Document Number: AN5015
Rev 0, 09/2014

MPC574xP ADC Self Test

by: Arun Kumar, Sanjoy Dey, and Jamaal Fraser

Contents

1 Introduction

The Successive Approximation Register (SAR) Analog Digital Converter (ADC) supports run-time hardware built in self test to verify the operation of the ADC. The ADC self test feature supports the testing of power supply integrity and structural component integrity, e.g. capacitors, switches, and comparators etc. The goal of this feature is to catch and flag any run-time catastrophic errors leading to ADC functional failure. The ADC self test includes two different self tests:

- Supply self test: Also referred to as algorithm S it is used to verify the bandgap, supply (VDD_HV_ADV) and reference (VDD_HV_ADR) voltages
- Capacitive self test: Also referred to as algorithm C it is used to to check for opens or shorts in the capacitive array

This document details supplemental information required to operate the ADC self test feature. Two use case samples are also given to help users understand how to program the ADC self test feature.

2 ADC Self test feature description

For safety devices used in very critical applications it is important to check at regular intervals that the ADC is functioning correctly. For this purpose the self testing feature has been incorporated inside the ADC. The self tests use analog watchdogs to verify the results of the self test conversions. The upper and lower thresholds of these watchdogs are saved in the UTest flash area. Before running the self test the user must copy these values from the UTest flash to the Self Test Analog Watchdog Registers (ADC_STAWxR) or directly program their own values into the ADC_STAWxR registers. The ADC also have watchdog timers that can be used to monitor the sequence of the self test algorithm and ensure that it completes within a safe time period.

Two types of self testing algorithms have been implemented inside the ADC:

- Supply self test (algorithm S): It includes the conversion of the internal bandgap voltage, ADC supply voltage, and ADC reference voltage. It includes a sequence of three test conversions (steps S0-S2) that should be executed sequentially.
- Capacitive self test (algorithm C): It includes a sequence of 12 test conversions (steps) which set the capacitive elements comprising the sampling DAC capacitors.

The ADC implements the following functions in order accomplish self testing:

- An additional test channel dedicated for self tests
- Signals to schedule self test algorithms using configuration registers
- Monitors the converted data using analog watchdog registers and flags the error at the output port of the ADC (for the Fault Collection and Control Unit (FCCU)) in case any of the algorithms fail

See Figure 1 below.

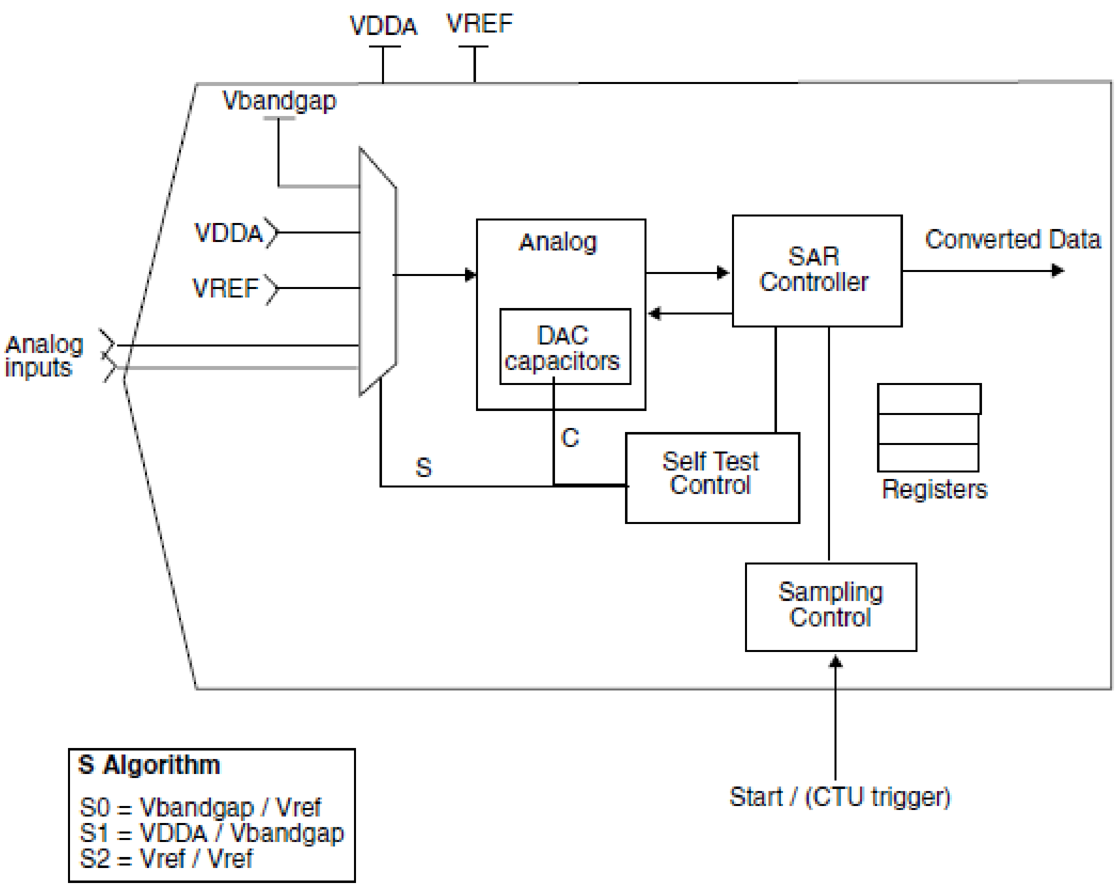

Figure 1. ADC block diagram with self test feature

3 ADC Self test parameters

The ADC uses two types of parameter settings to define the ADC self test operation:

- Sample phase duration settings programmed into the INPSAMP_S (algorithm S) and INPSAMP_C (algorithm C) bit fields of the Self-Test Configuration Register (ADC_STCR1).
- Upper threshold (THRH) and lower threshold (THRL) values for the analog watchdogs programmed into the ADC_STAWxR registers.

Sample phase duration settings define the amount of time required during the ADC sample phase. As noted above each algorithm (C and S) has a dedicated register field to define the sample phase duration. The recommended settings are shown in Table 1 below.

Table 1. Sample Phase Settings

Register field	Recommended Setting
ADC_STCR1[INPSAMP_C]	0x18
ADC_STCR1[INPSAMP_S]	0x50[1]

1. Recommended setting is for a longer sample time due to slow sample capacitor settling time at low temperature for S0 algorithm. This is not the register default.

The threshold values used by the analog watchdogs are stored in UTest flash and retrieved by the user application at configuration time and loaded into ADC_STAWxR registers. Table 2 below defines the UTest flash mapping to the corresponding ADC_STAWxR.

Table 2. Sample Values for ADC Self-Test Thresholds

Flash Address	Value in Flash	Registers	Step	THRH	THRL	THRH (signed)	THRL (signed)
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

The recommended analog watchdog threshold values are set according to the expected and tested results with a clean environment (noise introduction from external environment minimized) for ADC operation. In a more noisy set up running the self test with these analog watchdog values may cause failures. To overcome the failures the user can relax the threshold values if the needs of the application can tolerate the existing noise level. The more the threshold values are relaxed the more the magnitude of true error that can pass. For example, looking at the values for the ADC_STAW4R and ADC_STAW5R registers, THRH of 0x010 (16d) and THRL of 0xFF0 (-16d), means the algorithm C will fail for an error of magnitude greater than (16/8 =2) 2LSB@12b. This is the recommended value for a clean environment. Similarly, THRH of 0x020 (32d) and THRL of 0xFE0 (-32d) means the algorithm C will fail for an error of a magnitude greater than 4LSB@12b. So to pass the self test in a noisy environment where the application can live with the noise level the thresholds can be relaxed from +/-16 to +/-32 or as needed. However, it is preferable that the environment is cleaned up to match the expectation for better operation instead of threshold relaxation. Relaxing the thresholds too much (more than +/-32) can lead to a false pass and is not recommended for safety applications. Relaxation of the self test thresholds due to noise does not directly relate to normal use conversions and it is not possible to tell what the noise impact to those measurements is with respect to the threshold relaxation. The noise is application specific and not under control/judgment of the device. In this scenario not passing with the default threshold values are caused by external noise and not by the constituent capacitors. Limits are relaxed knowing that there is noise in the system that rides on top of the actual capacitor error and the magnitude of the noise component is so high that limits need to be relaxed as a result. So in reality whenever tests are run environment noise will be there and if there is any true shift of the capacitor error noise it will be added to this environment noise and should result in a violation of the relaxed threshold limits. The self test is designed to catch and flag run-time and catastrophic errors. It is not able to and not meant to test for specification limits.

4 Considerations for software based comparison

It is possible to implement software methods comparing the ADC self test converted results in Self Test Data registers (ADC_STDR1/2) to the high threshold (THRH) and low threshold (THRL) values that are stored in UTest flash or programmed into ADC_STAWxR registers.

Software based comparison methods allow the users to average two or more ADC converted data results which can reduce the effects of system, ground, and/or power supply noise.

- Algorithm S step 0 (S0), algorithm S step 2 (S2) and all algorithm C steps (Cn) can be simply averaged using the results present in STDR1.TCDATA after every conversion.
- For algorithm S step 1 (S1) both the integer data (STDR2.IDATA) and fractional data (STDR2.FDATA) are available for each conversion so the user software must account for both of them.

There are two possible ways to account for S1 integer and fractional parts:

1. A logical comparison of IDATA and FDATA results against THRH and THRL
2. Calculated voltage comparison

These two methods are detailed in the following sections.

4.1 S1 Algorithm: Logical comparison

The user software can implement a logical step-by-step comparison of IDATA and FDATA results against THRH and THRL. Since the S1 results are represented by both an integer part (IDATA) and a fractional part (FDATA) it is necessary to consider a flow chart based approach to compare the S1 results against THRH and THRL.

Figure 2 below illustrates the logical requirements for comparing IDATA and FDATA results to THRH and THRL.

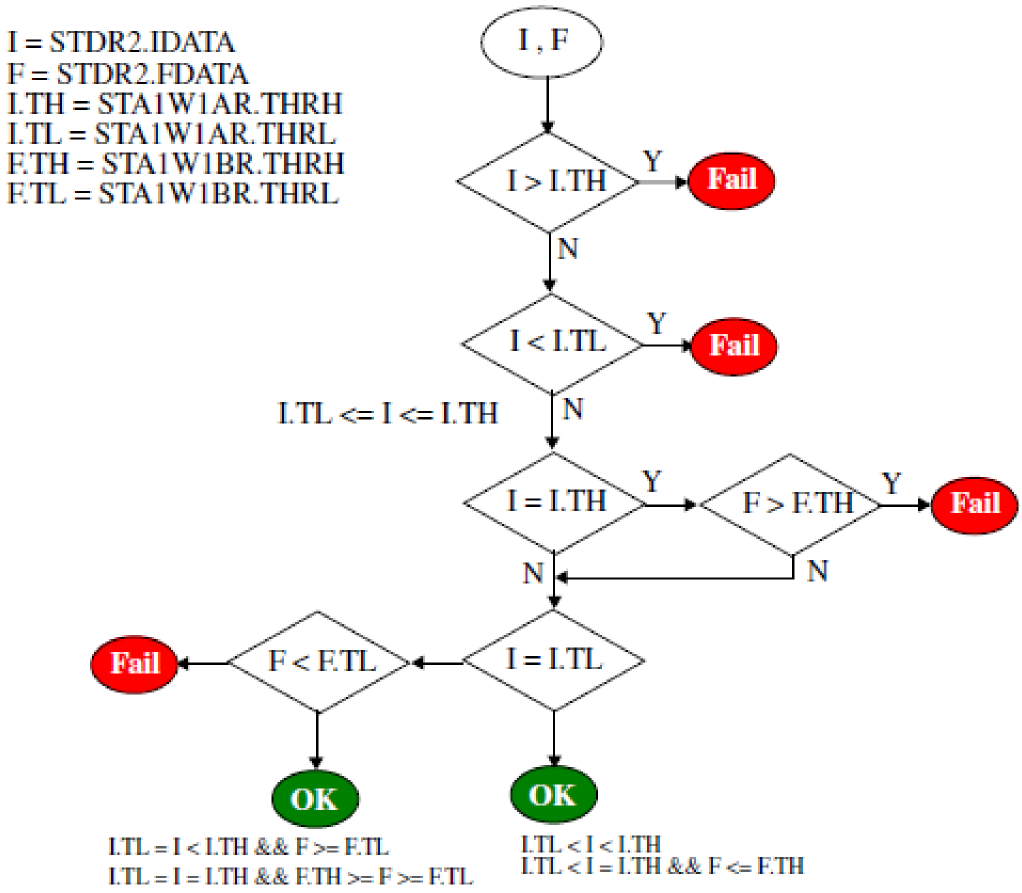

Figure 2. Flow chart showing IDATA and FDATA comparison checking against THRH and THRL values

4.2 S1 Algorithm: Calculated comparison

The user software can convert algorithm S step 1 results and corresponding threshold values to voltages and then compare the voltage results.

The steps involved in this method are as follows:

1. Calculate the high threshold voltage:

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

2. Calculate the low threshold voltage:

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

3. Calculate the S1 voltage:

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

4. Compare results from step 3 to step 1 and 2. Algorithm S step 1 passes if:

$$\text{THRL\_CALC} <= \text{S1\_CALC} <= \text{THRH\_CALC}$$

5 Sample code

The following sample code shows a typical use case for setting both the analog watchdogs which monitor various capacitors (open / short) and power supplies (in range / out of range) and the ADC watchdog timers (ADC conversions finished within watchdog time).

There are two examples functions provided.

- Operating algorithm S in one-shot mode with analog watchdogs enabled.
- Operating algorithm S+C in scan mode with analog watchdogs and watchdog timers enabled.

```
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);
}
```

The users might find it beneficial to add software for storing ADC self-test conversions to SRAM for troubleshooting purposes.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5015
Revision 0, 09/2014