PVRScope

User Manual

Filename	:	PVRScope.User Manual.doc
Version	:	1.0.5 External Issue External Developer.
Issue Date	:	30 Aug 2012
Author	:	Imagination Technologies Ltd

Contents

List of Figures

No table of figures entries found.

1. Introduction

1.1. Document Overview

The purpose of this document is to serve as a reference for the PVRScope library, as well as to provide a number of examples of its use.

1.2. Library Overview

PVRScope is a utility library that can be used to access the hardware performance counters of PowerVR SGX hardware via a driver library called PVRScopeServices. It also allows an application to send user defined information to PVRTune via PVRPerfServer, both as counters and marks, or as editable data that can be passed back to the application.

The following files are provided:

- `'PVRScope.h'` – The header file defining the PVRScope libraries functionality.
- `'PVRScopeDeveloper.lib'` – The PVRScope library file.

1.2.1. Limitations

- Only one instance of PVRScope may communicate with PVRScopeServices at any given time. If a PVRScope enabled application attempts to communicate with PVRScopeServices at the same time as another such application, or at the same time as PVRPerfServer, conflicts can occur that may make performance data unreliable.
- Multiple PVRScope enabled applications may be active, and may send custom counters or marks to PVRTune, but only one may communicate with PVRScopeServices for the retrieval of hardware performance counter information at any given time.
- PVRPerfServer must be running on the host device if a PVRScope enabled application wishes to send custom counters or marks to PVRTune. If the application in question also wishes to communicate with PVRScopeServices without experiencing any undesired behaviour PVRPerfServer should be run with the `'--disable-hwperf'` flag.
- Hardware performance counters may only be read on devices whose drivers have PVRScopeServices active.
- Currently, only the basic types `'float'` and `'string'` may be transmitted as editable data to PVRTune.

2. Installation

PVRScope is available as part of the PowerVR Insider SDK which can be downloaded from the PowerVR Insider website.

2.1. From Installer

Download one of the PowerVR Insider SDKs and run the installer following the on screen instructions. Once the package has successfully installed, the library files will be available in the SDK folder at:

```
<InstallDir>\PVRScope\
```

2.2. Accessing the Library

To access the functionality of the library within an application the library must be linked to and the header files included at build time.

3. Example Code

PVRScope allows the use of one of two modes of functioning, either ‘Hardware Performance Counter Mode’ or ‘Custom Counter/Mark Mode’.

3.1. Hardware Performance Counter Mode

3.1.1. On Initialisation

```
#include "PVRScope.h"

// Create the PVRScope data storage area
SPVRScopeImplData* scopeData;

// Initialise PVRScope
const EPVRScopeInitCode returnCode = PVRScopeInitialise(scopeData);

// Check initialisation succeeded
if(returnCode != ePVRScopeInitCodeOk)
{
        // Handle the error
}

// Create an array of counter definitions
unsigned int numCounters = 0;
SPVRScopeCounterDef *counterArray = null;

// Create the data structure to contain the counter readings
SPVRScopeCounterReading counterReadings;

// Continue initialisation
if(scopeData)
{
        PVRScopeGetCounters(scopeData, &numCounters, &counterArray, &counterReadings);
}
```

3.1.2. As Required

```
// Select the desired counter group (3 in this instance)
unsigned int desiredGroup = 3;

// Store which group is currently active
unsigned int activeGroup = 0;

// Do a quick check to save some processing
if(desiredGroup == activeGroup)
        desiredGroup = 0xffffffff;

// Update the counters and store the active group
if(PVRScopeReadCountersThenSetGroup(scopeData, &counterReadings, desiredGroup))
        activeGroup = counterReadings.nReadingActiveGroup;

// Read the counters from the above group, and process them accordingly.
// The counters will match (in order) the elements in ‘counterArray’ whose group
// matches ‘counterReadings.nReadingActiveGroup’
```

3.1.3. On Shutdown

```
// De-initialise PVRScope
PVRScopeDeInitialise(&scopeData, &counterArray, &counterReadings);
```

3.2. Custom Counter/Mark Mode

It should be noted that PVRScope, when sending custom counters or marks does not connect to PVRScopeServices; as such, when in this operating mode, PVRScope will not send the standard performance counters to PVRTune. As no connection to PVRScopeServices is made it is possible to use PVRPerfServer to retrieve the counters; as an additional option 'Hardware Performance Counter Mode' may be used to retrieve the counters, which can then be sent manually to PVRTune by a PVRScope enabled application in the same manner as a custom counter.

3.2.1. On Initialisation

```
#include "PVRScope.h"

// Create the PVRScope data storage area
SSPSCommsData* commsData;

// Initialise PVRScopeComms
commsData = pplInitialise();

// Check the initialisation was successful
if(commsData)
{
        // Create Remotely Editable Items
        // Create Custom Counters
}
```

3.2.2. On Shutdown

```
// De-initialise PVRScopeComms
pplShutdown(commsData);
```

3.2.3. Sending Custom Marks

```
// Send a Custom Mark
std::string markName = "Custom Mark";
pplSendMark(commsData, markName.c_str(), markName.length());
```

3.2.4. Creating Custom Counters

```
// Create an array of custom counters (in this instance, 1 counter)
unsigned int counterListLength = 1;
SSPSCommsCounterDef counterList[counterListLength];

// Create the custom counter
std::string counterName = "Custom Counter";
counterList[0].pszName = counterName.c_str();
counterList[0].nLength = counterName.length();

// Submit the counter array to the comms library
if(pplCountersCreate(commsData, counterList, counterLength))
{
        // Handle error
}
```

3.2.5. Sending Custom Counter Updates

```
// The length of the counter list submitted to pplCountersCreate(...)
unsigned int counterListLength = 1;

// It is important that the counters be in the same order as they were
// when submitted to pplCountersCreate(...)

// Give each counter a new value if one is needed
unsigned int values[counterListLength];
for(unsigned int i = 0; i < counterListLength; i++)
        Values[i] = 0;

// Submit the updated counters for transmission
pplCountersUpdate(commsData, values);
```

3.2.6. Creating Remotely Editable Items

String

```
// Included for file handling
#include "OGLES2Tools.h"

// Create an array of remotely editable items (in this instance, 2 items)
SSPSCommsLibraryItem itemArray[2]

// Create a remotely editable string (in this example, a fragment shader from a file)
CPVRTResourceFile fragShaderFile("FragmentShaderFile.fsh");

// If the file is open, add it to the library
if(fragShaderFile.IsOpen())
{
        // Create the library item
        itemArray[0].pszName           = "FragShader";
        itemArray[0].nNameLength       = (unsigned int)strlen(itemArray[0].pszName);
        itemArray[0].eType             = eSPSCommsLibTypeString;
        itemArray[0].pData             = fragShaderFile.DataPtr();
        itemArray[0].nDataLength       = (unsigned int)fragShaderFile.Size();
}

// Create another remotely editable string (this time, just a string)
std::string value = "This is the title bar!";

// Create the library item
itemArray[1].pszName           = "Title Bar";
itemArray[1].nNameLength       = (unsigned int)strlen(itemArray[0].pszName);
itemArray[1].eType             = eSPSCommsLibTypeString;
itemArray[1].pData             = value.c_str();
itemArray[1].nDataLength       = value.length();

// Submit all library items
if(!pplLibraryCreate(commsData, itemArray, itemArrayLength))
{
        // Handle error
}
```

Float

```
// Create an array of remotely editable items (in this instance, 1 items)
SSPSCommsLibraryItem itemArray[1]

// Create the float
SSPSCommsLibraryTypeFloat value;
value.fCurrent = 0.5f;
value.fMin     = 0.0f;
value.fMax     = 1.0f;

// Create the library item
itemArray[0].pszName           = "Float";
itemArray[0].nNameLength       = (unsigned int)strlen(itemArray[0].pszName);
itemArray[0].eType             = eSPSCommsLibTypeFloat;
itemArray[0].pData             = (const char*)&value;
itemArray[0].nDataLength       = sizeof(value);

// Submit all library items
if(!pplLibraryCreate(commsData, itemArray, itemArrayLength))
{
        // Handle error
}
```

3.2.7. Retrieving Edited Items

```
// Update every item that has changed since we last checked
unsigned int itemNum, itemLength;
const char* data;
while(pplLibraryDirtyGetFirst(commsData, &itemNum, &itemLength, &data)
{
        // Update the item referred to by 'itemNum' with 'data'
}
```

4. Related Materials

Training Courses

- PVRScopeExample

Software

- PVRTune

Documentation

- PVRTune User Manual

5. Contact Details

For further support contact:

devtech@imgtec.com

PowerVR Developer Technology
Imagination Technologies Ltd.
Home Park Estate
Kings Langley
Herts, WD4 8LZ
United Kingdom

Tel: +44 (0) 1923 260511
Fax: +44 (0) 1923 277463

Alternatively, you can use the PowerVR Insider forums:

www.imgtec.com/forum

For more information about PowerVR or Imagination Technologies Ltd. visit our web pages at:

www.imgtec.com