Introduction to SoundWire®

Vinod Koul
Linaro

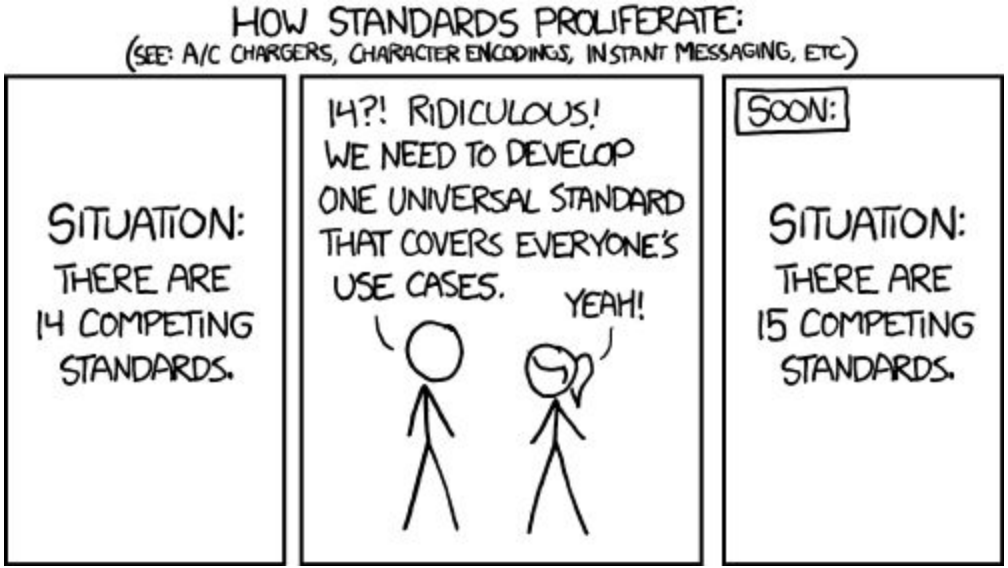
ELC-EU’18
Edinburgh

LEADING
COLLABORATION
IN THE ARM
ECOSYSTEM

Agenda

- Why SoundWire?®
- Topologies
- Protocol
- Linux “soundwire” subsystem

Standards...



Source: https://xkcd.com/927/

Existing Audio Standards!

	HDA	I2S/TDM	PDM	SlimBus
Domain	PC Centric	Mobile/ Embedded	MIC, AMPs	Mobile/ Embedded
Issues	Pin Count, Power	Pin Count, Limited Connectivity, No control	No command, control 2 device/link	Gate count, Complexity

Desire...

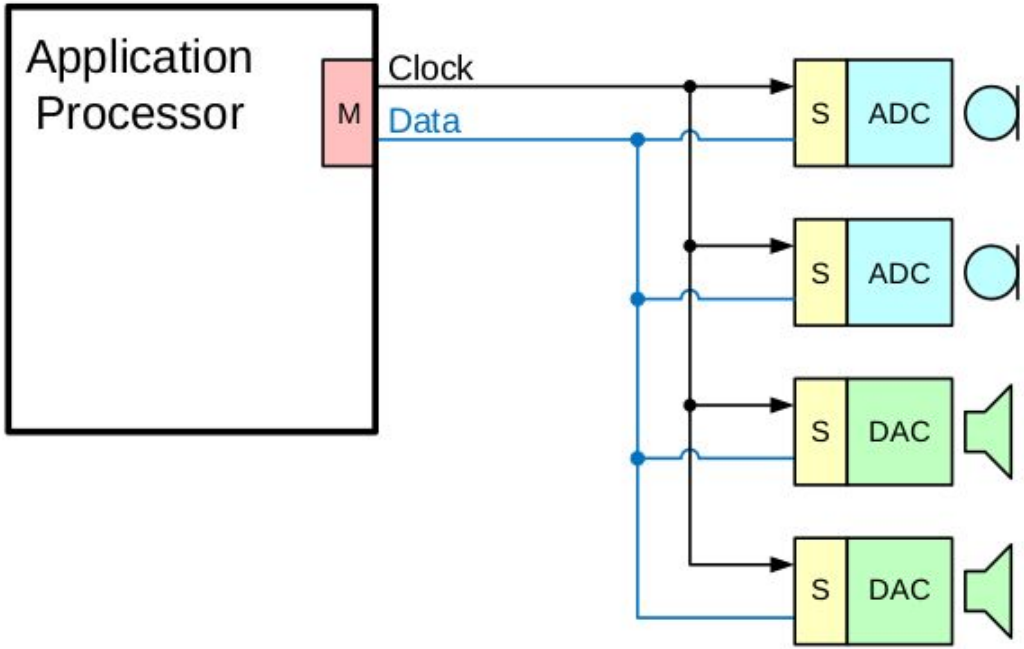
- Reduced pin count bus
- Command and control
- PCM & PDM
- Multidrop
- PC, mobile, embedded...

Agenda

- Why SoundWire®
- Topologies
- Protocol
- Linux “soundwire” subsystem

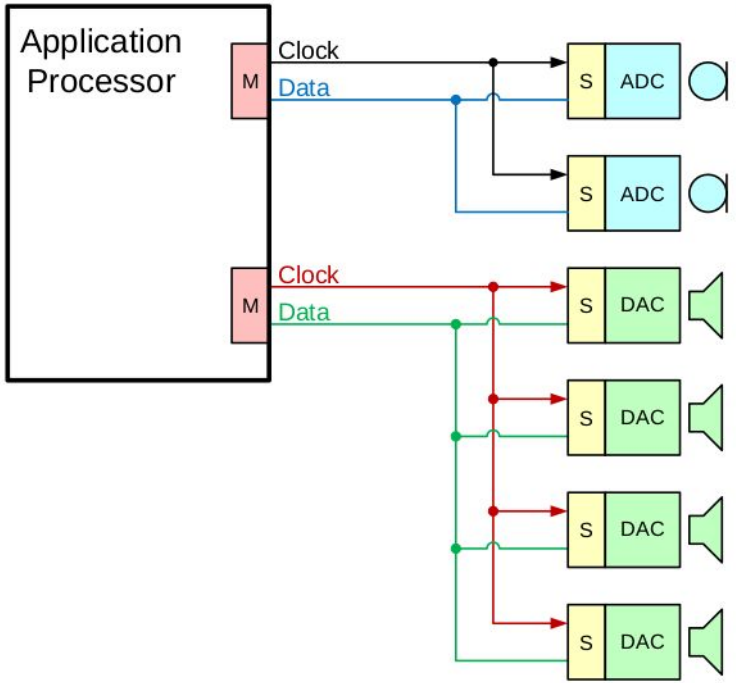
Topologies

Basic System, Stereo Input & Output



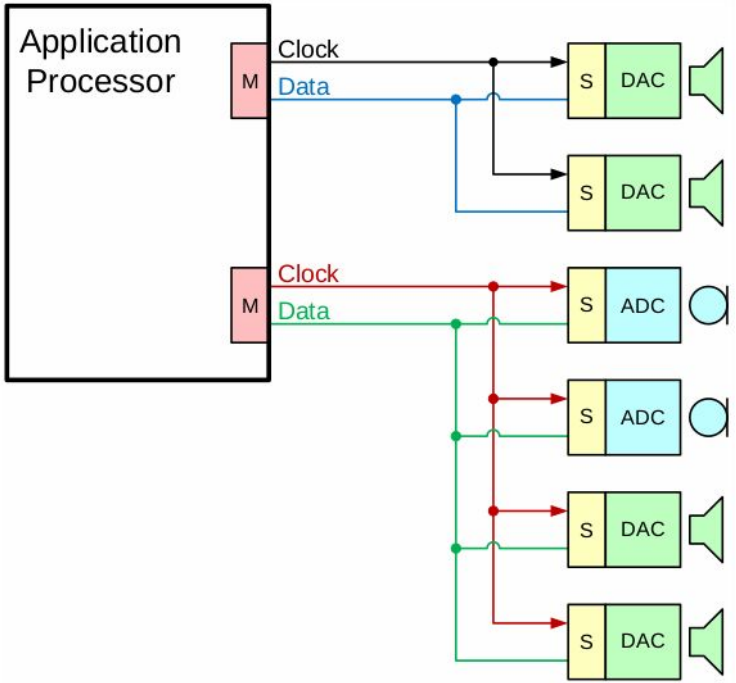
Topologies

Multiple Masters, function split



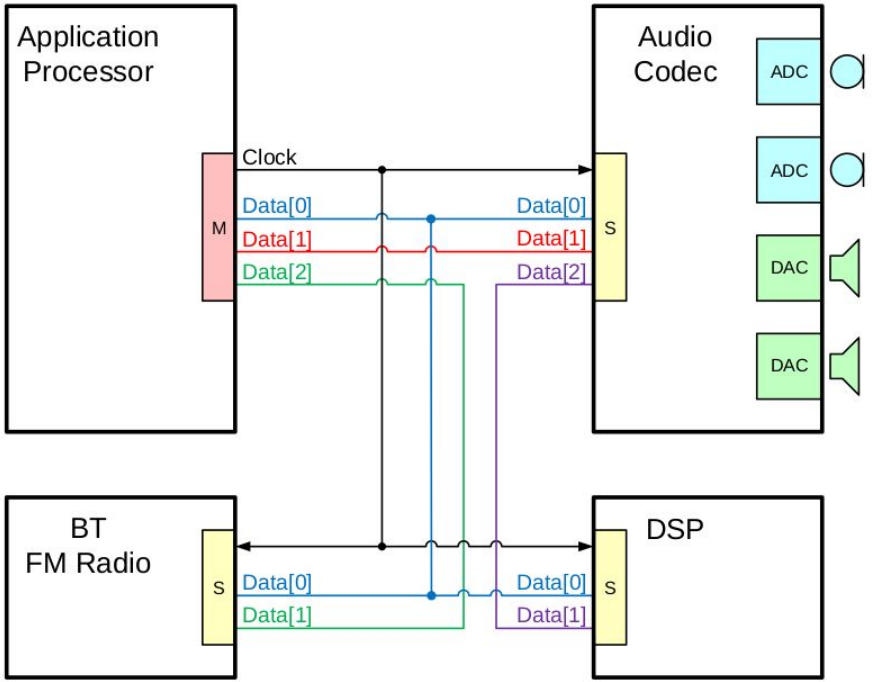
Topologies

Multiple Masters, Alternate partitioning



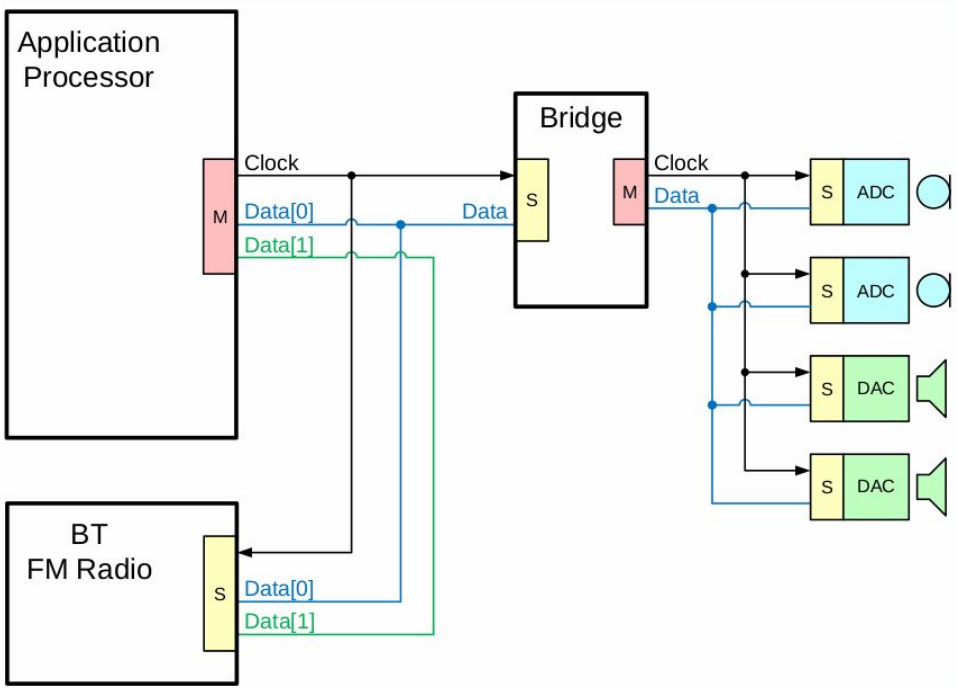
Topologies

Multi-lane



Topologies

Bridge



Agenda

- Why SoundWire®
- Topologies
- Protocol
- Linux “soundwire” subsystem

SoundWire® Features

- 2 pin (data & clock), 1.2V or 1.8V
  - Multi-lane allowed
- Dual Data Rate
  - sampled on each edge
  - NRZI encoding
- Frame length adjustable
- Clocking Scaling up to 12.288 MHz
- Clock Stop
- PCM & PDM Support

SoundWire® Features...

- Isochronous and asynchronous mode
- Bulk transfer capability
- Event signalling
- Low gate count (simpler Slave designs possible)
- Enumerable bus, SW handles enumeration
- Device classes possible

Device Types

- Master
  - Clock, data handling
  - Bus Keeper: (Bus management, bit slot allocation)

- Slave
  - Audio Peripheral (Codec, MIC, Amp)
  - Upto 11 Slave(s) on a bus
  - Can interrupt
  - Can wake
  - Report Status

- Monitor
  - Test equipment, snoop/analyze
  - Take over, issue commands

Data Port

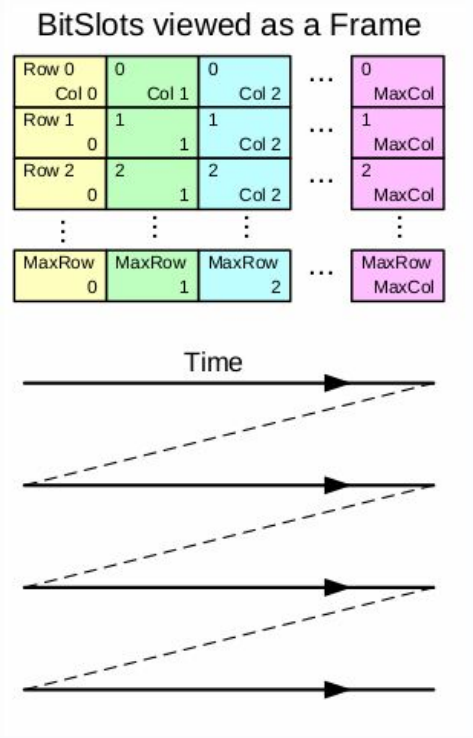
- Port for tx/rx payload data

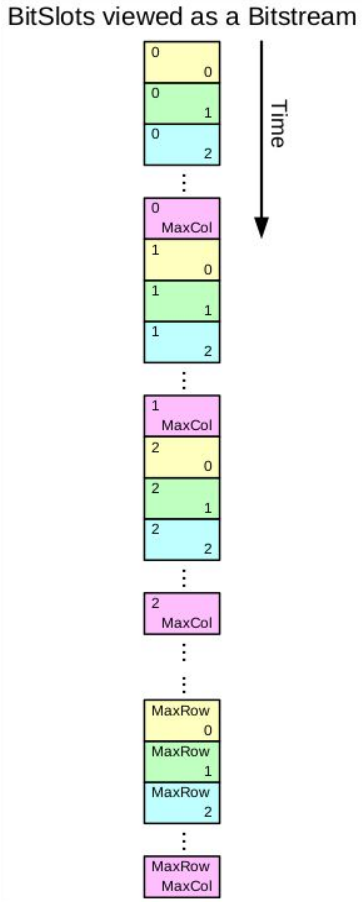
- Types
  - Simple
  - Reduced
  - Full

- Data Port (DP)
  - DP0: Bulk protocol
  - DP1..DP14: Audio functionality
  - DP15: Alias to all DP1..DP14 ports

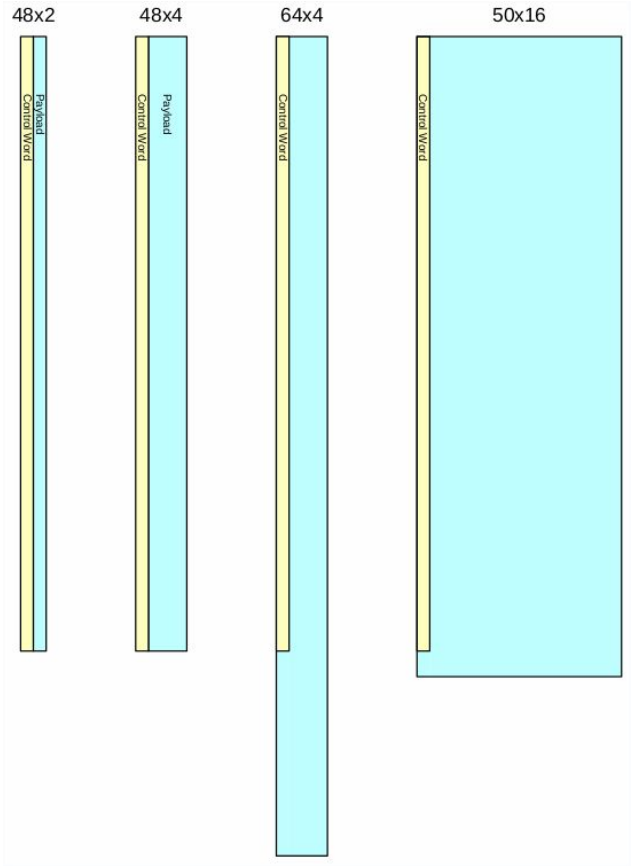
Frame

- Serial transmission
- Frame, 2D pattern.. Row & Col
  - Rows: 2, 4, 6, 8, 10, 12, 14, 16
  - Cols: 48, 50, 60, 64, 75, 80, 125, 147, 96, 100, 120, 128, 150, 160, 250, 192, 200, 240, 256, 72, 144, 90, 180
- Command/Control:
  - 48 Rows of Col0
- PCM
  - Multiple Rows/Cols
- PDM
  - Vertical stripes
  - No conflict with PCM or command/control





Frame Shapes



Control Word

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Ping	Ping REQ	0	0	0	Reserved	SSP	Bus REQ	Bus REL	Slv_Stat_11			Slv_Stat_10			Slv_Stat_09			Slv_Stat_08	
Read		0	1	0	Device Address				Register Address [15:8]										
Write		0	1	1															
Reserved		5 Reserved opcodes			Reserved														

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Ping	Slv_Stat_07		Slv_Stat_06		Slv_Stat_05		Slv_Stat_04		1	0	1	1	0	0	0	1
Read	Register Address [7:0]															
Write																
Reserved	Reserved															

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Ping	PHY_ Sync	Slv_Stat_03		Slv_Stat_02		Slv_Stat_01		Slv_Stat_00		Dynamic Sync [3:0]				Parity	NAK	ACK
Read		Read RegData [7:0]														
Write		Write RegData [7:0]														
Reserved		Reserved														

Device ID

48 bit value, 6 registers DevId 0/5

- Manufacturer ID,
  - 16 bits
  - MIPI standard manufacturer code
- Part ID
  - 16 bits
  - Implementation-defined
- Unique Id
  - 4 bits
  - uniquify multiple instances
- Version
  - 4 bits
  - soundwire protocol implemented
- Class
  - 8 bits
  - Reserved

Device Number

- 4 bits
- 0: Attached but not enumerated
- 1..11: Enumerated Device
- 12..13: Groups
- 14: Reserved for Master
- 15: Broadcast

Enumeration

- On clk sync, Slave reports ATTACHED on DevNum0
- SW reads DevID 0 thru 5
- (re)assigns it a DevNum (1..11)
- Programs DevNum to DevNumber Register
- Slave reports ATTACHED on assigned DevNum
- DevNum reprogrammed on sync loss, PM events

Soundwire® DisCo℠

- Enumerable but not discoverable bus
- MIPI Discovery and Configuration (DisCo℠) mechanism
- Optional, but mandatory for SW
- Implemented as ACPI _DSD method or DT property
- Describes SoundWire® Master and Slave

Agenda

- Why SoundWire®
- Topologies
- Protocol
- Linux “soundwire” subsystem

Bus

```
struct sdw_bus {
        struct device *dev;
        unsigned int link_id;
        struct list_head slaves;
        DECLARE_BITMAP(assigned, SDW_MAX_DEVICES);
        struct mutex bus_lock;
        struct mutex msg_lock;
        const struct sdw_master_ops *ops;
        const struct sdw_master_port_ops *port_ops;
        struct sdw_bus_params params;
        struct sdw_master_prop prop;
        struct list_head m_rt_list;
        struct sdw_defer defer_msg;
        unsigned int clk_stop_timeout;
        u32 bank_switch_timeout;
        bool multi_link;
};
```

Bus Add

```
int sdw_add_bus_master(struct sdw_bus *bus);

void sdw_delete_bus_master(struct sdw_bus *bus);
```

- Master allocates sdw_bus
- Initialized by invoking add API
- Scans (firmware) and add Slaves
- Cleanup by exit API

Bus Ops

```
struct sdw_master_ops {
        int (*read_prop)(struct sdw_bus *bus);

        enum sdw_command_response (*xfer_msg)
                        (struct sdw_bus *bus, struct sdw_msg *msg);
        enum sdw_command_response (*xfer_msg_defer)
                        (struct sdw_bus *bus, struct sdw_msg *msg,
                        struct sdw_defer *defer);
        enum sdw_command_response (*reset_page_addr)
                        (struct sdw_bus *bus, unsigned int dev_num);

        int (*set_bus_conf)(struct sdw_bus *bus,
                        struct sdw_bus_params *params);

        int (*pre_bank_switch)(struct sdw_bus *bus);
        int (*post_bank_switch)(struct sdw_bus *bus);
};
```

Slave & Driver

```
struct sdw_slave {
        struct sdw_slave_id id;
        struct device dev;
        enum sdw_slave_status status;
        struct sdw_bus *bus;
        const struct sdw_slave_ops *ops;
        struct sdw_slave_prop prop;
        struct list_head node;
        struct completion *port_ready;
        u16 dev_num;
};


struct sdw_driver {
        const char *name;
        int (*probe)(struct sdw_slave *sdw,
                        const struct sdw_device_id *id);
        int (*remove)(struct sdw_slave *sdw);
        void (*shutdown)(struct sdw_slave *sdw);
        const struct sdw_device_id *id_table;
        const struct sdw_slave_ops *ops;
        struct device_driver driver;
};
```

Slave Registration

```
int sdw_register_driver(struct sdw_driver *drv);
void sdw_unregister_driver(struct sdw_driver *drv);
```

- Probe on matching Manufacturer ID, Part ID only
- Instance not used for matching
- On device enumeration, update status to driver

Slave Ops

```
struct sdw_slave_ops {
        int (*read_prop)(struct sdw_slave *sdw);

        int (*interrupt_callback)(struct sdw_slave *slave,
                        struct sdw_slave_intr_status *status);

        int (*update_status)(struct sdw_slave *slave,
                        enum sdw_slave_status status);

        int (*bus_config)(struct sdw_slave *slave,
                        struct sdw_bus_params *params);

        int (*port_prep)(struct sdw_slave *slave,
                        struct sdw_prepare_ch *prepare_ch,
                        enum sdw_port_prep_ops pre_ops);
};
```

DisCo^{SM} Properties

```
int sdw_master_read_prop(struct sdw_bus *bus);
int sdw_slave_read_prop(struct sdw_slave *slave);
```

- Spec Optional, but SW mandatory
- Implementation provided, can be overridden
- Driver to set `.read_prop()` callback for bus/slave
- Can choose own implementation
- Uses `device_property_read_xxx()` APIs
  - Firmware agonistic

soundwire IO

```
int sdw_read(struct sdw_slave *slave, u32 addr);

int sdw_write(struct sdw_slave *slave, u32 addr, u8 value);

int sdw_nread(struct sdw_slave *slave, u32 addr,
              size_t count, u8 *val);

int sdw_nwrite(struct sdw_slave *slave, u32 addr,
              size_t count, u8 *val);
```

- Read and Write implementation defined registers
- “N” consecutive read/write
- Regmap supported

stream

```
struct sdw_stream_runtime {
        char *name;
        struct sdw_stream_params params;
        enum sdw_stream_state state;
        enum sdw_stream_type type;
        struct list_head master_list;
        int m_rt_count;
};
```

- Represents Audio Stream
- May contain one or more masters
- Contains at least one slave

stream

```
struct sdw_master_runtime {
        struct sdw_bus *bus;
        struct sdw_stream_runtime *stream;
        enum sdw_data_direction direction;
        unsigned int ch_count;
        struct list_head slave_rt_list;
        struct list_head port_list;
        struct list_head stream_node;
        struct list_head bus_node;
};


struct sdw_slave_runtime {
        struct sdw_slave *slave;
        enum sdw_data_direction direction;
        unsigned int ch_count;
        struct list_head m_rt_node;
        struct list_head port_list;
};
```

Streaming APIs

```
struct sdw_stream_runtime *sdw_alloc_stream(char *stream_name);

void sdw_release_stream(struct sdw_stream_runtime *stream);

int sdw_stream_add_master(struct sdw_bus *bus,
                struct sdw_stream_config *stream_config,
                struct sdw_port_config *port_config,
                unsigned int num_ports,
                struct sdw_stream_runtime *stream);

int sdw_stream_add_slave(struct sdw_slave *slave,
                struct sdw_stream_config *stream_config,
                struct sdw_port_config *port_config,
                unsigned int num_ports,
                struct sdw_stream_runtime *stream);
```

streaming

```
int sdw_stream_remove_master(struct sdw_bus *bus,
                struct sdw_stream_runtime *stream);

int sdw_stream_remove_slave(struct sdw_slave *slave,
                struct sdw_stream_runtime *stream);

int sdw_prepare_stream(struct sdw_stream_runtime *stream);

int sdw_enable_stream(struct sdw_stream_runtime *stream);

int sdw_disable_stream(struct sdw_stream_runtime *stream);

int sdw_deprepare_stream(struct sdw_stream_runtime *stream);
```

Streaming flow

- Calculates Bandwidth & Frame shape required
- Program the transport parameters
- Bank Switch to enable new transport
- Configure Ports
- Enable Ports
- Bank Switch to enable ports
- Converse on tear down

Status

- soundwire subsystem merged into v4.16
- streaming support in v4.18
- Multi-link support in v4.20
- Regmap IO available
- Intel soundwire controller and Cadence IP

- ToDo:
  - Sysfs interface for properties
  - Debugfs support
  - DT support

Links

- SoundWire® Spec v1.1
  https://members.mipi.org/wg/All-Members/document/70290

- SoundWire® DisCo℠ Spec v1.0
  https://members.mipi.org/wg/All-Members/document/71260

- SoundWire® Source Tree
  https://git.kernel.org/pub/scm/linux/kernel/git/vkoul/soundwire.git/

- Documentation:
  https://www.kernel.org/doc/html/latest/driver-api/soundwire/index.html

Thank You

vkoul@kernel.org