



SIGMA

SIGMAP01 - Reading STF File

Application Note



Address: ASIX s.r.o.
Staropramenna 4
150 00 Prague
Czech Republic

E-Mail: sales@asix.net (sales inquiries, ordering)
support@asix.net (technical support)

WWW: tools.asix.net (development tools)
www.asix.net (company website)

Tel.: +420-257 312 378

Fax: +420-257 329 116

1. File composition

The file format is closely linked to *SIGMA* hardware and may not be suitable for storing data from other logic analyzers than *SIGMA*.

The file is divided into three parts, each separated by *NULL* character (last byte of first and second part is *0x00*). First and second parts are 8-bit text only, encoding does not matter, third part contains binary data, so it can contain *ASCII* control characters including *NULL*.

2. First part: magic

This part is fixed length, containing 16 bytes, 15 characters and *NULL*. It contains string „Sigma Test File“ (the white spaces are *0x20*) terminated by *NULL* character.

If further (incompatible) file format version would be needed to introduce, this magic will be changed to indicate such incompatibility.

3. Second part: settings

This part is variable length and contains only text characters. It is terminated by *NULL* character. Each line is separated by *CR-LF* characters (bytes *0x0D 0x0A*).

The line is in format *Something=Value*. *Something* is identifier and can contain characters A-Z, a-z, numbers 0-9, dots „.“ and underscore „_“, but it cannot start with dot „.“. *Value* is value of that identifier. It can represent string, integer number, ... *Value* can be any combination of characters containing semicolons, equal signs, ... except quotes, which are not allowed. Special formatting inside *Value* may apply.

The *Values* to be written into the file are withdrawn from all components in the system which are registered to be so, containing plugins. This means that number and types of *Values* stored in the file depends on installed plugins in the system. Unknown *Values* should be quietly ignored by the reader.

Next table summaries fundamental *Values* which are found in every test file.

SIGMAP01 - Reading STF File

Value	Meaning	Notes	Example
DateTime	Creation time of file, number of seconds since 1.1.1970		DateTime=1165156868
TestFirstTS	First valid <i>TS</i> in test. Never smaller than 1.	For meaning of <i>TS</i> , see third part of file. All <i>TS</i> values can range to more than 2^{37} .	TestFirstTS=8018015
TestLengthTS	Last valid <i>TS</i> in test. Length of the test in <i>TS</i> is <i>TestLengthTS-TestFirstTS+1</i> .		TestLengthTS=14740456
TestTriggerTS	<i>TS</i> where is trigger. If zero, trigger did not occur.		TestTriggerTS=8926421
TestCLKTime	Sample rate in <i>PU</i> . If indeterministic, value of 15016 is here.	15015 <i>PU</i> =1 ns	TestCLKTime=300300
Sigma.ClockSource	Several options separated by semicolons		Fall=0;Rise=0;...
ClockScheme	0 for 50 MHz and less 1 for 100 MHz 2 for 200 MHz 3 for Asynchronous mode 4 for Synchronous mode		ClockScheme=0
Period	Used only when ClockScheme=0 Clock period=Period*20 ns	Maximum 256	Period=1
Pin	Used in Asynchronous and Synchronous mode Input for clock source. Pin=0 is Input1.	In synchronous mode, only Pin=0 and 8 are supported by <i>SIGMA</i> .	Pin=0
Fall	Used in Asynchronous and Synchronous mode Fall=1 clocks from falling edge.	In synchnous mode, Fall=1 and Rise=1 cannot be set simultaneously.	Fall=0
Rise	Used in Asynchronous and Synchronous mode		Rise=0

SIGMAP01 - Reading STF File

		Rise=1 clocks from rising edge.		
	Sigma.SigmaInputs	Input names separated by semicolons	Unallowed characters are escaped using % sign. See note below table.	SCLK;MISO;MOSI;;;...;
	Sigma.Trigger	Several options separated by semicolons	Triggering options are documented in separate file, <i>SIGMAP04 – Trigger Options</i> .	
	Traces.Traces	Each trace separated by semicolon		Trace1;Trace2;...
		One trace Each suboption separated by colon		Caption=SCLK:Radix=16:Digits=2:...
	Caption	Caption visible in GUI		Caption=SCLK
	Radix	Number format when configured as Bus. Separator contains number of grouped digits. (typically 2 for Radix=16, 3 for Radix=10 and 4 for Radix=2)		Radix=16
	Digits			Digits=2
	Separator			Separator=0
	Type	<p><i>Trace</i> type Input consists of one <i>Input</i>. <i>Trace</i> type Bus consists of many <i>Inputs</i>. <i>Trace</i> type Plugin is trace imported from plugin. Older versions of <i>SIGMA</i> software used also types Analog and Digital. These types are identical to Input type.</p>		Type=Input
	Input0	<p><i>InputIDs</i> which generates <i>Trace</i>. Bus traces are generated by more than one <i>Input</i>, other types are generated by just one <i>Input</i>.</p>	For meaning of <i>InputIDs</i> , see notes below.	Input0=0
	Input1 ...			Input1=...

TS stands for *TimeStamp*. *TimeStamp* increments by one each time *SIGMA* can produce one sample, which is always 16 bits.

PU stands for *PicoUnit*. One ns is 15015 *PicoUnits*.

Test length in seconds is $TestCLKTime * (TestLengthTS - TestFirstTS + 1)$.

In synchronous mode, *TestCLKTime* is not known, value of *TestCLKTime* is 15016, test length in seconds is unknown and formula above has no meaning.

In asynchronous mode, *TestCLKTime* is valid, equal to 300300 (*SIGMA* is sampling at 50 MHz).

In 100 MHz and 200 MHz sampling mode *TS* increments every 20 ns, *TestCLKTime* is 300300. *SIGMA* makes „one sample“ in 20 ns by combining 4 or 8 inputs into 16 bits.

Maximum clock rate is $TestCLKTime = 76876800 PU$, maximum *TS* is $\sim 2^{37}$, so maximum $TestCLKTime * TS$ is about $1.1 * 10^{19}$, $\sim 2^{63.2}$. Limit $2^{63} PU$ is more than 170 hours.

Where applicable, unallowed characters are escaped by % sign followed by two hexadecimal digits.

InputID is identifier of *Input*, either of *SIGMA* or plugin. Plugin Traces have separate *ID* space, Plugin Input 0x10000 and Plugin Trace 0x10000 may not be the same.

InputID	Meaning
0x0000 - 0x00FF	<i>SIGMA</i> analyzer Inputs
0xFFFF8	<i>SIGMA</i> analyzer's virtual rising edge sampling clock
0xFFFF9	<i>SIGMA</i> analyzer's virtual falling edge sampling clock
0x00010000 - 0x7FFFFFFF	Plugin Input. Upper word is identifier of plugin (hash generated from plugin's file name), lower word is plugin's own identifier.

4. Third part: binary data

Binary data are organized in series of *records*. Each *record* holds several *chunks*. One *chunk* holds 64 *clusters*. One *cluster* contains info on one *TimeStamp* and seven consecutive *samples*. To improve compression techniques used in the file, *TimeStamps* and *samples* from all *clusters* in one *chunk* are rearranged, so *TimeStamps* are consecutive and then *samples* are consecutive.

Multibyte integer values are stored in **little endian** format.

In *SIGMA* hardware (but not in STF file!) *cluster* size is 16 bytes and *chunk* size is 1024 bytes.

record				record
<i>compression / decompression</i>				
<i>rearrange</i>				
chunk		chunk		
cluster	cluster	cluster	cluster	
TS smp	TS smp	TS smp	TS smp	

4.1 Record structure

Meaning	Offset [bytes]	Length [bytes]	Notes
payload length	0	4	little endian; in bytes; max 1M
CRC32 of payload	4	4	little endian, same alg. as ethernet
data payload	8	payload length	compressed LZ01X
next record	8+payload length		

Last record of file is indicated by *payload length* = 0xFFFFFFFF and *CRC32* = 0x00000000 (correct checksum for zero length data). Every *STF* file ends with bytes 0xFF 0xFF 0xFF 0xFF 0x00 0x00 0x00 0x00. This does not mean that this sequence cannot be found in the file itself.

For purpose of error checking, *payload length* must not be greater than 1 Mbyte (1048576). *SIGMA* software itself do not create *record* larger than 128 kbyte.

SIGMAP01 - Reading STF File

Structure names used in file are in next table:

	Structure name	Size after decompression	Notes
largest	<i>record data payload</i>	$num_chunks \cdot 1440$	$num_chunks \times chunk = record$
↓	<i>chunk</i>	1440	$64x\ cluster = chunk$
↓	<i>cluster</i>	22	$1x\ TimeStamp \ \& \ 1x\ samples = cluster$
↓	<i>samples</i>	14	$7x\ sample = samples$
smallest	<i>TimeStamp</i>	8	
	<i>sample</i>	2	

Data payload is compressed using *miniLZO* library, algorithm *LZO1X*. The library can be downloaded at address <http://www.oberhumer.com/opensource/lzo/>. Data payload of one *record* is compressed/decompressed at once.

When decompressed, one cluster occupies 1440 bytes. Than,
 $number_of_chunks_in_record = record_length / 1440$.

4.2 Record structure data payload

Decompressed data payload holds rearranged *chunk* and *cluster* data as shown in next table, number_of_chunks_in_record as n .

Meaning	Offset [bytes]	Length [bytes]	Notes
Chunk #1 info	0	32	<i>chunk infos</i> total size = $chunk_count \cdot 32$
...		...	
Chunk #n info	$(n-1) \cdot 32$	32	
Chunk #1 Cluster #1 TimeStamp	$n \cdot 32$	8	<i>chunk.cluster TimeStamps</i> total size = $chunk_count \cdot 64 \cdot 8$
...		...	
Chunk #1 Cluster #64 TimeStamp	$n \cdot 32 + 63 \cdot 8$	8	
Chunk #2 Cluster #1 TimeStamp	$n \cdot 32 + 1 \cdot 64 \cdot 8$	8	
...		...	
Chunk #n Cluster #64 TimeStamp	$n \cdot 32 + (n-1) \cdot 64 \cdot 8 + 63 \cdot 8$	8	
Chunk #1 Cluster #1 Samples	$n \cdot 32 + n \cdot 64 \cdot 8$	14	<i>chunk.cluster Samples</i> total size = $chunk_count \cdot 64 \cdot 14$
...			
Chunk #n Cluster #64 Samples	$n \cdot 32 + n \cdot 64 \cdot 8 + (n-1) \cdot 64 \cdot 14 + 63 \cdot 14$	14	
structure length	$n \cdot 1440$		

4.3 Chunk info structure

Chunk info holds several useful information about *clusters* inside.

Meaning	Offset [bytes]	Length [bytes]	Notes
min	0	2	minimum vector of all <i>samples</i> in <i>chunk</i>
max	2	2	maximum vector of all <i>samples</i> in <i>chunk</i>
Chunk ID	4	4	reserved, can be zero, used only in communication with <i>SIGMA</i> hardware
FirstTS	8	8	<i>TimeStamp</i> of first <i>cluster</i> in <i>chunk</i>
LastTS	8	16	<i>TimeStamp</i> of last <i>cluster</i> in <i>chunk</i>
ChunkLength	8	24	$LastTS - FirstTS + 7$

4.4 Cluster structure

Cluster is *TimeStamp*, a 64 bit integer value of *TimeStamp* of first sample and 7 *samples*, each 16 bits. One *cluster* is 22 bytes long.

Each sample is 16 bits. When sampling 8 inputs at 100MHz, one sample is produced per 20 ns by sampling 8 inputs two times. When sampling 4 inputs at 200MHz, one sample is produces per 20 ns by sampling 4 inputs four times.

TimeStamp increases by one per one 16 bit sample. Thus, when sampling at 100MHz and 200MHz, *TimeStamp* increases by one per 20 ns.

5. Document revision history

Version	When	What
1.00	23.7.2008	First official release