# CPU-DAI

**ASOC**

Platform

Machine

Codec

The main function of the Platform driver is to complete the management of audio data, and finally transmit the audio data to the Codec through the digital audio interface (DAI) of the CPU.

snd\_soc\_dai\_driver.

snd\_soc\_platform\_driver

Platform Driver

**ASOC**

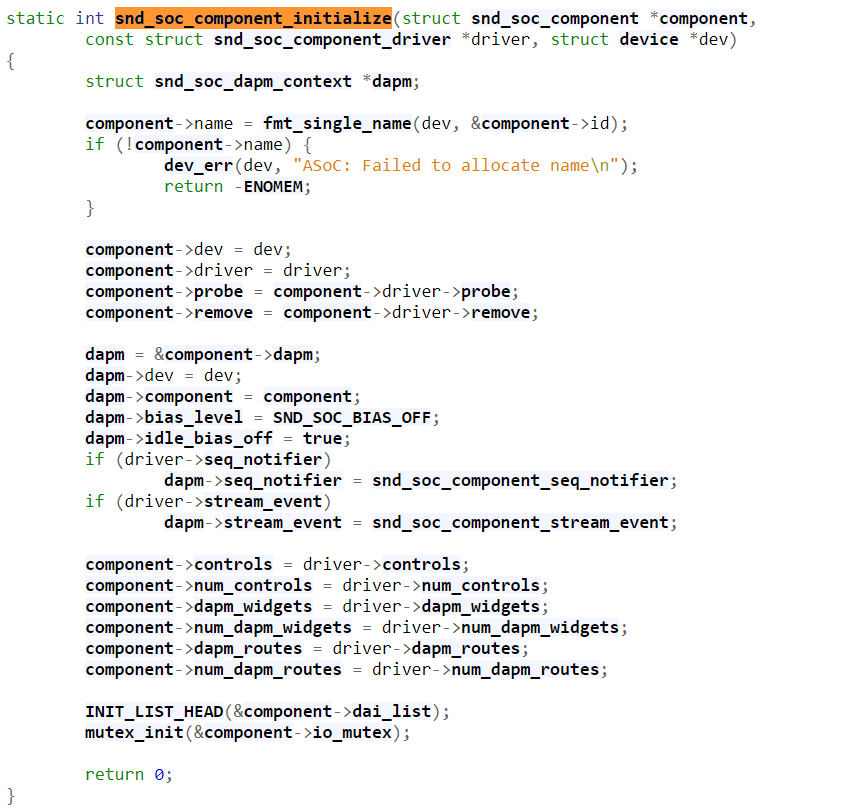
ASoC divides the Platform driver into two parts: *snd\_soc\_platform\_driver* and *snd\_soc\_dai\_driver*. Among them, platform\_driver is responsible for managing audio data and transferring audio data to cpu dai through dma or other operations. Dai\_driver mainly completes the parameter configuration of dai on the cpu side. At the same time, it will also pass necessary dma and other parameters with snd\_soc\_platform\_driver through certain channels to interact.

# Code flow

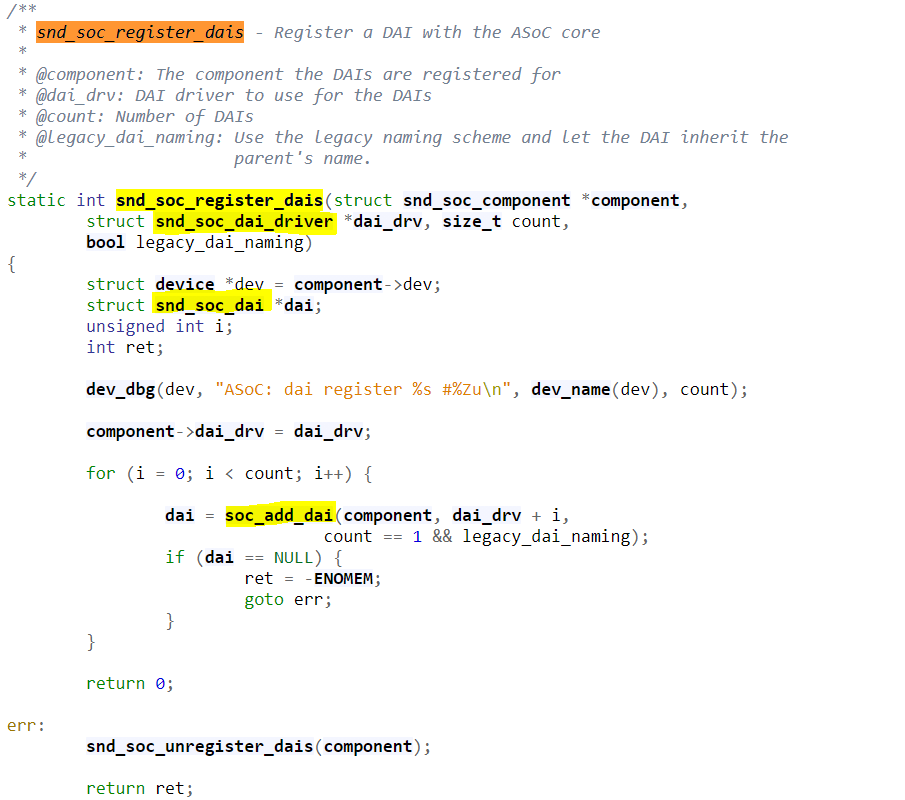
1. The CPU DAI is registered by calling *snd\_soc\_register\_component*.



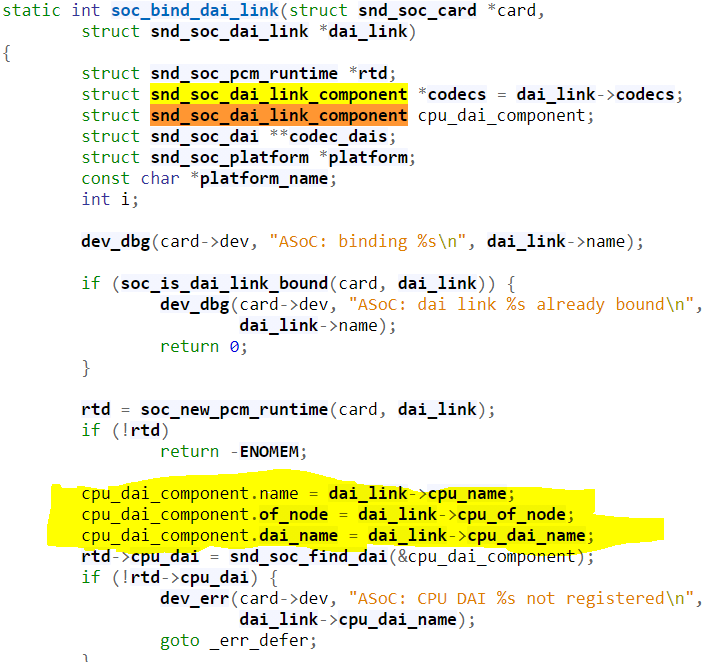
1. In *snd\_soc\_register\_component* function, *snd\_soc\_component*  is created and memory is allocated. *snd\_soc\_component\_driver*  and *snd\_soc\_dai\_driver* are then registered.
2. Call *snd\_soc\_component\_initiallize* to initialize the component, using *snd\_soc\_component\_driver* to initialize the component. In CPU DAI, dapm mainly describes how *FE DAI* and *BE DAI* are linked.
3. assign the component driver to the driver member of the component, use the component driver dapm\_widget and dapm\_route to assign the corresponding members of the component, and initialize the dai\_list of the component.



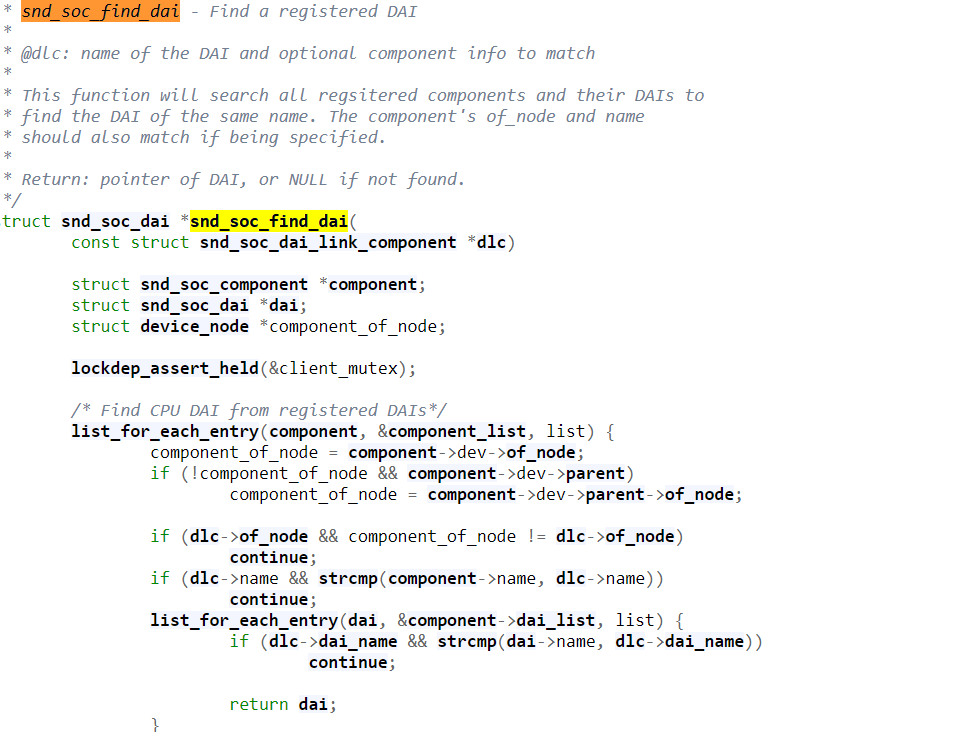
1. **Registration of cpu's snd\_soc\_dai driver**
   1. Define an instance of snd\_soc\_dai\_driver structure;
   2. Register the snd\_soc\_dai instance through API: snd\_soc\_register\_dai or snd\_soc\_register\_dais in the probe callback in the corresponding platform\_driver;
   3. Implement callbacks such as probe and suspend in the snd\_soc\_dai\_driver structure;
   4. Implement the callback function in the snd\_soc\_dai\_ops field in the snd\_soc\_dai\_driver structure



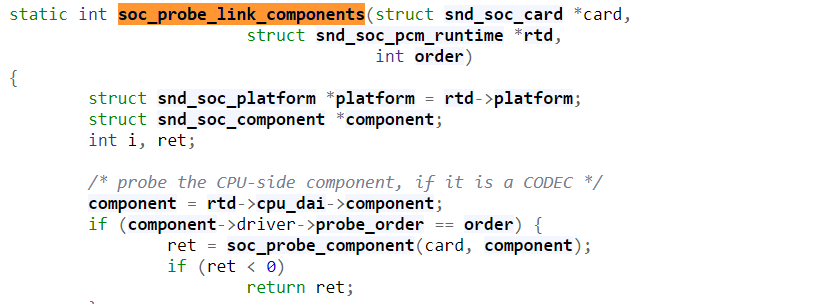
1. The structure of snd\_soc\_dai\_driver needs to be defined according to different SOC chips. The key fields are introduced as follows:
   1. The probe and remove callback functions are called when the sound card is loaded and unloaded respectively;
   2. suspend, resume power management callback function;
   3. ops points to the snd\_soc\_dai\_ops structure, which is used to configure and control the dai;
   4. The playback snd\_soc\_pcm\_stream structure is used to indicate the number of channels, bit rate, data format and other capabilities supported by the dai;
   5. The capture snd\_soc\_pcm\_stream structure is used to indicate the number of channels, bit rate, data format and other capabilities supported by the dai;
2. Traverse the snd\_soc\_dai\_driver list in the snd\_soc\_register\_dais function, and add each dai to the component dai\_list through soc\_add\_dai.
3. **The ops field in snd\_soc\_dai\_driver**
   1. all the configuration and control of dai are implemented through these callback functions.
   2. The working clock configuration function is usually called by the machine driver:
      1. set\_sysclk sets the main clock of dai;
      2. set\_pll set PLL parameters;
      3. set\_clkdiv set frequency division coefficient;
      4. The format configuration function of dai is usually called by the machine driver:
      5. *set\_fmt* sets the format of dai;
      6. *set\_tdm\_slot* If dai supports time division multiplexing, it is used to set the time division multiplexing slot;
      7. *set\_channel\_map* channel time division multiplexing mapping setting;
      8. *set\_tristate* sets the state of the dai pin. This callback is required when the same pin is used in parallel with other dais;
4. The following apis are usually used by the machine driver. The machine driver uses these apis in the hw\_params callback in the *snd\_pcm\_ops* field:
   1. *snd\_soc\_dai\_set\_fmt()* will actually call *snd\_soc\_dai\_ops* or the *set\_fmt* callback in the codec driver;
   2. *snd\_soc\_dai\_set\_pll()* will actually call *snd\_soc\_dai\_ops* or the set\_pll callback in the codec driver;
   3. *snd\_soc\_dai\_set\_sysclk()* will actually call *snd\_soc\_dai\_ops* or the set\_sysclk callback in the codec driver;
   4. *snd\_soc\_dai\_set\_clkdiv()* will actually call *snd\_soc\_dai\_ops* or the set\_clkdiv callback in the codec driver;
   5. The second parameter fmt of *snd\_soc\_dai\_set\_fmt(struct snd\_soc\_dai \*dai, unsigned int fmt)* is specifically mentioned here. ASoC currently only uses its low 16 bits, and some macros are specifically defined for it to facilitate our use
5. Call *snd\_soc\_component*\_add to add component to the global list component\_list.
6. When the card is subsequently registered (*snd\_soc\_register\_card*), the *soc\_bind\_dai\_link* function will be called in *snd\_soc\_instantiate\_card*. In this function, the cpu dai on a certain dai\_link of the machine driver can be found by name, as follows:



1. Find the corresponding cpu dai through the cpu\_name and cpu\_dai\_name of dai\_link. The specific search process is in the snd\_soc\_find\_dai function. First find the component through cpu\_name, then traverse component->dai\_list, and use cpu\_dai\_name to match the name of dai.



1. Probe the component of CPU DAI in soc\_probe\_link\_components:



1. In the soc\_probe\_component function, probe the CPU DAI, mainly including
   1. Assign value to component and dapm context card.
   2. Call snd\_soc\_dapm\_new\_controls to create the dapm\_widget of the component, and add the widget to the card->widgets list.
   3. Call snd\_soc\_dapm\_new\_dai\_widgets for each dai widget in the component dai\_list, the name of the widget is dai->driver->playback/capture.name. And the widget is added to the card->widgets list.
   4. Call the probe function of the component.
   5. Call snd\_soc\_dapm\_add\_routes to add dapm path to component's dapm\_routes.
   6. Add the dapm context of the component to the dapm context list of the card.