Freescale Semiconductor
Application Note

AN2843
Rev. 2, 02/2012

Using the Speed Controller (SC) eTPU Function

Covers the MCF523*x*, MPC5500, MPC5600 and all eTPU-equipped devices

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

Table of Contents

1 Introduction

The Speed Controller (SC) Enhanced Time Processor Unit (eTPU) function is one of the functions in the motor-control set of the eTPU functions (set3 and set4). This application note is intended to provide simple C interface routines to the SC eTPU function. The routines are targeted at the MCF523*x,* MPC5500, and MPC5600 families of devices, but they can easily be used with any device that has an eTPU.

2 Function Overview

The SC function is not intended to process an input or output signal. The purpose of the SC function is to control another eTPU function’s input parameter. The SC function includes a general controller algorithm. The controller calculates its output based on two inputs: a measured value and a desired value. The measured value —the actual motor speed—is calculated based on inputs provided by the HD, QD or RSLV function. The desired value is an output of a speed ramp, whose input is a SC function parameter, and can be provided by the CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the speed outer-loop.

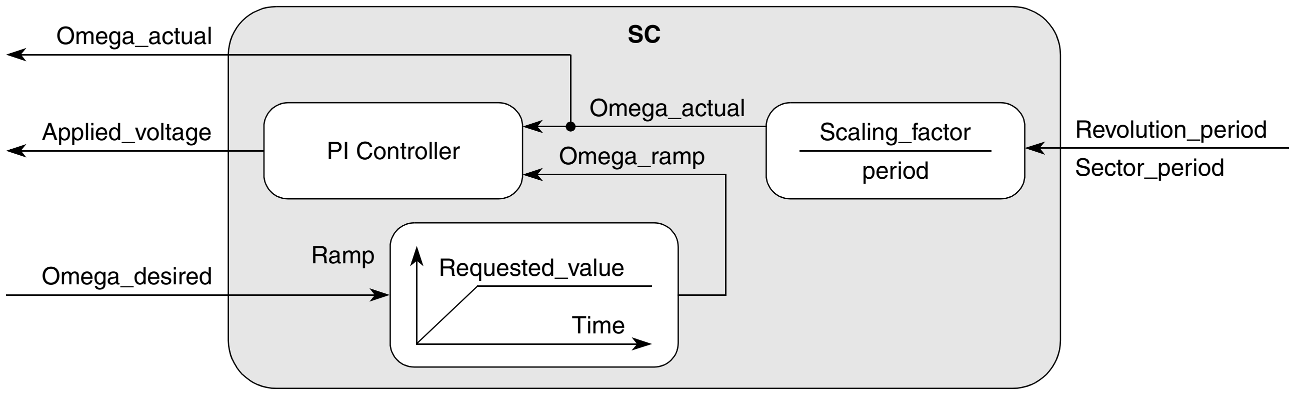

Figure 1. Functionality of SC

The controller algorithm is a general Proportional-Integral-Derivative (PID) algorithm. It can be configured as the following:

- P controller
- PI controller
- PD controller
- PID controller

3 Function Description

The SC performs the calculation of the actual angular motor speed. If the Quadrature Decoder (QD) function is used to capture the shaft encoder signals, the speed controller periodically reads the QD parameters “Position Counter” and “TCR Value of The Last Transition,” and uses them to calculate the actual speed. If the Hall Decoder (HD) function is used to capture the Hall sensor signals, the speed controller periodically reads one of the HD parameters, either the “Revolution Period” or “Commutation Period,” and uses it to calculate the actual speed. If the Resolver (RSLV) function is used to calculate rotor position, the speed is read from the Resolver internal Angle Tracking Observer algorithm.

The desired input is optionally passed through a ramp, in order to refine the step changes towards the desired value. The ramp performs a linear Ramp Generation algorithm (see Figure 2). If the *desired_value* is greater than the *actual_value*, the ramp output steps up by *ramp_increment_up* until the *desired_value* is reached, at which point the desired_value is returned. And vice versa: if the *desired_value* is less than *actual_value*, the ramp output steps down by *ramp_increment_down* until the *desired_value* is reached.

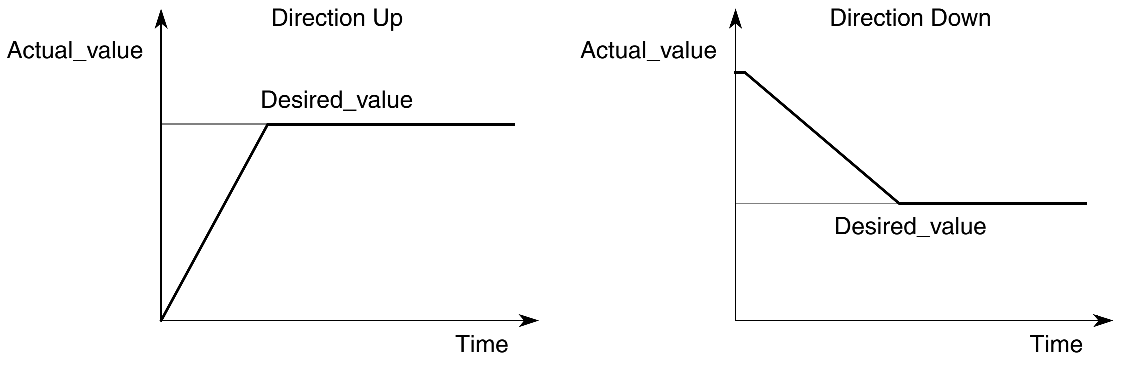

Figure 2. Ramp Generation Algorithm

The controller algorithm included in the SC function calculates the output according to the following equations:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$
$$e(k) = w(k) - m(k)$$
$$u_P(k) = G_P * e(k)$$
$$u_I(k) = u_I(k-1) + G_I * e(k)$$
$$u_D(k) = G_D * (e(k) - e(k-1))$$

Where:

$u(k)$	– PID algorithm output (the output of SC) in step *k*.
$u_P(k)$	– Proportional portion in step *k*.
$u_I(k)$	– Integral portion in step *k*.
$u_D(k)$	– Derivative portion in step *k*.
$e(k)$	– Input error in step *k*.
$w(k)$	– Desired value in step *k*.
$m(k)$	– Measured value in step *k*.
G_P	– Proportional gain.
G_I	– Integral gain.
G_D	– Derivative gain.

During SC initialization, a user can set what portion will be calculated and included in the controller output.

The gains are applied with 16-bit precision. The measured and desired values are applied with 24-bit precision. The Integral portion is stored with 14-bit precision.

Like most of the motor-control eTPU functions, the SC function also supports checking the eTPU latencies using an oscilloscope. The SC function channel, if connected to an output pin, turns the output pin high and low, so that the high-time identifies the period of time in which the SC update is executed.

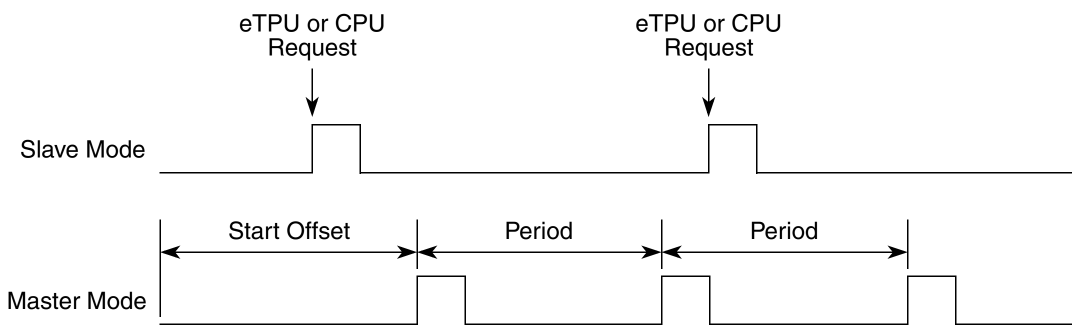

Figure 3. Master Mode and Slave Mode Illustration

The SC function update, in which the actual desired value and the measured value are taken and the control signal is adjusted, can be executed periodically, or by another process:

- **Master Mode**

  The SC update is executed periodically with a given period.
- **Slave Mode**

  The SC update is executed by the Analog Sensing (ASDC or ASAC) eTPU function, other eTPU function, or by the CPU.

4 C Level API for Function

The following routines provide easy access to the SC function for the application developer. Use of these functions eliminate the need to directly control the eTPU registers. There are 13 functions added to the application programming interface (API). The routines can be found in the `etpu_sc.h` and `etpu_sc.c` files, which should be included in the link file along with the top level development file(s). These routines will be described in order and are listed below:

- Initialization Function:

```
int32_t fs_etpu_sc_init( uint8_t channel,
                         uint8_t priority,
                         uint8_t mode,
                         uint8_t decoder_type,
                         uint8_t configuration,
                        uint24_t period,
                        uint24_t start_offset,
                        uint24_t services_per_irq,
                slc_pid_params_t* p_pid_params,
               sc_ramp_params_t* p_ramp_params,
                         uint8_t HD_QD_RSLV_chan,
                         uint8_t output_chan,
                        uint16_t output_offset,
                         uint8_t link_chan,
                        uint24_t omega_max_rpm,
                        uint24_t omega_min_rpm,
                         uint8_t pole_pairs,
                        uint32_t HD_QD_etpu_tcr_freq,
                        uint32_t rslv_freq,
                        uint24_t qd_pc_per_rev )
```

- Change Operation Functions:

```
int32_t fs_etpu_sc_update(uint8_t channel)
int32_t fs_etpu_sc_set_configuration(uint8_t channel,
                                     uint8_t configuration)
int32_t fs_etpu_sc_set_pid_params(uint8_t channel,
                          s_pid_params_t* p_pid_params)
int32_t fs_etpu_sc_set_ramp_params(uint8_t channel,
                          s_ramp_params_t* p_ramp_params)
int32_t fs_etpu_sc_set_omega_desired(uint8_t channel,
                                   fract24_t omega_desired)
int32_t fs_etpu_sc_reset_integral_portion(uint8_t channel)
```

- Value Return Functions:

```
fract24_t fs_etpu_sc_get_omega_actual( uint8_t channel)
uint8_t fs_etpu_sc_get_saturation_flag(uint8_t channel)
fract24_t fs_etpu_sc_get_ramp_output(uint8_t channel)
fract24_t fs_etpu_sc_get_output(uint8_t channel)
fract24_t fs_etpu_sc_get_error(uint8_t channel)
fract24_t fs_etpu_sc_get_integral_portion(uint8_t channel)
```

4.1 Initialization Function

4.1.1 int32_t fs_etpu_sc_init(...)

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

- **channel (uint8_t)** - This is the SC channel number. This parameter should be assigned a value of 0-31 forETPU_A, and 64-95 for ETPU_B.
- **priority (uint8_t)** - This is the priority to assign to the SC function. This parameter should be assigned a value of:
  - — FS_ETPU_PRIORITY_HIGH
  - — FS_ETPU_PRIORITY_MIDDLE
  - — FS_ETPU_PRIORITY_LOW
  - — FS_ETPU_PRIORITY_DISABLED
- **mode (uint8_t)** -This is the function mode. This parameter should be assigned a value of:
  - — FS_ETPU_SC_MASTER
  - — FS_ETPU_SC_SLAVE

- **decoder_type (uint8_t)** – This is the type of decoder (HD, QD or RSLV). This parameter should be assigned a value of:
  - `FS_ETPU_SC_QD` (Quadrature Decoder)
  - `FS_ETPU_SC_HD_REV_PERIOD` (any Hall Decoder - Revolution period is measured)
  - `FS_ETPU_SC_HD_SEC_PERIOD_1` (1-phase Hall Decoder - Sector period is measured)
  - `FS_ETPU_SC_HD_SEC_PERIOD_2` (2-phase Hall Decoder - Sector period is measured)
  - `FS_ETPU_SC_HD_SEC_PERIOD_3` (3-phase Hall Decoder - Sector period is measured)
  - `FS_ETPU_SC_HD_SEC_PERIOD_4` (4-phase Hall Decoder - Sector period is measured)
  - `FS_ETPU_SC_RSLV` (Resolver)
- **configuration (uint8_t)** – This is the required configuration of SC. This parameter should be assigned a value of:
  - `FS_ETPU_SC_RAMP_OFF_PID_OFF` (Ramp algorithm is disabled, PID controller is disabled)
  - `FS_ETPU_SC_RAMP_OFF_PID_ON` (Ramp algorithm is disabled, PID controller is enabled)
  - `FS_ETPU_SC_RAMP_ON_PID_OFF` (Ramp algorithm is disabled, PID controller is disabled)
  - `FS_ETPU_SC_RAMP_ON_PID_ON` (Ramp algorithm is enabled, PID controller is enabled)
- **period (uint24_t)** - This is the update period, as a number of TCR1 clocks. This parameter applies in master mode only (mode=FS_ETPU_SC_MASTER).
- **start_offset (uint24_t)** - This parameter is used to synchronize various eTPU functions that generate a signal. The first SC update starts *start_offset* TCR1 clocks after initialization. This parameter applies in master mode only (mode=FS_ETPU_SC_MASTER).
- **services_per_irq (uint24_t)** - This parameter defines the number of updates after which an interrupt service request is generated to the CPU. When set to 0, no interrupt service requests are generated.
- **p_pid_params (s_pid_params_t*)** – This is the pointer to a s_pid_params_t structure. The `s_pid_params_t` structure is defined in `etpu_sc.h`:

```
typedef struct {
   int32_t P_gain;
   int32_t I_gain;
   int32_t D_gain;
   int32_t positive_limit;
   int32_t negative_limit;
} s_pid_params_t;
```

  Where:
  - **P_gain (int32_t)** - This is the proportional gain and its value must be in the 9.15 format that means in the range of (-256, 256). To switch off proportional portion set this parameter to zero.
  - **I_gain (int32_t)** - This is the Integral gain and its value must be in the 9.15 format that means in the range of (-256, 256). To switch off calculation of integral portion set this parameter to zero.

  - **D_gain (int32_t)** - This is the derivative gain and its value must be in the 9.15 format that means in the range of (-256, 256). To switch off calculation of derivative portion set this parameter to zero.
  - **positive_limit (int32_t)** - This is the positive output limit and its value must be in the 9.15 format that means in the range of (-256, 256).
  - **negative_limit (int32_t)** - This is the negative output limit and its value must be in the 9.15 format that means in the range of (-256, 256).

- **p_ramp_params (s_ramp_params_t*)** – This is the pointer to a s_ramp_params_t structure. The s_ramp_params_t structure is defined in `etpu_sc.h`:

```
typedef struct  {
    fract24_t ramp_incr_up;
    fract24_t ramp_incr_down;
} s_ramp_params_t;
```

  Where:

  - **ramp_incr_up (fract24_t)** - This is the step increment up, and its value must be in the range (0, 1).
  - **ramp_incr_down (fract24_t)** - This is the step increment down, and its value must be in the range (0, 1).

- **HD_QD_RSLV_chan (uint8_t)** – This is the number of the channel the HD, QD or RSLV functions is assigned to. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **output_chan (uint8_t)** – SC writes the PID output to a recipient function input parameter. This is the recipient function channel number. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **output_offset (uint16_t)** – SC writes the PID output to a recipient function input parameter. This is the recipient function parameter offset. Function parameter offsets are defined in etpu_<func>_auto.h file..
- **link_chan (uint8_t)** – This is the channel number of a channel which receives a link after SC updates output. If SC updates PWM duty-cycles it should be a PWMMDC channel. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **omega_max_rpm (uint24_t)** – This is the maximum possible motor speed in rpm.
- **omega_min_rpm (uint24_t)** – This is the minimum possible motor speed in rpm.
- **pole_pairs (uint8_t)** – This is the number of motor pole-pairs.
- **HD_QD_etpu_tcr_freq (uint32_t)** – This is the frequency of TCR clock used by the HD or QD channel, in Hz. If Resolver is used (decoder_type=FS_ETPU_SC_RSLV) this parameter value does not apply.
- **rslv_freq (uint32_t)** – If Resolver is used (decoder_type=FS_ETPU_SC_RSLV), this is the resolver update frequency in Hz.

- **qd_pc_per_rev (uint24_t)** – This is the number of QD position counter increments per revolution. This parameter applies only if quadrature decoder is used (`decoder_type=FS_ETPU_SC_QD`).

4.2 Change Operation Functions

4.2.1 int32_t fs_etpu_sc_update(uint8_t channel)

This function executes the SC update. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

4.2.2 int32_t fs_etpu_sc_set_configuration(uint8_t channel, uint8_t configuration)

This function changes the SC configuration. This function has the following parameters:

- **channel (uint8_t)** - This is the speed controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.
- **configuration (uint8_t)** – This is the required configuration of SC. This parameter should be assigned a value of:
  - `FS_ETPU_SC_RAMP_OFF_PID_OFF` (Ramp algorithm is disabled, PID controller is disabled)
  - `FS_ETPU_SC_RAMP_OFF_PID_ON` (Ramp algorithm is disabled, PID controller is enabled)
  - `FS_ETPU_SC_RAMP_ON_PID_OFF` (Ramp algorithm is disabled, PID controller is disabled)
  - `FS_ETPU_SC_RAMP_ON_PID_ON` (Ramp algorithm is enabled, PID controller is enabled)

4.2.3 int32_t fs_etpu_sc_set_pid_params(uint8_t channel, s_pid_params_t* p_pid_params)

This function changes the PID parameter values. This function has the following parameters:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.
- **p_pid_params (s_pid_params_t*)** - This is the pointer to the PID control structure. The `s_pid_params_t` structure is defined in `etpu_sc.h`.

4.2.4 int32_t fs_etpu_sc_set_ramp_params(uint8_t channel, s_ramp_params_t* p_ramp_params)

This function changes the RAMP parameter values. This function has the following parameters:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.
- **p_ramp_params (s_ramp_params_t*)** - This is the pointer to the RAMP params structure. The `s_ramp_params_t` structure is defined in `etpu_sc.h`.

4.2.5 int32_t fs_etpu_sc_set_omega_desired(uint8_t channel, fract24_t omega_desired)

This function changes the desired value, as a portion of the maximum value. This function has the following parameters:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.
- **omega_desired (fract24_t)** - Desired input value. The value must be in the range MIN24 to MAX24. This value can be obtained by dividing the desired angular rotor speed in [rpm] by *omega_max_rpm*, which was used in initialization. The sign determines the rotor direction.

4.2.6 int32_t fs_etpu_sc_reset_integral_portion(uint8_t channel)

This function sets the integral portion to zero. This function has the following parameters

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

4.3 Value Return Function

4.3.1 fract24_t fs_etpu_sc_get_omega_actual(uint8_t channel)

This function gets the actual angular value, as a portion of the maximum value. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

The value of the actual rotor speed is returned as a fract24_t in the range (-1, 1). The actual angular rotor speed, in [rpm], can be obtained by multiplying of the returned value by *omega_max_rpm*, which was used in initialization. The sign determines the rotor direction.

4.3.2 uint8_t fs_etpu_sc_get_saturation_flag(uint8_t channel)

This function returns the PID controller saturation flags. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

The returned value can be one of the following:

- — 0 for no flag
- — 1 for positive saturation flag
- — 2 for negative saturation flag

4.3.3 fract24_t fs_etpu_sc_get_ramp_output(uint8_t channel)

This function returns the RAMP output. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

4.3.4 fract24_t fs_etpu_sc_get_output(uint8_t channel)

This function returns the PID controller output. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

4.3.5 fract24_t fs_etpu_sc_get_error(uint8_t channel)

This function returns the PID controller error. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

4.3.6 fract24_t fs_etpu_sc_get_integral_portion(uint8_t channel)

This function returns the PID controller integral portion. This function has the following parameter:

- **channel (uint8_t)** - This is the Speed Controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more SCs running simultaneously on the eTPU(s), the channel parameter distinguishes which SC function is accessed.

5 Summary and Conclusions

This application note provides the user with a description of the speed controller (SC) eTPU function. The simple C interface routines to the SC eTPU function enable easy implementation of the SC in applications.

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. "Using the AC Motor Control eTPU Function Set (set4)," AN2968
5. *Enhanced Time Processing Unit Reference Manual*, ETPURM
6. eTPU Graphical Configuration Tool, http://www.freescale.com/etpu, ETPUGCT
7. "3-Phase BLDC Motor Demo Application With Speed Closed Loop," AN2892.
8. "Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MPC5500," AN3206

6 Revision history

Table 1. Revision history

Revision number	Revision date	Description of changes
2	02 May 2012	Updated for support of motor drives with resolver position sensor.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN2843
Rev. 2
02/2012