Freescale Semiconductor
Application Note

AN2989
Rev. 2, 02/2010

Design, Accuracy, and Calibration of Analog to Digital Converters on the MPC5500 Family

by: Mike Garrard, TSPG Powertrain Systems Engg.
Pauline Ryan, TSPG MCD Applications Engg.

1 Introduction

A new architecture of analog to digital converter (ADC) is introduced with the MPC5554 device to meet requirements of increased speed and accuracy. This architecture consists of a converter engine with improved linearity and correction hardware to remove linear gain and offset errors. This correction hardware is the ADC calibration subsystem.

This application note outlines the advantages of the new redundant signed digit (RSD) architecture of the ADC introduced with the MPC5554, compared to the successive approximation architecture used in the MPC500 family. It discusses the gain and offset errors introduced by the RSD ADC and how the hardware multiply accumulate (MAC) unit is used to remove them. The ADC design features used to recover full range are explained along with design features in the MAC required to avoid lost codes, and the half-count offset adder that increases accuracy. Levels of calibration and their resulting accuracy are described. The parameters affecting total error are also discussed.

Table of Contents

Section 3, “Applying Calibration to the ADC” describes how to accomplish calibration. A procedure for power up calibration is provided as well as a procedure for run time calibration, including when to schedule it. A test is given to evaluate a system to tune the calibration. The relevant ADC internal registers are described. Calculation of the coefficients is explained with both integer and floating point examples. Software to achieve the calibration function is described, and provided in Appendix A.

1.1 Objective

The objective of this application note is two-fold. Section 2, “ADC and MAC Design Features” aims to educate on the structure and design features of the MPC5500 family ADC that make it suitable for embedded control. Section 3, “Applying Calibration to the ADC” is a tutorial on using the calibration feature to obtain the best performance from the ADC.

2 ADC and MAC Design Features

2.1 ADC Design Requirements

The MPC5500 family of devices was developed with a faster ADC and improved linearity and monotonicity. A low capacitance input was required. Maintaining full 0 V to 5 V and DC capability with architecture capable of further development. A new RSD architecture of ADC was introduced to replace the successive approximation (SA) architecture used in the previous MPC500 family. Compared to the MPC500 family ADC, the ADC on the MPC5500 family typically offers:

- 4x the resolution
- 3x the linearity
- 4x the conversion speed
- At least the same absolute error
- 0.8 pF input capacitance

2.2 ADC Design Solutions

2.2.1 MPC500 Successive Approximation ADC

The MPC500 ADC is a successive approximation design. The core operation of this design is straightforward, as shown in Figure 1. The input voltage is converted one bit at a time starting with the most significant bit. The DAC (digital to analog converter) is set to 50% of the voltage range and the comparator reports whether the input voltage is higher or lower than this. The most significant bit (MSB) is set to 1 or 0 as appropriate. For example, say the voltage is higher than the midpoint, therefore MSB = 1. The second most significant bit is now determined: the DAC is set to a 75% range and 50% for MSB = 1 plus 25% as the midpoint for the second most significant bit. The comparator reports whether the input voltage is higher or lower than this and bit 2 is set as appropriate. The conversion continues in this manner until all bits are determined. The input voltage remains connected during the conversion. The MPC500

uses a buffer to charge this up initially. This ADC architecture runs at moderate speeds and is economical to implement, but relies on the DAC and a good amplifier offset for accuracy.

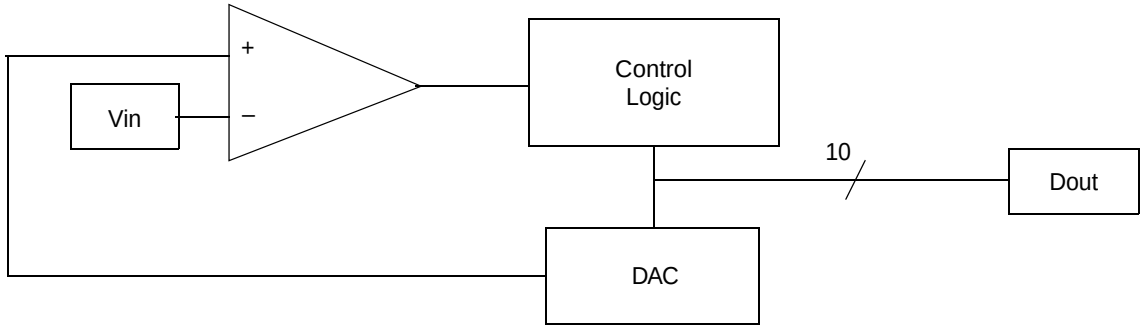


Figure 1. MPC500 Architecture Successive Approximation ADC

2.2.2 MPC5500 Redundant Signed Digit ADC

To meet the requirements for the MPC5500 family, the ADC is changed to the RSD architecture. This also uses successive approximation. Instead of a DAC an accurate x2 amplifier is used. The SA ADC makes comparisons at progressively smaller voltage differentials for each bit, whereas the RSD doubles the voltage each time and compares at the same level. The table below outlines a simplified structure of such a converter. This is sufficient to explain the significant features of this type of ADC. The actual design implemented on the MPC5500 uses two reference points a 25% and 75% with a three-option summation and a two-bit digital answer per phase combined in the digital logic to provide the result. These enhancements provide greater noise immunity. For a description refer to the MPC5553/54 microcontroller reference manual.

In the RSD architecture (Figure 2), the input voltage is connected only for the initial sample via S1. This is compared to the 50% reference voltage to provide the MSB. If the current bit is 1, then the reference voltage is subtracted using S2; otherwise no subtraction occurs. For subsequent bits, the residue of the sum is multiplied by two using S3, then compared to the reference voltage again using S4. Assume Vin is 65%. It is greater than 50%, therefore MSB = 1 and 50% is subtracted leaving a residue of 15%. This is multiplied by two, the next compare is then 30%. This is less than mid voltage (50%), therefore the next bit is zero. Nothing is subtracted, therefore doubled to 60%. The next bit is 1, and so on. The table below shows an example for a 6-bit conversion of 65%. 65% of 6 bits = 0.65*64 = 41 = 0b101001. The input voltage is connected only for sampling (MPC5500 default two clocks) and as the capacitance on the MPC5500 is kept small, no buffer is required.

Residue	Compare	Adder
Sample 65%	>50% = 1	65% – 50%=15%
2 x 15% = 30%	<50% = 0	30% – 0 = 30%
2 x 30% = 60%	>50% = 1	60% – 50% = 10%
2 x 10% = 20%	<50% = 0	20% – 0 = 20%
2 x 20% = 40%	<50% = 0	40% – 0 = 40%
2 x 40% = 80%	>50% = 1	

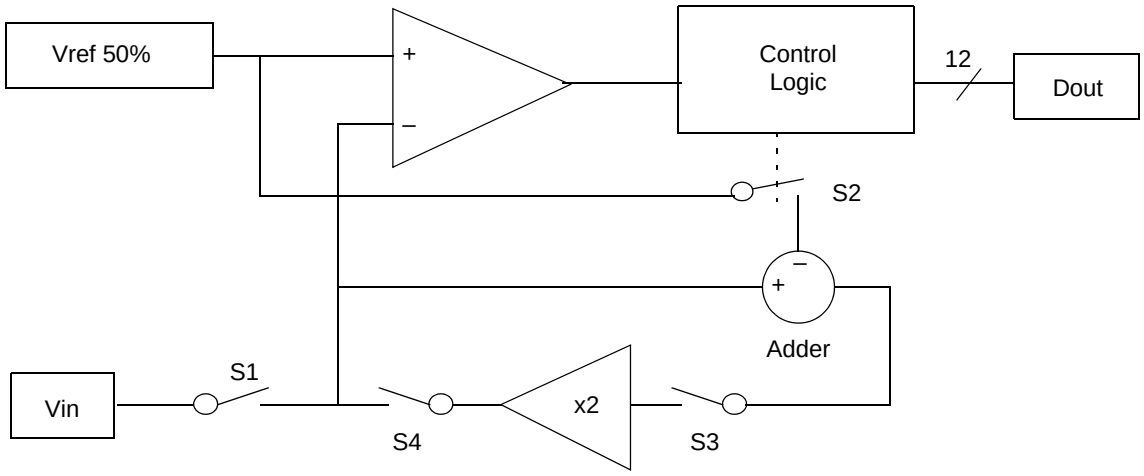


Figure 2. MPC5500 Architecture RSD ADC Engine

The precision elements of the RSD architecture are the x2 amplifier and adder. The advantage is, fewer accurately matched components are required. These components are physically larger and matching is better. The design offers great linearity because the same components are used for each bit. Because the comparisons are made on scaled up voltages via the amplifier, the design is less sensitive to amplifiers settling and can be driven faster.

The first source of linear error for this design is, capacitor matching. This is because the x2 amplifier is based on capacitors. Any error away from the ideal gain of two is doubled each time another bit is computed. For example, the sample voltage might go through 1.99 x 1.99 x 1.99 x … This results in an overall gain error for the ADC transfer function. To obtain the performance achieved by the MPC5500, the capacitors must be matched to about 0.0003%. The second source of linear error for this design is amplifier input offset. Again, this is doubled each time another bit is computed. To obtain the performance specification of the MPC5500, the input offset must be less than 18 nV. Special techniques are used to achieve these impressive figures.

2.2.3 Multiply Accumulate Unit

The digital output of the RSD stage passes to the second stage of the ADC subsystem which is the Multiply Accumulate (MAC) unit. The MAC unit is invoked whenever a conversion command message with calibration enabled (CAL = 1) is received by an ADC. To calibrate a conversion, the MAC unit requires two prior pieces of information; the offset calibration coefficient (OCC) and the gain calibration coefficient (GCC). The offset of an ideal ADC is 0. The gain (slope) of an ideal ADC is 1. The purpose of this unit is to remove the gain and offset errors discussed at the end of Section 2.2.2, "MPC5500 Redundant Signed Digit ADC", using a simple y = mx + c linear correction:

Calibrated result = Gain * (Uncalibrated result) + Offset ***Eqn. 1***

This correction is shown graphically in Figure 3.

NOTE

The ADC is engineered by design to have a slope less than one and a positive offset (of codes). This means that the full 0 V to 5 V range is always available. Although this change was achieved through adjustment to the reference taps used in the RSD algorithm, the reference channels read for calibration reference remain at precisely 25% and 75%. However, the 50% channel is the analog ground and does show the 20 mV offset. The expected code after calibration is not 2048 but 2032.

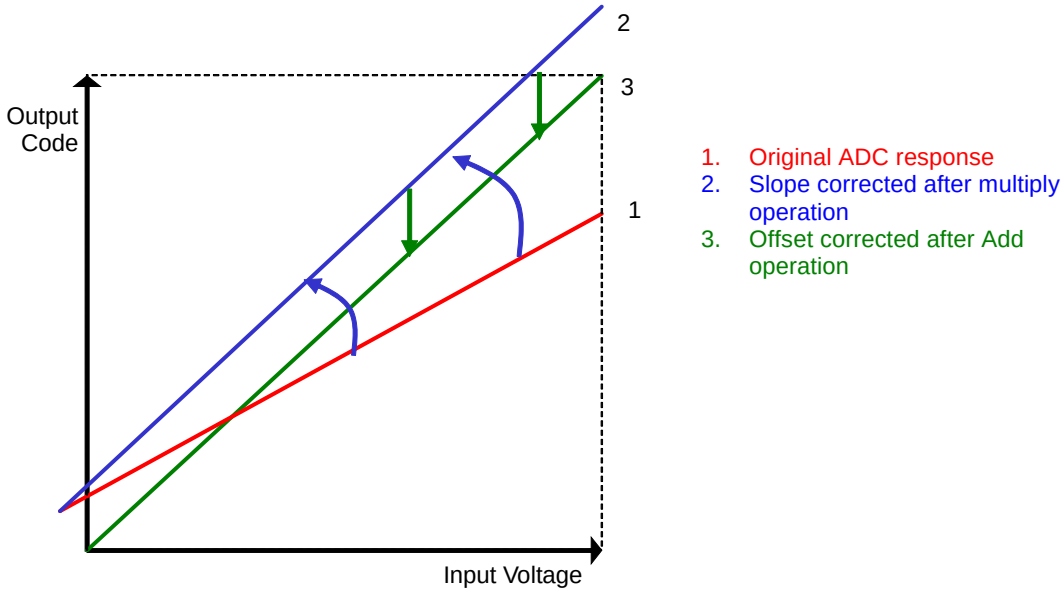


Figure 3. MAC Correction to ADC Response

A block diagram of the MAC is shown in Figure 4.

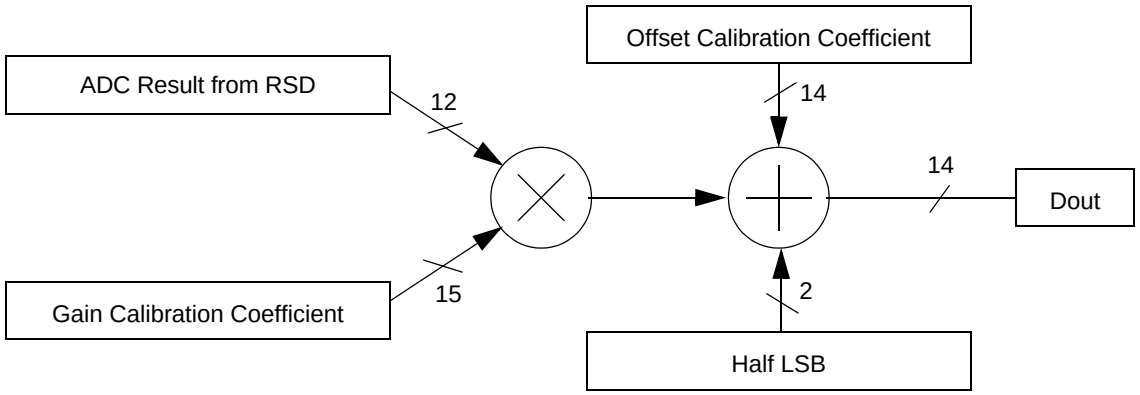


Figure 4. MPC5500 MAC Architecture

It can be seen that although the ADC provides a 12-bit result, the MAC uses fifteen bits for the gain coefficient and fourteen bits for the offset coefficient. This is to avoid lost codes or stuck codes. For example, the ADC in Figure 3 has 4096 output codes. This staircase is then stretched by the gain correction and pushed downwards by the offset correction. In doing this, a few codes (steps on the staircase) fall outside the input voltage range either below 0 V or above 5 V. Therefore, the ADC between 0 V and 5 V now delivers about 4080 codes. However, the ideal response has 4096 available steps that must be dealt with. The only way for the MAC to do this is to take one extra step in sixteen places. This is undesirable because in these places an input voltage change of one least significant bit (LSB) to the ADC results in an output code change of two LSB. The ADC looses codes.

Because the MAC uses 14-bit arithmetic, it is able to take 0.25 LSB steps (0.3 mV). Therefore, the 4080 codes can be distributed evenly over the 0 V to 5 V range. Each time the ADC delivers a result, the MAC is able to select the most appropriate of the 16384 numbers available. The difference between a 12-bit result and a 14-bit result is shown in Figure 5. Blue is the input analog voltage, green is the response of a low-gain ADC, and red is the output of the MAC. The lost codes can be seen as jumps in the red staircase.

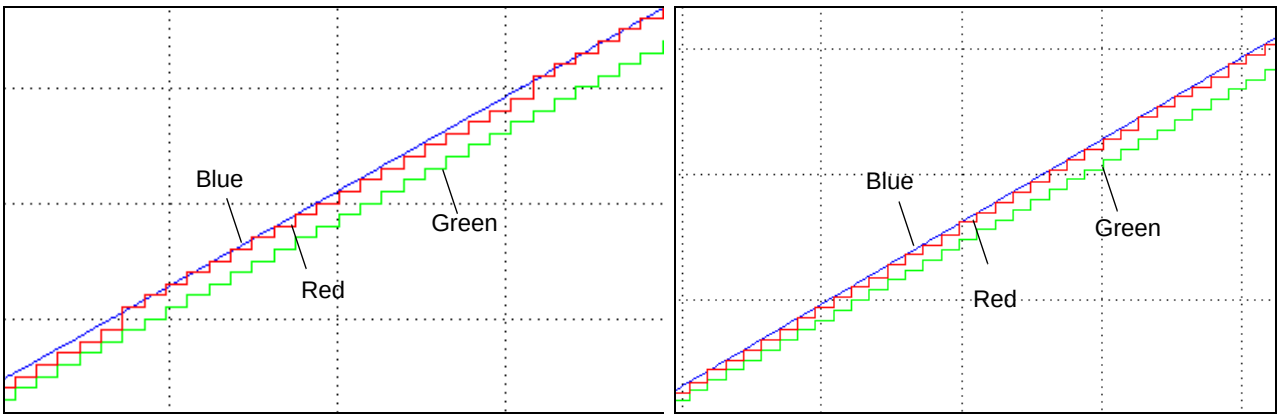


Figure 5. Comparison of 12-bit MAC Result (left) vs. 14-bit MAC Result (right)

One final design feature is worthy of mention; the half LSB compensation. This reduces the error by centering the quantized ADC response over the linear analog input. The effect can be seen in Figure 6. Half LSB on the 12-bit ADC result is an adder of two in the 14-bit MAC calculations.

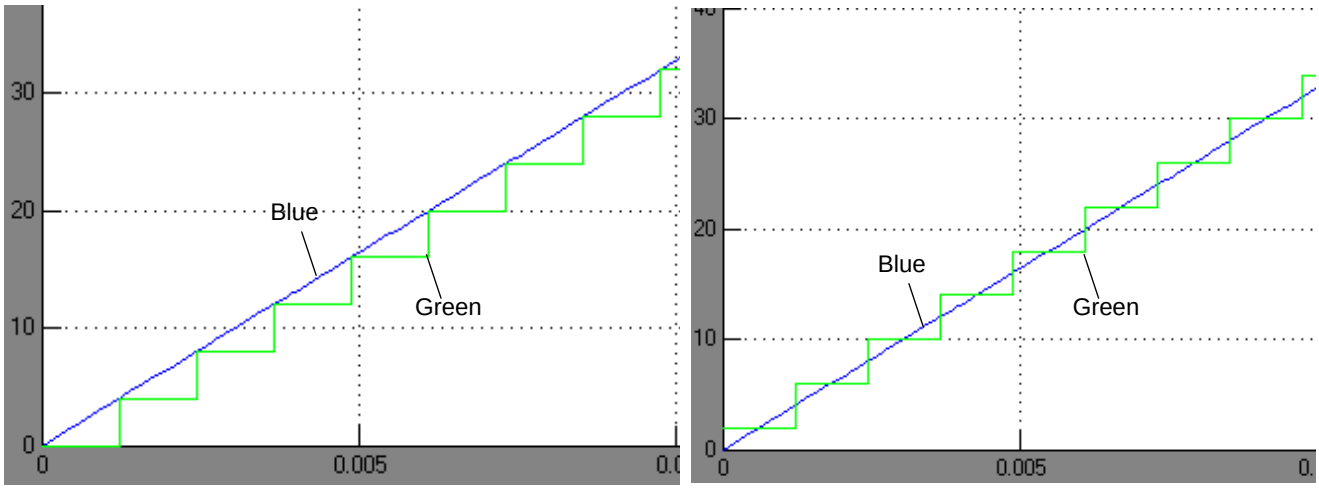


Figure 6. MAC Output Without Half LSB Adder (left) and With (right)

When a conversion command message is received and does not request calibration (CAL = 0), the MAC continues to add the half LSB correction of two and continues to multiply by four. This provides consistency of 14-bit result format for both calibrated and uncalibrated conversions.

2.3 ADC Accuracy

2.3.1 Levels of Calibration

Calibration compensates for the errors below (listed in order of decreasing magnitude).

1. Reference offset to permit full 0 V to 5 V range
2. Silicon manufacturing variation
3. Temperature drift

Three levels of calibration are possible.

2.3.1.1 No Calibration

This results in an absolute error (TUE – total unadjusted error) of around six to seven bits of accuracy. It is dominated by the uncalibrated gain and uncalibrated offset terms in the data sheet. Although most devices are well within the data sheet limits. It must be noted that they are not centred around zero, to ensure full 0 V to 5 V range. Therefore, an uncalibrated ADC always displays a gain and offset error. It is expected that systems that require performance from the MPC5500 always requires calibration.

2.3.1.2 Power Up Calibration

A single calibration on power up removes the most significant errors; those engineered in for full voltage range capability. The ADC measures about eight to nine bits accurately. This suits the systems requiring MPC500 levels of accuracy. One advantage of power up calibration is that it can occur before the control module begins to run large noisy loads such as ignition coils and transmission solenoids. The power supplies and references of the ADC are then less noisy and a more precise calibration is likely to be obtained.

2.3.1.3 Run Time Calibration

The highest absolute accuracy is obtained by run time calibration this is because it removes temperature drift. The ADC measures nine to ten bits TUE, if the remainder of the ECU system is capable of that. It is also the most complex to design and implement. The procedure for power up and run time calibration is provided in Section 3, “Applying Calibration to the ADC”.

2.3.2 Level of Calibration Required

This depends upon the accuracy required by the system, the accuracy available to the ADC, and the types of sensors being measured.

2.3.2.1 Signals Dependent Upon ADC Absolute Accuracy TUE

The ADC is capable of nine to ten bits TUE. To measure, this requires highly accurate and linear test equipment (preferably analog), stable low noise references, a quiet circuit board without peripheral activities, and a test environment without external interference. To make use of this level of accuracy in a production environment, further processing of the signal read by the control module is necessary. This is due to the noise levels present on the signal itself and inside the engine control unit (ECU), particularly the ADC references. For example, if the ADC references have 3 mV of noise and the signal read has 10 mV of noise, then a perfect ADC provides only eight bits of accuracy. In this case, multiple reads and a form of averaging could remove the noise, but there are other error sources where compensation is nearly impossible. For example, the tracking supply to a ratiometric sensor has 20 mV regulation for line and load. To achieve better than eight bits requires reading the supply directly instead of relying on the tracking function. If the sensor plus filter is high impedance (above 100 kΩ) a low offset buffer IS required to avoid a voltage drop that reduces accuracy to nine bits. For an ideal ADC to read nine bits of accuracy, if a sensor requires a voltage divider then these two resistors must be better than 0.1% over temperature and age. Finally, there are error sources where compensation is not possible. For example, the absolute reference voltage supplied to the ADC in the control module might vary by 15 mV over manufacture, temperature, and aging.

NOTE

Although the Freescale data sheet provides results from single ended conversions, the ADC is differential in nature and can be used in this mode. The ultimate ADC accuracy is achieved using differential sensors read differentially.

2.3.2.2 Signals Dependent Upon ADC Linearity INL

There are signals where the ADC is able to deliver ten or even eleven bits of accuracy from closed loop or a.c. sensors. For these no calibration is required at all. The only data sheet specifications that are significant are integral non linearity (INL) this gives linearity and differential non linearity (DNL) this indicates monotonicity. These specifications must not be added for total error. Three examples are given below.

- Knock sensor — This is an a.c. sensor here the important information is the energy contained in a frequency band. On both counts the d.c. component is removed. ADC offset is irrelevant. At the first it might appear the ADC gain is significant because it affects the energy read. However, engines vary in noise initially and over time knock sensors vary in output amplitude. The amplitude a knock sensor hears each cylinder varies due to distance and mounting effectiveness. The knock phenomenon is combustion related and has considerable variation. Each of these variations are many times greater than the fraction of a percent variation in ADC gain reducing the need for calibration. As a result of this variation most knock algorithms include a means to dynamically adjust the threshold where knock is declared therefore negating any gain calibration. The ADC when used with a knock sensor operates at the accuracy of the INL specification at about eleven bits of accuracy.
- Throttle potentiometer — This is a d.c. ratiometric sensor this means it provides a voltage that is a ratio of its supply voltage. It is a critical sensor particularly at low throttle openings, but there is a mechanical error as fitted. To increase precision, the interface algorithm usually determines the closed voltage. This occurs at idle and no lower voltage is normally seen. The fully open position is also determined as the maximum normal voltage and when the engine manifold reads maximum pressure. The significant accuracy for the ADC is linearity and for identifying changes in position and monotonicity.
- Airbag accelerometer — A common sensor type for airbag systems is a micro-electromechanical system (MEMS) capacitor. There is a significant d.c. (zero g) tolerance as well as significant gain (mV/g) tolerance making ADC gain and offset calibration irrelevant. ADC linearity is much better than that of the sensor. However, in this safety critical application it is essential to determine correctly a rising or falling g, therefore monotonicity (DNL) is the most significant factor.
- Exhaust oxygen sensor (HEGO/Lambda sensor) — This is an active sensor that provides a voltage as a function of the partial pressure of oxygen across it. Nominally, the ideal air-to-fuel ratio causes it to produce around 450 mV, but the absolute voltage depends upon sensor temperature varying significantly even with a heated sensor. There is also a temporary sensor condition characteristic shift downward (CSD) that causes the voltage to shift. Often the algorithm searches for the midpoint between high and low peaks to determine the stoichiometric point. The ADC delivers the information with INL levels of accuracy.

2.3.2.3 Pragmatism

Users are strongly advised to evaluate their system realistically. For example, a typical 0.1% resistor may be as much as 1.5% over temperature, meaning 6 bits accurate. A good system is designed to work robustly in-spite of inaccuracy. It is unrealistic to expect 10 bits of accuracy in a production system and it has yet to be demonstrated. Levels of board space, board layers, wiring, supply decoupling, reference and component accuracy, and sensor accuracy required for this are not usually affordable in production ECUs.

This table makes some assumptions about system accuracies, noise elimination, and impedances and must be used as a guide only.

Table 1. Summary Examples

Sensor	Requirement	Calibration Level	Comments
Battery voltage	Absolute accuracy	Power up	0.1% divider resistors and 10mV reference
Battery voltage	Absolute accuracy	Run time	0.05% resistors over age and temperature, 3 mV reference, averaging
Temperature	Absolute accuracy	Power up	0.1% load resistor and 10mV reference
Temperature	Absolute accuracy	Run time	0.1% sensor, 3 mV tracking supply, 0.05% load resistor and 2 mV reference
Injector PWM current	Absolute accuracy	Power up	0.2% sense resistor
Injector current inflexion	Monotonicity (DNL)	None	Only waveform shape is important
Knock	Linearity (INL) Monotonicity (DNL)	None	a.c. sensor with wide amplitude variation
Throttle position sensor	Linearity (INL) Monotonicity (DNL)	None	Algorithm closes the loop to remove d.c. errors
Manifold pressure sensor	Absolute accuracy	Power up	0.2% sensor
HEGO for AFR	Absolute accuracy,	Power up	Fixed sense threshold voltage for stoichiometric
HEGO for AFR	Linearity (INL)	None	Algorithmic determination of stoichiometric
HEGO for individual cylinder	Monotonicity (DNL)	None	Only waveform shape is important

2.3.3 Total Unadjusted Error (TUE) as a Component in System Error

The ADC is measured in accordance with IEEE Standard 1241-2000 for gain, offset, DNL, and INL. TUE is not one of the specifications in this standard, but is provided for consistency with the MPC500 family. For MPC5500, TUE applies to results using the full ADC subsystem including the MAC.

Unfortunately, it is not possible to calculate TUE from the individual errors provided in the data sheet. This is because the errors are not independent or simple (linear). A simplified model of the errors is shown in Figure 7. INL is the divergence from the dark blue/diamond ideal line. The yellow/triangle line shows the output after calibration using the 25% and 75% reference points. The cyan/unmarked line shows the resulting linear response used to determine the calibrated gain (green/+, vertical top right) and offset (red/circle) lines.

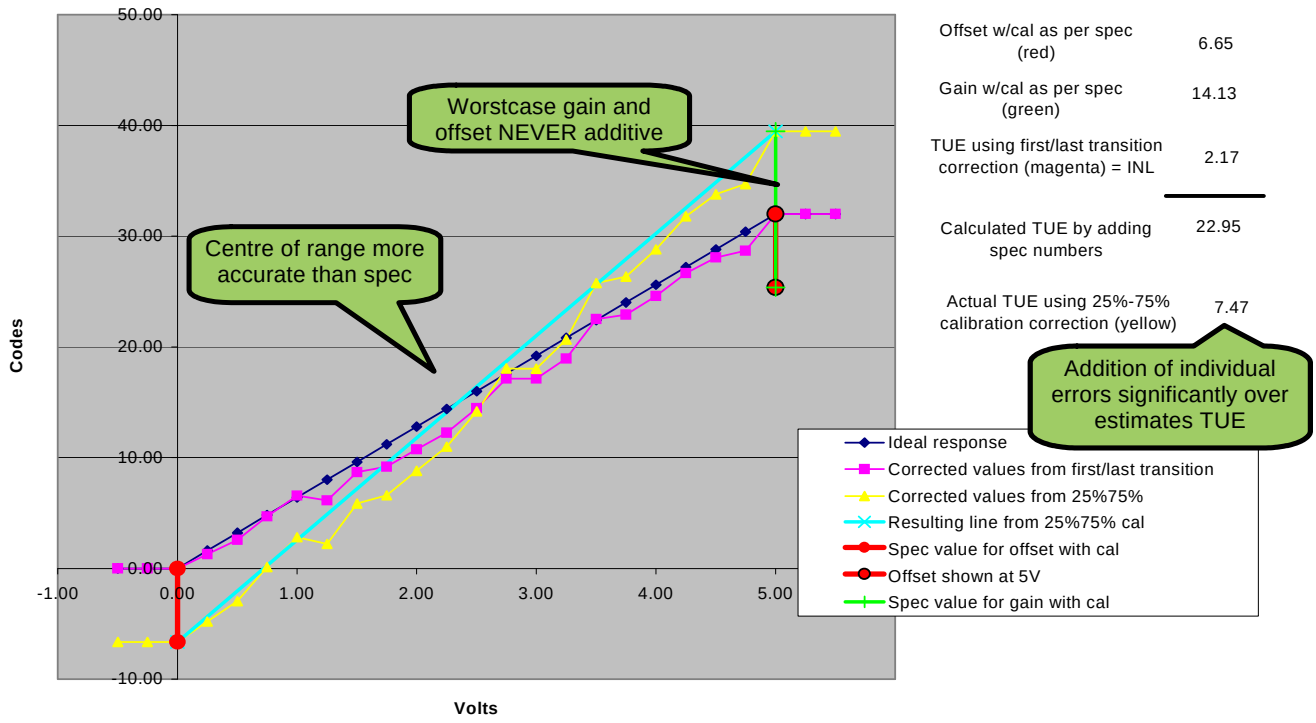


Figure 7. Model of ADC Errors

Graph observations:

TUE is dominated by gain and offset. This is true because the model is built to reflect MPC5500 performance.

Worst case TUE occurs when one reference goes high and the other low. In this case, offset always cancels out some of the gain error. This is one reason why addition of the gain and offset numbers does not provide TUE. The sum of actual errors is shown top right at 22.95 LSB, but the real TUE is only 7.47.

Worst case errors always occur at the ends of the response. This is good for control systems because the active operating range is usually centred with the ends being used for failure mode and effect analysis (FMEA). This means the ADC operates at a more accurate level than the data sheet suggests.

It might be thought, if the absolute accuracy of the references were known then the TUE can be calculated. This is not the case. The references are not points for calibration they are used by the RSD algorithm therefore the relationship is complex. The references are also subject to INL when read this is not a calculable error. To determine their accuracy directly they have to be tested, but the action of bringing the references outside the device for measurement significantly degrades the ADC performance. Instead, the method used to test TUE includes any error in the references.

It also possible that external references can be used to increase the ADC TUE accuracy. In theory this is possible. In practice there are several hurdles. It is better to do this with the ECU in a production-ready state, but this means that there must be access to the internal ADC reference supplies. The sensor tracking supply is available, but unloaded it has a few mV differences. The ADC inputs have protection resistors

therefore the input leakage of the ADC causes a few mV differences (+/-) between the pin and the ADC. If the ECU is open there are problems remain. Software must drive the calibration disturbing the power supplies and references. Connection must be made directly to the ADC references and test channels. The test voltages must be able to respond to the measured ADC reference voltage (both VRH and VRL) to provide the calibration points. To make the exercise worthwhile, the calibration points must be delivered and read with better accuracy than the internal points and based on the measured TUE it is less than 4 mV. This is not the end of the problem. The internal calibration software must be written to adjust the values with the temperature. The exercise is negated if the module is re-flashed at the dealer.

3 Applying Calibration to the ADC

This section describes the method for ADC calibration. Two levels of calibration are considered; on power up only, and during run-time. The details of calibration depend upon the characteristics of the system where the ADC is used therefore experiments to optimize calibration are explained. Calibration software is provided in Appendix A.

3.1 Outline of Power Up Calibration

Figure 8 is a flow chart of the procedure for power up calibration. The steps are described in detail. Although it is written for a single ADC, both ADCs require calibration. This can be performed in parallel if desired.

1. Power is applied to the module. The device comes out of reset at default frequency with ADCs disabled. Run initialization code includes ramping to the final frequency and configuring the DMA channels to run the ADC queues.
2. Enable the ADC. This involves writing to registers associated with the ADC that are not memory mapped. A queue can be used for this. The ADC must be set to final frequency.
3. After the current change caused by changing the device frequency, allow a delay to permit the power supply to settle. The delay depends upon the power supply typically no more than 100 µs. The ADC also takes about 10 µs to charge internal voltages.
4. Ensure the reference voltages have settled. This delay is due to the charge up of the capacitor on REFPYBC. If 100 nF is used as recommended, 8 ms after the power is applied to the MCU is sufficient. The ADC reads low until this capacitor is charged.
5. Take multiple readings of the 25% and 75% reference channels. Ensure that simultaneous reads of the 25% reference channel of the ADC 0 and ADC 1 do not occur. Simultaneous reads of the 75% reference channel of both ADCs must also be avoided. Long sample time must be used, but the calibration bit must not be set. Read Section 3.4, “Determining Noise Level” to determine numbers. It is possible to write to both ADCs with one queue. It is also possible to have the results sorted by the DMA by using the tag field in the conversion command word. Reads from each reference can be sent to a separate results queue.
6. Wait for the conversions to finish. This is calculable duration based on ADC setup. It is not necessary to be idle, but it is sensible to avoid activities that create circuit board noise near the ADC.
7. Average the results of each reference using a mean or median filter. See Section 3.4, “Determining Noise Level”.

8. Calculate the calibration coefficients. See Section 3.5, “ADC Internal Registers”. These must be checked against limits to ensure correct system operation. The maximum value for uncalibrated offset is in the data sheet at 12 LSB resolution. 4x this value must contain the offset as calculated by software (14-bit), plus a margin for the noise as determined on the target system. The maximum value for uncalibrated gain is also given in the data sheet again at 12 LSB resolution. This must be converted to a ratio to compare against the calculated GCC. The ADC has a range of 4096 counts. The ratio limit becomes (4096+/-(offset error))/4096.
9. The calculated values of OCC and GCC can now be written to the ADC.
10. It is advisable to re-read the reference channels using the new calibration. They must match the ideal 25% and 75% results plus a margin that depends upon the noise on the system.

This completes power up calibration. The process must be performed at least once per device. However, at that point one option is to store the calibration using EEPROM emulation. This might be used to save a small amount of time from boot up or if run time calibration is employed it begins without having to repeat the power up calibration.

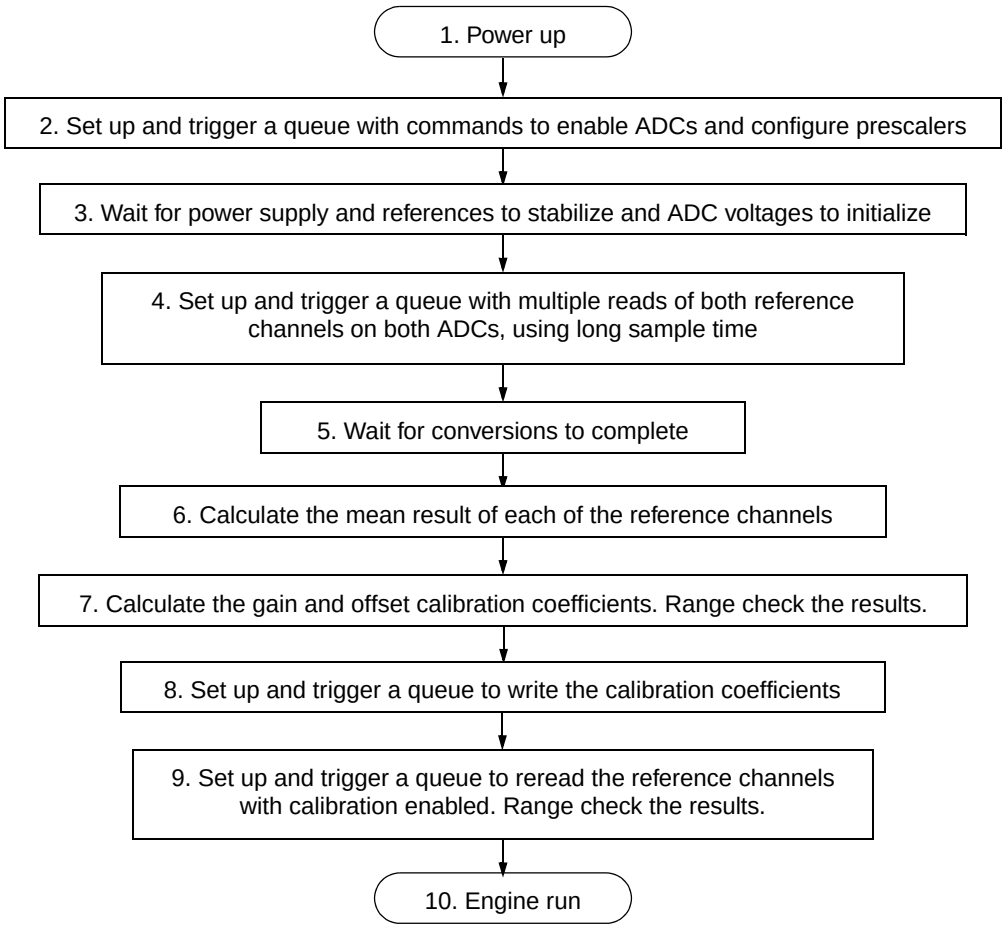


Figure 8. Power Up Calibration Procedure

3.2 Outline of Run Time Calibration

Figure 9 is a flow chart of the procedure for run time calibration. The first two steps describe power up calibration and the subsequent steps describe new software that must be written.

- Step 3. This represents the start of the operating system and repetitive application code.
- Step 4. As with power up calibration, a sample time of 2 clocks for the 75% reference channel and 64 clocks for the 25% reference channel is recommended. Simultaneous reads of the same reference channel of both ADC0 and ADC1 must be avoided. Only a single read is required because averaging is carried out later. Neither timing nor coherency of the reference reads is critical. They can be performed using low priority commands. The aim of run time calibration is to compensate for temperature drift. Although the range of temperature drift is small, self heating of the device can cause a relatively fast initial temperature rise (seconds) if the ambient temperature is low. The rate of reads must be several times faster than the rate of drift of the ADC. A time based queue at 100 ms can be a suitable for this. See Step 5.
- Step 5. The recommended filter for run time calibration is a standard digital (software) low pass. This filter uses few resources and is easy to tune with a single gain constant. The filtering must be sufficiently severe for the result to be free from noise leaving a temperature drift. To establish this, the gain constant must be reduced until it is sufficiently small for noise not to affect the calibration. This can be established by monitoring the filter output when the device is at a constant temperature. If this means that the filter is now too slow to respond to temperature changes then the rate of reads must be increased. The level of noise rather than the rate of temperature drift is now the determining factor for the filter.

NOTE

The time constant of the filter depends upon both the gain constant and the rate of reads. The gain constant must be recalculated if the rate of reads is changed.

- Step 6. Drift of the ADC can be seen from drift in the reference channel results. It is not necessary to calculate the calibration coefficients to identify drift. The main factor to declare drift is the accuracy required by the application. The fact that run time calibration is in use implies that accuracy to the last few LSB is the aim and other inaccuracies including noise on the system and references demonstrate themselves capable of this. In which case, declaring drift at only one or two LSB is advantageous. See Step 7.
- Step 7. There is a coherency to be avoided when re-writing calibration coefficients. This occurs when the coefficients are written using any queue other than the highest priority, Q0. What occurs is a conversion is scheduled due to its higher priority after the update of one calibration coefficient (for example OCC) but before the update of the other coefficient (for example GCC). The issue is circumnavigated when a change in coefficients is declared with minimal drift (Step 6). In this case, there is only one or two LSB changes in offset or gain. Therefore, even if a high priority conversion takes place in between the update of coefficients, the effect of the incoherency is within the error of the system. Using this method, calibration coefficients can be updated using a low priority queue.

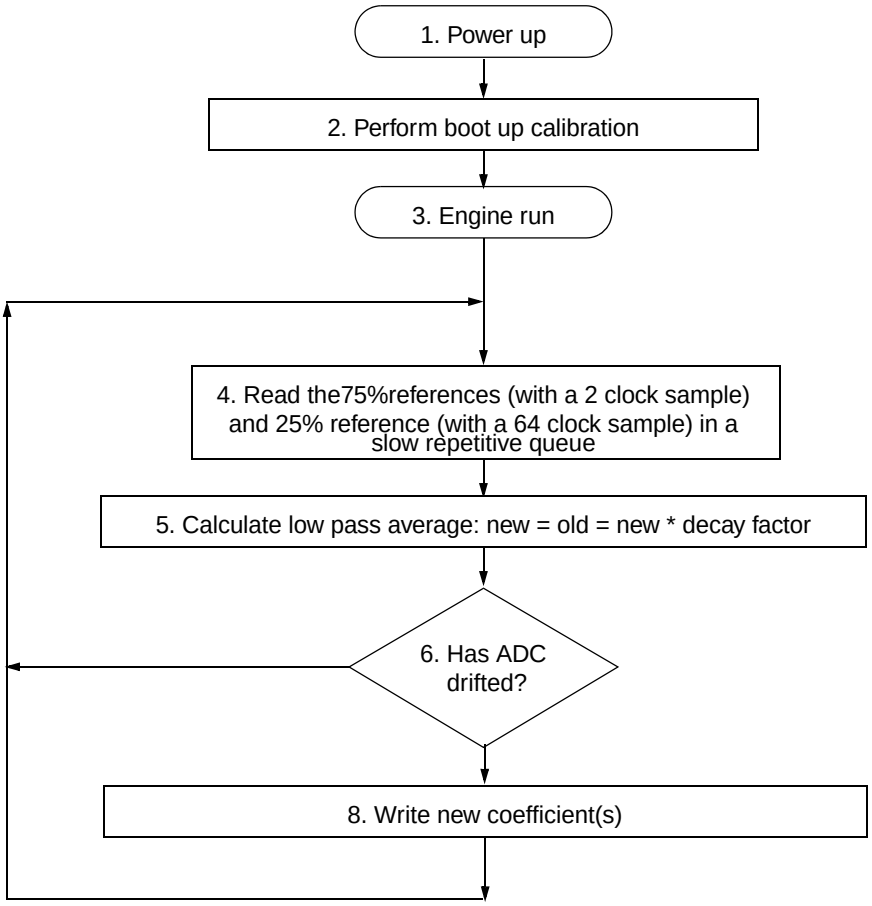


Figure 9. Flow Chart of Calibration Procedure

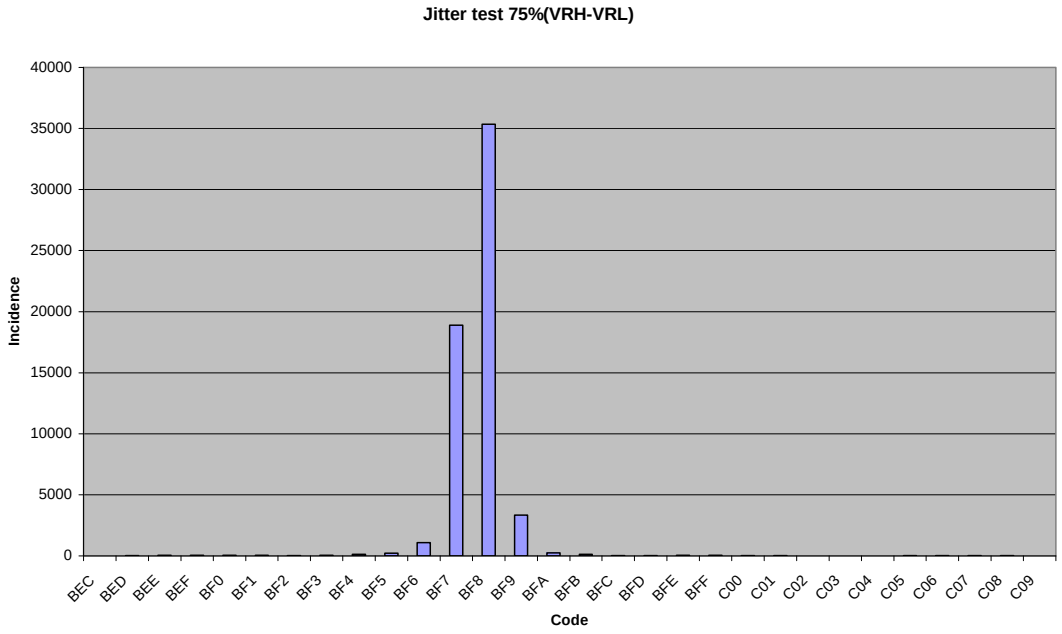
3.3 Simultaneous Reads of Reference Channels

Simultaneous reads of a reference channel by both ADC's result in incorrect results being returned from a conversion. It must be ensured when reading the 25% and 75% reference channels that ADC0 and ADC1 do not simultaneously read the same reference channel. To avoid this occurrence it is recommended to calibrate alternatively the ADCs. This consists of run time calibration being performed every 1 second and calibrating alternative ADCs on each occurrence. For example, calibrate ADC0 on the first second, ADC1 on the second second, ADC0 on the third second, and so on. This time period can be reduced if required, however as run time calibration is used to compensate for temperature drift this frequency of the run time calibration is acceptable for most engine environments.

3.4 Determining Noise Level

Because the calibration coefficients affect every subsequent reading by the ADC, make them as precise as possible. Precision can be improved by averaging out noise on the reference channels. This is achieved using multiple reads then calculating an average in software. There are two items to be decided, the number of samples and the average to use.

To determine the noise on the system the only thing required is to take multiple readings on the reference channels and look at the data. Figure 10 shows an example set of results obtained by 60,000 reads on the 75% reference channel. A single read might lead to poor calibration, The span of results is 29 codes, however nine out of ten results are either 0xBF7 or 0xBF8. It does not take many reads to see the trend. In this case five or six reads are sufficient.



Code	Incidence
BEC	9
BED	17
BEE	47
BEF	54
BF0	57
BF1	38
BF2	34
BF3	50
BF4	128
BF5	230
BF6	1083
BF7	18897
BF8	35341
BF9	3332
BFA	248
BFB	118
BFC	32
BFD	36
BFE	39
BFF	45
C00	29
C01	32
C02	11
C03	11
C04	11
C05	15
C06	20
C07	16
C08	14
C09	6
Total	**60000**

Figure 10. Results of a Jitter Test

The results in Figure 10 suggest that this board has a generally low level of noise because the majority of reads span only a few codes. They also suggest occasional large noise spikes because the spread is large, but with low incidence. With a distribution like this it can be seen that a mean average is not a good choice. A single extreme read may lead to a poor result. Looking for the modal (most common) result, or median (centre value) is better signal processing techniques.

Significant variation has been found in different applications. If the distribution were closer to a Gaussian (bell shaped) curve (over 10 codes) this would suggest more persistent noise, but at a lower amplitude. The greater number of common codes mean that more reads are required. With a Gaussian response, a mean average might be most suitable.

Noise appearing on the references can come from a wide variety of sources, through capacitive, inductive coupling, or conduction via the reference pins VRH and VRL. Running an experiment to determine the level of jitter on the target system generates data to allow an averaging scheme to be determined.

Two final notes:

Firstly, a sample time of 2 and 64 counts for the 75% and 25% reference channels is recommended. This is due to the impedance and settling time of the internal references.

Secondly, if the system proves to be very noisy, eliminate the coupling of the noise to not run excessive sample numbers. If more than 30 reads are required it is likely that noise is getting onto the references. Even though averaging may be used to provide the ideal calibration each read during operation is subject to a similar level of noise. Such interference negates the effort in determining the ideal calibration.

3.5 ADC Internal Registers

There are six 16-bit registers in each MPC5500 ADC engine, one for conversion command messages, and five control registers for configuring the ADC. These registers are non memory-mapped and are only accessible via read and write command messages. To read a value from an ADC register, a read command message with the address of the register is sent to the ADC. The contents are returned as a 16-bit result. To write to an ADC register, the 16-bit value is embedded in a write command message along with the a ADC register address. The header field of the read or write command message directs the register value to the appropriate ADC engine.

Table 2. ADC Engine Internal Register Map

Register Address	Register Name	Description
0x00	—	Used for conversion commands
0x01	ADCn_CR	Control register
0x02	ADCn_TSCR	Time stamp control register
0x03	ADCn_TBCR	Time base control register
0x04	ADCn_GCCR	Gain calibration control register
0x05	ADCn_OCCR	Offset calibration control register
0x06 – 0xFF	Reserved	—

Two of these non memory-mapped ADC internal registers are involved in ADC calibration, the gain calibration constant and the offset calibration constant registers. These must be written with an application software calibration routine before the MAC calibration unit can be invoked by conversion command messages.

3.5.1 Gain Calibration Constant Register (ADCn_GCCR)

The gain calibration constant register is a 15-bit fixed point register. It has fifteen significant bits in two bit fields. Bit 1, GCC_INT, contains the integer portion of the gain value. Bits 2–15, GCC_FRAC, contain the fractional portion of the gain value. There is an independent GCC register for each ADC0 and ADC1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	GCC_INT	GCC_FRAC													

Bit(s)	Bit Field Name	Bit Field Description
0	—	Reserved
1	GCC_INT	Integer part of the gain calibration constant for ADCn.
2–15	GCC_FRAC	Fractional part of the gain calibration constant for ADCn. GCC_FRAC can express decimal values ranging from 0 to 0.999938964: $2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-11} + 2^{-12} + 2^{-13} + 2^{-14}$

3.5.2 Offset Calibration Register (ADCn_OCCR)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	OCC													

Bit(s)	Bit Field Name	Bit Field Description
0–1	—	Reserved
2–15	OCC	ADCn offset calibration constant. Contains the offset calibration constant used to fine-tune ADCn conversion results. Negative values must be expressed using the twos complement representation.

3.6 Calculating the Calibration Coefficients

3.6.1 Obtaining the Equation Terms

The following steps can be used to calculate GCC and OCC. The ideal 75% VREF and 25% VREF conversion values of a 12-bit ADC are:

```
2^12 * 75% = 3072 = 0xC00; 0xC00 << 2 = 0x3000

2^12 * 25% = 1024 = 0x400; 0x400 << 2 = 0x1000

IDEAL_RES75 = 0x3000
IDEAL_RES25 = 0x1000
```

It is necessary to left-shift the ideal result twice to match the left-shifts applied by the MAC on the actual reference reads. All terms are 14 bits. In the reference manual, the ideal results are called CAL_RES 75%VREF and CAL_RES 25%VREF.

Determining the uncalibrated (raw) performance of the ADC. Read the two reference points, 25% (VRH-VRL) and 75% (VRH-VRL). These conversions must be done using conversion command words with CAL = 0.

```
0x00002B00;     // BN=0 ADC0
                // CAL=0
                // Message Tag=0 Result in RFIFO 0
                // LST=0b00            2 ADC clks
                // Convert 75% VREF channel 43(0x2B)

0x00082C00;     // BN=0 ADC0
                // CAL=0
                // Message Tag=0 Result in RFIFO 0
                // LST=0b10         64 ADC clks
                // Convert 25% VREF channel 44(0x2C)
```

These conversion command words return the uncalibrated conversion results for the ADC; `raw_res75` and `raw_res25`. They are compared against the ideal results for these reference voltages.

3.6.2 Gain Calibration Coefficient

Calculate the gain calibration coefficient (GCC) using the familiar formula for the slope of a line:

$$m = (y2 - y1) / (x2 - x1) \qquad \textit{Eqn. 2}$$

Using the terms of the ADC, the equivalent formula is:

$$GCC = (IDEAL\_RES75 - IDEAL\_RES25) / (raw\_res75 - raw\_res25) \qquad \textit{Eqn. 3}$$

GCC is an unsigned 15-bit fixed point value with a single-bit integer portion and a 14-bit fractional portion, GCC_int.GCC_frac. GCC can have the range of 0 to 1. 999938964 (see Section 3.5.1, “Gain Calibration Constant Register (ADCn_GCCR)”). Numerically this is a 15-bit integer divided by 2^14.

The result of Equation 3 must be converted from the native floating point or integer format into the 15-bit format of the GCC register. There are several to do this.

The fastest way is using long (32-bit) integer arithmetic. Use the formula:

$$(a * 2^{14})/b \qquad \textit{Eqn. 4}$$

The equivalent of this in C code is:

```
gcc = ((IDEAL_RES75 - IDEAL_RES25)<<14) / (raw_res75 - raw_res25)
```

The bracketed ideal half of this equation is a constant.

```
(IDEAL_RES75 - IDEAL_RES25) = 0d8192 = 0x2000
```

Making the equation:

```
gcc = 134217728/(raw_res75 - raw_res25);
```

If using floating-point arithmetic, use the formula (a/b) * 2^14 and round the result to the nearest integer. This 15-bit integer can then be written directly into the GCC register:

```
gcc = (IDEAL_RES75 – IDEAL_RES25) / (raw_res75 – raw_res25) * 16384
```

Or, using constants:

```
gcc = 8192/(raw_res75 - raw_res25) * 16384;
```

These can be simplified further into the same code as used with integers.

3.6.3 Offset Calibration Coefficient

Solving the MAC equation for OCC gives the formulae below:

$$\text{OCC} = \text{IDEAL\_RES75} - \text{GCC} * \text{raw\_res75} - 2 \qquad \textit{Eqn. 5}$$

or

$$\text{OCC} = \text{IDEAL\_RES25} - \text{GCC} * \text{raw\_res25} - 2 \qquad \textit{Eqn. 6}$$

The –2 is there to take account of the half LSB correction always added by the MAC. Because it is present on the raw results, remove it to calculate the appropriate OCC.

In C code using long integer arithmetic can be done simply as:

```
occ = IDEAL_RES75 - ((gcc * raw_res75)>>14) - 2;
```

The OCC can be a positive or negative number centred around zero (ideal offset). Negative numbers are represented as two’s complement. The ADC register for OCC is a 14-bit value. Writing the 16-bit or 32-bit result to the 14-bit register ignores the leading bits.

If OCC and GCC are defined as a floats then the integer portion of occ

```
occ = IDEAL_RES75 - (gcc * raw_res75) - 2;
```

is the correct value to be programmed to the OCC register. Ideally it is rounded rather than truncated.

When calibrated conversion results are required after the GCC and OCC values are known and formatted they must be written to the appropriate ADC internal registers for use by the MAC. Write GCC value to ADCn gain calibration registers and OCC value to ADCn offset calibration constant registers using write configuration commands.

NOTE

There will be different GCC and OCC values for each ADC. The correct values must be identified independently and written to the appropriate ADC.

4 Conclusion

The RSD architecture of ADC combined with the hardware MAC delivers a significant advance in ADC performance for the MPC5500 family. Accuracy is beyond previously achievable levels, in fact, beyond the capability of typical measuring equipment. Linearity is particularly good providing closed loop sensors with precision of 2 or 3 mV. This available precision brings the need to understand the application. In many cases, the ADC accuracy is better than the accuracy of the control system to which it is fitted, therefore the discussion on sensor types, total accuracy, linearity, and experiment to determine noise floor.

After the task is understood and the required accuracy determined, the code required to achieve calibration is straightforward. Some simple arithmetic is required and some results reformatting to suit the hardware MAC. The processes described achieve calibration either on power up, or during run time.

5 References

1. *MPC5553/MPC5554 Microcontroller Reference Manual (MPC5553/54RM)*
2. *AN2438 — ADC Definitions and Specifications, J. Feddeler and Bill Lucas*

NOTE

Reference 1 uses integral non linearity (INL) for linear error, differential non linearity for monotonicity, and total unadjusted error (TUE) after calibration for absolute error. To understand these terms in detail, refer to the document reference 2.

6 Glossary

ADC	analog to digital converter
CSD	characteristic shift downward
DAC	digital to analog converter
DMA	direct memory access
DNL	differential non linearity
ECU	engine control unit
FMEA	failure mode and effect analysis
GCC	offset gain coefficient
INL	integral non linearity
LSB	less/least significant bit
MAC	multiply accumulate
MEMS	micro-electromechanical system
MSB	more/most significant bit
OCC	offset calibration coefficient
RSD	redundant signed digit
SA	successive approximation
TUE	total unadjusted error

Appendix A Software Example

```
#include "mpc5554.h"
#include "common.h"

#define REGSIZE 15

int cal_eqadc(uint16_t dac)
{
   const IDEAL_RES75 = 0x3000;
   const IDEAL_RES25 = 0x1000;

   unsigned long raw_res75;
   unsigned long raw_res25;

   unsigned long gcc, occ;
            float gccf, occf;

   unsigned short gain;
            short offset;

   unsigned long i, numerator, denominator;
   unsigned long bit, mask;

   // configure Queue0 and send conversion command messages
   EQADC.CFCR[0].B.MODE = SW_TRIG_SS;
  EQADC.CFCR[0].B.SSE = 1;  /*  software trigger */ EQADC.CFPR[0].R = 0x00002B00; + (dac<<25);
/* Convert channel 43 (75%VRH)with CAL=0  */
   EQADC.CFPR[0].R = 0x00182C00; + (dac<<25);   /* Convert channel 44 (25%VRH)with CAL=0  */

   // wait for results
   while (EQADC.FISR[0].B.RFDF==0) {};
   raw_res75 = EQADC.RFPR[0].R;  /* 75%VRH */
   while (EQADC.FISR[1].B.RFDF==0) {};
   raw_res25 = EQADC.RFPR[1].R;  /* 25%VRH  */

/* -------------- Method 1: (Fastest) ------------------ */

   gcc = occ = 0;

   gcc = ((IDEAL_RES75-IDEAL_RES25)<<14)/(raw_res75-raw_res25);
   occ = IDEAL_RES75 - ((gcc * raw_res75)>>14) - 2;

   gain = gcc;
   offset = occ;

/*------------------ End of Method 1 -------------------- */
```

```
/* -------- Method 2:  Floating Point (Slowest) --------- */

   gccf = (8192.0)/(raw_res75 - raw_res25);
   gain = gccf * 16384;
   occf = IDEAL_RES75 - (gccf * raw_res75) - 2.0;
   offset = occf;

/*------------------ End of Method 2 -------------------- */


/* ------------ Method 3 (Alternative) ------------------ */
   bit  = 1 << (REGSIZE - 1);
   mask = (1 << REGSIZE) - 1;

   gcc = occ = 0;

   denominator = raw_res75 - raw_res25;
   numerator = IDEAL_RES75 - IDEAL_RES25;

   for(i = 0; i < REGSIZE; i++)
   {
      if(numerator >= denominator)
                {
                        numerator -= denominator;
                        if(numerator > denominator)
                        {
                                //GCC is greater than 1.999 and we have an overflow
                                return 0;
                        }
                        gcc |= bit;
                        occ += raw_res75;
      }
                bit >>= 1;
                numerator <<= 1;
                occ <<= 1;
                numerator &= mask;
   }
   occ = occ >> REGSIZE;
   occ = IDEAL_RES75 - occ -2;

   gain = gcc;
   offset = occ;

/* ------------------ End of Method 3 ------------------ */

   // now store Gain in ADCn_GCCR
   EQADC.CFPR[0].R = (gain<<8) | 0x00000004 | (dac<<25);

   // now store offset in ADCn_OCCR
   EQADC.CFPR[0].R = ((unsigned short)offset << 8) | 0x00000005 | (dac<<25);

   return 1;
}
```

Appendix B Example Coefficient Calculations

In this example, the results of multiple reads and averaging is:

raw_res75 = 0x2FE7 = 12263, raw_res25 = 0x100D = 4109

and we know:

IDEAL_RES75 = 0x3000 = 12288, IDEAL_RES25 = 0x1000 = 4096

and to recap on the two equations:

$$GCC = (IDEAL\_RES75 - IDEAL\_RES25) / (raw\_res75 - raw\_res25) \quad \textit{Eqn. 7}$$

$$OCC = IDEAL\_RES75 - GCC*raw\_res75 - 2 \quad \textit{Eqn. 8}$$

Floating point version:

Define variables gccf and occf as floats with gcc and occ these are the values to be programmed into the registers. The integer ADC values in decimal are shown for visibility.

gccf = 8192/(12263 – 4109) = 1.00466

occf = 12288 – (1.00466*12263) – 2 = -34.149

To get the register format:

gcc = ROUND(1.00466 * 16384) = 16460

occ = ROUND (-34.149) = -34

Integer version:

Define variables gcc and occ as ints. The integer ADC values in decimal are shown for visibility. (IDEAL_RES75 - IDEAL_RES25)<<14) is a constant 134217728.

gcc = 134217728/(12263-4109) = 16460

occ = 12288 – (16460*12263)/16384 – 2 = -33

The integer gcc result was automatically truncated from 16460.354 giving an occ that is again truncated from –33.88 therefore the slight difference in occ.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2989
Rev. 2
02/2010

freescale™
semiconductor