STF Specification

Simple Trace Format Specification

Version 1.3

Contents

ChangeLog

This section captures major incremental changes, and what motivated the changes.

Motivation/Intent	What changed
Version 1.3	
Open Sourcing Prep	Cosmetic changes, rewrite of the introductory paragraph, and rewrite of introduction to STF file format section.
Version 1.2	
InstExceptionPCTarget record covers a subset of what EventRecord and the associated EventPCTargetRecord cover per STF Exception Table definitions.	Removed InstExceptionPCTargetRecord Breaks backward compatibility
Provide a mechanism to record transitions between execution privilege modes (User, Supervisor, Hypervisor, Machine)	Added STF_EVENT_MODE_CHANGE as a meta-event for capturing transitions into a new execution mode. Modified EventPCTargetRecord to be optional, because this meta-event does not have a PC change associated with it.
Reduce specification complexity and improve processing efficiency by combining two Event records which always go together.	Merged event meta-data fields from EventContentRecord into the EventRecord, and removed EventContentRecord. Breaks backward compatibility
Version 1.1	
Provide a summary of changes to make it easier for developers to update tools and comply with the new spec.	Added ChangeLog section
There is a benefit, purely from implementation standpoint, to have Records which precede other Records have lower numerical values. This allows for ordered traversal through related records, ending with the highest number record, concluding a group of records. This is most frequently done with Instruction Encoding (a.k.a. opcode) Records. Recognising that we can easily get into a bind	Records which have “precede” in their semantic definition are set to have lower numbers than the records which follow them, and conversely records which have “follow” in their semantic definition are set to have higher numbers than the records which come before them. Breaks backward compatibility

with numbering, we moved the Instruction Encoding records to a much higher enum, and moved PTE ahead of the memory accesses.	
PTE records provide translation, so they should come before the memory accesses records.	PTE records shall precede memory accesses Records. Breaks backward compatibility
Shuffled ID's for ordering which seems to be more logical.	Moved change of flow records to be at the top as we want them to appear first, followed by register changes, and then address translation (table walks), memory accesses, external events, microops, and finally instruction encodings (opcodes) Breaks backward compatibility
Events can have varying amounts of data that need to be expressed.	EventContentRecord now allows for an extensible number of fields. Definitions of the fields are provided in the STF Exception Event and Event Meta-Data table. It is placed after the EventRecord, to allow potentially adding another related record in the future if need arises. Breaks backward compatibility
Support capturing vector register contents	Added vector type for registers, so that contents can be captured. This is the first step in vector instructions support.
Version 1.0	
Initial Spec	Initial Spec

Introduction

Simple Trace Format (STF) is a binary file format for storing instruction traces, agnostic of instruction set architecture. The format defines a standard for capturing information related to instructions, associated register values, memory access addresses and data associated with them, as well as additional context information such as page table walk, interrupts, bus/fabric transaction addresses/data, etc. This document also specifies how tools generating and/or modifying traces can express information which can aid the tools consuming the traces to interpret them appropriately.

Use Cases

STF is generated by producer tools, such as functional models or hardware, and consumed by tools such as trace-driven performance models, trace analysis tools, and hardware (if the trace contains the relevant functional information).

What is Standardized

The specification covers the following aspects:

- Header that provides enough context to allow proper interpretation of the remainder of the trace
- Entities that describe the execution environment for the program (instruction encoding mode, privilege level, process ID, etc.)
- Entities that describe the per-instruction attributes and changes to (non-memory) program state (instruction encoding, source register values, destination register values, side-effect changes, synchronous exceptions)
- Entities that describe the per-instruction attributes and changes to memory program state
- Entities that capture non-instruction-based changes to program state (exceptions, external interrupts, non-traced TLB changes, etc.)
- Data syntax of each entity including their bit-field encoding
- Data semantics of each entity
- Relationships among entities

What is NOT Standardized

The specification does NOT cover the following aspects:

- Instruction Set Disassembly - external binutils helper packages are used to disassemble instructions at run-time

- Compression format - file compression format is chosen independently of this specification from a plethora of existing compression formats and accompanying tools. The standard, however, requires that the API be extensible to allow developers to use his/her own compression formats.

How to read this specification

- "SHALL" clause - mandatory
- "MUST" clause - mandatory
- "MAY" clause - optional
- "width" - refers to size of data in bits
- *Instruction encoding vs Opcode:*
  - *Instruction encoding* refers to the entire instruction encoding (e.g. 32 bits)
  - *Opcode* refers to the 7 least significant bits [6:0] of the *Instruction Encoding* (per RV32I)
- *Instruction record* is synonymous to *instruction encoding record* (instruction is identified by its instruction encoding record)

Organization of the Specification

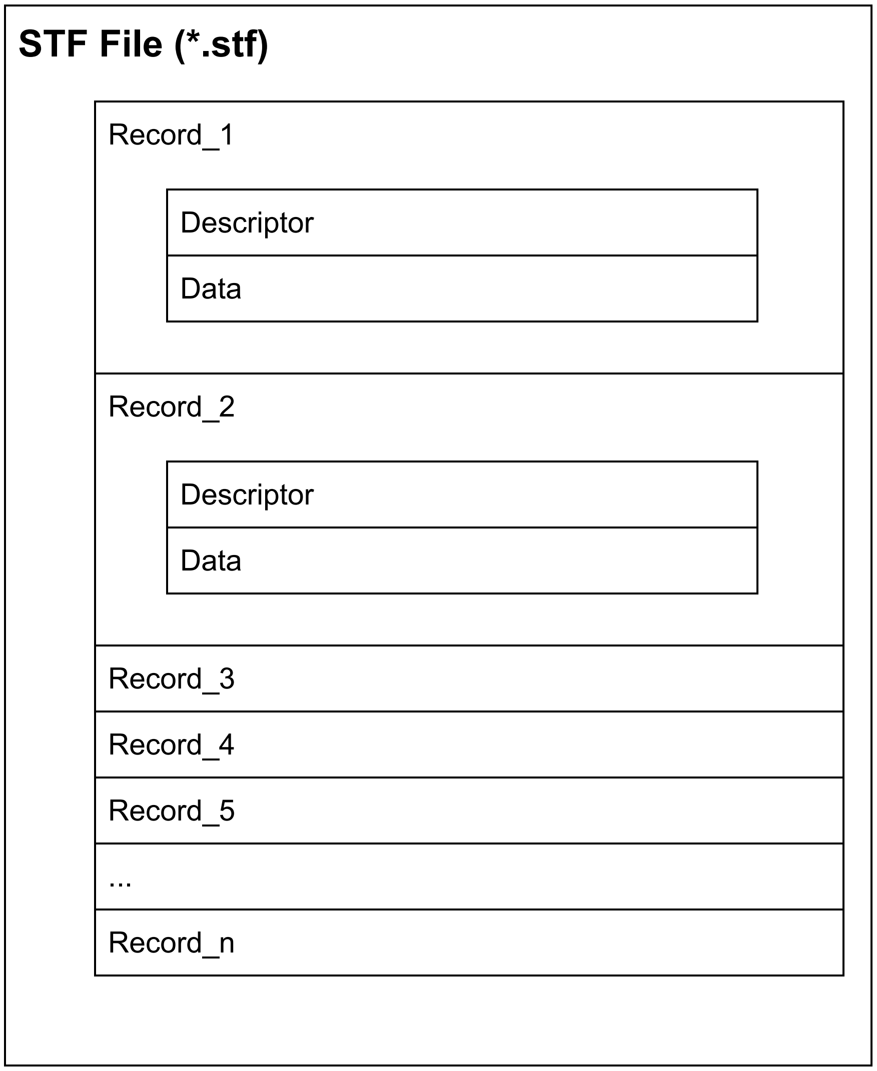
STF specification has 2 domains:

- Syntactic – This part of the specification addresses syntax of entities in an STF file - types of data and associated data structures. It does NOT address the correctness of the relationships among data entities
- Semantic – This part of the specification addresses two main areas:
  - Understanding of each data entity and clarify any ambiguity in interpretation. This area often has to do with the state or pre-condition of the data entities being injected
  - The inter-relationships among various entities of data

STF File Anatomy

STF **record** is an atomic container which holds a fixed width **descriptor** and a variable width **data** associated with the record. Though the data has variable width, it's structure is well defined by the specification.

An STF file is composed of STF records. Basic anatomy of an STF file is shown in this figure:



Record is always a complete atomic entity.

An STF file shall have a minimum of one record.

Multiple records can be attributed to an instruction. Attributing a set of records to an instruction follows general principles:

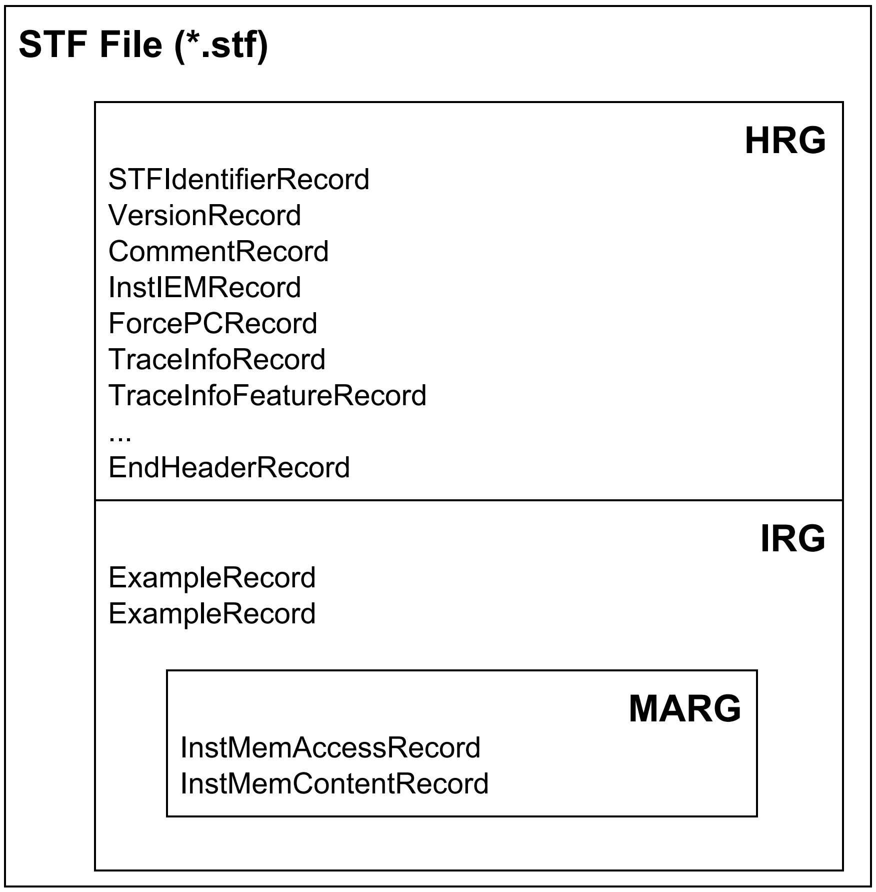
- All records, except an instruction encoding record (STF_INST_16 or STF_INST_32), that precede an instruction encoding record are attributed to the instruction
- There are exceptions to the above rule when it comes to event related records (STF_EVENT, STF_EVENT_CONTENT). Events due to various (mostly asynchronous) exceptions may not be triggered by the instruction they are attributed to according to the STF file.

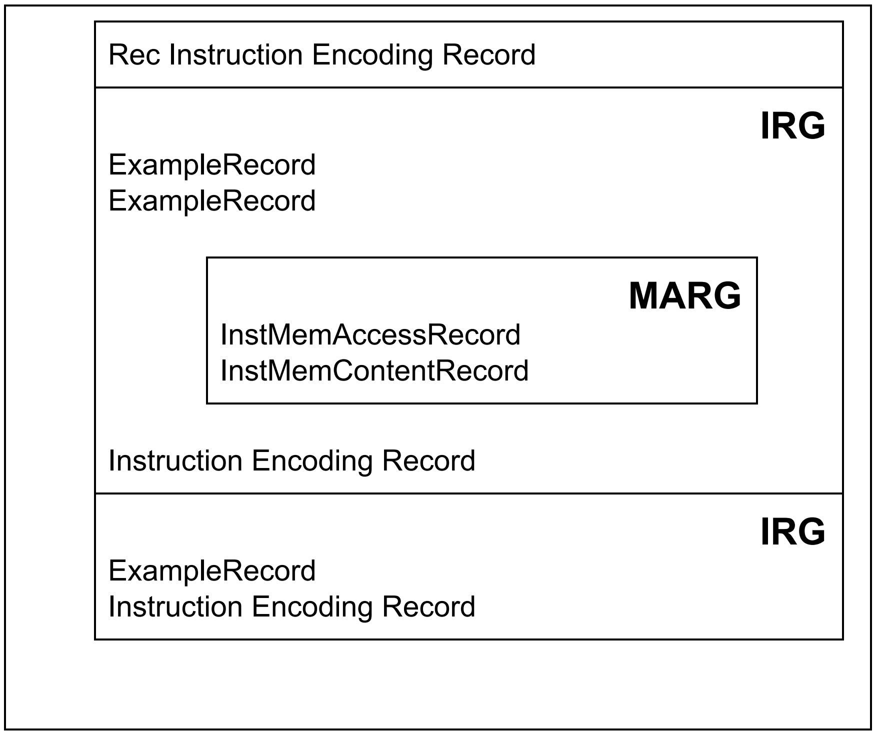
Header Record Group (HRG) - Group of record providing information at a trace level. EndHeaderRecord completes and HRG. There shall be only one HRG at the start of a file. If an STF file is modified by a tool, this tool shall update all relevant fields of the HRG (e.g. stitching).

Instruction Record Group (IRG) - Group of records attributed to an instruction (i.e. all of the records after previous and before the *instruction encoding record* which completes it’s IRG)

Memory Access Record Group (MARG) - Group of records associated with a memory access, at minimum including Memory Address and Content.

Relationship between IRG, MARG, various Records, and Instruction Encoding Record is shown in the following figure:




Rec Instruction Encoding Record
IRG
ExampleRecord
ExampleRecord
MARG
InstMemAccessRecord
InstMemContentRecord
Instruction Encoding Record
IRG
ExampleRecord
Instruction Encoding Record

STF Record Specification

The following table specifies the encoding and semantics of valid STF records.

Example on how to read the table for the CommentRecord:

- Record Data Structure: type name is *CommentRecord*, included in the API header stf.h
- Record Descriptor: *STF_COMMENT* descriptor string from the API header stf.h. The number in parenthesis () is the enumerated sequence number of the descriptor.
- Mandatory: The field is used for specifying if the record is mandatory. If a record is not mandatory in all cases, this field describes which condition triggers the record. It is mandatory to have a *CommentRecord*.
- Record Data Encoding: Data fields are in series, where the first number within brackets [] is the data field size in bytes. *CommentRecord* encodes the number of bytes of comment data = n bytes into the first 32-bit field, followed by the n-byte long actual string data field.
- Semantics & Additional Comments: Semantics of the specification, as well as free form comments providing details, use cases, and clarification for the record specification.

Record Format

Record Data Structure Type see API header stf.h **STF RECORD DESCRIPTOR ()** 1-byte wide descriptor with () enumerated value - may be discontinuous	
Mandatory Specifies if the record is mandatory, and under what conditions.	**Record Data Encoding** b[bit-span] bit encoding, b[-1] record has only a descriptor with no Record Data (zero bit-encoding)
	Semantics & Additional Comments Semantic specification, use-cases, accompanying information, etc.

Specification of Records

N/A STF_RESERVED (0)	
	b[-1] Record has no data, just the descriptor.
	Reserved for error detection

STFIdentifierRecord STF_IDENTIFIER (1)	
YES	b[23:0] = “STF” expressed as the magic number
	Shall exist as the first record of the file. Every tool that creates/modifies stf file shall ensure STFIdentifierRecord exists and is the first record of the file. Display magic number spelling out “STF” at the beginning of the file, to enable identification of STF files without STF tools.

VersionRecord STF_VERSION (2)	
YES	b[31:0] Major version number of the STF specification b[63:32] Minor version number of the STF specification
	Shall exist in all STF files as the second record. Every tool that creates/modifies an stf file shall append/update the VersionRecord indicating compliance to a particular STF specification version. Version numbers are maintained in API header stf.h

CommentRecord STF_COMMENT (3)	
YES	b[31:0] Size of comment string = n bytes b[(8*n+31):32] Comment string data
	Comment string data is non-null terminated. Every tool that creates/modifies stf file shall append a CommentRecord Shall include git SHAs of all projects used to build the trace generator, and names and versions of converter tools tools for reproducibility. CommentRecord may be added at any point in the trace. Use-case: Log git information about projects used to produce the tracing environment

	Trace converter tools (e.g. trim, morph) - append name and version of STF converter tool that transformed the trace since original raw trace generation

ISARecord STF_ISA (4)	
YES	b[15:0] ISA 0 = Reserved 1 = RISC-V 2 = ARM 3 = x86 4 = Power
	Every tool that creates/modifies stf file shall append/update the ISARecord Shall precede InstIEMRecord, to accomodate provisioning for different Instruction Encoding Modes (IEM) for different ISAs.

InstIEMRecord STF_INST_IEM (5)	
YES	b[15:0] Instruction encoding mode 0 = Reserved 1 = STF_INST_IEM_RV32 2 = STF_INST_IEM_RV64
	Every tool that creates/modifies stf file shall append/update theInstIEMRecord. Shall precede the very first instruction encoding record and every instruction encoding change (STF_INST_16 or STF_INST_32) Note: other instruction encoding lengths (e.g.48-bit are not supported at this time)) Instruction encoding mode interpretation will depend on the ISA being traced, as captured in the ISARecord.

TraceInfoRecord STF_TRACE_INFO (6)	
YES	Information about a trace generator or trace modifier: b[7:0] Trace generator/modifier name: e.g. spike, imperas, sail b[15:8] = major version b[23:16] = minor version b[31:24] = minor minor version b[47:32] Size of comment string = n bytes b[(8*n+48):48] Comment string data
	Every tool that creates/modifies stf file shall append a TraceInfoRecord.

TraceInfoFeatureRecord STF_TRACE_INFO_FEATURE(7)	
YES	b[63:0] Features Supported
	Every tool that creates/modifies stf file shall append a TraceInfoFeatureRecord. Provides information for all features supported or not supported in this trace. When traces are manipulated by tools there should be consistency checking of supported features (e.g. trace stitching).

ProcessIDExtRecord STF_PROCESS_ID_EXT (8)	
YES	b[31:0] TGID/PID b[63:32] TID b[95:64] ASID
	Shall precede any instruction record that represents a change from prior instruction record in any of the listed ID's Every tool that creates/modifies stf file shall append/update the ProcessIDExtRecord Indicates any change in thread, process, or address space ID

ForcePCRecord STF_FORCE_PC (9)	
YES	b[63:0] = Virtual address of PC when program COF happens due to non-deterministic cases
	Shall precede the very first instruction encoding record, and be emitted for any program change of flow (COF) due to non-deterministic cases. Every tool that creates/modifies stf file shall append a ForcePCRecord to indicate the starting PC address. This record indicates virtual address of PC when program change of flow (COF) happens due to non-deterministic cases The next STF_INST_16 / STF_INST_32 record's virtual address will match this record's virtual address Trace tools shall output the architectural PC (even if the low/high bits are non-zero - e.g. ARM). Some architectures may add behaviours based on the low/high bits. Note this record is always 64 bit, even when running in 32bit IEM. stf_dump shows this record as a standalone line item prefix "FORCE_PC"

EndHeaderRecord STF_END_HEADER (19)	
YES	b[-1] Record has no data, just the descriptor.
	Every tool that creates an stf file shall append the EndHeaderRecord

	Shall be issued as the last header record in an STF file. Its only purpose is to complete the Header Record Group.

InstPCTargetRecord STF_INST_PC_TARGET (31)	
Whenever COF condition exists due to a branch	b[63:0] Virtual address of target PC when current instruction's branch is taken, causing a change-of-flow (COF).
	Shall be emitted only for branch based COF (not an exception based COF). Omit this record if the branch is not taken. stf_dump shows this records with a prefix of "PC "

InstRegRecord STF_INST_REG (40)	
YES	b[15:0] Register number. Encoding of the register is outlined in stf_reg_def.h as an enum of type STF_REG. See the register encoding table. b[23:16] Register type encoding: b[19:16] Register type: 0000 = reserved 0001 = integer 0010 = floating point 0011 = vector 0100 = CSR b[21:20]: Register operand type: 00 = reserved 01 = state 10 = source register 11 = destination register b[23:22] reserved b[87:24] Register value/content
	Description and content of a register relevant to an instruction. Multiple records are used to convey the state of all/required set of registers for trace consumers. Use-case: Functional model uses this record to dump register state if periodic register dump is set (which is the case for default Functional model run) or specific condition for register dump is met (i.e. at the very beginning of a trace) Periodic register dumps are used as reference points in stf2elf flow to speed up mid-trace machine register state determination.

InstReadyRegRecord STF_INST_READY_REG (41)	
NO	b[15:0] Register number of ready register

	This is used when we artificially modify dependencies between instructions. Mark destination register as ready

PageTableWalkRecord STF_PAGE_TABLE_WALK (50)	
Whenever STF_CONTAIN_PTE bit is set in STF_TRACE_INFO record	b[63:0] Virtual address of the page being accessed b[127:64] Instruction count, count starts at index = 0 based (from the beginning of the trace, that encountered the first memory access occurrence from this page) b[159:128] Page size (Functional model definition: "size of translated page in bytes") b[167:160] Number of PTEs accessed by table walk b[295:168] PTE 0 b[231:168] Physical address of PTE b[295:232] Raw PTE (including page attributes,etc.) ... b[128*n+295:128*n+168] PTE n
	This record captures the page table entries (PTEs) accessed during a page table walk. This is a variable length record dependent on the depth of the page table walk.The last PTE in the record should be the leaf PTE that provides the memory translation. Shall precede any new memory access to a page, where new memory access has one or more of the following meanings: o The very first memory access, since the beginning of the trace, to an address (virtual) that belongs to a new page o An access to a memory address (virtual) which has been accessed before, but has had an update to the virtual to physical page mapping since. If there are multiple pages being accessed by an instruction meeting any of the "new memory access" requirements, then multiple page table walk records corresponding to those accesses shall precede the instruction record. There shall be a page table walk record every time a unique page mapping changes or is newly introduced. Unique page mapping is uniquely identifiable by vmid, asid, and VA. i.e. when vmid and asid are not available, the trace format does not recognize PTE information, so we won't write it. For RV32 only valid modes are Bare (no translation) and Sv32 (32 bit Virtual Addressing) For RV64, besides Bare, Sv39 and Sv48 are supported Attributes [7:0] are mapped the same for all cases, but the other bits are different between S32 vs. S39/S48 Attributes indicate, among other things, if this is a leaf PTE or not

InstMemAccessRecord STF_INST_MEM_ACCESS (60)	
YES	b[63:0] Virtual address of the target memory being accessed by current instruction (read or write). b[79:64] memory access data size b[95:80] memory access attributes. Access type encoding: ToDo: List all access type encodings here (e.g. non-cacheable, write back write allocate, etc.) as well as encodings for prefetches, preloads, etc. b[103:96] access type

	0 = Reserved 1 = Read 2 = Write
	Shall exist for every instruction doing memory read/write The record shall not be used to indicate memory accesses related to instruction address (i.e. fetch). Because we already have explicit information on instruction address (see FAQ item-3) and its instruction encoding content When VA→PA translation is available, this record shall follow a record with PA (not supported by the spec at this time)

InstMemContentRecord STF_INST_MEM_CONTENT (61)	
YES	b[63:0] Data/content of a memory being accessed by current instruction
	This happens only for memory read/write instructions Shall follow STF_INST_MEM_ACCESS For memory accesses with data size less than 8 bytes, data shall be right justified, while exact address and size are expressed in the STF_INST_MEM_ACCESS record. For memory accesses with data size greater than 8 bytes, use multiple STF_INST_MEM_CONTENT records, where the first STF_INST_MEM_CONTENT record refers to the address specified in the STF_INST_MEM_ACCESS record, and subsequent STF_INST_MEM_CONTENT records refer to subsequent target addresses appropriately incremented to preserve continuity of data..

BusMasterAccessRecord STF_BUS_MASTER_ACCESS (62)	
Whenever there are masters other than a single core in the traced environment.	b[63:0] Virtual address of the target memory being accessed b[79:64] memory access data size b[87:80] memory access initiator type 0 = Core 1 = GPU 2 = DMA 3 = PCIe 4 = SRIO 5 = ICN 6 = ACCEL b[95:88] memory access initiator index b[127:96] memory access attributes b[135:128] access type 0 = Reserved 1 = Read 2 = Write
	Shall be used to capture memory accesses by masters other than the primary core being traced. Memory access initiator index distinguishes between multiple instances of one type of master (another core, second accelerator port, etc.)

	Use Cases: Another core’s snoop I/O device read/write

BusMasterContentRecord STF_BUS_MASTER_CONTENT (63)	
YES	b[63:0] Data/content of memory being accessed
	This happens only for bus master read/write Shall follow STF_BUS_MASTER_ACCESS FAQs How is information of memory access data size for STF_BUS_MASTER_ACCESS record conveyed so that we know how much valid data data is there in STF_BUS_MASTER_CONTENT? Using STF_BUS_MASTER_ACCESS record's "mem access data size" field How are memory access content conveyed for access data size > 8 bytes? Using multiple STF_BUS_MASTER_CONTENT records.

EventRecord STF_EVENT (100)	
Whenever COF condition exists due to an event external to the hart/core	b[30:0] STF Exception Event ID number. b[31]: Type 0 = Fault 1 = Interrupt b[39:32] Number of metadata fields b[103:40] Event metadata field 0 ... b[64*n+103:64*n+40] Event metadata field n See table in the STF Exception Event and Event Meta-Data section for more details.
	Shall be used to capture any type of exception or external interrupt being generated in the course of program execution. (synchronous or asynchronous). EventRecord with STF_EVENT_MODE_CHANGE Event ID is a special meta-event, and shall be emitted any time there is a change in privilege mode of execution (e.g. User, Supervisor). It shall be emitted at the start of each trace to indicate the starting privilege mode and with any FORCE_PC_RECORD if there is a change in privilege mode of execution. This record will be associated with the instruction at which the event occurs. This instruction will appear again once it successfully executes, at a later time (e.g. upon returning from an interrupt/exception handler). If this event is a fault, thrown due to an invalid instruction encoding (i.e. fetch issue), generator tool shall report it as a no-op, while preserving and reporting the PC that triggered the fault. Event metadata fields shail be 0-extended to 64-bit stf_dump prints this record prefixed by "EVT "

EventPCTargetRecord STF_EVENT_PC_TARGET (101)	
Whenever needed with an STF_EVENT	b[63:0] Virtual address of target PC when an event exception is taken, causing a change-of-flow (COF).
	May follow STF_EVENT Provides information on which PC should follow this event.

InstMicroOpRecord STF_INST_MICROOP (230)	
Whenever STF_CONTAIN_MICROOP bit is set in STF_TRACE_INFO record	b[7:0] Size of micro-op b[39:8] Micro-op
	Use-case: Used by trace morphing tools to inject new instruction encoding records, a technique used to run micro-architecture "what-if" studies when original instruction is replaced by more than one instruction UseCase 1: Instruction replacement 1:1 in which case trace's replaced instruction would be indicated by STF_INST_32 record. In this case there is no microop record emitted in the trace UseCase 2: Instruction replacement 1:n where n = (#of microops emitted + 1). In both cases following hold true: The original Instruction record STF_INST_MICROOP replacement and/or microop related changes to memory accesses, operand reg records etc. are emitted(or removed) in the trace before the original STF_INST_32/16 record that defines instruction boundary for all artifacts related to the instruction(IRG)

Inst32Record STF_INST_32 (240)	
Whenever instruction is a 32-bit instruction	b[31:0] 32-bit instruction encoding
	Shall conform to the last STF_INST_IEM record before this record All preceding instruction attributes are pulled together with this record to create an instruction record group (IRG (i.e. STFInst)).

Inst16Record STF_INST_16 (241)	
Whenever instruction is a 16-bit instruction	b[15:0] 16-bit instruction encoding
	Shall conform to the last STF_INST_IEM record before this record. All preceding instruction attributes are pulled together with this record to create an instruction record group (IRG (i.e. STFInst))

STFReserveEndRecord STF_RESERVE_END (255)	
	b[-1] Record has no data, just the descriptor.
	This shall be the last record of a trace. Reserved for error detection.

STF Exception Event and Event Meta-Data

The following shows definitions of various STF exception event identifiers and associated meta-data for the events captured by an EventRecord (STF_EVENT).

STF Exception ID	Exception Related Data/Content shall Include	Sync/Async Exception	Description
STF_EVENT_USER_SOFTWARE	Field 0: Source of the interrupt	Async	User software interrupt
STF_EVENT_SUPERVISOR_SOFTWARE		Async	Supervisor software interrupt
STF_EVENT_HYPERVISOR_SOFTWARE		Async	Hypervisor software interrupt
STF_EVENT_MACHINE_SOFTWARE		Async	Machine software interrupt
STF_EVENT_USER_TIMER		Async	User timer interrupt
STF_EVENT_SUPERVISOR_TIMER		Async	Supervisor timer interrupt
STF_EVENT_HYPERVISOR_TIMER		Async	Hypervisor timer interrupt
STF_EVENT_MACHINE_TIMER		Async	Machine timer interrupt
STF_EVENT_USER_EXT		Async	User external interrupt
STF_EVENT_SUPERVISOR_EXT		Async	Supervisor external interrupt
STF_EVENT_HYPERVISOR_EXT		Async	Hypervisor external interrupt
STF_EVENT_MACHINE_EXT		Async	Machine external interrupt
STF_EVENT_COPROCESSOR		Async	Supervisor guest external interrupt
STF_EVENT_INST_ADDR_MISALIGN	Field 0: Virtual address of the instruction	Sync	Instruction Address Misaligned
STF_EVENT_INST_ADDR_FAULT	Field 0: Virtual address of the instruction	Sync	Instruction access fault
STF_EVENT_ILLEGAL_INST	Field 0: Virtual address of the instruction Field 1: Instruction opcode Field 2: Instruction encoding mode	Sync	Illegal instruction
STF_EVENT_BREAKPOINT	Field 0: Virtual address of the instruction	Sync	Breakpoint
STF_EVENT_LOAD_ADDR_MISALIGN	Field 0: Virtual address of the instruction	Sync	Load address misaligned

STF_EVENT_LOAD_ACCESS_FAULT	Field 1: Instruction encoding Field 2: Target address that caused the exception	Sync	Load access fault
STF_EVENT_STORE_ADDR_MISALIGN		Sync	Store address misaligned
STF_EVENT_STORE_ACCESS_FAULT		Sync	Store access fault
STF_EVENT_USER_ECALL	Field 0: System call number	Sync	Environment call from User mode
STF_EVENT_SUPERVISOR_ECALL		Sync	Environment call from Supervisor mode
STF_EVENT_HYPERVISOR_ECALL		Sync	Environment call from Hypervisor mode
STF_EVENT_MACHINE_ECALL		Sync	Environment call from Machine mode
STF_EVENT_INST_PAGE_FAULT	Field 0: Virtual address of the instruction Field 1: Instruction encoding	Sync	Instruction page fault
STF_EVENT_LOAD_PAGE_FAULT	Field 0: Virtual address of the instruction Field 1: Instruction encoding Field 2: Target address that caused the exception	Sync	Load page fault
STF_EVENT_STORE_PAGE_FAULT		Sync	Store page fault
STF_EVENT_MODE_CHANGE	Field 0: Mode in which subsequent instructions execute Field 0 Encoding: 00 - User Mode 01 - Supervisor Mode 10 - Hypervisor Mode 11 - Machine Mode	Sync/Async	Meta-event for capturing transition into a new execution mode

Terminology / Definitions

Change of Flow (COF) - Change of instruction execution flow for reasons that are deterministic as well as non-deterministic. COF is logged in the trace using the STF_INST_PC_TARGET record for deterministic and with STF_FORCE_PC , STF_INST_EXCEPTION_PC_TARGET and STF_EVENT_PC_TARGET for non-deterministic cases.

STF_FORCE_PC applies COF to the current instruction, while all other cases apply it to the subsequent instruction.

1. Deterministic COF cases: These COF cases happen within the scope of a thread context and without having any exceptions being raised within that context.
    a. Function returns through "ret" like instruction
    b. Direct branches through branch instructions where target address of where the execution flow will jump to an address which is either a PC relative(relative to current address) or absolute address
    c. Indirect branches through branch instructions where the target address of where the execution flow will jump to is an address stored in a register. The address was placed in that register by calculation by earlier instruction/s. Typical use case of this is jump to a function called by a function pointer.
2. Non-deterministic COF cases: These COF cases happen due to interruptions in a program thread due to exceptions being raised either by the program thread context itself or due to external interruptions. This category of COF is indicated by
    a. Program flow changes, due to actions of current instruction, to exception handler within the same exception level. Example can be kernel code accessing data from address whose entry is missing in PTE, thus triggering page fault exception within the same exception level.
    b. Program flow changes, due to actions of current instruction, to exception handler to a higher exception level. Example: user-space program trying to access memory causing page fault causing exception level changes to supervisor mode or user-space syscall.
    c. Program flow changes due to return from exception handler
    d. Program flow changes, due to context switches. Example: thread switch, process switch, Address Space Identifier (ASID) switch.
    e. Program flow changes, due to external interrupts to the same or higher exception level.
    f. Program flow changes due to instruction mode change from this set (not applicable to RISC-V)
    g. Program flow changes due to manipulation of a trace file.