Introduction

In automotive more and more often comes in foreground the question of safety. For this reason the ST’s PowerPC microcontrollers are embedded with various self-test features and procedures. This document provides detailed view on ADC built-in self-tests (BIST’s), along with demonstration examples to guide the customer for correct use of the ADC BIST’s.
Contents

1 ADC built-in self-test ................................................. 5
  1.1 ADC Built-in self-test description ......................... 5
  1.2 ADC self-test important note ............................. 6
  1.3 ADC calibration data .................................. 6
  1.4 Reading from Test Flash .................................. 6
    1.4.1 Flash content .................................. 8

2 ADC built-in self-test algorithms ................................. 10
  2.1 Self-test execution flow .................................. 10
  2.2 Supply self-test – Algorithm S ............................ 11
    2.2.1 Supply self-test in CPU mode ....................... 14
  2.3 Resistive-Capacitive self-test - Algorithm RC ............... 16
    2.3.1 Resistive-Capacitive self-test in CPU mode .......... 16
  2.4 Capacitive self-test - Algorithm C ........................ 18
    2.4.1 Capacitive self-test in CPU mode ................... 18

3 Appendix A ......................................................... 20
  3.1 Test Flash copy function example ......................... 20
  3.2 Algorithm S example in CPU scan mode ................... 20
  3.3 Algorithm RC example in CPU one-shot mode ............... 22
  3.4 Algorithm RC example in CPU scan mode ................... 23
  3.5 Algorithm C example in CPU one-shot mode ............... 24
  3.6 Algorithm C example in CPU scan mode ................... 25

Appendix A Reference document ................................... 27

Revision history .................................................. 28
List of tables

Table 1. Test Flash Content ............................................. 8
Table 2. Typical values stored in Test sector of Flash memory ......................... 8
Table 3. Sampling phase duration for the test conversions .............................. 10
Table 4. Sample phase settings ............................................. 10
Table 5. Document revision history ........................................... 28
List of figures

Figure 1. ADC block diagram ................................................................. 6
Figure 2. Test Flash reading ............................................................... 7
Figure 3. Self-test execution sequence example C and RC in scan mode .... 11
Figure 4. Self-test execution sequence flow example S in scan mode ........ 11
Figure 5. Self-test baud rate example C and RC .................................... 11
Figure 6. ADC self-test Algorithm S ..................................................... 13
Figure 7. ADC self-test configuration – Algorithm S .............................. 14
Figure 8. Example test execution flow for Algorithm S ......................... 15
Figure 9. Test execution flow - Algorithm S ......................................... 16
Figure 10. Self-test flow in one-shot mode- algorithm RC ..................... 17
Figure 11. Test flow - Algorithm RC .................................................... 17
Figure 12. Self-test flow in one-shot mode- algorithm C ....................... 19
1 ADC built-in self-test

For safety relevant applications, it is important to perform the ADC functionality check at regular intervals(a). For this purpose, hardware build in self testing feature has been incorporated inside the ADC. Tests at application level can be used in place of the ADC self-test to verify the integrity of the ADC itself. It is a user's tasks to choose the counter-measure which better fits his needs.

Three types of Self Testing algorithms have been implemented inside ADC analog.

- Supply Self test: Algorithm S
- Resistive-Capacitive Self test: Algorithm RC
- Capacitive Self test: Algorithm C

The built-in self-tests use analog watchdogs to verify the result of self-test conversions. During the production factory testing, accurate threshold values for these watchdogs are stored in the Test sector of the flash memory. Each testing algorithm contains its own threshold values. Each microcontroller has its own threshold values. Most values are identical or very similar across the microcontrollers. The build-in self-tests can be executed in CPU mode as well as in CTU (Cross Triggering Unit) mode(b).

1.1 ADC Built-in self-test description

The ADC implements a channel (self-test channel) dedicated for self-testing. The self-test schedule is done via special set of self-test registers in ADC. ADC also offers capability of monitoring the converted data using analog watchdog registers and flags the error to the Fault Collection and Control Unit (FCCU) in case any of the algorithms fails.

---

(a) depending on the specific safety concept, the ADC self-test can be run only once after the boot
(b) this AN focuses on ADC working in CPU mode
1.2 ADC self-test important note

In case of Injected conversions, test channel conversion is not performed. It is performed only during Normal conversions. If during a test channel conversion, injection conversion arrives, then the test channel is aborted (just as a normal functional channel) and injected conversions are done. After injected conversions are completed, the test channel resumes from the Step at which it was aborted.

For self testing, Mode bit should be programmed (at least one cycle) BEFORE setting MCR.NSTART bit and should not be changed thereafter until conversion is ongoing.

1.3 ADC calibration data

To verify if the measured ADC BIST results are correct, the threshold registers are used. For correct execution of self-test sequence it is necessary to compare if measured values are in range of factory defined thresholds. This is done automatically by the hardware watchdog. In case the ADC BIST results are out of boundaries a fault is triggered to the FCCU.

1.4 Reading from Test Flash

There are some factory settings stored during final test in the Test Flash (mostly calibration information), which are needed by application software. The related data is:

a) Temperature Sensor 1 Calibration Word 1...4
b) Temperature Sensor 2 Calibration Word 1...4
c) ADC 0 Calibration Word 1...8
d) ADC 1 Calibration Word 1...8
Accessing the Test Flash requires to set the SCTR TFE bit in the SSCM (System Status and Configuration Module), which will replace the “normal” Flash memory space with the Test Flash block (while TFE is set).

**Note:** This is only possible one time after a reset of the device and only after destructive, power-on or long external reset!

After other sources of reset, than the three listed above, the test flash enable bit cannot be set.

The Test Flash content is read via function executed out of the RAM as the “normal” Flash content is not accessible while TFE flag is asserted. After reading Test Flash, just before jump back to the “normal” Flash, it is recommended to check if Flash content switch was finished properly. This is done via checking bit TFE = 0 in SCTR register in SSCM module.

Therefore retrieving these values usually requires several steps.

**Figure 2. Test Flash reading**

The diagram shows the process of accessing the Test Flash. The steps include:

1. **Start**: Copy the `read_cal_data()` function from the “normal” Flash into RAM.
2. **Set TFE bit in SCTR register**: This must be executed from RAM.
3. **Verify if TFE is already mapped on “normal” Flash**: The copy function must be executed from RAM as the “normal” Flash content is not accessible when Test Flash is active.
4. **Read Test Flash content**: This function is executed from RAM and reads Test Flash content.
5. **Verify if Test Flash is not anymore mapped on “normal” Flash**: After reading Test Flash, just before jump back to the “normal” Flash, it is recommended to check if Flash content switch was finished properly.
6. **Return to execution from Flash**: The diagram shows the end of the process.
1.4.1 Flash content

Flash memory Test sector contains calibration and other chip-specific data. This information is summarized in Table 1 below.

<table>
<thead>
<tr>
<th>Address</th>
<th>Word name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>TSENS_0_CAL W2/W4</td>
<td>TSENS_0 Calibration data W2 (P2/PTAT) and W4 (C2/CTAT) at cold temperature</td>
</tr>
<tr>
<td>0x0004</td>
<td>TSENS_1_CAL W2/W4</td>
<td>TSENS_1 Calibration data W2 (P2/PTAT) and W4 (C2/CTAT) at cold temperature</td>
</tr>
<tr>
<td>0x0008</td>
<td>TSENS_0_CAL W1/W3</td>
<td>TSENS_0 Calibration data W1 (P1/PTAT) and W3 (C1/CTAT) at hot temperature</td>
</tr>
<tr>
<td>0x000C</td>
<td>TSENS_1_CAL W1/W3</td>
<td>TSENS_1 Calibration data W1 (P1/PTAT) and W3 (C1/CTAT) at hot temperature</td>
</tr>
<tr>
<td>0x0010</td>
<td>ADC0_CAL W1</td>
<td>ADC0 Self-Test calibration - RC algo</td>
</tr>
<tr>
<td>0x0014</td>
<td>ADC0_CAL W2</td>
<td>ADC0 Self-Test calibration - C algo S0 step</td>
</tr>
<tr>
<td>0x0018</td>
<td>ADC0_CAL W3</td>
<td>ADC0 Self-Test calibration - C algo Sn step</td>
</tr>
<tr>
<td>0x001C</td>
<td>ADC0_CAL W4</td>
<td>ADC0 Self-Test calibration - S algo S0 step - 3.3 V</td>
</tr>
<tr>
<td>0x0020</td>
<td>ADC0_CAL W5</td>
<td>ADC0 Self-Test calibration - S algo S0 step - 5.0 V</td>
</tr>
<tr>
<td>0x0024</td>
<td>ADC0_CAL W6</td>
<td>ADC0 Self-Test calibration - S algo S1 step - integer</td>
</tr>
<tr>
<td>0x0028</td>
<td>ADC0_CAL W7</td>
<td>ADC0 Self-Test calibration - S algo S1 step - float</td>
</tr>
<tr>
<td>0x002C</td>
<td>ADC0_CAL W8</td>
<td>ADC0 Self-Test calibration - S algo S2 step</td>
</tr>
<tr>
<td>0x0030</td>
<td>ADC0 Reserved</td>
<td>ADC0 reserved</td>
</tr>
<tr>
<td>0x0034</td>
<td>ADC1_CAL W1</td>
<td>ADC1 Self-Test calibration - RC algo</td>
</tr>
<tr>
<td>0x0038</td>
<td>ADC1_CAL W2</td>
<td>ADC1 Self-Test calibration - C algo S0 step</td>
</tr>
<tr>
<td>0x003C</td>
<td>ADC1_CAL W3</td>
<td>ADC1 Self-Test calibration - C algo Sn step</td>
</tr>
<tr>
<td>0x0040</td>
<td>ADC1_CAL W4</td>
<td>ADC1 Self-Test calibration - S algo S0 step - 3.3 V</td>
</tr>
<tr>
<td>0x0044</td>
<td>ADC1_CAL W5</td>
<td>ADC1 Self-Test calibration - S algo S0 step - 5.0 V</td>
</tr>
<tr>
<td>0x0048</td>
<td>ADC1_CAL W6</td>
<td>ADC1 Self-Test calibration - S algo S1 step - integer</td>
</tr>
<tr>
<td>0x004C</td>
<td>ADC1_CAL W7</td>
<td>ADC1 Self-Test calibration - S algo S1 step - float</td>
</tr>
<tr>
<td>0x0050</td>
<td>ADC1_CAL W8</td>
<td>ADC1 Self-Test calibration - S algo S2 step</td>
</tr>
<tr>
<td>0x0054</td>
<td>ADC1 Reserved</td>
<td>ADC1 reserved</td>
</tr>
</tbody>
</table>

Table 2. Typical values stored in Test sector of Flash memory

<table>
<thead>
<tr>
<th>Word</th>
<th>Flash location offset</th>
<th>Value in flash</th>
<th>Loads to STAWxR</th>
<th>Step</th>
<th>THRH</th>
<th>THRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>10h</td>
<td>0xFE20F1E0</td>
<td>STAW3R</td>
<td>RC</td>
<td>0xE20h</td>
<td>0x1E0h</td>
</tr>
<tr>
<td>W2</td>
<td>14h</td>
<td>0xF856F732</td>
<td>STAW4R</td>
<td>C0</td>
<td>0x856h</td>
<td>0x732h</td>
</tr>
</tbody>
</table>
Table 2. Typical values stored in Test sector of Flash memory (continued)

<table>
<thead>
<tr>
<th>Word</th>
<th>Flash location offset</th>
<th>Value in flash</th>
<th>Loads to STAWxR</th>
<th>Step</th>
<th>THRH</th>
<th>THRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>W3</td>
<td>18h</td>
<td>0xF873F732</td>
<td>STAW5R</td>
<td>C1-C17</td>
<td>0x873h</td>
<td>0x732h</td>
</tr>
<tr>
<td>W4</td>
<td>1Ch</td>
<td>0xF75AF4DF</td>
<td>STAW0R</td>
<td>S0_3.3V</td>
<td>0x75Ah</td>
<td>0x4DFh</td>
</tr>
<tr>
<td>W5</td>
<td>20h</td>
<td>0xF4D0F2DB</td>
<td>STAW0R</td>
<td>S0_5.0V</td>
<td>0x4D0h</td>
<td>0x2DBh</td>
</tr>
<tr>
<td>W6</td>
<td>24h</td>
<td>0xF003F002</td>
<td>STAW1AR</td>
<td>S1(INT)</td>
<td>0x3h</td>
<td>0x2h</td>
</tr>
<tr>
<td>W7</td>
<td>28h</td>
<td>0xF3D9F1E3</td>
<td>STAW1BR</td>
<td>S1(FRAC)</td>
<td>0x3D9h</td>
<td>0x1E3h</td>
</tr>
<tr>
<td>W8</td>
<td>2Ch</td>
<td>0xFFFFFFF9</td>
<td>STAW2R</td>
<td>S2</td>
<td>0xFFFh</td>
<td>0xFF9h</td>
</tr>
</tbody>
</table>

Note: These values are unique for each sample.
2 ADC built-in self-test algorithms

ADC contains three self test algorithms in order to test its functionality. The SPC56xx/RPC56xx uses two types of parameter settings to define the ADC self-test operation:

- Sample phase duration settings programmed into the INPSAMP_S, INPSAMP_RC, and INPSAMP_C fields of the Self-Test Configuration Register (STCR1).
- Threshold values for analog watchdog algorithms

In order to support different loading and switching times, several different conversion timing registers (CTR) are present. There is one register per channel type.

Table 3. Sampling phase duration for the test conversions

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPSAMP_C</td>
<td>Sampling phase duration for the test conversions related to the algorithm C. Valid self-test values for this field are given below. Minimum: 0x18 Maximum: 0xFF</td>
</tr>
<tr>
<td>INPSAMP_RC</td>
<td>Sampling phase duration for the test conversions related to the algorithm RC. Valid self-test values for this field are given below. Minimum: 0x60 Maximum: 0xFF</td>
</tr>
<tr>
<td>INPSAMP_S</td>
<td>Sampling phase duration for the test conversions related to the algorithm S. Valid self-test values for this field are given below. Minimum: 0xFF Maximum: 0xFF</td>
</tr>
</tbody>
</table>

Table 2 represents the typical values for INPSAMP_x register. For detailed description of INPSAMP_x calculation please refer to the Reference Manual.

Table 4. Sample phase settings

<table>
<thead>
<tr>
<th>Register field</th>
<th>Recommended setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPSAMP_C</td>
<td>0x18h</td>
</tr>
<tr>
<td>INPSAMP_RC</td>
<td>0x60h</td>
</tr>
<tr>
<td>INPSAMP_S</td>
<td>0xFFh</td>
</tr>
</tbody>
</table>

Recommended setting for INPSAMP_S is the maximum value of this register field due to slow sample capacitor setting time at low temperature for S0 algorithm.

2.1 Self-test execution flow

The number of channels converted at the end of each chain is 1 (except for Algorithm S, in which all the steps are performed at once without any functional conversion interleaved). In Scan Mode, consecutive steps of selected self test algorithm are converted continuously at the end of each chain of normal conversions.
In the case it is required to schedule self-test channel execution in a defined sequence the baud rate is implemented in ADC. This control defines the scheduling of test channel between the normal conversions. The scheduling rate is specified by STBRR[BR].

By default, if test channel is enabled, one Step of selected algorithm is executed after every chain of normal conversion. STBRR[BR] field provides flexibility by scheduling the test channel conversion to be performed not at the end of every chain but at the end of BR+1 number of chains.

**Note:** This feature is applicable only for scan mode of operation and not for one shot mode. The STBRR.BR should be set to zero for one shot mode.

### 2.2 Supply self-test – Algorithm S

Algorithm S consists of 3 steps, not allowed to be interleaved with normal conversion. At the end of the chain of the normal conversions, (assuming default value of STCR3.ALG) all steps of Algorithm S are performed (as Algorithm S is always atomic).

MSR.SELF_TEST_S is set.
It includes the conversion of:

- ADC internal bandgap voltage (S0 - step0) – (Vbandgap / Vref)
- ADC supply voltage (S1 - step1) – (VDDA / Vbandgap)
- ADC reference voltage (S2 - step2) – (Vref\(^{(c)}\) / Vref)

\(c. \) Vref\(^{(c)}\) is an internal positive reference voltage.
Figure 6. ADC self-test Algorithm S

1. **Vbandgap / Vref**
   - **S0**: Measure Vref
   - **Vref = OK?**
     - NO: Fail
     - YES: **S1**
   - **S1**: Logical checking of integer data and fractional data method
     - **S1 = OK?**
       - NO: Fail
       - YES: **S2**
   - **S2**: ADC reference voltage
     - **S2 = OK?**
       - NO: Fail
       - YES: END

Performing S1 step, used are S0 results. Measure in short possible time to avoid reference voltage drop.
2.2.1 Supply self-test in CPU mode

To properly execute supply self-test it is required to use ADC in scan mode. The scan mode is requested for S algorithm, because S1 (ratio) result is based on S0 result. If these conversions are distant in terms of time, there is a temporal non-correlation that makes the result unreliable. With S algorithm S0 (step) is measured if the ADC internal bandgap voltage ($V_{ref}$) is proper. If this is so, then it is feasible to measure the supply voltage. In case that S1 measurement is delayed from the S0 measurement, then a reference voltage drop may occur. Performing then S1 measurement, where a potentially good S0 result was available does not provide a correct measurement. The problem may be masked here. This is the reason why the ADC one-shot conversion mode is not suitable for algorithm S measurement.

In the ideal environment, results must be the same in both modes one-shot and scan mode because there is no voltage drop in the ADC reference or supply in such measurement condition.

Figure 7. ADC self-test configuration – Algorithm S

In Figure 8 below it is presented the basic self-test flow for algorithm S. The test channel is started automatically after the normal channel conversion as presented in Figure 9.
Figure 8. Example test execution flow for Algorithm S

1. **START**
2. MCR NSTART = 1
   - Start ADC conversion in scan mode
3. MCR NSTART = 0
   - Stop ADC conversion in scan mode
4. **NO**
   - ECH = 1?
     - Wait for conversion to finish
5. **YES**
   - ISR ECH = 1
     - Clear End of Channel interrupt flag
6. **NO**
   - ST_EOC = 1?
     - Wait for Self Test End of Channel flag
7. **YES**
   - ISR ST_EOC = 1
     - Clear Self Test End of conversion flag
8. **Test passed?**
   - Check if No S-algorithm error has occurred
     - STSR1 ERR_S0 =1 ?
     - STSR1 ERR_S1 =1 ?
     - STSR1 ERR_S2 =1 ?
9. **END**

Test Failed!
2.3 Resistive-Capacitive self-test - Algorithm RC

Algorithm RC includes a sequence of 19 test conversions (steps) by setting the ADC internal resistive digital-to-analog converter (DAC).

2.3.1 Resistive-Capacitive self-test in CPU mode

Resistive-Capacitive self-test can be executed either in one shot or scan mode. In Figure 11 a basic test flow is demonstrated. After the execution of normal channel the single step of self-test is executed. This process continues for all Steps of RC algorithm. State Machine returns to IDLE state when MCR[NSTART] is cleared.

The demonstration example of RC algorithm in one-shot mode is shown in Figure 10. It is necessary to check self-test result after every step of the conversion.
Figure 10. Self-test flow in one-shot mode - algorithm RC

START

ADC configuration
MCR PWDN = 0
MCR CTUEN = 0
NCMR[0] CH0 = 1
MCR MODE = 0
STCR1 INPSAMP_RC = 0xFF
STCR3 ALG = 0x1
STCR2 EN = 1
STAW3R AWDE = 1

Configure ADC for RC algorithm in one shot mode

STAWxR

Step = 0

Start RC algorithm from step 0

Repeat measurement for all 19 steps

Step = 18?

NO

Stcr3 MSTEP = step

Load actual step into MSTEP register

YES

END

NSWART = 1

Start conversion

Failure check

STSR1

STEP_RC = 0 ?

YES

Step = Step + 1

Increment step

NO

FAIL

Figure 11. Test flow - Algorithm RC

Algorithm RC

NSWART = 1

Normal Channel

Test Channel

Step "n"

ST_EOC = 1 ?

YES

NSWART = 0

NO

for (n=0, n<19; n++)
2.4 Capacitive self-test - Algorithm C

Capacitive Self-test Algorithm C contains a sequence of 17 test conversions (steps) by setting the capacitive elements including the sampling capacitor/ capacitive DAC. This algorithm is designed to highlight faults in the sampling capacitor matrix. Each step considers two elements of the sampling capacitor matrix. For each step a sequence of pre-sampling, sampling and evaluation phases are executed.

2.4.1 Capacitive self-test in CPU mode

*Figure 12* demonstrates a basic self-test flow for algorithm C in one-shot mode. After executing each step of the self-test the error bit STSR1[STEP_C] is checked.
Figure 12. Self-test flow in one-shot mode- algorithm C

Start

ADC configuration

MCR.PWDN = 0
MCR.CTUEN = 0
NCMR[2].CH0 = 1
MCR.MODE = 0
STCR1.INPSAMP_C = 0xFF
STCR3.ALG = 0x2
STCR2.EN = 1
STAW3R.AWDE = 1

Configure ADC for C algorithm in one shot mode

STAWxR

Step = 0

Start RC algorithm from step 0

Repeat measurement for all 17 steps

Step = 16?

NO

STCR3.MSTEP = step

Load actual step into MSTEP register

YES

END

STSR1_STEP_C = 0 ?

NO

FAIL

YES

NSTART = 1

Start conversion

Step = Step + 1

Increment step
3 Appendix A

Example source code can be found in these appendixes.

3.1 Test Flash copy function example

Read_Self_Test_calibration_data function is necessary to be executed from RAM.

```c
#include <stdio.h>

#define RAM_BASE_ADDR 0x1000

void Read_Self_Test_calibration_data (void)
{
    uint32_t address = 0x0;
    uint32_t i = 0;

    /* map Test Flash sector into Flash memory stating 0x0 */
    SSCM.SCTR.B.TFE = 1;

    for (i=0;i<22;i++)
    {
        Raw_Test_Flash_Calibration_Data[i] = (*(volatile uint32_t *)address);
        address = address + 0x4;
    }
    SSCM.SCTR.B.TFE = 0;
}
```

3.2 Algorithm S example in CPU scan mode

```c
#include <stdio.h>

#define RAM_BASE_ADDR 0x1000

void ADC0_SUPPLY_SELF_TEST_ScanMode(void)
{
    // Algorithm S
    ADC0.MCR.B.PWDN = 0;      // Enable ADC0
    /* 1 */
    ADC0.MCR.B.CTUEN = 0;     // CPU mode
    /* 2 */
    ADC0.NCMR[0].B.CH0 = 1;   // ADC0 channel 0 is set. Program NCMR0 to select channels to be converted for normal conversion
    /* 3 */
```
ADC0.MCR.B.MODE = 1;  // Program MCR[MODE] = 1 to select Scan Mode

/* 4 */

ADC0.STCR1.B.INPSAMP_S = 0xFF;  // Program sampling phase duration for
the test conversions
/* 5 */

ADC0.STCR3.B ALG = 0x0;  // Select the Self Testing algorithm in
STCR3.ALG. Default is Algorithm S
/* 6 */

ADC0.STCR2.B.EN = 1;  // ADC0 Enable self testing channel by setting
EN bit in the STCR2 register
/* 7 */

/* Step 0 - ADC0 internal bandgap voltage register configuration */
ADC0.STCR3.B.MSTEP = 0;  // For Scan-shot mode, defines the first
step for algorithms S
ADC0.STAW0R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[6];  //
Set ADC0 Self-Test calibration data - S algo S0 step - 3.3V
ADC0.STAW0R.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[7];  //
Set ADC0 Self-Test calibration data - S algo S0 step - 3.3V
ADC0.STAW0R.B.AWDE = 1;  // Enable analog watchdog related to the
algorithm S (step 0)
/* Step 1 - ADC supply voltage register configuration */
/* Integer */
ADC0.STAW1AR.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[10];  //
Set ADC0 Self-Test calibration data - S algo S1 step - (integer) - Low
Treshold Value
ADC0.STAW1AR.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[11];  //
Set ADC0 Self-Test calibration data - S algo S1 step - (integer) - High
Treshold Value
/* Float */
ADC0.STAW1BR.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[12];  //
Set ADC0 Self-Test calibration data - S algo S1 step - (float) - Low
Treshold Value
ADC0.STAW1BR.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[13];
ADC0.STAW1BR.B.AWDE = 1;  // Enable analog watchdog related to the
algorithm S (step 1)
/* Step 2 - ADC reference voltage register configuration */
ADC0.STAW2R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[14];  //
Set ADC0 Self-Test calibration data - S algo S2 step) - Low Treshold Value
ADC0.STAW2R.B.AWDE = 1;  // Enable analog watchdog related to the
algorithm S (step 2)
ADC0.MCR.B.NSTART = 1;  // Start ADC0 normal conversion by setting
NSTART bit in the MCR
ADC0.MCR.B.NSTART = 0;  // Stop ADC0 normal conversion by setting
NSTART bit in the MCR
while (ADC0.ISR.B.ECH == 0);
ADC0.ISR.B.ECH = 1;
while (ADC0.STSR1.B.ST_EOC == 0);
ADC0.STSR1.B.ST_EOC = 1;

if (ADC0.STSR1.B.ERR_S0 == 1 || ADC0.STSR1.B.ERR_S1 == 1 ||
ADC0.STSR1.B.ERR_S2 == 1)    // Check if No S-algorithm error has occurred
{
    while(1);  // Self test failed
}

/* 8 */
// On receiving end of conversion for test channel, the digital result is
// written in
// STDR1.TCDATA and STDR1.VALID bit is set. Also, EOC and ECH bits are
// set in the
// ISR and ST_EOC bit is set in STSR1.
/* 9 */
// State Machine returns to IDLE state.

3.3 Algorithm RC example in CPU one-shot mode

void ADC0_RESISTIVE_CAPACITIVE_SELF_TEST (void)
{
    uint8_t step = 0;

    // Algorithm RC
    ADC0.MCR.B.PWDN = 0; // Enable ADC0
    /* 1 */
    ADC0.MCR.B.CTUEN = 0; // CPU mode
    /* 2 */
    ADC0.NCMR[0].B.CH0 = 1; // ADC0 channel 0 is set. Program NCMR0 to
    // select channels to be converted for normal conversion
    /* 3 */
    ADC0.MCR.B.MODE = 0; // Program MCR[MODE] = 0 to select one shot
    // mode
    /* 4 */
    ADC0.STCR1.B.INPSAMP_RC = 0xFF; // Program sampling phase duration for
    // the test conversions
    /* 5 */
    ADC0.STCR3.B.ALG = 0x1; // Select the RC Self Testing algorithm in
    STCR3.ALG. Default is Algorithm S
    /* 6 */
    ADC0.STCR2.B.EN = 1; // ADC0 Enable self testing channel by setting
    // EN bit in the STCR2 register
    /* 7 */
    ADC0.STAW3R.B.AWDE = 1; // Enable analog watchdog related to the
    // algorithm RC
    /* 8 */
ADC0.STAW3R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[0]; // ADC0 Self-Test calibration - RC algo - Low Treshold Value
ADC0.STAW3R.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[1]; // ADC0 Self-Test calibration - RC algo - High Treshold Value
/* 9 */

for(step=0;step<19;step++)
{
  ADC0.STCR3.B.MSTEP = step;   // For one-shot mode, defines the current step for algorithms S
  ADC0.MCR.B.NSTART = 1;    // Start ADC0 normal conversion by setting NSTART bit in the MCR
  if (ADC1.STSR1.B.STEP_RC != 0)
  {
    ADC_Self_Test_Fail[step+6] = 1;   // Check if No RC-algorithm [step] error has occurred
  }
}
}

3.4 Algorithm RC example in CPU scan mode

void ADC0_RESISTIVE_CAPACITIVE_SELF_TEST_ScanMode (void)
{
  // Algorithm RC
  ADC0.MCR.B.PWDN = 0;      // Enable ADC0
  /* 1 */
  ADC0.MCR.B.CTUEN = 0;     // CPU mode
  /* 2 */
  ADC0.NCMR[0].B.CH0 = 1;   // ADC0 channel 0 is set. Program NCMR0 to select channels to be converted for normal conversion
  /* 3 */
  ADC0.MCR.B.MODE = 1;      // Program MCR[MODE] = 0 to select one scan mode
  /* 4 */
  ADC0.STCR1.B.INPSAMP_RC = 0xFF;  // Program sampling phase duration for the test conversions
  /* 5 */
  ADC0.STCR3.B.ALG = 0x1;   // Select the RC Self Testing algorithm in STCR3.ALG. Default is Algorithm S
  /* 6 */
  ADC0.STCR2.B.EN = 1;      // ADC0 Enable self testing channel by setting EN bit in the STCR2 register
  /* 7 */
  ADC0.STAW3R.B.awde = 1;   // Enable analog watchdog related to the algorithm RC
  /* 8 */
ADC0.STAW3R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[0]; // ADC0 Self-Test calibration – RC algo – Low Treshold Value
ADC0.STAW3R.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[1]; // ADC0 Self-Test calibration – RC algo – High Treshold Value
/* 9 */
ADC0.MCR.B.NSTART = 1; // Start ADC0 normal conversion by setting NSTART bit in the MCR
while (ADC0.STSR1.B.ST_EOC != 0);

ADC0.MCR.B.NSTART = 0; // Stop ADC0 normal conversion by setting NSTART bit in the MCR

if (ADC0.STSR1.B.STEP_RC != 0) // Check if No RC-algorithm error has occurred
{
    while(1); // Self test failed
}
ADC0.STSR1.B.ST_EOC = 0;

3.5 Algorithm C example in CPU one-shot mode

void ADC1_CAPACITIVE_SELF_TEST (void)
{
    uint8_t step = 0;

    // Algorithm C
    ADC1.MCR.B.PWDN = 0; // Enable ADC1
    /* 1 */
    ADC1.MCR.B.CTUEN = 0; // CPU mode
    /* 2 */
    ADC1.NCMR[0].B.CH0 = 1; // ADC_0 channel 0 is set. Program NCMR0 to select channels to be converted for normal conversion
    /* 3 */
    ADC1.MCR.B.MODE = 0; // Program MCR[MODE] = 0 to select one shot mode
    /* 4 */
    ADC1.STCR1.B.INPSAMP_C = 0xFF; // Program sampling phase duration for the test conversions
    /* 5 */
    ADC1.STCR3.B.ALG = 0x2; // Select the Self Testing algorithm in STCR3.ALG. Default is Algorithm S
    /* 6 */
    ADC1.STCR2.B.EN = 1; // ADC_0 Enable self testing channel by setting EN bit in the STCR2 register
    /* 7 */
ADC1.MCR.B.NSTART = 1;  // Start ADC1 normal conversion by setting NSTART bit in the MCR
/* 8 */
ADC1.STAW4R.B.THRL = ADC1_Self_Test_Flash_Calibration_Data[2];  // ADC1 Self-Test calibration - C algo S0 step) - Low Treshold Value
ADC1.STAW4R.B.THRH = ADC1_Self_Test_Flash_Calibration_Data[3];  // ADC1 Self-Test calibration - C algo S0 step) - High Treshold Value
ADC1.STAW5R.B.THRL = ADC1_Self_Test_Flash_Calibration_Data[4];  // ADC1 Self-Test calibration - C algo Sn step) - Low Treshold Value
ADC1.STAW5R.B.THRH = ADC1_Self_Test_Flash_Calibration_Data[5];  // ADC1 Self-Test calibration - C algo Sn step) - High Treshold Value
for(step=0;step<17;step++)
{
    ADC1.STCR3.B.MSTEP = step;  // For one-shot mode, defines the current step for algorithms S
    ADC1.MCR.B.NSTART = 1;  // Start ADC1 normal conversion by setting NSTART bit in the MCR
    if (ADC1.STSR1.B.STEP_C != 0)
    {
        ADC_Self_Test_Fail[step+60] = 1;  // Check if No RC-algorithm [step] error has occurred
    }
}

3.6 Algorithm C example in CPU scan mode

void ADC0_CAPACITIVE_SELF_TEST_ScanMod (void)
{
    // Algorithm C
    ADC0.MCR.B.PWDN = 0;  // Enable ADC0
    /* 1 */
    ADC0.MCR.B.CTUEN = 0;  // CPU mode
    /* 2 */
    ADC0.NCMR[0].B.CH0 = 1;  // ADC_0 channel 0 is set. Program NCMR0 to select channels to be converted for normal conversion
    /* 3 */
    ADC0.MCR.B.MODE = 1;  // Program MCR[MODE] = 0 to select one shot mode
    /* 4 */
    ADC0.STCR1.B.INPSAMP_C = 0xFF;  // Program sampling phase duration for the test conversions
    /* 5 */
    ADC0.STCR3.B.ALG = 0x2;  // Select the Self Testing algorithm in STCR3.ALG. Default is Algorithm S
    /* 6 */
    ADC0.STCR2.B.EN = 1;  // ADC_0 Enable self testing channel by setting EN bit in the STCR2 register
ADC0.STAW4R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[2]; // ADC0 Self-Test calibration - C algo S0 step) - Low Threshold Value
ADC0.STAW4R.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[3]; // ADC0 Self-Test calibration - C algo S0 step) - High Threshold Value
ADC0.STAW5R.B.THRL = ADC0_Self_Test_Flash_Calibration_Data[4]; // ADC0 Self-Test calibration - C algo Sn step) - Low Threshold Value
ADC0.STAW5R.B.THRH = ADC0_Self_Test_Flash_Calibration_Data[5]; // ADC0 Self-Test calibration - C algo Sn step) - High Threshold Value

ADC0.MCR.B.NSTART = 1;    // Start ADC0 normal conversion by setting
NSTART bit in the MCR
while (ADC0.STSR1.B.ST_EOC != 0);
ADC0.MCR.B.NSTART = 0;    // Stop ADC0 normal conversion by setting
NSTART bit in the MCR
if (ADC0.STSR1.B.STEP_C != 0)   // Check if No C-algorithm error has
occurred
{
    while(1); // Self test failed
}
ADC0.STSR1.B.ST_EOC = 0;
Appendix A  Reference document

- *SPC56ELxx Reference manual (Doc ID 15265)*
# Revision history

## Table 5. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-Oct-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>08-Oct-2015</td>
<td>2</td>
<td>Robust root part numbers added.</td>
</tr>
</tbody>
</table>