required to define the operation mode of the interface and to pass the parameters to the timing circuit, which will take care about the time intervals between successive voltages at the output from the DAC.

- The interface can synchronize the generation of voltages with many sources; here the periodical clock signal present within the interface is used. This is selected from a drop-down menu just beneath the block where the option »Sample Clock« is selected.
- The interface should generate voltages continuously, therefore the timing circuitry within the interface must also work continuously. The mode for continuous generation is selected from the drop-down menu above the block as »Continuous samples«.
- The time interval between successively generated voltages is selected to be 100μs, which gives 10000 samples per second. This number is wired to the second input from the top at the left side of the block.

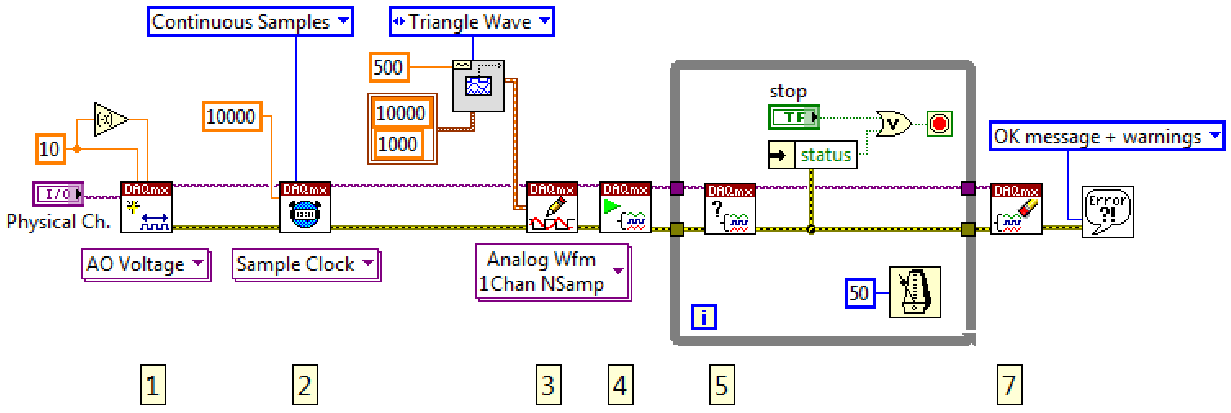

Figure 5: The program to generate the same analog signal repeatedly

The block returns the identification code and the error code; both are passed to the next block in the chain. The next block (step 3) is used to pass the string of codes to be converted to analog voltages. The example program uses a triangle wave generator to prepare codes in a format of a waveform, and the triangle wave generator requires its own parameters. It seems logical to generate the codes at the same frequency as later used to trigger the DAC. The length of the string of codes should be long enough to cover at least one full period of the triangle wave, but not too long to overload the memory of the interface or the USB. The example selects 10000 as the frequency of sampling, and 1000 as the number of codes within the string. The frequency of the generated triangular signal is 500 Hz, giving 50 periods of the signal within the string of codes. The shape of the generated signal is selected from the drop-down menu above the block. This concludes the preparation of the string of codes within the PC.

- The block in step 3 should be aware of the format for the sting of codes. This is selected from the drop-down menu just beneath the block, where option »Analog« is first selected; this selection tells the block to pass parameters to the analog section of the interface, and opens a sub-drop-down menu. The codes are destined for one channel only, so the option »Single Channel« is selected next, and opens another sub-drop-down menu. A string of samples is to be sent, and this is selected by the option »Multiple Samples«. The last sub-drop-down menu gets opened, and the option »Waveform« gets selected, since this is the format of the string of codes prepared for the block.

- The prepared waveform with the string of codes is passed to the block through an input at the left side of the block.

The return values from the block are, again, the identification code and the error message. Both are passed on to the fourth block. The timing has been defined, and the string of codes is now in the interface. The generation can now be started. The fourth block does just this: it starts the generation. The interface starts by generating the first code is the string, and after the time interval defined in the second step takes the second code from the string and generates it, and after another time interval takes the third code... until it reaches the last code from the string and generates it. After another time interval it again takes the first code from the string and generates it... The interface repeats the generation autonomously, and interventions from the user program are not needed any more. However, an intervention may be required in case something goes wrong in the interface or when the user wants to stop the generation of the signal. To accomplish this a while loop follow the fourth step. Within the loop the status of the interface (step 5) is checked periodically every 50 ms; the time interval of 50 ms is defined by the timing function within the loop. The block in step 5 passes the identification number of the interface and returns the error code, if any. The status information is derived from the error code, and is OR-er with the status of the button »stop«. If both are false the loop continues and the so does the generation. If any of them is true, the looping ends, and the block in step 7 gets executed. This block breaks the connection between the driver and the user program, and returns the error code, which can be presented to the user.

Generating an analog signal continuously; samples are defined by a PC in real time

The data processing applications usually do not know in advance what signal is to be generated. This is either calculated from the sampled data or defined by the user, and changed during the execution of the program. In those cases the signal to be generated must be calculated on-line.

The codes representing the signal must be sent to the interface in strings. It seems reasonable to prepare / calculate these strings of codes in packets of the same length. The Labview example in Fig. 6 demonstrates such technique by generating a triangular signal, where the user can dynamically change the frequency of the generated signal by changing the numeric control named »Frequency«. The block for the generation of the triangular signal and the block for step 3 are inserted into a loop to be repeated.

The program commences the same way as previous programs for generating: by connecting the driver of the interface to the user software. The result of the block in step 1 are again the identification code of the interface and the error message, and both are passed to the rest of the program.

By default the interface is ready to repeatedly generate a string of codes. However, this is not what is needed in this case. The default value must be changed, and this is done by a function, where the repeatable mode must be changed to a non-repeatable. The function »DAQmx write« is capable of fine tuning the operation of the interface by changing each individual setting, and can be put on the screen by pointing the mouse to the first block and pressing the right mouse button. This brings out a menu, where the option »DAQmx - Data Acquisition Palette« must be selected. The selection opens another menu, and the option »Write Node« is the one searched. The function obtained is to be placed in the program, and the blue text should be clicked. From the opened menu the option »RegenMode« should be selected. The function requires the identification code and the input error message, and additionally the newly required mode; the last is defined in the drop-down menu above the function as »Do Not Allow Regeneration«. The outputs from the function are again the error code and the identification code, which are passed to the next block (step 2).

The next block in the chain defines time intervals between successive samples, as already seen in the previous example. The third step is almost identical to the third step in previous example. This step fills-up the memory of the interface with initial data, which are be used at the beginning of the generation. Note that this string is composed of 2000 codes to thoroughly fill the interface, and then in step 4 the generation is started. The reason for extending the initial string of codes is that in continuous mode the PC must send next string of codes to the interface before the previous string is exhausted or an error occurs. In order to prevent the error we shall make the interface busy enough, and this requires the lengthier initial string of codes. Now the PC has ample time to calculate next string of codes, and send them to the interface in the repeated step3 (both within the loop; note the normal length of the string of codes for the triangular wave generator).

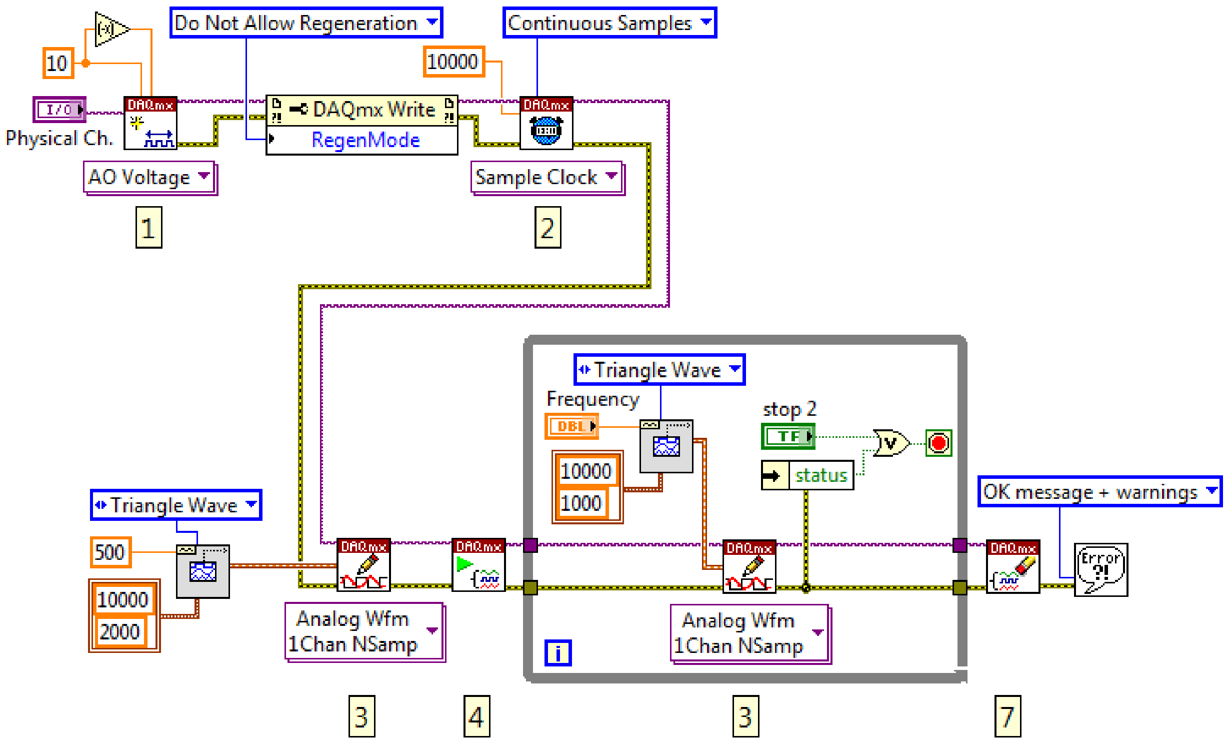

Figure 6: The Labview program to generate an analog signal continuously

The execution of the loop is stopped when the user clicks the button »stop« or when there is an error, as in the previous example, and then the connection between the driver and the user software is broken and the possible error message presented to the user.

Continuous sampling, calculating, and generating of analog signals: on-line data processing

By combining the two examples on continuous sampling of analog signals and continuous generation of analog signals we get a program for the on-line signal processing. The simplest example will only pass the acquired signal to the output of the DAC, block diagram in Fig. 7. Once the skeleton for the on-line data processing is available the passing of samples can be replaced by more sophisticated signal processing, including filtering, modulation, demodulation, phase shifting, and alike.

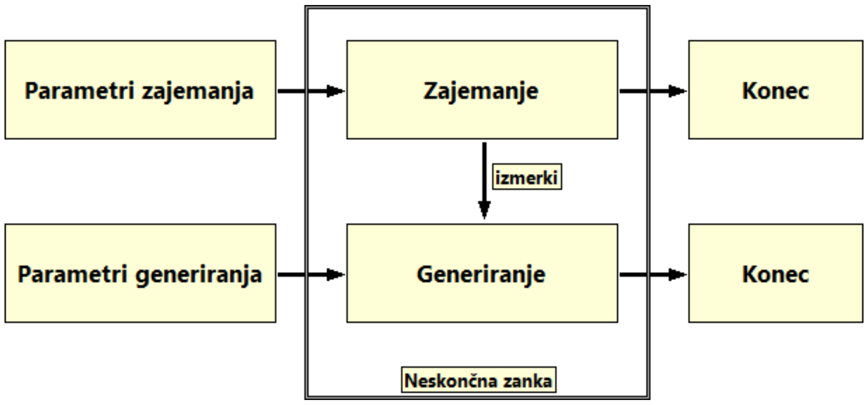

Figure 7: The block diagram of the structure for on-line signal processing; whatever is sampled as analog input signal is passed to the analog output for generation

The combined skeleton for signal processing is given in Fig. 8. The upper chain serves for the sampling and the lower chain for the generation. Both chains are copies of those given in figures 3 and 6. Both chains exchange data using a single wire inside of the loop; the sampled data from the sampling chain is passed directly to the generation chain. The generator of triangular signal is removed from the generation chain within the loop, since the samples are now supplied by the sampling chain. The sampling rate of both chains and the number of samples within a string must be the same for both chains. The execution of the loop can be stopped by clicking on the button “stop”, and the execution is automatically stopped in case of an error in any of the chains.

The example in Fig. 8 uses a relatively short string of 1000 samples. With the sampling rate selected (10 kHz) the program requests the transfer of the acquired samples from the interface to the computer and the generated samples from the computer back to the interface 10 x 2 = 20 times per second. In each transfer 1000 samples each consisting of 2 bytes is transferred; this gives a sum of 4 kB of data, and is easily handled by the USB. When the sampling rate is increased to for instance 200 kHz, the same amount of data is involved in each transfer, but the transfers are much more frequent (400/s); there should be one transfer every 2.5 milliseconds. This might be more than a USB can handle in real time, and the interface does not receive data on time, which leads to an error and stops the execution of the program. The problem can be solved by increasing the number of samples in one transfer by increasing the length of the string of samples to, say, 10000. Then only 40 transfers per second are needed, and this can be handled by the USB.

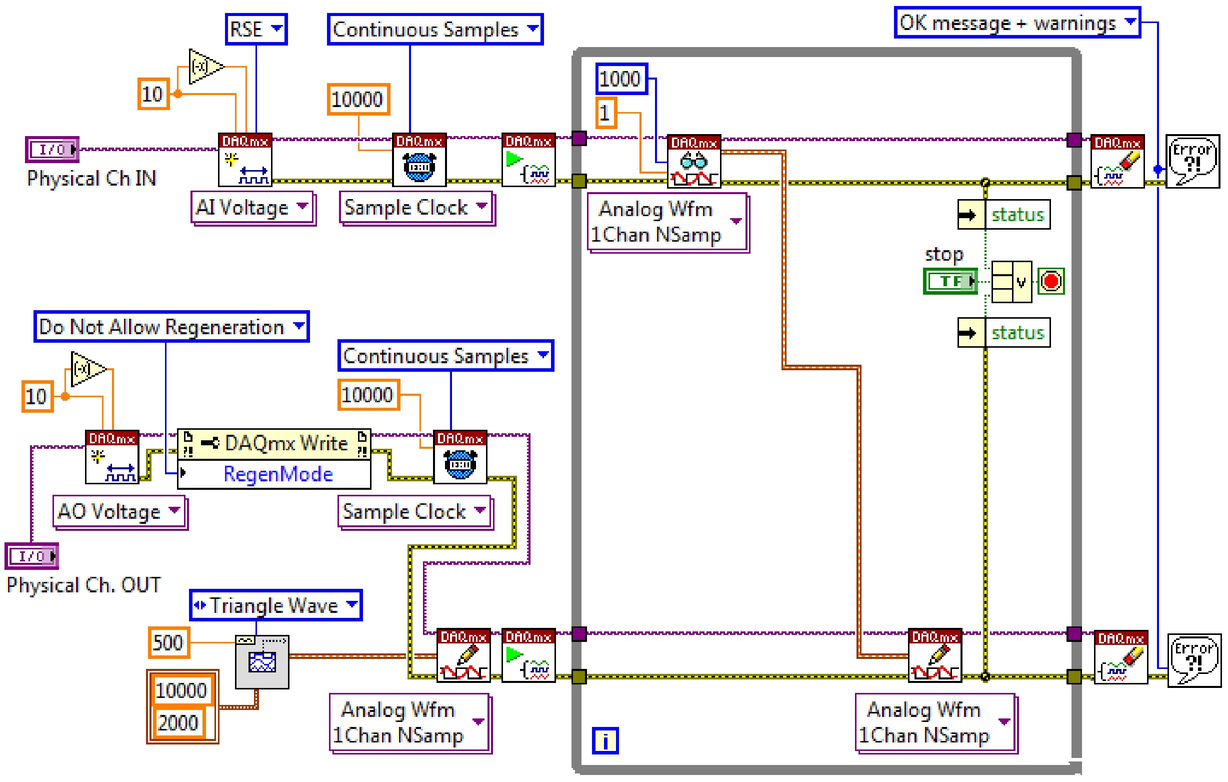

Figure 8: An example of a Labview program for continuous sampling and generation of analog signals

Soundcard: continuous data processing using Labview

The sound card consists of a two channel ADC and DAC, and includes a timing unit and memory for samples. It is optimized for the acquisition of analog signals with frequencies from 20 Hz to 20 kHz. The operation of the sound card is autonomous; the PC supplies only the data and operating parameters through sound card drivers and libraries.

The sampling frequency is selectable from a list of possible values, but the most often used is the 44100 Hz; this is just above the Nyquist frequency for audio signals. A typical amplitude of signals is up to few hundred 100 mV.

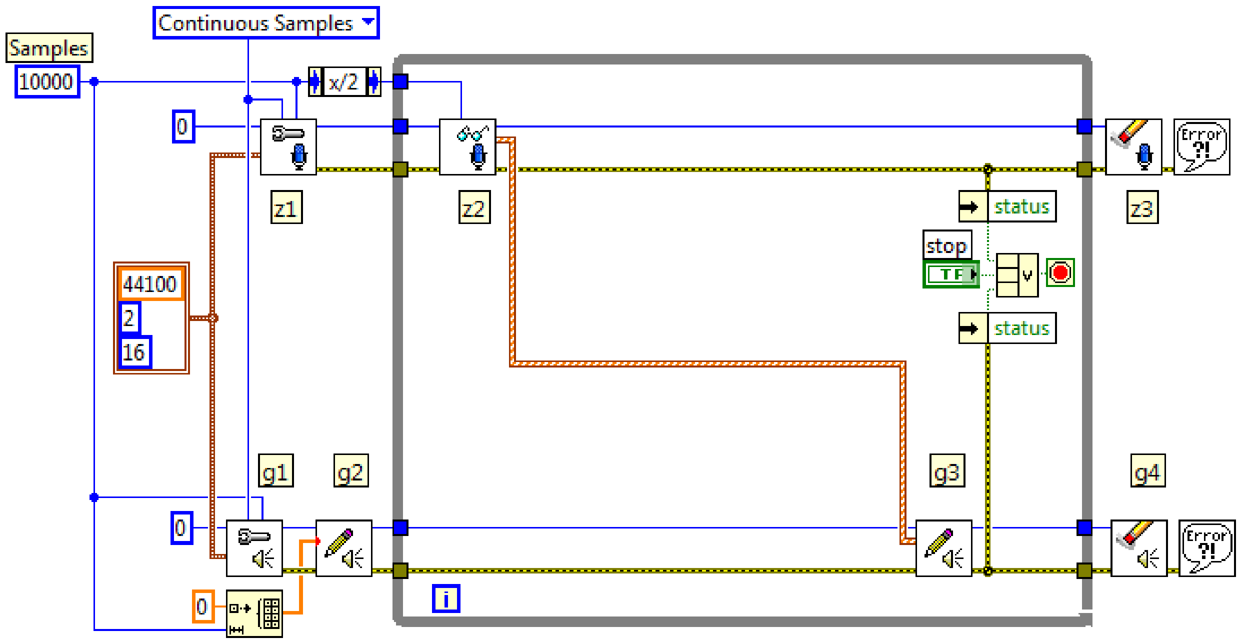

Figure 9: An example of a Labview program for continuous data processing using a sound card

An example of a program to simultaneously sample, process, and generate audio signals using the sound card and a PC is given in Fig. 9. The upper chain is used for sampling of the input audio signal, and the lower chain for generating. The two chains are connected within the loop by simply passing the acquired data to be generated. Here the card simply repeats whatever is connected to its input, but the wire can easily be replaced by blocks for continuous data processing. The execution of the loop can be stopped by clicking the button »stop«, and is also stopped in case of an error.

Both chains run using the same parameters: the sampling rate is 44100 s^{-1}, stereo (2 channels), 16 bit. These parameters are stated at the left side of the program, and are passed as a cluster to both chains. The first block in each chain (z1 in g1) is used to connect the driver of the sound card with the program. The two blocks require some parameters, and the cluster describing the sampling is one of them. The second parameter to each block is the identification number of the sound card used; typically there is only one sound card in the computer, and the identification number of this card is 0 (connected to the first input from the top, left side of each block). The length of the string of samples needs to be passed as well, and is 10000 in the example. Both chains operate continuously, there are no time intervals caused by the retrieval of the samples from the card or to the card. This mode is selected from the drop-down menu above the upper chain as »Continuous samples«. Both blocks return identification codes at upper right corners and error codes at lower right corners. These are passed to subsequent blocks.

As with the NI USB interface the sound card must be kept busy all the time. This means that the newly acquired samples must be retrieved from the card regularly, and that the new samples must be sent to the sound card before the current string of samples is exhausted. It is therefore mandatory to fill the sound card with artificial string of samples before the actual data processing starts. This is done in the lower chain just before the entry to the loop within block named g2. The artificial data is constructed as an array of 10000 elements, all having value of zero. The red dot at the input to block g2 signals that the Labview makes necessary reformatting to adjust for the requirements of the block.

Within the infinite loop that follows both chains are active. The upper is used to retrieve a string of samples from the sound card; the string is 5000 samples in length. The block z2 waits until the appropriate number of samples is prepared by the sound card, and then reads them and passes them to the lower chain in a format of an array with two elements. Each element is a waveform, a cluster which includes the string of samples taken and the sampling rate. This format of data can be directly passed to the chain for generating.

When the execution of the loop is stopped by pressing the button »stop« or due to an error, the blocks z3 and g4 break the connection between the driver and the program, and pass the code of the error to the display.

The wire connecting the two chains can be replaced for more sophisticated data processing, like filtering, amplification, and other mathematical operations. The data retrieved by the upper chain can be displayed to the screen by simply adding a graph and wiring the output of block z2 to the graph.