1 概述

DSP 模块的电源和固件加载，已经在驱动程序里面完成了，有需要可以查看《Rockchip_Developer_Guide_RT-Thread_DSP_CN.pdf》，里面有对驱动使用有较详细的描述。本概述主要是描述如何在 M4 中调用 DSP 的算法以及 DSP 工程代码结构。

2 M4 代码开发

2.1 DSP 调用方法

驱动程序参考<SDK>/bsp/rockchip/common/drivers/drv_dsp.c。

上层应用参考<SDK>/bsp/rockchip/common/tests/dsp_test.c。

调用方法如下：

```
struct rt_device *dsp_dev = rt_device_find("dsp0");
rt_device_open(dsp_dev, RT_DEVICE_OFLAG_RDWR);
rt_device_control(dsp_dev, RKDSP_CTL_QUEUE_WORK, work);
rt_device_control(dsp_dev, RKDSP_CTL_DEQUEUE_WORK, work);
rt_device_close(dsp_dev);
```

“rt_device_open”会调用驱动“rk_dsp_open”函数，主要进行 DSP 的复位、CLK 和 PD 使能、固件加载和启动操作。如果没有使能 RT_USING_DSPFW_LOADER，那么将不会加载固件和启动 dsp，这种情况主要是给 JTAG 下载和调试用的，具体的使用方法参考《DSP-Debug 概述.pdf》。

“rt_device_control”会调用驱动“rk_dsp_control”，参数 RKDSP_CTL_QUEUE_WORK 和 RKDSP_CTL_DEQUEUE_WORK 分别对应 work 的发送和接收。Queue 时发送 work 到 DSP，然后紧跟着 Dequeue，等待 dsp 处理完成接收 work。因为驱动不做 work 的处理，直接将上层输入的 work 指针发送到 dsp 处理，接收也是直接接收 work 指针。所以上层一般只申请一次 work，不需要频繁 create、destory，省去额外的开销。

rt_device_close 会将 CLK 和 PD 关闭。

2.2 Work 介绍

dsp_work 结构体有 8 个成员：

Id：当前的算法 ID，识别单个算法

Type：当前的算法类型，识别需要执行哪种算法类型，比如音频算法配置、音频算法启动。现在识别算法都是使用 type，ID 暂时没用到。

Param：当前算法传送的参数结构指针。地址必须 128Byte 对齐（dsp 的 cache 为 128Byte）。

Param_size：当前算法传送的参数大小，必须 128Byte 对齐。

Result：算法返回结果。结果比较复杂或者长度无法满足，那么也可以放在 Param 结构中，需要自行定义。

WorkType：算法都是使用 RKDSP_ALGO_WORK；RKDSP_CFG_WORK 是给驱动做配置用的，上层不能使用。

Rate：当前 dsp 的频率。

Cycles：算法执行的 cycles 数。

从结构也可以看出，调用的时候需要创建 Work 和 Param 两个结构，并且都需要 128Byte 对齐。创建 Work 使用“rk_dsp_work_create”，释放使用“rk_dsp_work_free”函数。申请 Param 内存使用“rkdsp_malloc”,释放使用“rkdsp_free”函数。Param 申请必须使用“rkdsp_malloc”函数，这个函数做了 size 和地址的对齐。

如果参数结构比较简单，可以不要申请 param，那么可以使用 param 和 result 这两个结构作为输入和输出，而且 param_size 必须为 0。切记 param_size 不能随意使用，因为当 param_size 不为 0 时，M4 和 DSP 两端均会对 param 这个地址进行 cache 操作，容易造成内存错误。

因为 M4 和 DSP 相关传递数据是需要进行 cache 同步操作，work 和 param 这两个结构的 cache 不需要开发人员操作，驱动函数已经做好了。但是需要传送其他的内存数据，就需要通过代码实现 cache 同步操作。比如地址 0x20090000,这段地址的长度是 128Byte。那么在 M4 中需要调用 rt_hw_cpu_dcache_ops(RT_HW_CACHE_FLUSH, 0x20090000,128)，将数据同步到内存中；然后 DSP 在收到相关结构体后，需要调用 `xthal_dcache_region_invalidate`（0x20090000,128）。

3 DSP 工程代码介绍

Dsp 工程代码主要分为：application、driver、include、library、system。开发人员一般需要关心 main.cpp 和 algorithm_handler.cpp，以及自己移植的算法库。

3.1 重要函数介绍

Main 函数主要进行算法初始化以及 Cache、中断、Mailbox 的使能。

System/init/init.cpp 中进行相关的系统级配置，和 M4 的握手以及 Config 就是这完成的。

System/work/work.cpp 中主要进行了 Work 结构的预处理，包括 Cache 同步，以及运算 cycles 结果统计等。

algorithm_handler.cpp 根据不同算法类型进行分发，调用各个算法的执行入口。这是开发人员需要关心的函数。

3.2 串口使用介绍

由于系统自带的 printf 打印函数比较占内存，我们系统中禁止使用这种方式。而是使用轻量化的 LOGD 函数进行打印，包含头文件“trace.h”即可调用。LOGD 会将字符串输出到串口中。

因为 M4 和 DSP 可以用共用串口，所以可以在 DSP 中不需要 UART 的初始化，只要知道 M4 使用 UART 端口即可。修改 UART 端口可以在<DSP>/rkdsp/drivers/uart/uart.cpp 中修改 UART_BASE_ADDR ； 可 以 在 M4 的 menuconfig 中 指 定 （ 参 考《Rockchip_Developer_Guide_RT-Thread_DSP_CN.pdf》）3.2 章。

如果是进行大规模测试的时候，或者担心打印乱码，这时候需要 DSP 使用单独的 UART。需要在 M4 代码的 menuconfig 中打开相应的 UART，这样会在板级进行 IOMUX 的配置。并且在 main 函数入口执行 uart_init(port)。

```
RT-Thread bsp drivers    --->
    RT-Thread rockchip rk2108 drivers    --->
        Enable UART    --->
            [*] Enable UART
            [*]     Enable UART0
            [ ]     Enable UART1
            [*]     Enable UART2
```

uart_init函数会进行串口初始化操作，这里需要注意的是，分频设置需要根据UART基准时钟的不同设置不同的值。26M设置为0x0e,24M设置为0x0d。2108默认是24M分频，所以默认使用0x0d。

```
/* Set baud rate to 115200 */
writel(uart_addr + UART_DLL, 0xE); // 26000000M:0x0e 24000000M:0x0d
writel(uart_addr + UART_DLH, 0x0);
```

4 DSP 算法调用

算法的调用在<DSP>/rkdsp/projects/RK2108/algorithm_handler.cpp 中进行。Algo_handler 函数会根据 work 的 type（在 2.2 中介绍），调用不同的算法函数入口。

4.1 VAD 音频

因为 DSP 支持直接获取 VAD 的数据，这种方式不需要 M4 传输音频数据。可以参考工程原有的 libasr 封装接口的做法，中，进行建立新的“struct wakeup_algo_param algo_param”结构体。结构体成员定义如下：

voice_len：单次声音获取数据量，单位为 Byte。当前获取的声音数据是 16000hz，2 声道，16bit。

advan_len：首次唤醒是，获取音频位置的提前量，默认为 0。

channel：算法需要获取的声道数。因为声卡输出是双声道，这里配置为 1 的话，那么每次送给算法的数据长度是：voice_len。

algo_init：算法初始化。

algo_process：算法执行函数，参数 voice_data 为音频数据地址，voice_len 为数据大小，单位为 Byte。

algo_deinit：算法反初始化。

data：参数数据，可以为空。

M4 调用方法参考<SDK>/bsp/rockchip/common/tests/dsp_test.c 中的 dsp_vad_wakeup。使用 WAKEUP_CONFIG 进行参数配置，然后使用 WAKEUP_START 启动算法。这时候启动 vad（使用 audio_capture），输入语音，检测到关键词后会返回结果，这个算法的返回值为 1 则表示检测到关键词。相关 log 如下：

```
msh />dsp_vad_test
dsp wakeup_test
Hmsh />ifi3: Hifi3 config done
Hifi3: kwsSetConfig ok
Hifi3: init uv_asr ok
ringbuf_addr:0x30260000, period_size:0x00000280
msh />audio_capture
audio_capture
vad buf: 0x30260000, size: 0x20000 bytes
vad periodsize: 0x280 kbytes
msh />Hifi3: xiaodu_wakeup--------xiaoduxiaodu-------
Hifi3: process return value = 1
work result:0x00000001
```

4.2 ASR 算法

ASR 初始化函数为“libasr_init”，算法执行函数为“libasr_process_pre”。Process 时候，传入的 Param 为“struct wake_work_param”，传入了音频 buffer，所以需要执行 xthal_dcache_region_invalidate 进行 cache 同步操作。然后将 buffer 和 size 传给算法，将返回值存入 work_param->result 中。如果有其他的算法调用方法也类似。