USB MPEG Capture Device

Software Manual

Model 2250 / 2251 | Rev.A | September 10

SENSORAY | embedded electronics

Designed and manufactured in the U.S.A.

SENSORAY | p. 503.684.8005 | email: info@SENSORAY.com | www.SENSORAY.com

7313 SW Tech Center Drive | Portland, OR 97203

Table of Contents

Limited warranty

Sensoray Company, Incorporated (Sensoray) warrants the hardware to be free from defects in material and workmanship and perform to applicable published Sensoray specifications for two years from the date of shipment to purchaser. Sensoray will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. Sensoray shall have the right of final determination as to the existence and cause of defect.

As for items repaired or replaced under warranty, the warranty shall continue in effect for the remainder of the original warranty period, or for ninety days following date of shipment by Sensoray of the repaired or replaced part, whichever period is longer.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. Sensoray will pay the shipping costs of returning to the owner parts that are covered by warranty. A restocking charge of 25% of the product purchase price, or $105, whichever is less, will be charged for returning a product to stock.

Sensoray believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Sensoray reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult Sensoray if errors are suspected. In no event shall Sensoray be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SENSORAY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SENSORAY SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SENSORAY WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEROF.

Software

Feature Summary

Models 2250 & 2251 are shipped with the drivers for Microsoft Windows 2000/XP/Vista. A full-featured demo application allows preview and saving of MPEG video on the host computer, control of the compression and video settings, text overlays, and snapshot capture.

An included SDK can be used to integrate video capture into any application. The SDK allows maximum flexibility by providing an API for all the 2250/51's functions. The source code of the demo application is a suggested starting point for custom application development.

Installation

**The model 2251 is software compatible to the model 2250. The drivers, installation, and demo programs may identify the 2251 as a 2250. There is no separate software package for the model 2251; the standard 2250 SDK should be used.*

The software may be distributed on a CD or downloaded from the Sensoray's web site. If the file is downloaded, it will need to be unzipped into a folder on the local drive prior to connecting the 2250/51 to the USB port.

Run the setup program (2250setup.exe) from the distribution disk or folder. Software components, including a demo application with the source code, will be installed into the /Program Files/Sensoray/2250 folder.

During the installation the program will install ffdshow. During the setup accept the defaults. (If MPEG1 and 2 are not selected, the setup program will automatically enable them in the registry.)

After the 2250/51 is connected to the USB port for the first time, a Windows "Found new hardware…" dialog may appear. Select "No, not this time" and click "next". Choose "Install the software automatically". The driver is preinstalled with the setup program but may pop up a Windows Logo warning. Click "Continue Anyway". The driver will install without prompting for file locations. A second "Found new hardware…" dialog may appear shortly afterwards. Please repeat the above procedure.

If using Windows 2000, the setup program will check for DirectX9 and launch a browser on the Microsoft website if not present. DirectX9 is required on Windows 2000, but not XP nor Vista. Please download the latest DirectX9 runtime from the Microsoft website if setup requests it.

Software targets

The software is provided as is. Only one copy of the software is allowed and the software is tied to one Sensoray device only. It must not be copied or redistributed by end users.

The software contains targets in the API directory. The following files are required as part of the software. Only one installation per Sensoray hardware is permitted. This software must not be used except with Sensoray hardware.

- API\mid2250.dll (installed in application exe directory. This DLL is built with the cdecl calling convention. Please see General Notes below if distributing with a VB6 program or other program expecting stdcall.)
- API\filters\ffmp2encoder.ax (needs registered with regsvr32)
- API\filters\SRAYBridge.ax (needs registered with regsvr32)
- API\filters\DSKernel2dll (needs registered with regsvr32)
- API\filters\LMMpg1Mx2.dll(needs registered with regsvr32)
- API\filters\LMMpg2Mx2.dll(needs registered with regsvr32)
- API\filters\sensmpgds.ax (needs registered with regsvr32)
- API\filters\wavdest.ax (needs registered with regsvr32)
- API\windir\s2250param.ini (install to windows directory)

Additionally, ffdshow-20051221.exe should be installed if preview is desired.

The drivers must also be be installed for proper function. They are included in the drivers directory.

There are two drivers for the 2250/51. One driver, with the files 2250WisLoader.inf and 2250WisLoader.sys, loads the firmware to the USB chip. The other files are associated with the MPEG driver (2250strm.inf). Both drivers must be installed on the end-users computer. A hardware first install will detect the 2250WisLoader device first and then prompt to install the 2250strm driver.

SDK Reference

Release Notes

V1.19

- Windows 7 support

The common API flow is described below. Refer to the complete functions reference for the details on individual functions.

General Notes

Important: The DLL is built with "cdecl". If using the DLL in a programming environment expecting "stdcall", please use the DLL built with stdcall. The stdcall DLL is called mid2250_vb.dll, but it is not exclusive to Visual Basic 6. This file may be renamed mid2250_stdcall.dll in the future. Please note that newer versions of VB may use the cdecl calling convention.

A short example for including the DLL in a C# application is shown in Appendix A.

The common API flow is described below. Refer to the complete functions reference for the details on individual functions.

1. Initialization. This is performed by a call to `SN_Open()` function. Initial default capture settings are loaded.
2. A call to `SN_Open()` may be optionally followed by calls to the functions controlling various settings:
    - input source: `SN_SetSource();`
    - video system and geometry: `SN_SetVidSys(), SN_SetVidSize();`
    - video parameters (brightness, contrast, saturation, hue): `SN_SetLevels();`
    - compression configuration: `SN_SetEncodeType(), SN_SetBitrate();`
    - audio input configuration: `SN_SetAudioInput(), SN_SetMute();`
    - record mode: `SN_SetRecordMode();`
    - OSD: `SN_SetOverlayText()`.
3. A call to `SN_StartStream()` starts the 2250. The stream received from the USB is decoded and displayed in the user window specified with `SN_Open()`.

4. If recording is required, it is initiated by a call to `SN_StartRecord()`, and stopped with a call to `SN_StopRecord()`.

5. During the recording the following functions could be used to obtain some useful information:

    - `SN_GetStatus()` – current status (record, playback), current recorded file size and path;

    - `SN_GetTimeLeft()` - returns estimated recording time left on disk;

    - `SN_IsEnoughSpace()` - returns true if enough space to record at specified bitrate for length of time specified (estimate).

6. `SN_Close()` must be called before application terminates to clean up properly.

Demo applications

The SDK includes several applications provided with the source code to illustrate the use of SDK's functions.

DemoApp – an MFC application. Displays the stream in the window, allows modification of compression, video, and audio parameters, recording the stream to the hard drive.

DemoCallback – A simple console application showing the use of the 2250 SDK in callback mode. This can be used for Motion JPEG capture or MPEG capture. The predefined Preview, record, and some other functionality (Snapshot to file) are not supported in this mode. This mode was defined for possibly embedded applications, which just want to retrieve data. The callback should return quickly and MUST not block. See the functions SN_StartCBCaptureOnly, SN_StopCBCaptureOnly, SN_RegisterAudCB, Sn_RegisterVidCB in mid2250func.h and the source code include in the SDK.

DemoMotionJpeg – a simple console application. Records a given number of JPEG frames to the hard drive without display. Does not require installation of ffdshow. Demonstrates simple MotionJpeg capture only. An alternative using callbacks is shown in DemoCallback

DemoConsoleApp – console application using the 2250 SDK

DemoMpegCap – demonstrate MPEG only mode. Another way to get MPEG only or MotionJpeg only is using callbacks (see DemoCallback)

DemoVB6 – demonstrates the basics of using the SDK in Visual Basic 6

DemoVBNet – demonstrates the basics of using the SDK in Visual Studio .NET 2003(or higher)

Functions Reference

All API functions are declared using the following definition:

#define MID2250_API extern "C" __declspec(dllimport)

```
MID2250_API int SN_Open( void );
```

Must be called before any other API functions are called to open the SDK.

Parameters

None.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_Close(
);
```

Must be called before application terminates for proper clean-up of the SDK and SDK objects.

Parameters

None.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetBoardWindow(
    HWND hwnd,
    BOOL bResize,
    int board
);
```

Sets the window for display or rendering the video stream. If called, will automatically set SN_SetRenderVideo(TRUE).

hwnd

A handle to the window to display video in.

bResize

If set to TRUE, the window will be resized to match native video size.

board

board number in the system (use 0 for single board setups).

```
MID2250_API int SN_SetPlaybackWindow(
    HWND hwnd,
    BOOL bResize
);
```

Sets the window for playback.

hwnd

A handle to the window to display video in.

bResize

If set to TRUE, the window will be resized to match native video size.

```
MID2250_API int SN_SetSource(
   MID2250_SOURCE input,
   int board
);
```

Selects between composite and S-video inputs.

Parameters

input

enumerated input MID2250_SOURCE (see mid2250types.h). For model 2250 the allowed values are MID2250_SOURCE_COMPOSITE_0 and MID2250_SOURCE_SVIDEO_0.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetSource(
   MID2250_SOURCE *pSource,
   int board
```

```
);
```

Retrieves current input settings.

Parameters

pSource

pointer to the value of current input.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetVidSys(
   MID2250_VIDSYS vidsys,
   int board
);
```

Sets the input video system (NTSC, PAL).

Parameters

vidsys

video system enumerated type (see mid2250types.h).

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetEncodeType(
   MID2250_ENCODING encodeType,
   int board
);
```

Sets the desired encoding type (MPEG1,2,4, MJPEG).

Parameters

encodeType

encoding enumerated type (see mid2250types.h).

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetEncodeType(
   int board
);
```

Retrieves current encoding type.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

Current encoding setting, -1 if error.

```
MID2250_API int SN_SetBitrate(
   int bitrate,
   int board
);
```

Set the desired bitrate of the output stream.

Parameters

bitrate

desired bitrate (in bits per second). Recommended values are 1,000,000 to 6,000,000 for D1 resolutions, 300,000 to 2,000,000 for CIF resolutions.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetBitrate(
   int board
```

```
);
```

Retrieves current bitrate setting.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

current bitrate (in bits per second), -1 if error.

```
MID2250_API int SN_GetStatus(
   MID2250STATUS *pStatus,
   int board
);
```

Retrieves current status information (see mid2250func.h for `MID2250STATUS` type definition).

Parameters

pStatus

pointer to status variable.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_StartStream(
   int board
);
```

Starts video and audio streaming.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_StopStream(
   int board
);
```

Stops video and audio streaming. If recording is enabled it is stopped as well.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetRecordMode(
   MID2250_REC recMode,
   int board
);
```

Sets the record mode (see mid2250types.h for `MID2250_REC` type definition).

Parameters

recMode

record mode.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_StartRecord(
   char *fileName,
   int size,
   int breakonrecsize,
   int board
);
```

Starts recording to a file. (Note: The "breakonrecsize" parameter and feature will be obsoleted in the future. It is recommended that SDK users do their own clip management in order to directly control the clip names). Please also see the release notes for V115G.

Parameters

fileName

full path to the target file, no extension.

size

maximum file size in megabytes. If the maximum file size is reached, a new file is started with an incremental number appended to the file name (depending on the *breakonrecsize* value).

breakonrecsize

If set to 0, recording continues to the new file. Otherwise recording is stopped when the file size reaches the limit.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PauseRecord(
   int board
);
```

Pauses the recording. Available only for Video elementary stream. Not available for Audio/Video multiplexed streams.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_ResumeRecord(
   int board
);
```

Resumes the recording. Available only for Video elementary stream. Not available for Audio/Video multiplexed streams.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_StopRecord(
   int board
);
```

Stops the recording. Preview will continue.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetTimeLeft(
   char *szDrive,
);
```

Estimates recording time left based on free disk space.

Parameters

szDrive

current drive, e.g. "C:\".

Returns

Estimated recording time left, seconds, or -1 in case of an error.

```
MID2250_API int SN_SnapshotToFile(
   char *fileName,
   int filetype,
   int qual,
   int board
```

```
);
```

Takes a snapshot and saves it to a file.

Parameters

fileName

full path to the target file, no extension.

filetype

file type(s) to save to (`MID2250_FILE_JPEG`, `MID2250_FILE_BMP`, see mid2250types.h).

qual

JPEG quality (25-100, higher value yields better quality and bigger files).

board

board number in the system (use 0 for single board setups).

Returns

Image size in bytes or a negative value in case of an error.

```
MID2250_API int SN_SetPlayback(
   char *filename,
   int rawtype
   int vidnum
);
```

Starts playback of the specified file in current window.

Parameters

filename

full path to the target file.

raw_type

unused. set as RAWTYPE_RGB24.

vidnum

Use –1 for playing back to hWnd. Other values are reserved.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PlaybackVideo(
   int vidnum
);
```

Starts playback of the specified file in current window.

Parameters

vidnum

number of assigned video stream.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_StopPlayback(
   int vidnum
);
```

Stops video playback.

Parameters

vidnum

number of assigned video stream.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PausePlayback(
   BOOL bPause,
   int vidnum,
);
```

Pauses/resumes playback.

Parameters

bPause

`TRUE` for pause, `FALSE` for resume.

vidnum

number of assigned video stream.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PlaybackSetRate(
   double drate,
   int vidnum,
);
```

Changes playback speed.

Parameters

drate

a double specifying playback speed. 0.5 corresponds to half of normal speed, 2.0 - double the normal speed. Minimum speed is 0.5.

vidnum

number of assigned video stream.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PlaybackSetSeekPosition(
   int percent,
   int range,
   int vidnum
);
```

Seeks the position in the file being played back. The position is calculated as file_size * percent / range.

Parameters

percent

a numerator of the position in the file expressed as a fraction of a total file size.

range

a denominator of the position in the file expressed as a fraction of a total file size.

vidnum

number of assigned video stream.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_PlaybackGetSeekPosition(
   int range,
   int vidnum
);
```

Retrieves the current position in the file being played back. The position is calculated as file_size * percent / range.

Parameters

range

a denominator of the position in the file expressed as a fraction of a total file size.

vidnum

number of assigned video stream.

Returns

a numerator of the position in the file expressed as a fraction of a total file size, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetLevels(
   int param,
   char value,
    int board
);
```

Sets brightness, contrast, saturation and hue of the captured video.

Parameters

param

defines the parameter to set (`MID2250_LEVEL_CONTRAST`, `MID2250_LEVEL_BRIGHTNESS`, `MID2250_LEVEL_SATURATION`, `MID2250_LEVEL_HUE`). See see mid2250types.h for definitions.

value

defines the value of selected parameter (-50 to +50, default 0 for hue; 0 to 100, default 50 for the rest).

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetAudioInputs(
   MID2250_AUDIO_INPUT audio,
   int micBoost,
    int board
);
```

Selects an audio input (default is line in).

Parameters

audio

one of the following: `MID2250_AUDIO_LINE`, `MID2250_AUDIO_MIC`. See mid2250types.h for definitions.

micBoost

0 – normal, 1 – 20 dB boost.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetMute(
   BOOL bMute,
```

```
    int board
);
```

Mutes audio on the host computer. Audio will still be recorded. If called while streaming, the function will stop and restart the sream.

Parameters

bMute

TRUE to mute the audio.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetMute(
   BOOL *bMute,
    int board
);
```

Retrieves current muting setting.

Parameters

bMute

pointer to retrieved setting.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetOverlayText(
   int idx,
   const POINT *pPos,
   const overlay_text_t *pOvlText,
```

```
   BOOL bEnable,
   BOOL bUpdate,
   int board
);
```

Configures text overlays (OSD).

Parameters

idx

Overlay number (index), 0-5.

pPos

position of top left corner of overlay window (in pixels). Position is rounded to the nearest 16x16 pixel block.

pOvlText

text to display (a maximum of 96 characters total for all overlay windows). See mid2250types.h for the definition of `overlay_text_t`.

bEnable

`TRUE` enables current overlay region.

bUpdate

if setting multiple regions has to be `FALSE` for all regions except the last.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, -1 on failure, -2 – out of text space.

```
MID2250_API int SN_ClearOverlay(
    int board
);
```

Clears all overlays.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetVidSize(
   int iVidSize,
   int board
);
```

Sets the captured video size (resolution).

Parameters

iVidSize

video size: 0 – full resolution, default (720x480 NTSC, 720x576 PAL), 1 – CIF (360x240 NTSC, 360x288 PAL).

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetRenderVideo(
   BOOL bDisplayVideo,
    int board
);
```

Turns the preview on and off. Recording will continue regardless of this setting.

Parameters

bDisplayVideo

`TRUE` to enable preview.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetRenderVideo(
```

```
   BOOL *bDisplayVideo,
   int board
);
```

Retrieves the preview setting.

Parameters

bDisplayVideo

pointer to retrieved setting.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetAspectRatio(
   MID2250_ASPECT_MODE mode,
   int board
);
```

Allows modifying the video aspect ratio for preview.

Parameters

mode

MID2250_ASPECT_NONE allows stretched preview image, MID2250_ASPECT_CONST maintains original aspect ratio (see mid2250types.h).

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_GetAspectRatio(
   MID2250_ASPECT_MODE *mode,
   int board
);
```

Retrieves the aspect ratio control setting.

Parameters

mode

pointer to retrieved setting.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_Repaint(
   HDC hdc
);
```

Repaint callback, should be called when application receives a WM_PAINT event. This is necessary to notify the video renderer of a repaint event.

Parameters

hdc

device context handle or `NULL` if unknown.

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_DisplayChange(
);
```

Informs the video renderer that a WM_DISPLAYCHANGE message has been received by the application. This callback should be called when the application has a WM_DISPLAYCHANGE event.

Parameters

none

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_UpdateClippingWindow(
   HWND hwnd
```

```
);
```

Removed with V.1.03. Please use SN_SetBoardWindow.

```
MID2250_API int SN_SetRemoveMsg(
   HWND hwnd,
   int removeMsg,
   int board
);
```

Sets the value of the message that is sent to the application's window in case of device removal (e.g. USB cable unplugged).

Parameters

hwnd

handle of the window that will receive the message.

removeMsg

message value to be sent.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_TestDeviceRemoval(
    int board
);
```

Tests whether device was removed or not. When `SN_SetRemoveMessage` is called, a message window handle and remove message is set. When remove message is received by the application, this function is called to confirm the device was removed and if it was, the DLL should be shut down.

Parameters

board

board number in the system (use 0 for single board setups).

Returns

0 in case the device was not removed, 1 in case it was.

```
MID2250_API int SN_SetVideoPosition(
   int xpos,
   int ypos,
   int xsize,
   int ysize,
   int left_clip,
   int top_clip,
   int right_clip,
   int bottom_clip,
   int board
);
```

Sets the video position in the clipping window.

Parameters

xpos, ypos

x and y coordinates of the top left corner.

xsize, ysize

horizontal and vertical sizes of video display.

left_clip, top_clip, right_clip, bottom_clip

number of pixels to clip of the left, top, right and bottom sides of source video.

board

board number in the system (use 0 for single board setups).

Returns

0 on success, negative value if error (see mid2250types.h for error codes list).

```
MID2250_API int SN_SetOpMode(
   MID2250_OPMODE mode,
   int board
);
```

Sets the operation mode to normal, or MJPEG capture. In normal mode the stream from the 2250 is rendered on the display and optionally recorded. MJPEG capture mode is for plain capture of

JPEG frames, no rendering is performed. No DirectShow filters are required to be installed for MJPEG mode to function.

Parameters

mode

`MID2250_OPMODE_NORMAL` – normal operation mode,

`MID2250_OPMODE_MJPEGCAPTURE` – MJPEG capture mode.

board

board number in the system (use 0 for single board setups).

Returns

0 if success, negative if error.

Example

The following code illustrates simple capture of JPEG compressed frames in MJPEG mode.

```
int main()
{
    int i;
    char filename[260];
    SN_Open( NULL);
    SN_SetOpMode(MID2250_OPMODE_MJPEG_CAPTURE);
    SN_SetEncodeType( MID2250_ENCODE_MOTIONJPEG);
    SN_SetBitrate( 6000000);    // Controls JPG picture quality
    SN_StartStream( );
    for( i = 0; i< 50; i++)
    {
        sprintf(filename, "c:\\snapshot%03d", i);
        SN_SnapshotToFile(filename, MID2250_FILE_JPEG);
    }
    SN_StopStream();
    SN_Close();
    return 0;
}
```

```
MID2250_API int SN_GetNumBoards(
   int *pNumBoards
);
```

Retrieve the number of boards in the system.

Parameters

pNumBoards

Returns the number of boards.

Returns

0 if success, negative if error.

```
MID2250_API int SN_SetMpegSequence(
   MID2250_SEQMODE seqMode,
   int board
);
```

Set alternate MPEG sequence mode. For MPEG 1,2,4 only.

Parameters

seqMode

MID2250_SEQMODE_IFRAMESONLY – only use I-frames;

MID2250_SEQMODE_DEFAULT – default sequence mode (IP, or IPB mode).

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_GetFrame(
   long inSize,
   unsigned char *pFrame,
   BITMAPINFOHEADER *pBMI,
   int type,
   int board
);
```

SN_GetFrame retrieves current frame. Type Currently supported for MPEG 1,2,4 only. If running in MOTIONJPEG mode, type is ignored and full JPG image is returned with size pBMI->biSizeImage (The returned MotionJPG buffer can be saved directly to a JPG file). Please see GetFrame.cpp in demo app for an example of using SN_GetFrame.

Parameters

inSize

size of pFrame buffer supplied (should be at least 768*576*3)

pFrame

pointer to buffer of size inSize where frame is stored

pBMI

pointer to bitmap info. Currently only the following parameters are updated: pBMI->biHeight, biWidth, biSizeImage.

type

type of buffer to retrieve.

type = 0 for YcbCr buffer formatted as follows. Y plane first 768*576 bytes. Cb plane next 768*576/4 bytes. Cr plane next 768*576/4 bytes.

type = 1 for RGB8 buffer.

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_SetMD(
   MID2250_MDConfig_t *pMD,
   HANDLE event,
   int board
);
```

Sets or updates the motion detection region according to MID2250_MDConfig_t (see mid2250types.h file for description). The main demo app shows an example of using motion detection.

Parameters

**pMD*

pointer to MD structure

event

handle to event to notify.

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_StopMD(
   int board
);
```

Stops motion detection.

Parameters

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_DisplayMDRegions(
   BOOL bOnOff,
   int board
);
```

Displays motion detection regions on displayed image (if rendering the video to screen). Requires Windows XP or DirectX 9 (Windows 2000).

Parameters

bOnOff

display on or off

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_GetMDData(
   int *region,
   unsigned char *buf,
   int bufsize,
   int board
);
```

Retrieves the motion detection data. See demo application for example.

Parameters

region

the returned region mask. 0x02: region 1, 0x04: region 2, 0x08 region 3. The region mask should be used to detect if motion detected or not.

buf

a pointer to motion macroblock map (must be size 208).

bufsize

size of bufmap. MUST use 208.

Board

board number in system.

The primary detection is by region. If further information is wanted, the MB map will show if motion passed threshold in a given region on a macroblock by macroblock level (16 pixels x 16 pixels) on a bit level. For example, (buf[0] & 0x01) represents macroblock 1, (buf[0] & 0x02) represents macroblock 2, (buf[1] & 0x01) represents macroblock 8.

Returns

0 if success, negative if error.

```
MID2250_API int SN_SetTimestampOverlay(
   int idx,
   const POINT *pPos,
   const overlay_timestamp_t *pTimestamp,
   int board
);
```

Starts automated timestamp overlay.

Parameters

idx

overlay index position to use

pPos

pointer to position of overlay

pTimestamp

pointer to timestamp structure

```
typedef struct {
char labelText[MAX_LABEL_SIZE]; // label to prepend to text
int starttime; // start time in seconds
int displayMS; // display milliseconds
int fontidx; // font index to use
        //allowed fontidx values (0-white, 1-green, 2-white double size)
} overlay_timestamp_t;
```

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_ClearTimestampOverlay(
   int board
);
```

Clears the automated timestamp overlay.

Parameters

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_SetEmbeddedData(
   int insize,
   const void *data,
   int board
);
```

Adds user data to the MPEG stream(MPEG1,2,4 only). Mpeg data is added as a standard 0x000001b2 structure in each picture header. The maximum size of the embedded data is 1024 bytes, although smaller sizes are preferred for lower bitrates. User data may NOT contain MPEG start codes, which means there may not be more than 23 consecutive zero bits in the data. This

function will check for start codes. It is recommended, therefore to separate the data. For instance, if you want to store a 32 bit (4 byte) latitude position in the stream, you could put the high order bytes in the first 2 bytes of user data, followed by a "marker byte" of 0xff, and put the other 2 bytes in the next 2 user data bytes.

Eg. For a latitude with hex value 0x70000111, the user data would be

data[0] = 0x70, data[1] = 0x00, data[2]=0xff(marker), data[3]=0x11, data[4]=0x11

Similarly, if other user data needs inserted, simply add a market byte to separate the data into sections.

Parameters

int insize

input size of the embedded data.

*const void *data*

pointer to user data

board

Board number in system.

Returns

0 if success, negative if error.

```
MID2250_API int SN_GetPlaybackFrame(
   long inSize,
   unsigned char *pFrame,
   BITMAPINFOHEADER *pBMI,
   int type,
   int block,
   int timeout,
   int vidnum
);
```

SN_GetPlaybackFrame retrieves current frame being decoded after SN_PlaybackVideo. It can retrieve Uncompressed frames and MPEG data from the playback. The pBMI returned contains the image parameter. It is up to the user to determine a reasonable image size for pFrame beforehand. For example, if the user needs RGB24, they should allocate at least 720x480x3 for NTSC and 768x576*3 for PAL. For compressed data, the video output comes out of the demux in very small packets. Therefor, this function does not work the same as SN_GetFrame where the images are packetized by MPEG picture headers, but returns all of the MPEG data received since the last call.

Parameters

inSize

size of pFrame buffer supplied (use 768*576*3 if unsure)

pFrame

pointer to buffer of size inSize where frame is stored

type

0, uncompressed(colorspace determined beforehand with SN_SetPlayback), 1 compressed

pBMI

pointer to bitmap info. For compressed images, the length of data is stored as biSizeImage.

block

0 – non blocking, 1 blocking with timeout

timeout

blocking timeout in milliseconds

vidnum

assigned number of the playback stream

Returns

0 if success, negative if error.

Motion Detection

Motion Detection on the 2250/51 (and 314) involves regions of interest and motion thresholds. The chip firmware has 3 user programmable regions of interest. The SDK shows 4 regions of interest in the structure MID2250_MDConfig_t. The 4th region, corresponding to index 3(starts from zero) is reserved for possible future use.

The regions are described in pixels. The user specifies an upper left co-ordinate and a lower right pixel coordinate for each region.

There is a requirement that the regions do not overlap on a macro-block(MB) level. They should be spaced 16 pixels apart as one macro-block corresponds to a 16x16 pixel block. The MDSettingDlg.cpp file in the demo app demonstrates verification of the MD regions.

If a region is unused, the coordinates {0,0} should be used for both the upper left corner and bottom right corner.

In each region, motion is detected on the basis of threshold values. The first threshold is the SAD (sum of absolute differences), which represents changes in intensity. The recommended range for the SAD is 20-150. In general SAD will catch fast motion across the target. For testing purposes, start at SAD=40 (with sensitivity = 80).

The second threshold is the motion vector threshold. The recommended range is 0-1600. For testing purposes(with sensitivity = 80), start at MV=700. The MV threshold will detect slow motion across the target better than SAD.

Finally, there is a sensitivity value. It may be set from 0-100. A recommended start setting is 80. In general, higher thresholds may be used for higher sensitivity settings.

Mathematically, motion detection in the chip is calculated as follows:

H/w comes up with a set of MV for each MB in each direction, MV = ((Vx * Vx) + Vy*Vy)) >> 2).

If ((Vx*Vx+Vy*Vy)>>2)>MV_THRESHOLD), then this MB is motion detected.

If (Sad>SAD_THRESHOLD), then this MB is also motion detected.

If the detected MB is in a specified region, a motion map is produced and a notification sent to the user.

The motion detection configuration structure is as follows (mid2250types.h) (The last region is reserved):

```
typedef struct
{
    MID2250_PointCorrds_t ULPoint[ 4 ];
    MID2250_PointCorrds_t BRPoint[ 4 ];
    unsigned short u32SADThresholdValues[ 4 ];
    unsigned short u32MVThresholdValues[ 4 ];
    unsigned short u32SensitivityValues[ 4 ];
} MID2250_MDConfig_t;
```

Motion Map

When motion is detected, an event is generated with data containing the region(s) of interest with motion and a motion map. The motion map contains all the MBs which were hit inside a given region. Note that some MBs will be shown outside the region (please see Example 2 in the motion examples below). If the user sets one region to cover the whole image, then the Motion Map will represent motion hits across the whole image. This allows the user to implement a custom MD method on a macro-block by macro-block level.

A sample motion map where the region covers the full screen is shown below. The blank lines on the bottom are not part of the image because the image was NTSC(480) not PAL(576). The demo map shows macro-blocks up to number 36(576/16) in the vertical direction. The motion map (and hit statistics) can be viewed by selecting View->MDStats. Select View->MDStats again to hide the statistics and motion map.

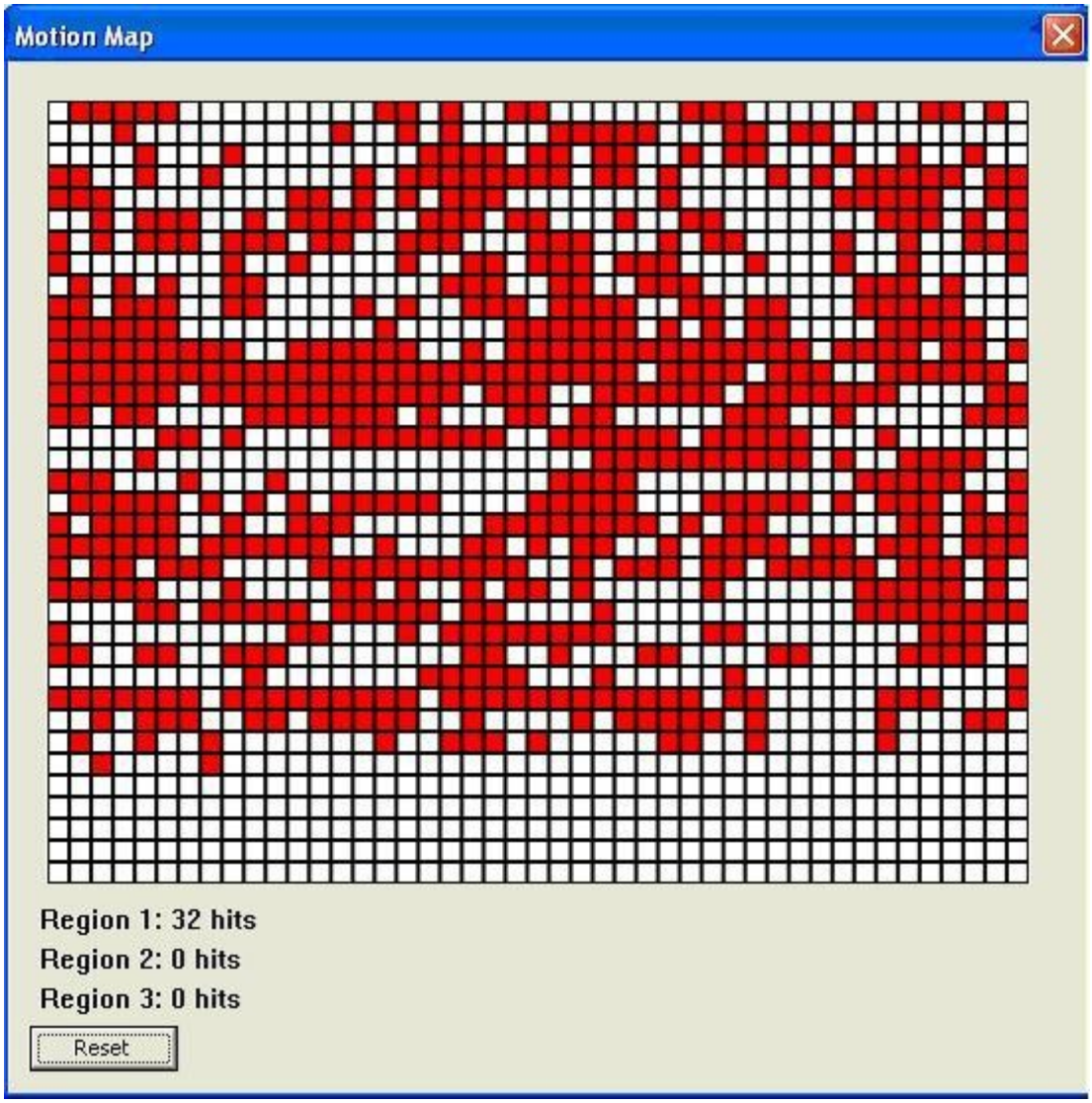

Figure 1 Motion Map(statistics represent number of motion events per region).

The motion-map above shows a hit at macro-blocks 2,3,4,5 and other macro-blocks where the squares are red. The white color MB #1 does not have a hit in this motion map. Macroblock 2 = pixels (16,0) to (31,16). The 32 hits corresponds to 32 motion events received by region 1. It does **not** represent the total number of red macroblocks(motion macroblocks) in the motion map.

Motion Detection Quickstart

The demo application shows an example of motion detection. A step by step example is shown below:

- Select Tools->MDSetup.
- Select a region or regions. Click on Enable Region1 Setting.

- Enter 720 for BR_x and 320 for BR_y.
- Click OK.
- Start stream (if not already started).
- Select View->MDRegions. Region selected above will be shown on the image.
- Select View->MDStats. A macro-block map with detailed hits for region 1 above will be shown along with a counter of the number of hits in region 1.
- To disable motion detection, call SN_StopMD or in the demo app, disable all MD regions.

Motion Map Example

Example 1: Three regions of interest

The following motion detection regions were defined. An overlay of the defined regions can be seen by selecting View->MDRegions on the menu bar.

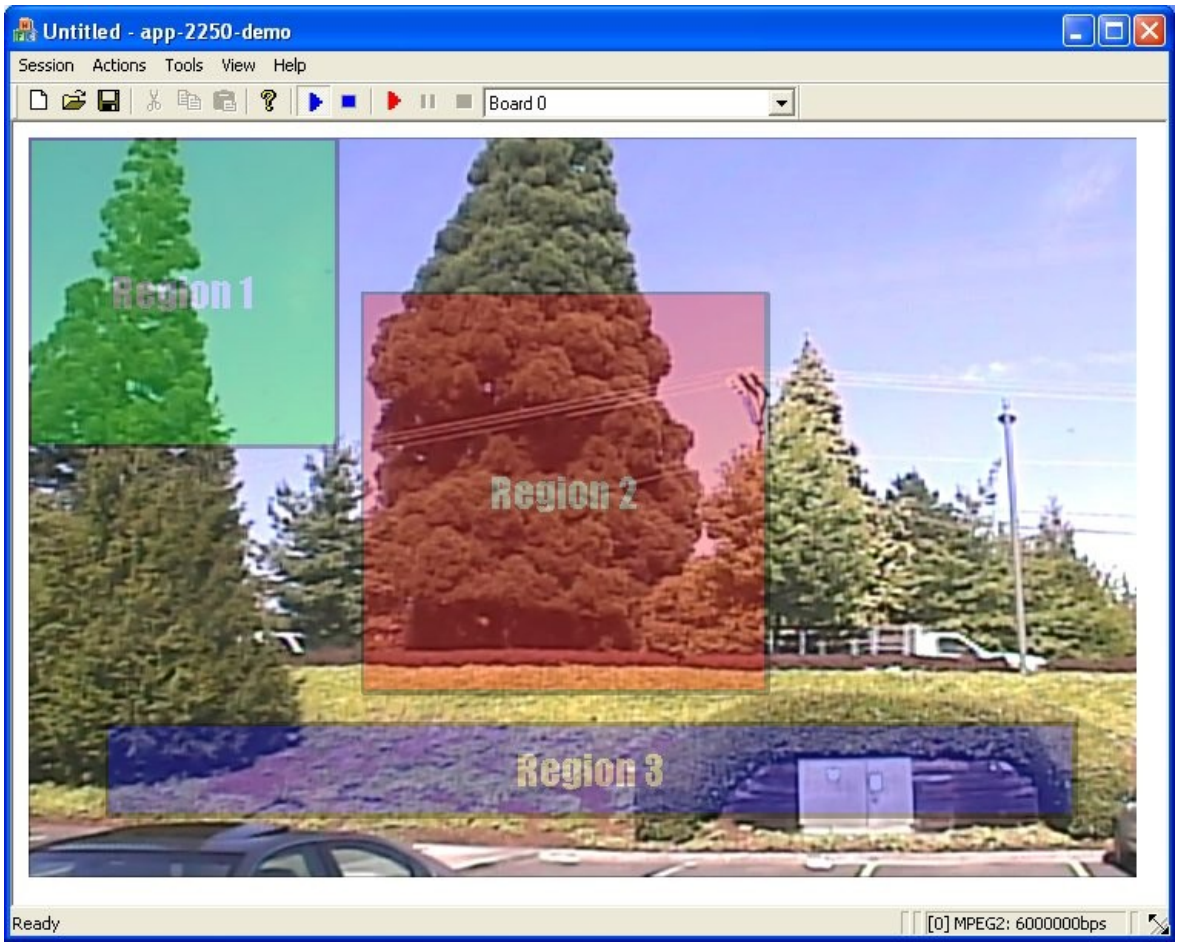

Figure 2 : Three(3) Motion Detection Regions of Interest

The above regions were set-up by selecting Tools->MDSetup. A screen shot of the setup for the regions above is show in figure 3.

Figure 3: Setup for regions shown in Figure 2

Example 2 : Advanced Motion Map

The motion map may show macroblocks outside the range of interest. This example demonstrates how the motion map works. In short, only those macroblocks inside the defined areas of interest will show motion. Some macroblocks outside the region of interest may show motion. These should be ignored.

In this example, a small motion detec tion region in the upper left corner is produced as shown in Figure 4.

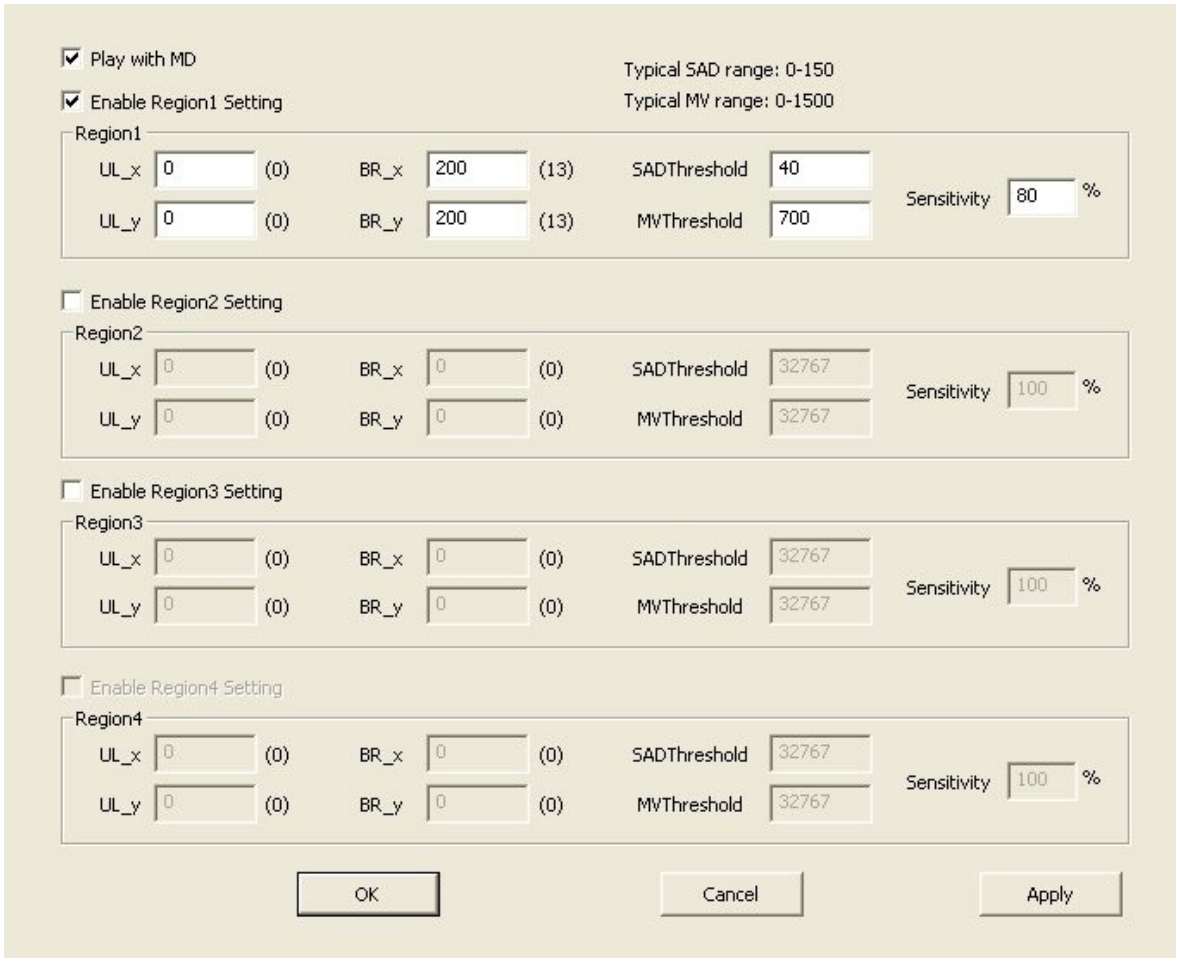

Figure 4

This corresponds to the region show in Figure 5.

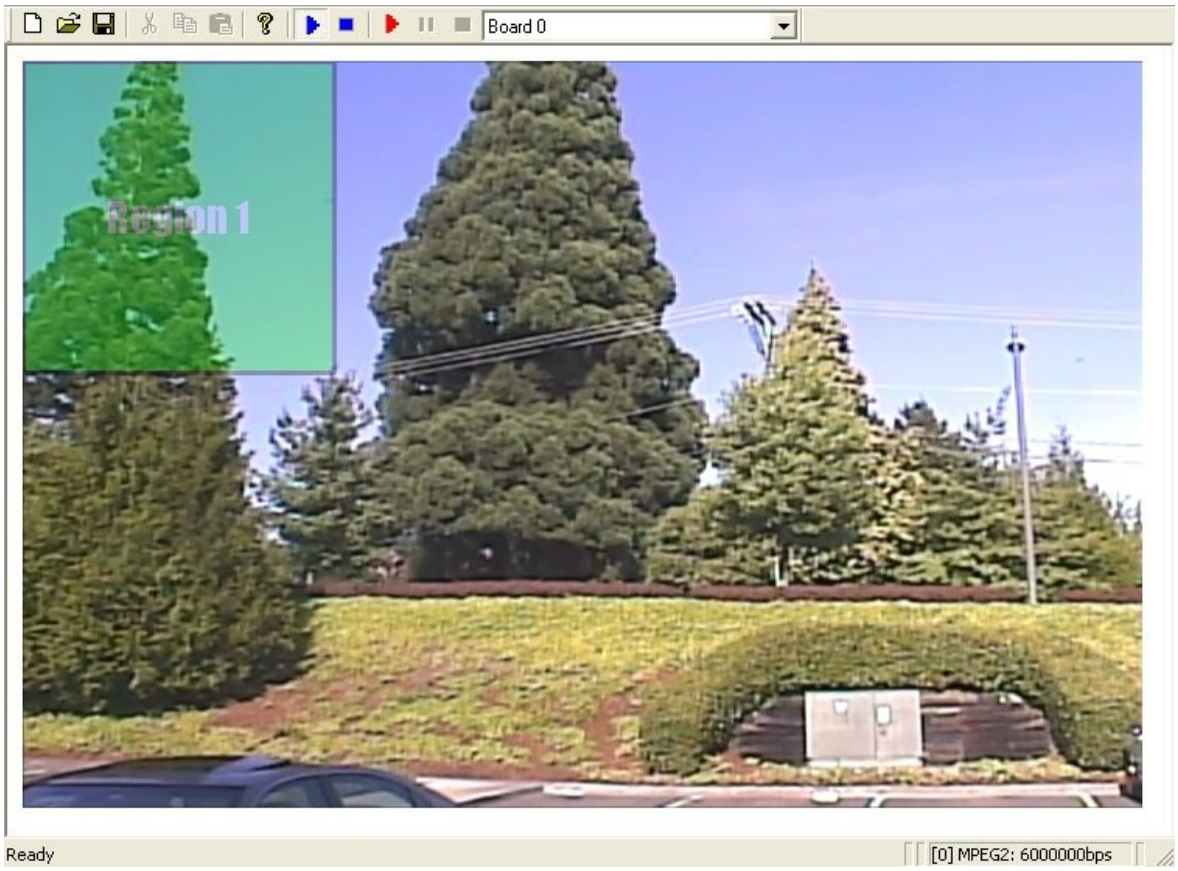

Figure 5: Small region in upper left corner

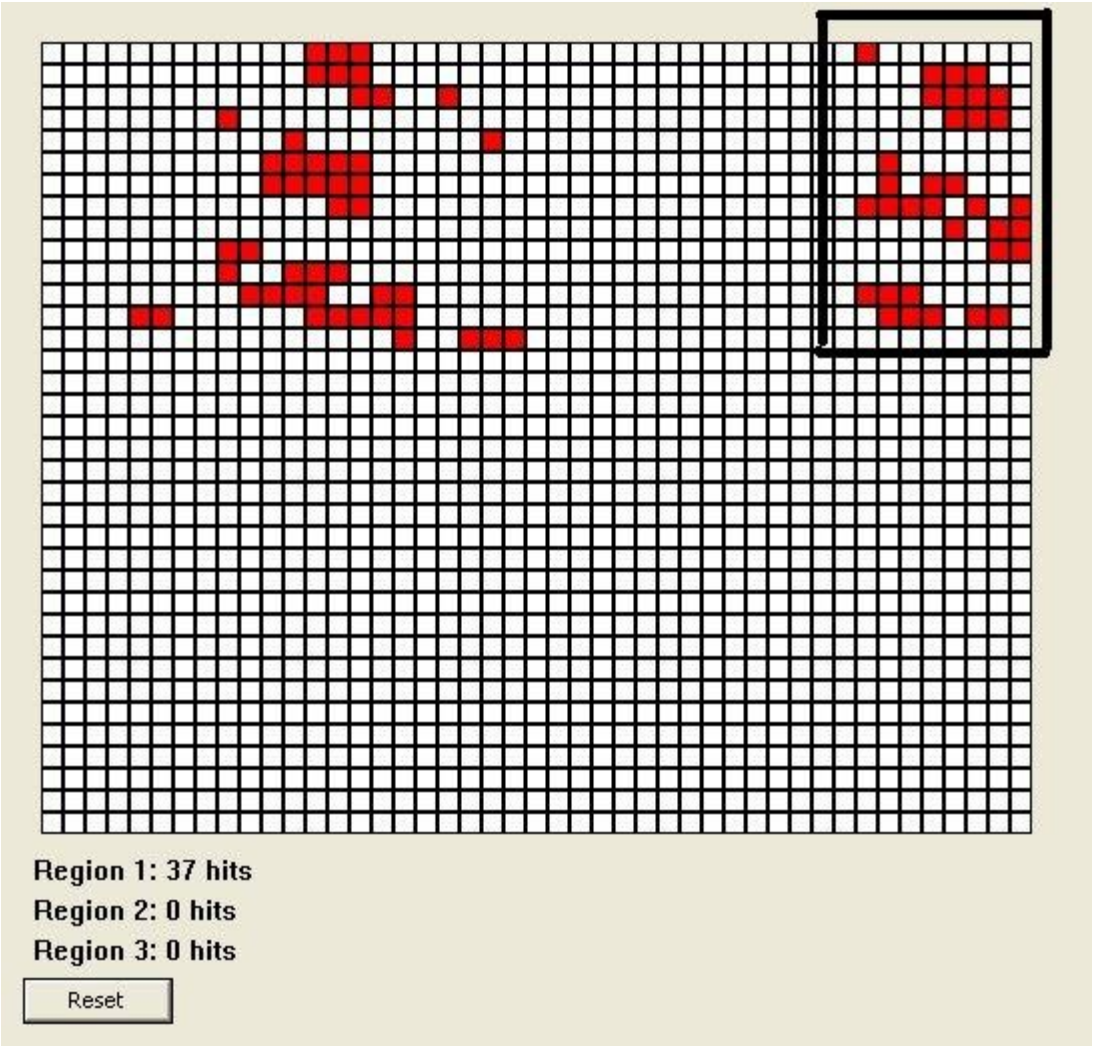

Figure 6

The produced motion map when motion occurs is shown in Figure 6. The black rectangle shows macroblocks outside the area of interest. As example earlier, these may be ignored. The number of hits 37 is the number of times a motion event occurred in region 1. It is not a count of the macroblocks in the region.

Advanced: Note that no macroblocks are shown below the region in the y direction. When a motion map is created, it scans from left to right starting at the top and going to the bottom. Any macroblock motion along the scan line is included in the motion map. So stray motion events(which may be ignored) occur beside the region, but not above or below the region.

Example 3: (Advanced) Custom motion detection

Some customers may want motion information on a macroblock by macroblock basis. If this is the case, then only one region should be defined. This region should encompass the whole picture in order to get all the macroblock events included in the motion map.

Step 1:

Set up the region to cover the full screen as shown in Figure 7.

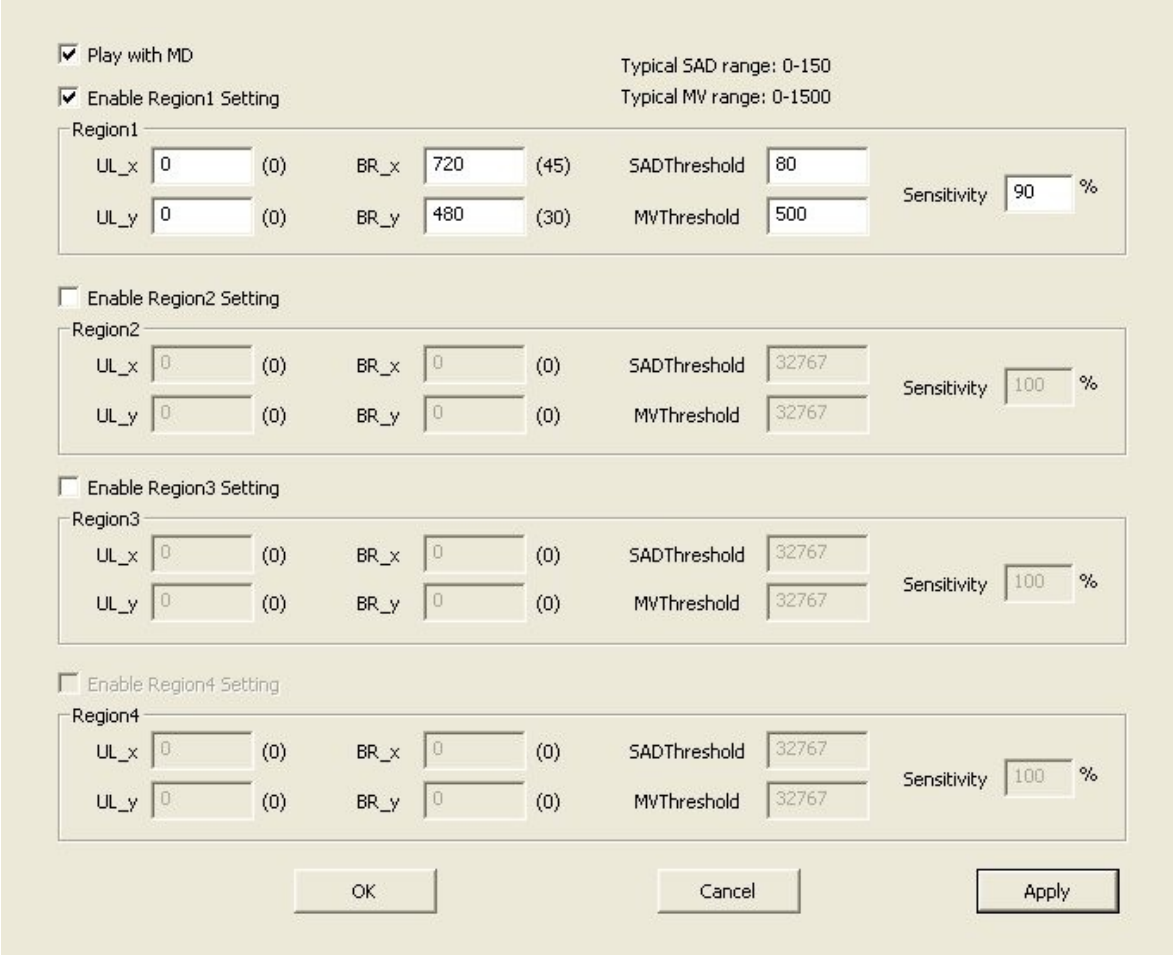

Figure 7

Step 2:

Process the motion map. Figure 8 shows a processed image map from the returned event data.

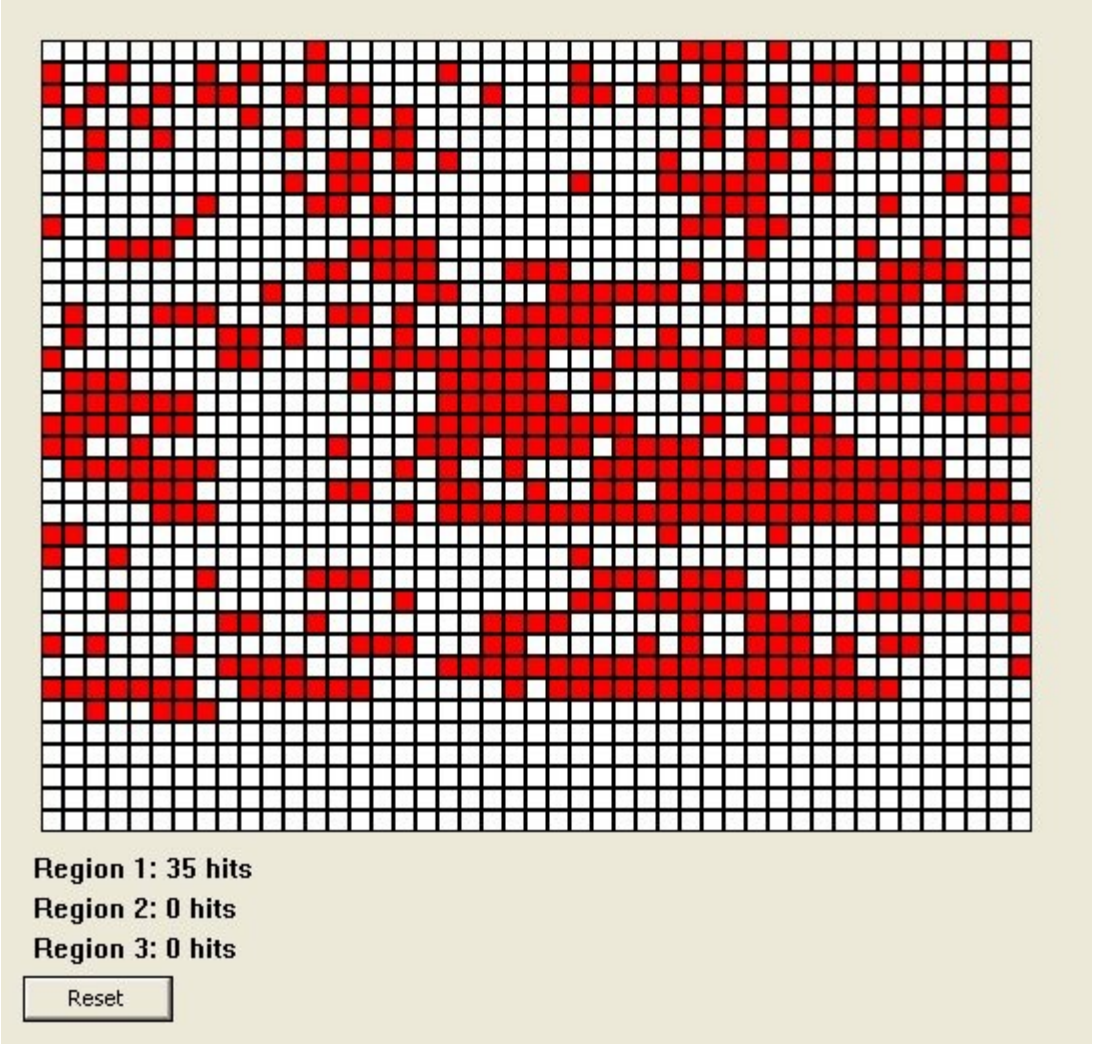

Figure 8

When the event is captured by the waiting user program(see demo source code), it calls SN_GetMDData(&region, pBuf, BUFSIZE, board_number). The region(s) where motion was detected is given in a bitmask in the region variable. BUFSIZE should be set to 208, the size of the motion map. pBuf will contain the macroblock map on a bit by bit basis.

For example, assume the following is returned by SN_GetMDData where pBuf is an array of unsigned 8 bit characters.

pBuf[0] = 0x7c;

pBuf[1] = 0x00;

pBuf[2] = 0x89;

From pBuf[0], we have the motion detect events for the first 8 macroblocks. 0x7c= 01111100b which represents the following:

Macroblock 1 : bit field = 0, so no motion.

Macroblocks 2-6 : bit field = 1, so motion detected

Macroblocks 7-8: no motion detected.

pBuf[1] = 0x00, so macroblocks 9-16 have no motion in them.

The motion map given in pBuf thus allows a user to create a custom motion detection algorithm(The threshold will be the same for each macroblock) where they can generate motion detection on an individual macroblock level (16x16 pixels).

Appendix A : Importing the DLL in C# .NET

Sensoray provides a C DLL for maximum portability between programming languages. This allows the use of Sensoray SDKs in Visual Basic, C# .NET, Visual C++ and of course C.

The question of how to send structures (with arrays) to any C dll from C# for instance is fairly common.

There are also some reference materials available from MSDN for using WIN32 DLL calls in C#.

C# example for using the function SN_SetOverlayText:

In C#, the following can be defined to use this function:

```
[StructLayout(LayoutKind.Sequential)]
public struct POINT
{
    public int x;
    public int x;
};

[StructLayout(LayoutKind.Sequential)]
public struct overlay_text_t{
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 96)]
    public string text;
    public int  fontidx;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 28)]
    public string reserved;
};

[DllImport("mid2250.dll")]
static extern int SN_SetOverlayText(int idx, ref POINT position,
                                    ref overlay_text_t text,
                                    bool enable, bool update,
                                    int board);
```