System Trace Module

System Trace Module (STM) is a device described in MIPI STP specs as STP trace stream generator. STP (System Trace Protocol) is a trace protocol multiplexing data from multiple trace sources, each one of which is assigned a unique pair of master and channel. While some of these masters and channels are statically allocated to certain hardware trace sources, others are available to software. Software trace sources are usually free to pick for themselves any master/channel combination from this pool.

On the receiving end of this STP stream (the decoder side), trace sources can only be identified by master/channel combination, so in order for the decoder to be able to make sense of the trace that involves multiple trace sources, it needs to be able to map those master/channel pairs to the trace sources that it understands.

For instance, it is helpful to know that syslog messages come on master 7 channel 15, while arbitrary user applications can use masters 48 to 63 and channels 0 to 127.

To solve this mapping problem, stm class provides a policy management mechanism via configfs, that allows defining rules that map string identifiers to ranges of masters and channels. If these rules (policy) are consistent with what decoder expects, it will be able to properly process the trace data.

This policy is a tree structure containing rules (policy_node) that have a name (string identifier) and a range of masters and channels associated with it, located in "stp-policy" subsystem directory in configfs. The topmost directory's name (the policy) is formatted as the STM device name to which this policy applies and an arbitrary string identifier separated by a stop. From the example above, a rule may look like this:

```
$ ls /config/stp-policy/dummy_stm.my-policy/user
channels masters
$ cat /config/stp-policy/dummy_stm.my-policy/user/masters
48 63
$ cat /config/stp-policy/dummy_stm.my-policy/user/channels
0 127
```

which means that the master allocation pool for this rule consists of masters 48 through 63 and channel allocation pool has channels 0 through 127 in it. Now, any producer (trace source) identifying itself with "user" identification string will be allocated a master and channel from within these ranges.

These rules can be nested, for example, one can define a rule "dummy" under "user" directory from the example above and this new rule will be used for trace sources with the id string of "user/dummy".

Trace sources have to open the stm class device's node and write their trace data into its file descriptor.

In order to find an appropriate policy node for a given trace source, several mechanisms can be used. First, a trace source can explicitly identify itself by calling an STP_POLICY_ID_SET ioctl on the character device's file descriptor, providing their id string, before they write any data there. Secondly, if they chose not to perform the explicit identification (because you may not want to patch existing software to do this), they can just start writing the data, at which point the stm core will try to find a policy node with the name matching the task's name (e.g., "syslogd") and if one exists, it will be used. Thirdly, if the task name can't be found among the policy nodes, the catch-all entry "default" will be used, if it exists. This entry also needs to be created and configured by the system administrator or whatever tools are taking care of the policy configuration. Finally, if all the above steps failed, the write() to an stm file descriptor will return a error (EINVAL).

Previously, if no policy nodes were found for a trace source, the stm class would silently fall back to allocating the first available contiguous range of master/channels from the beginning of the device's master/channel range. The new requirement for a policy node to exist will help programmers and sysadmins identify gaps in configuration and have better control over the un-identified sources.

Some STM devices may allow direct mapping of the channel mmio regions to userspace for zero-copy writing. One mappable page (in terms of mmu) will usually contain multiple channels' mmios, so the user will need to allocate that many channels to themselves (via the aforementioned ioctl() call) to be able to do this. That is, if your stm device's channel mmio region is 64 bytes and hardware page size is 4096 bytes, after a successful STP_POLICY_ID_SET ioctl() call with width==64, you should be able to mmap() one page on this file descriptor and obtain direct access to an mmio region for 64 channels.

Examples of STM devices are Intel(R) Trace Hub [1] and Coresight STM [2].

stm_source

For kernel-based trace sources, there is "stm_source" device class. Devices of this class can be connected and disconnected to/from stm devices at runtime via a sysfs attribute called "stm_source_link" by writing the name of the desired stm device there, for example:

```
$ echo dummy_stm.0 > /sys/class/stm_source/console/stm_source_link
```

For examples on how to use stm_source interface in the kernel, refer to stm_console, stm_heartbeat or stm_ftrace drivers.

Each stm_source device will need to assume a master and a range of channels, depending on how many channels it requires. These are allocated for the device according to the policy configuration. If there's a node in the root of the policy directory that matches the stm_source device's name (for example, "console"), this node will be used to allocate master and channel numbers. If there's no such

policy node, the stm core will use the catch-all entry "default", if one exists. If neither policy nodes exist, the write() to stm_source_link will return an error.

stm_console

One implementation of this interface also used in the example above is the "stm_console" driver, which basically provides a one-way console for kernel messages over an stm device.

To configure the master/channel pair that will be assigned to this console in the STP stream, create a "console" policy entry (see the beginning of this text on how to do that). When initialized, it will consume one channel.

stm_ftrace

This is another "stm_source" device, once the stm_ftrace has been linked with an stm device, and if "function" tracer is enabled, function address and parent function address which Ftrace subsystem would store into ring buffer will be exported via the stm device at the same time.

Currently only Ftrace "function" tracer is supported.

- [1] https://software.intel.com/sites/default/files/managed/d3/3c/intel-th-developer-manual.pdf
- [2] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0444b/index.html