Digitizer~

Real-Time connection between Max/MSP & Universal Library
## I. Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>II. Preface</td>
<td>3</td>
</tr>
<tr>
<td>III. Digitizer~ Capabilities</td>
<td>4</td>
</tr>
<tr>
<td>IV. Software &amp; Hardware installation requirements</td>
<td>5</td>
</tr>
<tr>
<td>V. Digitizer~ Installation</td>
<td>6</td>
</tr>
<tr>
<td>VI. How to use the Digitizer~ in MSP</td>
<td>10</td>
</tr>
<tr>
<td>VII. How does the samples are collected from the channels</td>
<td>16</td>
</tr>
<tr>
<td>VIII. Error messages</td>
<td>19</td>
</tr>
<tr>
<td>IX. Verifications</td>
<td>20</td>
</tr>
<tr>
<td>X. Relevant docs</td>
<td>25</td>
</tr>
<tr>
<td>XI. Thanks</td>
<td>25</td>
</tr>
<tr>
<td>XII. The MSP digitizer.c Code</td>
<td>26</td>
</tr>
</tbody>
</table>
II. Preface

The Digitizer~ is a Max/MSP external object that performs analog to digital interface that connects Max/MSP projects with Universal Library boards. This object was developed and tested for the PMD (=Personal Measurement Device) -1608FS.

This project has been written as part of a workshop in Multivariate Modeling Sound and Interactive Multi-Media, course number 0368-3500-07, autumn 2004-5, under supervision of Prof. Nathan Intrator.
III. Digitizer~ Capabilities

- Connect signals received via boards from Universal Library into Max/MSP environment.
- Serve as a signal source to MSP objects.
- Analog to Digital 16 Bit Sampler.
- Sample up to 8 different signal channels simultaneously.
- Configurable sample rate: from 1000 Hz to 8000 KHz.
- Configurable signal vector size: from 16 samples to 2048 samples per vector.
- Configurable input resolution: for 10V peak to peak input voltage down to 1V peak to peak input.
IV. Software & Hardware installation requirements

- Universal library installed (checked & verified for version 5.51).
- Max/MSP version 4.3 and higher.
- PMD-1608FS or any other Universal Library device connected to your PC.
V. Digitizer~ Installation

First Installation
- Load Max/MSP
- Select File-Install.
- A file open box will be displayed on screen.
- Browse and select the Digitizer~.mxe file and double click it.
- The Following message will be displayed on the Max message window: “digitizer msp object loaded...”.

Automatic Digitizer~.mxe Installation
- After the first installation you can configure the Max/Msp to load the Digitizer~ object automatically by entering the Digitizer~.mxe path in Options-File Preferences.

Test your installation
A small test program for Max/MSP (=patch) as been written to test the installation of the Digitizer~. We strongly recommend that you will perform this test. In order to test your installations follow the following instructions:

- Use File-Open menu and select Digitizer_test.txt file
- The following messages will be displayed on the Max message window:

![Max Message Window]

This screen describes the default configuration, which can be controlled as will be explained later on this document.

- The following screen should be displayed on the patch window:
- Check that your device is connected to the PC and is ready to perform samples.

- Check and configure the various parameters of the sampling (circled in orange color on the following screen print):
  - Board (0-4): the source board your device is connected to.
  - Low Ch (0-7): Low Channel, the lowest channel number in the board you wish to sample from.
  - High Ch (0-7): High Channel the lowest channel number in the board you wish to sample from.
  - Rate (0 – 3): The desired sample rate index. Select 0 for 1KHz, 1 for 2KHz, 2 for 4 KHz & 3 for KHz.
  - Gain (0-16): The desired gain index, the index table will be explained in the How To Use the Digitizer~ in MSP section. For the installation you can use the default value which is 0 (5 V peak to peak).
Control the MSP sampling rate: make sure that the values in the Options – DSP status menu are configured as following:

- The Sampling rate must be configured to 16000 Hz! The Signal Vector Size must be greater than 16. The I/O Vector Size must be no lower than the Signal Vector Size. We recommend choosing Signal Vector Size and I/O vector size of 2048 in order to decrease software interrupts overhead.
- Press “ON” (the green box button on the upper right corner of the patch window). The message “PMD started” will be displayed on your Max window.
- 8 (or less, according the channels being sampled) signals should be displayed on the scopes. The screen should look like something this:
In this screen print we can see 8 channels are being sampled, while 3 of them are getting a signal in there input.

- Press “Off” to stop the sampling. The message “PMD started” will be displayed on your Max window.
- The test passed successfully.
VI. How to use the Digitizer~ in MSP

If you need to use the connection between Max/MSP and Universal library device we recommend following the following simple steps:

- **Add a new Digitizer~ object and an adc~ object by using one of the following methods:**
  - **The best way:**
    Is to use the “digitizer_test.txt” (or “digitizer_test.mxb” which is exactly the same patch only with a different file format) file as a template file. All you have to do is to save this file in a different name and build your patch on the working-patch.
  - **An alternative way:**
    Start a new patch.
    Declare a new object box and rename the object as “Digitizer~” (don’t forget the ~’). Declare a new object box “adc~”, which serves as a trigger to the Digitizer~ object and the whole DSP chain in Max/MSP.

- **Digitizer~ configurations:**
  The digitizer has 8 signal outlets and 5 message inlets as described in the following figure:

![Digitizer~ configuration](image)

The 5 input outlets are on the top of the object while the 8 output outlets are in the bottom of the object (like any other Max object).

**Setting the input outlets:**
There are two options to configure the Digitizer~: the first by using the 5 number boxes just above the digitizer~ object, the second is by using the “command line” message box which is located just left from number box.

First of all we will explain how to **configure the Digitizer~ using the number boxes:**

![Configuration boxes](image)

The configuration boxes

Pay attention to the fact that each time the user changes a value in one of these number boxes the new configuration setting will be displayed on the Max patch window under the title “sampling configuration”. A sampling configuration message is displayed in the following figure:

-10-
Also pay attention that if sampling process is on, each change to the number boxes will stop the PMD from sampling in order to allow the user to configure its new settings, so in order to apply the new sampling configuration the user must press the “ON” button.

- The left input controls the board number to sample from. Valid inputs are (0-4). Board number 0 is the demo board and the other boards are the boards which are installed in your PC and currently connected to your PC.

- To the left of the board number configuration box located the number box which controls the lowest channel to scan. Valid input are channels 0 to 7 but a number which is equal or lower to the highest channel number to scan.

- To the left of the lowest channel to scan located the number box which controls the highest channel number to scan. Just like the previous number box valid inputs ranges from 0 to 7, and not lower then the lowest channel number to scan.

- To the left of the highest number box to scan, located the number box which controls the sampling rate of the Digitizer~. Valid inputs are 0-3 which represents the 4 available sampling rates as described in the following table:

<table>
<thead>
<tr>
<th>Rate number box</th>
<th>Digitizer~ sampling rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000 Hz</td>
</tr>
<tr>
<td>1</td>
<td>2000 Hz</td>
</tr>
<tr>
<td>2</td>
<td>4000 Hz</td>
</tr>
<tr>
<td>3</td>
<td>8000 Hz</td>
</tr>
</tbody>
</table>

- The rightmost number box controls the input gain of the sampled channels. The input to this number box ranges from 0 to 16, but valid input is depends on the specific board installed. For example, the PMD-1608FS may samples only with gains 5V peak to peak, 10V peak to Peak, 1V peak to peak, and 2V peak to peak. The input values and their corresponding Input range described in the following table:

<table>
<thead>
<tr>
<th>input</th>
<th>Input range[volt]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-5 to +5</td>
</tr>
<tr>
<td>1</td>
<td>-10 to +10</td>
</tr>
<tr>
<td>2</td>
<td>-2.5 to +2.5</td>
</tr>
<tr>
<td>3</td>
<td>-1.25 to +1.25</td>
</tr>
<tr>
<td>4</td>
<td>-1 to +1</td>
</tr>
<tr>
<td>5</td>
<td>-.625 to +.625</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>6</td>
<td>-.5 to +.5</td>
</tr>
<tr>
<td>7</td>
<td>-.1 to +.1</td>
</tr>
<tr>
<td>8</td>
<td>-.05 to +.05</td>
</tr>
<tr>
<td>9</td>
<td>-.01 to +.01</td>
</tr>
<tr>
<td>10</td>
<td>-.005 to +.005</td>
</tr>
<tr>
<td>11</td>
<td>-1.67 to +1.67</td>
</tr>
<tr>
<td>12</td>
<td>-0.25 to +0.25</td>
</tr>
<tr>
<td>13</td>
<td>-0.2 to +0.2</td>
</tr>
<tr>
<td>14</td>
<td>-2.0 to +2.0</td>
</tr>
<tr>
<td>15</td>
<td>-20 to +20</td>
</tr>
<tr>
<td>16</td>
<td>-4 to + 4</td>
</tr>
</tbody>
</table>

The default value is -5 to 5 volt. The input voltage range affects all the current sampled channels. There is no way to set a different gain between two or more different channel (This is a Universal Library setting in the command cbAInScan which we can not override).

Now we will explain how to configure the Digitizer~ using the “command line message box”:

The purpose of the command line is to enable the user to enter selected configuration of sampling by entering and controlling a single object. Any setting done by the message control box can be entered using the number configuration boxes and vice versa.

The command line format is as following: The first number (=the leftmost) is the Board number to sample from. The second number is the lowest channel to scan, the third number is the High channel to scan, the forth number is the sampling rate, & the rightmost number is the sampling gain. Pay attention that unlike the number configuration box the rate is entered explicitly and not by the sampling index. For example, in order to sample in 4Khz the user should enter the number 4000 the command line while if the user wise to use the forth (from left) number box to sample in the same rate he should enter the number “2”.

**Setting configuration to the Demo Board.**
The demo board number is 0. Unfortunately, there is a lot of difference between working with the demo board and working with real boards. Anyhow, the Digitizer~ allows the user to check connection with the demo board. When working with the demo board the following changes are applied:
- Only one channel can be tested instead of up to 8 with a real board.
- The Signal vector size can be any number allowed by Max.

An explanation about the arrangement of the sample in the Vector size will be brought later on this document.

**Setting the output outlets:**
As already mentioned above the Digitizer~ object has 8 outlets stationed in its bottom. The leftmost outlets samples channel number 0, the outlet to its right samples channel number 1, and so on... so the rightmost outlet samples channel number 7 (which means the Digitizer~ can sample 8 channels at the same time).

There is no problem in by not sampling all the channels at the same time.

The Digitizer~ output values range is from 0 to 65535 (unsigned 16 bit), in order to place the output between -32768 to +32767 (16 bit signed) the “Digitizer_test.txt” uses a MSP object “+~ -32767”.

Setting the DSP options

The setting of the MSP output rate and buffer size is done by using the standard interface of the Max/MSP workspace.

From the menu, choose Option – DSP status... option. A new purple window will be display on screen. The middle part of this window is as the following figure:

![The DSP Status Window](image)

The Signal Vector Size configures the buffer size of the samples. The Sampling Rate should be set to 160000. The Signal Vector Size should be set to 2048 and the I/O Vector Size should be set to 2048.

The User can enter Signal Vector Size of 16 and upper but this may affect the interrupt overhead of the program. See “How does the Digitizer~ works” for further explanations”.

Using more then one Digitizer~ objects in your patch:

Will just won’t work.

The preset board:

Can store and output previously used configuration. For more info about this device please refer to the Max help about “preset”.

-13-

Digitizer~ documentation
How does the Digitizer~ works

**MSP vs. Max overview:**
MSP objects are connected by patch cords. Rather than establishing a path for messages to be sent as in Max, MSP connections establish a relationship between the connected objects, and that relationship is used to calculate the signal information necessary at any particular instant. This configuration of MSP objects is known as the signal network. The Digitizer~ is an external object written in C that samples analog board and translates the samples into digital signal.

Another way to think of a MSP signal network is as a portion of a patch that runs at a faster (audio) rate than Max. Max, and you the user, can only directly affect that signal portion of the patch every millisecond. What happens in between those millisecond intervals is calculated and performed by MSP. If you think of a signal network in this way—as a very fast patch—then it still makes sense to think of MSP objects as sending and receiving messages (even though those messages are sent faster than Max can see them), so we will continue to use standard Max terminology such as send, receive, input, and output for MSP objects.

**Universal library overview:**
Universal Library is an interface that connects any of Measurement Computing data acquisition and control boards. The library is universal in three ways:

Universal across boards: The library contains high level functions for all of the common operations for all boards. Each of the boards has different hardware but the Universal Library hides these differences from your program. So, for example, a program written for use with one A/D board will work as is with a different A/D board.

Universal across languages: The Universal Library provides the identical set of functions and arguments for each supported language. If you switch languages, you will not have to learn a new library, with new syntax, and different features. If you are programming for the NET framework, you will find that the Universal Library for NET has the same look and feel as the Universal Library for 32-bit windows applications, and is just as easy to program. Languages supported by the Universal Library, at this time, are listed in the following table. Both 16 and 32 bit versions are supported where applicable.

Microsoft windows Visual Basic C++ Visual C/ C++ Quick C for Windows Microsoft C
Languages Borland Windows Languages Borland C++ Builder Delphi
Watcom C++
Microsoft DOS Languages QuickBasic 4.5 Professional Basic 7.0 Visual Basic for DOS Quick C
Borland DOS Languages Turbo C Turbo C++ Borland C++
Hewlett Packard (Now Agilent) VEE HP.
.NET Languages VB, NET C#. NET

Universal across platforms: The Universal Library provides the same sets of functions for DOS, Windows 3.x and 32-bit Windows (95/98/ ME/ NT/ 2000/ XP). Additionally, these functions have been extended to support the. NET environment.

**The Perform routine:**

-14-
The perform routines in the C++ code are the routines that the MSP performs at interrupt time. In the Digitizer~ case each time a perform routine is called a Signal Vector Size is filled with data.

As with any interrupt routine, the perform routine is written as efficiently as possible. It cannot call routines that would move memory, nor should it call post (for debugging), since at a 44.1 kHz sampling rate and vector size of 256 samples, each perform routine is called about every 5.8 milliseconds. This is the reason why we recommend working with signal vector size of 2048, in this case less interrupts are performed.

**Digitizer~ Performances:**
The faster your computer’s CPU, the better will be the performance of MSP. Turning off background processes (such as file sharing) will improve performance. Reducing the audio sampling rate will reduce how many numbers MSP has to compute for a given amount of sound, thus improving its performance (although a lower sampling rate will mean degradation of high frequency response).

Note: To see how much of the processor’s time your patch is taking, look at the CPU Utilization value in the DSP Status window. Choose DSP Status… from the Options menu to open this window.

According to our tests (AMD Athlon XP 2500 Hardware):
CPU Utilization for sampling 8 channels at the same time, at maximum sampling rate of 8 KHz and a Signal Vector size of 2048: around 2%-1%.
VII. How does the samples are collected from the channels

The Universal Library command used in the Digitizer~ is “cbAnScan”. In this section we will explain its options and its affects to the user.

cbAInscan() Scans a range of A/D channels and stores the samples in an array.

CbAInScan() reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the specified board. If the A/D board has programmable gain, then it sets the gain to the specified range. The collected data is returned to the data array. The command syntax is:

```c
int cbAInScan(int BoardNum, int LowChan, int HighChan, long Count, long * Rate, int Range, int MemHandle, int Options);
```

Arguments:
- **BoardNum**: The board number used to collect the data.
- **LowChan**: The first A/D channel of scan.
- **HighChan**: The last A/D channel of scan. Low/High Channel # -The maximum allowable channel depends on which type of A/D board is being used. On the PMD this number is 8.
- **Count**: Number of A/D samples to collect. Specifies the total number of A/D samples that will be collected. If more than one channel is being sampled then the number of samples collected per channel equals Count/(HighChan-LowChan+1).
- **Rate**: The sample rate at which acquisitions are triggered, in samples per second per channel.
- **Range**: A/D range code.
- **MemHandle**: Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the cbWinBufAlloc() function.
- **Options**: Bit fields that control various options.

The option we used for the Digitizer~ are: BACKGROUND, CONTINUOUS, and BLOCKIO.

**BACKGROUND**: cbAInScan() function will not return to the program until all of the requested data has been collected and returned to the buffer. When the BACKGROUND option is used, control will return immediately to the next line in your program and the data collection from the A/D into the buffer will continue in the background.

**CONTINUOUS**: This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that the programs will regain control.

**BLOCKIO**: the mode of transfer of the samples collected by the PMD to the PC memory. This mode transfers each time 31 samples and it is used to decrease the
transfer overhead. This is the fastest transfer method for boards without DMAIO transfer mode, like the PMD 1608FS.

When the user presses on samples are collected and transfers to the PC memory. On the pc memory they are stored in a memory buffer. This buffer is circular so no memory overrun can occur. At the same time the MSP read data from the same buffer at a rate of 16KHz, this data is being sent out from the Digitizer~ outlets.

In order to synchronize between the samples that goes into the memory buffer and the samples going out of the memory buffer we used duplication of samples. For example: is the board is sampling at 8000Hz each sample will be duplicated twice. If the board sampling rate is 2000Hz each sample will be duplicated 8 times, and so on.

**Collecting samples with the demo board:**
As mentioned earlier, the demo boards works differently from real boards so also its outputs is different: Each sample will be inserted as is to the whole buffer. For example if u set a signal vector size of 4, each buffer in the MSP will have the size of 4 places and in each place the same sample will be copied.

The sampling rate is how many total elements the Digitizer~ will deliver on its output.

Pay attention the “effective sample rate” which can be defined as how many different samples the Digitizer~ outputs per second is:

\[
\frac{\text{Sampling Rate}}{\text{Signal Vector Size}} \ [\text{Hz}]
\]

For example: for an effective sample rate of about 5.5 KHz set the Sampling rate of 11025 Hz and a signal vector size of 2. for this example each two samples will have the same value.

Pay attentions that due to Max/MSP restrictions the minimum sampling rate is 11025Hz and the minimum signal vector size is 2 (not 1!).

The values configure in this menu will applied to all the signal network of your MSP application.

The maximum values are depend on the speed of your machine, more about performance is described in “How does the Digitizer~ works” section.

Quantization effects:
Setting a large signal buffer size may lead to quantization-like effects like in the following figure:
Latency Issue:
The Digitizer has a latency of 1 – 2 seconds. This latency is due to the size of its internal buffer.
VIII. Error messages

Three levels of errors exist for the Digitizer~:
Universal Library errors: will be displayed in a special window in the middle of the screen.
Digitizer~ errors: will be displayed in the max patch window.

MSP/MAX : errors : will be displayed it the middle of the screen.

Universal Library Errors:
This error format:

Digitizer~ errors:
Invalid voltage (or current) range
Invalid Board number.

Error 1: wrong number of set parameters, parameter number must be 5
Error 2: board number should be between 0 to 4
Error 3: low/high channel must be between 0 to 7
Error 4: high channel number must be greater then low channel
Error 5: rate must be 8000/4000/2000/1000 Hz
Error 6: gain index parameter must be between 0 to 16
Error 7: vector size must be greater then 16
Error 8: sampling rate of MSP must be 16K
And out of memory error that immediately will close the Max/MSP.
IX. Verifications

One of the major problems in this project was to verify that the Digitizer~ actually works as it should work. This section describes the various aspects of the verification process.

The verification Problems

- The Digitizer~ works in interrupt level, so no conventional debugging methods like Variable Watch, Stepper, Printf, Post just don’t work.
- The Digitizer~ is a .mxe file that can’t run stand alone. It can only by run from the MAX/MSP environment. This environment does not support explicit debugging methods for external objects.
- The Digitizer~ is a real time device so verification method should be used without affecting the real time process.
- The demo board of the Insta-cal works completely different from real board and does not simulates memory transfer, commands propagation delay and interrupts, so it can’t help with the verification process.
- During most of the project developing real board wasn’t available.

Verifications made to the Digitizer~:

- User settings are applied (Board, Low Ch, High Ch, Sampling Rate, and Gain).
- The duplication method works. Tested using the “capture~” MSP object.

In the window above we can observe that every 2 samples are duplicated for the case of 8Khz sampling rate.

- The Digitizer~ outputs 16K sample per channel per second.
- Clock drift: Since the PMD has its own clock and the PC has its own clock, and these two clocks are not synchronized together, a clock drift can occur. This may result in a data which is lost (if the PMD clock is faster) or trying to outputs data that is not ready yet (if the PMD clock is slower then PC clock). We tried to monitor this drift by leaving the PMD working a long time (a few hours) and no drift had recognized. Also we managed to monitor the values of the two pointers (in and out pointers) to the circular buffer, and, again, no drift was monitored. However we believe that such drift must exist with a small rate (few ppms). So we recommend turning off and on the Digitizer~ every few hours.
Real inputs were checked using a Functions Generator device. 4 of the 14 test results we made are outlined in the following Matlab figures:

Result for: Saw signal at rate 500Hz. Digitizer sampling rate of 8Khz

We can notice the saw signal is preserved and exactly 32 samples between two peaks which confirms that the output rate is $32 \times 500 = 16000$Hz. Also we can spot that every two adjacent samples are identical.

Sin wave at 100Hz sampled at 8000Hz:

Step wave at 2000Hz sampled at 8000Hz:
Notice 8 samples to return to the same state.

Triangle wave at 1000Hz sampled at 8000Hz after dropping duplicate samples:
The ECG test:

Real ECG signal (tested on Guy Amit heart) was recorded in order to check for the expected waveform the result is outlined in the following matlab figure:
It can be easily seen that the signal has a typical ECG wave shape.

Zoom in of the signal:
Known Bugs

- When sampling with small vector size (16-128) with less than 8 channel it seems that if more than the number of sampled channel is connected to the digitizer~ outputs, some kind of ‘spike signals’ appears on the outputs of the connected outputs which are not sampled. These ‘spikes’ appear from time to time. After consulting in the MSP forum it seems that this is a MSP bug. This bug should have no affect if the user will connect the same number of inputs and outputs.
- The PMD-1608FS may have problems working with P-4 hyper-treading. The hyper-treading should be turned off using the BIOS setup. This bug is due to the MCC PMD-1608FS device capabilities, there nothing we can do about it.

X. Relevant docs
About Universal library:
- SM UL USER’S GUIDE
- SM UL FUNCTION REF

About MSP and Max:
- MSP Getting Started / Tutorials and topics / Reference
- MAX Getting Started
- Writing External objects for Max and MSP

XI. Thanks
We would like to thank:

- Mr. Guy Amit, Tel – Aviv University, for his support and ideas for testing and debugging this project.
- Cyclic 74 support team and Max/MSP community for their help in writing the Digitizer~. Special thanks for Mr. Joshua Kit Clayton for his insights regarding this project.
- Mr. Brant Davis, Measurement Computing support team.
XII. The MSP digitizer.c Code

// digitizer~.c
///////////////////////////////////////////////////////////////////////////
//includes
#include "ext.h"
#include "z_dsp.h"
#include "cbw.h"

///////////////////////////////////////////////////////////////////////////
//globals
void *digitizer_class;
int First_time=1;

///////////////////////////////////////////////////////////////////////////
//the digitizer~ struct
typedef struct _digitizer{
    t_pxobject x_obj;
    int Gain; //gain
    int BoardNum; //board number
    int UDStat; //global handler for the universal library
    int status;
    int LowChan;
    int HighChan;
    long CurCount;
    long CurIndex;
    long Count;
    long Rate;
    WORD *ADData;
    unsigned Options;
    long OutCount;
    int dup;
    long memsize;
} t_digitizer;

///////////////////////////////////////////////////////////////////////////
void *digitizer_new(double val);
void *digitizer_free(t_digitizer *x);
t_int *UL_sample_perform(t_int *w);
void digitizer_dsp(t_digitizer *x, t_signal **sp, short *count);
void show_status(t_digitizer *x);
void digitizer_sw_board(t_digitizer *x, long board);
void digitizer_sw_low(t_digitizer *x, long low);

-26-
void digitizer_sw_high(t_digitizer *x, long high);
void digitizer_sw_rate(t_digitizer *x, long high);
void digitizer_sw_gain(t_digitizer *x, long high);

void startSample(t_digitizer *x);
void stopSample(t_digitizer *x);

void handlemsgParameters(t_digitizer *x, t_symbol *msg, short argc, t_atom *argv);
//set sampling parameters
void setParameters(t_digitizer *x);
long calcMemSize(long rate, int channels);

void postErrors(int errnum);//post errors to user

>Description

void main(void)
{
    setup((t_messlist **)&digitizer_class, (method)digitizer_new,
        (method)digitizer_free, (short)sizeof(t_digitizer), 0L, A_DEFFLOAT, 0);

    addmess((method)digitizer_dsp, "dsp", A_CANT, 0);  // respond to the dsp message
    // (sent to MSP objects when audio is turned on/off)

    addmess((method)handlemsgParameters,"list",A_GIMME,0);
    addmess((method)startSample,"start",A_GIMME,0);
    addmess((method)stopSample,"stop",A_GIMME,0);

    addint((method)digitizer_sw_board);  // the method for an int in the left inlet (inlet 0)
    addinx((method)digitizer_sw_low,1);
    addinx((method)digitizer_sw_high,2);
    addinx((method)digitizer_sw_rate,3);
    addinx((method)digitizer_sw_gain,4);

    dsp_initclass();  // must call this function for MSP object classes
    post("digitizer msp object loaded...",0);
}

void *digitizer_new(double val)
{
    int i;

    t_digitizer *x = (t_digitizer *)newobject(digitizer_class);

    "Digitizer~ documentation"}
```c

dsp_setup((t_pxobject *)x,0);  // set up DSP for
the instance and create signal inlets

outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this
outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this
outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this
outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this
outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this
outlet_new((t_pxobject *)x, "signal");  // signal outlets are
created like this

intin(x,1);  // create a second int inlet
(intleftmost inlet is automatic)
intin(x,2);
intin(x,3);
intin(x,4);

x->UDStat=0;
x->BoardNum=1;
x->Count=78120;
x->memsize=78120;
x->LowChan=0;
x->HighChan=7;
x->Gain=0;
x->Rate=8000;
x->OutCount=0;
x->status=0;
x->dup=16000/x->Rate-1;

x->ADData= (WORD*)cbWinBufAlloc(x->memsize);//Memzise sample not
bytes

if (!x->ADData) /* Make sure it is a valid pointer */
{
    printf("out of memory\n");
    exit(1);
}
show_status(x);
return (x);
```
t_int *UL_sample_perform1(t_int *w)
{
    t_digitizer *x = (t_digitizer *)(w[1]); //object is first arg
    t_float *out[1];
    int n = (int)(w[3]); //length
    int dup,i;
    short status;
    long curcount,curindex;
    WORD val;
    out[0] = (t_float *)(w[2]); //output signal

    if (x->x_obj.z_disabled) goto out;
    //this for making sure the data half filled the buffer
    if (First_time==1)
    {
        cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
        if (curindex<(x->memsize)/2)
        {
            for (i=0;i<n;i++)
                *out[0]++ = 0;
        }
        else First_time=0;
    }
    else
    {
        if (x->BoardNum!=0)
        {
            while (n>0)
            {
                if (x->OutCount==x->memsize) x->OutCount=0;
                for (dup=x->dup;dup>0;dup--) //number of duplicate same sample
                    *out[0]++ = x->ADData[x->OutCount];
                *out[0]++ = x->ADData[x->OutCount++];
                n=n-(x->dup+1);
            }
        }
        else//Board is demo
        {
            x->UDStat = cbAIn (x->BoardNum, 0, x->Gain, &val);
            while (n--)
                *out[0]++ = val;
        }
    }

out:
}
t_int *UL_sample_perform2(t_int *w)
{
    t_digitizer *x = (t_digitizer *)(w[1]); //object is first arg
    t_float *out[2];
    int n = (int)(w[4]);//length
    int dup,i;
    short status;
    long curcount,curindex;

    out[0] = (t_float *)(w[2]); //output signal
    out[1] = (t_float *)(w[3]); //output signal

    if (x->x_obj.z_disabled) goto out;

    //this for making sure the data half filled the buffer
    if (First_time==1)
    {
        cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
        if (curindex<(x->memsize)/2)
        {
            for (i=0;i<n;i++)
                *out[0]++=0;
        }
        else First_time=0;
    }
    else
    {

        while (n>0)
        {
            if (x->OutCount==x->memsize) x->OutCount=0;

            for (dup=x->dup;dup>0;dup--) //number of duplicate same sample
                *out[0]++ = x->ADData[x->OutCount++];
            for (dup=x->dup;dup>0;dup--) //number of duplicate same sample
                *out[1]++ = x->ADData[x->OutCount++];

    }  
}

out:
    return (w+4);
}
n=n-(x->dup+1);
}
}
out:
return (w+5);
}

t_int *UL_sample_perform3(t_int *w)
{

t_digitizer *x = (t_digitizer *)(w[1]);
t_float *out[3];
int n = (int)(w[5]);
int nofout;
int dup,i;
short status;
long curcount,curindex;

out[0] = (t_float *)(w[2]);
out[1] = (t_float *)(w[3]);
out[2] = (t_float *)(w[4]);

if (x->x_obj.z_disabled) goto out;
//this for making sure the data half filled the buffer
if (First_time==1)
{

cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
    if (curindex<(x->memsize)/2)
    {
        for (i=0;i<n;i++)
            *out[0]++=0;
    }
    else First_time=0;
}
else
{
while (n>0)
{
    nofout=0;
    while (nofout<3)
    {
        for (dup=x->dup;dup>0;dup--)
            *out[nofout]++ = x->ADData[x->OutCount];
        *out[nofout]++ = x->ADData[x->OutCount++];
        if (x->OutCount==x->memsize) x->OutCount=0;
        nofout=nofout+1;
    }
    n=n-(x->dup+1);
t_digitizer *x = (t_digitizer *)(w[1]);
t_float *out[4];
int n = (int)(w[6]);
int dup;    //number of duplicate same sample
int nofout; //number of outs signal from digitizer
int i;
short status;
long  curcount,curindex;

out[0] = (t_float *)(w[2]);
out[1] = (t_float *)(w[3]);
out[2] = (t_float *)(w[4]);
out[3] = (t_float *)(w[5]);

if (x->x_obj.z_disabled) goto out;

//this for making sure the data half filled the buffer
if (First_time==1)
{

cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
    if (curindex<(x->memsize)/2)
    {
        for (i=0;i<n;i++)
            *out[0]++=0;
    }
    else First_time=0;
}
else
{

while (n>0)
{
    nofout=0;
    while (nofout<4)
    {
        for (dup=x->dup;dup>0;dup--)
            *out[nofout]++ = x->ADData[x->OutCount];
    }
}
}
*out[nofout]++ = x->ADData[x->OutCount++];
if (x->OutCount==x->memsize) x->OutCount=0;
nofout=nofout+1;
}
n=n-(x->dup+1);
}

out:
return (w+7);
}

t_int *UL_sample_perform5(t_int *w)
{
    t_digitizer *x = (t_digitizer *)(w[1]);
    t_float *out[5];
    int n = (int)(w[7]);
    int dup;  //number of duplicate same sample
    int nofout; //number of outs signal from digitizer
    int i;
    short status;
    long  curcount,curindex;

    out[0] = (t_float *)(w[2]);
    out[1] = (t_float *)(w[3]);
    out[2] = (t_float *)(w[4]);
    out[3] = (t_float *)(w[5]);
    out[4] = (t_float *)(w[6]);

    if (x->x_obj.z_disabled) goto out;

    //this for making sure the data half filled the buffer
    if (First_time==1)
    {
        cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
        if (curindex<(x->memsize)/2)
        {
            for (i=0;i<n;i++)
                *out[0]++=0;
        }
        else First_time=0;
    }
    else
    {
        while (n>0)
        {
            nofout=0;
        }
    }
}

Digitizer~ documentation
while (nofout<5)
{
    for (dup=x->dup;dup>0;dup--)
        *out[nofout]++ = x->ADData[x->OutCount];

    *out[nofout]++ = x->ADData[x->OutCount++];
    if (x->OutCount==x->memsize) x->OutCount=0;
    nofout=nofout+1;
}

n=n-(x->dup+1);

out:
    return (w+8);
}

t_int *UL_sample_perform6(t_int *w)
{
    t_digitizer *x = (t_digitizer *)(w[1]);
    t_float *out[6];
    int n = (int)(w[8]);
    int dup;    //number of duplicate same sample
    int nofout; //number of outs signal from digitizer
    int i;
    short status;
    long curcount,curindex;

    out[0] = (t_float *)(w[2]);
    out[1] = (t_float *)(w[3]);
    out[2] = (t_float *)(w[4]);
    out[3] = (t_float *)(w[5]);
    out[4] = (t_float *)(w[6]);
    out[5] = (t_float *)(w[7]);

    if (x->x_obj.z_disabled) goto out;
    //this for making sure the data half filled the buffer
    if (First_time==1)
    {
        cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
        if (curindex<(x->memsize)/2)
        {
            for (i=0;i<n;i++)
                *out[0]++=0;
        }
        else First_time=0;
    }

    return (w+8);
}
else
{

while (n>0)
{
    nofout=0;
    while (nofout<6)
    {
        for (dup=x->dup;dup>0;dup--)
        {
            *out[nofout]++ = x->ADData[x->OutCount];

            *out[nofout]++ = x->ADData[x->OutCount++];
            if (x->OutCount==x->memsize) x->OutCount=0;
            nofout=nofout+1;
        }
        n=n-(x->dup+1);
    }
}

out:
    return (w+9);
}

t_int *UL_sample_perform7(t_int *w)
{
    t_digitizer *x = (t_digitizer *)(w[1]);
t_float *out[7];
    int n = (int)(w[9]);
    int dup;    //number of duplicate same sample
    int nofout; //number of outs signal from digitizer
    int i;
    short status;
    long curcount,curindex;

    out[0] = (t_float *)(w[2]);
    out[1] = (t_float *)(w[3]);
    out[2] = (t_float *)(w[4]);
    out[3] = (t_float *)(w[5]);
    out[4] = (t_float *)(w[6]);
    out[5] = (t_float *)(w[7]);
    out[6] = (t_float *)(w[8]);

    if (x->x_obj.z_disabled) goto out;

    //this for making sure the data half filled the buffer
    if (First_time==1)
    {

        cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
    }

-35- Digitizer-- documentation
if (curindex<(x->memsize)/2)
    {
        for (i=0;i<n;i++)
            *out[0]++ = 0;
    }
else First_time=0;
}
else
{

while (n>0)
{
    nofout=0;
    while (nofout<7)
    {
        for (dup=x->dup;dup>0;dup--)
            *out[nofout++] = x->ADData[x->OutCount];

        *out[nofout++] = x->ADData[x->OutCount++];
        if (x->OutCount==x->memsize) x->OutCount=0;
        nofout=nofout+1;
    }
    n=n-(x->dup+1);
}
}

return (w+10);
}

Digitizer documentation
if (x->x_obj.z_disabled) goto out;

// this for making sure the data half filled the buffer
if (First_time==1)
{
    cbGetStatus(x->BoardNum,&status,&curcount,&curindex,AIFUNCTION);
    if (curindex<(x->memsize)/2)
    {
        for (i=0;i<n;i++)
            *out[0]++=0;
    }
    else First_time=0;
}
else
{

while (n>0)
{
    nofout=0;
    while (nofout<8)
    {
        for (dup=x->dup;dup>0;dup--)
            *out[nofout]++ = x->ADData[x->OutCount];
        *out[nofout]++ = x->ADData[x->OutCount++];
        if (x->OutCount==x->memsize) x->OutCount=0;
        nofout=nofout+1;
    }
    n=n-(x->dup+1);
}
out:
    return (w+11);
}

/* Here is the data collection stuff */

void digitizer_dsp(t_digitizer *x, t_signal **sp, short *count) // method called when dsp is turned on
{
    int nofCh;
}

-37- Digitizer-- documentation
nofCh=x->HighChan-x->LowChan+1;

    if (count[7])
        dsp_add(UL_sample_perform8, 10,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[3]->s_vec,sp[4]->s_vec,sp[5]->s_vec,sp[6]->s_vec,sp[7]->s_vec,sp[0]->s_n);
    else if (count[6])
        dsp_add(UL_sample_perform7, 9,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[3]->s_vec,sp[4]->s_vec,sp[5]->s_vec,sp[6]->s_vec,sp[0]->s_n);
    else if (count[5])
        dsp_add(UL_sample_perform6, 8,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[3]->s_vec,sp[4]->s_vec,sp[5]->s_vec,sp[0]->s_n);
    else if (count[4])
        dsp_add(UL_sample_perform5, 7,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[3]->s_vec,sp[4]->s_vec,sp[0]->s_n);
    else if (count[3])
        dsp_add(UL_sample_perform4, 6,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[3]->s_vec,sp[0]->s_n);
    else if (count[2])
        dsp_add(UL_sample_perform3, 5,x,sp[0]->s_vec,sp[1]->s_vec,sp[2]->s_vec,sp[0]->s_n);
    else if (count[1])
        dsp_add(UL_sample_perform2, 4,x,sp[0]->s_vec,sp[1]->s_vec,sp[0]->s_n);
    else if (count[0])
        dsp_add(UL_sample_perform1, 3,x,sp[0]->s_vec,sp[0]->s_n);
    }

/////////////////////////////////////////////////////////////////////////////
//control routines
/////////////////////////////////////////////////////////////////////////////

void digitizer_sw_low(t_digitizer *x, long low)
{
    if (low<0 || low>7 || low>x->HighChan )
    {
        postErrors(3);
        return;
    }
    x->LowChan=low;
    show_status(x);
}

/////////////////////////////////////////////////////////////////////////////
void digitizer_sw_high(t_digitizer *x, long high)
{
    if (high<0 || high>7 || high<x->LowChan )
    {
        postErrors(3);
    }
return;
}
x->HighChan=high;
show_status(x);
}

void digitizer_sw_board(t_digitizer *x, long board)
{
    if (board<0 || board>4)
    {
        postErrors(2);
        return;
    }
    x->BoardNum=board;
    show_status(x);
}

void digitizer_sw_rate(t_digitizer *x, long rate)
{
    if (rate!=0 && rate!=1 && rate!=2 && rate!=3 ){
        postErrors(5);
        return;
    }
    if (rate==0) rate=1000;
    if (rate==1) rate=2000;
    if (rate==2) rate=4000;
    if (rate==3) rate=8000;
    x->Rate=rate;
    show_status(x);
}

void digitizer_sw_gain(t_digitizer *x, long gain)
{
    if (gain<0 || gain>16){
        postErrors(6);
        return;
    }
    x->Gain=gain;
    show_status(x);
}
void show_status(t_digitizer *x)
{
    post("----------- Sampling Configuration -----------",0);
    post("Board:%d",x->BoardNum,0);
    post("Board:%d to Channel:%d",x->BoardNum,x->LowChan,x->HighChan,0);
    post("sampling Rate:%d Hz",x->Rate,0);
    switch (x->Gain){
    case 0: post("Input range: -5 to +5 V\n",0);
            break;
    case 1: post("Input range: -10 to +10 V\n",0);
            break;
    case 2: post("Input range: -2.5 to +2.5 V\n",0);
            break;
    case 3: post("Input range: -1.25 to +1.25 V\n",0);
            break;
    case 4: post("Input range: -1 to +1 V\n",0);
            break;
    case 5: post("Input range: -.625 to +.625 V\n",0);
            break;
    case 6: post("Input range: -.5 to +.5 V\n",0);
            break;
    case 7: post("Input range: -.1 to +.1 V\n",0);
            break;
    case 8: post("Input range: -.05 to +.05 V\n",0);
            break;
    case 9: post("Input range: -.01 to +.01 V\n",0);
            break;
    case 10: post("Input range: -.005 to +.005 V\n",0);
            break;
    case 11: post("Input range: -1.67 to +1.67 V\n",0);
            break;
    case 12: post("Input range: -0.25 to +0.25 V\n",0);
            break;
    case 13: post("Input range: -0.2 to +0.2 V\n",0);
            break;
    case 14: post("Input range: -2.0 to +2.0 V\n",0);
            break;
    case 15: post("Input range: -20 to +20 V\n",0);
            break;
    case 16: post("Input range: -4 to + 4 V\n",0);
            break;
    }
}

void *digitizer_free(t_digitizer *x)
{
    x->UDStat = cbStopBackground (x->BoardNum,AIFUNCTION);
    if (x->ADData)

-40-

Digitizer documentation
cbWinBufFree(x->ADData);
return NULL;
}

void handlemsgParameters(t_digitizer *x, t_symbol *msg, short argc, t_atom *argv)
{
    int i;
    long params[5];

    if (argc!=5)
    {
        postErrors(1);
        return;
    }

    params[0]=argv++->a_w.w_long;
    params[1]=argv++->a_w.w_long;
    params[2]=argv++->a_w.w_long;
    params[3]=argv++->a_w.w_long;
    params[4]=argv++->a_w.w_long;
    if (params[0]<0 || params[0]>4) {
        postErrors(2);
        return;
    }

        postErrors(3);
        return;
    }

    if (params[1]>params[2])
    {
        postErrors(4);
        return;
    }

    if (params[3]!=8000 && params[3]!=4000 && params[3]!=2000 && params[3]!=1000 )
    {
        postErrors(5);
        return;
    }

        postErrors(6);
        return;
    }

    //all tests passed set parameters
    x->BoardNum=params[0];
    x->LowChan=params[1];
    x->HighChan=params[2];
    x->Rate=params[3];
x->Gain=params[4];

    show_status(x);
}

void setParameters(t_digitizer *x)
{
    x->memsize=calcMemSize(x->Rate,x->HighChan-x->LowChan+1);
    x->ADData = (WORD*)cbWinBufAlloc(x->memsize);//Menzise sample not
    bytes
    if (!x->ADData) /* Make sure it is a valid pointer */
    {
        printf("out of memory\n");
        exit(1);
    }

cbErrHandling (PRINTALL, DONTSTOP);

    if (x->BoardNum!=0)
    x->Options = BACKGROUND + CONTINUOUS + BLOCKIO ;
    x->Count=x->memsize;
    x->dup=16000/x->Rate-1;
    x->OutCount=0;
}

void stopSample(t_digitizer *x)
{
    if (x->status==0) return;
    x->UDStat = cbStopBackground (x->BoardNum,AIFUNCTION);
    if (x->ADData)
        cbWinBufFree(x->ADData);
    post("PMD stopped");
    x->status=0;
    First_time=1;
}

void startSample(t_digitizer *x)
{
    if (x->status==1) return;

    if (sys_getsr()!=16000)
    {
        postErrors(8);
        return;
    }

    Digitizer documentation
if (sys_getblksize()<16 && x->BoardNum!=0 )
{
    postErrors(7);
    return;
}
setParameters(x);
if (x->BoardNum!=0)
    x->UDStat = cbAInScan (x->BoardNum, x->LowChan, x->HighChan, x->Count, &x->Rate, x->Gain,x->ADData, x->Options);
    post("PMD started");
    x->status=1;
}

void postErrors(int errnum)
{
    switch (errnum){
    case 1:
        post("error 1: wrong number of set parameters, parameter number must be 5");
        break;
    case 2:
        post("error 2: board number should be between 0 to 4");
        break;
    case 3:
        post("error 3: low/high channel must be between 0 to 7");
        break;
    case 4:
        post("error 4: high channel number must be greater then low channel");
        break;
    case 5:
        post("error 5: rate must be 8000/4000/2000/1000 Hz");
        break;
    case 6:
        post("error 6: gain index parameter must be between 0 to 16");
        break;
    case 7:
        post("error 7: vector size must be greater then 16");
        break;
    case 8:
        post("error 8: sampling rate of MSP must be 16K");
        break;
    }
long calcMemSize(long rate, int channels)
{
    long vecsize, ret;

    switch (rate)
    {
    case 1000:
        ret = 496 * channels * 8;
        break;
    case 2000:
        ret = 496 * 2 * channels * 8;
        break;
    case 4000:
        ret = 496 * 4 * channels * 8;
        break;
    case 8000:
        ret = 496 * 8 * channels * 8;
        break;
    }

    return (ret);
}