Using the CoreEl Typhoon Card and SDK

Typhoon Supplementary Information

# Overview

This document provides information for installing, using and developing software for the CoreEl Typhoon video processing card. The document is intended to supplement, and refers to, the Typhoon manual and other materials supplied by CoreEl themselves.

# Materials Supplied by CoreEl

The following materials are supplied by CoreEl and provide most of the information required to set-up and use the CoreEl Typhoon card:

|  |  |
| --- | --- |
| **Item** | **Description** |
| typhoon\_user\_manual.docx | The Typhoon User Manual. This contains information about the card itself, installing the card and configuring the card. |
| typhoon\_sdk\_api\_specification.docx | The Typhoon SDK API Specification. This contains information about APIs and data structures used to control the card using C++. |
| Typhoon\_Package\_vX.X.X.zip | The Typhoon drivers, application and firmware package. As of writing the latest version is Typhoon\_Package\_v1.2.0.zip  **Note:** there is an example app provided in Typhoon\_Package\_v1.1.9.zip, but absent from other package versions, that provides some guidance on using the APIs |
| TyphoonSDK.h | The header file for the Typhoon C++ API. This defines the set of classes, structures and functions needed to programmatically control the Typhoon card |
| TyphoonSDK.lib (distributed inside TyphoonSDK\_64bit.zip) | The static library for the C++ API. This must be linked to a client application using the **Microsoft Visual Studio 2013** tool-chain. |

## Streampunk Styphoon

In addition to material supplied by CoreEl, this document also refers to the Streampunk ‘Styphoon’ project (available on github here: <https://github.com/Streampunk/styphoon>) which contains a working set of bindings to use the Typhoon card from node.js and C++ on Windows.

# Installation

Prior to installation, as described in the [Typhoon User Manual](#Typhoon_User_Manual), it is necessary to carry out a number of steps to enable the successful installation of the Typhoon drivers, and to ensure the on-going successful operation of the card:

## Disable driver signing in OS

The Typhoon drivers are not signed, and so for the drivers to be loaded correctly in Windows 10 it is necessary to disable driver signature checks in Windows. This is done (permanently, until these steps are reversed) using the follow sequence of operations:

1. Open an Administrator command prompt, by clicking the ‘Start’ button and typing ‘cmd’. When ‘Command Prompt’ shows up as the best match, right click this and select ‘Run as administrator’
2. Once the command prompt is showing, type and run the following two commands:

bcdedit.exe -set loadoptions DDISABLE\_INTEGRITY\_CHECKS

bcdedit.exe -set TESTSIGNING ON

1. Reboot the computer

## Ensure the Card is installed in a compatible slot

There are two considerations for locating the card inside a PC:

1. It must be located in a position where there is sufficient airflow over the card’s cooling fans: the card generates a lot of heat, and if the card is located directly between other cards this can cause it to overheat. If possible leave a slot’s gap between the cooling-fan side of the card and the next card, or locate the Typhoon in the last slot so the cooling fans have plenty of space above them. **Note:** card temperature is shown at the bottom of the ‘Scoreboard’ panel of the CoreEl Typhoon GUI. Click the ‘refresh’ button to view the current temperature. If correctly positioned this should never go above 80 degrees Celsius. Going above this temperature can cause hardware malfunction.
2. The PCIe slot into which the card is inserted must either be PCIe 2, or else (in the case of a PCIe 3 slot) configured in the BIOS to be PCIe 2. Putting the card into a slot running as PCIe 3 can cause PCIe bus errors.

## Reboot the PC with driver signing disabled

Windows 10 enforces driver signatures by default. This can be disabled to install drivers that are not digitally signed. As the Typhoon drivers are unsigned, then once the card is physically installed, it is necessary to reboot the PC with driver signing disabled in order to run the driver installation program. This is achieved as follows:

1. Click the Start menu button (Start) and select settings (either by typing it, or by clicking the settings ‘cog wheel’ icon on the start menu.

2. Click Update and Security.

3. Click on Recovery.

4. Click Restart now under Advanced Startup.

5. Click Troubleshoot.

6. Click Advanced options.

7. Click Startup Settings.

8. Click on Restart.

9. On the Startup Settings screen press 7 or F7 to disable driver signature enforcement.

Your computer will restart and you will be able to install non-digitally signed drivers. If you restart your computer again the driver signature enforcement will be re-enabled.

## Install the Typhoon Drivers and GUI

* Unpack the Typhoon installation package [zip file](#typhoon_package_zip) to a suitable location.
* Follow the driver installation instructions in the [Typhoon User Manual](#Typhoon_User_Manual) to complete installation. This essentially consists of installing the three sets of drivers by running the 'Typhoon\_Driver\_Installer.exe' under the ‘Drivers\All’ folder in the Typhoon installation package.
* Follow the GUI installation instructions in the [Typhoon User Manual](#Typhoon_User_Manual). This consists of installing the GUI by running ‘Typhoon-1.x.0-windows-installer.exe’ under the GUI folder in the Typhoon installation package.

## Update firmware

It is recommended that the Typhoon card is updated to the latest firmware (at time of writing, version 1.2.0). This is achieved using the following steps:

1. Open the Typhoon UI.
2. Select the Upgrade tab (for more information about this tab see the [Typhoon User Manual](#Typhoon_User_Manual).
3. Click on the Browse button and browse to the ‘Firmware’ folder in the Typhoon installation package extracted previously.
4. Once you have selected a valid location the ‘Upgrade’ button will enable. Click this and wait whilst the firmware is upgraded.

## Select Firmware Design

The Node-RED bindings have been designed to work with Firmware design 4, ‘4 Encoders design without FEC support’. The Node-RED binding code will not initialize if a different design is selected. Ensure this design is enabled using the following steps:

1. Open the Typhoon UI.
2. Select the System tab.
3. Select design 4 from the list at the bottom of this tab.
4. Click ‘Apply’.

Upon applying a new design, the Typhoon card has been observed to initiate a system reboot by causing a PCIe bus error, which manifests itself in the host PC rebooting without warning, and then displaying a hardware error notification. In *most* cases you can ignore this behaviour: when the PC boots back up, check that Typhoon UI recognises the Typhoon card, and that the desired design is now showing in the System tab. If this is the case then design switch has been successful.

**Note:** occasionally this design change operation has been observed to cause the card installation to become corrupted, resulting in the card no longer being recognised by the Typhoon UI. If this occurs it requires that the three Typhoon drivers be uninstalled, and the driver installation process described above be repeated.

## Configure the Typhoon’s Network Adapter

The Typhoon’s Network Adapter needs to be bound to a range that is different to the normal network range - for instance if the normal network range (i.e. that configured for the PCs main network adapter) is 192.168.0.xxx then the adapter should be bound to something like 172.16.0.100 - it must at least be on a different subnet to the normal network adapter; and with no gateway.

This Typhoon’s network adapter IP address is set from the System tab on the Typhoon UI.

## Enable audio and compressed content

The Typhoon APIs have functionality to enable audio and compressed video; however, these APIs only allow audio and compressed video to flow if this functionality is ***also***enabled in the Typhoon UI. To enable this for a particular channel, go to the ‘Configuration’ tab and make sure the ‘Enable receiving video and audio from stagebox’ checkboxes are checked for the channels you wish to use.

# Using the SDK

## General Notes on the SDK

In general the Typhoon SDK is straightforward to use. The API definition file, [TyphoonSDK.h](#typhoon_sdk_h), defines a simple object model consisting of a TyphoonBoard class representing the Typhoon card itself, which provides access to several (4 for Firmware design 4) TyphoonChannel class objects for controlling the individual channels. For the most part the APIs are self-explanatory; however the API doesn’t quite provide all the control required to successfully use the card, in the following ways:

* Some significant card functionality can only be controlled or queried via on-card registers, and these register addresses aren’t described in any Typhoon documentation or API headers.
* The format of audio and video data supplied by the ‘TyphoonChannel::GetFrame()’ API is limited in its configurability, and typically requires some post-processing to turn it into a useable well-known format.
* Certain APIs don’t work quite in the way you would expect from the API definition

## Configuration of the card using Register Read/Write

Two areas of functionality that must be accessed via a card register Read or Write operation are:

* The ability to ***read*** the Input Signal Standard that is being seen at the input to a channel.
* The ability to ***set*** the output format in which frames are delivered to the client software by the Typhoon SDK.

The register locations and appropriate values aren’t published by CoreEl. Helper code to access these registers is defined in Styphoon in the following files:

|  |  |
| --- | --- |
| [TyphoonSDKWrapper.h](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonSDKWrapper.h) | Numeric constants representing values that can be read from or written to the Typhoon’s registers |
| [TyphoonRegister.h](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonRegister.h) | Declaration of a helper class for reading from and writing to the registers associated with each of the four input channels. |
| [TyphoonRegister.cpp](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonRegister.cpp) | Implementation of the above, including a declaration of the register addresses |

### Reading InputSignalStandard

Reading the InputSignalStandard for each channel is carried out by calling TyphoonBoard::RegRead() on the following registers

|  |  |
| --- | --- |
| **Channel** | **Register** |
| 1 | 0x11C4 |
| 2 | 0x25C4 |
| 3 | 0x35C4 |
| 4 | 0X45C4 |

With the possible return values being as follows:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0x00000000 | FORMAT\_1080P\_60 |
| 0x00000001 | FORMAT\_1080P\_5994 |
| 0x00000002 | FORMAT\_1080P\_50 |
| 0x00000003 | FORMAT\_1080P\_30 |
| 0x00000004 | FORMAT\_1080P\_2997 |
| 0x00000005 | FORMAT\_1080P\_25 |
| 0x00000006 | FORMAT\_1080p\_24 |
| 0x00000007 | FORMAT\_1080p\_2398 |
| 0x00000010 | FORMAT\_720P\_60 |
| 0x00000011 | FORMAT\_720P\_5994 |
| 0x00000012 | FORMAT\_720P\_50 |
| 0x00000013 | FORMAT\_720P\_30 |
| 0x00000014 | FORMAT\_720P\_2997 |
| 0x00000015 | FORMAT\_720P\_25 |
| 0x00000018 | FORMAT\_1080I\_60 |
| 0x00000019 | FORMAT\_1080I\_5994 |
| 0x0000001A | FORMAT\_1080I\_50 |
| 0x0000001E | FORMAT\_NTSC |
| 0x0000001F | FORMAT\_PAL |

One further detail to note is that if you wish to read the signal standard of the signal actually being received using this approach, and then subsequently programmatically set the signal standard using the ‘SignalStandard’ parameter of TyphoonChannel::Open(), then the values used for specific standards in the Open() call are different to the values retrieved from the register; and so a mapping operation is required to convert them. Such a map is defined within the Styphoon project as ‘TPH\_DISPLAY\_MODE\_TRANSLATION\_MAP’ defined in [TyphoonTypeMaps.h](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonTypeMaps.h) and [TyphoonTypeMaps.cpp](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonTypeMaps.cpp)

### Setting Uncompressed Video Output Format

This setting can cause some confusion, as there is a parameter called FrameFormat, supplied to the TyphoonChannel::Open() API that looks as though it defines the frame format, but doesn’t. This parameter can have a value of either TPH\_UYVY or TPH\_V210, which would appear to request the card to output uncompressed video in either UYVY or V210 format. However, what this parameter does is to simply specify that sufficient buffer should be made available to accommodate each of these formats, should it be available from the FPGA. By default, the card always outputs UYVY: in other words, without doing anything else, then even if v210 content is being presented to the card input, and this parameter is set to TPH\_V210, the uncompressed output buffers will still be in UYVY format. The only effect of the TPH\_V210 FrameFormat parameter to Open() is that a frame buffer sufficient to hold v210 content is allocated; but then this is still populated with UYVY format frame data.

In order to actually configure the card to output V210-compatible-format frames it is necessary to write a value into a channel-specific register as follows:

|  |  |
| --- | --- |
| **Channel** | **Register** |
| 1 | 0x1004 |
| 2 | 0x2004 |
| 3 | 0x3004 |
| 4 | 0x4004 |

With the values being

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0x00 | UYVY |
| 0x03 | V210 |

With this in mind, and for further clarification, a table of expected FrameFormat behaviours with the various settings is shown below (where known):

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Format** | **FrameFormat value for Open** | **Value written to Register** | **Format of Frames Produced by TyphoonChannel::GetFrame()** |
| UYVY | UYVY | - | UYVY |
| UYVY | V210 | - | UYVY |
| V210 | UYVY | - | None – Frame Buffer too small |
| V210 | V210 | - | UYVY |
| V210 | V210 | V210 | V210 |

**Note:** in addition to the settings above, it is also necessary to apply a byte-level transform on the 10bit video pixel data to get it into actual V210 format. This is explained in the following section.

## Video Processing

As mentioned in the previous section, several steps need to occur in order to configure the Typhoon card to output video with 10bit-per-channel pixel data. However, once this is done, a final transform needs to be carried out to convert the pixel data into the correct byte-order for V210. The algorithm to do this is implemented inside TyphoonCapture::ForwardNextFrame() in [TyphoonCapture.cpp](https://github.com/Streampunk/styphoon/blob/master/src/TyphoonCapture.cpp); but is detailed here for information. This code only needs to be run over the uncompressed video buffer returned from TyphoonChannel::GetFrame() when the Frame Format is V210:

char tempBuf[4];

for (int j = 0; j < frameItem.VideoBufferSize/4; ++j) {

/\*\*

Word swapped and saved to temp buffer

\*/

tempBuf[0] = \*((char \*)frameItem.pBufferVideo + (j \* 4 + 3));

tempBuf[1] = \*((char \*)frameItem.pBufferVideo + (j \* 4 + 2));

tempBuf[2] = \*((char \*)frameItem.pBufferVideo + (j \* 4 + 1));

tempBuf[3] = \*((char \*)frameItem.pBufferVideo + (j \* 4 + 0));

/\*\*

Temporary buffer copied back

\*/

\*((char \*)frameItem.pBufferVideo + (j \* 4 + 0)) = tempBuf[0];

\*((char \*)frameItem.pBufferVideo + (j \* 4 + 1)) = tempBuf[1];

\*((char \*)frameItem.pBufferVideo + (j \* 4 + 2)) = tempBuf[2];

\*((char \*)frameItem.pBufferVideo + (j \* 4 + 3)) = tempBuf[3];

}

## Audio Processing

The audio data buffer returned by TyphoonChannel::GetFrame() is in a format with the following characteristics:

* There are 18 channels made up of 9 LR stereo pairs.
* Each audio sample is 24 bits, with each sample packed into a 32 bit word with the first 8 bits zero.
* The channel data is sequential in the buffer (as opposed to interleaved), with each channel’s data starting 2000 bytes offset from the previous channel’s data.
* Each channel block consists of a 32 bit size field specifying a value (n) followed by (n) *bytes* of single-channel audio data.

A transform algorithm to extract the first two channels as either 16 or 24 bit PCM audio can be found in the file [AudioTransform.h](https://github.com/Streampunk/styphoon/blob/master/src/AudioTransform.h). This is relatively complex, and so hasn’t been reproduced in this document.

## General Timing Issues

The Typhoon board seems sensitive to the timing of API calls. The sample code provided by CoreEl contains a number of Sleep() calls to pause between operations. Removing these calls caused instability, and so the [Styphoon](#styphoon) node.js binding implementation also makes use of a number of Sleep() calls during card initialization to prevent the card from causing system instability.

# Caveats / Troubleshooting

There are a number of caveats and things to be aware of when using the Typhoon board, as follows:

* Overheating: as mentioned in the installation section, board overheating can cause instability; so if you are experiencing regular system crashes it is worth checking the temperature of the board, using the Typhoon UI application, and making sure it isn’t going above 80 degrees Celsius.
* PCIe Bus Errors: the board can cause a host system halt due to PCIe bus errors. A frequent occurrence was identified as being caused by the boot-up of an SDI source connected to the board’s input: using a BlackMagic Video Assist to generate SDI video, the card would always cause its host PC to reboot if the Video Assist device was switched on *after* the PC. This was one deterministic cause of failure; however, anecdotally there appear to be a number of other less easily identified causes as well. However, most issues seem to occur around start-up and initialization. Once the card is working and streaming video it appears to be fairly stable.