Germanium System Tables

Lilly Anderson and listed contributors

January 4, 2024

1 Introduction

This is the GeST (Germanium System Tables) specification v0.1 Rev 2.0. This gives information on how to discover devices, memory regions, and CPU features on a GeST system.

1.1 Definitions

Any case of ”int” just means any valid integer type, or a type reference to it, and any case of ”cint” refers to a constant integer value

- i64 - 8 byte signed integer
- i32 - 4 byte signed integer
- i16 - 2 byte signed integer
- i8 - 1 byte signed integer
- u64 - 8 byte unsigned integer
- u32 - 4 byte unsigned integer
- u16 - 2 byte unsigned integer
- u8 - 1 byte unsigned integer
- [int; cint] - An array of (non-u16) integers with the length of cint

All values should be stored in little endian format, and strings should be in UTF-8 format but not null terminated.

1.2 Contributors

Contributors, besides Lilly Anderson, to the GeST specification and utilites include the following.

- Arcade Wise

2 Header Format

This section lays out how the GeST Header is laid out, and can be used as a reference when trying to parse the header.

2.1 Version

The implementation version can be used to determine compatibility, see the version encoding for information on backwards compatibility. Versions are encoded in the GeST Header's 'vers' section.

Bit(s)	Release type	What it's for
Bits 0-15	Minor Release	Bug fixes
Bits 16-30	Major Release	Extra features
Bit 31	Stable Release	Specifying stablilization status

2.2 GeST Header

A pointer to the GeST Header should be passed when control is handed to your program. This helps to identify where the GeST device tables can be found.

Byte	Size	Name	Use
0	8	Signature	Used to verify a header's validity
8	4	Version	Used to verify header version
12	4	Reserved	Unused
16	8	GDTD Pointer	Points to the GDTD structure

2.3 GDTD

The GDTD (Germanium Device Tree Device) is a device that handles keeping track of devices connected to the system. It should be directly attatched to the system's IO bus, and has a layout as follows. To signal the device to copy the device tree into the DMA region, you must set the DMA Size field to an integer evenly divisible by 4. To check if the copy is complete, check the last 4 bytes of the DMA Region, if it failed it will read as a End of Tree Failed token, otherwise it should read as a End of Tree Success token.

Byte	Size	Name	Use	Access
0	8	Region	DMA region address	Write
8	8	Size	Size of the DMA buffer	Write
16	8	Table Size	The current size of the table[1]	Read
24	8	Valid Flags	Supported GDTD flags	Read
32	8	Flags	Current used GDTD flags[2]	Read/Write
40	8	Interrupt	GDTD interrupt number	Read
48	8	MSI Address	MSI write address	Write
56	8	MSI Number	Number used for MSI writes	Write
64	32	Reserved	Reserved for future use	None
96	4	Version	GDTD version	Read

The interrupt value identifies what interrupt number that the GDTD will use if a new device is added, hotswapping is enabled in the flags, and it supports it. The allowed GDTD flags identifies what flags are compatible with the device. The GDTD flags identifies different features and which are enabled, the layout is as follows.

Bit(s)	Name	Use
0	Hotswapping	Device tree changes generate an interrupt
1	MSIs	Allows the device to use MSIs[3]
2..63	Reserved	Reserved for future use

[1] Table size cannot change if hotswap is not enabled, we suggest clearing the hotswap bit before allocating the DMA region

[2] Any unaccepted flags will be ignored by the device

[3] MSIs will not be sent if hotswapping is not enabled

3 GeST Device Table Bytecode

GeST uses a bytecode to express tables, and values. These can be split up into different tokens, each 2 bytes in size. Unless specified otherwise, all tokens must have 2 byte alignment.

- 0x0000 - Nop, used for padding
- 0x0003 - Start of table token
- 0x0007 - Start of value token
- 0x000B - End of table token
- 0x000F - End of value token
- 0x0303 - End of Tree Success token
- 0x0307 - End of Tree Failure token
- 0xE5NN - Type token

3.1 Nop

This is purely used for padding, and should not otherwise effect the interpretation of a bytecode stream.

3.2 Table tokens

A Start of Table token identifies the start of a table, and must be 4 byte aligned. The bytes following it should be used as follows

- Name length - u16 identifying the length of the name, padding should not be included in the length
- Parent - u32 identifying the offset backwards in bytes to get to it's parent's Start of Table token.
- Name - An array of bytes for the name, must be padded with null characters to 2 bytes

It should be immediately followed by a u16 identifying the length of it's name, following this is a u32 identifying the offset in bytes to get to it's parent table's start token, and then a string, the end of this string must be padded with null characters to align to 2 bytes. This is where the table scope begins, and it can be filled with other tables, or values, type tokens can not be used here. To end the table scope use a End of Table token which must be 2 byte aligned.

3.3 Value tokens

A Start of Value token identifies the start of a table. The bytes following it should be used as follows

- Name length - u16 identifying the length of the name, padding should not be included in the length
- Name - An array of bytes for the name, must be padded with null characters to 2 bytes
- Value type - A token identifying the type of value
- Value length - u16 identifying the length of the value, padding should not be included in the length
- Value - Type is specified in 'value type', size varies by type, it must be padded with null characters to 2 bytes

To identify the end of a value the End of Value token is used.

3.4 Type token

The type token can be used to identify the expected value's size and use. The type token must be 2 byte aligned.

- 0xE503 - u8
- 0xE507 - u16
- 0xE50B - u32
- 0xE50F - u64
- 0xE523 - u8 array
- 0xE52B - u32 array
- 0xE52F - u64 array
- 0xE533 - UTF-8 String

[1]

[1] Interpreter implementations can ignore the difference between different integer sizes and treat them as u64's, but all array sizes must be maintained.

4 Device Table Source Language

The Device Table Source Language, or DeTS Lang, is a language used to describe hardware available in a system.

4.1 DeTS Keywords

- table - Identifies a new table, should be followed by a name in quotes, and then curly braces identifying the table's scope
- int - Identifies a new integer value, must be located within a table's scope, followed by a name, type assignment, and value assignment
- str - Identifies a new string value, must be located within a table's scope, and followed by a name, value assignment
- arr - Identifies a new integer array value, must be located within a table's scope, followed by a name, type assignment, and value assignment
- type - Identifies a type reference, should be followed by a name, and a type assignment

A type assignment can be defined as ": " followed by one of the following types, or a type reference A value assignment can be defined as "= " followed by a constant integer, or string, and a ";".

4.2 Example

```
{ Root
    u32 version 0x01
    arr usize reserved_mem_addr [ 0x1_0000 0x4_0000 ]
    arr usize reserved_mem_len [ 0x2_0000 0x300 ]
    usize io_base 0x0
    usize io_len 0x8_0000

    { Virt0
        str compat "VirtIO"
        usize base 0x1_0000
        usize size 0x1000
    }

    { Processors
        { Core0
            str compat "GeCo Generic"
            { Thread0
                usize base 0x2_0000
                usize len 0x100
            }
        }
        { Core1
            str compat "GeCo Generic"
            { Thread0
                usize base 0x2_0100
                usize len 0x100
            }
        }
        { Core2
            str compat "GeCo Generic"
            { Thread0
                usize base 0x2_0200
                usize len 0x100
            }
        }
    }
}
```