Getting MIDAS data into MATLAB

Convert MIDAS files into binary format

Folder *Tools/midas-to-matlab* contains a RootAna-based program for decoding MIDAS files and saving results in binary files. To build it, simply type *make*. The program requires only two parameters – an input and an output file. The syntax is:

hkAnalyzer.exe *input_file* -o*output_file*

with the '-o' being program option rather than part of filename.

Once invoked, the program should automatically determine digitizer used for taking data as well as length of each event. Supported digitizers are so far V1720, V1730 and DT5724 (all from CAEN).

The output file contains uncompressed binary data, with events organized as following structures (header fields marked with blue fonts):

Type and name	Default value	Description
uint32_t matlab_test_word	0xAAAAAAAA	Test word used to check if everything is OK during reading the data in MATLAB.
uint32_t total_waveforms	2	Total number of waveforms in this event.
int32_t total_samples		Total samples saved for each waveform.
Double_t time_step_seconds		Time step of the ADC, expressed in seconds.
Double_t conversion_factor_volts		LSB size in Volts.
uint16_t samples for 1st waveform		An array of unsigned 16-bit numbers with samples of first waveforms. Value of each sample is expressed in LSBs (least significant bits). The length of array is determined by '*total_samples*'.
uint16_t samples for 2nd waveform		An array of unsigned 16-bit numbers with samples of second waveforms. Value of each sample is expressed in LSBs (least significant bits). The length of array is determined by '*total_samples*'.
...	...	...

uint16_t samples for n-th waveform		An array of unsigned 16-bit numbers with samples of second waveforms. Value of each sample is expressed in LSBs (least significant bits). The length of array is determined by '*total_samples*'. The 'n' is determined by '*total_waveforms*'.

For convenience, there are scripts which can be called to automate decoding multiple files – their names start with '*multiAnalyzer*' and they take two input parameters – first and last run number to decode. See their source to find out where they look for data files. The scripts also compress binary files using bzip2, to save disk space.

Read data using MATLAB

If the binary files were compressed, then the first step is to create a temporary file by decompressing a file with run:

```
if isunix()
    copyfile([DATA_FOLDER '/' src_file], TEMP_FOLDER);
    command = [EXEC_BZIP2 ' -d ' TEMP_FOLDER '/' src_file];
else
    command = [EXEC_7ZIP ' e -y -o' TEMP_FOLDER ' ' DATA_FOLDER '/' src_file];
end

[status,result] = system(command);
if status ~= 0
    fprintf(1, '  Error: %s\n', result');
end
```

In the above code the '*src_file*' is the name of the file to decompress, 'DATA_FOLDER' is a folder with compressed binary files (created using the tool described above), 'TEMP_FOLDER' is a folder for temporary decompressed binary files, 'EXEC_BZIP2' is an executable name for bzip2 utility (unix/linux systems) and finally 'EXEC_7ZIP' is an executable name for 7-zip utility (Windows systems).

Once the binary file is decompressed, the only thing left to do is to open it and read the events:

```
% Open file with data
[~, tmp_filename] = fileparts(src_file);
temp_file = [TEMP_FOLDER '/' tmp_filename];
fprintf(1, 'Reading data from file: %s\n', temp_file);
FID = fopen(temp_file, 'r');

% Read the data
while ~feof(FID)
    % Read a single event; skip if there is no data
    data = read_entry(FID, show_plots);
    if isempty(data.time)
        continue;
    end

    % Do something with the data ...
end

% Close data file
fclose(FID);
```

In here, '*src_file*' is the name of the original, compressed binary file, '*tmp_filename*' is the name of the temprary (decompressed) binary file, '*FID*' if the file handle, 'data' is a structure containing data from a single event and '*read_entry*' is a function for reading events from binary files.

The '*data*' structure has three fields, all vectors (equal lengths):

- *time* – time of each sample, expressed in seconds
- *reference* – samples of reference waveforms, expressed in volts
- *shaper* – samples of shaper waveforms, expressed in volts.

The syntax for calling '*read_entry*' function is of the following form:

```
[data, event_size] = read_entry( FID, show_plots )
```

where '*FID*' is the file handle, '*show_plots*' turns on/off display of the read waveforms, '*data*' is the output data structure and '*event_size*' returns number of read bytes.

After finalizing all the processing, remember to remove temporary file:

```
% Remove temporary file
fprintf(1, 'Removing temporary file\n');
delete([TEMP_FOLDER '/*.dat']);
if status ~= 0
    fprintf(1, '  Error: %s\n', result');
end
```

And that is all :)