Automotive Audio Bus (A^2B) on Linux

Alvin Šipraga

Lund Linux Conference 2024

2024-05-23

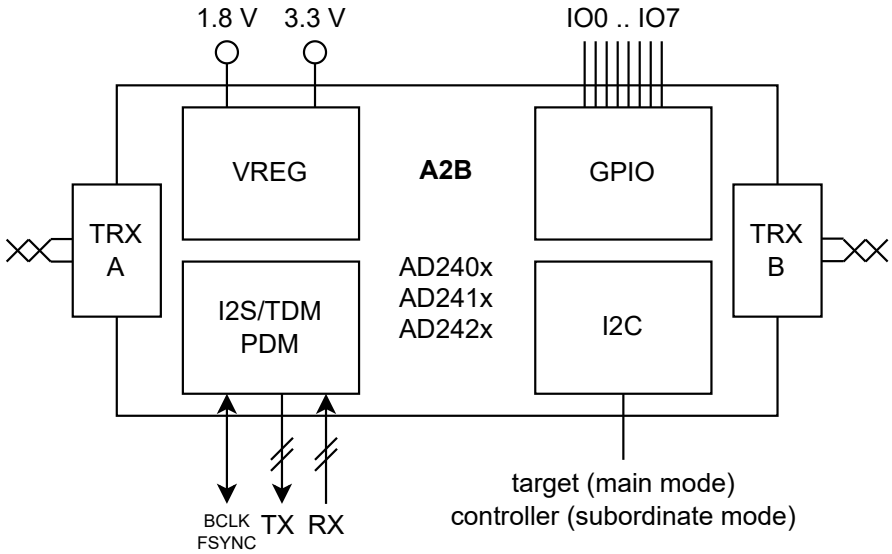
Hello

- Alvin Šipraga
  - born in UK, in DK since 2013
  - background:
    - Linux since ???
    - mathematics until 2015
    - embedded since 2016
    - kernel since 2020
  - Bang & Olufsen a/s (Kgs. Lyngby, Denmark)
  - `alsi@bang-olufsen.dk` / `alvin@pqrs.dk`
  - `https://pqrs.dk/~alvin`

Agenda

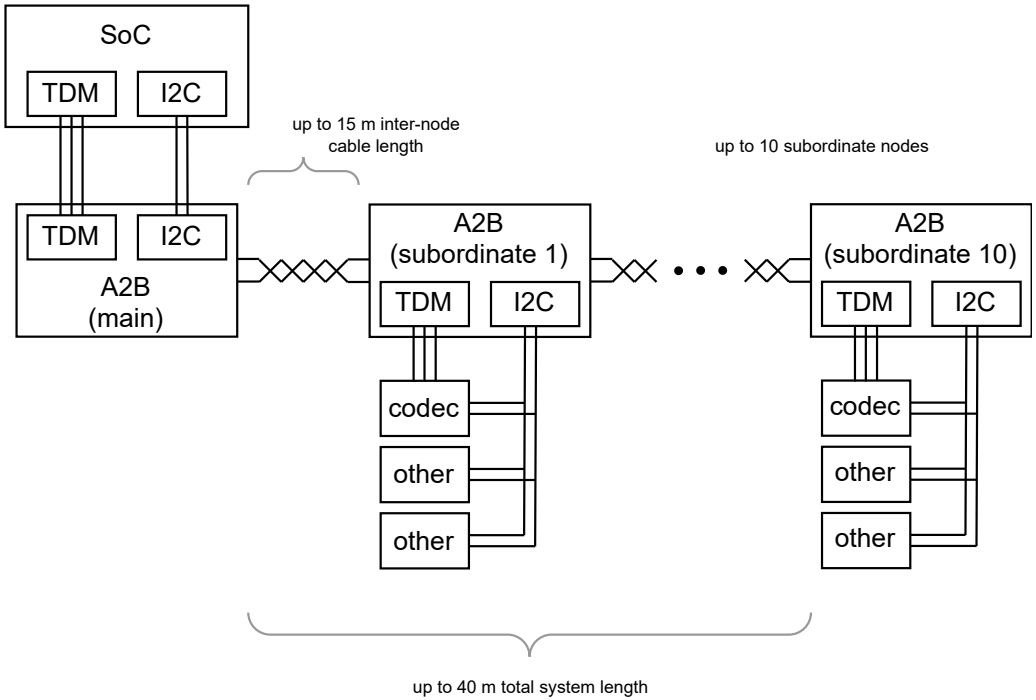
- what is A2B?
- inner workings
- how to model it in Linux?
- configuration of A2B
- case study: Beosound Shape and its challenges
- future work

What is A2B? (1/3)



- control over I2C
- each chip instance is an A2B “node”
- nodes are connected together via unshielded twisted pair (UTP)
- synchronous multi-channel I2S/TDM over distance
- up to 32 channels of digital audio upstream and downstream
- GPIO and I2C controller function over distance

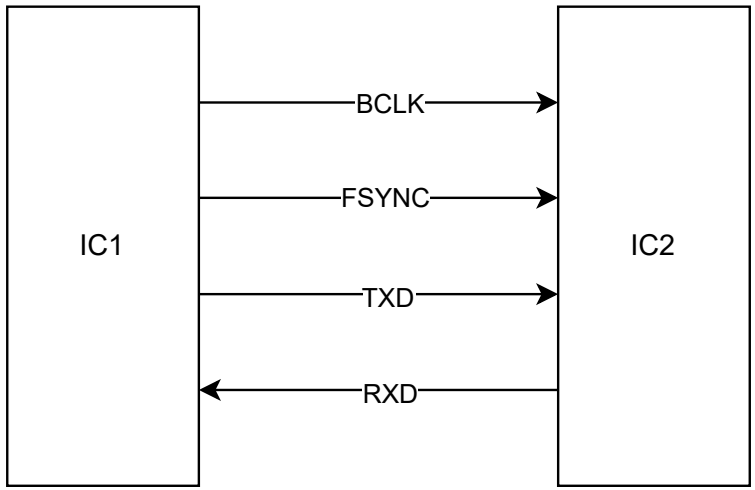
What is A2B? (2/3)



- line topology – nodes are daisy-chained
- discovery algorithm allows for runtime enumeration of the bus
- local- and bus-supplied power available to downstream nodes
- 1.8 V and 3.3 V supplies available for local peripherals
- TDM clock consumer/provider role depends on node role (main/subordinate respectively)

Interlude - I2S/TDM

I2S is a common inter-IC serial bus for transport of 2-channel PCM digital audio.

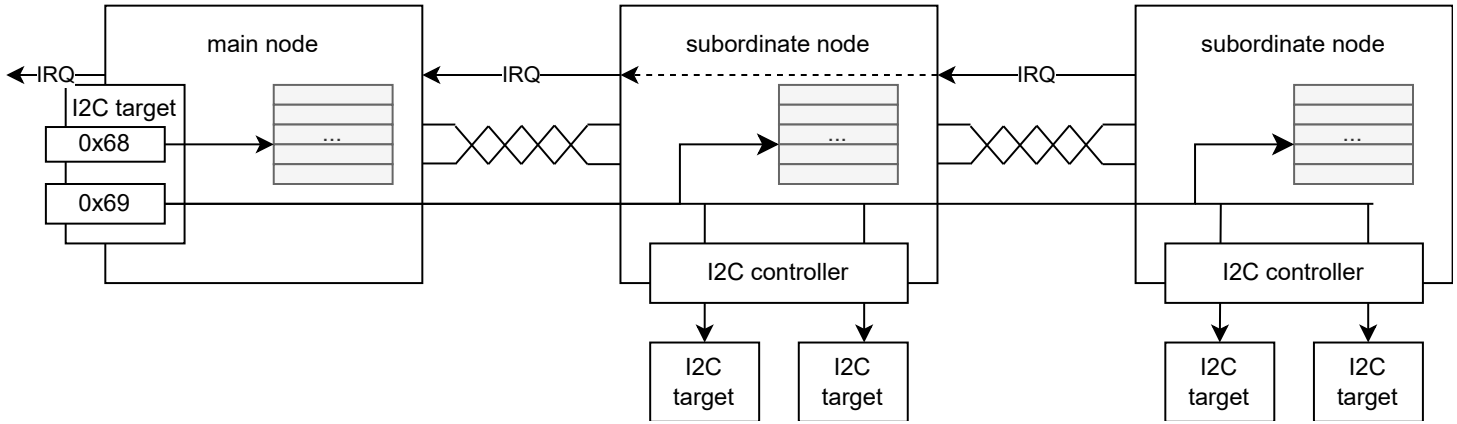


- for more channels, use *time-division multiplexing*; this is called TDM
- both ICs typically have an I2S/TDM interface
- specific formats vary but fundamentals are the same
- 32-bit 32-channel audio $\implies$ $\sim$50 MHz BCLK
- presents challenges for EMC

What is A2B? (3/3)

A2B is a robust and flexible interconnect between ICs or PCBs for digital audio and control transport with reduced EMI.

Inner workings - node control

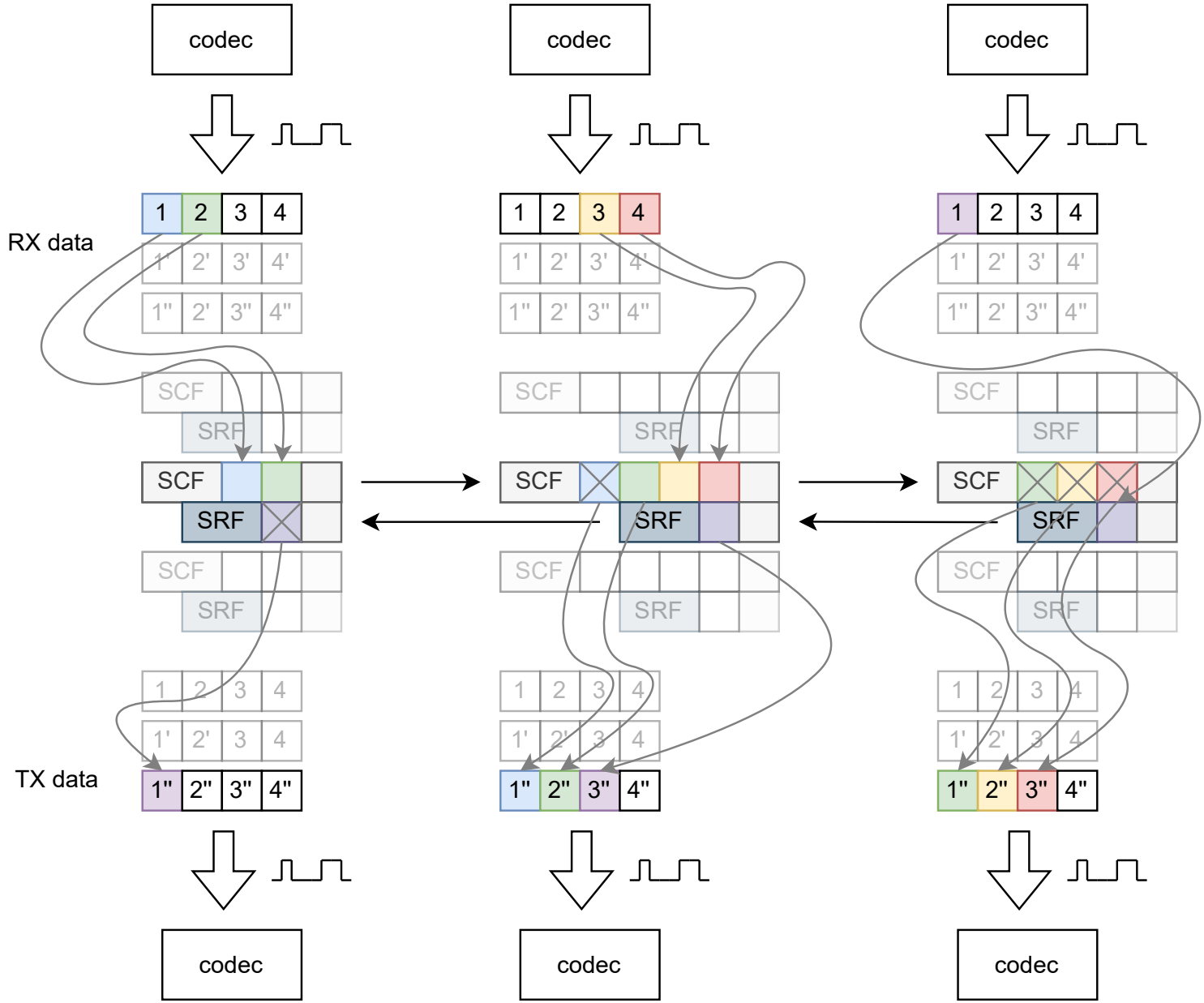


- nodes have same register map; some registers are main/sub only
- main node responds on two I2C addresses
  - one address is for the main node's register map
  - the other address access subordinate nodes' register maps, or I2C targets by proxy
- interrupts are multiplexed through main node's interrupt status registers
- subordinate nodes are discovered sequentially, nearest to farthest

Inner workings - synchronization and bus data

- bus synchronizes to the main node's FSYNC input (typ. 48 kHz)
- bus bitrate is 1024 times the FSYNC frequency $\implies$ $\sim$50 Mbit/s
- data transmission on the bus takes the form of "superframes"
- superframe divided into downstream (SCF) and upstream (SRF) portions
- superframe is transmitted for every FSYNC period
- superframes contain control data and TDM slots sampled on, or to be transmitted by, the TDM interface of the nodes in the bus

Inner workings - example bus data structure



Inner workings - superframes and response cycles

- last subordinate node is responsible for generating the SRF
- each subordinate node must be programmed with a response time (RESPCYCS)
- response time determines when last node will generate SRF, or when subordinates closer to the main node will generate one and signal SRF miss error
- RESPCYCS a function of superframe data structure and node bus location
- superframe data structure is configurable based on TDM slot routing

Modeling A2B in Linux - considerations

- bus is dynamic in nature and supports detection of bus drops (disconnection of nodes/device removal)
- bus also supports runtime rediscovery of nodes, whence hotplug
- Linux bus driver chosen to model the system and support hotplug semantics
- superframe structure should be configurable at runtime to support different use-cases
- future A2B chips support different control interface (SPI rather than I2C), so driver core must be interface agnostic and support code re-use
- A2B node peripheral functions (GPIO, codec, I2C, etc.) modeled in their respective subsystems as A2B drivers
- hence each node adds an additional gpiochip/irqchip, I2C adapter, codec, etc. to the Linux host; control is transparent

Modeling A2B in Linux - the interface driver

Newer A2B chips support SPI as a control interface as well as I2C. The bus interface is decoupled via an interface driver (`module_i2c_device`, etc.), which must register an A2B bus (a class device) and the A2B main node (an `a2b_node` device) of that bus.

Specific responsibilities of the interface driver:

- offer register read/write access to nodes on the registered bus
- expose `i2c_xfer` function for control of subordinate nodes' I2C controllers
- demultiplex IRQ signal and invoke interrupt handler of the relevant node on the bus

The class device representing an A2B bus offers generic bus control, such as triggering of discovery.

Modeling A2B in Linux - the node driver

Nodes are registered with the driver model either by the interface driver (for main nodes), or by the A2B bus driver core (when new subordinate nodes are discovered).

The node driver (`module_a2b_driver`) is interface agnostic but must provide the following to the core:

- setup and teardown functions, wherein any peripheral functions (GPIO, codec, etc.) are to be registered with the A2B bus driver core
- ability to set RESPCYCS in the given node; used when applying new structure
- (main only) apply a new structure
- (main only) trigger the discovery of one more subordinate node
- etc.

Modeling A2B in Linux - bus driver core

The A2B bus driver core has the following responsibilities:

- perform the node discovery algorithm
- synchronize application of a new A2B bus data structure
- synchronize registration of new nodes and their peripheral functions with the discovery process and application of new structures
- receive bus error reports from nodes and handle bus drops
- make it easy to add a driver for a new peripheral function or custom node driver without worrying about A2B bus specifics

Modeling A2B in Linux - peripheral function drivers

Any peripheral function of an A2B node - GPIO, I2C adapter, etc. - is controlled by its own driver which registers with the relevant Linux subsystem. These devices are also A2B devices parented to the relevant node device.

The following peripheral function drivers exist:

- `ad24xx-codec`: I2S/TDM ASoC codec driver
- `ad24xx-gpio`: gpiochip/irqchip driver
- `clk-ad24xx`: driver for CLKOUT1/CLKOUT2 clock outputs
- `i2c-ad24xx`: I2C controller adapter driver

Modeling A2B in Linux - device topology

```
1-0068
 +- a2b-0
     +- a2b-0.0
         +- a2b-0.1
         |   +- a2b-0.2
         |   |   +- a2b-0.2-codec
         |   +- a2b-0.1-gpio
         |   +- a2b-0.1-codec
         +- a2b-0.0-clk
         +- a2b-0.0-codec
```

- 1-0068 is the top-level I2C device
- a2b-0 is the `a2b_bus` class device
- a2b-0.0 is the main node `a2b_node` device
- a2b-0.N is the N^{th} subordinate node `a2b_node` device
- a2b-0.N-codec is the N^{th} node's audio codec `a2b_func` device
- ditto for a2b-0.N-{clk,gpio} etc.

Configuring A2B - superframe structure

One of the design goals was to enable the application of a new superframe structure depending on use-case. The bus core allows nodes or their peripherals to request a particular number of slots and synchronizes this. When all nodes have made their request, a new structure is applied.

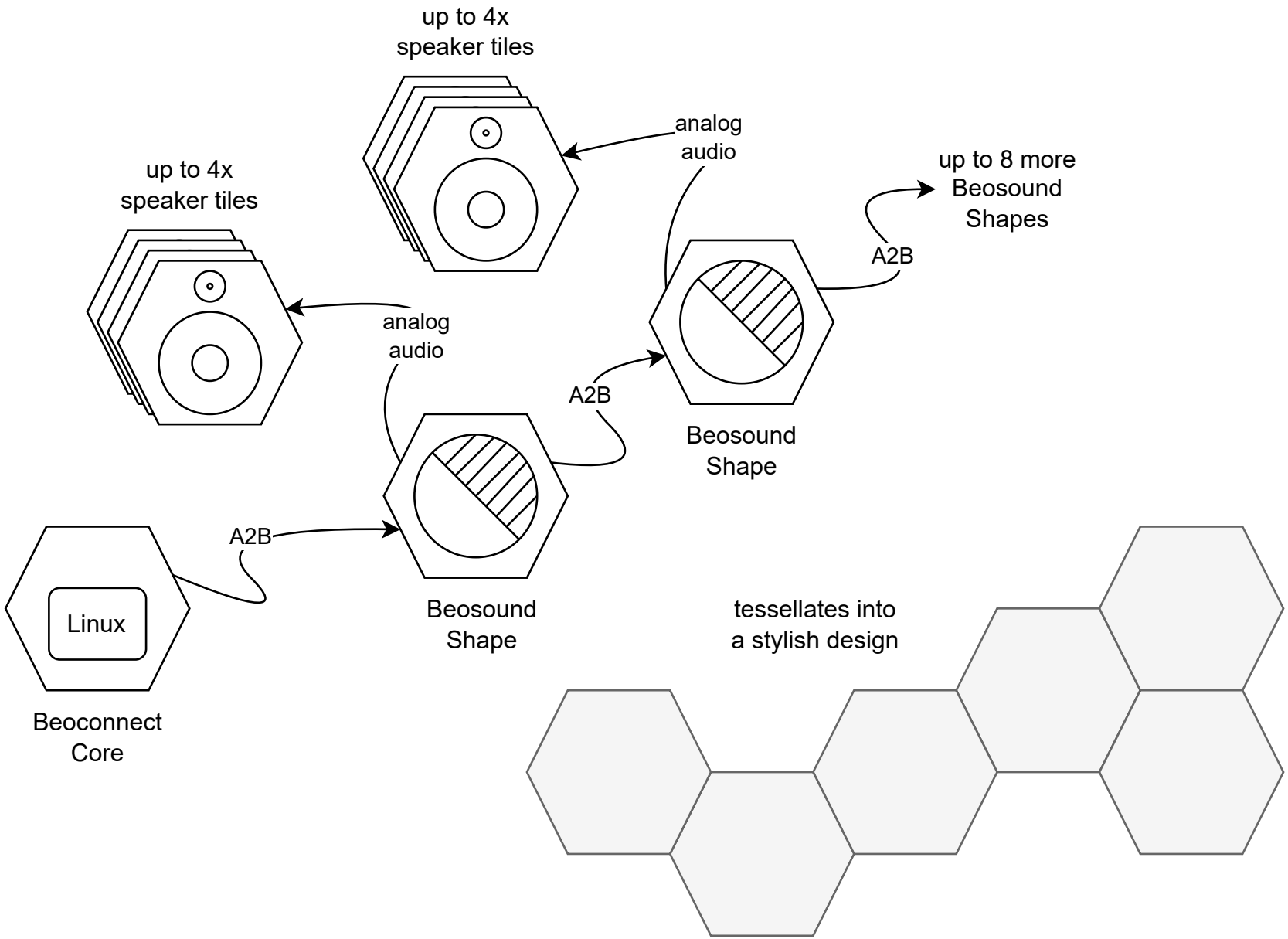
- bus structure is only coherent if nodes agree on the amount of traffic (number of slots) transmitted between A and B, both upstream and downstream
- driver core will check that the structure is balanced before applying and print diagnostic errors if not
- the ASoC codec driver exposes kcontrols to configure this
- in `hw_params` and `hw_free` the slots are requested and freed
- new structure gets applied in coordination with sound card usage

Configuring A2B - device tree (OF)

Currently A2B is configured through device free (firmware), i.e. statically. The driver model does not preclude a more dynamic approach along the lines of USB, but this is not implemented yet.

- ASoC and embedded ARM is heavily OF based, so it makes sense
- challenges with respect to device (node) absence (`fw_devlink=on`)

Case study - Beosound Shape



Other points of discussion

- mapping kcontrols to DAPM graph automatically
- custom node drivers: the Beosound Shape
- building a Shape sound card: nodes are not always present
- modelling A2B muxes and USB altmodes
- future work: EEPROM parsing and A2B node type detection

EOF

Thank you for listening!

Link to v1:
`https://lore.kernel.org/lkml/20240517-a2b-v1-0-b8647554c67b@bang-olufsen.dk/`

Special thanks to...

- Emil Svendsen @ B&O
- my employer
- ADI, for their great hardware and documentation

Bang & Olufsen is hiring for embedded Linux and kernel development!