

# Using the Analog Sensing for DC Motors (ASDC) eTPU Function

Covers the MCF523x, MPC5500, and all eTPU-equipped Devices

by: Milan Brejl, Michal Princ  
System Application Engineers, Roznov Czech System Center  
Valeriy Phillipov  
System Application Engineer, Kiev Embedded Software Lab

## 1 Introduction

The analog sensing for DC motors (ASDC) enhanced time processor unit (eTPU) function is one of the functions included in the DC motor control eTPU function set (set3). This application note is intended to provide simple C interface routines to the ASDC eTPU function. The routines are targeted at the MCF523x and MPC5500 families devices, but they could be easily used with any device that has an eTPU.

## 2 Function Overview

The ASDC function is useful for preprocessing analog values that are measured by an AD converter and transferred to the eTPU data memory by DMA transfer. The ASDC function is also useful for triggering the AD converter and synchronizing other eTPU functions.

### Table of Contents

1	Introduction.....	1
2	Function Overview.....	1
3	Function Description.....	2
4	C Level API for Function.....	4
5	Example Use of Function .....	11
6	Summary and Conclusions .....	12

### 3 Function Description

ASDC function performs the following operations:

- **Get values from an AD converter result queue**

Up to two values from the queue can be processed. The converted values are read from the queue as 16-bit words at specified queue address offsets.

- **Perform bit alignment**

Bit alignment is performed by shifting the value from the result queue left by 8, 10, 12 or 16 bits, in order to get a 24-bit fractional value.

- **Remove DC offset**

DC offsets are removed from the measured samples. The DC offsets can be set manually or measured.

- **Filter the measured values**

The measured values can be filtered using an exponentially-weighted moving average (EWMA) filter. The EWMA filter is defined by the following equation:

$$y(n) = forget\_factor \cdot y(n - 1) + (1 - forget\_factor) \cdot x(n)$$

where

$x(n)$  is the  $n$ -th step filter input,

$y(n)$  is the  $n$ -th step filter output, and

$forget\_factor$  is the only filter parameter - forgetting factor. It is a value between 0 and 1, usually close to 1.

- **Optionally switch the sign of measured value**

This option enables to change the sign of measured samples after filtering based on the sign of another eTPU function parameter. For example, when the current is measured on DC-bus, the current polarity, in motoring motor mode (quadrant 1 and 3), is positive in both directions. A sign of the actual motor speed, calculated by speed controller, can be applied to the measured current in order to control the motor in all 4 quadrants.

- **Trigger the AD convertor by the generated signal**

The first edge of the generated pulse (the low-high edge on [Figure 1](#)) triggers the AD converter. The pulse width is adjusted so that the analog signals are sampled, converted to digital values, and transferred to the eTPU data memory by the DMA transfer, before the second edge of the pulse (the high-low edge on [Figure 1](#)) is raised. On the second edge, the ASDC function executes the processing of the just measured value(s).

The generated signal polarity is selectable. The position of the triggering edge relative to PWM period edge times is adjustable.

On MPC5500, the ASDC function can be assigned to one of 5 eTPU channels (channels 26 to 31) in order to activate one of 5 eQADC triggers internally.

- **Generate DMA request**

A DMA request is generated on the first edge of the generated signal.

- **Synchronize processing of other eTPU functions**

The ASDC function can execute processing of two other eTPU functions via an eTPU link. Link to one of the functions (inner-loop controller) is executed just after the ASDC process of the

measured values (on the second edge). Link to the second of the functions (outer-loop controller) is executed on the first edge of the generated pulse, and only once per a defined number of periods.

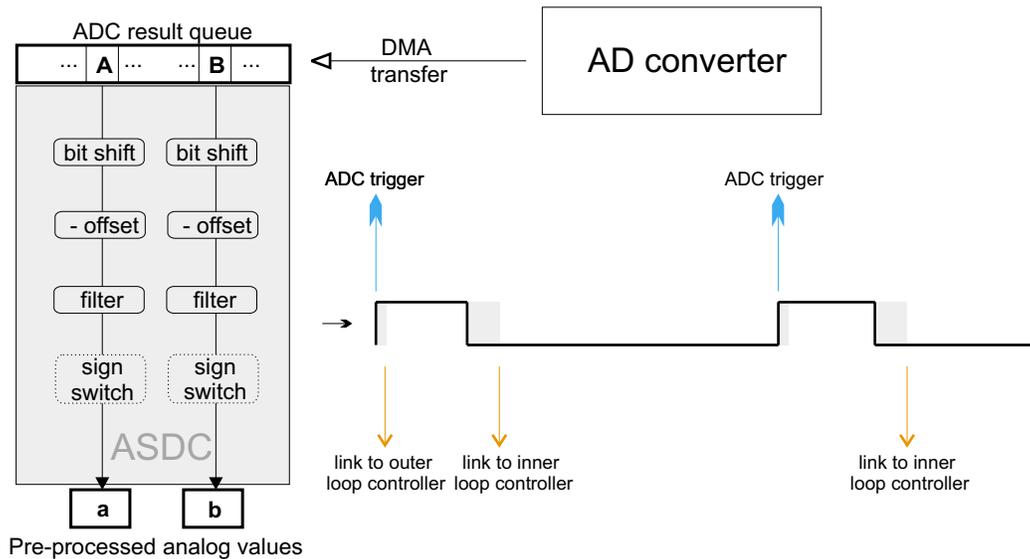


Figure 1. ASDC Processing Overview

### 3.1 Modes of Operation

The ASDC function can operate in one of the following modes:

- Periodic Mode**  
 In this mode, the ASDC function generates the pulses that trigger the AD converter periodically in a defined period.
- Synchronized Mode**  
 This mode is useful when the AD triggering and other ASDC function processing must be synchronized with the PWM signals generated by Motor-Control PWM eTPU functions (PWMMDC, PWMC, PWMF). Even when the PWM periods are changed during the run, the ASDC function generates the pulses that trigger the AD converter synchronously with the PWM period. The first edge of the pulse is generated, in an adjustable time, before or after the PWM edge-time.

### 3.2 Interrupts

The ASDC function periodically generates an interrupt service request to the CPU on the first edge.

### 3.3 Performance

Like all eTPU functions, the ASDC function performance in an application is to some extent dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the scheduler.

## C Level API for Function

The influence of the ASDC function on the overall eTPU performance can be expressed by the following parameters:

- **Maximum processing time on the first edge**
- **Maximum processing time on the second edge**
- **Maximum eTPU busy time per one period**

This value is a sum of processing time on the first and on the second edge. This value, compared to the period value, determines the proportional load of the eTPU engine caused by the ASDC function.

Table 1 lists the above mentioned ASDC performance limits.

**Table 1. ASDC Performance Limits**

Number of Samples	MAximum Processing Time on the Fist Edge [eTPU cycles]	Maximum procEssing Time on the Second Edge [eTPU cycles]	Maximum eTPU Busy Time per One Period [eTPU cycles]
One sample is processed	42	80	122
Two samples are processed	42	142	184

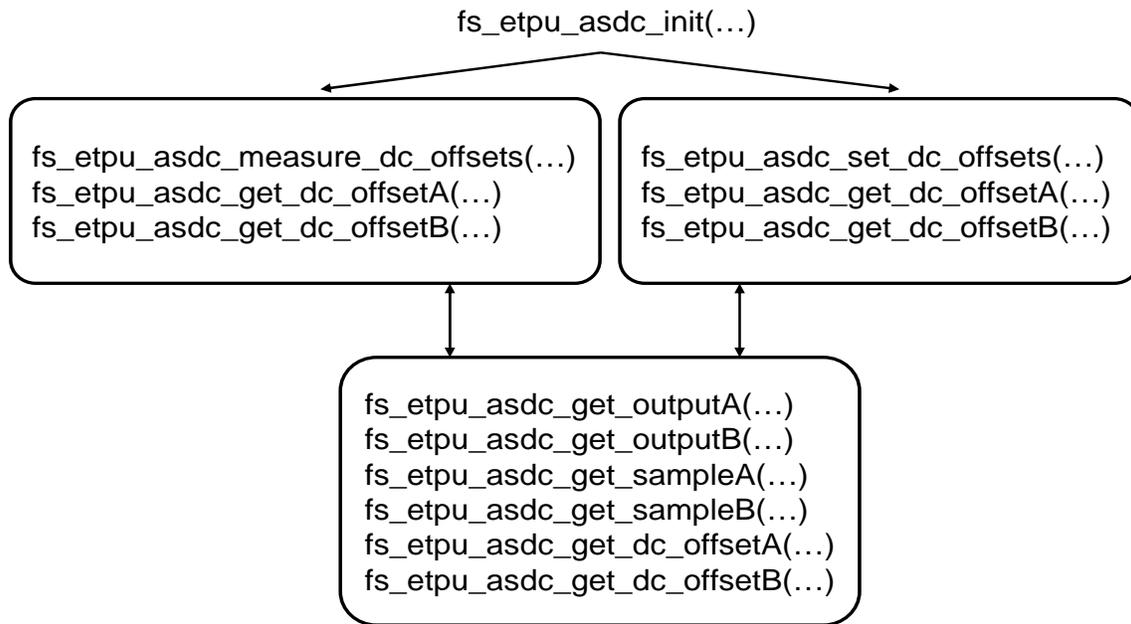
The eTPU module clock is equal to the CPU clock on MPC5500 devices. It is also equal to the peripheral clock, which is a half of the CPU clock, on MCF523x devices. For example, the eTPU module clock is 132 MHz on a 132-MHz MPC5554, and one eTPU cycle takes 7.58ns; it is only 75 MHz on a 150-MHz MCF5235, and one eTPU cycle takes 13.33ns.

The performance is influenced by compiler efficiency. The above numbers, that were measured on the code compiled by eTPU compiler version 1.0.0.5, are given for guidance only and are subject to change. For up to date information, refer to the information provided in the particular eTPU function set release available from Freescale.

## 4 C Level API for Function

The following routines provide easy access, for the application developer, to the ASDC function. Use of these functions eliminates the need to directly control the eTPU registers. There are 9 functions added to the application programming interface (API). The routines can be found in the `etpu_asdc.h` and `etpu_asdc.c` files, which should be included in the link file along with the top level development file(s).

Figure 2 shows the ASDC API state flow and lists API functions which can be used in each of its states.



**Figure 2. ASDC API State Flow**

All ASDC API routines will be described in order and are listed below:

- Initialization Function:

```
int32_t fs_etpu_asdc_init( uint8_t  channel,
                          uint8_t  priority,
                          uint8_t  polarity,
                          uint8_t  mode,
                          uint24_t  period,
                          uint24_t  start_offset,
                          int24_t   egde_offset,
                          uint24_t  measure_time,
                          uint8_t   periods_per_outerloop,
                          uint8_t   SC_BC_outerloop_chan,
                          uint8_t   CC_innerloop_chan,
                          uint8_t   measure_samples_mask,
                          uint32_t * result_queue,
                          uint8_t   bit_shift,
                          uint8_t   queue_offset_a,
```

```
uint8_t    queue_offset_b,
fract24_t  forget_factor_a,
fract24_t  forget_factor_b,
uint8_t    PWMMDC_chan,
uint8_t    outputA_chan,
uint16_t   outputA_offset,
uint8_t    outputB_chan,
uint16_t   outputB_offset,
uint8_t    sign_switch,
uint8_t    signA_chan,
uint16_t   signA_offset,
uint8_t    signB_chan,
uint16_t   signB_offset)
```

- Change Operation Functions:

```
int32_t fs_etpu_asdc_measure_dc_offsets(uint8_t channel,
                                         int8_t measure_dc_offsets_mask)
int32_t fs_etpu_asdc_set_dc_offsets(uint8_t channel,
                                     ufract24_t dc_offset_a,
                                     ufract24_t dc_offset_b)
```

- Value Return Functions

```
fract24_t fs_etpu_asdc_get_outputA(uint8_t channel)
fract24_t fs_etpu_asdc_get_outputB(uint8_t channel)
fract24_t fs_etpu_asdc_get_sampleA(uint8_t channel)
fract24_t fs_etpu_asdc_get_sampleB(uint8_t channel)
ufract24_t fs_etpu_asdc_get_dc_offsetA(uint8_t channel)
ufract24_t fs_etpu_asdc_get_dc_offsetB(uint8_t channel)
```

## 4.1 Initialization Function

### 4.1.1 int32\_t fs\_etpu\_asdc\_init(...)

This routine is used to initialize the eTPU channel for the ASDC function. It has the following parameters:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter should be assigned a value of 0-31 for ETPU\_A, and 64-95 for ETPU\_B.

- **priority (uint8\_t)** - This is the priority to assign to the ASDC function. This parameter should be assigned a value of:
  - FS\_ETPU\_PRIORITY\_HIGH,
  - FS\_ETPU\_PRIORITY\_MIDDLE,
  - FS\_ETPU\_PRIORITY\_LOW or
  - FS\_ETPU\_PRIORITY\_DISABLED
- **polarity (uint8\_t)** - This is the generated pulse polarity. This parameter should be assigned a value of:
  - FS\_ETPU\_ASDC\_PULSE\_HIGH or
  - FS\_ETPU\_ASDC\_PULSE\_LOW
- **mode (uint8\_t)** - Mode configuration parameter. This parameter should be assigned a value of:
  - FS\_ETPU\_ASDC\_MODE\_PERIODIC or
  - FS\_ETPU\_ASDC\_MODE\_SYNC
- **period (uint24\_t)** - This is the ASDC period, as a number of TCR1 clocks. This parameter applies in the periodic mode only (mode=FS\_ETPU\_ASDC\_MODE\_PERIODIC).
- **start\_offset (uint24\_t)** - This parameter is used to synchronize various eTPU functions that generate a signal. For ASDC, the first pulse starts the *start\_offset* TCR1 clocks after initialization. This parameter applies in the periodic mode only (mode=FS\_ETPU\_ASDC\_MODE\_PERIODIC).
- **edge\_offset (int24\_t)** – Offset, either positive or negative, between PWM period edge\_times and the ASDC pulse first edge as number of TCR1 clocks. This parameter applies in the synchronized mode only (mode = FS\_ETPU\_ASDC\_MODE\_SYNC).
- **measure\_time (uint24\_t)** – Time from the first (triggering) edge to the second edge, at which the result queue is supposed to be ready in the DATA\_RAM (in TCR1 clocks).
- **periods\_per\_outerloop (uint8\_t)** – A link service request is generated to the *outerloop\_chan* each *periods\_per\_outerloop* period.
- **SC\_BC\_outerloop\_chan (uint8\_t)** – Number of a channel on which an outer-loop controller (in slave mode) runs. In order to not activate any inner-loop controller, set the number to a channel with priority disabled. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **CC\_innerloop\_chan (uint8\_t)** – Number of the channel on which an inner-loop controller (in slave mode) runs. In order to not activate any inner-loop controller, set the number to a channel with priority disabled. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **measure\_samples\_mask (uint8\_t)** - This parameter defines which measured samples are processed by ASDC function. The sample\_A is processed each time. This parameter should be assigned a values of:
  - FS\_ETPU\_ASDC\_SAMPLE\_A or
  - FS\_ETPU\_ASDC\_SAMPLE\_A\_B.

## C Level API for Function

- **result\_queue (uint32\_t \*)** – Pointer to the result queue. Result queue is an array of 16-bit words that contains the measured values.
- **bit\_shift (uint8\_t)** – This parameter defines how to align data from the result queue into fract24 (or int24). This parameter should be assigned a value of:
  - FS\_ETPU\_ASDC\_SHIFT\_LEFT\_BY\_8,
  - FS\_ETPU\_ASDC\_SHIFT\_LEFT\_BY\_10,
  - FS\_ETPU\_ASDC\_SHIFT\_LEFT\_BY\_12 or
  - FS\_ETPU\_ASDC\_SHIFT\_LEFT\_BY\_16
- **queue\_offset\_a (uint8\_t)** – Sample A position in the result queue - offset in bytes.
- **queue\_offset\_b (uint8\_t)** – Sample B position in the result queue - offset in bytes.
- **forget\_factor\_a (fract24\_t)** – EWMA filter forgetting factor applied to sample A. This parameter must be a value between 0 (0x000000) and 1 (0x7FFFFFFF), usually close to 1, assigned in fract24.
- **forget\_factor\_b (fract24\_t)** – EWMA filter forgetting factor applied to sample B. This parameter must be a value between 0 (0x000000) and 1 (0x7FFFFFFF), usually close to 1, assigned in fract24.
- **PWMMD\_Chan (uint8\_t)** – PWMMD\_C channel number from which the PWM edge\_times are taken. This parameter applies in the synchronized mode only (mode=FS\_ETPU\_ASDC\_MODE\_SYNC).
- **outputA\_chan (uint8\_t)** – ASDC writes processed sample\_A to a recipient function input parameter. This is the output\_A recipient function channel number. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **outputA\_offset (uint16\_t)** – ASDC writes processed sample\_A to a recipient function input parameter. This is the output\_A recipient function parameter offset. Function parameter offsets are defined in etpu\_<func>\_auto.h file.
- **outputB\_chan (uint8\_t)** – ASDC writes processed sample\_B to a recipient function input parameter. This is the output\_B recipient function channel number. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **outputB\_offset (uint16\_t)** – ASDC writes processed sample\_B to a recipient function input parameter. This is the output\_B recipient function parameter offset. Function parameter offsets are defined in etpu\_<func>\_auto.h file.
- **sign\_switch (uint8\_t)** – This option enables to change the sign of measured samples based on the sign of another eTPU function parameter. For example, when a motor current is measured on DC-bus, the current polarity, in motoring motor mode (quadrant 1 and 3), is positive in both directions. This parameter should be assigned a value of:
  - FS\_ETPU\_ASDC\_SIGN\_NO\_SAMPLE - sign of no sample is changed, or
  - FS\_ETPU\_ASDC\_SIGN\_SAMPLE\_A - sign of sample A is changed, or
  - FS\_ETPU\_ASDC\_SIGN\_SAMPLE\_B - sign of sample B is changed, or
  - FS\_ETPU\_ASDC\_SIGN\_SAMPLE\_A\_B - signs of both samples A and B are changed.

- **signA\_chan (uint8\_t)** – This is the number of the channel that includes eTPU parameter determining sign of measured sample A. This parameter applies only if `sign_switch=FS_ETPU_ASDC_SIGN_SAMPLE_A` or `sign_switch=FS_ETPU_ASDC_SIGN_SAMPLE_A_B`. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **signA\_offset (uint16\_t)** – This parameter defines offset of eTPU parameter determining sign of measured sample A in scope of `signA_chan` function parameters. Function parameter offsets are defined in `etpu_<func>_auto.h` file.
- **signB\_chan (uint8\_t)** – This is the number of the channel that includes eTPU parameter determining sign of measured sample B. This parameter applies only if `sign_switch=FS_ETPU_ASDC_SIGN_SAMPLE_B` or `sign_switch=FS_ETPU_ASDC_SIGN_SAMPLE_A_B`. This parameter should be assigned a value of 0-31 for ETPU\_A and of 64-95 for ETPU\_B.
- **signB\_offset (uint16\_t)** – This parameter defines offset of eTPU parameter determining sign of measured sample B in scope of `signB_chan` function parameters. Function parameter offsets are defined in `etpu_<func>_auto.h` file.

## 4.2 Change Operation Functions

### 4.2.1 `int32_t fs_etpu_asdc_measure_dc_offsets(uint8_t channel, int8_t measure_dc_offsets_mask)`

This function enables setting the *dc\_offsets* by measurement. Ensure the power-stage is in a stand-by mode and call this function. The values measured represent the DC offsets, and are stored as *dc\_offsets* parameters. Later the *dc\_offsets* are used to remove the DC-offsets from measured values. This function has the following parameters:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.
- **measure\_dc\_offsets\_mask (int8\_t)** – This parameter determines which samples can be used as *dc\_offsets*. This parameter should be assigned a value of:
  - `FS_ETPU_ASDC_DC_OFFSET_SAMPLE_A` or
  - `FS_ETPU_ASDC_DC_OFFSET_SAMPLE_B` or
  - `FS_ETPU_ASDC_DC_OFFSET_SAMPLE_A_B`.

This function returns 0 if the DC offsets were successfully measured. In case the ASDC channel has any pending HSRs the DC offsets are not set and this function should be called again later. In this case a sum of pending HSR numbers is returned.

## 4.2.2 `int32_t fs_etpu_asdc_set_dc_offsets(uint8_t channel, ufract24_t dc_offset_a, ufract24_t dc_offset_b)`

This function enables setting the *dc\_offsets* by values. It has the following parameters:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.
- **dc\_offset\_a (ufract24\_t)** – This parameter is the DC offset to remove from sample A. The DC offset must be in the same format as the sample - after the bit alignment.
- **dc\_offset\_b (ufract24\_t)** – This parameter is the DC offset to remove from sample B. The DC offset must be in the same format as the sample - after the bit alignment.

## 4.3 Value Return Functions

### 4.3.1 `fract24_t fs_etpu_asdc_get_outputA(uint8_t channel)`

This function returns outputA value - after filtration. It has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

### 4.3.2 `fract24_t fs_etpu_asdc_get_outputB(uint8_t channel)`

This function returns outputB value - after filtration. It has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

### 4.3.3 `fract24_t fs_etpu_asdc_get_sampleA(uint8_t channel)`

This function returns outputA value - prior to filtration. It has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

### 4.3.4 `fract24_t fs_etpu_asdc_get_sampleB(uint8_t channel)`

This function returns outputB value - prior to filtration. It has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

### 4.3.5 `ufract24_t fs_etpu_asdc_get_dc_offsetA(uint8_t channel)`

This function returns sample A DC offset. It has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

### 4.3.6 `ufract24_t fs_etpu_asdc_get_dc_offsetB(uint8_t channel)`

This function returns sample B DC offset. This function has the following parameter:

- **channel (uint8\_t)** - This is the ASDC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.

## 5 Example Use of Function

### 5.1 Demo Applications

The usage of the ASDC eTPU function is demonstrated in the following two applications:

- "BLDC Motor with Speed Closed Loop and Break Controller, driven by eTPU on MCF523x," AN2954.
- "DC Motor with Speed and Current Loop, driven by eTPU on MCF523x," AN2955.

For a detailed description of the demo applications refer to the mentioned application notes.

#### 5.1.1 Function Calls

Example of ASDC initialization and using of ASDC API functions are stated in the following lines. It deals with a pieces of code used in the demo application "DC Motor with Speed and Current loop". The ASDC function is initialized in PWM synchronized mode with the pulse low polarity (high-low edge triggers the A/D converter). Only one sample is measured and processed with sign switch of output value based on Speed Controller function OMEGA ACTUAL parameter. The ASDC function synchronizes processing of Speed Controller and Current Controller eTPU functions.

```
err_code = fs_etpu_asdc_init (ASDC_CHANNEL, /* engine: A; channel: 14 */
FS_ETPU_PRIORITY_HIGH, /* priority: High */
FS_ETPU_ASDC_PULSE_LOW, /* polarity: Pulse low, high-low edge triggers */
FS_ETPU_ASDC_MODE_SYNC, /* mode: PWM synchronized mode */
1000, /* period: 1000 TCR */
5000, /* start_offset: 5000 TCR */
etpu_a_tcr1_freq/16000/4, /* egde_offset: 1/4 of PWM period */
400, /* measure_time: 400 TCR */
8, /* periods_per_outerloop: 8 */
SC0_CHANNEL, /* SC_BC_outerloop_chan: SC0_CHANNEL */
CC0_CHANNEL, /* CC_innerloop_chan: CC0_CHANNEL */
FS_ETPU_ASDC_SAMPLE_A, /* measure_samples_mask */
p_result_queue, /* pointer to result queue */
```

## Summary and Conclusions

```

FS_ETPU_ASDC_SHIFT_LEFT_BY_12, /* bit_shift */
2,      /* queue_offset_a */
0,      /* queue_offset_b */
0x547AE0, /* forget_factor_a: 0.66 */
0,      /* forget_factor_b: 0 */
PWMMDC0_MASTER, /* PWMM_chan: PWMMDC0_MASTER */
CC0_CHANNEL,    /* outputA_chan: CC0_CHANNEL*/
FS_ETPU_CC_IMEASURED_OFFSET, /* outputA_offset:FS_ETPU_CC_IMEASURED_OFFSET */
ETPU_CHAN_NOT_USED, /* outputB_chan */
0,              /* outputB_offset */
FS_ETPU_ASDC_SIGN_SAMPLE_A, /* sign_switch: sign_switch of sample A */
SC0_CHANNEL, /* signA_chan: SC0_CHANNEL */
FS_ETPU_SC_OMEGAACTUAL_OFFSET, /* signA_offset: FS_ETPU_SC_OMEGAACTUAL_OFFSET */
ETPU_CHAN_NOT_USED, /* signB_chan: ETPU_CHAN_NOT_USED */
0); /* signB_offset: 0 */

/* measure DC offset */
fs_etpu_asdc_measure_dc_offsets( ASDC_CHANNEL, FS_ETPU_ASDC_DC_OFFSET_SAMPLE_A);

/* read preprocessed sample A */
outputA = fs_etpu_asdc_get_outputA( ASDC_CHANNEL);

/* read sample A prior to filtration */
sampleA = fs_etpu_asdc_get_sampleA( ASDC_CHANNEL);

/* read sampleA DC offset */
dc_offsetA = fs_etpu_asdc_get_dc_offsetA( ASDC_CHANNEL);

```

## 6 Summary and Conclusions

This eTPU application note provides the user with a description of the analog sensing for DC motors eTPU function usage and examples. The simple C interface routines to the ASDC eTPU function enable easy implementation of this function in applications. The demo application is targeted at the MCF523x family of devices, but it could be easily reused with any device that has an eTPU.

**References:**

1. “The Essential of Enhanced Time Processing Unit,” AN2353.
2. “General C Functions for the eTPU,” AN2864.
3. “Using the DC Motor Control eTPU Function Set (set3),” AN2958.
4. Enhanced Time Processing Unit Reference Manual, ETPURM/D.
5. eTPU Graphical Configuration Tool, <http://www.freescale.com/etpu>, ETPUGCT.
6. “BLDC Motor with Speed Closed Loop and Break Controller, driven by eTPU on MCF523x,” AN2954.
7. “DC Motor with Speed and Current Loop, driven by eTPU on MCF523x,” AN2955.

**THIS PAGE INTENTIONALLY LEFT BLANK**

**THIS PAGE INTENTIONALLY LEFT BLANK**

## How to Reach Us:

### Home Page:

[www.freescale.com](http://www.freescale.com)

### E-mail:

[support@freescale.com](mailto:support@freescale.com)

### USA/Europe or Locations Not Listed:

Freescale Semiconductor  
Technical Information Center, CH370  
1300 N. Alma School Road  
Chandler, Arizona 85224  
+1-800-521-6274 or +1-480-768-2130  
[support@freescale.com](mailto:support@freescale.com)

### 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)  
[support@freescale.com](mailto:support@freescale.com)

### 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](mailto:support.japan@freescale.com)

### Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.  
Technical Information Center  
2 Dai King Street  
Tai Po Industrial Estate  
Tai Po, N.T., Hong Kong  
+800 2666 8080  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

### For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center  
P.O. Box 5405  
Denver, Colorado 80217  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2005. All rights reserved.

Document Number: AN2846  
Rev. 0  
04/2005