FS4500 Data Extractor

**2017**

FuturePlus Systems

Company Confidential

6/7/2017

# Revisions

* 1. Rough Draft

# Introduction

The FS4500 product will have Data Extractor software that will allow the user access to the trace data where all of the states are stored.

This will allow the user to quickly access statedata, get the index of the trigger state, and get the number of states. A list of functions that the data extractor uses will be listed in one of the sections in this document.

The Data Extractor will extract data from states one state at a time.

# Data Extractor Functions

The functions that the data extractor uses are listed below:

public interface IProbeMgrGen2

{

event LogMsgEvent OnLogMsgEvent;

event ProbeCommEvent OnProbeCommEvent;

event TBUploadEvent OnTBUploadEvent;

void CloseProbe();

bool Configure(int deviceNum, string serialNumberStr, bool inDemoMode);

bool DisplayForm(string FormName);

string GetAssemblyVersion();

string GetFPGAVersion();

string GetLogMsgs();

int GetTriggerChannelID();

long GetTriggerStateIndex(int virtualChannelID);

long GetNumberOfStates(int virtualChannelID);

byte[] GetStateData(int virtualChannelID, int index);

string GetTitleString();

bool Initialize();

void MiscOperation(string title, object parameters = null);

bool ProcessTimerTick(RunTimeParameters runtimeParameters);

bool Run();

bool SaveConfiguration(string fileName, int selectedProtocolIndex);

bool SetDefaultConfiguration();

bool SetStoredConfiguration(string fileName, int selectedProtocolIndex);

bool ShutDown();

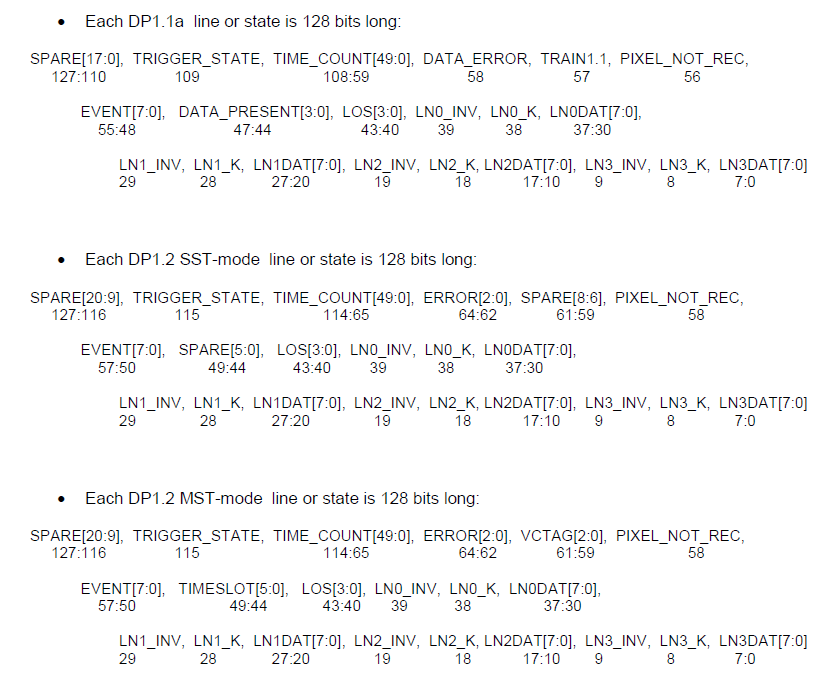
bool Stop();

bool Stopped();

}

# 4.0 Saved Data Field Format

The organization of data fields and their widths are shown in the picture below.



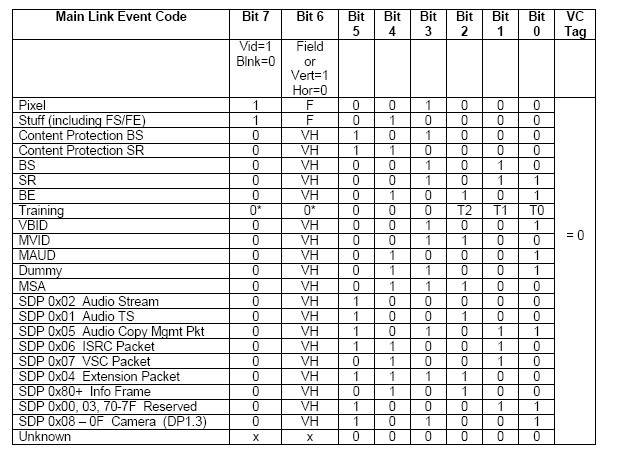
# 4.1 SST Field Definitions

The following chart contains all the fields in the SST format and a comment on what they do.

|  |  |  |
| --- | --- | --- |
| **Field** | **Location** | **Comment** |
| SPARE[20:9] | 127:116 | Spare bits unused |
| TRIGGER\_STATE | 115 | Indicates that trigger has occurred |
| Time\_Count[49:0] | 114:65 | Indicates number of states since the run began and can be read by the PM at any time. |
| ERROR[2:0] | 64:62 | Indicates if there is an error |
| SPARE[8:6] | 61:59 | Spare bits unused |
| PIXEL\_NOT\_REC | 58 | Pixel not Recognized |
| EVENT[7:0] | 57:50 | Event Code Decode see below |
| SPARE[5:0] | 49:44 | Spare bits unused |
| LOS[3:0] | 43:40 | Loss of Sync |
| LN0\_INV | 39 | Invalid, this is 1, there is an error |
| LN0\_K | 38 | Command, 1 = Command 0 = Data |
| LN0DAT[7:0] | 37:30 | Data found in Lane 0 |
| LN1\_INV | 29 | Invalid, this is 1, there is an error |
| LN1\_K | 28 | Command, 1 = Command 0 = Data |
| LN1DAT[7:0] | 27:20 | Data found in Lane 1 |
| LN2\_INV | 19 | Invalid, this is 1, there is an error |
| LN2\_K | 18 | Command, 1 = Command 0 = Data |
| LN2DAT[7:0] | 17:10 | Data found in Lane 2 |
| LN3\_INV | 9 | Invalid, this is 1, there is an error |
| LN3\_K | 8 | Command, 1 = Command 0 = Data |
| LN3DAT[7:0] | 7:0 | Data found in Lane 3 |

# 4.2 SST EventCodes

This chart will help the user identify an event code based on what the field value is.



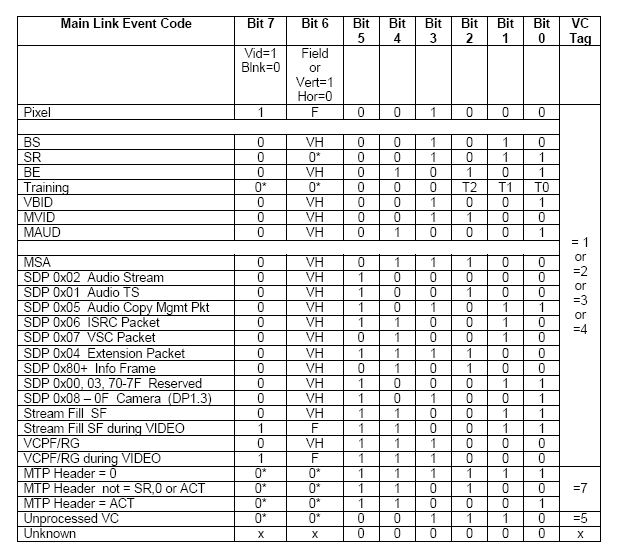
# 4.3 MST File Format

The following chart contains all the fields in the SST format and a comment on what they do.

|  |  |  |
| --- | --- | --- |
| **Field** | **Location** | **Comment** |
| SPARE[20:9] | 127:116 | Spare bits unused |
| TRIGGER\_STATE | 115 | Indicated that the trigger has occurred |
| TIME\_COUNT[49:0] | 114:65 | Indicates number of states since the run began and can be read by the PM at any time. |
| ERROR[2:0] | 64:62 | Indicated if there was an error |
| VCTAG[2:0] | 61:59 | Virtual Channel Tag |
| PIXEL\_NOT\_REC | 58 | Pixel not Recognized |
| EVENT[7:0] | 57:50 | Event Codes see decode below |
| TIMESLOT[5:0] | 49:44 | Time allocated to a virtual channel in MST mode |
| LOS[3:0] | 43:40 | Loss of Sync |
| LN0\_INV | 39 | Invalid, this is 1, there is an error |
| LN0\_K | 38 | Command, 1 = Command 0 = Data |
| LN0DAT[7:0] | 37:30 | Data in Lane 0 |
| LN1\_INV | 29 | Invalid, this is 1, there is an error |
| LN1\_K | 28 | Command, 1 = Command 0 = Data |
| LN1DAT[7:0] | 27:20 | Data in Lane 1 |
| LN2\_INV | 19 | Invalid, this is 1, there is an error |
| LN2\_K | 18 | Command, 1 = Command 0 = Data |
| LN2DAT[7:0] | 17:10 | Data in Lane 2 |
| LN3INV | 9 | Invalid, this is 1, there is an error |
| LN3\_K | 8 | Command, 1 = Command 0 = Data |
| LN3DAT[7:0] | 7:0 | Data in Lane 3 |

# 4.4 MST EventCodes

This chart will help the user identify an event code based on what the field value is.



# 5.0 Example

To use the Data Extractor, the user must include 5 .dll files in the project. These dll files will be on Github in a folder called “Common DLLs”.

In the project, the user must have the following variables at the top of the file, the following code is in C#.

using DP12MSTClassLibrary;

using DP14MSTClassLibrary;

using DP12SSTClassLibrary;

using DP14SSTClassLibrary;

using FPSProbeMgr\_Gen2;

private DP12SST m\_DP12SSTProbe = null;

private DP12MST m\_DP12MSTProbe = null;

private DP14SST m\_DP14SSTProbe = null;

private DP14MST m\_DP14MSTProbe = null;

IProbeMgrGen2 m\_IProbe = null;

The DP12SST, DP12MST, DP14SST, DP14MST, and IProbeMgrGen2 are references dll files. The m\_IProbe will be set to one of the above variables depending on the version of the Probe Manager. This function must be used to set the m\_IProbe to the correct version.

private bool createInterfaceObject()

{

string protocol = getprotocol(); <- user can create this function

bool status = true;

switch (protocol)

{

case "SST12":

if (m\_DP12SSTProbe != null)

m\_DP12SSTProbe = null;

m\_DP12SSTProbe = new DP12SST();

m\_IProbe = (IProbeMgrGen2)m\_DP12SSTProbe;

break;

case "MST12":

if (m\_DP12MSTProbe != null)

m\_DP12MSTProbe = null;

m\_DP12MSTProbe = new DP12MST();

m\_IProbe = (IProbeMgrGen2)m\_DP12MSTProbe;

break;

case "SST14":

if (m\_DP14SSTProbe != null)

m\_DP14SSTProbe = null;

m\_DP14SSTProbe = new DP14SST();

m\_IProbe = (IProbeMgrGen2)m\_DP14SSTProbe;

break;

case "MST14":

if (m\_DP14MSTProbe != null)

m\_DP14MSTProbe = null;

m\_DP14MSTProbe = new DP14MST();

m\_IProbe = (IProbeMgrGen2)m\_DP14MSTProbe;

break;

}

return status;

}

After the createInterfaceObject() has been called, the user must use the following function, m\_IProbe.Initialize();

The user can now use any function shown earlier in section 3.0 of this document. For example, if the user wants the state data, the user would type this function,

byte[] examplelist = m\_IProbe(GetStateData(virtualchannel, index);

# 6.0 Summary

The Data Extractor allows the user to extract the trace buffer data and use functions to return state data of interest to the user. The Data Extractor will be on the FuturePlus Github page and the link is below.