Hi Sreeju and Raj,

To optimize time to turn on WiFi/BT chip directly using either of GPIO framework and Pinctrl driver along with using DTS to pass required Pin configuration of a GPIO, I performed analysis on this topic which is mentioned as below.

1. Two frameworks are available to configure and control a given pin using DTS:
2. pinctrl: - Using pinctrl DTS property, pins are configured using Pinctrl core framework.
3. GPIOLib: -  Using GPIO DTS property for controlling GPIO(s) dynamically at run time, which is done by binding with GPIOLib framework.

|  |
| --- |
| Using pinctrl framework: - |
| Driver side: -  foo\_probe\_function()  {  ….  *devm\_pinctrl\_get();*  *pinctrl\_lookup\_state();*  *pinctrl\_select\_state();*  ….  }  - - - - - - - - - - - - - - - - - - - - - - - - - - -  DTS side: -  &foo\_device  {  ….  pinctrl-names = "default", "idle";  pinctrl-0 = <&power\_gpio\_active>;  pinctrl-1 = <&power\_gpio\_idle>;  ….  }; |

|  |
| --- |
| Using GPIOLib framework: - |
| Driver side: -  foo\_work()  {  ….  gpio\_set\_value(foo\_device->power\_gpio, 0);  …..  }  foo\_probe\_function()  {  ….  *foo\_device->power\_gpio = of\_get\_gpio(np, 0);           ret = devm\_gpio\_request\_one(foo->dev, foo\_device->power\_gpio,                                 GPIOF\_OUT\_INIT\_LOW,                                 dev\_name(foo->dev));* /\*Configure GPIO as output and Low\*/  ….  }  - - - - - - - - - - - - - - - - - - - - - - - - - - -  DTS side: -  &foo\_device  {  ….  *gpios = <&gph0 5 GPIO\_ACTIVE\_LOW>;*  ….  }; |

1. Both of the above framework calls land into same low-level SOC specific pinctrl driver.

Pinctrl DTS: -

By using dump\_stack it was found that pinctrl framework API, for example - devm\_pinctrl\_get() API, finally falls into SOC specific pinctrl driver “**drivers/pinctrl/samsung/pinctrl-samsung.c**”.

Driver's probe function - > devm\_pinctrl\_get -> pinctrl\_get -> create\_pinctrl -> pinctrl\_dt\_to\_map -> samsung\_dt\_node\_to\_map() { pinctrl-samsung.c}

GPIOLib DTS: -

Using ftrace, it was found that GPIO lib’s API like gpiod\_get\_direction(), gpiochip\_get\_data() etc falls into low-level pinctrl driver’s APIs samsung\_gpio\_get\_direction() and samsung\_gpio\_get() respectively.

Ftrace:

gpiod\_direction\_output\_raw() {

1) | gpiod\_direction\_output\_raw\_commit() {

1) | samsung\_gpio\_direction\_output() {

1) 0.599 us | gpiochip\_get\_data();

1) | samsung\_gpio\_set\_value() {

1) 0.390 us | gpiochip\_get\_data();

1) + 14.349 us | }

1) | samsung\_gpio\_set\_direction() {

1) 0.417 us | gpiochip\_get\_data();

1) 5.885 us | }

1) + 23.490 us | }

1) + 25.286 us | }

It was found that above samsung\_gpio\_get\*() APIs are defined in same SOC specific pinctrl driver “**drivers/pinctrl/samsung/pinctrl-samsung.c**”.

1. SOC specific pinctrl driver: -

It is clear that both Pinctrl and GPIOLib frameworks are serviced by same SOC specific driver “pinctrl-samsung.c”, for hardware register read and write.

If we look into code of “**pinctrl-samsung.c”,** then it can be found that it registers to different frameworks like “pinmux\_ops”, “pinconf\_ops”, “pinctrl\_ops” and also registers with “gpiolib framework”.

The driver “pinctrl-samsung.c” does not have capability to read GPIO related DTS properties and to configure it, instead it is get it done by different frameworks with whom it registers.

Hence it seems that we can not configure or control a GPIO without dts node for any device.