Freescale Semiconductor
应用笔记

Document Number: AN5015
Rev 0, 09/2014

MPC574xP ADC 自检

作者: Arun Kumar, Sanjoy Dey, 和 Jamaal Fraser

内容

1 介绍 1
2 ADC 自检功能介绍 1
3 ADC 自检参数 2
4 基于软件的比较考量 3
5 示例代码 5

1 介绍

逐次逼近型寄存器（SAR）模数转换器（ADC）支持内建的运行时硬件自检，以验证 ADC 的运行情况。ADC 的自检功能支持电源完整性和结构组件的完整性测试，如电容、开关、比较器等。该功能的目的是捕捉和标记导致 ADC 功能失效的任何运行时灾难性错误。ADC 自检包括两种不同的自检：

- 电源自检：也称为算法 S，用于验证带隙、电源（VDD_HV_ADV）和参考（VDD_HV_ADR）电压
- 电容自检：也称为算法 C，用于检测电容阵列的开路或短路

本文档详细描述了运行 ADC 自检功能所需的补充信息。同时提供了两个用例样本，帮助用户了解如何对 ADC 自检功能编程。

2 ADC 自检功能介绍

对于极关键应用中的安全装置，务必定期检查 ADC 是否正常工作。为了达到该目的，自检功能已整合到了 ADC 内部。自检采用模拟看门狗来验证自检转换的结果。这些看门狗的上下限阈值存储在 UTest flash 区域。运行自检之前，用户必须将这些值从 UTest flash 复制到自检模拟看门

狗寄存器（ADC_STAWxR）或直接将它们的值编程到 ADC_STAWxR 寄存器。ADC 还具有看门狗定时器，可用于监测自检算法的序列并确保其在安全的时间段内完成。

ADC 内部已经实现了两种自检算法：

- 电源自检（算法 S）：它包括内部带隙电压、ADC 供电电压和 ADC 参考电压的转换。它包括应按顺序执行的三步测试转换（步骤 S0-S2）。
- 电容自检（算法 C）：它包括 12 步测试转换，用于设置包括采样 DAC 电容在内的电容元件。

为了完成自检，ADC 实现了以下功能：

- 设置额外的专用自检通道
- 采用配置寄存器发送信号去调度自检算法
- 采用模拟看门狗寄存器监测转换后的数据，并标记 ADC 输出端口的错误（故障收集和控制单元（FCCU））以防任何算法失败

参见以下的 图 1 。

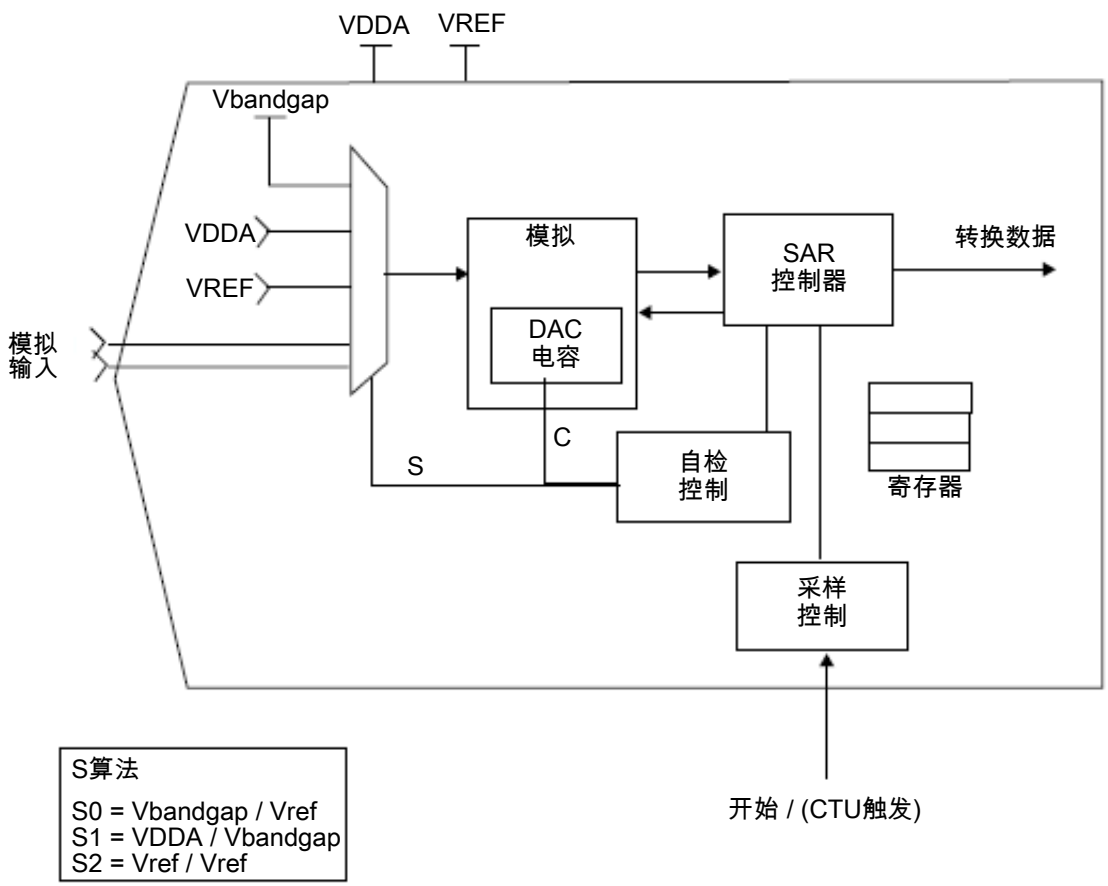

图 1. ADC 自检功能的结构框图

3 ADC 自检参数

ADC 使用两类参数设置来定义 ADC 自检操作：

- 采样阶段持续时间被编程写入自检配置寄存器（ADC_STCR1）的 INPSAMP_S （算法 S）和 INPSAMP_C （算法 C）位字段。
- 模拟看门狗的上限阈值（THRH）和下限阈值（THRL）被编程写入 ADC_STAWxR 寄存器。

采样阶段持续时间设置定义了 ADC 采样阶段所需的时间。如上所述，每一个算法（C 和 S）都有专门的寄存器字段来定义采样阶段持续时间。推荐的设置如下 表 1 所示。

表 1. 采样阶段设置

寄存器字段	推荐设置
ADC_STCR1[INPSAMP_C]	0x18
ADC_STCR1[INPSAMP_S]	0x50[1]

1. 推荐的设置对应于较长的采样时间，这是因为低温时采样电容在 S0 算法下的建立时间较长。这不是寄存器默认设置。

模拟看门狗所使用的阈值存储在 UTest flash 里，在配置时被用户应用程序获取并加载到 ADC_STAWxR 寄存器。 下面的 表 2 定义了 UTest flash 到相应的 ADC_STAWxR 的映射。

表 2. ADC 自测阈值的采样值

Flash 地址	Flash 中的值	寄存器	步骤	THRH	THRL	THRH（有符号的）	THRL（有符号的）
0x004000D0	0xF75AF4DF	STAW0R	S0_3.3V	0x75A	0x4DF	1882	1247
0x004000D4	0xF4D0F2DB	STAW0R	S0_5.0V	0x4D0	0x2DB	1232	731
0x004000D8	0xF003F002	STAW1AR	S1(INT)	0x3	0x2	3	2
0x004000DC	0xF3D9F1E3	STAW1BR	S1(FRAC)	0x3D9	0x1E3	985	483
0x004000E0	0xFFFFFFF9	STAW2R	S2	---	0xFF9	---	4089
0x004000E4	0xF010FFF0	STAW4R	C0	0x010	0xFF0	16	-16
0x004000E8	0xF010FFF0	STAW5R	C1-C11	0x010	0xFF0	16	-16

推荐的模拟看门狗阈值是在纯净的 ADC 工作环境下（外部环境引入的噪声最小），根据预期结果和测试结果进行设置的。在一个比较嘈杂的环境下，采用这些模拟看门狗阈值运行自检操作可能会失败。如果应用程序可以容忍现有噪声水平，用户可以放宽阈值来克服失败。阈值幅度放宽得越大，则有越多的真实错误能通过。例如，在 ADC_STAW4R 和 ADC_STAW5R 寄存器的值中，THRH 为 0x010（16d），THRL 为 0xFF0（-16d），意味着算法 C 会因为一个大于（16 / 8 = 2）2LSB@12b 的错误而失败。这是适用于纯净环境的推荐值。同样地，THRH 为 0x020（32d）和 THRL 为 0xFE0（-32d）意味着算法 C 会因为大于 4LSB@12b 的错误而失败。在应用程序可以容忍噪声水平的情况下，为了使自检在嘈杂环境中通过，阈值可以从+/-16 放宽到+/-32，或者根据需要调整。然而，最好还是对环境进行纯化以达到预期，从而更好的操作，而不是放宽阈值。放宽阈值太多（超过+/-32），可导致虚假通过，不建议用于安全应用。由于噪声而进行的自检阈值放宽与一般的使用转换没有直接关系，就阈值放宽而言，不可能知道它会对测量产生什么影响。噪声因应用而异，超出了设备控制/判断的范畴。在这种情况下，采用默认的阈值而不通过，是由于外部噪声而不是内部电容所造成的。当噪声在系统中的影响超过实际电容的错误，阈值会放宽；噪声分量的幅度过高也会使得阈值需要放宽。因此，在现实中，每当测试运行环境出现噪声时，如果有任何电容错误导致的噪声偏移，它会被添加到这个环境噪声，并会导致违反放宽的阈值限制。自检设计用于捕捉和标记运行时错误和灾难性错误。它不能也不应该用于规格限制下的测试。

4 基于软件的比较考量

可以采用软件方法比较自检数据寄存器 (ADC_STDR1/2) 中的 ADC 自检转换结果与上限阈值（THRH）及下限阈值 (THRL)，其中上限阈值（THRH）和下限阈值（THRL）存储于 UTest flash 或被编程写入 ADC_STAWxR 寄存器。

基于软件的比较方法允许用户对两个或两个以上的 ADC 转换数据结果求均值，可以减少系统、接地和/或电源噪声的影响。

- 算法 S 步骤 0 (S0)、算法 S 步骤 2 (S2) 和算法 C 的所有步骤 (Cn) 可以对每次转换后 STDR1.TCDATA 的结果简单地求均值。
- 对于算法 S 步骤 1 (S1)，每次转换后即存在整数部分 (STDR2.IDATA)，也存在小数部分 (STDR2.FDATA)，所以用户软件必须两者兼顾。

考虑 S1 整数部分和小数部分的比较有两种可能的方式:

1. IDATA 和 FDATA 结果与 THRH 和 THRL 进行逻辑比较
2. 计算电压比较

这两种方法会在下面的章节中详细说明。

4.1 S1 算法: 逻辑比较

用户软件可以实现将 IDATA 和 FDATA 结果与 THRH 和 THRL 进行逻辑逐步比较，由于 S1 结果由整数部分 (IDATA) 和小数部分 (FDATA) 组成，有必要考虑基于流程图的方法来比较 S1 结果与 THRH 和 THRL。

如下图 2 展示了比较 IDATA 和 FDATA 结果与 THRH 和 THRL 值的逻辑要求。

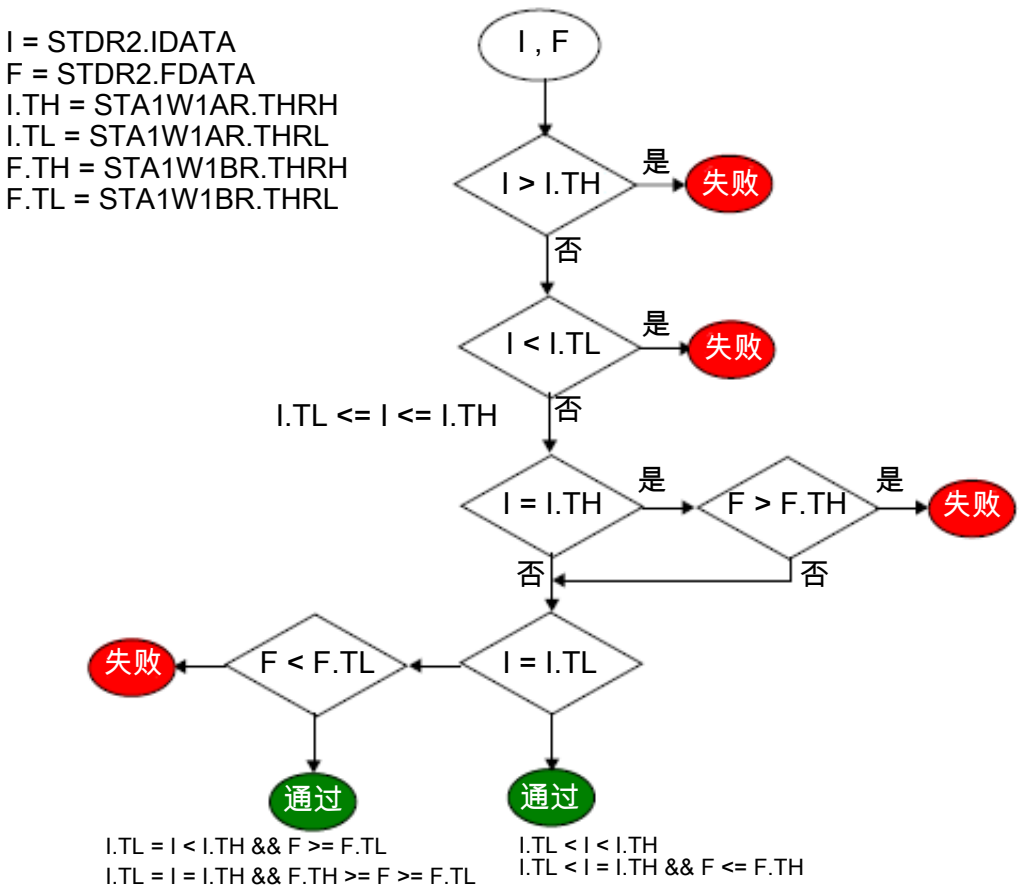

图 2. 将 IDATA 和 FDATA 与 THRH 和 THRL 值进行比较的流程图

4.2 S1 算法: 计算比较

用户软件可以将算法 S 步骤 1 的结果和相应的阈值转换成电压，然后比较电压结果。

本方法所涉及的步骤如下:

1. 计算上限阈值电压:

$$THRH\_CALC = \left(STAW1AR[THRH] + \frac{STAW1BR[THRH]}{4096}\right) \times V_{bandgap}$$

2. 计算下限阈值电压:

$$THRL\_CALC = \left(STAW1AR[THRL] + \frac{STAW1BR[THRL]}{4096}\right) \times V_{bandgap}$$

3. 计算 S1 电压:

$$S1\_CALC = \left(STDR2[IDATA] + \frac{STDR2[FDATA]}{4096}\right) \times V_{bandgap}$$

4. 比较步骤 3 与步骤 1 和步骤 2 的结果。如果满足以下条件，则算法 S 步骤 1 通过检测:

   THRL_CALC <= S1_CALC <= THRH_CALC

5 示例代码

以下示例代码为一个典型用例，介绍如何设置监测各种电容（开路/短路）和电源（在范围内/超出范围）的模拟看门狗以及 ADC 看门狗定时器（ADC 转换在看门狗时间内完成）。

提供了两个实例功能。

- 以单次模式运行算法 S，模拟看门狗使能。
- 以扫描模式运行算法 S+C，模拟看门狗和看门狗定时器使能。

```
uint8_t ADC_self_test_one_shot(volatile struct ADC_tag *ADC)
{
    // ADC Configuration
    ADC->MCR.B.ADCLKSEL = 1;        // ADC clock = ipg_clk
    ADC->MCR.B.MODE = 0;            // One Shot Mode
    ADC->MCR.B.PWDN = 0;            // Exit from power down state
    ADC->MCR.B.OWREN = 1;           // Enable overwrite

    // IMR (Interrupt Mask register)
    ADC->IMR.R = 0x0;               // Disable all interrupts

    // CTR0/1 Need to be set for 1us @ 80 MHz. Leave sampling at default
    ADC->CTR0.R = 0x0000002C;       // = 44
    ADC->CTR1.R = 0x0000002C;       // = 44, TSENSOR = 0

    // NCMR: Enable normal sampling for for Channel 10 (Band Gap)
    ADC->NCMR0.R = 0x00000400;

    // Set up for Self Test
    ADC->STSR1.B.ST_EOC = 0x1;                              // Clear End of Conversion flag
    ADC->STCR1.R = 0x18005000;                              // Self test Sampling Settings
    ADC->STCR3.B.ALG = 0x0;                                 // Self test Algorithm S
    ADC->STCR2.R = 0x00000080;                              // Enable Self test
    ADC->STBRR.R = 0x00000000;                              // BR=0 WDT=0.1ms
    ADC->STAW0R.R = 0x0fff0fff & (*(uint32_t *)0x004000D0);    // S step0
    ADC->STAW1AR.R = 0x0fff0fff & (*(uint32_t *)0x004000D8);   // S step1
    ADC->STAW1BR.R = 0x0fff0fff & (*(uint32_t *)0x004000DC);   // S step1
    ADC->STAW2R.R = 0x00000fff & (*(uint32_t *)0x004000E0);    // S step2
    ADC->STAW4R.R = 0x0fff0fff & (*(uint32_t *)0x004000E4);    // C step 0
    ADC->STAW5R.R = 0x0fff0fff & (*(uint32_t *)0x004000E8);    // C step 1 to 11

    ADC->STAW0R.B.AWDE = 1;        // S analog WDT enable
    ADC->STAW1AR.B.AWDE = 1;       // S analog WDT enable
```

```
    ADC->STAW2R.B.AWDE = 1;        // S analog WDT enable

    // Initiate conversion and Step 0
    ADC->STCR3.B.MSTEP = 0;             // MSTEP = 0
    ADC->MCR.B.NSTART = 1;              // Start the ADC trigger
    while(ADC->STSR1.B.ST_EOC == 0);    // Wait for end of conversion flag
    ADC->STSR1.B.ST_EOC = 0x1;          // Clear end of conversion flag

    if(ADC->STSR1.R != 0)               // Check for any errors
        return(ERROR);

    if(!ADC->CDR[10].B.VALID)           // Verify Band Gap sample is valid
        return(ERROR);

    if(!ADC->STDR1.B.VALID)             // Verify Self Test sample is valid
        return(ERROR);

    // Initiate conversion and Step 1
    ADC->STCR3.B.MSTEP = 1;             // MSTEP = 1
    ADC->MCR.B.NSTART = 1;              // Start the ADC trigger
    while(ADC->STSR1.B.ST_EOC == 0);    // Wait for end of conversion flag
    ADC->STSR1.B.ST_EOC = 0x1;          // Clear end of conversion flag

    if(ADC->STSR1.R != 0)               // Check for any errors
        return(ERROR);

    if(!ADC->CDR[10].B.VALID)           // Verify Band Gap sample is valid
        return(ERROR);

    if(!ADC->STDR2.B.VALID)             // Verify Self Test sample is valid
        return(ERROR);

    // Initiate conversion and Step 2
    ADC->STCR3.B.MSTEP = 2;             // MSTEP = 2
    ADC->MCR.B.NSTART = 1;              // Start the ADC trigger
    while(ADC->STSR1.B.ST_EOC == 0);    // Wait for end of conversion flag
    ADC->STSR1.B.ST_EOC = 0x1;          // Clear end of conversion flag

    if(ADC->STSR1.R != 0)               // Check for any errors
        return(ERROR);

    if(!ADC->CDR[10].B.VALID)           // Verify Band Gap sample is valid
        return(ERROR);

    if(!ADC->STDR1.B.VALID)             // Verify Self Test sample is valid
        return(ERROR);

    ADC->STCR2.B.EN = 0;                // Disable Self Test
    ADC->MCR.B.PWDN = 1;                // Enter power down state

    return(PASS);
}

uint8_t ADC_self_test_scan(volatile struct ADC_tag *ADC)
{
    int i;

    // ADC Configuration
    ADC->MCR.B.ADCLKSEL = 1;            // ADC clock = ipg_clk
    ADC->MCR.B.MODE = 1;                // Scan mode
    ADC->MCR.B.PWDN = 0;                // Exit from power down state
    ADC->MCR.B.OWREN = 1;               // Enable overwrite

    // IMR (Interrupt Mask register)
    ADC->IMR.R = 0x0;                   // Disable all interrupts

    // CTR0/1 Need to be set for 1us @ 80 MHz. Leave sampling at default
    ADC->CTR0.R = 0x0000002C;           // = 44
    ADC->CTR1.R = 0x0000002C;           // = 44, TSENSOR = 0
```

```
    // NCMR: Enable normal sampling for for Channel 10 (Band Gap)
    ADC->NCMR0.R = 0x00000400;

    // Set up for Self Test
    ADC->STCR1.R = 0x18005000;                                  // Self test Sampling Settings
    ADC->STCR3.R = 0x00000300;                                  // S+C for SCAN, MSTEP = 0.
    ADC->STCR2.R = 0x00000080;                                  // Enable Self Test
    ADC->STBRR.R = 0x00000000;                                  // BR=0 WDT=0.1ms
    ADC->STAW0R.R = 0x0fff0fff & (*(uint32_t *)0x004000D0);     // S step0
    ADC->STAW1AR.R = 0x0fff0fff & (*(uint32_t *)0x004000D8);    // S step1
    ADC->STAW1BR.R = 0x0fff0fff & (*(uint32_t *)0x004000DC);    // S step1
    ADC->STAW2R.R = 0x00000fff & (*(uint32_t *)0x004000E0);     // S step2
    ADC->STAW4R.R = 0x0fff0fff & (*(uint32_t *)0x004000E4);     // C step 0
    //ADC->STAW5R.R = 0x0fff0fff & (*(uint32_t *)0x004000E8);   // C step 1 to 11
    ADC->STAW5R.R = 0x00200FE0;                                 // C step 1 to 11
                                                                // Change to +/-32 per note

    ADC->STAW0R.B.AWDE = 1;         // S analog WDT enable
    ADC->STAW1AR.B.AWDE = 1;        // S analog WDT enable
    ADC->STAW2R.B.AWDE = 1;         // S analog WDT enable
    ADC->STAW4R.B.AWDE = 1;         // C analog WDT enable

    ADC->MCR.B.NSTART = 1;          // Start the ADC trigger

    // wait loop for desired period
    for(i=0 ; i <180000 ;i++);

    ADC->STAW0R.B.AWDE = 0;         // S analog WDT disable
    ADC->STAW1AR.B.AWDE = 0;        // S analog WDT disable
    ADC->STAW2R.B.AWDE = 0;         // S analog WDT disable
    ADC->STAW4R.B.AWDE = 0;         // C analog WDT disable

    ADC->MCR.B.NSTART = 0;          // Stop the ADC trigger
    ADC->STCR2.B.EN = 0;            // Disable Self test

    if(ADC->STSR1.R != 0x00800000) // Check for any errors
        return(ERROR);
    else
        return(PASS);
}
```

用户可能会发现，增加将 ADC 自检转换结果存储到 SRAM 的软件有利于故障排除。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5015
Revision 0, 09/2014