SoundWire Subsystem Summary

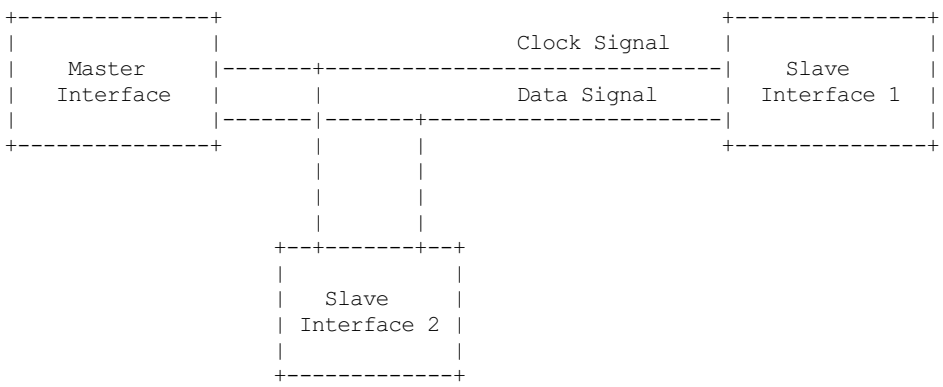
SoundWire is a new interface ratified in 2015 by the MIPI Alliance. SoundWire is used for transporting data typically related to audio functions. SoundWire interface is optimized to integrate audio devices in mobile or mobile inspired systems.

SoundWire is a 2-pin multi-drop interface with data and clock line. It facilitates development of low cost, efficient, high performance systems. Broad level key features of SoundWire interface include:

1. Transporting all of payload data channels, control information, and setup commands over a single two-pin interface.
2. Lower clock frequency, and hence lower power consumption, by use of DDR (Dual Data Rate) data transmission.
3. Clock scaling and optional multiple data lanes to give wide flexibility in data rate to match system requirements.
4. Device status monitoring, including interrupt-style alerts to the Master.

The SoundWire protocol supports up to eleven Slave interfaces. All the interfaces share the common Bus containing data and clock line. Each of the Slaves can support up to 14 Data Ports. 13 Data Ports are dedicated to audio transport. Data Port0 is dedicated to transport of Bulk control information, each of the audio Data Ports (1..14) can support up to 8 Channels in transmit or receiving mode (typically fixed direction but configurable direction is enabled by the specification). Bandwidth restrictions to ~19.2..24.576Mbits/s don't however allow for 11*13*8 channels to be transmitted simultaneously.

Below figure shows an example of connectivity between a SoundWire Master and two Slave devices.



Terminology

The MIPI SoundWire specification uses the term 'device' to refer to a Master or Slave interface, which of course can be confusing. In this summary and code we use the term interface only to refer to the hardware. We follow the Linux device model by mapping each Slave interface connected on the bus as a device managed by a specific driver. The Linux SoundWire subsystem provides a framework to implement a SoundWire Slave driver with an API allowing 3rd-party vendors to enable implementation-defined functionality while common setup/configuration tasks are handled by the bus.

Bus: Implements SoundWire Linux Bus which handles the SoundWire protocol. Programs all the MIPI-defined Slave registers. Represents a SoundWire Master. Multiple instances of Bus may be present in a system.

Slave: Registers as SoundWire Slave device (Linux Device). Multiple Slave devices can register to a Bus instance.

Slave driver: Driver controlling the Slave device. MIPI-specified registers are controlled directly by the Bus (and transmitted through the Master driver/interface). Any implementation-defined Slave register is controlled by Slave driver. In practice, it is expected that the Slave driver relies on regmap and does not request direct register access.

Programming interfaces (SoundWire Master interface Driver)

SoundWire Bus supports programming interfaces for the SoundWire Master implementation and SoundWire Slave devices. All the code uses the "sdw" prefix commonly used by SoC designers and 3rd party vendors.

Each of the SoundWire Master interfaces needs to be registered to the Bus. Bus implements API to read standard Master MIPI properties and also provides callback in Master ops for Master driver to implement its own functions that provides capabilities information. DT support is not implemented at this time but should be trivial to add since capabilities are enabled with the `device_property_` API.

The Master interface along with the Master interface capabilities are registered based on board file, DT or ACPI.

Following is the Bus API to register the SoundWire Bus:

```
int sdw_bus_master_add(struct sdw_bus *bus,
                       struct device *parent,
```

```
                        struct fwnode_handle)
{
        sdw_master_device_add(bus, parent, fwnode);

        mutex_init(&bus->lock);
        INIT_LIST_HEAD(&bus->slaves);

        /* Check ACPI for Slave devices */
        sdw_acpi_find_slaves(bus);

        /* Check DT for Slave devices */
        sdw_of_find_slaves(bus);

        return 0;
}
```

This will initialize sdw_bus object for Master device. "sdw_master_ops" and "sdw_master_port_ops" callback functions are provided to the Bus.

"sdw_master_ops" is used by Bus to control the Bus in the hardware specific way. It includes Bus control functions such as sending the SoundWire read/write messages on Bus, setting up clock frequency & Stream Synchronization Point (SSP). The "sdw_master_ops" structure abstracts the hardware details of the Master from the Bus.

"sdw_master_port_ops" is used by Bus to setup the Port parameters of the Master interface Port. Master interface Port register map is not defined by MIPI specification, so Bus calls the "sdw_master_port_ops" callback function to do Port operations like "Port Prepare", "Port Transport params set", "Port enable and disable". The implementation of the Master driver can then perform hardware-specific configurations.

Programming interfaces (SoundWire Slave Driver)

The MIPI specification requires each Slave interface to expose a unique 48-bit identifier, stored in 6 read-only dev_id registers. This dev_id identifier contains vendor and part information, as well as a field enabling to differentiate between identical components. An additional class field is currently unused. Slave driver is written for a specific vendor and part identifier, Bus enumerates the Slave device based on these two ids. Slave device and driver match is done based on these two ids . Probe of the Slave driver is called by Bus on successful match between device and driver id. A parent/child relationship is enforced between Master and Slave devices (the logical representation is aligned with the physical connectivity).

The information on Master/Slave dependencies is stored in platform data, board-file, ACPI or DT. The MIPI Software specification defines additional link_id parameters for controllers that have multiple Master interfaces. The dev_id registers are only unique in the scope of a link, and the link_id unique in the scope of a controller. Both dev_id and link_id are not necessarily unique at the system level but the parent/child information is used to avoid ambiguity.

```
static const struct sdw_device_id slave_id[] = {
        SDW_SLAVE_ENTRY(0x025d, 0x700, 0),
        {},
};
MODULE_DEVICE_TABLE(sdw, slave_id);

static struct sdw_driver slave_sdw_driver = {
        .driver = {
                   .name = "slave_xxx",
                   .pm = &slave_runtime_pm,
                   },
        .probe = slave_sdw_probe,
        .remove = slave_sdw_remove,
        .ops = &slave_slave_ops,
        .id_table = slave_id,
};
```

For capabilities, Bus implements API to read standard Slave MIPI properties and also provides callback in Slave ops for Slave driver to implement own function that provides capabilities information. Bus needs to know a set of Slave capabilities to program Slave registers and to control the Bus reconfigurations.

Future enhancements to be done

1. Bulk Register Access (BRA) transfers.
2. Multiple data lane support.

Links

SoundWire MIPI specification 1.1 is available at: https://members.mipi.org/wg/All-Members/document/70290

SoundWire MIPI DisCo (Discovery and Configuration) specification is available at: https://www.mipi.org/specifications/mipi-disco-soundwire

(publicly accessible with registration or directly accessible to MIPI members)

MIPI Alliance Manufacturer ID Page: mid.mipi.org