Application Report
SLIA087A–May 2015–Revised December 2017

Guide to Using the GPADC in TPS65903x, TPS65917-Q1, TPS65919-Q1, and TPS65916 Devices

ABSTRACT

This application report describes the detailed methods of using the GPADC in the TPS659038-Q1, TPS659039-Q1, TPS659037, TPS65917-Q1, TPS65919-Q1, and TPS65916 devices.

Contents

List of Figures

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

This application note describes the details of using the GPADC module in the TPS65916, TPS65917-Q1, TPS65919-Q1, TPS659037, TPS659038-Q1, and TPS659039-Q1 devices. The GPADC offers two methods of conversion: software conversion and automatic conversion. This document also describes how to use each of the conversion methods and the workarounds for known issues with the GPADC. This document does not describe any specifications for the GPADC. For such information, refer to the corresponding device data sheet.

2 GPADC Conversion Modes

The devices offer multiple GPADC channels to monitor voltages that are internal and external to the power-management IC (PMIC). For detailed information on these channels, refer to the GPADC section in the corresponding device data sheet. The GPADC offers two different conversion modes: software conversion and automatic conversion. The following sections describe how to set up and use each conversion mode.

2.1 Starting a Software Conversion

A software (SW) conversion, also called an asynchronous conversion, is a one-time conversion that must be initiated by the host microprocessor. When this conversion is initiated, the PMIC asserts a GPADC_EOC_SW interrupt if unmasked which indicates that the conversion is complete and the results are available.

The following pseudo-code can be used to initiate a software conversion. This example shows how to read the voltage on external channel 1:

```
# Unmask interrupt for GPADC_EOC_SW
INT3_MASK.GPADC_EOC_SW = 0             # I2C address 0x59, register 0x1B

# Enable software conversion, select channel 1, and start conversion
GPADC_SW_SELECT = 0x91                 # I2C address 0x59, register 0xCD

# Wait for any pending conversion to complete
while GPADC_STATUS == 0x00             # I2C address 0x59, register 0xC4
    sleep
end while

# Read result of SW conversion 0
lsb = GPADC_SW_CONV0_LSB               # I2C address 0x59, register 0xCE
msb = GPADC_SW_CONV0_MSB               # I2C address 0x59, register 0xCF
result = (msb << 8) + lsb
```

2.2 Starting an Automatic Conversion

An automatic conversion is a conversion that runs periodically on one or two channels during operation. The conversion period is configurable in powers of 2 from 1/32 s (31.25 ms) to 1024 s. During each period one GPADC conversion begins and the results are written to the corresponding results registers. Each automatic conversion that follows overwrites the previous results. A threshold can also be configured so that an interrupt is generated whenever the automatic conversion result is either above or below the defined threshold depending on the GPADC_THRES_CONVx_MSB.THRES_CONVx_POL bit.

In this example, automatic conversion 0 will be configured to monitor the voltage on external channel 1 every 31.25 ms, and generate an interrupt when this voltage exceeds 1 V. Use Equation 1 to calculate the ideal threshold value for a target voltage.

$$\frac{V_{threshold}}{V_{maximum}} \times 4095 = \text{Ideal code} \quad (1)$$

where

- $V_{threshold}$ is the target threshold voltage.
- $V_{maximum}$ is the maximum voltage for that channel.

The maximum voltage for channel 1 is 1.25 V and the target threshold voltage is 1 V. Therefore, Equation 2 shows the resulting ideal code.

$$\frac{1}{1.25\ \text{V}} \times 4095 = 3276 \tag{2}$$

The result is 3276 in decimal, which is 0xCCC in hexadecimal.

To account for tolerances of the ADC, the ideal code, 0xCCC, should be adjusted based on the ADC specifications of the device. These specifications include the integral nonlinearity (INL), offset, and gain error. To adjust the automatic conversion threshold, use Equation 3.

$$(\text{Ideal code} + \text{INL}) \times \text{Gain error} + \text{Offset} = \text{Maximum code} \tag{3}$$

The maximum code describes the worst case code the ADC would output given an ideal code. The gain error includes both the gain-error specification and the gain-error drift which includes the variance due to temperature. In the case of TPS65903x device before ADC calibration, INL = 3.5 LSB, gain error = 3.5% + 0.2% = 3.7%, and offset error = 50 LSB. Equation 4 shows the equation using these values.

$$(3276 + 3.5) \times 1.037 + 50 = 3451 \tag{4}$$

The maximum code that could be output for a 1-V input is 3451 or 0xD7B in hexadecimal. To achieve a more accurate ADC reading, the result can be calibrated to correct for gain error and offset error. For more information on calibrating the ADC results, see Section 2.4.

The following pseudo-code can be used to setup an automatic conversion for this example.

```
# Unmask interrupt for GPADC_AUTO_0
INT3_MASK.GPADC_AUTO_0 = 0x00        # I2C address 0x59, register 0x1B

# Set automatic conversion 0 to monitor channel 1
GPADC_AUTO_SELECT = 0x01             # I2C address 0x59, register 0xC8

# Set threshold to 1 V and configure threshold for overvoltage detection
GPADC_THRESH_CONV0_MSB = 0x0D        # I2C address 0x59, register 0xD1
GPADC_THRESH_CONV0_LSB = 0x7B        # I2C address 0x59, register 0xD0

# Enable automatic conversion 0, and set period to 31.25ms
GPADC_AUTO_CTRL = 0x10               # I2C address 0x59, register 0xC3
```

2.3 Using Automatic Conversion to Generate a Shutoff Request

The GPADC can also be configured to shutdown based on the results of a GPADC automatic conversion. To generate a shutoff request based on the automatic conversion results, the GPADC_AUTO_CTRL.SHUTDOWN_CONVx bit should be set to 1.

The following pseudo-code shows the same conditions as the previous example, but configured to use automatic conversion 0 to generate a shutoff request when the voltage on external channel 1 is higher than 1 V.

```
# Unmask interrupt for GPADC_AUTO_0 and enable shutdown control
INT3_MASK.GPADC_AUTO_0 = 0x40        # I2C address 0x59, register 0x1B

# Set automatic conversion 0 to monitor channel 1
GPADC_AUTO_SELECT = 0x01             # I2C address 0x59, register 0xC8

# Set threshold to 1 V
GPADC_THRESH_CONV0_MSB = 0x0D
GPADC_THRESH_CONV0_MSB = 0x7B

# Enable automatic conversion 0, and set period to 31.25ms
GPADC_AUTO_CTRL = 0x10               # I2C address 0x59, register 0xC3
```

2.4 Calibrating the GPADC Result

To correct for the gain error and offset error of the ADC result, the ADC result code should be calibrated based on the values stored in the one-time programmable (OTP) memory of the device. The gain and offset errors after calibration are much smaller than before calibration, and therefore a calibrated result provides a code much closer to the actual voltage than a noncalibrated result.

Each GPADC_TRIM register stores a 7-bit calibration value in bits 7 through 1, and a sign-bit in bit 0 of the register. When the sign bit is 1, the calibration value is negative. When the sign bit is 0, the calibration value is positive.

Figure 1. GPADC_TRIMx Register

7	6	5	4	3	2	1	0
Calibration value							Sign

The calibration formulas are as follows:

- Gain

$$k = 1 + \frac{(D2 - D1)}{(X2 - X1)} \quad (5)$$

  where

  - D1, D2, X1, and X2 can be found in the calibration section of the corresponding device data sheet.

- Offset

$$b = D1 - (k - 1) \times X1 \quad (6)$$

- Corrected code

$$a' = \frac{(a - b)}{k} \quad (7)$$

  where

  - a is the ADC code result from a conversion.

The following pseudo-code shows how to calibrate a SW conversion result from channel 1 on the TPS65903x device and stores the calibrated result in the sw_calibrated variable. This example assumes that a SW conversion has already been completed using SW conversion 0.

```
# Read result of SW conversion 0
lsb = GPADC_SW_CONV0_LSB              # I2C address 0x59, register 0xCE
msb = GPADC_SW_CONV0_MSB              # I2C address 0x59, register 0xCF
sw_result = (msb << 8) + lsb

# Calculate D1 and D2 from the TRIM registers
trim1 = GPADC_TRIM1                   # I2C address 0x5A, register 0xCD
trim2 = GPADC_TRIM2                   # I2C address 0x5A, register 0xCE

if trim1 % 2 == 0 then                # Check if bit 0 is equal to 0
    D1 = trim1 >> 1
else
    D1 = - (trim1 >> 1)
end if

if trim2 % 2 == 0 then
    D2 = trim2 >> 1
else
    D2 = - (trim2 >> 1)
End if

# Calculate gain, offset, and corrected value.
# For channel 1, X1 = 2064, X2 = 3112.
gain = 1 + ((D2 - D1)/(3112 - 2064))
offset = D1 - (gain - 1) * 2064
sw_calibrated = (sw_result - offset) / gain
```

The GPADC can also be calibrated for an automatic conversion. In this case, an ideal voltage target must be converted to a real ADC code, thus requiring the inverse of Equation 7. Use Equation 8 to calculate the real code, a.

$$a = (a' \times k) + b \quad (8)$$

where

- a' is the ideal corrected code.
- k is the gain in Equation 5.
- b is the offset in Equation 6.

The gain error, offset error, and INL should also be considered using Equation 3.

Using the same example in Section 2.2, the goal is to detect when the external channel 1 voltage exceeds 1 V which is represented by the ideal code, 0xCCC. The following pseudo-code takes this ideal code and converts it to the real code corresponding to 1 V based on the calibration parameters of the device. This pseudo-code then configures the automatic conversion to generate an interrupt when the input exceeds this value.

```
# Calculate D1 and D2 from the TRIM registers
trim1 = GPADC_TRIM1                                  # I2C address 0x5A, register 0xCD
trim2 = GPADC_TRIM2                                  # I2C address 0x5A, register 0xCE

if trim1 % 2 == 0 then                               # Check if bit 0 is equal to 0
    D1 = trim1 >> 1
else
    D1 = - (trim1 >> 1)
end if

if trim2 % 2 == 0 then
    D2 = trim2 >> 1
else
    D2 = - (trim2 >> 1)
end if

# Calculate calibrated value based on equations 5, 6, and 8
gain = 1 + ((D2 - D1)/(3112 - 2064))
offset = D1 - (gain - 1) * 2064
threshold = (0xCCC * gain) + offset

# Add tolerances to threshold based on equation 3
threshold = (threshold + 3.5) * 1.002 + 2
threshold_msb = threshold >> 8
threshold_lsb = threshold & 0xFF

# Unmask interrupt for GPADC_AUTO_0
INT3_MASK.GPADC_AUTO_0 = 0                           # I2C address 0x59, register 0x1B

# Set automatic conversion 0 to monitor channel 1
GPADC_AUTO_SELECT = 0x01                             # I2C address 0x59, register 0xC8

# Set threshold to the calculated threshold value
GPADC_THRESH_CONV0_MSB = threshold_msb               # I2C address 0x59, register 0xD1
GPADC_THRESH_CONV0_LSB = threshold_lsb               # I2C address 0x59, register 0xD0

# Enable automatic conversion 0, and set period to 31.25ms
GPADC_AUTO_CTRL = 0x10                               # I2C address 0x59, register 0xC3
```

3 Workarounds for Known Issues

3.1 Software Conversion After Warm Reset

After a warm reset has occurred, the next software conversion might not return a value which is possible depending on the internal state of the device. The first software conversion finishes but the conversion result registers read all zeroes. After this first conversion, all subsequent software conversions finish as expected. This issue is present in TPS65903x, TPS65917-Q1, TPS65919-Q1, and TPS65916 devices.

If SW conversions are used in the application, adding a dummy SW conversion before the first SW conversion after warm reset is recommended. The following pseudo-code shows one way to add this dummy SW conversion using SW conversion 0 on channel 1.

```
# Unmask interrupt for GPADC_EOC_SW
INT3_MASK.GPADC_EOC_SW = 0              # I2C address 0x59, register 0x1B

# Enable software conversion and select channel 1
GPADC_SW_SELECT = 0x81                  # I2C address 0x59, register 0xCD

#Start software conversion
GPADC_SW_SELECT = 0x91                  # I2C address 0x59, register 0xCD

# Wait for any pending conversion to complete
while GPADC_STATUS == 0x00              # I2C address 0x59, register 0xC4
    sleep
end while

# Read result of SW conversion 0
lsb = GPADC_SW_CONV0_LSB                # I2C address 0x59, register 0xCE
msb = GPADC_SW_CONV0_MSB                # I2C address 0x59, register 0xCF

# If results show all zeroes, start a second software conversion
if lsb == 0x00 and msb == 0x00 then
    #Start software conversion
    GPADC_SW_SELECT = 0x91              # I2C address 0x59, register 0xCD

    # Wait for any pending conversion to complete
    while GPADC_STATUS == 0x00          # I2C address 0x59, register 0xC4
        sleep
    end while

    # Read result of SW conversion 0
    lsb = GPADC_SW_CONV0_LSB            # I2C address 0x59, register 0xCE
    msb = GPADC_SW_CONV0_MSB            # I2C address 0x59, register 0xCF
end if

result = (msb << 8) + lsb
```

If it is unknown which software conversion will be first after a warm reset, one dummy software conversion could be added to the code that is executed immediately after warm reset and the result is unused.

3.2 Recovering from Locked GPADC State

Whenever an automatic conversion is in progress and interrupted by either a cold reset or disabling the automatic conversion, the GPADC gets stuck in a busy state. While in the busy state, no SW or automatic conversions finish. To get out of this state, the user must flush the ADC. If the application does not use an automatic conversion, this issue can be ignored. This issue is only present in TPS65903x devices and is not present in the TPS65917-Q1, TPS65919-Q1, and TPS65916 devices.

When the GPADC is in the stuck state, the GPADC_STUCK.STUCK bit is set to 1. If this bit is set, the GPADC can be recovered by setting the GPADC_FLUSH_EN.FLUSH_EN bit, followed by toggling the GPADC_FLUSH.FLUSH bit to 1 and back to 0. When flushed, the GPADC_STUCK register bits 3 to 0 are set to random bits which can be ignored. The GPADC_STUCK.STUCK bit reads out 0, and any automatic or SW conversion then finishes as expected. Because the GPADC can get stuck after a cold reset or after disabling the automatic conversion, running this recovery routine after the boot process and after an automatic conversion is disabled is recommended. Alternatively, this recovery routine could be placed before every GPADC conversion.

The following pseudo-code describes the recovery routine to check if GPADC is in the stuck state and then flush the ADC if it is stuck. The order of the last two writes to GPADC_FLUSH_EN and GPADC_FLUSH should not be changed to ensure that the ADC does not lock up again.

```
# Check if GPADC is in stuck state
If GPADC_STUCK.STUCK != 0 Then            # I2C address 0x59, register 0xC7

# Make sure GPADC AUTO mode is disabled
# This is only required if any GPADC auto conversion might be enabled
    GPADC_AUTO_CTRL = 0x00                # I2C address 0x59, register 0xC3

    # Enable flush operation
    GPADC_FLUSH_EN = 0x80                 # I2C address 0x59, register 0xC5

    # Start the flush operation
    GPADC_FLUSH = 0x01                    # I2C address 0x59, register 0xC2

    # Wait for flush operation to complete
    while GPADC_STUCK.STUCK != 0 then     # I2C address 0x59, register 0xC7
        sleep
    end while

    # Reset flush bits
    GPADC_FLUSH_EN = 0x00                 # I2C address 0x59, register 0xC5
    GPADC_FLUSH = 0x00                    # I2C address 0x59, register 0xC2
end if
```

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (May 2015) to A Revision	Page
• Added the TPS65919-Q1 device to the document	2
• Deleted ECCN number and *Export Control Notice* from document	2

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265