Tina Linux AVS
使用指南

版本号: 1.1
发布日期: 2020-09-04

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.07.02	AWA1379	创建
1.1	2020.02.19	AWA1402	重构 AVS 部分相关软件包的代码，更新配置文件和软件包路径，与之前的版本不兼容

目　录

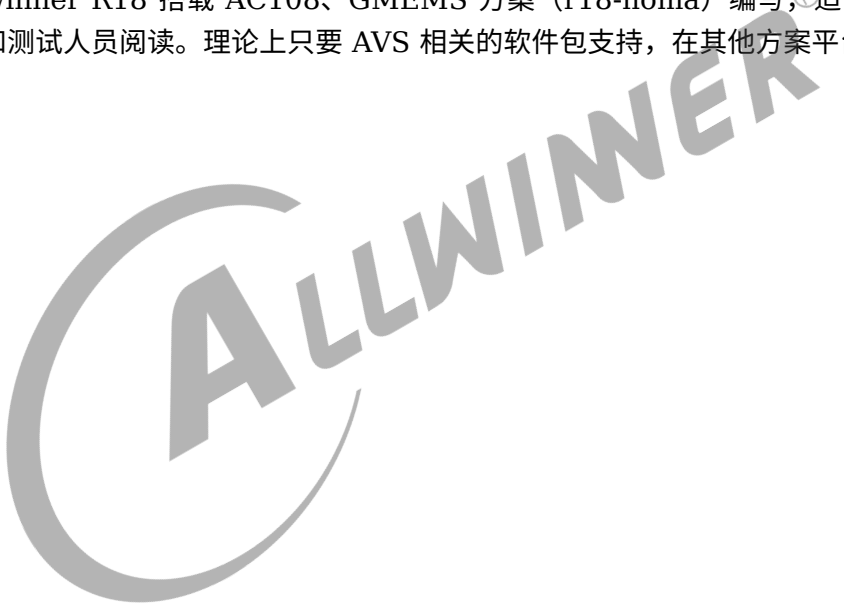
1 概述

1.1 编写目的

介绍 TinaLinux 平台上 AVS SDK 的简易使用，实现 Alexa Demo 开发。

1.2 适用范围

文档基于 Allwinner R18 搭载 AC108、GMEMS 方案（r18-noma）编写，适合 TinaLinux 的 AVS 开发和测试人员阅读。理论上只要 AVS 相关的软件包支持，在其他方案平台上也可用。



2 整体介绍

2.1 涉及到的软件包

要运行一个 Alexa Demo，主要涉及到以下软件包（package）：

- **avs-device-sdk**：从 GitHub 上下载下来的 AVS SDK 官方源码包。
- **avs-sampleapp**：源码来自于 avs-device-sdk 中的 SampleApp，在其中做了一定的修改以对接设备端。是 Alexa Demo 最顶层的软件入口。
- **avs-fetch-device-sn**：用于获取设备的 serial number，供 avs-sampleapp 使用。
- **libpryon-lite**：亚马逊提供的唤醒词识别算法库。
- **tutuclear-lib**：GMEMS 的音频前端处理算法库。
- **tconfigs**：以 JSON 进行配置的中间件，供 avs-sampleapp 使用，方便其适配不同的硬件平台。

Alexa Demo 的运行还依赖于其他一些软件包（如播放依赖 GStreamer），详细请查看上述软件包中 Makefile 的依赖关系。

2.2 avs-device-sdk

avs-device-sdk 是对接 Alexa 服务的 SDK，由亚马逊负责开发维护，在 GitHub 上发布。作为设备端对接，只需要关注三件事情：

1. AFE（音频采集处理）输出的数据灌入 Shared Data Stream。
2. WakeWord Detection 对接对应的唤醒算法。SDK 中已经对接好 Sensory、Kitt.AI 等一些唤醒算法，如果是使用它们，只要把对应的算法实现放进入，指定路径就可以使用。
3. SDK 处理好云端返回来的结果，通过 Gstreamer 去播放。GStreamer 播放的兼容性，需要设备端解决。

在 Tina 中，avs-device-sdk 是直接使用从 GitHub 上下载的源码，只针对设备端的使用做了以下改动：

- 为解决在 Tina 环境中的编译报错问题，将 MediaPlayer 中的部分 `NULL` 改为 `nullptr`。
- 加入 AmazonLite（即 libpryon-lite）的 adapter 来对接它的唤醒算法。

其余更多对接设备端做的改动放在 avs-sampleapp 进行。

2.3 avs-sampleapp

avs-sampleapp 是将 avs-device-sdk 官方源码中的 SampleApp 抽取出来后，再加入与设备端对接的改动而成。

它的软件包目录的 configs 文件夹中有当前支持的平台的配置文件，以 r18-noma 为例：

```
r18-noma
├── AlexaClientSDKConfig.json
├── tconfigs.d
│   ├── default.json
│   └── dump_udisk.json
├── tconfigs.json
├── tutuClearA1_ns4wakeup_mono.prm
└── tutuClearA1_ns4wakeup_stereo.prm
```

- AlexaClientSDKConfig.json：AVS SDK 的配置文件。
- tconfigs.json 和 tconfigs.d：tconfigs 的配置文件，设备端不同方案平台的差异在里面配置。avs-sampleapp 运行时会寻找 AlexaClientSDKConfig.json 所在同一目录下的 tconfigs.json，在 tconfigs.json 里指定选用 tconfigs.d 里的哪一个配置文件。
- tutuClearA1_xxx.prm：tutuclear-lib 的配置文件，选择哪一个来使用是在 tconfigs 的配置文件中指定。

2.4 avs-fetch-device-sn

获取设备的 serial number，写入到 AlexaClientSDKConfig.json 的 `deviceSerialNumber` 中（因此需要确保 AlexaClientSDKConfig.json 是可写的）。

serial number 需要是设备唯一的，在 AVS SDK 的授权认证阶段使用。

2.5 libpryon-lite

亚马逊的唤醒词识别算法库，由亚马逊维护和授权使用。

2.6 tutuclear-lib

tutuclear-lib 是 GMEMS 的音频前端处理算法库，主要的功能有：

1. 噪声压制
2. 回音消除
3. 识别音频流输出，适合机器做识别，用于云端识别
4. 识别音频流输出，没有加任何的数字增益，适合做能量检测，用于 ESP 处理
5. 通话音频流输出，适合人耳听，用于通话
6. DOA 信息输出

2.6.1 配置

tutuclear-lib 通过 prm 文件配置，在库初始化时，把 prm 文件解析成数据传给库。

2.7 tconfigs

tconfigs 是全志提供的以 JSON 进行配置的中间件，用来解决平台差异化问题，目的是通过配置文件去让一套应用代码适配多个方案平台。目前支持音频采集通路和按键检测的配置，以及与 GMEMS 算法的对接。

2.7.1 tconfigs 配置文件说明

以 avs-sampleapp 中 r18-noma 的配置文件 default.json 和 dump_udisk.json 为例子。

2.7.1.1 default.json

```
{
    // “record_pipeline”为整个录音的通路，这个名字会在 avs-sampleapp 中使用。
    // 其中 pipeline、src、sink 等都是借鉴 GStreamer 的概念。
    // 一个 pipeline 由多个 element 连接而成，element 相互间通过 pad 传输音频数据，
    // 数据从一个 element 的 src pad 流出，流向下一个 element 的 sink pad
    "record_pipeline" : {
        // “engine”表示由哪一个 element 控制整个 pipeline 的数据流动，
        // 一般选择数据源头的 element 作为 engine
        "engine" : "alsa_src",
        // “elements”中括起来的即表示这个 pipeline 由哪些 element 组成。
        // “element_type”、“src_pads”、“sink_pads”这些关键字在不同 element 中都是通用的
        "elements" : {
            "alsa_src" : {  // element 的名字，可以自行定义名字
                // “element_type”表明该 element 的类型，每一个 element 都有这个关键字。
```

```
        //  “AlsaSrc” 是可从 ALSA 设备中获取音频数据的 element
        "element_type" : "AlsaSrc",
        //  “src_pads” 表明这个 element 有哪些 src pad。
        //  (如果一个 element 没有这个关键字表明它没有 src pad)
        "src_pads" : {
            // src pad 的名字，有些 element 可以自定义 pad 的名字，但有些不行。
            // 此处 AlsaSrc 的 src pad 可以自定义名字
            "capture_both" : {
                //  “peer_element” 表明该 src pad 连接到哪个 element 的 sink pad
                "peer_element" : "converter",
                //  “peer_pad” 表明该 src pad 连接到 peer_element 的哪个 sink pad
                "peer_pad" : "from_alsa_src"
            }
        },
        //  “devices” 关键字表明 AlsaSrc 使用的录音设备配置
        "devices" : {
            // 此处“capture_both” 是给 AlsaSrc 使用的名字，可以自行定义，
            // 与实际使用的 ALSA 设备名字无关
            "capture_both" : {
                "device" : "hw:sndac10810035",  // 打开的 ALSA 设备名字
                "loop_frames" : 160,            // 每次循环处理多少 frame 的数据
                "access" : "RW_INTERLEAVED",    // 数据传输的方式
                "format" : "S16_LE",            // 采样精度
                "rate" : 16000,                 // 采样率
                "channels" : 8,                 // 通道数
                "period_frames" : 1024,         // 即 ALSA 的参数 period_size
                "periods" : 4                   // 即 ALSA 的参数 periods
            }
        }
    },
    "converter" : {
        //  “Converter” 类型的 element 可用于改变数据的存储方式（交错/非交错），
        // 并可将某一通道的数据拷贝到另一通道
        "element_type" : "Converter",
        //  “from_alsa_src” 是该 sink pad 的名字，可以自行定义。
        // 因为在上面 “alsa_src” 的配置中已经指定它名为 “capture_both” 的 src pad 是
        // 连接到 “converter” 的名为 “from_alsa_src” 的 sink pad，因此此处可将
        // sink pad 的 “peer_element” 和 “peer_pad” 省略掉
        "sink_pads" : { "from_alsa_src" : {} },
        // 此处定义了 “common_out” 和 “reference_out” 两个 src pad，
        // 它们的名字可自行定义
        "src_pads" : {
            "common_out" : {
                //  “channel” 表示从 “common_out” 输出的通道数，此处为 3 通道
                "channels" : 3,
                //  “storage” 表示从 “common_out” 输出的数据的存储方式，
                // 可以为交错（interleaved）/非交错（noninterleaved）
                "storage" : "noninterleaved",
                "peer_element" : "tutuclear",
                "peer_pad" : "common"
            },
            "reference_out" : {
                // 从 “reference_out” 输出的通道数为 2
                "channels" : 2,
                // 从 “reference_out” 输出的数据以非交错的方式存储
                "storage" : "noninterleaved",
                "peer_element" : "tutuclear",
                "peer_pad" : "reference"
            }
        },
```

```
            //  “channel_map” 表示 Converter 将哪一个 sink pad 的什么通道的数据拷贝到
            // 哪一个 src pad 的什么通道，格式为：
            //   "src_pad_name.channel_index" : "sink_pad_name.channel_index"
            //
            // 例如，"common_out.0" : "from_alsa_src.1" 表示：
            //  “common_out” 的 channel 0 的数据来自于 “from_alsa_src” 的 channel 1;
            // "reference_out.1" : "from_alsa_src.7" 表示：
            //  “reference_out” 的 channel 1 的数据来自于 “from_alsa_src” 的 channel 7;
            // 如此类推。
            //
            // 如果对应的 sink pad 与 src pad 的数据存储方式（交错/非交错）不相同，
            // Converter 内部会自动进行转换。
            "channel_map" : {
                "common_out.0" : "from_alsa_src.1",
                "common_out.1" : "from_alsa_src.3",
                "common_out.2" : "from_alsa_src.5",
                "reference_out.0" : "from_alsa_src.6",
                "reference_out.1" : "from_alsa_src.7"
            }
        },
        "tutuclear" : {
            //  “TutuclearElement” 类型的 element 用于对接 GMEMS 的 tutuclear 算法
            "element_type" : "TutuclearElement",
            "sink_pads" : {
                // 此处 sink pad 的名字固定为 “common” 和 “reference”，
                // 通道数的配置看实际算法库而定，由 prm 文件控制
                "common" : { "channels" : 3 },
                "reference" : { "channels" : 2 }
            },
            "src_pads" : {
                "output" : {
                    "channels" : 3,
                    "peer_element" : "tutuclear_output_converter",
                    "peer_pad" : "from_tutuclear"
                }
            },
            // 指定 prm 文件的路径
            "prm_file" : "/etc/avs/tutuClearA1_ns4wakeup_stereo.prm",
            // 采样精度
            "format" : "S16_LE",
            // 采样率
            "rate" : 16000,
            // 每次循环处理多少 frame 的数据。
            // GMEMS 算法要求每次循环处理 10ms 的数据，因此采样率为 16000Hz 时，为：
            //   16000 * 0.01 = 160
            "loop_frames" : 160,
            // GMEMS 算法要求输入输出数据的存储方式均为非交错
            "storage" : "noninterleaved"
        },
        // 此处定义一个 Converter 将 “tutuclear” 输出数据中给云端识别用的通道取出来，
        // 并转成非交错的存储方式，给 AVS SDK 使用
        "tutuclear_output_converter" : {
            "element_type" : "Converter",
            "sink_pads" : { "from_tutuclear" : {} },
            "src_pads" : {
                "output" : {
                    "channels" : 1,
                    "format" : "S16_LE",
                    "storage" : "interleaved",
                    "peer_element" : "common_sink",
```

```
                    "peer_pad" : "from_tutuclear_output_converter"
                }
            },
            "channel_map" : {
                "output.0" : "from_tutuclear.0"
            }
        },
        //  “common_sink”的名字会在 avs-sampleapp 中使用
        "common_sink" : {
            //  “CommonSink”类型的 element 单纯只接收音频数据，不做额外处理，
            // 通常放在一个 pipeline 的结尾，当收到数据时可发送一个 signal，
            // 通知 pipeline 外部已获取到数据
            "element_type" : "CommonSink",
            "sink_pads" : { "from_tutuclear_output_converter" : {} },
            //  “data_got_signal”表示收到音频数据时发送什么信号，
            // 此处信号的名字为“DataGot”，在 avs-sampleapp 中会用上
            "data_got_signal" : "DataGot"
        }
    }
},

//  “key_manager”用于配置按键，这个名字会在 avs-sampleapp 中使用。
"key_manager" : {
    //  “built_in_executor_threads”表示使用多少个线程处理按键的回调函数。
    // 如果使用的线程数为 1，则上一个按键的回调函数执行完之前，不会响应其他按键的回调函数。
    "built_in_executor_threads" : 2,
    //  “behaviors”表示有哪些按键行为，其名字和对应的回调函数都在 avs-sampleapp 中定义。
    // 当前支持的按键行为有“VolumeUp”（音量增）、“VolumeDown”（音量减）和“Mute”（静音）
    "behaviors" : {
        // 音量增
        "VolumeUp" : {
            // 此处“SW1”为按键的名字，可以自行定义。
            // 如果单个 behavior 中定义了多个按键名字，表示该行为由组合按键触发。
            "SW1" : {
                //  “input_device”表示该按键对应哪一个输入设备，
                // 名字要与 /sys/class/input/inputXX/name 中的相同
                "input_device" : "sunxi-keyboard",
                // 按键的键值
                "code" : 115,
                //  “motion”为需要检测的按键的动作，当前支持：
                //    “Press”：按下
                //    “Release”：释放
                //    “LongPressPreRelease”：长按一定时间，在释放前响应
                //    “LongPressPostRelease”：长按一定时间，在释放后响应
                // 长按的动作还会多一个配置项“duration_sec”表示长按的时间，数值为浮点型
                "motion" : "Press"
            }
        },
        // 音量减
        "VolumeDown" : {
            "SW2" : {
                "input_device" : "sunxi-keyboard",
                "code" : 114,
                "motion" : "Press"
            }
        },
        // 静音
        "Mute" : {
            "SW3" : {
                "input_device" : "sunxi-keyboard",
```

```
                    "code" : 113,
                    "motion" : "Press"
                }
            }
        }
    }
}
```

2.7.1.2 dump_udisk.json

dump_udisk.json 和 default.json 的配置项大体相同，只是增加了将 ALSA 录音数据和 GMEMS 处理结果数据保存到文件中的功能。大体的做法是将需要保存的数据使用 Converter 拷贝一份，传输给 DataQueue 缓冲，最后送到 FileSink 写入到文件中。

```
{
    "record_pipeline" : {
        "engine" : "alsa_src",
        "elements" : {

            "alsa_src" : {
                "element_type" : "AlsaSrc",
                "src_pads" : {
                    "capture_both" : {
                        "peer_element" : "converter",
                        "peer_pad" : "from_alsa_src"
                    }
                },
                "devices" : {
                    "capture_both" : {
                        "device" : "hw:sndac10810035",
                        "loop_frames" : 160,
                        "access" : "RW_INTERLEAVED",
                        "format" : "S16_LE",
                        "rate" : 16000,
                        "channels" : 8,
                        "period_frames" : 1024,
                        "periods" : 4
                    }
                }
            },

            // “converter”增加了“common_out_copy”和“reference_out_copy”两个 src pad,
            // 相当于将原先“common_out”和“reference_out”的数据拷贝一份，然后传输给
            // “common_dump_queue”和“reference_dump_queue”这两个 element。
            "converter" : {
                "element_type" : "Converter",
                "sink_pads" : { "from_alsa_src" : {} },
                "src_pads" : {
                    "common_out" : {
                        "channels" : 3,
                        "storage" : "noninterleaved",
                        "peer_element" : "tutuclear",
                        "peer_pad" : "common"
                    },
                    "reference_out" : {
```

```
                "channels" : 2,
                "storage" : "noninterleaved",
                "peer_element" : "tutuclear",
                "peer_pad" : "reference"
            },
            "common_out_copy" : {
                "channels" : 3,
                "storage" : "interleaved",
                "peer_element" : "common_dump_queue",
                "peer_pad" : "from_converter_common_out_copy"
            },
            "reference_out_copy" : {
                "channels" : 2,
                "storage" : "interleaved",
                "peer_element" : "reference_dump_queue",
                "peer_pad" : "from_converter_reference_out_copy"
            }
        },
        "channel_map" : {
            "common_out.0" : "from_alsa_src.1",
            "common_out.1" : "from_alsa_src.3",
            "common_out.2" : "from_alsa_src.5",
            "reference_out.0" : "from_alsa_src.6",
            "reference_out.1" : "from_alsa_src.7",

            "common_out_copy.0" : "from_alsa_src.1",
            "common_out_copy.1" : "from_alsa_src.3",
            "common_out_copy.2" : "from_alsa_src.5",
            "reference_out_copy.0" : "from_alsa_src.6",
            "reference_out_copy.1" : "from_alsa_src.7"
        }
    },

    "tutuclear" : {
        "element_type" : "TutuclearElement",
        "sink_pads" : {
            "common" : { "channels" : 3 },
            "reference" : { "channels" : 2 }
        },
        "src_pads" : {
            "output" : {
                "channels" : 3,
                "peer_element" : "tutuclear_output_converter",
                "peer_pad" : "from_tutuclear"
            }
        },
        "prm_file" : "/etc/avs/tutuClearA1_ns4wakeup_stereo.prm",
        "format" : "S16_LE",
        "rate" : 16000,
        "loop_frames" : 160,
        "storage" : "noninterleaved"
    },

    // “tutuclear_output_converter”中同样增加一个 src pad “output_copy”
    // 拷贝 GMEMS 算法输出的数据，传输给“tutuclear_dump_queue”
    "tutuclear_output_converter" : {
        "element_type" : "Converter",
        "sink_pads" : { "from_tutuclear" : {} },
        "src_pads" : {
            "output" : {
```

```
                "channels" : 1,
                "format" : "S16_LE",
                "storage" : "interleaved",
                "peer_element" : "common_sink",
                "peer_pad" : "from_tutuclear_output_converter"
            },
            "output_copy" : {
                "channels" : 1,
                "format" : "S16_LE",
                "storage" : "interleaved",
                "peer_element" : "tutuclear_dump_queue",
                "peer_pad" : "from_tutuclear_output_converter"
            }
        },
        "channel_map" : {
            "output.0" : "from_tutuclear.0",
            "output_copy.0" : "from_tutuclear.0"
        }
    },

    "common_sink" : {
        "element_type" : "CommonSink",
        "sink_pads" : { "from_tutuclear_output_converter" : {} },
        "data_got_signal" : "DataGot"
    },

    // 以下“DataQueue”类型的 element 的作用是缓存数据，
    // 避免音频数据写入文件时速度太慢而影响原先的录音流程。
    // 不同于一般的 element，DataQueue 的 sink pad 接收数据与
    // src pad 发送数据是异步的，它们在不同的线程中执行：
    // 只要 DataQueue 内部缓冲的 buffer 没有满，sink pad 接收数据后会马上返回；
    // 而 src pad 是只要 buffer 中存在数据，就会持续发送。
    "common_dump_queue" : {
        "element_type" : "DataQueue",
        // “max_buffers”为 DataQueue 内部缓冲 buffer 的大小，
        // 表示最大可缓冲多少个循环周期的数据
        "max_buffers" : 20,
        "sink_pads" : { "from_converter_common_out_copy" : {} },
        "src_pads" : {
            "output" : {
                "peer_element" : "common_dump",
                "peer_pad" : "from_common_dump_queue"
            }
        }
    },
    "reference_dump_queue" : {
        "element_type" : "DataQueue",
        "max_buffers" : 20,
        "sink_pads" : { "from_converter_reference_out_copy" : {} },
        "src_pads" : {
            "output" : {
                "peer_element" : "reference_dump",
                "peer_pad" : "from_reference_dump_queue"
            }
        }
    },
    "tutuclear_dump_queue" : {
        "element_type" : "DataQueue",
        "max_buffers" : 20,
        "sink_pads" : { "from_tutuclear_output_converter" : {} },
```

```
                "src_pads" : {
                    "output" : {
                        "peer_element" : "tutuclear_dump",
                        "peer_pad" : "from_tutuclear_dump_queue"
                    }
                }
            },

            // 以下“FileSink”类型的 element 用于将音频数据保存到文件中
            "common_dump" : {
                "element_type" : "FileSink",
                // 保存的文件类型，支持“wav”或“pcm”
                "type" : "wav",
                // 保存的文件路径
                "path" : "/mnt/UDISK/common.wav",
                "sink_pads" : { "from_common_dump_queue" : {} }
            },
            "reference_dump" : {
                "element_type" : "FileSink",
                "type" : "wav",
                "path" : "/mnt/UDISK/reference.wav",
                "sink_pads" : { "from_reference_dump_queue" : {} }
            },
            "tutuclear_dump" : {
                "element_type" : "FileSink",
                "type" : "wav",
                "path" : "/mnt/UDISK/tutu_out.wav",
                "sink_pads" : { "from_tutuclear_dump_queue" : {} }
            }
        }
    },

    "key_manager" : {
        "built_in_executor_threads" : 2,
        "behaviors" : {
            "VolumeUp" : {
                "SW1" : {
                    "input_device" : "sunxi-keyboard",
                    "code" : 115,
                    "motion" : "Press"
                }
            },
            "VolumeDown" : {
                "SW2" : {
                    "input_device" : "sunxi-keyboard",
                    "code" : 114,
                    "motion" : "Press"
                }
            },
            "Mute" : {
                "SW3" : {
                    "input_device" : "sunxi-keyboard",
                    "code" : 113,
                    "motion" : "Press"
                }
            }
        }
    }
}
```

3 编译配置

menuconfig 配置如下：

首先在 Thirdparty ---> AVS 中选上 avs-fetch-device-sn 和 avs-sampleapp，此时其他大多数依赖的软件包会自动选上：

```
Allwinner --->
    -*- tconfigs
Thirdparty --->
    AVS --->
        -*- avs-device-sdk
        <*> avs-fetch-device-sn
        <*> avs-sampleapp
    -*- libpryon-lite
    -*- tutuclear-lib
```

因为 AVS SDK 播放使用 GStreamer，需要把 Multimedia 中 GStreamer 各个 plugin 的模块手动选上（plugin 整体的 package 已经被自动选上，但里面的 modules 需要手动选上。其余没有被自动选上的 plugin package（如 ugly）可以不用管）：

```
-*- gst1-libav..................................... GStreamer Libav plugin
      Select GStreamer libav configuration options  --->
-*- gstreamer1-libs.............................. GStreamer core libraries
      Select GStreamer libraries  --->
< > gstreamer1-plugins-aw.................. GStreamer plugins collection (aw)
-*- gstreamer1-plugins-bad................. GStreamer plugins collection (bad)
      Select GStreamer bad modules and libraries  --->
-*- gstreamer1-plugins-base.............. GStreamer plugins collection (base)
      Select GStreamer base modules and libraries  --->
-*- gstreamer1-plugins-good.............. GStreamer plugins collection (good)
      Select GStreamer good modules  --->
< > gstreamer1-plugins-ugly.............. GStreamer plugins collection (ugly)
< > gstreamer1-utils................................... GStreamer utilities
```

gst1-libav 的 Select GStreamer libav configuration options 维持默认值即可：

```
[ ] Enable IPv6 (NEW)
[*] Include patented codecs and technologies (NEW)
[*] Include support for common audio/video decoders (NEW)
    *** Encoders --- ***
[ ] AC3 (NEW)
[ ] JPEG-LS (NEW)
```

```
[ ] MPEG-1 Video (NEW)
[ ] MPEG-2 Video (NEW)
[ ] MPEG-4 (NEW)
[ ] PCM signed 16-bit big-endian (NEW)
[ ] PCM signed 16-bit little-endian (NEW)
[ ] PNG (NEW)
[ ] Vorbis (NEW)
[ ] Zlib (NEW)
    *** Decoders --- ***
-*- AAC (Advanced Audio Coding)
-*- AC3
-*- ATRAC3
[ ] GIF (NEW)
-*- H.264
-*- JPEG-LS
[ ] MP2 (MPEG Audio Layer 2) (NEW)
-*- MP3 (MPEG Audio Layer 2)
-*- MPEG Video
-*- MPEG-1 Video
-*- MPEG-2 Video
-*- MPEG-4
-*- MPEG-4 (AAC)
-*- PCM signed 16-bit big-endian
-*- PCM signed 16-bit little-endian
-*- PNG
-*- Vorbis
-*- WMAv1
-*- WMAv2
-*- Zlib
    *** Muxers --- ***
[ ] AC3 (NEW)
[ ] FFM (ffserver live feed) (NEW)
[ ] H.264 (NEW)
[ ] MP3 (MPEG Audio Layer 3) (NEW)
[ ] MP4 (NEW)
[ ] MPEG-1 Video (NEW)
[ ] MPEG-2 Video (NEW)
[ ] MPEG-2 (TS) (NEW)
[ ] Ogg (NEW)
[ ] OSS (Open Sound System playback) (NEW)
[ ] RTP (NEW)
    *** Demuxers --- ***
-*- AC3
[ ] FFM (ffserver live feed) (NEW)
-*- H.264
-*- MP3 (MPEG Audio Layer 3)
-*- MPEG Video
[ ] MPEG-2 (PS) (NEW)
[ ] MPEG-2 (TS) (NEW)
-*- Ogg
[ ] RM (NEW)
[ ] RTSP (NEW)
[ ] SDP (NEW)
    *** Parsers --- ***
-*- AAC (Advanced Audio Coding)
-*- AC3
-*- H.264
-*- MPEG Audio
-*- MPEG Video
-*- MPEG-4 Video
```

```
    *** Protocols --- ***
[ ] file: (NEW)
[ ] http: (NEW)
[ ] pipe: (NEW)
[ ] rtp: (NEW)
[ ] tcp: (NEW)
[ ] udp: (NEW)
```

gstreamer1-libs 的 `Select GStreamer libraries` 全选上：

```
[*] Include all GStreamer1 core libraries
    *** Libraries ***
-*- GStreamer core library libgstreamer1
-*- GStreamer core library libgst1check
-*- GStreamer core library libgst1controller
-*- GStreamer core library libgst1net
```

gstreamer1-plugins-bad 的 `Select GStreamer bad modules and libraries` 中把常用的音频解码模块选上，其余的按需选择，可参考以下例子：

```
[ ] Include all GStreamer bad plugins (NEW)
    *** Modules ***
<*> GStreamer adpcmdec module
<*> GStreamer adpcmenc module
<*> GStreamer aiff module
<*> GStreamer asfmux module
<*> GStreamer autoconvert module
< > GStreamer bayer module (NEW)
< > GStreamer camerabin module (NEW)
<*> GStreamer debugutilsbad module
< > GStreamer dvdspu module (NEW)
< > GStreamer fbdevsink module (NEW)
< > GStreamer waylandsink module (NEW)
< > GStreamer festival module
<*> GStreamer hls module
< > GStreamer frei0r module (NEW)
<*> GStreamer id3tag module
< > GStreamer jpegformat module (NEW)
<*> GStreamer mpegpsdemux module
<*> GStreamer mpegpsmux module
<*> GStreamer mpegtsdemux module
<*> GStreamer mpegtsmux module
< > GStreamer mxf module (NEW)
<*> GStreamer opusparse module
< > GStreamer pcapparse module (NEW)
< > GStreamer pnm module (NEW)
<*> GStreamer legacyrawparse module
< > GStreamer rfbsrc module (NEW)
< > GStreamer sdpelem module (NEW)
< > GStreamer segmentclip module (NEW)
< > GStreamer shm module (NEW)
< > GStreamer siren module (NEW)
< > GStreamer speed module (NEW)
```

```
< > GStreamer subenc module (NEW)
< > GStreamer videoparsersbad module (NEW)
< > GStreamer smoothstreaming module (NEW)
< > GStreamer openglmixers module (NEW)
    *** Libraries ***
-*- GStreamer adaptivedemux library
< > GStreamer badvideo library (NEW)
< > GStreamer photography library (NEW)
< > GStreamer basecamerabinsrc library (NEW)
-*- GStreamer uridownloader library
-*- GStreamer mpegts library
< > GStreamer wayland library (NEW)
-*- GStreamer codecparsers library
< > GStreamer isoff library (NEW)
< > GStreamer opencv library (NEW)
```

gstreamer1-plugins-base 的 `Select GStreamer base modules and libraries` 中也把常用的音频解码模块选上，其余按需选择，可参考：

```
[ ] Include all GStreamer base plugins (NEW)
    *** Modules ***
-*- GStreamer alsa module
<*> GStreamer app module
<*> GStreamer audioconvert module
<*> GStreamer audiorate module
<*> GStreamer audioresample module
<*> GStreamer audiotestsrc module
<*> GStreamer playback module
< > GStreamer gio module (NEW)
-*- GStreamer ogg module
<*> GStreamer opus module
<*> GStreamer tcp module
< > GStreamer theora module (NEW)
<*> GStreamer typefindfunctions module
< > GStreamer videoconvert module (NEW)
< > GStreamer videorate module (NEW)
< > GStreamer videoscale module (NEW)
< > GStreamer videotestsrc module (NEW)
<*> GStreamer volume module
<*> GStreamer vorbis module
    *** Libraries ***
< > GStreamer allocators library (NEW)
-*- GStreamer app library
-*- GStreamer audio library
<*> GStreamer fft library
-*- GStreamer pbutils library
-*- GStreamer riff library
-*- GStreamer rtp library
< > GStreamer rtsp library (NEW)
< > GStreamer sdp library (NEW)
-*- GStreamer tag library
-*- GStreamer video library
```

同样，gstreamer1-plugins-good 的 `Select GStreamer good modules` 把常用的音频解码模块选上，其余按需选择。

⚠ 警告

如果当前方案平台的 PulseAudio 没有配置好，不能选上 PulseAudio 的模块，不然播放时可能没有声音。因为 GStreamer 播放时，若 PulseAudio 与 ALSA 同时存在，会优先使用 PulseAudio。

模块的选择可参考：

```
[ ] Include all GStreamer good plugins (NEW)
    *** Modules ***
<*> GStreamer alaw module
< > GStreamer alpha module
< > GStreamer alphacolor module (NEW)
<*> GStreamer apetag module
<*> GStreamer audiofx module
<*> GStreamer audioparsers module
<*> GStreamer auparse module
<*> GStreamer autodetect module
< > GStreamer avi module (NEW)
<*> GStreamer cutter module
<*> GStreamer debug module
< > GStreamer deinterlace module (NEW)
<*> GStreamer dtmf module
< > GStreamer effectv module (NEW)
<*> GStreamer equalizer module
<*> GStreamer flac module
< > GStreamer flv module (NEW)
< > GStreamer flxdec module (NEW)
< > GStreamer goom2k1 module (NEW)
< > GStreamer goom module (NEW)
<*> GStreamer icydemux module
<*> GStreamer id3demux module
< > GStreamer imagefreeze module (NEW)
<*> GStreamer interleave module
< > GStreamer isomp4 module (NEW)
< > GStreamer jpeg module (NEW)
<*> GStreamer level module
< > GStreamer matroska module (NEW)
<*> GStreamer mulaw module
-*- GStreamer multifile module
< > GStreamer multipart module (NEW)
<*> GStreamer navigationtest module
< > GStreamer oss4 module
< > GStreamer ossaudio module (NEW)
< > GStreamer png module (NEW)
< > GStreamer pulseaudio module (NEW)
<*> GStreamer replaygain module
<*> GStreamer rtpmanager module
<*> GStreamer rtp module
<*> GStreamer rtsp module
< > GStreamer shapewipe module (NEW)
< > GStreamer smpte module (NEW)
<*> GStreamer soup module
<*> GStreamer spectrum module
<*> GStreamer udp module
< > GStreamer video4linux2 module (NEW)
< > GStreamer videobox module (NEW)
< > GStreamer videocrop module (NEW)
```

```
< > GStreamer videofilter module (NEW)
< > GStreamer videomixer module (NEW)
< > GStreamer vpx module (NEW)
<*> GStreamer wavenc module
<*> GStreamer wavparse module
```

另外，可能需要把裁剪 rootfs 的选项去掉，不然 GStreamer 相关的库可能会被裁剪掉（因为 GStreamer 的库使用 dlopen 的方式打开，库之间的依赖关系可能无法完整检测到）：

```
Target Images --->
    取消选择 [ ] downsize the root filesystem or initramfs
```

4 运行 Alexa Demo

4.1 获取设备 serial number

第一次烧写固件后，需要先获取设备的 serial number 并写入到 AVS SDK 的配置文件中：

```
avs-fetch-device-sn /etc/avs/AlexaClientSDKConfig.json
```

说明

***/etc/avs/AlexaClientSDKConfig.json* 为 *AVS SDK* 的配置文件（需要确保它是可写的）。**

4.2 联网

可以使用任意方式联网，以下使用 wifimanager 为例子：

```
wifi_connect_ap_test <ssid> <password>
```

4.3 运行 SampleApp

```
SampleApp <path_to_AlexaClientSDKConfig.json> [log_level]
```

- `path_to_AlexaClientSDKConfig.json` 为 AVS SDK 的配置文件。
- `log_level` 为调试打印等级，是可选的配置，可设置为 `DEBUG0`～`DEBUG9`。

期间会往 `path_to_AlexaClientSDKConfig.json` 所在目录写入文件，需要确保该目录可写。

例如：

```
SampleApp /etc/avs/AlexaClientSDKConfig.json DEBUG0
```

第一次运行时间会稍微长一些，然后可看到类似以下等待授权的打印信息：

```
#################################
#       NOT YET AUTHORIZED      #
#################################

###############################################################################################
#   To authorize, browse to: 'https://amazon.com/us/code' and enter the code: XXXXXX   #
###############################################################################################

###############################################
#       Checking for authorization (1)...       #
###############################################
```

打开浏览器访问 https://amazon.com/us/code 并登录亚马逊账号，填入打印信息中的 code，即可授权。

授权成功后可看到打印：

```
##########################
#       Authorized!      #
##########################
```

稍候看到以下打印后即可与 Alexa 进行交互：

```
#######################################
#       Alexa is currently idle!      #
#######################################
```

授权成功后下次再运行 SampleApp 就不需要再授权，除非重新烧录固件。

著作权声明

商标声明

免责声明