AMD AGESA Arch2008 Debug Console *(HDTOUT)* Specification

Publication # 00000 Revision: **0.1**
Issue Date: **Jan. 15, 2011**

Revision History

Date	Revision	Description
1/1/2011	0.1	Initial Draft

References

Document Title	Version	PID
BIOS and Kernel Developer's Guides (BKDG)	1.0.0	31116
AGESA Interface Specification	5.0.1	44065
AMD64 HDT Scripting User Manual		

Contents

Contents

Chapter 1 Introduction

AGESA Arch2008 supports a Debug Console module, HDTOUT, as part of its Integrated Debug Support (IDS). HDTOUT is a protocol that allows AGESA to stream out and to log its debug data. Besides that main goal, HDTOUT also provides an open interface to allow external scripts to control and make modifications to AGESA code flow.

1.1 Goals

The goal of this document is to provide user-level description of how to enable and retrieve debug data from AGESA, as well as to write scripts to control AGESA code flow.

1.2 Audience

This document is directed to BIOS developers or system engineers, who want to look at AGESA initialization sequence to debug BIOS or hardware issues.

1.3 Related Documents

Following is a list of related AMD documents:

- BIOS and Kernel Developer's Guides (BKDG)
  - *BIOS and Kernel Developer's Guide for AMD Family 10h Processors*, order# 31116
- AGESA Interface Specification, order# 44065
- AMD64 HDT Scripting User Manual

Chapter 2 Quick User Guide

This section is to provide basic users a short description of how to retrieve AGESA debug data. Additional information and release notes can be found in *Readme.txt* that comes with *hdtout2008.pl* script.

2.1 Requirements:

- Active Perl 5.10.0 installed.
- HDT device *(Purple Possum or Wombat)* is connected *(MPHDT should also be connected if available).*
- HDT application with Perl support installed.
- BIOS built with HDTOUT enabled:
  - Ensure $(AGESA_OptsDir)\OptionsIds.h exists
  - IDSOPT_IDS_ENABLED is set to TRUE
  - IDSOPT_TRACING_ENABLED is set to TRUE.
- HDTOUT scripts come with AGESA package:
  `AGESA\AMDTools\HDTOUT`
- Debug register DR2 and DR3 must be reserved for HDTOUT.

*** We strongly recommend users to use the scripts that come with AGESA release that the BIOS is built with.**

2.2 Usage

- The HDTOUT BIOS will boots normally like regular BIOS.
- Make sure all above requirements are satisfied.
- From the host machine, run EnableHdtoutAtReset.pl script to enable HDTOUT.
  Note*: This script is deprecated since hdtout2008.pl rev.1.3.0 (02/11/2010). If your AGESA package has hdtout2008.pl rev.1.3.0 or later, you can skip this step. Otherwise you must take this step.*
- From the host machine, run hdtout2008.pl script in the command prompt as follow:
  > perl hdtout2008.pl
- If the script asks for reset to enable HDTOUT, type ‘y’

Note: If for some reason, scripts cannot enable HDTOUT on the fly, please refer to section 3.3 to enable HDTOUT in BIOS.

Guide to use HDTOUT for S3 resume, when HDTOUT isn't enabled by build or setup options:
- Open HDT application and enable "HDT on CPU reset"
- Power on target, make sure it been trap on reset state
- Run hdtout2008.pl
- When script asks to enable with reset, select "N" to let it enable without reset.

2.3 Command Line Options

We have introduced command line options since hdtout2008.pl rev 2.0.0. If you have older script, please reference next section, Inline Controls.

```
USAGE: perl hdtout2008.pl  [-help] [-possum number] [-buffersize size]
  [-YAAP [ip=ipaddress] [username=name] [password=password]]
  [-logfile FILE] [-filter Filters] [-spd] [-wp]
```

Tips: All parameter is case-insensitive. Use first letter instead of whole word.

`[-logfile FILE]`	Specifies the logfile name and path
`[-YAAP [ip=ipaddress] [username=name] [password=password]]`	Specifies YAAP configuration. E.g.: IP=192.168.0.1 define the IP address as 192.168.0.1 username=root define the username as "root" password=111111 define the password as "111111"
`[-possum number]`	Specifies HDT device number
`[-buffersize size]`	Specifies size of print buffer. Buffer size is 4kB by default. The bigger the buffer, the faster script runs. However, to debug system hard lock, buffer size of 0 is recommended.
`[-spd]`	Select to dump SPD data
`[-reset]`	When specified, the script resets system to enable HDTOUT without asking
`[-wp]`	Specify the path to AGESA source code. When Assert occurs, the script will get more detail information rather only Filecode

`[-filter filters]`	Define the filter of the output string. Supported filters:
	`MAIN_FLOW S3 EVENT_LOG PEFORMANCE`
	`GNB_TRACE GNB_PCIE_MISC GNB_PORTREG GNB_HOSTREG`
	`GNB_NB_MISC GNB_GFX_MISC GNB_SMU`
	`MEM_FLOW MEM_GETREG MEM_SETREG MEM_STATUS`
	`CPU_TRACE HT_TRACE FCH_TRACE IDS_TRACE`
	`IDS_REG`

2.4 Inline Controls

Before hdtout2008.pl rev. 2.0.0, we allow users to customize HDTOUT by modifying hdtout2008.pl script itself. Below is the list of variables that users can modify:

`$dft_emulator_num`	Specifies HDT device number

`$dft_ip_addr`
`$default_password`
`$default_username`
	Specifies YAAP configuration. E.g.:
	IP=192.168.0.1 define the IP address as 192.168.0.1
	username=root define the username as "root"
	password=111111 define the password as "111111"

`$dft_buffer_size`	Specifies size of print buffer. Buffer size is 4kB by default. The bigger the buffer, the faster script runs. However, to debug system hard lock, buffer size of 0 is recommended.
`$dump_spd_data`	Select to dump SPD data
`$force_reset_to_enable`	When specified, the script resets system to enable HDTOUT without asking
`$bsp_only`	0: HDTOUT will be streamed out from all enabled cores 1: Only enable HDTOUT on BSP
`$show_pci_reg_set` `$show_pci_reg_get`	Stream out PCI register set during memory init Stream out PCI register get during memory init
`$reboot_test`	When enabled, reboot whenever the single string in HdtoutSelectTable is hit
`$use_cold_reset`	Note: some chipsets do not support this type of cold reset
`$use_relay_to_reset`	Must have relay cables connected

Besides specifying filter in command line, users can also modify the below table in hdtout2008.pl script to customize or to speed up stream-out data

```
my $HdtoutDefault = 1;              # 0: Skip HDTOUT from the beginning
                                    # 1: Enable HDTOUT from the beginning
my $HdtoutSelectTableEnable = 1;    # 0: Disable HdtoutSelectTable
                                    # 1: Enable HdtoutSelectTable
my $HdtoutSelectTable = [
    # Below are some examples of how this table is used:
    # - Only Stream out WL sequence of Node 1, Dct 0
    #     [["Start write leveling","!Node 1","!Dct 0"],["End write
leveling","!Node 1","!Dct 0"]]
    # - Stop (or reset) at the end of DRAM training
    #     [["AmdInitPost: End"]]
];
```

The '!' was used before "Node" and "Dct" conditions to indicate that they are secondary conditions, which does not occur at the same time as the primary condition "Start write leveling" but sometimes earlier. HDTOUT engine does not match "Node" and "Dct" to the current HDTOUT entry, but match them to HDTOUT Status String, which will be described in section 4.3.4.

2.5 Troubleshooting

Below is the list of commonly seen problems with HDTOUT and their solutions:

2.5.1 HDTOUT script dies at some line

1. Make sure HDT cables are connected.
2. Run HDT to make sure HDT firmware is up-to-date.
3. Make sure system is not hard locked. When system is hard-locked, users need to trap before the hard-lock point manually before running HDTOUT.
4. If all of the above has been checked but HDTOUT script still dies at reading some register, the enable-at-reset feature may not work with the system's chipset. Please see section 3.3 and enable HDTOUT in the BIOS instead.

2.5.2 HDTOUT data is corrupted

1. Make sure MP cable is connected if it is an MP system
2. Try changing HDT cables. Sometimes, a damaged cable can cause HDTOUT data corruption.

2.5.3 HDTOUT stops in the middle of printing data

1. Try pressing Ctrl+C to retrieve the rest of the data. If it works, system is soft-locked, probably due to a long wait, jmp $, or dead loop.
2. If pressing Ctrl+C cannot stop the script, system is hard-locked. Users need to press Ctrl+Break to end the script. Note that, HDTOUT cannot retrieve the data left in buffer when system is hard-locked. Hence, buffer size must be set to 0 to get as much data as possible in this case. After pressing Ctrl+Break, HDT may be locked in trap-at-reset, please see next section for solution.

2.5.4 System cannot be reset or stuck at some POST code

1. Try opening HDT GUI, and clicking on "HDT On CPU Reset" button a couple of times until it is un-checked, and reset the system.
2. If that does not work, try reset HDT as described in next section.

2.5.5 Conflict due to using HDT while HDTOUT script is running

1. Close HDT Application if opened.
2. Close command line window that runs HDTOUT script.
3. Power off the test system.
4. Unplug the cable that connected to the HDT device to power it off completely.
5. Power on the system.
6. Plug HDT device back in.
7. Rerun HDT or HDTOUT script.

Chapter 3 Developer Guide

This section is directed to AGESA and BIOS developers to describe how to build in HDTOUT support, as well as how to add HDTOUT entries.

3.1 Build In HDTOUT Support

HDTOUT is part of AGESA Integrated Debug Support, IDS. Hence, IDS must be enabled for HDTOUT to be pulled in to the BIOS build. Below is the checklist to build in HDTOUT support:

- Ensure OptionsIds.h exists in the path defined by $(AGESA_OptsDir). AGESA reference $(AGESA_OptsDir)\OptionsIds.h to enable HDTOUT. If OptionsIds.h does not exist, AGESA will reference the default AGESA\Proc\IDS\OptionsIds.h instead.
- Define IDSOPT_IDS_ENABLED to TRUE in OptionsIds.h.
- Define IDSOPT_TRACING_ENABLED to TRUE in OptionsIds.h.

3.2 Add New HDTOUT Entry

For MarG34PI and DanniPI packages, HDTOUT only covers memory initialization.
For other packages, HDTOUT covers from AmdInitReset to AmdInitLate.

If the code is anywhere in the HDTOUT coverage, one can add the printf like macro, CONSOLE, like below to output his or her debug data:

```
CONSOLE ("1 + 1 = %d\n", 1 + 1);
```

CONSOLE macro is for debug purpose only. If a debug macro is needed to be checked in to AGESA, IDS_HDT_CONSOLE macro should be used. IDS_HDT_CONSOLE macro requires a Flag argument, which is used to classify the output data. Below is the example:

```
IDS_HDT_CONSOLE (MAIN_FLOW, "1 + 1 = %d\n", 1 + 1);
```

3.3 Enable HDTOUT in BIOS

Sometimes, due to different behaviors of different chipset, hdtout2008.pl fails to enable on the fly. In these cases, we need special BIOS build to enable HDTOUT.

To enable HDTOUT, BIOS needs to set DR2 register to 0x99CC before AGESA AmdInitReset. Note that only DR2 registers need to be set. It is recommended to not enable the DR2 breakpoint in DR7. BIOS developers may also create setup options to control HDTOUT in runtime.

3.4 Running HDTOUT on SimNow:

To run HDTOUT on SimNow, the script needs to be modified as follow:

- The HDTOUT BIOS will boots normally like regular BIOS.
- Modify hdtout2008.pl script as follow:
  - Uncomment (remove the leading '#' from) the block:

```
# $IsSimNow = 1;
# use Win32::OLE;
# use Win32::OLE::Variant;
# $cmd = Win32::OLE->new('SimNow.Command');
# $MyResponse = Variant(VT_BSTR | VT_BYREF, "");
```

  - Comment out (adding '#' to the beginning of) the line:

```
use hdtPerl;
```

- Reset SimNow simulation
- After that, run hdtout2008.pl script to retrieve data from BIOS.

** Note: On SimNow, hdtout2008.pl needs to be run at simulation reset.*
Platform BIOS needs to write 99CC to DR2 before AmdInitReset.

3.5 Stream Data with Other Transport Method

Even though AGESA does not officially support different transport methods other than HDT yet, BIOS developers can still redefine AGESA `HDT_CONSOLE` macros to invoke their own transport procedures. Below is the list of main macros that can be redefined in AGESA\Include\Ids.h:

`IDS_HDT_CONSOLE_INIT (AMD_CONFIG_PARAMS *StdHeader)`

This macro is called to initialize HDT protocol and data structure. Currently, it is called at the beginning of these three AGESA entries:

1) AmdInitReset
2) AmdInitEnv
3) AmdS3LateRestore

`IDS_HDT_CONSOLE_EXIT (AMD_CONFIG_PARAMS *StdHeader)`

This macro is called to stop HDTOUT stream and deallocate HDTOUT data.

`IDS_HDT_CONSOLE (UINT64 Flag, CHAR8 *FormatStr, ...)`

This macro has printf like format to stream data out with an addition of 'Flag' variable, which is used to classify and select output data.

Other macros listed below should be defined as empty function if not used in the non-HDT transport method:

```
IDS_HDT_CONSOLE_S3_EXIT (AMD_CONFIG_PARAMS *StdHeader)
IDS_HDT_CONSOLE_S3_AP_EXIT (AMD_CONFIG_PARAMS *StdHeader)
IDS_HDT_CONSOLE_FLUSH_BUFFER (AMD_CONFIG_PARAMS *StdHeader)
IDS_HDT_CONSOLE_ASSERT (AMD_CONFIG_PARAMS *StdHeader)
CONSOLE (UINT64 Flag, CHAR8 *FormatStr, ...)
```

Chapter 4 HDTOUT Architecture

4.1 Overview

HDTOUT is a client-server protocol that allows AGESA to transmit its debug data to HDT script for displaying. HDT serves as the transport layer connecting HDT script and AGESA. HDTOUT script works as a print server, which waits and displays any data that BIOS requests. AGESA acts like a print client. When AGESA needs to stream data out, it leaves data in a buffer, then triggers a dummy IO breakpoint (unused port 99CC) to halt itself in HDT mode. HDTOUT script detects the breakpoint and pulls data out from the buffer, then exits HDT mode to allow BIOS to continue booting.

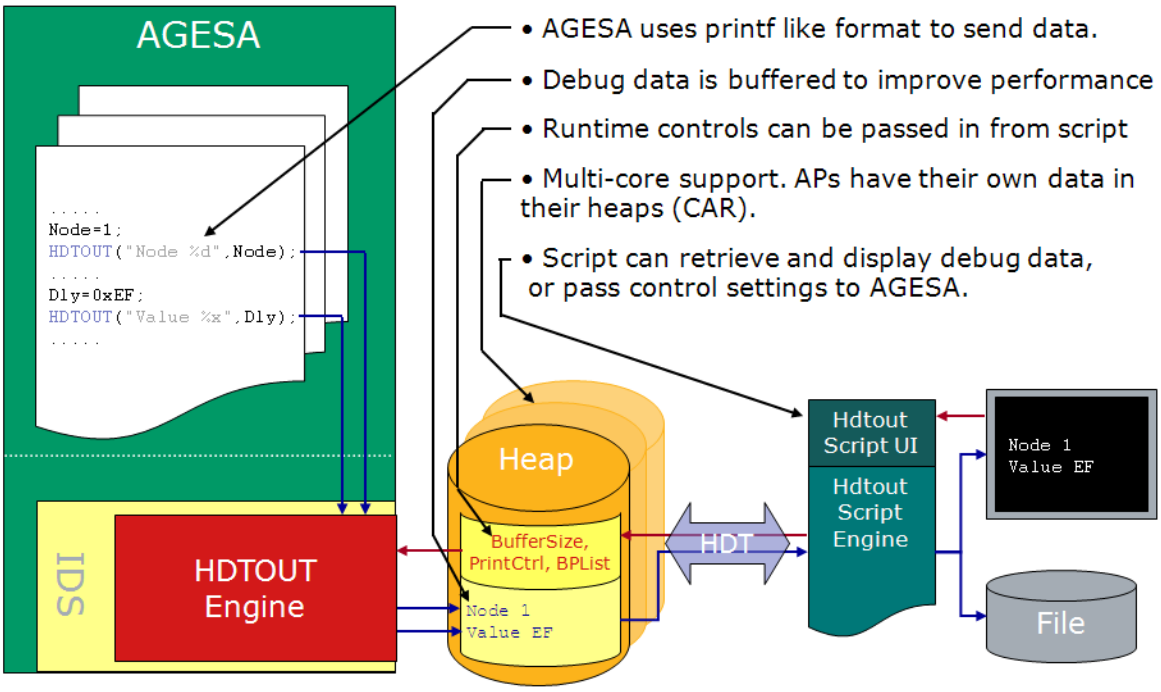

***Fig. 1**: HDTOUT block diagram*

4.2 HDTOUT Engine

HDTOUT engine is part of AGESA's Integrated Debug Support, or IDS. Like IDS, it is removed from the AGESA build by default. When it is selected to be built in and enabled (see section 3.1), HDTOUT macros will be assigned to HDTOUT handlers. Otherwise, HDTOUT macros will be defined as nothing and will take zero code size in ROM. There are three main HDTOUT macros:

```
IDS_HDT_CONSOLE_INIT (AMD_CONFIG_PARAMS *StdHeader);
IDS_HDT_CONSOLE_EXIT (AMD_CONFIG_PARAMS *StdHeader);
IDS_HDT_CONSOLE (UINT64 Flag, CHAR8 *FormatStr, ...);
```

4.2.1 HDTOUT Initializer

IDS_HDT_CONSOLE_INIT macro is the HDTOUT engine initializer. Once enabled, it initializes itself at AGESA's first entry call, AmdInitReset. The initialization includes the following steps:

1. Allocating heap buffer for HDTOUT header and its data.
2. Setting up IO breakpoint to dummy port 0x99CC. HDTOUT will use debug register DR2 to setup this breakpoint.
3. Trigger the very first 99CC breakpoint to start communication with HDTOUT script by executing:
   ```
   out dx, al
   ```
   with
   ```
   EDX = 10BF99CC
   ```
   and
   ```
   EAX = Address to HDTOUT header.
   ```
4. At this first transaction, HDTOUT engine expect HDTOUT script to make all necessary changes in HDTOUT header to customize HDTOUT engine's behavior. Please see section 5.1.1 for HDTOUT header structure.
5. After HDTOUT header is successfully initialized, the address to HDTOUT header will be written to debug register DR3. This is to allow HDTOUT script to retrieve HDTOUT data anytime after this point.

4.2.2 HDTOUT Finalizer

IDS_HDT_CONSOLE_EXIT macro is the HDTOUT finalizer. It does the followings:

1. Trigger 99CC breakpoint with EDX=E0BF99CC to signal HDTOUT script to exit.
2. Disable the 99CC breakpoint.
3. Deallocate HDTOUT data.

4.2.3 HDTOUT Stream-out Function

After IDS_HDT_CONSOLE_INIT and before IDS_HDT_CONSOLE_EXIT, IDS_HDT_CONSOLE macro can be called anytime to stream out debug data. It has printf like format, which can take variable number of arguments and integrate them into a single string for display. Starting HDTOUT version 2.0.0, we had added to IDS_HDT_CONSOLE macro one more argument, Flag. The Flag argument is a 64-bit bitmap that is used by HDTOUT engine to classify and to filter debug data. Below is the processing sequence of each IDS_HDT_CONSOLE entry:

1. Check Flag to see if it is selected to be streamed out in HdtoutHeader.ConsoleFilter. If it is, continue. Otherwise, exit.
2. Resolve all optional arguments and integrate them in the format string
3. Put the resolved string in a buffer.
4. Trigger 99CC breakpoint to allow script to pull the data:

```
out dx, al
EDX = C0BF99CC
EAX = Address to HDTOUT data
EBX = Data length
```

4.3 Features

4.3.1 Runtime Enablement

To allow HDTOUT to be enabled easily in runtime, HDTOUT uses debug register DR2 as the enablement flag rather than using CMOS or other BIOS's NV storage. Before processing any HDTOUT macros, HDTOUT engine ensures DR2 is 0x99CC first. This way, script can trap at reset and set DR2=0x99CC to enable HDTOUT. Platform BIOS can also set DR2 the same way to enable HDTOUT with setup option.

4.3.2 Direct and Buffer Mode

In direct mode (no buffering), HDTOUT triggers 99CC breakpoint and sends data out at every IDS_HDT_CONSOLE entry. However, due to slow response when 99CC breakpoint is trigger, streaming in direct mode is extremely slow. Because of that, HDTOUT uses a 4 kB buffer by default to speed up HDTOUT streaming. With buffering, HDTOUT triggers 99CC breakpoint only when the buffer is full. The drawback of buffer mode is that a portion of debug data cannot be retrieved once system is hard locked. In that case, direct mode is recommended. Please refer to section 2.3 to see how to enable/disable HDTOUT buffer.

4.3.3 HDTOUT Breakpoint

HDTOUT breakpoint is one of the most useful HDTOUT features. It allows users to stop in the middle of AGESA code flow to read registers or override AGESA settings. Basically, any strings or substrings that are seen in HDTOUT output can be used as breakpoints. Users can add breakpoint strings by using $HdtoutSelectTable as described in section 2.4. Scripts can add breakpoint strings directly to HDTOUT header during HDTOUT init as described in section 5.1.3. These breakpoint strings will be compared to the output string at every HDTOUT entry. When any of the breakpoint strings matches, HDTOUT will trigger 99CC breakpoint as follow:

```
out dx, al
EDX = B0BF99CC
EAX = Address to HDTOUT data
EBX[15:0] = Data length
EBX[31:16] = BP string that just matched
```

When setting breakpoint strings that repeat multiple times in HDTOUT output (e.g. "MR1" commands are sent per chip select, per channel, and per node), HDTOUT engine will also stops multiple times. To allow users to track where the breakpoint string is, we also provide Status String, which is described in the next section.

4.3.4 Status String

Status string is a single NULL-ended string that contains tracking data separated by '\n' character. Scripts can parse HDTOUT status string to track which node or which DCT that HDTOUT breakpoint has triggered. Status string can also be used as a secondary condition for HDTOUT breakpoint. Section 2.4 explains how to use status string to setup breakpoints.

The algorithm to compose status string is simple. HDTOUT uses the first word of the HDTOUT entry that has Flag=MEM_STATUS as the keyword to search and replace the status string entry that has the same keyword.

For example, at one point, status string is "Node 0\nDct 1\nStart write leveling"

When HDTOUT gets to IDS_HDT_CONSOLE (MEM_STATUS, "Start RxEn training"), the status string will become "Node 0\nDct 1\nStart RxEn training"

4.3.5 Invoke BIOS Functions

HDTOUT module also provides an interface to couple of BIOS functions that users can invoke via scripts. Currently, only three functions are supported:

1. Write cache lines
2. Read cache lines
3. Flush cache lines

These functions allow users to generate the same memory traffic as BIOS does, which cannot be done with HDT alone because HDT generates read-modify-write disruptive traffic.

However, this feature is deprecated for processors that later than Family 10h because they have their own engine to generate continuous DRAM traffic.

Chapter 5 Scripting Interface

This section is directed to script developers, who want to make use of HDTOUT breakpoint features to control or modify AGESA behaviors. This section only describes the accessible APIs provided by AGESA to scripts. Please refer to Chapter 3 to know how HDTOUT works.

5.1.1 HDTOUT Data Structure

To maintain script and HDTOUT engine communication, HDTOUT has a fixed data structure as follow:

```
typedef struct _HDTOUT_DATA {
  HDTOUT_HEADER HdtoutHeader; ///< 56 byte fixed size
  CHAR8  BreakpointList[300]; ///< Breakpoint list
  CHAR8  StatusStr[156];      ///< Status string
  CHAR8  Data[2];             ///< HDTOUT content. Its size will
                              ///< be determined by BufferSize.
} HDTOUT_DATA;

typedef struct _HDTOUT_HEADER {
  UINT32 Signature;          ///< 0xDB1099CC
  UINT16 Version;            ///< HDTOUT version.
  UINT16 BufferSize;         ///< Size in bytes.
  UINT16 DataIndex;          ///< Data Index.
  UINT8  PrintCtrl;          ///< 0 off no print
                             ///< 1 on print
  UINT8  NumBreakpointUnit;  ///< Number of BP strings
  UINT32 FuncListAddr;       ///< 32 bit address to the list of
                             ///< functions that script can
                             ///< execute
  UINT8  ConsoleType;        ///< Deprecated
  UINT8  Event;              ///< Deprecated
  UINT8  OutBufferMode;      ///< Deprecated
  UINT32 EnableMask;         ///< Deprecated
  UINT64 ConsoleFilter;      ///< Filter use to select which
                             ///< part should be streamed out
  UINT8  BspOnlyFlag;        ///< HDTOUT only on BSC
  UINT8  Reserved[56 - 32];  ///< Reserved for header expansion
} HDTOUT_HEADER;
```

At HDTOUT initialization (EDX=10BF99CC), EAX points to HDTOUT_DATA for script to modify entire structure. However, at later HDTOUT trigger (EDX=C0BF99CC), EAX points to HDTOUT_DATA.Data directly; but DR3 points to HDTOUT_DATA.

5.1.2 Communication States

At each 99CC breakpoint, HDTOUT engine put a signature in the high 16 bits of EDX register to let scripts know the trigger state and take appropriate actions. Below is the list of states and their signatures:

```
#define DEBUG_PRINT_INIT        0x10BF0000
#define DEBUG_PRINT_ASSERT      0xA0BF0000
#define DEBUG_PRINT_EXIT        0xE0BF0000
#define DEBUG_PRINT_COMMAND     0xC0BF0000
#define DEBUG_PRINT_BREAKPOINT  0xB0BF0000
#define DEBUG_PRINT_ERROR       0x1EBF0000
```

5.1.3 HDTOUT Breakpoint Structure

It is strongly recommended that HDTOUT breakpoints to be setup at HDTOUT initialization (first 'out dx,al' breakpoint that has EDX=10BF99CC). To set breakpoint, HDTOUT_HEADER.NumBreakpointUnit takes the number of breakpoint strings. Then HDTOUT_DATA.BreakpointList will need to be populated as follow:

```
HDTOUT_DATA.BreakpointList[] = {
  BREAKPOINT_UNIT_1,
  BREAKPOINT_UNIT_2,
  ...
  BREAKPOINT_UNIT_n,
  BP_String_1,
  BP_String_2,
  ...
  BP_String_n
};

typedef struct _BREAKPOINT_UNIT {
  UINT8 AndFlag : 1;   ///< If 1, next string is ANDed to current string,
                       ///< breakpoint only triggers when all ANDed strings
                       ///< match.

  UINT8 BpFlag : 1;    ///< Format string or Status string
                       ///<  0: BP string will be matched with Data[]
                       ///<  1: BP string will be matched with StatusStr[]

  UINT8 Action : 4;    ///< Halt, start HDTOUT, or stop HDT,...

  ///<  Action[0]-Halt: trigger B0BF99CC
  ///<  Action[1]-Start: set HEADER.PrintCtrl to start recording debug data
  ///<  Action[2]-Stop: clear HEADER.PrintCtrl to stop recording debug data
  ///<  Action[3]-Reserved

  UINT8 BpStrOffset;   ///< Offset from BreakpointList to the BP string
} BREAKPOINT_UNIT;
```

When a group of BP strings that are tied together by BREAKPOINT_UNIT.AndFlag are all matched, an action selected by BREAKPOINT_UNIT.Action will be applied.

5.1.4 Function List Structure

HDTOUT_HEADER.FuncListAddr will be pointing to a structure like below when Function Invoke feature is supported.

```
CONST SCRIPT_FUNCTION ROMDATA ScriptFuncList[] = {
  {(UINT32) MemUWriteCachelines, "WriteCl(PhyAddrLo,BufferAddr,ClCnt)"},
  {(UINT32) MemUReadCachelines,  "ReadCl(BufferAddr,PhyAddrLo,ClCnt)"},
  {(UINT32) MemUFlushPattern,    "FlushCl(PhyAddrLo,ClCnt)"}
};
```

Each entry has two DWORDs and defines one function. The first DWORD is the 32bit physical address to the function. The second DWORD is the 32bit pointer to the NULL-ended string that describes the function and its arguments.

Please refer to FuncList.pm script to know how script can invoke these functions, pass arguments, and trap on return…