SPC1169 ADC 使用指南

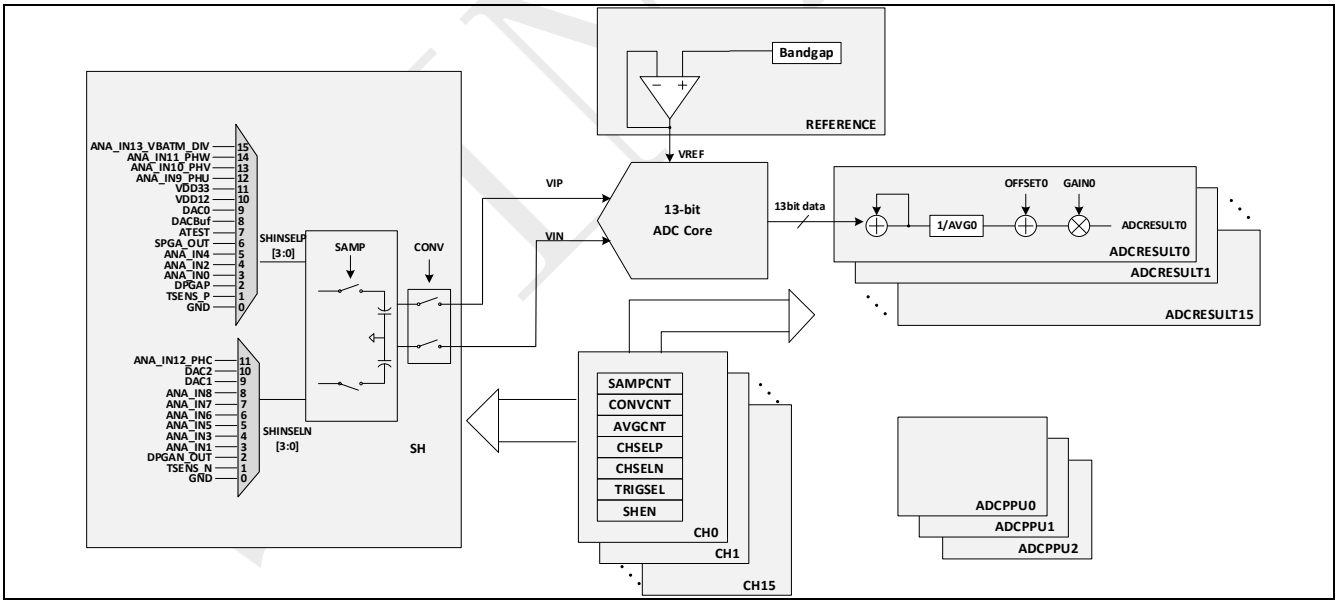
版本 A/0 – 2023 年 4 月

概述

在电力电子系统中，通常使用 ADC 进行电压采样，将模拟电压转换为数字电压。SPC1169/SPD1179/SPD1176 中有一个 13bit SAR ADC，其结构框图如图 1-1：ADC 结构框图所示，其特性描述如下：

- 13bit 精度的采样保持器，其采样速率高达 2.5 M samples/s
- 满量程输入为 3.657V
- 16 路独立通道
- 多种采样触发源
- 3 路后处理单元

图 1-1：ADC 结构框图



目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 5 月 3 日	CanChai	Released	首次发布。

术语或缩写

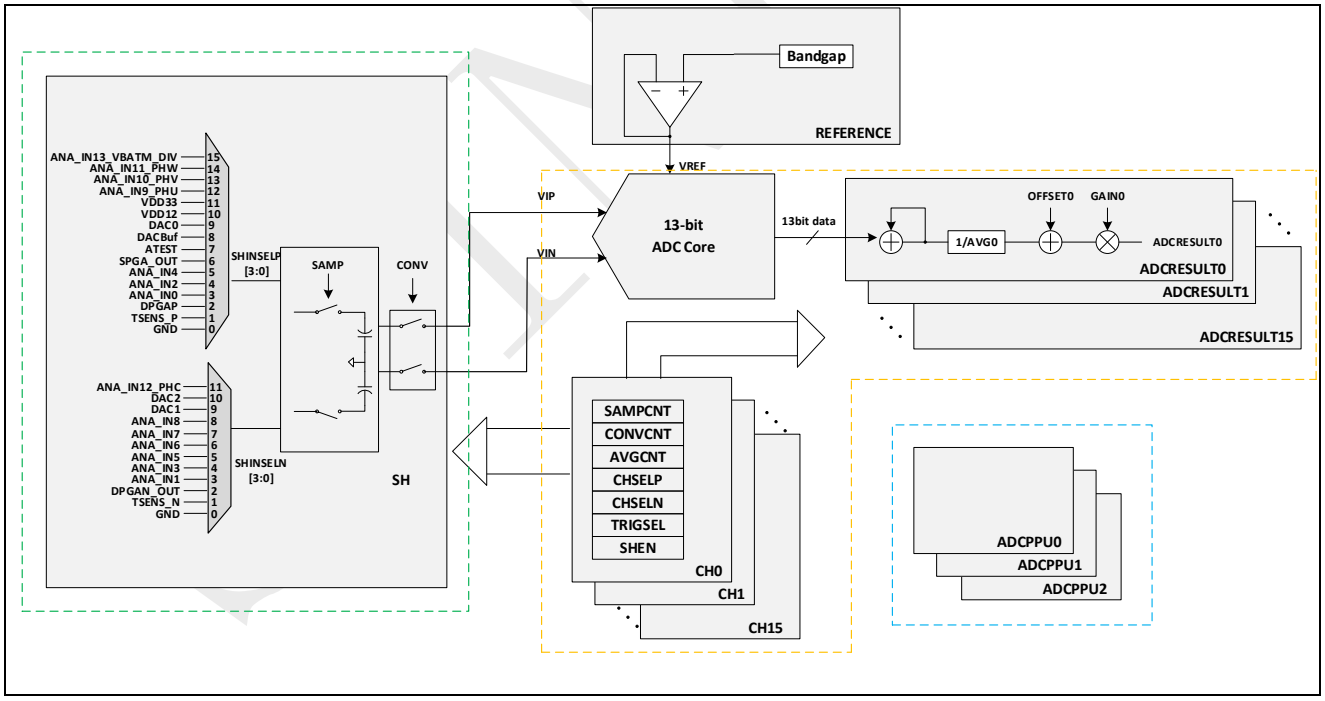
术语或缩写	描述
共模电压	差分电路两个输入端电压的平均值，也称共模信号。
SAR ADC	逐次逼近型 ADC

1 整体功能解释

在实际使用 ADC 过程中，可以按照功能区块来帮助理解怎样使用 ADC，ADC 大体上可以按照图 1-1：ADC 功能分区框图中的不同颜色虚线分成 3 个功能区块，在使用过程中按照如下步骤进行配置：

- 首先绿色虚线框内只需要使能采样及转换开关，这个步骤可以等其它配置都设置完成之后再进行；
- 然后设置黄色虚线框部分，在这部分中，有 16 路独立的采样转换配置（SOC，Set-of-Convert）可以预先设置，每个 SOC 均可根据不同的目的而配置不同的参数，例如：采样时间（具体需要设置多长的采样时间，请参考《ADC 建立时间计算方法使用指南》），ADC core 转换时间，是否需要多次采样且取平均值，ADC 输入的正负端，触发源选择等；在设置完这些信息之后，对应 SOC 的设置产生的 ADC 转换结果将存储在对应的 ADCRESULTSx（x=0,1，…，15）中。
- 最后，如果有需求，可以继续设置蓝色虚线框部分，此部分即为 ADC PPU 单元。通常工程使用中，需要知道 ADC 转换的结果相对某个参考值而言是过高还是过低，这部分的工作就可以交给 PPU 单元处理，以减少 CPU 的计算压力。除此之外，PPU 单元还可以检测出从发起 ADC 转换请求至开始 ADC 转换时的时间计数。

图 1-1：ADC 功能分区框图



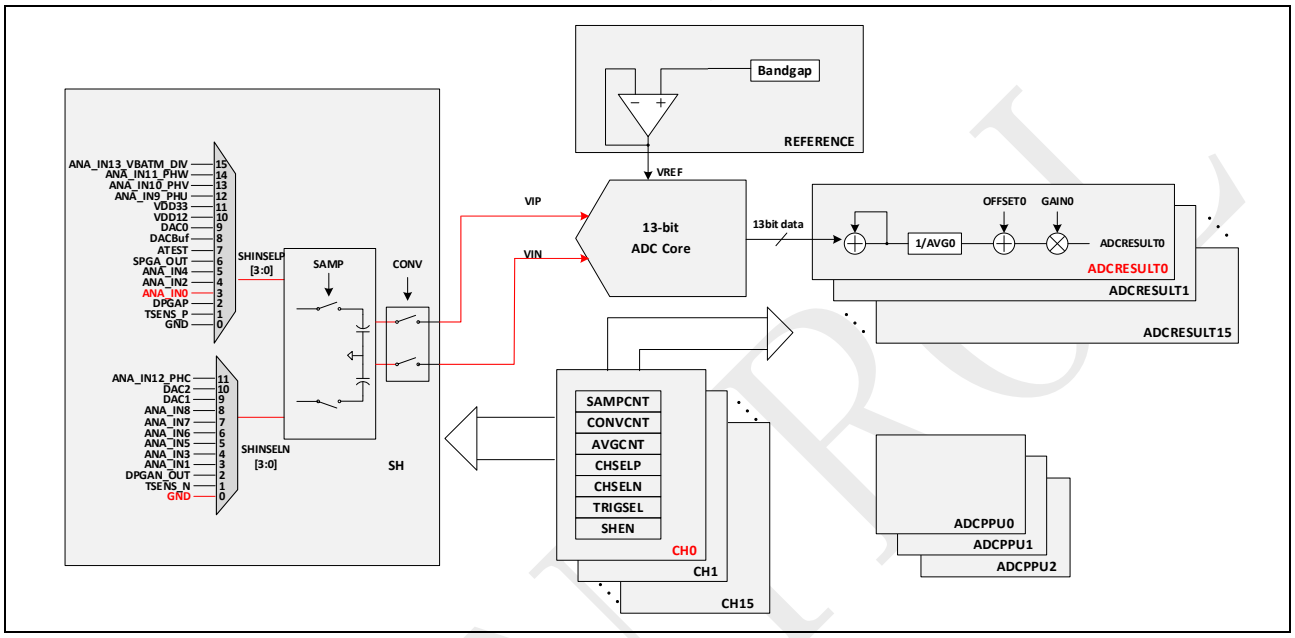
注意：需要注意的一点是，SPC1169/SPD1179/SPD1176 的 ADC 正端或者负端输入的绝对电压必须保持大于等于零，否则 ADC 工作不正常。

2 ADC 示例

2.1 ADC 单端采样

利用 ADC 进行单端（也即有一端接 GND）采样，如图 2-1：ADC 单端采样信号流所示。

图 2-1：ADC 单端采样信号流



实现图示中描述的功能的代码步骤如下：

在 ADC_EasyInit1 函数接口中填入相关参数，即可完成 ADC 时钟等的初始化，ADC 通道选择，输入选择（在如下代码中，选择输入为 PIN_GPIO0，此时函数会自动判断出 PIN_GPIO0 为正端，并将负端接 GND），触发源，采样时间，转换时间的相关设置。需要注意的是，函数接口内会将对应的 GPIO 设置成为模拟 IO。

若想单独设置其它参数，可调用其它接口进行设置，例如，在以下代码中，调用 ADC_SetChannelResultAverageCount 进行对目标电压采样 32 次，且结果取均值的设置。

Example Code

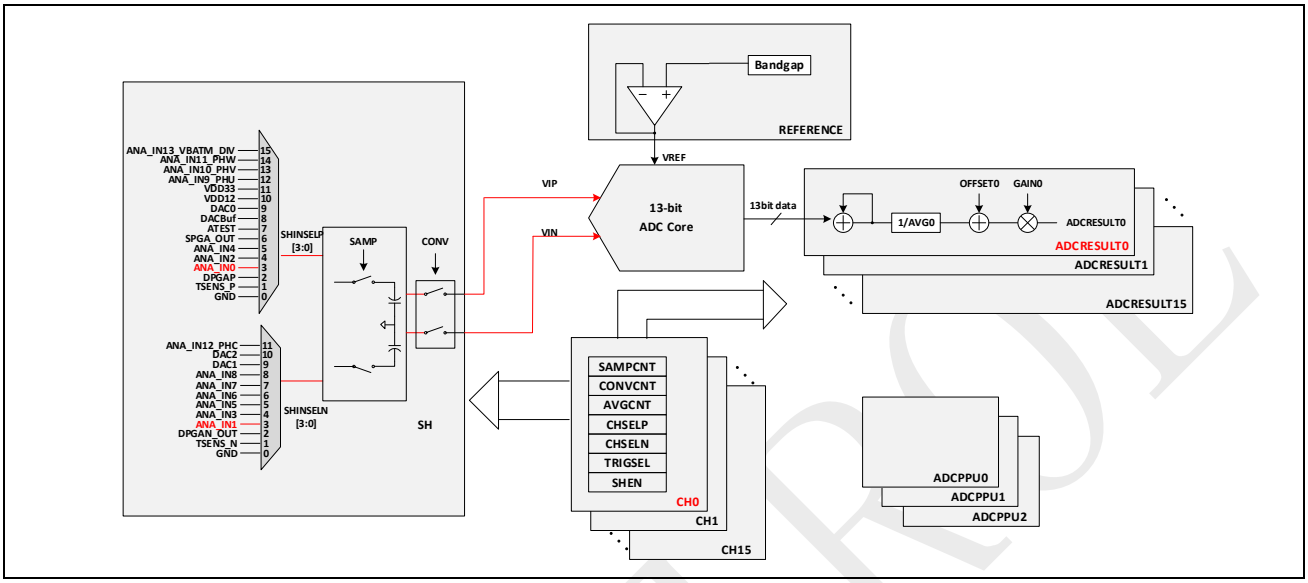
```
#include <stdio.h>
#include "spd1179.h"

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);
```

```
    printf("Enter the test\n");

    /*ADC Init*/
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Set Average Times */
    ADC_SetChannelResultAverageCount(ADC, ADC_CH0, ADC_AVERAGE_COUNT_32);

    while (1)
    {
        /* Use software to trigger ADC SOC0 start to work */
        ADC_ForceChannelSOC(ADC, ADC_CH0);

        /* Wait until ADC conversion finished (Interrupt flag was set) */
        while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0);


        /* Get result */
        i32VSP = ADC_GetChannelResult(ADC, ADC_CH0);

        i32VSP = ABS(i32VSP);

        /* Clear ADC SOC0 INT flag */
        ADC_ClearChannelInt(ADC, ADC_CH0);

        printf("ADC Single End Result = %d\n", i32VSP);
        printf("ADC Single End Voltage %dmv\n", ValueToVoltage(i32VSP));

        Delay_Ms(500);
    }
}
```

2.2 ADC 双端采样

利用 ADC 进行双端采样，如图 2-2：ADC 双端采样信号流所示。

图 2-2：ADC 双端采样信号流



实现图示中描述的功能的代码步骤如下：

在 ADC_EasyInit2 函数接口中填入相关参数，即可完成 ADC 时钟等的初始化，ADC 通道选择，输入选择（在如下代码中，选择输入为 PIN_GPIO0 及 PIN_GPIO1，此时函数会自动判断出 PIN_GPIO0 为正端，而 PIN_GPIO1 位负端），触发源，采样时间，转换时间的相关设置。需要注意的是，函数接口内会将对应的 GPIO 设置成为模拟 IO。

若想单独设置其它参数，可调用其它接口进行设置，例如，在以下代码中，调用 ADC_SetChannelResultAverageCount 进行对目标电压采样 32 次，且结果取均值的设置。

Example Code

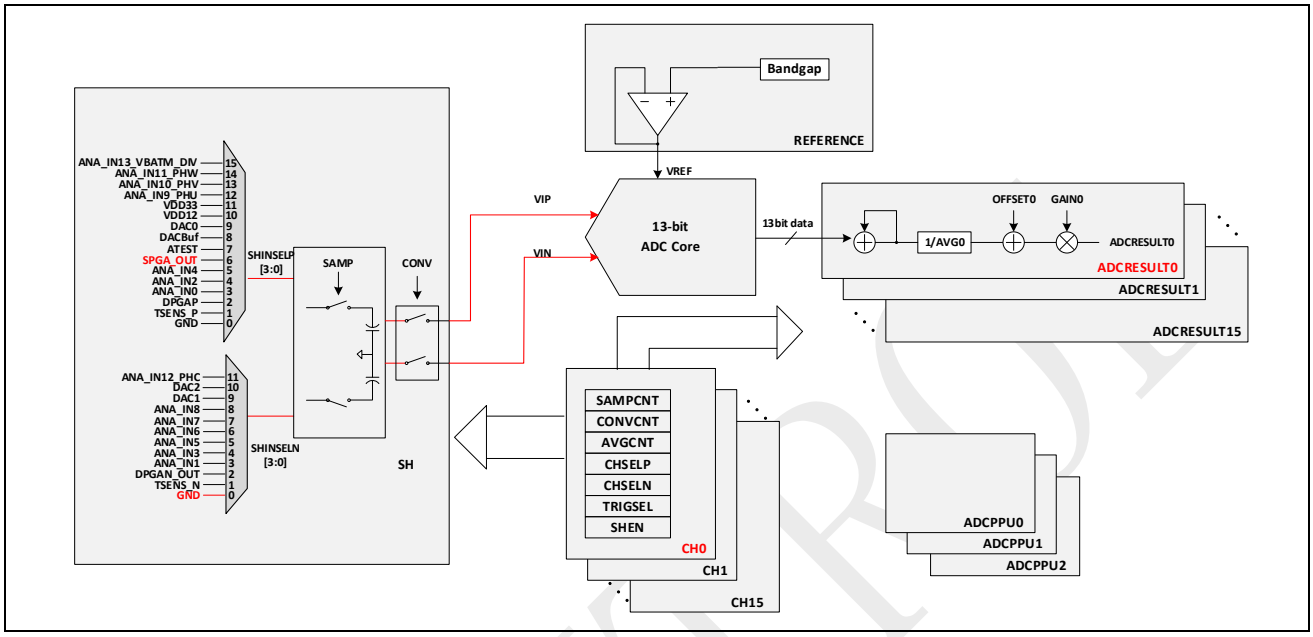
```
#include <stdio.h>
#include "spd1179.h"

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /*ADC Init*/
    ADC_EasyInit2(ADC, ADC_CH0, PIN_GPIO0, PIN_GPIO1,
ADC_SOC_TRIGGER_FROM_SOFTWARE);
```

```
    /* Set Average Times */
    ADC_SetChannelResultAverageCount(ADC, ADC_CH0, ADC_AVERAGE_COUNT_32);

    while (1)
    {
        /* Use software to trigger ADC SOC0 start to work */
        ADC_ForceChannelSOC(ADC, ADC_CH0);

        /* Wait until ADC conversion finished (Interrupt flag was set) */
        while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0);

        /* Get result */
        i32VSP = ADC_GetChannelResult(ADC, ADC_CH0);

        i32VSP = ABS(i32VSP);

        /* Clear ADC SOC0 INT flag */
        ADC_ClearChannelInt(ADC, ADC_CH0);

        printf("ADC Differential Result = %d\n", i32VSP);
        printf("ADC Differential Voltage %dmv\n", ValueToVoltage(i32VSP));

        Delay_Ms(500);
    }
}
```

2.3 ADC SPGA 采样

利用 ADC 对 SPGA 的输出进行采样，如图 2-3：ADC 对 SPGA 采样信号流所示。

图 2-3：ADC 对 SPGA 采样信号流



实现图示中描述的功能的代码步骤如下：

设置 PIN_GPIO0 为模拟 IO，并调用 PGA_InitSPGA 接口，将 SPGA 的输入设置为模拟 IO0（也即 PIN_GPIO0），且将放大倍数设置为 1 倍。

在 ADC_EasyInit1 函数接口中填入相关参数，即可完成 ADC 时钟等的初始化，ADC 通道选择，输入选择（在如下代码中，选择输入为 ADC_IN_SPGA_OUT（SPGA 的输出），此时函数会自动判断出 ADC_IN_SPGA_OUT 为正端，并将负端接 GND），触发源，采样时间，转换时间的相关设置。需要注意的是，函数接口内会将对应的 GPIO 设置成为模拟 IO。

若想单独设置其它参数，可调用单独的设置接口，例如，在以下代码中，调用 ADC_SetChannelResultAverageCount 进行对目标电压采样 16 次，且结果取均值的设置。

Example Code

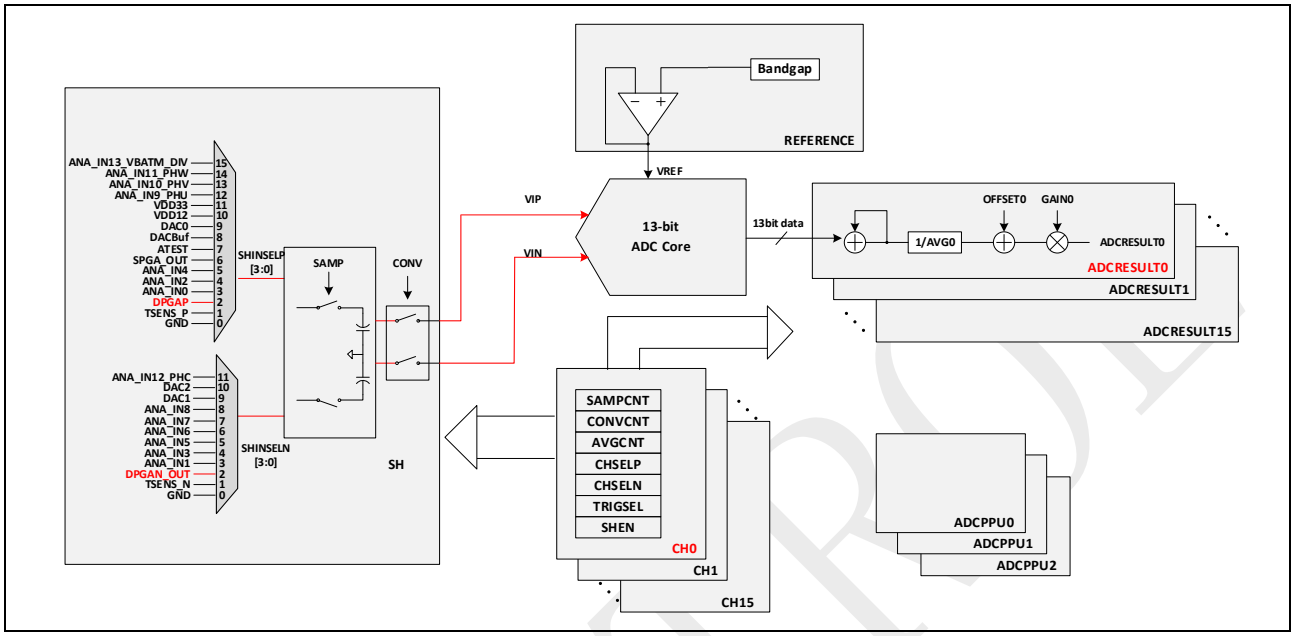
```
#include <stdio.h>
#include "spd1179.h"

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);
```

```
    /*
    * set the GPIO as ADC.
    */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_ANA_IN0);

    /*
    * Set PGA in  single-ended mode.
    */
    PGA_InitSPGA(SPGA_FROM_ANA_IN0, SPGA_GAIN_1X);

    /*ADC Init*/
    ADC_EasyInit1(ADC, ADC_CH0, ADC_IN_SPGA_OUT,
ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Set Average Times */
    ADC_SetChannelResultAverageCount(ADC, ADC_CH0, ADC_AVERAGE_COUNT_16);

    while (1)
    {
        /* Use software to trigger ADC SOC0 start to work */
        ADC_ForceChannelSOC(ADC, ADC_CH0);

        /* Wait until ADC conversion finished (Interrupt flag was set) */
        while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0);

        /* Get result */
        i32VSP = ADC_GetChannelResult(ADC, ADC_CH0);

        i32VSP = ABS(i32VSP);

        /* Clear ADC SOC0 INT flag */
        ADC_ClearChannelInt(ADC, ADC_CH0);

        printf("ADC Single End Result = %d\n", i32VSP);
        printf("ADC Single End Voltage %dmv\n", ValueToVoltage(i32VSP));

        Delay_Ms(500);
    }
}
```

2.4 ADC DPGA 采样

利用 ADC 对 DPGA 的输出进行采样，如图 2-4：ADC 对 DPGA 采样信号流所示。

图 2-4：ADC 对 DPGA 采样信号流



实现图示中描述的功能的代码步骤如下：

调用 PGA_InitDPGA 接口完成对 DPGA 的初始化，且将放大倍数设置为 2 倍。

在 ADC_EasyInit1 函数接口中填入相关参数，即可完成 ADC 时钟等的初始化，ADC 通道选择，输入选择（在如下代码中，选择输入为 ADC_IN_SPGA_OUT（SPGA 的输出），此时函数会自动判断出 ADC_IN_SPGA_OUT 为正端，并将负端接 GND），触发源，采样时间，转换时间的相关设置。需要注意的是，函数接口内会将对应的 GPIO 设置成为模拟 IO。

若想单独设置其它参数，可调用单独的设置接口，例如，在以下代码中，调用 ADC_SetChannelResultAverageCount 进行对目标电压采样 16 次，且结果取均值的设置。

Example Code

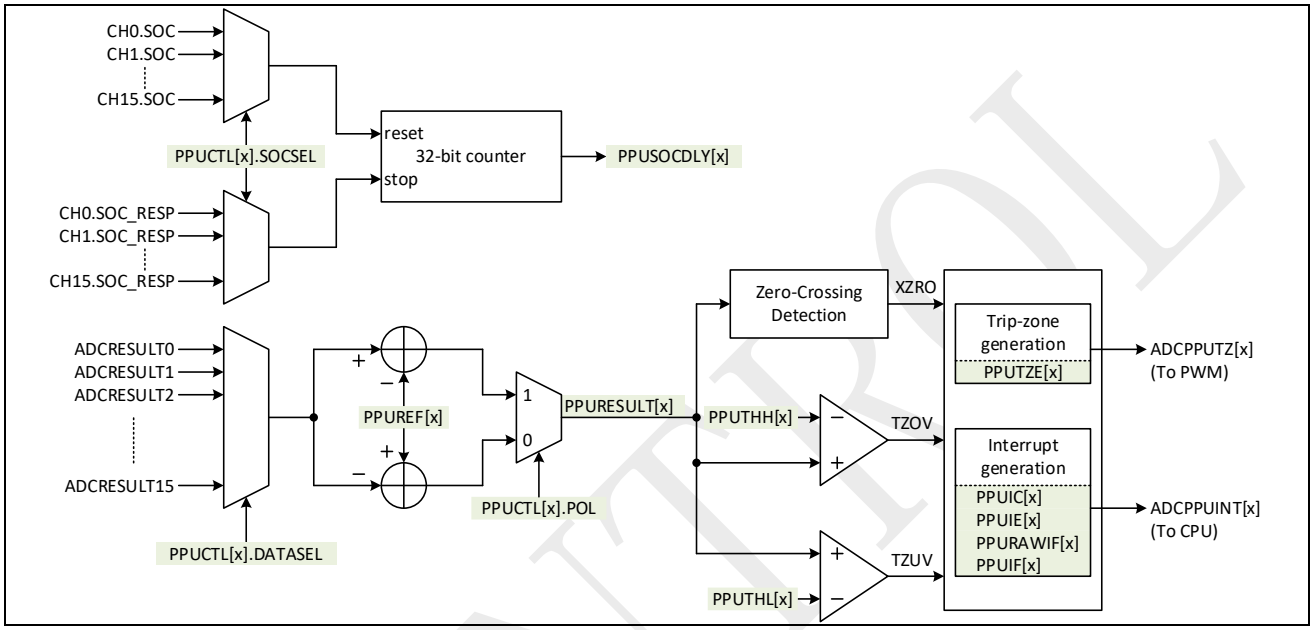
```
#include <stdio.h>
#include "spd1179.h"

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /*
```

```
    * Set PGA in  differential-ended mode.
    *
    */
    PGA_InitDPGA(DPGA_GAIN_2X);

    /*ADC Init*/
    ADC_EasyInit2(ADC, ADC_CH0, ADC_IN_DPGAP_OUT, ADC_IN_DPGAN_OUT,
ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Set Average Times */
    ADC_SetChannelResultAverageCount(ADC, ADC_CH0, ADC_AVERAGE_COUNT_16);
    while (1)
    {
        /* Use software to trigger ADC SOC0 start to work */
        ADC_ForceChannelSOC(ADC, ADC_CH0);

        /* Wait until ADC conversion finished (Interrupt flag was set) */
        while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0);

        /* Get result */
        i32VSP = ADC_GetChannelResult(ADC, ADC_CH0);

        i32VSP = ABS(i32VSP);

        /* Clear ADC SOC0 INT flag */
        ADC_ClearChannelInt(ADC, ADC_CH0);

        printf("ADC differential Result = %d\n", i32VSP);
        printf("ADC differential Voltage %dmv\n", ValueToVoltage(i32VSP));

        Delay_Ms(500);
    }
}
```

2.5 ADC 后处理单元

如手册第一章节所述，通常工程使用中，需要知道 ADC 转换的结果相对某个参考值而言是过高还是过低，这部分的工作就可以交给 PPU 单元处理，以减少 CPU 的计算压力。除此之外，PPU 单元还可以检测出从发起 ADC 转换请求至开始 ADC 转换时的时间计数。PPU 单元的框图如图 2-5：PPU 结构框图所示，每个 PPU 单元均可独立选择一路 SOC 的结果进行侦测。

图 2-5：PPU 结构框图



以下示例代码将举例说明 PPU 单元的使用方法：

- 调用 ADC_EasyInit1 初始化 ADC，并进行相关设置；
- 调用 ADC_PPU_Init 函数，选择 PPU 的输入，设置比较极性，设置参考电压值，随后使能 PPU；
- 打开 PPU 过压、欠压、过参考电压的中断；
- 设置 PPU 过压以及欠压阈值；
- 在示例代码中，当 PIN_GPIO0 超过设定阈值时，将会产生对应中断。

Example Code

```
#include <stdio.h>
#include "spd1179.h"

/*As described below, ADC result convert to voltage can calculate as the
follow*/
#define                 ValueToVoltage(x)             ((x * 3657) / 4096)


int32_t              i32Result;


void ADC_PPU_Init(ADC_PostProcessUnitEnum ePPU, ADC_ChannelEnum u8DataSel,
int32_t i32Ref, ADC_PPU_DataProcessEnum ePol)
{
    /* Enable PPU and Select ADC result */
    ADC->PPUCTL[ePPU] = ADC->PPUCTL[ePPU] | PPUCTL_EN_ENABLE;
```

```
    ADC->PPUCTL[ePPU] = ADC->PPUCTL[ePPU] & (~(PPUCTL_DATASEL_Msk |
PPUCTL_POL_Msk));

    /* Polarity for comparision */
    ADC->PPUCTL[ePPU] = ADC->PPUCTL[ePPU] | (u8DataSel << PPUCTL_DATASEL_Pos)
| (ePol << PPUCTL_POL_Pos);

    /* Set Reference */
    ADC->PPUREF[ePPU] = i32Ref;
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* ADC Init */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Init ADC PPU */
    ADC_PPU_Init(ADC_PPU0, ADC_CH0, 100, ADCRESULT_MINUS_REFERENCE);

    /* Enable zero-crossing interrupt of ADC PPU0 */
    ADC_EnablePPUInt(ADC, ADC_PPU0, ADC_PPU_INT_CROSS_ZERO);

    /* Enable too high interrupt of ADC PPU0 */
    ADC_EnablePPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_HIGH);

    /* Enable too low interrupt of ADC PPU0 */
    ADC_EnablePPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_LOW);

    /* Set PPU0 high threshold is 1500 */
    ADC_SetPPUHighThreshold(ADC, ADC_PPU0, 1500);

    /* Set PPU0 low threshold is 1000 */
    ADC_SetPPULowThreshold(ADC, ADC_PPU0, 1000);

    /* Clear CH0 INT flag */
    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Clear PPU Unit INT */
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_GLOBAL);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_CROSS_ZERO);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_HIGH);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_LOW);

    NVIC_EnableIRQ(ADCPPU0_IRQn);

    while (1)
    {
        /* Continues to use software to trigger the ADC to work */
```

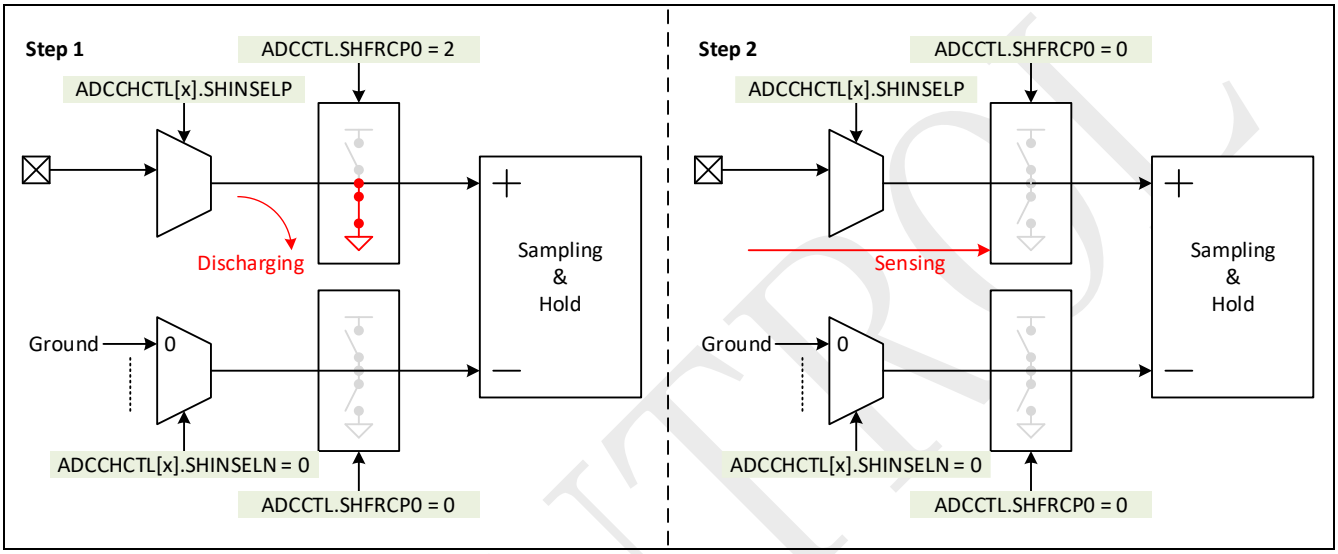
```
        ADC_ForceChannelSOC(ADC, ADC_CH0);

        Delay_Ms(100);
    }
}


void ADCPPU0_IRQHandler(void)
{
    if (ADC_GetPPUIntFlag(ADC, ADC_PPU0, ADC_PPU_INT_CROSS_ZERO) != 0)
    {
        printf("Zero-cross INT had happened\n");
    }

    if (ADC_GetPPUIntFlag(ADC, ADC_PPU0, ADC_PPU_INT_TOO_HIGH) != 0)
    {
        printf("ADC channel voltage too-high INT had happened\n");
    }

    if (ADC_GetPPUIntFlag(ADC, ADC_PPU0, ADC_PPU_INT_TOO_LOW) != 0)
    {
        printf("ADC channel voltage too-low INT had happened\n");
    }

    /* Result gotten from 'ADC_GetChannelResult()' can be a negative value or
a positive value */
    i32Result = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("ADC CH0 data is %d\n", i32Result);
    printf("ADC CH0 Voltage %dmv\n", ValueToVoltage(i32Result));

    /* Clear the PPU0 INTs and global INT of PPU0 */
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_CROSS_ZERO);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_HIGH);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_TOO_LOW);
    ADC_ClearPPUInt(ADC, ADC_PPU0, ADC_PPU_INT_GLOBAL);

    ADC_ClearChannelInt(ADC, ADC_CH0);
}
```

2.6 ADC 开路检测

如果用于模拟输入的引脚未与外部传感器正常连接，电压将无法正确测量，可能导致某些应用程序的错误。ADC 单元提供了 ADC 输入浮动检测电路，用以帮助检测此类错误。

ADC 单元支持两种可选方案来检测输入浮空，分别是基于通过预充电节点到电源和将节点接地放电两种方式。如图 2-6：检测 ADC 输入端引脚是否浮空（预放电故障注入）所示，为了检测连接到采样器正输入端的引脚是否浮动，如下描述的预放电故障注入法进行检测。

图 2-6：检测 ADC 输入端引脚是否浮空（预放电故障注入）



Step1，将输入节点放电：

- 设置 ADCCHCTL[x].SHINSELP，将采样器的正输入连接到外部引脚上；
- 将 ADCCTL.SHFRCP0 设置为 2，将采样器的正输入拉到地面上；
- 等待约 10 微秒，以便输入节点可以完全放电；

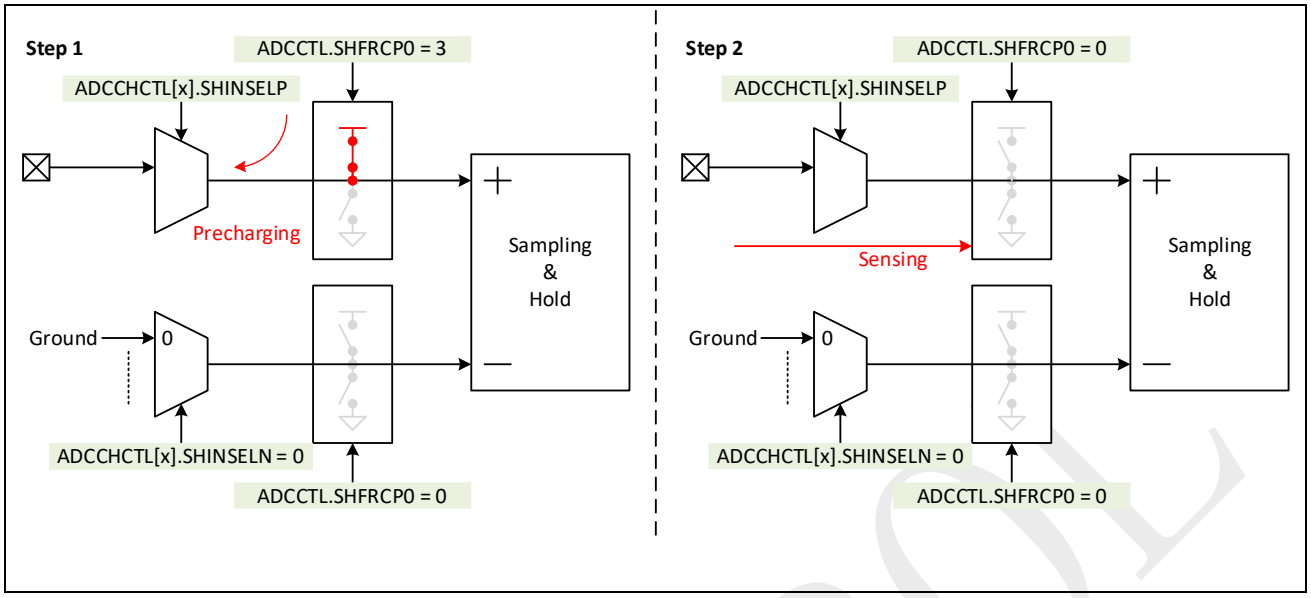
Step2，使用正常连接检查是否存在开路：

- 将 ADCCTL.SHFRCP0 设置为 0，以禁用下拉功能；
- 将 ADCCHCTL[x].SHINSELN 设置为 0，将采样器的负输入连接到地面；
- 触发通道 x 的 SOC，获取相应的 ADCRESULT[x]；

如果读回始终在 0 左右，表明采样器的输入未链接外部输入，即正输入端处于漂浮状态。类似的步骤也可以用来检测采样器的负输入引脚是否悬空。

也可以使用另一种检测输入端是否悬空的方法，即预充电故障注入法。如图 2-7：检测 ADC 输入端引脚是否浮空（预充电故障注入）所示，检测正输入端是否悬空也可以采用以下描述的两个步骤。

图 2-7：检测 ADC 输入端引脚是否浮空（预充电故障注入）



Step1，将输入端放电：

- 通过设置 ADCCHCTL[x].SHINSELP 将采样器的正输入连接到外部引脚；
- 将 ADCCTL.SHFRCP0 设置为 3，将采样器的正输入拉高到电源；
- 等待约 10us，使输入端完全预充电；

Step2，使用正常连接检查代码：

- 将 ADCCTL.SHFRCP0 设置为 0 以禁用上拉路径；
- 将 ADCCHCTL[x].SHINSELN 设置为 0，将采样器的负输入连接到地面；
- 触发通道 x 的 SOC 并获取 ADCRESULT[x]的结果；

如果读回的结果始终在 3696 左右（即 3.3 / 3.657 * 4096），则表示采样器的输入端未跟随外部输入，即正输入端悬空。类似的步骤也可以用于检测连接到采样器负输入端的引脚是否悬空。

Example Code

```
#include <stdio.h>
#include "spd1179.h"

int32_t             i32lresult;
int32_t             i32hresult;

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);
```

```
    printf("======= Bgin to try to tied the ADC0 low and then sample the
voltage ======\n");

    /* Div ADC clk, the more lower clock the more accurate the sample result
*/
    CLOCK_SetModuleDiv(ADC_MODULE, 128);
    UART_Init(UART0, 38400);

    /* Turn off the two switch */
    ADC_SetChannelSH(ADC, ADC_CH0, ADC_SH_SEL_NONE);

    /* Set PIN_GPIO0 as GPIO mode */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /*
    * Set the PIN_GPIO0 as pull down model to discharge, keep in mind that the
pull up and pull down
    * is not exclusive.
    */
    PIN_DisablePullUp(PIN_GPIO0, PIN_GPIO0_GPIO0);
    PIN_EnablePullDown(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /* Force the ADC positive PIN(ADC test point) tied to low to discharge */
    ADC_ForceSHPositiveInput(ADC, ADC_CH0, ADC_SH_FORCE_GND);

    /* Delay for a short time to let the discharge over */
    Delay_Ms(100);

    /*
    * ADC Init, config the PIN_GPIO0 as the positive channel and the negative
is GND.
    * Then On two switch.
    */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Use software to trigger ADC SOC0 to work */
    ADC_ForceChannelSOC(ADC, ADC_CH0);

    /* Wait until ADC conversion finished (Interrupt flag is set) */
    while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0)
    {
    }

    /* Get the result */
    i32lresult = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("The voltage is %d\n", i32lresult);

    /* Turn off the two switch */
    ADC_SetChannelSH(ADC, ADC_CH0, ADC_SH_SEL_NONE);

    /*
    * Last 'ADC_EasyInit1()', PIN_GPIO0 was set as analog ADC mode, but pull
up and down function
    * is only valid in GPIO mode.
    */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /*
    * Set the PIN_GPIO0 as pull up model to precharge, keep in mind that the
pull up and pull down
```

```
    * is not exclusive.
    */
    PIN_EnablePullUp(PIN_GPIO0, PIN_GPIO0_GPIO0);
    PIN_DisablePullDown(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /* Force the ADC positive PIN tied high to charge */
    ADC_ForceSHPositiveInput(ADC, ADC_CH0, ADC_SH_FORCE_VDD33);

    /* Delay for a short time to let the precharge over */
    Delay_Ms(100);

    /*
    * ADC Init, config the PIN_GPIO0 as the positive channel and the negative
is GND.
    * Then On two switch.
    */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Use software to trigger ADC SOC0 to work */
    ADC_ForceChannelSOC(ADC, ADC_CH0);

    /* Wait until ADC conversion finished (Interrupt flag is set) */
    while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0)
    {
    }

    /* Get the ADC SOC result */
    i32hresult = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("The voltage is %d\n", i32hresult);

    /* Check the result */
    if (ABS(i32lresult) < 200 && ABS(i32hresult) > 3200)
    {
        printf("PIN_GPIO0 is disconnected\n");
    }
    else
    {
        printf("PIN_GPIO0 is connected\n");
    }

    while (1)
    {

    }
}
```

2.7 ADC 短路检测

模拟输入可能存在短路情况，可以使用预充电或放电电路进行故障注入来检测，具体步骤如下。

使用正常连接获取代码：

- 将 ADCCTL.SHFRCP0 和 ADCCTL.SHFRCN0 设置为 0，以禁用采样器输入的上拉和下拉；
- 通过 ADCCHCTL[x].SHINSELP 和 ADCCHCTL[x].SHINSELN 选择期望的引脚和通道；
- 触发通道 x 的 SOC 并获取第 1 个 ADCRESULT[x]；

通过预充电进行故障注入方式进行检测，则执行如下步骤：

- 根据采样器连接到模拟输入引脚的节点，将 ADCCTL.SHFRCP0 及 ADCCTL.SHFRCN0 设置为 3 以启用预充电；
- 等待约 10us，以使输入节点完全预充电；
- 触发通道 x 的 SOC 并获取第 2 个 ADCRESULT[x]；
- 通过放电进行故障注入方式进行检测，则执行如下步骤：
- 根据采样器连接到模拟输入引脚的节点，将 ADCCTL.SHFRCP0 或 ADCCTL.SHFRCN0 设置为 2 以启用放电；
- 等待约 10us，以使输入节点完全放电；
- 触发通道 x 的 SOC 并获取第 3 个 ADCRESULT[x]；

如果三个 ADCRESULT[x]几乎相同，则认为输入引脚存在短路。

注意：这种情况只有在引脚短接到低阻抗节点时才能检测到，此时该节点的驱动强度足够强，可以覆盖由上拉和下拉电路提供的 8uA 预充电和放电能力。

Example Code

```
#include <stdio.h>
#include "spd1179.h"

int32_t             VALUE1, VALUE2, VALUE3;

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("======= Bgin to try to tied the ADC0 low and then sample the
voltage ======\n");

    /* Div ADC clk, the more lower clock the more accurate the sample result
```

```
*/
    CLOCK_SetModuleDiv(ADC_MODULE, 128);
    UART_Init(UART0, 38400);

    /*
    * ADC Init, config the PIN_GPIO0 as the positive channel and the negative
is GND.
    * Turn On two switch.
    */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Use software to trigger ADC SOC0 to work */
    ADC_ForceChannelSOC(ADC, ADC_CH0);

    /* Wait until ADC conversion finished (Interrupt flag is set) */
    while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0)
    {
    }

    /* Get the result */
    VALUE1 = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("The voltage is %d\n", VALUE1);

    /* Turn off the two switch */
    ADC_SetChannelSH(ADC, ADC_CH0, ADC_SH_SEL_NONE);

    /* Set PIN_GPIO0 as GPIO mode */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /*
    * Set the PIN_GPIO0 as pull down model to discharge, keep in mind that the
pull up and pull down
    * is not exclusive.
    */
    PIN_DisablePullUp(PIN_GPIO0, PIN_GPIO0_GPIO0);
    PIN_EnablePullDown(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /* Delay for a short time to let the discharge over */
    Delay_Ms(100);

    /*
    * ADC Init, config the PIN_GPIO0 as the positive channel and the negative
is GND.
    * Then On two switch.
    */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Use software to trigger ADC SOC0 to work */
    ADC_ForceChannelSOC(ADC, ADC_CH0);

    /* Wait until ADC conversion finished (Interrupt flag is set) */
    while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0)
    {
    }

    /* Get the result */
    VALUE2 = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("The voltage is %d\n", VALUE2);
```

```
    /* Turn off the two switch */
    ADC_SetChannelSH(ADC, ADC_CH0, ADC_SH_SEL_NONE);

    /*
    * Last 'ADC_EasyInit1()', PIN_GPIO0 was set as analog ADC mode, but pull
up and down function
    * is only valid in GPIO mode.
    */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /*
    * Set the PIN_GPIO0 as pull up model to precharge, keep in mind that the
pull up and pull down
    * is not exclusive.
    */
    PIN_EnablePullUp(PIN_GPIO0, PIN_GPIO0_GPIO0);
    PIN_DisablePullDown(PIN_GPIO0, PIN_GPIO0_GPIO0);

    /* Delay for a short time to let the precharge over */
    Delay_Ms(100);

    /*
    * ADC Init, config the PIN_GPIO0 as the positive channel and the negative
is GND.
    * Then On two switch.
    */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_SOFTWARE);

    ADC_ClearChannelInt(ADC, ADC_CH0);

    /* Use software to trigger ADC SOC0 to work */
    ADC_ForceChannelSOC(ADC, ADC_CH0);

    /* Wait until ADC conversion finished (Interrupt flag is set) */
    while (ADC_GetChannelIntFlag(ADC, ADC_CH0) == 0)
    {
    }

    /* Get the ADC SOC result */
    VALUE3 = ADC_GetChannelResult(ADC, ADC_CH0);
    printf("The voltage is %d\n", VALUE3);

    /* Check the result */
    if ((ABS(VALUE1 - VALUE2) < 200) && (ABS(VALUE1 - VALUE3) < 200))
    {
        printf("PIN_GPIO0 is short\n");
    }
    else
    {
        printf("PIN_GPIO0 is not short\n");
    }

    while (1)
    {

    }
}
```

3 常见问题 QA

3.1 ADC 转换结果为负数

如果 ADC 正端电压高于负端，则 ADC 转换结果会有负的码值，此时负码值，并非表示 ADC 输入端的某个端口电压为负电压，应该称之为相对负电压或者差分负电压。如手册第一章所述，SPC1169/SPD1179/SPD1176 的 ADC 端口的输入的绝对电压不允许为负电压，否则 ADC 将无法正常工作。以 S/H A 为例：

Voltage ADC = AIP - AIN

其中，0 < AIP < +3.657，0 < AIN < +3.657

则在 ADC 端电压范围为：

-3.657 < Voltage ADC < +3.657

此时转换出的码值就分为两种量程[0, 8191]，[-4096, 4095]，本质只是表示方法不同，SDK 中提供了对应的两种接口可供调用。

Example Code

```
/*******************************************************************************
 * @brief      Get SH[x] result after EOC (End Of Convertion)
 *
 * @param[in]  eSHx  : Sampling Holder x defined by ADC_SamplingHolderEnum
 *                     Following value is valid:
 *                       - \ref ADC_SH0
 *
 * @return     Signed value.
 *             Range:
 *               [   -4096
 *                 :  4095
 *               ]
 *
*******************************************************************************/
#define ADC_GetSHResult(ADCx,eSHx)
\
    ( (int32_t)READ_REG( (ADCx)->SHRAWCODE[eSHx] ) )

/*******************************************************************************
 * @brief      Get SH[x] raw result after EOC (End Of Convertion)
 *
 * @param[in]  eSHx  : Sampling Holder x defined by ADC_SamplingHolderEnum
 *                     Following value is valid:
 *                       - \ref ADC_SH0
 *
 * @return     Unsigned value.
 *             Range: [ 0 : 8191 ]
 *
*******************************************************************************/
#define ADC_GetSHRawResult(ADCx,eSHx)
\
    ( (uint32_t)( ADC_GetSHResult(ADCx, eSHx) + 4096 ) )

/*******************************************************************************
 * @brief      Get CH[x] result value
 *
```

```
 * @param[in]  eCHx:  Channel x defined by ADC_ChannelEnum
 *                    Following value is valid:
 *                      - \ref ADC_CH0 ~ ADC_CH15
 *
 * @return     Signed value.
 *             Range:
 *               [   -4096
 *                 :  4095
 *               ]
 *
*******************************************************************************/
#define ADC_GetChannelResult(ADCx,eCHx)
\
    ( (int32_t)READ_REG( (ADCx)->ADCRESULT[eCHx] ) )

/*******************************************************************************
 * @brief      Get CH[x] raw result value
 *
 * @param[in]  eCHx:  Channel x defined by ADC_ChannelEnum
 *                    Following value is valid:
 *                      - \ref ADC_CH0 ~ ADC_CH15
 *
 * @return     Unsigned value.
 *             Range: [ 0 : 8191 ]
 *
*******************************************************************************/
#define ADC_GetChannelRawResult(ADCx,eCHx)
\
    ( (uint32_t)( ADC_GetChannelResult(ADCx, eCHx) + 4096 ) )

/*******************************************************************************
 * @brief      For Single End Mode Only (One terminal is GND)
 *             Get Trim result(code) from ADCx result register
 *
 * @param[in]  eCHx:  Channel x defined by ADC_ChannelEnum
 *                    Following value is valid:
 *                      - \ref ADC_CH0 ~ ADC_CH15
 *
 * @return     Signed value.
 *             Range: [ 0 : 8191 ]
 *
*******************************************************************************/
#define ADC_GetChannelAbsoluteResult(ADCx,eCHx)
\
    ( ABS( ADC_GetChannelResult(ADCx, eCHx) ) )
```

所以请注意，当使用 ADC 进行单端采样时，即使被测电压是绝对正电压，但如图 3-1：使用 ADC 负端进行采样所示，是若将此电压接在 ADC 负端，则测量出来的码值采用[-4096, 4095]量程的话，此时最后的码值是负值，但并不代表输入电压是绝对负电压。

图 3-1：使用 ADC 负端进行采样