BasicScope Internal Design Document

06-JAN-2017

1. Philosophy

The program will be compilable under a commonly available and free environment (Microsoft Visual Studio Express), with standard Hantek SDK. Ok to use comctl32.lib. Use plain c so anyone can understand. Program will not be huge - put it all in one source file, if possible.
Clarify SDK doc where possible through experimentation. Publish results.
Open source - extra features can be added by others. Eventually release into public domain? This frees the original author to narrow the original software down to something achievable.
Not all things to all users. Focus on capture and examination of one-time event.
Recognize that hardware is limited, and document limitations.
Code things the easy way when it doesn't matter. For example, if there's an infrequently-used case where a small part of the screen needs to be redrawn, it will be fair to just invalidate and redraw the whole screen, rather than manage surgical update regions and refresh timing. Don't waste time avoiding floating-point operations and nested function calls if it doesn't improve performance meaningfully.
Not written for expansion, so low degree of abstraction where not needed. For example, if a buffer must always be 1048576 in size, the number will be used right in the program, not a #define of BUFFER_SIZE. If a constant is used all over the program, a define might be used as a convenience measure. Also, will not put wrappers around SDK calls without a clear reason.
No situations under which software will get stuck, and the user can't back out. For example, deleting a pager button that can't come back. Other than that, don't protect the user from themselves. If they want to resize the window to be unusably tiny and clip vital information, let them.

It is better to show no trace, than to show a wrong one.

2. Mandatory Features (what BasicScope is...)

1. Capture a trace with normal triggering
2. Capture a trace with one-shot triggering
3. Traces consistently positioned with respect to trigger in time
4. Ability to scroll horizontally to anywhere in trace
5. Mouse-wheel zoom to focus on a particular time region of trace
6. Drag-Pan that works as expected
7. Ability to do simple time, frequency, and voltage measurements
8. Save display as a PNG or GIF file
9. Ability to dump most of program state and data into a file, and restore it, a debugging feature included with first release for bug reports

3. Excluded Features (what BasicScope isn't...)

1. Run on any platform other than Windows XP. Will probably run on other versions of Windows, but testing is left to others.
2. Math, Invert, AC coupling, FFT, Reference Waveforms, Demo Mode. This kind of stuff is left to other authors through plugins.
3. Any probe support other than 1x or 10x.
4. Internationalization. Strings will be hard-coded right into the source in English, making core development easier. However, Unicode will be supported, so that other languages could be supported. If someone else wants to pull all the strings into resources, they can. Zero effort will be put into supporting languages that read in any direction except left to right.
5. Support of any hardware other than a single Hantek 6022BE or 6022BL. BasicScope will use the first connected scope it finds, without consideration to what else might be connected. This also implies that only two channels of data will be supported. The logic analyzer portion of the 6022BL will not be supported.
6. Flawless display during transitions. If there are artifacts when settings are being changed, that is acceptable, as long as the steady-state display is correct.
7. Real-time display of data being collected at slow scan rates. This means that even at 1 second per division, the data is displayed only after the whole buffer is collected, i.e., the user cannot watch the beam moving across the screen like a normal oscilloscope.
8. Auto-set based on input waveform.
9. Auto-measure of anything.
10. Unattended data capture. This could be done in a plugin, however.
11. User interface without a pointing device. It will be assumed that the user has a way to move the cursor to any position on the screen, has at least a left mouse button, and a way to hold that button down for dragging. A mouse with a scroll wheel will be the recommended setup.
12. Design for aesthetics, except as a consequence of functionality. There will be no user-selectable color schemes, for example.
13. Design for use without color. Color will be used as a key element to group data. For example, all Channel 1 related information will be one color.
14. Design for accessibility. Apologies to those who are color-blind, or blind, period.
15. Design for precision measurements. The underlying hardware has only 8 bits (256 levels) of voltage resolution, and moreover, the hardware is noisier than1 bit. BasicScope will output 3 significant figures of readout, but the last digit will be really iffy.

4. Possible Features (what BasicScope aspires to be...)

1. Save CSV
2. Triggers other than rising and falling edge
3. Plugins
4. Warn when displayed data might be clipped
5. Run on a 640x480 display (800x600 minimum preferred)

5. Theory of Operation

BasicScope uses controls such as sliders and checkboxes from the standard and common control Windows libraries. In general, each control sets a global variable to indicate its state.

The rest of the program reads these variables when they are needed for an action, such as taking a trace.

No Hardware Trigger

The 6022BE does not have any hardware triggering circuit, nor any provision for external trigger. What the scope does is blindly take about 1k samples on each channel, then ship these to the host (the PC), to determine whether a trigger event has occurred. This paradigm has the serious deficiency of missing possible trigger events, especially at high sample rates, but does save significant cost. In sum, unreliable triggering is a given expectation of the product, and cannot be cured in software.

Support for 6022BL by Loading a Different HTMarch.dll

The 6022BL is substantially similar to the 6022BE, except that it has a different device name for its USB interface. The name of the 6022BE file in Windows is \\.\6022-n, where n is the device index. Similarly, the 6022BL file is \\.\602a-n. These names are hardwired into the HTMarch support libraries, making each library only compatible with the hardware for which it was explicitly intended. This leaves the following options for supporting both devices:

- Compile two separate programs (fork the source), one linked to the BE static library, and distributed with the BE HTMarch.dll, and the other linked to the BL static library, and distributed with the BL dll.
- Write a universal HTMarch that would handle both devices. This would be trivial if the source were available, but is an onerous reverse engineering job without source. This is, in fact, the approach taken by Open6022BE by RichardK.
- Have a single program which links to the static 6022BE library, and at install time, install either the 6022BE or 6022BL HTMarch.dll. This was a previous approach for BasicScope, but relies on the oscilloscope section of a 6022BL being selected by default, since dsoChooseDevice() is not available in the 6022BE library. More likely, a custom dosChooseDevice() equivalent would need to be written.
- Instead of linking to a particular static library, load the needed version of HTMarch.dll at runtime. This is what BasicScope does. To make this work, a custom GetDeviceType() determines which scope is connected, then the LoadHantekLibrary() routine loads the appropriate library. GetDeviceType() replaces the function of dosOpenDevice(), which simply checks for the presence of the USB device file.

Hantek Trigger

The detection of a trigger event is embedded somewhere in the Hantek API, probably not at the driver level, but in the code for *dsoReadHardData()*.

Software Trigger

Consider a trace buffer that is 2000 µS, in which the user wishes to view a point 1500 µS before the trigger event. This means that the trigger event must occur within the last 500 µS of the buffer. But if the input waveform is repetitive with a period of 100 µS, then the first trigger found will occur within 100 µS of the start of the buffer. There will be other trigger events later in the buffer, but these will be disregarded by the hardware, once the first trigger is found. BasicScope solves this problem by having a software function to search for a trigger event only within a certain range in the buffer - that range which would result in the trace being displayable on the screen. This is the *FindTrigger()* function.

The search algorithm of *FindTrigger()* is modeled after the one used by Hantek, returning the index of the sample *just before* the waveform reaches the trigger level.

FindTrigger() is used to implement the First Trig and Next Trig command buttons. If the scope is actively acquiring data, these buttons will first stop the data acquisition, so that there will be no contention for the data buffer..

Auto Trigger Mode Always Used

dsoReadHardData() allows Single, Normal, and Auto trigger modes. Auto mode delivers samples in the range of -160 to +160 in theory, although actual samples may come in from -164 to +164. In Single and Normal mode, the samples range from 0 to 255, but only sometimes. Because of this inconsistent behavior, BasicScope always uses Auto mode.

TimeParms[ ] and VoltParms[ ] Tables

In general, the horizontal direction means time, and the vertical direction means voltage. There is no x-y mode in BasicScope. Two read-only tables, TimeParms[ ] and VoltParms[ ], contain information relevant to the time and voltage settings, respectively. The TimeDiv trackbar controls the time ("sweep") settings, and the Volt1 and Volt2 trackbars control the voltage settings for each channel. Each trackbar returns a position, which is used as an index into the TimeParms[ ] or VoltParms[ ] table as appropriate. These tables serve the same function as the methods of a time or voltage object, if the program were written in c++.

Multiple Threads, but Not Thread-Safe

There are 3 threads, the *Acquisition Thread*, which runs only a single function to collect data, the timer thread, associated with a timer used to flash the color on a control, and the main program thread (*Main Thread*), which is everything else. The timer thread is part of a Windows function, and no more will be said about it. The main program makes no general effort to be thread-safe. There are no atomic semaphores (except for the mutex that assures only one copy of BasicScope is running). Some race conditions are permitted between the *Main Thread* and the Acquisition Thread, but only when the effect would be self-healing. Special care was taken to insure proper interlock between the *Main Thread* and *Acquisition Thread* during data collection (see below).

Double Buffering for Data Collection

BasicScope uses the dsoReadHardData() call from the SDK to read a buffer of data at a specified sample rate and voltage setting. The size of the buffer read depends on the sample rate, and BasicScope seeks to mimic the buffer sizes used in the OEM Hantek application. The buffer structure holds the digitized samples representing the waveforms for both channels, the voltage and sample rate parameters under which the trace was captured, as well as a **valid** flag.

There are actually two buffers employed in a double-buffering scheme. A pointer, **BufToFill**, points to the buffer to be filled. If that buffer is invalid (that is, its **valid** flag is false), The *Acquisition Thread* calls dsoReadHardData() to fill the buffer. That buffer is then marked valid, and the *Display Window Paint Procedure* is called indirectly by the winapi call InvalidateRect().

The *Display Window Paint Procedure* sees that the non-displayed buffer pointed to by **BufToFill** is now valid. The presently displayed buffer is marked invalid, **BufToFill** is changed to point at the other buffer, and the most recently filled buffer is displayed on the

screen appropriately. As **BufToFill** now points to an invalid buffer, the *Acquisition Thread* will proceed to fill that buffer.
The *Acquisition Thread* can be thought of as the producer of buffers, being the only code that ever writes to a buffer (except for the **valid** flag, discussed below). The *Display Window Paint Procedure* can be thought of as the consumer of buffers. The *Acquisition Thread* will only operate on a buffer pointed to by **BufToFill**, and marked invalid. The *Display Window Paint Procedure* will only read a buffer that is *not* pointed to by **BufToFill**, and is marked valid.
The **valid** flag defines the ownership of a buffer. Neither the *Acquisition Thread* nor the *Display Window Paint Procedure* can forcibly claim a buffer, they each can only yield it. The *Acquisition Thread* yields a buffer by marking it valid, while the *Display Window Paint Procedure* yields a buffer by marking it invalid. The *Display Window Paint Procedure* is the only code allowed that swaps buffers by altering **BufToFill**.
Another paradigm is that generally, no code should ever write or alter a buffer that is marked **valid**, and no code should read a buffer unless it marked **valid**, with the exception of changing or reading the **valid** flag itself. There is an exception to this rule for when the *Acquisition Thread* scans its own buffer to determine whether it is good, but that is considered part of the acquisition.

Vertical (Voltage) Scale

The voltage must be set in two places. First, whenever a voltage setting trackbar is moved, there is an immediate call to dsoSetVoltDIV(). It is also necessary to set the voltage when calling dsoReadHardData() in the *Acquisition Thread*. For the scope to run properly, both of these must be done. Because these two settings are made at different places in the code, and in different threads, there is a slight chance that the two will be out of sync. The window in which the race condition occurs is only a few instructions. It is a transient condition, and the proper display will be shown in the next trace if the voltage slider for the channel is stable. This was an intentional tradeoff to gain performance. For absolutely correct sampling every time, the *Acquisition Thread* would have to call dsoSetVoltDIV() before taking every sample. There are only 8 voltage settings with a 1x probe. If a 10x probe is selected, there is no change in the operation of the hardware or software, except to report the data on a different scale.

Horizontal (Time) Scale

Like the voltage setting above, the time setting must occur in two places, and takes the same performance vs. correctness tradeoff. Whenever the time setting trackbar is moved, there is an immediate call to dsoSetTimeDIV(). It is also necessary to set the timediv again when calling dsoReadHardData() in the *Acquisition Thread*.
The 6022BE has a top sampling rate of 48 Msps. This corresponds to the 2 uS/div time setting. If faster sample rates are selected, the Hantek driver simply interpolates data to artificially create more points in the returned data. For this reason, BasicScope never samples at faster than 2 µS/div. If a faster rate is selected, a smaller portion of the buffer is displayed. This allows a buffer of 1012 real samples to be taken, so that the user can scroll around later. (1012 is not very long in the first place, but better than the almost useless 10 real samples that would be taken at 20 nS/div).

Scrolling Left and Right

There is a horizontal scroll bar at the bottom of the Display Window used to move the displayed trace to the right or left of the trigger. The units of this scroll bar are display

samples. A variable, **DisplayTimeDiv**, keeps track of the TimeDiv of the trace being displayed. When a trace is about to be displayed, its **.timediv** field is compared to **DisplayTimeDiv**. If the two are different, **DisplayTimeDiv** is set to the **.timediv** of the trace, and the scroll bar range and page size are adjusted such that the beginning and end of the trace can be scrolled to, no matter where the trigger might be in the trace. This also means that the scroll bar can always scroll to a place where there is no trace data, meaning that nothing will be displayed on the screen. This is not a bug, it is intentional, as a subsequent trace might be able to display data there.

Two signed integers, **DelaySamples** and **DelayTicks**, keep track of the displayed center of trace, relative to the trigger event. As the name implies, the units of **DelaySamples** are samples, and in fact, **DelaySamples** is equal to the position of the the thumb on the scroll bar. When **DelaySamples** is 0, the trigger event is centered on the screen. If **DelaySamples** is +n or -n, then the center of the screen is, respectively, n samples to the right or left of the trigger. The units of **DelayTicks** are 1/(48 MHz), an absolute time. For example, when DelayTicks is -480, the display is centered 10 µS to the left of the trigger.

Any time the user scrolls or pans the display, **DelaySamples** is adjusted to reflect the new value, and **DelayTicks** is immediately recalculated also.

If a new trace comes in with a new **.timediv** as mentioned in the first paragraph, **DelayTicks** is used to recalculate **DelaySamples**. This is the only time the **DelayTicks** drives **DelaySamples**.

Note that **DisplayTimeDiv**, **DelaySamples**, **DelayTicks**, and the horizontal scroll bar itself, have no meaning if there is no trace being displayed.

Zooming

A user may zoom a displayed trace vertically or horizontally by adjusting the voltage or time trackbars, respectively. Additionally, horizontal zoom can be controlled by the keystrokes '+' and '-', or by the mouse wheel. Floating-point variable **HZoomFactor** controls the horizontal zoom, while **VZoomFactor1** and **VZoomFactor2** control the vertical zoom for channels 1 and 2, respectively. A zoom factor of 1.0 means that the trace is presented in original resolution, a factor of less than 1.0 means the trace is zoomed in, and a factor of greater than 1.0 means the trace is zoomed out.

Each time setting is kept in a table, **TimeParms[ ]**, along with a scale factor representing the ratio of that setting to the next. When the time trackbar has a different setting from that associated with the displayed trace buffer, data from the **TimeParms[ ]** table is used to calculate a new **HZoomFactor**. **VZoomFactor1** and **VZoomFactor2** use an analogous process, with the **VoltParms[ ]** table.

Because there are discrete time and voltage settings, zoom is not continuous - it is limited to display at one of the predefined settings.

The Hantek *HTDrawWaveInYT()* function is accepts the above zoom factors. At some extremely high magnifications (possibly that would equate to displaying 1 sample or less), nothing is drawn, but such a display would have been of little value, anyway.

Zooming never alters the original data in the displayed trace buffer. However, it does alter the voltage and/or time trackbars, meaning that subsequent traces will be taken with those settings.

Display Window Grabs Focus on Entry

The Display Window will grab focus for itself any time the cursor enters its client area, and release focus (SetFocus(NULL)) when the cursor leaves. This was a somewhat controversial decision, and disapproved of by some on StackOverflow. However, the author believes that it is entirely natural for the user to want to do something with the Display Window, if the cursor is positioned within that window.

Markers for Measurement

Markers are the only means provided for time and voltage measurement, other than the user visually estimating by looking at the grid. The Markers[ ] array contains records of the positions of the markers. Elements 0 and 1 correspond to channel 1, while elements 2 and 3 correspond to channel 2.

Each marker has two sets of (x,y) coordinates. The Measure coordinates represent the position of the marker relative to the displayed waveform, and are invariant with respect to changes in zoom and pan either horizontally or vertically. The units for MeasureX are ticks 1/(48 MHz) relative to the trigger for the waveform. MeasureY has the units of volts relative to the zero level for the waveform. ScreenX and ScreenY are the pixel coordinates of the marker, and are recalculated from Measure coordinates every time the voltage or time scale changes, or the display is scrolled horizontally or vertically. Measure coordinates only change when a marker is dragged to a new location.

The characters ⊕ and ⊞ are used to draw the markers. These come from the “DejaVu Sans” font, so that font must be loaded in order to display the markers.

Status Window

The Status Window just under the Display Window is a rolling log of messages to the user. The size is rather large, so it is actually unlikely that a message will roll off the top. Generally, the Status Window is only for reporting:

- An echo of errors that have popped up a MessageBox
- A warning of some kind
- Information that the User specifically requested

It is NOT for messages about ordinary operations (such as “Trace Started / Stopped”), and because of the overhead, should not be used when the scope is running, unless an error occurs.

6. Development Map

1. Basic trace capture with the SDK, to verify can do it.
2. General layout of UI, including all controls. Controls can be inert.
3. Get double-buffering scheme working for capture/display.
4. Single capture at one voltage and sample rate.
5. Capture at other sample rates
6. Capture at other voltage settings
7. Make hardware triggering work
8. Zoom
9. Scroll
10. Pan
11. Legend area

12. Measure
13. Save display
14. Software trigger

7. Still Left to Implement (not a complete list)

1. ~~Stroke modes – zoom, pan, and measure~~ abandon
2. Info dump to status area

8. Known Bugs

When upgrading between BasicScope versions, button information can get scrambled

9. FAQ (move to other document later)

How Do I Report a Bug?

My email address in on the About page of this site. A clear description of how to reproduce the bug reliably is best, but if not, any information could be useful. For example:

- The program version (for example 0.13)
- What operating system the program was running under
- The actions going on at the time of failure (best recollection, even if it seems irrelevant)
- Screenshots or Saved Traces

What are the Hardware and Software Requirements for BasicScope?

Aside from the Hantek 6022BE itself, any computer running Windows XP or later should be fine. BasicScope does not run on any platform other than Windows, since the SDK only supports Windows. An 800x600 display is recommended as a minimum for best results. A pointing device is required (not all functions are keyboard accessible), and a mouse with a scroll wheel is highly recommended. The Hantek 6022BE drivers must be installed (the easiest way to install them is to install the software that comes with the scope).

Why aren't the controls labeled better?

This was an attempt to un-crowd the screen. The main display will show the scope settings anyway, and tooltips on controls help in some cases. A drawback is that a novice user may be intimidated, but as BasicScope is not a commercial product, it is hoped that after a few minutes, the user will be keying in to the color and location of controls, and not notice the lack of labels.

Why the loud color scheme? It looks like a toy.

Well, the 6022BE sort of *is* a toy, but that notwithstanding, the bright yellow is to match things associated with channel 1, for which the BNC jack on the actual device has a bright yellow ring. Channel 2 has a similar philosophy.

The Icons on the Buttons are Terrible.

The images are low priority for fixing. If someone has a consistent icon set that will work better, the author would be glad to consider them. 48 pixel high, 256 color bmp, please.

Why is the fastest time division 25 nS, and not 20 nS?

Hantek's OEM software, as well as the Open6022BE software, have 20 nS/div as the top rate. However, this translates to 9.6 samples per 10-division screen at the 48 Msps maximum sample rate. With the 25 nS/div setting, there are 12 samples per screen, so at least some of the samples line up with grid lines. There is really scant advantage to the user in displaying only 12 or even 24 samples per screen. By this reasoning, perhaps the fastest time division should really be 100 nS/div, but the faster ranges are included because doing so costs nothing in software.

There are Transient Visual Artifacts on the Screen.

Sorry about that. There will be flicker, ghosts, perhaps an extra line drawn here and there, especially while the scope is running, and/or settings are being adjusted. This was a tradeoff between a visually clean screen at all times, and best performance at gathering data, especially on older and slower computers. When the scope is stopped, and no settings are being adjusted, the display should be free of artifacts.

It is Taking Forever to Capture a Trace / Update the Trace

There is no hardware trigger in the 6022BE. This means that a trace is captured blindly, then software determines whether a trigger has occurred. When the overhead of checking for a trigger is large compared to the time spent capturing the actual trace, the chances of catching a trigger decrease. These suggestions may help:

- Try to take traces at 5 µS/div or slower. At those rates, the capture buffer is of decent size. At 2 µS/div or faster, the capture buffer is tiny (it's a hardware limitation).
- Use the Reset Pan button, or use the scroll bar under the waveform display to scroll to approximately the middle. BasicScope will reject a trace that would not result in something displayable on the screen, and the closer you are to centering the trace, the better the chances of not having a trace rejected.
- If you have scrolled left of the center of the waveform display, turn **on** Software Trigger. *Without this, a repetitive waveform may never display*, because the default Hantek trigger will trigger too early, resulting in traces that are not displayable and always rejected.
- In cases other than the above, turning **off** Software Trigger will reduce processing overhead, and may improve the chances of triggering. The author believes that this improvement is slight, however, and that it does little harm to keep Software Trigger on all the time.
- The waveform is not displayed until the data is fully captured, which means *a wait of ten seconds if the time/div is set to 1 second*. Use faster sweep rates when possible.

The Trace Suddenly Vanishes When I Scroll it.

This is an artifact of the Hantek SDK plotting function. If a trace would not cover at least the entire right half of the display, no trace is plotted at all. Scroll left or right and the trace should become visible again.

Where is the Auto Trigger mode?

Without a reliable trigger holdoff, an Auto Trigger mode would be equivalent to triggering on anything, which is of limited value. But if you wish to simply display waveforms without regard to triggers, uncheck the "trg" boxes for both channels.

Why Can't the Trigger Level be Set to Exactly 711 mV?

The underlying hardware is digital, with limited resolution, so not all values are available. The software will display the closest approximation to the actual value for things such as time and voltage. This applies to anything on the display, including the waveform(s), the marker(s), and the text legend at the bottom.

The Markers Report an Interval of 1.00 mS, but the frequency is 1.01 kHz

This is a side effect of the quantizing that the scope does, plus floating point arithmetic in the software. In general, the last digit may be off by 1 or 2, and it is not a good idea to rely on this equipment for high accuracy measurements, anyway. Zooming in before placing a marker *may* allow better precision in placement.

How Precise and Accurate are the Displayed Values?

Although numbers are presented with 3 significant figures, the underlying hardware has a voltage resolution of only 8 bits at best. Time measurements are quantized to the sampling rate, which is sometimes not a round decimal number. Given the accuracy of the actual hardware, the author's opinion is that only 2 digits of accuracy can be relied upon.

10. Author's Remarks

I love a good sleuth. Most of my career was spent in diagnosis and forensics, trying to figure out how a thing works, or how a thing failed. In that vein, I would like to thank Hantek for their wonderfully abstruse and arcane SDK documentation (even in the original Chinese). Truly a tyro-class production, it provided many wonderful hours of diversion.

Seriously, though, it's a scope at a bargain price point, and for the cost-constrained hobbyist like myself, it's much better than the alternative, which is nothing.

I will be releasing my best-guess at how things work. If Hantek or others would like to chime in with corrections, they would be more than welcome.

Roderick Young
roderick.young@lycos.com