Intel(R) Trace Hub (TH)

Overview

Intel(R) Trace Hub (TH) is a set of hardware blocks that produce, switch and output trace data from multiple hardware and software sources over several types of trace output ports encoded in System Trace Protocol (MIPI STPv2) and is intended to perform full system debugging. For more information on the hardware, see Intel(R) Trace Hub developer's manual [1].

It consists of trace sources, trace destinations (outputs) and a switch (Global Trace Hub, GTH). These devices are placed on a bus of their own ("intel_th"), where they can be discovered and configured via sysfs attributes.

Currently, the following Intel TH subdevices (blocks) are supported:

- Software Trace Hub (STH), trace source, which is a System Trace Module (STM) device,
- Memory Storage Unit (MSU), trace output, which allows storing trace hub output in system memory,
- Parallel Trace Interface output (PTI), trace output to an external debug host via a PTI port,
- Global Trace Hub (GTH), which is a switch and a central component of Intel(R) Trace Hub architecture.

Common attributes for output devices are described in Documentation/ABI/testing/sysfs-bus-intel_th-output-devices, the most notable of them is "active", which enables or disables trace output into that particular output device.

GTH allows directing different STP masters into different output ports via its "masters" attribute group. More detailed GTH interface description is at Documentation/ABI/testing/sysfs-bus-intel_th-devices-gth.

STH registers an stm class device, through which it provides interface to userspace and kernelspace software trace sources. See Documentation/trace/stm.rst for more information on that.

MSU can be configured to collect trace data into a system memory buffer, which can later on be read from its device nodes via read() or mmap() interface and directed to a "software sink" driver that will consume the data and/or relay it further.

On the whole, Intel(R) Trace Hub does not require any special userspace software to function; everything can be configured, started and collected via sysfs attributes, and device nodes.

[1] https://software.intel.com/sites/default/files/managed/d3/3c/intel-th-developer-manual.pdf

Bus and Subdevices

For each Intel TH device in the system a bus of its own is created and assigned an id number that reflects the order in which TH devices were enumerated. All TH subdevices (devices on intel_th bus) begin with this id: 0-gth, 0-msc0, 0-msc1, 0-pti, 0-sth, which is followed by device's name and an optional index.

Output devices also get a device node in /dev/intel_thN, where N is the Intel TH device id. For example, MSU's memory buffers, when allocated, are accessible via /dev/intel_th0/msc{0,1}.

Quick example

figure out which GTH port is the first memory controller:

```
$ cat /sys/bus/intel_th/devices/0-msc0/port
0
```

looks like it's port 0, configure master 33 to send data to port 0:

```
$ echo 0 > /sys/bus/intel_th/devices/0-gth/masters/33
```

allocate a 2-windowed multiblock buffer on the first memory # controller, each with 64 pages:

```
$ echo multi > /sys/bus/intel_th/devices/0-msc0/mode
$ echo 64,64 > /sys/bus/intel_th/devices/0-msc0/nr_pages
```

enable wrapping for this controller, too:

```
$ echo 1 > /sys/bus/intel_th/devices/0-msc0/wrap
```

and enable tracing into this port:

```
$ echo 1 > /sys/bus/intel_th/devices/0-msc0/active
```

.. send data to master 33, see stm.txt for more details .. # .. wait for traces to pile up .. # .. and stop the trace:

```
$ echo 0 > /sys/bus/intel_th/devices/0-msc0/active
```

and now you can collect the trace from the device node:

```
$ cat /dev/intel_th0/msc0 > my_stp_trace
```

Host Debugger Mode

It is possible to configure the Trace Hub and control its trace capture from a remote debug host, which should be connected via one of the hardware debugging interfaces, which will then be used to both control Intel Trace Hub and transfer its trace data to the debug host.

The driver needs to be told that such an arrangement is taking place so that it does not touch any capture/port configuration and avoids conflicting with the debug host's configuration accesses. The only activity that the driver will perform in this mode is collecting software traces to the Software Trace Hub (an stm class device). The user is still responsible for setting up adequate master/channel mappings that the decoder on the receiving end would recognize.

In order to enable the host mode, set the 'host_mode' parameter of the 'intel_th' kernel module to 'y'. None of the virtual output devices will show up on the intel_th bus. Also, trace configuration and capture controlling attribute groups of the 'gth' device will not be exposed. The 'sth' device will operate as usual.

Software Sinks

The Memory Storage Unit (MSU) driver provides an in-kernel API for drivers to register themselves as software sinks for the trace data. Such drivers can further export the data via other devices, such as USB device controllers or network cards.

The API has two main parts::

- notifying the software sink that a particular window is full, and "locking" that window, that is, making it unavailable for the trace collection; when this happens, the MSU driver will automatically switch to the next window in the buffer if it is unlocked, or stop the trace capture if it's not;
- tracking the "locked" state of windows and providing a way for the software sink driver to notify the MSU driver when a window is unlocked and can be used again to collect trace data.

An example sink driver, msu-sink illustrates the implementation of a software sink. Functionally, it simply unlocks windows as soon as they are full, keeping the MSU running in a circular buffer mode. Unlike the "multi" mode, it will fill out all the windows in the buffer as opposed to just the first one. It can be enabled by writing "sink" to the "mode" file (assuming msu-sink.ko is loaded).