Introduction to SoundWire®

Vinod Koul
Linaro

ELC-EU’18
Edinburgh
Agenda

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

Source: https://xkcd.com/927/
**Existing Audio Standards!**

<table>
<thead>
<tr>
<th></th>
<th>HDA</th>
<th>I2S/TDM</th>
<th>PDM</th>
<th>SlimBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>PC Centric</td>
<td>Mobile/Embedded</td>
<td>MIC, AMPs</td>
<td>Mobile/Embedded</td>
</tr>
<tr>
<td>Issues</td>
<td>Pin Count, Power</td>
<td>Pin Count, Limited Connectivity, No control</td>
<td>No command, control 2 device/link</td>
<td>Gate count, Complexity</td>
</tr>
</tbody>
</table>
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
High-Level Block Diagram of a Bridge in an ECOSYSTEM

- **Application Processor**
  - Clock
  - Data[0]
  - Data[1]

- **Bridge**
  - Clock
  - Data

- **BT FM Radio**
  - Data[0]
  - Data[1]
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

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

<table>
<thead>
<tr>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>Slv_Stat_07</td>
<td>Slv_Stat_06</td>
<td>Slv_Stat_05</td>
<td>Slv_Stat_04</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Register Address [7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>PHY_Sync</td>
<td>Slv_Stat_03</td>
<td>Slv_Stat_02</td>
<td>Slv_Stat_01</td>
<td>Slv_Stat_00</td>
<td>Dynamic Sync [3:0]</td>
<td>Parity</td>
<td>NAK</td>
<td>ACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Read RegData [7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Write RegData [7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
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

```c
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

```c
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);
};
```
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

```c
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

```c
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\textsuperscript{SM} Properties

```c
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 \texttt{.read_prop()} callback for bus/slave
- Can choose own implementation
- Uses \texttt{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
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
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);
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: