1. 背景

之前发布会上，瑞萨提到RA系列芯片设计时就考虑了平台移植的维度，刚好有机会拿到瑞萨的RA2,RA4,RA6三款芯片的板子，便同步做实验，验证该信息的准确性。

2. 硬件描述对比

a. RA2L1

77	62	50	—	—	P501	—	GTIV_B	GTIOC2B_B	—	—	TXD1_C/MOSI1_C/SDA1_C	—	—	AN017	—	—	—	—
78	63	51	—	—	P502	—	GTIW_B	GTIOC3B_C	—	—	RXD1_C/MISO1_C/SCL1_C	—	—	AN018	—	—	—	—
79	64	—	—	—	P503	—	GTETRGA_E	—	—	—	SCK1_C	—	—	AN019	—	—	—	—
80	65	—	—	—	P504	—	GTETRGB_E	—	—	—	CTS1_RTS1_C/SS1_C	—	—	AN020	—	—	—	—

84	66	52	38	—	P015	—	—	—	—	—	—	—	—	AN010	—	—	TS28-CFC	IRQ7_A
85	67	53	39	—	P014	—	—	—	—	—	—	—	—	AN009	DA0	—	—	—
86	68	54	40	—	P013	—	—	—	—	—	—	—	—	AN008	—	—	TS33-CFC	—
87	69	55	41	—	P012	—	—	—	—	—	—	—	—	AN007	—	—	TS32-CFC	—

90	72	58	44	VREFL0	P011	—	—	—	—	—	—	—	—	AN006	—	—	TS31-CFC	—
91	73	59	45	VREFH0	P010	—	—	—	—	—	—	—	—	AN005	—	—	TS30-CFC	—
92	—	—	—	—	P008	—	—	—	—	—	—	—	—	AN014	—	—	—	—
93	—	—	—	—	P007	—	—	—	—	—	—	—	—	AN013	—	—	—	—
94	74	—	—	—	P006	—	—	—	—	—	—	—	—	AN012	—	—	—	—
95	75	—	—	—	P005	—	—	—	—	—	—	—	—	AN011	—	—	—	—
96	76	60	—	—	P004	—	—	—	—	—	—	—	—	AN004	—	—	TS25	IRQ3
97	77	61	—	—	P003	—	—	—	—	—	—	—	—	AN003	—	—	TS24	—
98	78	62	46	—	P002	—	—	—	—	—	—	—	—	AN002	—	—	TS23	IRQ2
99	79	63	47	—	P001	—	—	—	—	—	—	—	—	AN001	—	—	TS22	IRQ7
100	80	64	48	—	P000	—	—	—	—	—	—	—	—	AN000	—	—	TS21	IRQ6

b. RA4M2

P500	—	USB_VBUSEN/QSPCLK	GTIU/AGTOA0	AN016	—

P015	IRQ13	—	—	AN013/DA1	—
P014	—	—	—	AN012/DA0	—
P013	—	—	—	AN011	—

P008	IRQ12-DS	—	—	AN008

P007	—	—	—	AN007
P006	IRQ11-DS	—	—	AN006
P005	IRQ10-DS	—	—	AN005
P004	IRQ9-DS	—	—	AN004
P003	—	—	—	AN003
P002	IRQ8-DS	—	—	AN002
P001	IRQ7-DS	—	—	AN001
P000	IRQ6-DS	—	—	AN000

c. RA6M5

P800	D14	—	CTS0	AGTOA4	AN125
P801	D15	—	CTS8	AGTOB4	AN126
P802	—	IRQ3	—	—	AN127
P803	—	IRQ2	—	—	AN128

P500	—	—	CTS5/USB_VBUSEN/QSPCLK	GTIU/AGTOA0	AN116
P501	—	IRQ11	TXD5/USB_OVRCURA/QSSL	GTIV/AGTOB0	AN117
P502	—	IRQ12	CTS6/RXD5/USB_OVRCURB/QIO0	GTIW/AGTOA2	AN118
P503	—	—	CTS6_RTS6/SCK5/USB_EXICEN/QIO1	GTETRGC/AGTOB2	AN119
P504	ALE	—	SCK6/CTS5_RTS5/USB_ID/QIO2	GTETRGD/AGTOA3	AN120
P505	—	IRQ14	RXD6/QIO3	AGTOB3	AN121
P506	—	IRQ15	TXD6	—	AN122
P507	—	—	SCK6/SCK5	—	AN123
P508	—	—	CTS_RTS5_B	—	AN124

P015	—	IRQ13	—	—	AN013/DA1
P014	—	—	—	—	AN012/DA0

P010	—	IRQ14	—	—	AN010
P009	—	IRQ13-DS	—	—	AN009
P008	—	IRQ12-DS	—	—	AN008
P007	—	—	—	—	AN007
P006	—	IRQ11-DS	—	—	AN006
P005	—	IRQ10-DS	—	—	AN005
P004	—	IRQ9-DS	—	—	AN004
P003	—	—	—	—	AN003
P002	—	IRQ8-DS	—	—	AN002/AN102
P001	—	IRQ7-DS	—	—	AN001/AN101
P000	—	IRQ6-DS	—	—	AN000/AN100

d. 硬件对比结论

在不考虑芯片封装的情况下，RA2L1和RA4M2之间，在硬件上adc的功能定义并未做到完全的pin2pin，但这不算什么大问题，如果能在软件上实现底层驱动差异化，HAL层向上兼容，也可以实现减少不同方案移植的工作量要求。RA4M2和RA6M5之间在接口功能上实现了有功能口的pin2pin兼容，理论上工程可以顺切。

3. 代码层面分析

a. 对比计划

由于最近在用rtthread，因此便基于rtthread上rensa代码进行分析。

b. 代码分析

i. rtt代码层

见测试代码，应用层和驱动层通过device框架实现通信，该通信机制可以去RTT官网学习，具体实现方式为打开一个注册的adc设备，然后对应的ops即为顶层所调用的接口。

ii. HAL（driver）层代码

```
//具体HAL层代码见 \bsp\renesas\libraries\HAL_Drivers\drv_adc.c，此处仅提供核心代码
//其中 _ra_adc0_device 和 _ra_adc1_device为对应fsp部分代码

static rt_err_t ra_adc_enabled(struct rt_adc_device *device, rt_uint32_t channel, rt_bool_t enabled)
{
    RT_ASSERT(device != RT_NULL);
    struct ra_adc_map *adc = (struct ra_adc_map *)device->parent.user_data;
    /**< start adc*/
    if (enabled)
    {
        if (FSP_SUCCESS != R_ADC_ScanStart((adc_ctrl_t *)adc->g_ctrl))
        {
            LOG_E("start adc%c failed.", adc->name);
            return -RT_ERROR;
        }
    }
    else
    {
        /**< stop adc*/
        if (FSP_SUCCESS != R_ADC_ScanStop((adc_ctrl_t *)adc->g_ctrl))
        {
            LOG_E("stop adc%c failed.", adc->name);
            return -RT_ERROR;
        }
    }
    return RT_EOK;
}

static rt_err_t ra_get_adc_value(struct rt_adc_device *device, rt_uint32_t channel, rt_uint32_t *value)
{
    RT_ASSERT(device != RT_NULL);
    struct ra_adc_map *adc = (struct ra_adc_map *)device->parent.user_data;
    if (RT_EOK != R_ADC_Read32((adc_ctrl_t *)adc->g_ctrl, channel, value))
    {
        LOG_E("get adc value failed.\n");
        return -RT_ERROR;
```

```
    }
    return RT_EOK;
}

static const struct rt_adc_ops ra_adc_ops =
{
    .enabled = ra_adc_enabled,   // 对应应用层代码 rt_adc_enable 和 rt_adc_disable
    .convert = ra_get_adc_value, // 对应应用层代码 rt_adc_read
};


static int ra_adc_init(void)
{
#if defined(BSP_USING_ADC0)
    R_ADC_Open((adc_ctrl_t *)_ra_adc0_device.ra_adc_dev->g_ctrl,
               (adc_cfg_t const * const)_ra_adc0_device.ra_adc_dev->g_cfg);

    R_ADC_ScanCfg((adc_ctrl_t *)_ra_adc0_device.ra_adc_dev->g_ctrl,
                  (adc_cfg_t const * const)_ra_adc0_device.ra_adc_dev->g_channel_cfg);

    if (RT_EOK != rt_hw_adc_register(_ra_adc0_device.ra_adc_device_t, "adc0",
&ra_adc_ops, (void *)_ra_adc0_device.ra_adc_dev))
    {
        LOG_E("adc0 register failed");
        return -RT_ERROR;
    }
#endif

#if defined(BSP_USING_ADC1)
    R_ADC_Open((adc_ctrl_t *)_ra_adc1_device.ra_adc_dev->g_ctrl,
               (adc_cfg_t const * const)_ra_adc1_device.ra_adc_dev->g_cfg);

    R_ADC_ScanCfg((adc_ctrl_t *)_ra_adc1_device.ra_adc_dev->g_ctrl,
                  (adc_cfg_t const * const)_ra_adc1_device.ra_adc_dev->g_channel_cfg);

    if (RT_EOK != rt_hw_adc_register(_ra_adc1_device.ra_adc_device_t, "adc1",
&ra_adc_ops, (void *)_ra_adc1_device.ra_adc_dev))
    {
        LOG_E("adc1 register failed");
        return -RT_ERROR;
```

```
    }
#endif

    return RT_EOK;
}
INIT_BOARD_EXPORT(ra_adc_init);
```

iii. BSP层代码

```
//此部分代码须在keil工程中查看
// 所在文件为r_adc.c
// 从HAL层代码看，此部分代码为具体寄存器操作，所操作的参数为HAL层传递下来的参数

/*******************************************************************************************************************//**
 * Sets the operational mode, trigger sources, interrupt  priority, and configurations for the peripheral as a whole.
 * If interrupt is enabled, the function registers a callback function pointer for notifying the user whenever a scan
 * has completed.
 *
 * @retval FSP_SUCCESS                  Module is ready for use.
 * @retval FSP_ERR_ASSERTION            An input argument is invalid.
 * @retval FSP_ERR_ALREADY_OPEN         The instance control structure has already been opened.
 * @retval FSP_ERR_IRQ_BSP_DISABLED     A callback is provided, but the interrupt is not enabled.
 * @retval FSP_ERR_IP_CHANNEL_NOT_PRESENT  The requested unit does not exist on this MCU.
 * @retval FSP_ERR_INVALID_HW_CONDITION    The ADC clock must be at least 1 MHz

*******************************************************************************************************************/
fsp_err_t R_ADC_Open (adc_ctrl_t * p_ctrl, adc_cfg_t const * const p_cfg)
{
    adc_instance_ctrl_t * p_instance_ctrl = (adc_instance_ctrl_t *) p_ctrl;

    /*  Perform parameter checking */
#if ADC_CFG_PARAM_CHECKING_ENABLE

    /* Verify the pointers are valid */
    FSP_ASSERT(NULL != p_instance_ctrl);
```

```
26
27      /* Verify the configuration parameters are valid   */
28      fsp_err_t err = r_adc_open_cfg_check(p_cfg);
29      FSP_ERROR_RETURN(FSP_SUCCESS == err, err);
30
31      /* Check for valid argument values for options that are unique to the IP */
32      err = r_adc_open_cfg_resolution_check(p_cfg);
33      FSP_ERROR_RETURN(FSP_SUCCESS == err, err);
34
35      /* Verify this unit has not already been initialized   */
36      FSP_ERROR_RETURN(ADC_OPEN != p_instance_ctrl->opened, FSP_ERR_ALREADY_OPEN);
37
38      /* If a callback is used, then make sure an interrupt is enabled */
39      adc_extended_cfg_t const * p_extend = (adc_extended_cfg_t const *) p_cfg->p_extend;
40      if (NULL != p_cfg->p_callback)
41      {
42          FSP_ERROR_RETURN((p_cfg->scan_end_irq >= 0) || (p_extend->window_a_irq >= 0) ||
    (p_extend->window_b_irq >= 0),
43                           FSP_ERR_IRQ_BSP_DISABLED);
44
45          /* Group B interrupts are never required since group B can be configured in
    continuous scan mode when group A
46           * has priority over group B. */
47      }
48
49  #else
50      adc_extended_cfg_t const * p_extend = (adc_extended_cfg_t const *) p_cfg->p_extend;
51  #endif
52
53      /* Save configurations. */
54      p_instance_ctrl->p_cfg             = p_cfg;
55      p_instance_ctrl->p_callback        = p_cfg->p_callback;
56      p_instance_ctrl->p_context         = p_cfg->p_context;
57      p_instance_ctrl->p_callback_memory = NULL;
58
59      /* Calculate the register base address. */
60      uint32_t address_gap = (uint32_t) R_ADC1 - (uint32_t) R_ADC0;
61      p_instance_ctrl->p_reg = (R_ADC0_Type *) ((uint32_t) R_ADC0 + (address_gap * p_cfg-
    >unit));
62
```

```
63      /* Initialize the hardware based on the configuration. */
64      r_adc_open_sub(p_instance_ctrl, p_cfg);
65
66      /* Enable interrupts */
67      r_adc_irq_enable(p_cfg->scan_end_irq, p_cfg->scan_end_ipl, p_instance_ctrl);
68      r_adc_irq_enable(p_cfg->scan_end_b_irq, p_cfg->scan_end_b_ipl, p_instance_ctrl);
69      r_adc_irq_enable(p_extend->window_a_irq, p_extend->window_a_ipl, p_instance_ctrl);
70      r_adc_irq_enable(p_extend->window_b_irq, p_extend->window_b_ipl, p_instance_ctrl);
71
72      /* Invalid scan mask (initialized for later). */
73      p_instance_ctrl->scan_mask = 0U;
74
75      /* Mark driver as opened by initializing it to "RADC" in its ASCII equivalent for
    this unit. */
76      p_instance_ctrl->opened = ADC_OPEN;
77
78      /* Return the error code */
79      return FSP_SUCCESS;
80  }
81
82  /*******************************************************************************************
    ************************//**
83   * Configures the ADC scan parameters. Channel specific settings are set in this
    function. Pass a pointer to
84   * @ref adc_channel_cfg_t to p_channel_cfg.
85   *
86   * @note This starts group B scans if adc_channel_cfg_t::priority_group_a is set to
    ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN.
87   *
88   * @retval FSP_SUCCESS                  Channel specific settings applied.
89   * @retval FSP_ERR_ASSERTION            An input argument is invalid.
90   * @retval FSP_ERR_NOT_OPEN             Unit is not open.
91
    *******************************************************************************************
    ************************/
92  fsp_err_t R_ADC_ScanCfg (adc_ctrl_t * p_ctrl, void const * const p_channel_cfg)
93  {
94      adc_channel_cfg_t const * p_adc_channel_cfg = (adc_channel_cfg_t const *)
    p_channel_cfg;
95      adc_instance_ctrl_t     * p_instance_ctrl   = (adc_instance_ctrl_t *) p_ctrl;
96      fsp_err_t                 err               = FSP_SUCCESS;
```

```
97
98  #if ADC_CFG_PARAM_CHECKING_ENABLE
99      FSP_ASSERT(NULL != p_instance_ctrl);
100     FSP_ASSERT(NULL != p_adc_channel_cfg);
101     FSP_ERROR_RETURN(ADC_OPEN == p_instance_ctrl->opened, FSP_ERR_NOT_OPEN);
102
103     err = r_adc_scan_cfg_check(p_instance_ctrl, p_adc_channel_cfg);
104     FSP_ERROR_RETURN(FSP_SUCCESS == err, err);
105 #endif
106
107     /* Configure the hardware based on the configuration */
108     r_adc_scan_cfg(p_instance_ctrl, p_adc_channel_cfg);
109
110     /* Save the scan mask locally; this is required for the infoGet function. */
111     p_instance_ctrl->scan_mask = p_adc_channel_cfg->scan_mask;
112
113     /* Return the error code */
114     return err;
115 }
116
117 /*******************************************************************************************************************//**
118  * Starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in
119  * the R_ADC_Open call. If the unit was configured for ELC or external hardware triggering, then this function allows
120  * the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself.
121  * If the unit was configured for software triggering, then this function starts the software triggered scan.
122  *
123  * @pre Call R_ADC_ScanCfg after R_ADC_Open before starting a scan.
124  *
125  * @pre On MCUs that support calibration, call R_ADC_Calibrate and wait for calibration to complete before starting
126  * a scan.
127  *
128  * @retval FSP_SUCCESS                  Scan started (software trigger) or hardware triggers enabled.
129  * @retval FSP_ERR_ASSERTION            An input argument is invalid.
130  * @retval FSP_ERR_NOT_OPEN             Unit is not open.
```

```
 * @retval FSP_ERR_NOT_INITIALIZED     Unit is not initialized.
 * @retval FSP_ERR_IN_USE              Another scan is still in progress (software trigger).
 **********************************************************************************************************************/
fsp_err_t R_ADC_ScanStart (adc_ctrl_t * p_ctrl)
{
    adc_instance_ctrl_t * p_instance_ctrl = (adc_instance_ctrl_t *) p_ctrl;

    /* Perform parameter checking  */
#if ADC_CFG_PARAM_CHECKING_ENABLE

    /* Verify the pointers are valid */
    FSP_ASSERT(NULL != p_instance_ctrl);
    FSP_ERROR_RETURN(ADC_OPEN == p_instance_ctrl->opened, FSP_ERR_NOT_OPEN);
    FSP_ERROR_RETURN(ADC_OPEN == p_instance_ctrl->initialized, FSP_ERR_NOT_INITIALIZED);
    if (ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN != p_instance_ctrl->p_reg->ADGSPCR)
    {
        FSP_ERROR_RETURN(0U == p_instance_ctrl->p_reg->ADCSR_b.ADST, FSP_ERR_IN_USE);
    }
#endif

    /* Enable hardware trigger or start software scan depending on mode. */
    p_instance_ctrl->p_reg->ADCSR = p_instance_ctrl->scan_start_adcsr;

    return FSP_SUCCESS;
}

/***********************************************************************************************************************//**
 * Reads conversion results from a single channel or sensor register into a 32-bit result.
 *
 * @retval FSP_SUCCESS                 Data read into provided p_data.
 * @retval FSP_ERR_ASSERTION           An input argument is invalid.
 * @retval FSP_ERR_NOT_OPEN            Unit is not open.
 * @retval FSP_ERR_NOT_INITIALIZED     Unit is not initialized.
 **********************************************************************************************************************/
```

```
fsp_err_t R_ADC_Read32 (adc_ctrl_t * p_ctrl, adc_channel_t const reg_id, uint32_t * const p_data)
{
    uint16_t result    = 0U;
    uint32_t result_32 = 0U;

#if ADC_CFG_PARAM_CHECKING_ENABLE
    FSP_ASSERT(NULL != p_data);
#endif

    fsp_err_t err = R_ADC_Read(p_ctrl, reg_id, &result);
    FSP_ERROR_RETURN(FSP_SUCCESS == err, err);

    result_32 = result;

    /* Left shift the result into the upper 16 bits if the unit is configured for left alignment. */
    adc_instance_ctrl_t * p_instance_ctrl = (adc_instance_ctrl_t *) p_ctrl;
    if (ADC_ALIGNMENT_LEFT == p_instance_ctrl->p_cfg->alignment)
    {
        result_32 <<= ADC_SHIFT_LEFT_ALIGNED_32_BIT;
    }

    *p_data = result_32;

    return FSP_SUCCESS;
}

/*****************************************************************************************************************//**
 * Stops the software scan or disables the unit from being triggered by the hardware trigger (ELC or external) based on
 * what type of trigger the unit was configured for in the R_ADC_Open function. Stopping a hardware triggered scan via
 * this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software
 * triggered scan aborts an ongoing scan.
 *
 * @retval FSP_SUCCESS                  Scan stopped (software trigger) or hardware triggers disabled.
 * @retval FSP_ERR_ASSERTION            An input argument is invalid.
```

```
 * @retval FSP_ERR_NOT_OPEN            Unit is not open.
 * @retval FSP_ERR_NOT_INITIALIZED     Unit is not initialized.

**********************************************************************************************************************/
fsp_err_t R_ADC_ScanStop (adc_ctrl_t * p_ctrl)
{
    adc_instance_ctrl_t * p_instance_ctrl = (adc_instance_ctrl_t *) p_ctrl;

    /*  Perform parameter checking */
#if ADC_CFG_PARAM_CHECKING_ENABLE

    /* Verify the pointers are valid */
    FSP_ASSERT(NULL != p_instance_ctrl);
    FSP_ERROR_RETURN(ADC_OPEN == p_instance_ctrl->opened, FSP_ERR_NOT_OPEN);
    FSP_ERROR_RETURN(ADC_OPEN == p_instance_ctrl->initialized, FSP_ERR_NOT_INITIALIZED);
#endif

    /* Disable hardware trigger or stop software scan depending on mode. */
    p_instance_ctrl->p_reg->ADCSR = 0U;

    return FSP_SUCCESS;
}
```

c. FSP层

由于此层为FSP工具生成代码，我们仅需知道他和RTT对应关系为drv_adc.c中特定定义即可，且在瑞萨RA系列代码中，drv_adc.c为不会更改的部分，因此FSP层也不会更改。

4. 编码验证

a. fsp和KConfig配置

此处省略

b. 测试代码

```
// 此代码贴至hal_entry.c中
// P002为测试ADC口

#define DEV_ADC                "adc0"
#define DEV_ADC_CHANNEL        0

```

```c
#define REFER_VOLTAGE       330
#define CONVERT_BITS        (1 << 12)

void ad_test(void)
{
    rt_adc_device_t dev_adc = (rt_adc_device_t)rt_device_find(DEV_ADC);
    rt_uint32_t vol, value = 1000;

    if(dev_adc == RT_NULL)
    {
        rt_kprintf("no adc device named %s\n", DEV_ADC);
    }
    rt_adc_enable(dev_adc, DEV_ADC_CHANNEL);

    value = rt_adc_read(dev_adc, DEV_ADC_CHANNEL);

    vol = value * REFER_VOLTAGE / CONVERT_BITS;
    rt_kprintf("the adc voltage is :%d.%02d \n", vol / 100, vol % 100);

    rt_adc_disable(dev_adc, DEV_ADC_CHANNEL);
}
MSH_CMD_EXPORT(ad_test, ad_test);
```

c. 验证结果

外接DAC的情况下，DAC电压调节可以正确读到电压变化。

RA6M5、RA4M2和RA2L1在不改动应用层软件的情况下，可以实现同样的功能，仅需要在FSP层面做IO口功能配置即可。

5. 结论

瑞萨在硬件设计时，有在一定程度上实现不同芯片之间的兼容。在软件层面上，除了在HAL层做不同芯片之间的区分外，还在HAL层的下一层做了标准化，用户仅需要使用FSP配置对应功能，部分芯片需要更换一下功能口编号即可快速实现工程迁移。

理论上，在不跑RTT的情况下，用户也可以使用类似于RTT的方式，在HAL层参数上做标准化，实现工程在RA系列上的快速迁移。