One Technology Way · P.O. Box 9106 · Norwood, MA 02062-9106 · Tel: 781.329.4700 · Fax: 781.461.3113 · www.analog.com

SSM2602 Sound CODEC Linux Driver

Supported Devices

- SSM2602
- SSM2603
- SSM2604

Evaluation Boards

- SSM2603-EVALZ
- SSM2604-EVALZ

Reference Circuits

- CN0282

Source Code

Status

Source	Mainlined?
git	Yes

Files

Function	File
driver	sound/soc/codecs/ssm2602.c
include	sound/soc/codecs/ssm2602.h

Example device initialization

For compile time configuration, it’s common Linux practice to keep board- and application-specific configuration out of the main driver file, instead putting it into the board support file.

For devices on custom boards, as typical of embedded and SoC-(system-on-chip) based hardware, Linux uses platform_data to point to board-specific structures describing devices and how they are connected to the SoC. This can include available ports, chip variants, preferred modes, default initialization, additional pin roles, and so on. This shrinks the board-support packages (BSPs) and minimizes board and application specific #ifdefs in drivers.

21 Oct 2010 15:10 · Michael Hennerich

SPI

SPI can be used for the SSM2602 if in SPI mode (MODE pin set to 1).

Declaring SPI slave devices

Unlike PCI or USB devices, SPI devices are not enumerated at the hardware level. Instead, the software must know which devices are connected on each SPI bus segment, and what slave selects these devices are using. For this reason, the kernel code must instantiate SPI devices explicitly. The most common method is to declare the SPI devices by bus number.

This method is appropriate when the SPI bus is a system bus, as in many embedded systems, wherein each SPI bus has a number which is known in advance. It is thus possible to pre-declare the SPI devices that inhabit this bus. This is done with an array of struct spi_board_info, which is registered by calling spi_register_board_info().

For more information see: Documentation/spi/spi-summary

21 Oct 2010 15:10 · Michael Hennerich

```
static struct spi_board_info board_spi_board_info[] __initdata = {
    [--snip--]
    {
        .modalias = "ssm2602",
        .max_speed_hz = 25000000,     /* max spi clock (SCK) speed in HZ */
        .bus_num = 0,
        .chip_select = GPIO_PF10 + MAX_CTRL_CS, /* CS, change it for your
board */
        .mode = SPI_MODE_3,
    },
    [--snip--]
```

```
};
```

```
static int __init board_init(void)
{
    [--snip--]

    spi_register_board_info(board_spi_board_info, ARRAY_SIZE(
board_spi_board_info));

    [--snip--]

    return 0;
}
arch_initcall(board_init);
```

I2C

I2C can be used for the SSM2602, SSM2603, SSM2604. For the SSM2602 make sure that it is in I2C mode (MODE pin set to 0).

Declaring I2C devices

Unlike PCI or USB devices, I2C devices are not enumerated at the hardware level. Instead, the software must know which devices are connected on each I2C bus segment, and what address these devices are using. For this reason, the kernel code must instantiate I2C devices explicitly. There are different ways to achieve this, depending on the context and requirements. However the most common method is to declare the I2C devices by bus number.

This method is appropriate when the I2C bus is a system bus, as in many embedded systems, wherein each I2C bus has a number which is known in advance. It is thus possible to pre-declare the I2C devices that inhabit this bus. This is done with an array of struct i2c_board_info, which is registered by calling i2c_register_board_info().

So, to enable such a driver one need only edit the board support file by adding an appropriate entry to i2c_board_info.

For more information see: Documentation/i2c/instantiating-devices

21 Oct 2010 15:10 · Michael Hennerich

```
static struct i2c_board_info __initdata bfin_i2c_board_info[] = {

    [--snip--]
    {
        I2C_BOARD_INFO("ssm2604", 0x1b),
```

```
    },
    [--snip--]
}
```

```
static int __init stamp_init(void)
{
    [--snip--]
    i2c_register_board_info(0, bfin_i2c_board_info,
                ARRAY_SIZE(bfin_i2c_board_info));
    [--snip--]

    return 0;
}
arch_initcall(board_init);
```

ASoC DAPM Widgets

Name	Description	Model
LOUT	Line Output for Left Channel	SSM2602, SSM2603, SSM2604
ROUT	Line Output for Right Channel	SSM2602, SSM2603, SSM2604
LLINEIN	Line Input for Left Channel	SSM2602, SSM2603, SSM2604
RLINEIN	Line Input for Right Channel	SSM2602, SSM2603, SSM2604
LHPOUT	Headphone Output for Left Channel	SSM2602, SSM2603
RHPOUT	Headphone Output for Right Channel	SSM2602, SSM2603
MICIN	Microphone Input Signal	SSM2602, SSM2603

ALSA Controls

Name	Description	Model
Capture Volume	Line Input PGA Volume	SSM2602, SSM2603, SSM2604
Capture Switch	Mute/Unmute Line Input signal	SSM2602, SSM2603, SSM2604
ADC High Pass Filter Switch	Enable/Disable ADC high-pass filter	SSM2602, SSM2603, SSM2604
Store DC Offset Switch	Store dc offset when high-pass filter is disabled	SSM2602, SSM2603, SSM2604
Playback De-emphasis	Select playback de-emphasis. Possible values: “None”, “32Khz”, “44.1Khz”, “48Khz”	SSM2602, SSM2603, SSM2604
Master Playback Volume	Headphone volume	SSM2602, SSM2603
Master Playback ZC Switch	Enable/Disable zero cross detection for the playback volume	SSM2602, SSM2603
Sidetone Playback Volume	Microphone sidetone gain control	SSM2602, SSM2603

Name	Description	Model
Mic Boost (+20dB)	Primary microphone amplifier gain booster control.	SSM2602, SSM2603
Mic Boost2 (+20dB)	Additional microphone amplifier gain booster control.	SSM2602, SSM2603
Mic Switch	Mute/Unmute the Microphone signal	SSM2602, SSM2603
Output Mixer Line Bypass Switch	Mix Line Input signal into the output signal	SSM2602, SSM2603, SSM2604
Output Mixer HiFi Playback Switch	Mix DAC signal into the output signal	SSM2602, SSM2603, SSM2604
Output Mixer Mic Sidetone Switch	Mix Microphone signal into the output signal	SSM2602, SSM2603
Input Select	Select the ADC input signal. Possible values: “Line”, “Mic”	SSM2602, SSM2603

DAI configuration

The codec driver registers one DAI named **“ssm2602-hifi”**.

Supported DAI formats

Name	Supported by driver	Description
SND_SOC_DAIFMT_I2S	yes	I2S Justified mode
SND_SOC_DAIFMT_RIGHT_J	yes	Right Justified mode
SND_SOC_DAIFMT_LEFT_J	yes	Left Justified mode
SND_SOC_DAIFMT_DSP_A	yes	data MSB after FRM LRC
SND_SOC_DAIFMT_DSP_B	yes	data MSB during FRM LRC
SND_SOC_DAIFMT_AC97	no	AC97 mode
SND_SOC_DAIFMT_PDM	no	Pulse density modulation
SND_SOC_DAIFMT_NB_NF	yes	Normal bit- and frameclock
SND_SOC_DAIFMT_NB_IF	yes	Normal bitclock, inverted frameclock
SND_SOC_DAIFMT_IB_NF	yes	Inverted frameclock, normal bitclock
SND_SOC_DAIFMT_IB_IF	yes	Inverted bit- and frameclock
SND_SOC_DAIFMT_CBM_CFM	yes	Codec bit- and frameclock master
SND_SOC_DAIFMT_CBS_CFM	no	Codec bitclock slave, frameclock master
SND_SOC_DAIFMT_CBM_CFS	no	Codec bitclock master, frameclock slave
SND_SOC_DAIFMT_CBS_CFS	yes	Codec bit- and frameclock slave

Supported SYSCLK rates

The codecs system clock can be configured for various input rates. When configuring the codec system clock use SSM2602_SYSCLK for the clock id.

The following list contains the supported system clock rates and their resulting sample-rates.

SYSCLK	Supported sample-rates
11289600	8kHz, 44.1kHz 88.2kHz
12000000	8kHz, 32kHz, 44.1kHz 48kHz, 88.2kHz, 96kHz
12288000	8kHz, 32kHz, 48kHz, 96kHz
16934400	8kHz, 44.1kHz, 88.2kHz
18432000	8kHz, 32kHz, 48kHz, 96kHz

Example DAI configuration

```
static struct snd_soc_card bf5xx_ssm2602;

static int bf5xx_ssm2602_hw_params(struct snd_pcm_substream *substream,
    struct snd_pcm_hw_params *params)
{
    struct snd_soc_pcm_runtime *rtd = substream->private_data;
    struct snd_soc_dai *codec_dai = rtd->codec_dai;
    struct snd_soc_dai *cpu_dai = rtd->cpu_dai;
    int ret;

    ret = snd_soc_dai_set_fmt(cpu_dai, SND_SOC_DAIFMT_I2S |
        SND_SOC_DAIFMT_NB_NF | SND_SOC_DAIFMT_CBM_CFM);
    if (ret < 0)
        return ret;

    ret = snd_soc_dai_set_fmt(codec_dai, SND_SOC_DAIFMT_I2S |
        SND_SOC_DAIFMT_NB_NF | SND_SOC_DAIFMT_CBM_CFM);
    if (ret < 0)
        return ret;

    ret = snd_soc_dai_set_sysclk(codec_dai, SSM2602_SYSCLK, 12000000,
        SND_SOC_CLOCK_IN);
    if (ret < 0)
        return ret;

    return 0;
}

static struct snd_soc_ops bf5xx_ssm2602_ops = {
    .hw_params = bf5xx_ssm2602_hw_params,
};

static struct snd_soc_dai_link bf5xx_ssm2602_dai_link[] = {
    {
        .name = "ssm2602",
        .stream_name = "SSM2602",
```

```
        .cpu_dai_name = "bfin-i2s.0",
        .codec_dai_name = "ssm2602-hifi",
        .platform_name = "bfin-i2s-pcm-audio",
        .codec_name = "ssm2602.0-001b",
        .ops = &bf5xx_ssm2602_ops,
    },
};
```

SSM2604 evaluation board driver

Source

Status

Source	Mainlined?
git	Yes

As a Module

To add support for the built-in codec SSM2602 of BF52xC to the kernel build system, a few things must be enabled properly for things to work. The configuration is as following:

```
Linux Kernel Configuration
  Device Drivers  --->
      <M> Sound card support --->
        <M> Advanced Linux Sound Architecture  --->
          < > Sequencer support
          <M> OSS Mixer API
          <M> OSS PCM (digital audio) API
          <M> ALSA for SoC audio support  --->
            <M> SoC I2S Audio for the ADI BF5xx chip
              <M> SoC SSM2602 Audio support for BF52x ezkit
            < > SoC AC97 Audio support for BF5xx
            (0) Set a SPORT for Sound chip
```

I2C bus is used to configure the codec. So, if the audio driver is built into kernel, the I2c driver is also built into kernel automatically. But if the audio driver is built as module, then make sure that the I2C driver is loaded before the audio module.

```
Linux Kernel Configuration
  Device Drivers  --->
    <*> I2C support  --->
      --- I2C support
          I2C Hardware Bus Support --->
            <*> Blackfin TWI I2C support
```

Doing this will create modules (outside the kernel). The modules will be inserted automatically when it is needed. You can also build sound driver into kernel.

Testing the built in kernel driver

If audio is configured as modules, skip this section. If audio is built into kernel and you have booted the kernel, there are a few things to check to ensure audio is working:

1. Check the boot messages to see if you have booted the correct kernel. During kernel boot, it should print out: Advanced Linux Sound Architecture Driver Version 1.0.12rc1 (Thu Jun 22 13:55:50 2006 UTC).
   ASoC version 0.13.1
   dma rx:3 tx:4, err irq:15, regs:ffc00800
   ssm2602 Audio Codec 0.1<6>dma_alloc_init: dma_page @ 0x03011000 - 512 pages at 0x03e00000
   asoc: SSM2602 <-> bf5xx-i2s-0 mapping ok
   ALSA device list:
    #0: bf5xx_ssm2602 (SSM2602)

Testing the audio module

```
root:~> modprobe snd-ssm2602
root:~> modprobe snd-pcm-oss
root:~> lsmod
Module                  Size  Used by
snd_pcm_oss            31968  0
snd_mixer_oss          11360  1 snd_pcm_oss
snd_ssm2602             1412  0
snd_soc_ssm2602         8528  1 snd_ssm2602
snd_soc_bf5xx           2784  1 snd_ssm2602
snd_soc_bf5xx_i2s      10916  2 snd_ssm2602,snd_soc_bf5xx
snd_soc_core           17120  3 snd_ssm2602,snd_soc_ssm2602,snd_soc_bf5xx
snd_pcm                48356  3 snd_pcm_oss,snd_soc_bf5xx,snd_soc_core
```

```
snd_page_alloc          4232  1 snd_pcm
snd_timer              13796  1 snd_pcm
snd                    31092  6
snd_pcm_oss,snd_mixer_oss,snd_soc_ssm2602,snd_soc_core,snd_pcm,snd_timer
soundcore               3940  1 snd

root:~> tone
TONE: generating sine wave at 1000 Hz...
```

Testing Audio

1. Check the output root:~> **tone**
   TONE: generating sine wave at 1000 Hz...
   You should hear something out of the headphone Jack on the top of J8.
2. Set the audio mixer to Mic (the default is Line, assuming you have built ALSA utils): root:/> **amixer sset 'Input Mux' 'Mic'**
   Simple mixer control 'Input Mux',0
   Capabilities: enum
   Items: 'Line' 'Mic'
   Item0: 'Mic' Also you can run “alsamixer” to get graphic configuration interface, OSS-based “mixer” can work too.
3. Check to make sure mp3s work (assuming you have built mp3play),
   1. The first step is to download a mp3 file onto the platform. The `wget` command assumes that networking is properly configured (you have an IP number, the default gateway is set, and DNS servers can be accessed), and working. See the network setup page for more info. root:/> **cd /var**
      root:/var> **wget http://www.radiocrazy.com/shows/A/AbbottCostello/ABCOWhosOnFirstclip.mp3**
   2. Next, play it with mp3play: root:/var> **mp3play ABCOWhosOnFirstclip.mp3**
4. You can play it in one step with: root:~> **mp3play http://www.radiocrazy.com/shows/A/AbbottCostello/ABCOWhosOnFirstclip.mp3**
   http://www.radiocrazy.com/shows/A/AbbottCostello/ABCOWhosOnFirstclip.mp3: MPEG2-III (0 ms)
5. Optionally check to make sure the Line and headphone are working properly: root:/> **amixer sset 'Input Mux' 'Line'**
   Simple mixer control 'Input Mux',0
   Capabilities: enum
   Items: 'Line' 'Mic'
   Item0: 'Line'
   root:~> **arecord -d 10 test.wav**
   Recording WAVE "test.wav" : Unsigned 8 bit, Rate 8000 Hz, Mono
   root:~> **aplay test.wav**
   This should record 10 seconds of whatever is on the Line, and then play it back over the output.
6. You should also be able to do a “talkthrough”, and hear on the speakers anything you put on the line. root:~> **arecord | aplay**

Debugging

Most of the time, and issues we have seen have been tracked down to proper switch settings, which are detailed on bf527-ezkit getting started.