Institut für Datentechnik
und Kommunikationsnetze

Technische Universität
Braunschweig

GSEOS Driver
for
IDA Memory Tester

Revision 1.3

IDA-GSEOS-0008

August 2000

Prepared by
Kai Stoeckner

Tables and Figures

1 Scope

1.1 Purpose of this Document

Purpose of this document is to describe the GSEOS driver that is used for IDA memory tester. The driver is a Windows NT Kernel mode driver named **GseMTest.sys**.

1.2 Change Record

Table 1 Change Record

Date	Revision	Author	Affected Sections
1/13/2000	1.0	Stoeckner	All sections
8/1/2000	1.1	Stoeckner	1.4, 2, 3.5 - 3.12, 3.13
8/16/2000	1.2	Stoeckner	1.4, 2, 3.12, 3.13, 4
8/18/2000	1.3	Stoeckner	4

1.3 Reference Documentation

All GSEOS related documents are listed in the GSEOS Document Index IDA-GSEOS-0000.

1.4 Abbreviations

Table 2 Abbreviations

CUSIM	Camera Unit Simulator
DPPSIM	Digital Preprocessor Simulator
EGSE	Electrical Ground Support Equipment
GSEOS	Ground Support Equipment Operating System

HDRM	High Data Rate Module
NPD	Neutral Particle Detector
MGSE	Mechanical Ground Support Equipment
Q1355	Quad IEEE 1355
RTL	Runtime Library

2 Introduction

The GSEOS 4.0 is a program designed to run under Windows NT. Under Windows NT there is for a user program no way to access the hardware directly. The program needs a kernel mode driver. Only a kernel mode driver may serve interrupts and access hardware directly.

The driver for IDA memory tester projects is **GseMTest.sys**. The file **GseMTest.g** contains all necessary configuration for an easy access to all driver functions and generated blocks via the GSEOS language. This file should be included in the project main configuration file.

The driver is able to generate GSEOS blocks and to process GSEOS commands. The blocks and commands are described in the next chapters.

For IDA memory tester projects the driver supports several hardware units:

Table 3 Supported Hardware

H/W Name	Description
BTIF	A busmaster test I/F for the PCI bus. Used by IDA as test hardware for GSEOS and driver development and for performance measuring.
HDRM	A test I/F for High Data Rate Memory Modules.
NPD	A test I/F for Neutral Particle Detectors.
CUSIM	The Camera Unit Simulator for MSRS.
Q1355	An IEEE 1355 test I/F (4 channels).
DPPSIM	The Digital Preprocessor Simulator for MSRS.

3 Driver Generated Blocks

This section describes the various driver-generated blocks. The user should not change the names and the structure of the blocks. The usual way to use a driver-generated block is to write a decoder attached to the block.

3.1 _SelfStimulationTicker

Description The driver sends the block **_SelfStimulationTicker** if the self-stimulation of the driver is enabled. The user may enable/disable the self-stimulation and change the interval in the Control Panel for the H/W driver in the GSEOS. The block may be used for debugging purposes.

Declaration

```
typedef struct
{
  ULONG ul;
} tsLong;
```

Definition

```
blockdef tsLong tblkSelfStimulationTicker _SelfStimulationTicker;
```

Example

```
decode on (_SelfStimulationTicker)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulSelfStimulationTickerDecoded =
                                    _SelfStimulationTicker.ul * 2;

  send (TestNew);
}
```

3.2 _BTIF

Description The driver sends the BTIF (Busmaster Test I/F, PCI) generated data in the block **_BTIF**.

Declaration

```
typedef union
{
  BITMAP 32 [128, 256] bmp;     // picture like [x, y]
  ULONG  aulRaw[256][128];      // C like [y][x]
} tsBTIF;
```

Definition

```
blockdef tsBTIF tblkBTIF _BTIF;
```

Example

```
decode on (_BTIF)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulBtifCnt = Test.ulBtifCnt + 1;

  send (TestNew);
}
```

3.3 _HdrmStatus

Description The driver sends after the command HdrmSendStatus the HDRM status in the block **_HdrmStatus**.

Declaration

```
typedef struct
{
  ULONG aulRegFpga[6];
  ULONG ulPLLFrequency[2];
} tsHdrmStatus;
```

Definition

```
blockdef tsHdrmStatus tblkHdrmStatus _HdrmStatus;
```

Example

```
decode on (_HdrmStatus)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulHdrmStatusCnt = Test.ulHdrmStatusCnt + 1;

  send (TestNew);
}
```

3.4 _Hdrm

Description The driver sends received data on the HDRM I/Fs in the block **_Hdrm**.

Declaration

```
typedef struct
{
  ULONG ulChannel;                  // 0-channel A, 1 – channel B
  ULONG ulLength;                   // in bytes
  UCHAR auc[32 * 1024 * sizeof (ULONG)];
} tsHdrm;
```

Definition

```
blockdef tsHdrm tblkHdrm _Hdrm
```

Example

```
decode on (_Hdrm)
{
  switch (_Hdrm.ulChannel)
  {
    case 0:
      // channel A
      ProcessA (_Hdrm.auc, _Hdrm.ulLength);
      break;

    case 1:
      // channel B
      ProcessB (_Hdrm.auc, _Hdrm.ulLength);
      break;

    default:
      // error: invalid channel
      break;
  }
}
```

3.5 _NpdStatus

Description The driver sends after the command NpdSendStatus the NPD status in the block **_NpdStatus**.

Declaration

```
typedef struct
{
  ULONG aulRegFpga[19];
} tsNpdStatus;
```

Definition

```
blockdef tsNpdStatus tblkNpdStatus _NpdStatus;
```

Example

```
decode on (_NpdStatus)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulNpdStatusCnt = Test.ulNpdStatusCnt + 1;

  send (TestNew);
}
```

3.6 _NpdAdc

Description The driver sends ADC data in the block **_NpdAdc**.

Declaration

```
typedef struct
{
  UCHAR auc[8];
} tsNpdAdc;
```

Definition

```
blockdef tsNpdAdc tblkAdc _NpdAdc;
```

Example

```
decode on (_NpdAdc)
{
  // process NPD ADC data
}
```

3.7 _NpdTof

Description The driver sends TOF data in the block **_NpdTof**.

Declaration

```
typedef struct
{
  ULONG ulLen;
  ULONG aul[1024];
} tsNpdTof;
```

Definition

```
blockdef tsNpdTof tblkTof _NpdTof;
```

Example

```
decode on (_NpdTof)
{
  // process NPD TOF data
}
```

3.8 _NpdRaw

Description The driver sends RAW data in the block **_NpdRaw**.

Declaration

```
typedef struct
{
  ULONG ulLen;
  ULONG aul[1024];
} tsNpdRaw;
```

Definition

```
blockdef tsNpdRaw tblkRaw _NpdRaw;
```

Example

```
decode on (_NpdRaw)
{
  // process NPD RAW data
}
```

3.9 _CusimStatus

Description The driver sends after the command CusimSendStatus the CUSIM status in the block **_CusimStatus**.

Declaration

```
typedef struct
{
  ULONG ulHwId;
  ULONG aulRegFpga[6];
  ULONG ulPLLFrequency;
  ULONG ulMode;
} tsCusimStatus;
```

Definition

```
blockdef tsCusimStatus tblkCusimStatus _CusimStatus;
```

Example

```
decode on (_CusimStatus)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulCusimStatusCnt = Test.ulCusimStatusCnt + 1;

  send (TestNew);
}
```

3.10 _Q1355

Description The driver sends data received on the 1355 I/F in the block **_Q1355**.

Declaration

```
typedef struct
{
  ULONG ulHwId;                     // 0..1
  ULONG ulChannel;                  // 0..3
  ULONG ulEop;                      // EOP flags (bit0/1: EOP1/2)
  ULONG ulLength;                   // in bytes
  UCHAR auc[32 * 1024 * sizeof (ULONG)];
} tsQ1355;
```

Definition

```
blockdef tsQ1355 tblkQ1355 _Q1355;
```

Example

```
decode on (_Q1355)
{
  switch (_Q1355.ulChannel)
  {
    case 0: // process channel 0 data
            break;

    case 1: // process channel 1 data
            break;

    case 2: // process channel 2 data
            break;

    case 3: // process channel 3 data
            break;
  }
}
```

3.11 _Q1355Status

Description The driver sends after the command Q1355SendStatus the Q1355 status in the block **_Q1355Status**.

Declaration

```
typedef struct
{
  ULONG ulHwId;
  ULONG aulRegSmcs[4][256];
  ULONG aulRegFpga4[5];
  ULONG aulRegFpga5[5];
  ULONG aulRegFpga6[10];
} tsQ1355Status;
```

Definition

```
blockdef tsQ1355Status tblkQ1355Status _Q1355Status;
```

Example

```
decode on (_Q1355Status)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulQ1355StatusCnt = Test.ulQ1355StatusCnt + 1;

  send (TestNew);
}
```

3.12 _DppSim

Description The driver sends data received on the DPPSIM I/F in the block **_DppSim**.

Declaration

```
typedef struct
{
  ULONG  ulHwId;                                  // 0
  ULONG  ulChannel;                               // 0..1
  ULONG  ulSpare;                                 // spare
  ULONG  ulLength;                                // in bytes
  USHORT aus[32 * 1024 * sizeof (USHORT)];
} tsDppSim;
```

Definition

```
blockdef tsDppSim tblkDppSim _DppSim;
```

Example

```
decode on (_DppSim)
{
  switch (_DppSim.ulChannel)
  {
    case 0: // process channel 0 data
            break;

    case 1: // process channel 1 data
            break;
  }
}
```

3.13 _DppSimStatus

Description The driver sends after the command DppSimSendStatus the DPPSIM status in the block **_DppSimStatus**.

Declaration

```
typedef struct
{
  ULONG ulHwId;
  ULONG aulRegFpga0[5];
  ULONG aulRegFpga6[10];
  ULONG ulPLLFrequency;
} tsDppSimStatus;
```

Definition

```
blockdef tsDppSimStatus tblkDppSimStatus _DppSimStatus;
```

Example

```
decode on (_DppSimStatus)
{
  tblkTest TestNew;

  TestNew = Test;
  TestNew.ulDppSimStatusCnt = Test.ulDppSimStatusCnt + 1;

  send (TestNew);
}
```

4 Commands

The GSEOS uses command channels for all commanding operations. The full description of the command channel architecture is given in the document "GSEOS Language Runtime Library Description" (IDA-GSEOS-0004).

Table 4 GSEOS Command Channels

Destination	Value	Channel Identifier	Description	
Internal	`0x00ii0000`	`CMDCHANNEL_INTERNAL`	ii	tbd
Hardware	`0x01hhcc00`	`CMDCHANNEL_HARDWARE`	hh	H/W identifier, depends on the project specific H/W driver
			cc	Command identifier, depends on the project specific H/W driver.
Serial	`0x02ppww00`	`CMDCHANNEL_SERIAL`	pp	The serial port (COMx:) number.
Network	`0x03cc0000`	`CMDCHANNEL_NETWORK`	cc	The network connection number.
File	`0x04nn0000`	`CMDCHANNEL_FILE`	nn	The file number.

4.1 Command Overview

The driver is able to process following commands:

Table 5 Command Overview

H/W Identifier	Command Identifier	Command Name	Parameter Bytes	
0	0	SelfStimulation	[0]	0 – off 1 – on
0	1	HwStart	[0]	0 – off 1 – on
1	0	BtifLed	[0]	0 – LED blue 1 – LED green
			[1]	0 – off 1 – on

H/W Identifier	Command Identifier	Command Name	Parameter Bytes	
2	0	HdrmSendStatus	-	
	1	HdrmRegisterRead	[0..3]	register number
			out: [0..3]	value read
	2	HdrmRegisterWrite	[0..3]	register number
			[4..7]	value
	3	HdrmRegisterAnd	[0..3]	register number
			[4..7]	mask
	4	HdrmRegisterOr	[0..3]	register number
			[4..7]	mask
	5	HdrmSetPllFrequency	[0..3]	0 – channel A 1 – channel B
			[4..7]	frequency in Hz (391kHz..100MHz)
	6	HdrmWrite	[0..3]	0 – channel A 1 – channel B
			[4..]	data (byte)
	7	HdrmRead (test only)	[0..3]	0 – channel A 1 – channel B
			[4..7]	number to read (byte)

3	0	NpdSendStatus	-	
	1	NpdRegisterRead	[0..3]	register number
			out: [0..3]	value read
	2	NpdRegisterWrite	[0..3]	register number
			[4..7]	value
	3	NpdRegisterAnd	[0..3]	register number
			[4..7]	mask
	4	NpdRegisterOr	[0..3]	register number
			[4..7]	mask
4	0	CusimSendStatus	[0..3]	0 – CUSIM0 1 – CUSIM1
	1	CusimRegisterWrite	[0..3]	0 – CUSIM0 1 – CUSIM1
			[4..7]	register number
			[8..12]	value
	2	CusimSetPllFrequency	[0..3]	0 – CUSIM0 1 – CUSIM1
			[4..7]	frequency in Hz (391kHz .. 100Mhz)
	3	CusimWriteImage	[0..3]	0 – CUSIM0 1 – CUSIM1
			[4..]	data (long)
	4	CusimSetImageMode	[0..3]	0 – CUSIM0 1 – CUSIM1
			[4..7]	mode 0 – single 1 – repeat

5	0	Q1355SendStatus	[0..3]	0 – Q1355 0 1 – Q1355 1
	1	Q1355RegisterRead	[0..3]	0 – Q1355 0 1 – Q1355 1
			[4..7]	chip identifier 0..3 – SMCS 4..6 – FPGA
			[8..11]	register number
			out: [0..3]	value read
	2	Q1355RegisterWrite	[0..3]	0 – Q1355 0 1 – Q1355 1
			[4..7]	chip identifier 0..3 – SMCS 4..6 – FPGA
			[8..11]	register number
			[12..15]	value
	3	Q1355RegisterAnd	[0..3]	0 – Q1355 0 1 – Q1355 1
			[4..7]	chip identifier 0..3 – SMCS 4..6 – FPGA
			[8..11]	register number
			[12..15]	mask
	4	Q1355RegisterOr	[0..3]	0 – Q1355 0 1 – Q1355 1
			[4..7]	chip identifier 0..3 – SMCS 4..6 – FPGA
			[8..11]	register number
			[12..15]	mask

	5	Q1355Write	[0..3]	0 – Q1355 0 1 – Q1355 1
			[4..7]	1355 channel (0..3)
			[8..11]	EOP flags (Bit0/1: EOP1/2)
			[12..]	data (bytes)
6	0	DppSimSendStatus	[0..3]	0 – DPPSIM0
	1	DppSimRegisterRead	[0..3]	0 – DPPSIM0
			[4..7]	chip identifier 0, 6 – FPGA
			[8..11]	register number
			out: [0..3]	value read
	2	DppSimRegisterWrite	[0..3]	0 – DPPSIM0
			[4..7]	chip identifier 0, 6 – FPGA
			[8..11]	register number
			[12..15]	value
	3	DppSimRegisterAnd	[0..3]	0 – DPPSIM0
			[4..7]	chip identifier 0, 6 – FPGA
			[8..11]	register number
			[12..15]	mask
	4	DppSimRegisterOr	[0..3]	0 – DPPSIM0
			[4..7]	chip identifier 0, 6 – FPGA
			[8..11]	register number
			[12..15]	mask

	5	DppSimWrite	[0..3]	0 – DPPSIM0
			[4..7]	channel (0..1)
			[8..11]	spare
			[12..]	data (shorts)
	6	DppSimSetPllFrequency	[0..3]	0 – DPPSIM0
			[4..7]	frequency in Hz (391kHz..100MHz)
	7	DppSimReadEnable	[0..3]	0 – DPPSIM0
			[4..7]	0 – read enabled 1 – read disabled

5 Registry Settings

The driver has some configuration settings in the registry. The user should never change the settings.

Because the driver is a Windows NT kernel mode driver the settings are stored in the registry. They can be found in the registry path:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\GseMTest\Parameters`

The table below describes the driver parameter:

Table 6 Registry Keys

Name	Type	Description	
BreakOnEntry	DWORD	0	No INT 3 (debug breakpoint) on driver load. This is the default.
		1	Break in INT 3 (debug breakpoint) on driver load. The Windows NT Kernel Debugger must be present.
DebugLevel	DWORD	0	The driver generates no output on EventLog or Kernel Debugger.
		1	The driver generates EventLog output, but no Kernel Debugger output. This is the default.
		2	The driver generates EventLog and Kernel Debugger output, but no timing critical output.
		3	The driver generates full debug output including time critical output. This setting degrades the system performance.
HwBlkQueueLength	DWORD	The length of the block queue in bytes. Minimum: 10 times of the maximum block length in the driver plus 10 times sizeof (ULONG). Meaning the queue is minimum 10 blocks long. There is no maximum queue length. (limited by the system memory). The default is 1MByte.	
SelfStimulationIntervall_ms	DWORD	Time Interval between generated blocks of type **_SelfStimulationTicker** in ms. The minimum is 1 ms, the maximum is 10000 ms. The default is 1000 ms.	

Name	Type	Description
SpareIoPortBase0	DWORD	not used
SpareIoPortBase1	DWORD	not used
SpareIrq	DWORD	not used