Freescale Semiconductor
Application Note

Document Number: AN3943
Rev. 0, 10/2009

Using the Resolver Interface eTPU Function

Covers the MPC5500 and all eTPU-Equipped Devices

by: Andrzej Lara
System Application Engineer, Roznov Czech System Center

Contents

1 Introduction 1
2 Function Overview 1
3 Function Description 2
3.1 Resolver 2
3.2 Angle Tracking Observer 3
3.3 Resolver Driver eTPU Implementation 6
3.4 Performance 11
4 C Level API for the Function 12
4.1 API Functions Description 13
5 Example Use of Function 19
5.1 Data Synchronization Example 20
5.2 Timing Synchronization Example 23
5.3 Electrical and Mechanical Components 25
6 Summary and Conclusions 25
Appendix A 27

1 Introduction

The resolver driver (RSLV) enhanced time processor unit (eTPU) is a motor control utility aimed at providing a software interface to the resolver hardware.
The presented solution is based on the eTPU module. This application note describes briefly the theoretical background, details of implementation and C Level API, and presents two application examples. The presented utility is targeted at the MPC5500 family of devices but can be easily used with any device with an eTPU module.

2 Function Overview

The RSLV function provides the eTPU with the capability of tentatively processing the signals from the resolver hardware. The RSLV function running on the eTPU is able to calculate the position and speed of a connected resolver without requiring any computational load on the CPU.

For computation of position and speed, the RSLV function uses the Angle Tracking Observer algorithm. The Angle Tracking Observer algorithm is based on a closed-loop space-state system minimizing the difference between a computed angle and the angle read from the resolver signals.

The RSLV function is able to communicate asynchronously with the CPU and/or other eTPU functions through eDMA requests, CPU interrupts, and eTPU link requests.

The RSLV function is able to generate a simple left-aligned PWM wave, which can be used for synchronization with an ADC module. The function can be configured to execute sequential iterations of the Angle Tracking Observer periodically and synchronously with the generated PWM signal.

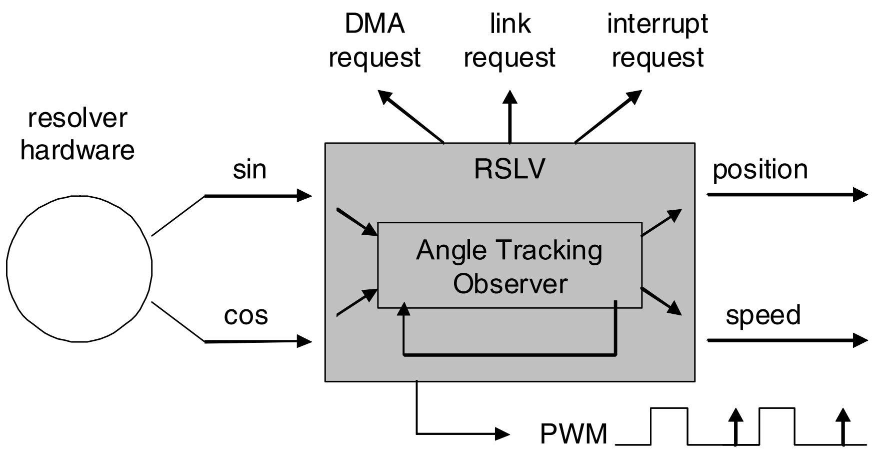


Figure 1 RSLV Function Overview

3 Function Description

The resolver driver is designed to work with the resolver hardware. This section starts with a brief description of a resolver and then moves to the presentation of the function’s operating principle.

3.1 Resolver

Resolver hardware can be viewed as two inductive position sensors, which, upon a supplied sinusoidal shaped signal on the input, generate two sinusoidal signals on the output. The output signals’ amplitudes depend on the position of the shaft. The amplitude of one signal is proportional to the sine. The amplitude of the other is proportional to the cosine of the shaft angle position.

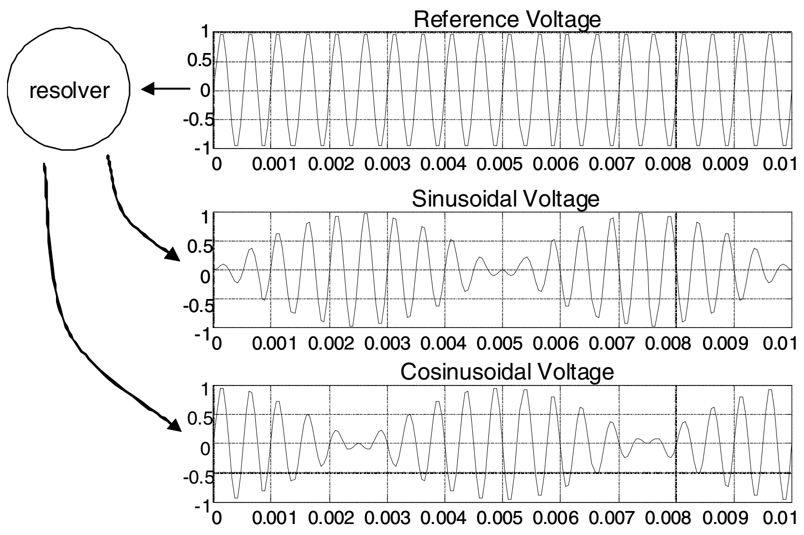


Figure 2 Resolver Operation Principle

The input signal to a resolver, called a resolver reference voltage, and the output signals can be connected according to a few simple equations:

$$U_{ref} = U_{amp} \times \sin\omega t \qquad \textbf{\textit{Eqn. 1}}$$

$$U_{sin} = K \times U_{ref} \times \sin\Theta \qquad \textbf{\textit{Eqn. 2}}$$

$$U_{cos} = K \times U_{ref} \times \cos\Theta \qquad \textbf{\textit{Eqn. 3}}$$

As the sine and cosine of the shaft angle position are provided, it can be computed directly by applying the inverse tangent function to the amplitudes of the resolver output voltages. The calculation needs to take into account the signs of the measured amplitudes in order to place the computed angle position correctly within a single 360 degree rotation:

$$\Theta = \mathrm{atan}(U_{sin}/U_{cos}) \qquad \textbf{\textit{Eqn. 4}}$$

A more detailed description of the resolver operation can be found in [6] and [1].

3.2 Angle Tracking Observer

3.2.1 Theoretical Background

By using the inverse tangent method for computing the angle, no information about the shaft angular velocity can be obtained. In order to overcome this disadvantage, another method, called the Angle Tracking Observer, can be applied. In the Angle Tracking Observer method, a closed-loop state system (state observer) replaces the simple inverse tangent function in order to compute the shaft angle and speed.

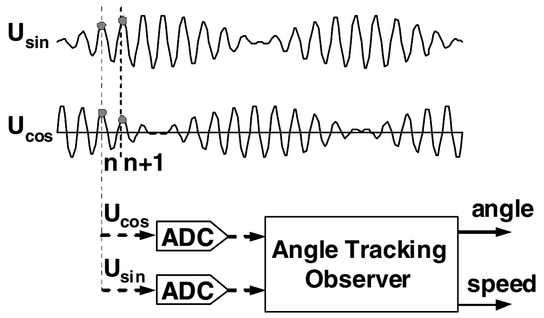


Figure 3 Use of the Angle Tracking Observer for Extracting the Angle and Speed

In the implementation of the Angle Tracking Observer, a simple state space system is created based on an integrator and a PI regulator connected in series and closed by a unit feedback loop. The error is computed as a sine of the difference of the resolver input angle and the computed output angle. Please note that the observer error is computed with use of the trigonometric identity of the sine of the difference of two angles: $\sin(\alpha - \beta) = \sin\alpha \times \cos\beta - \cos\alpha \times \sin\beta$.

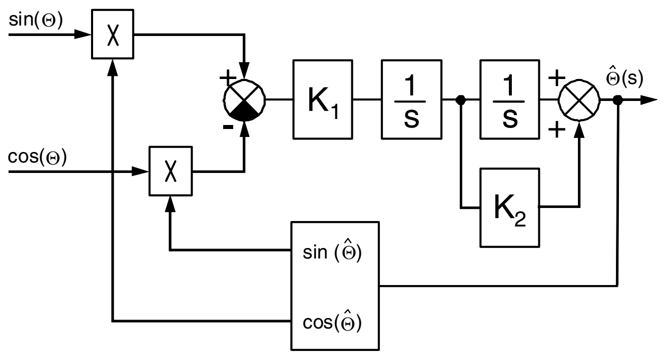


Figure 4 Block Diagram of the Angle Tracking Observer

After considering that for small error values the sine of the angle is approximately equal to the angle, one can derive the transfer function for the Angle Tracking Observer.

$$H(s) = \frac{\hat{\Theta}(s)}{\Theta(s)} = \frac{K_1 \times (1 + K_2 \times s)}{s^2 + K_1 \times K_2 \times s + K_1} \qquad \textbf{Eqn. 5}$$

The transfer function can be rewritten more conveniently as:

$$H(s) = \frac{\omega_n^2(1 + 2\zeta s/\omega_n)}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \textbf{Eqn. 6}$$

where: ω_n is the natural frequency $[\mathrm{rad\ s^{-1}}]$ and ζ is the damping factor $[-]$. The Angle Tracking Observer coefficients K_1 , K_2 can be calculated using the following expressions:

$$K_1 = \omega_n^2 \qquad \textbf{Eqn. 7}$$

$$K_2 = \frac{2\zeta}{\omega_n} \qquad \textbf{Eqn. 8}$$

3.2.2 Resolver Driver Implementation

For the implementation purposes, the Angle Tracking Observer analog model was transformed into digital representation with the use of the Forward Euler integration method. In the Forward Euler integration method, the analog integrator block is substituted by its digital equivalent:

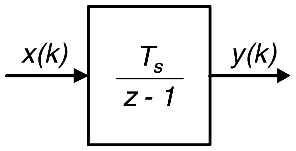


Figure 5 Forward Eurler Digital Integrator

In algebraic terms, the input and output of the Forward Euler integrator are connected by the equation:

$$y(k+1) = y(k) + x(k) \times T_s \qquad \textbf{Eqn. 9}$$

where T_s is the sampling time and k is the sample number.

The block diagram of the Angle Tracking Observer after transformation into the digital domain is shown here.

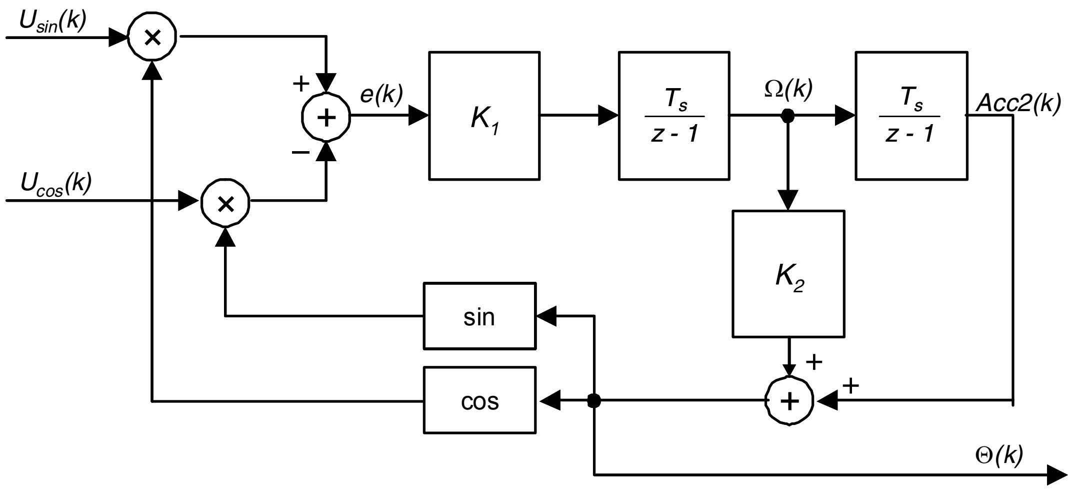


Figure 6 Block Diagram of the Angle Tracking Observer in the Digital Domain

For implementation purposes two additional operations need to be performed:

- The variables in the digital model need to be appropriately scaled to fit into the fractional range of –1 to 1.
- The digital integrators are replaced by accumulators, which perform only accumulation instead of multiplication by T_s and accumulation.

After performing the operations, the Angle Tracking Observer equations can be formulated as follows:

$$\begin{aligned}
\Omega_d(k+1) &= \Omega_d(k) + e(k) \times K_{1d} \\
Acc2_d(k+1) &= Acc2_d(k) + \Omega_d(k) \\
\Theta_d(k+1) &= K_{2d} \times \Omega_d(k+1) + Acc2_d(k+1) \\
e(k+1) &= U_{\sin}(k+1) \times \cos(\pi \times \Theta_d(k+1)) - (U_{\cos}(k+1) \times \sin(\pi \times \Theta_d(k+1)))
\end{aligned} \qquad \textbf{\textit{Eqn. 10}}$$

where the variables are defined as:

$$\begin{aligned}
\Theta_d(k+1) &= \frac{\Theta(k+1)}{\pi} \\
K_{1d} &= \frac{1}{\pi} \times T_s^2 \times \mathrm{K}_1 = \frac{1}{\pi} \times \omega_n^2 \times T_s^2 \\
K_{2d} &= \frac{\mathrm{K}_2}{T_s} = \frac{2 \times \zeta}{\omega_n \times T_s}
\end{aligned} \qquad \textbf{\textit{Eqn. 11}}$$

For the best accuracy in fractional arithmetic, the derived K_{1d} and K_{2d} coefficients need to be represented by two values: a scaling constant and a multiplication factor as follows:

$$\begin{aligned}
K_{1d} &= K1\_\mathrm{D} \times 2^{-K1\_\mathrm{SCALE}}, \quad K1\_\mathrm{D} \in (0.5,1) \\
K_{2d} &= K2\_\mathrm{D} \times 2^{K2\_\mathrm{SCALE}}, \quad K2\_\mathrm{D} \in (0.5,1)
\end{aligned} \qquad \textbf{\textit{Eqn. 12}}$$

All Angle Tracking Observer coefficients can be calculated by the Matlab function shown in Figure A-2.

A more in-depth study of the Angle Tracking Observer implementation and a detailed discussion regarding its stability and accuracy can be found in [1].

3.3 Resolver Driver eTPU Implementation

The implementation of the Angle Tracking Observer algorithm on the eTPU module requires the adoption of specific eTPU features. This section presents the major aspects of the use of the resolver driver on the eTPU module.

3.3.1 Operating Modes

The RSLV eTPU function can operate in various operating modes. Different operating modes result in different RSLV eTPU function behavior.

Operating modes are dependent on each other and can be hierarchically ordered as follows. There are two primary, mutually excluding, modes: `MASTER` and `SLAVE` mode. In the `MASTER` mode, the RSLV eTPU function generates a left-aligned PWM wave with a specified period and duty cycle. In the `SLAVE` mode, the generation of the PWM signal is not performed. The `SLAVE` mode is the default mode and does not need to be specified.

While in the MASTER mode, the RSLV eTPU function can be configured as being in some other operating mode. The basic operating modes are as follows (see [3] for the full list):

- PERIODIC and NOPERIODIC

  These modes determine whether the Angle Tracking Observer single computation is executed periodically, once in every PWM period, or not. If the PERIODIC mode is specified, then the execution will take place at the time defined by the delay parameter, defined in the call to the initialization routine. If the NOPERIODIC mode is specified, then the Angle Tracking Observer computation will not occur periodically and only on a Host Service or a Link Request. The NOPERIODIC mode is the default and does not need to be specified.

- TCR1 and TCR2

  These modes specify which eTPU timer is used for generation of the PWM wave. The TCR1 mode stands for the TCR1 timer, and TCR2 for the TCR2 timer. The TCR1 mode is the default mode and does not need to be specified.

In both MASTER and SLAVE modes, the RSLV eTPU function can be configured to issue a link service request at the end of the Angle Tracking Observer computation by defining an additional mode:

- LINK and NOLINK

  These modes determine whether the RSLV eTPU function issues an eTPU link request. If the LINK mode is specified, then the RSLV eTPU function will issue an eTPU link request to the other channel defined at initialization. By default, the mode NOLINK is active and does not need to be specified.

These modes should be specified by ORing the C pre-processor definitions, which can be found in the etpu_rslv.h file. A definition for specifying the RSLV eTPU function mode has the form of: FS_ETPU_RSLV_<mode_name>, where <mode_name> is the name of the mode to be set (see [3] for details). For example, in order to define the MASTER mode with periodic computation of the Angle Tracking Observer algorithm and a link request, write the following into the source code:

```
FS_ETPU_RSLV_MODE_MASTER | FS_ETPU_RSLV_MODE_PERIODIC | FS_ETPU_RSLV_MODE_LINK.
```

In order to ease the definition of the operating mode, a few additional definitions can be used, as in Table 1. The aliases have the advantage of being recognized by the eTPU Graphical Configuration Tool (see [2]).

Table 1. Aliases for Most Common Operating Modes

Operating Mode Alias	MASTER **or** SLAVE	PERIODIC **or** NOPERIODIC	NOLINK **or** LINK	TCR **or** TCR2
FS_ETPU_RSLV_MASTER_PERIODIC_NOLINK	MASTER	PERIODIC	NOLINK	TCR1
FS_ETPU_RSLV_MASTER_PERIODIC_LINK	MASTER	PERIODIC	LINK	TCR1
FS_ETPU_RSLV_MASTER_NOPERIODIC_NOLINK	MASTER	NOPERIODIC	NOLINK	TCR1
FS_ETPU_RSLV_MASTER_NOPERIODIC_LINK	MASTER	NOPERIODIC	LINK	TCR1
FS_ETPU_RSLV_MASTER_PERIODIC_NOLINK_TCR1	MASTER	PERIODIC	NOLINK	TCR1
FS_ETPU_RSLV_MASTER_PERIODIC_LINK_TCR1	MASTER	PERIODIC	LINK	TCR1
FS_ETPU_RSLV_MASTER_NOPERIODIC_NOLINK_TCR1	MASTER	NOPERIODIC	NOLINK	TCR1
FS_ETPU_RSLV_MASTER_NOPERIODIC_LINK_TCR1	MASTER	NOPERIODIC	LINK	TCR1

Table 1. Aliases for Most Common Operating Modes (continued)

Operating Mode Alias	MASTER or SLAVE	PERIODIC or NOPERIODIC	NOLINK or LINK	TCR or TCR2
FS_ETPU_RSLV_MASTER_PERIODIC_NOLINK_TCR2	MASTER	PERIODIC	NOLINK	TCR2
FS_ETPU_RSLV_MASTER_PERIODIC_LINK_TCR2	MASTER	PERIODIC	LINK	TCR2
FS_ETPU_RSLV_MASTER_NOPERIODIC_NOLINK_TCR2	MASTER	NOPERIODIC	NOLINK	TCR2
FS_ETPU_RSLV_MASTER_NOPERIODIC_LINK_TCR2	MASTER	NOPERIODIC	LINK	TCR2
FS_ETPU_RSLV_SLAVE_NOLINK	SLAVE	N/A	NOLINK	N/A
FS_ETPU_RSLV_SLAVE_LINK	SLAVE	N/A	LINK	N/A

3.3.2 Accessing the Angle Tracking Observer Algorithm

The Angle Tracking Observer can be accessed as follows:

- By means of the API call: `fs_etpu_rslv_compute()` which will execute a single iteration of the Angle Tracking Observer routine.
- By means of an eTPU link request from another channel, see Section 3.3.3, "Interrupts, Data Transfer, and Link Requests."
- In the MASTER mode by means of periodic invocation, see Section 3.3.3, "Interrupts, Data Transfer, and Link Requests."

It should be noted that in the API call `fs_etpu_rslv_compute()` and on an eTPU link request, a single iteration of the Angle Tracking Observer routine will take place always, regardless of how the RSLV eTPU function is configured.

At any time, the Angle Tracking Observer algorithm can be reset by a call to the `fs_etpu_rslv_reset()` function (see [3]).

3.3.3 Interrupts, Data Transfer, and Link Requests

The RSLV eTPU function can interact with other modules in a device through a CPU interrupt request, data transfer requests, and/or eTPU link requests.

The Angle Tracking Observer function will be executed if an eTPU link request is placed by another channel.

The Angle Tracking Observer function, at the end of its single iteration, issues a CPU interrupt request, a data transfer request, and/or an eTPU link request to another channel.

The RSLV eTPU function needs to be appropriately configured in order to issue link service requests, by setting the mode LINK, see Section 3.3.1, "Operating Modes."

CPU interrupts and eTPU link requests are issued at a specified rate defined at the RSLV eTPU function initialization. This mechanism enables the issuing of requests at every single, every second, every third, and so on invocation of the Angle Tracking Observer algorithm. The rates for the CPU interrupts and the eTPU link requests can be specified by the arguments `rate_irq` and `rate_link` in the API call `fs_etpu_rslv_init()`.

3.3.4 Angle Tracking Observer Inputs

The Angle Tracking Observer uses two 24-bit parameters for storing the resolver sine and cosine signals, with the sine parameter placed at the lower address. Access to these parameters is granted by two API calls:

- `fs_etpu_rslv_set_sincos()` used to set the sine and cosine parameters,
- `fs_etpu_rslv_get_sincos_addr()` used to acquire the address of the sine and cosine pair.

Additionally, two definitions are placed in the file `etpu_rslv_auto.h`, which contains the offsets of the local RSLV eTPU function parameter memory for the above parameters: `FS_ETPU_RSLV_SINA_OFFSET` and `FS_ETPU_RSLV_COSA_OFFSET`.

Both parameters are assumed to be within the signed fractional range of –1 to almost 1, which corresponds to `800000` and `7fffff` in hexadecimal notation.

Before being used by the Angle Tracking Observer algorithm, both the sine and cosine signals are first shifted left by the `scale` parameter and the `dc_offset` parameter is added to the results. After shifting and addition, the sine and cosine signals cannot exceed the fractional range and need to be positioned symmetrically around zero.

It should be noted that due to eTPU module requirements, the two parameters are placed in two 4-byte aligned, consecutive, 32-bit words in memory. However, the parameters occupy the 24 less significant bits of each 32-bit word. See [4] for more details.

3.3.5 Getting the Position and Speed

The current resolver position is represented by three RSLV eTPU function parameters: an angle within a single 360 degree rotation, the RSLV eTPU function offset, and the number of revolutions. The angle is a variable used by the Angle Tracking Observer algorithm for computation and represents the actual resolver angle within a single 360 degree rotation. The offset is an additional parameter, added to the computed angle in order to calculate the resolver position and may be used to conveniently settle the beginning of the angle axis. The number of revolutions informs on how many full 360 degree rotations the resolver has made.

The actual resolver position can be calculated as the sum of:

- the Angle Tracking Observer computed angle,
- the Angle Tracking Observer offset,
- the number of revolutions.

All parameters need to be transformed from fractional and integer into real ranges:

$$\Theta = angle \times \pi + offset \times \pi + revolutions \times \pi$$ ***Eqn. 13***

The resolver driver API supports three API calls for determining the actual resolver position:

- `fs_etpu_rslv_get_position()` to obtain the Angle Tracking Observer angle adjusted by the Angle Tracking Observer offset,
- `fs_etpu_rslv_set_position()` to set the Angle Tracking Observer offset,
- and `fs_etpu_rslv_get_revolutions()` to obtain the number of revolutions.

To provide access to the computed angular velocity of the resolver, the API call `fs_etpu_rslv_get_speed()` is provided. This API call returns a digital representation of the resolver angular velocity. The actual angular velocity can be obtained by multiplying the returned value by the sampling frequency and π:

$$\Omega = \Omega_d \times \pi \times f_s$$ ***Eqn. 14***

where: Ω is the actual resolver angular velocity in rad/s, Ω_d is the digital representation of the actual resolver angular velocity, and f_s is the sampling frequency in Hz.

3.3.6 PWM Generation

While in the `MASTER` mode, the RSLV eTPU function generates a simple, left-aligned PWM wave. The PWM wave is defined by the following parameters, which must be defined at the RSLV eTPU function initialization:

- `period` — period of the PWM wave expressed in the selected eTPU timer ticks,
- `duty` — duty cycle of the PWM wave expressed in the selected eTPU timer ticks,
- `delay` — computation delay against the beginning of a period expressed in the selected eTPU timer ticks.

It should be noted that the `period`, `duty`, and `delay` parameters are effective only if the `MASTER` operating mode is specified.

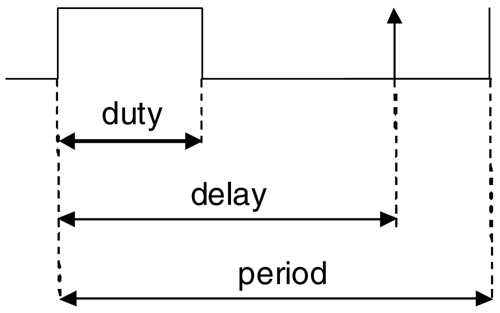


Figure 7 PWM Wave

The PWM parameters can also be changed while the PWM wave is being generated. This operation needs to be performed in two steps. In the first step, a PWM parameter needs to be modified by the appropriate API call: `fs_etpu_rslv_set_period()`, `fs_etpu_rslv_set_duty()`, or `fs_etpu_rslv_set_delay()`. In the second step, the RSLV eTPU function needs to be updated with the new parameters by means of a call to the function `fs_etpu_rslv_update()`. These two steps can also be performed by one function, `fs_etpu_rslv_update_ext()`. The new parameters will be applied at the beginning of the next PWM period.

3.4 Performance

The RSLV eTPU function performance depends on how the eTPU function is configured and under what conditions it operates.

The RSLV eTPU function computational load depends on whether it operates periodically or on requests.

For periodic operation, the computational load of the function depends on:

- whether the Angle Tracking Observer algorithm is executed periodically,
- whether the RSLV eTPU function is processing single or simultaneous events.

In general, under normal conditions, the function is processing single events, which means that all function events: the beginning of the PWM pulse, the end of the PWM pulse, and the Angle Tracking Observer routine invocation, do not occur at the same time. Under some specific conditions, the events may have to be processed simultaneously, for example, if the end of the PWM pulse and invocation of the Angle Tracking Observer routine is configured to occur at the same time.

In summary, the computational load of the RSLV eTPU function within one PWM period, for the periodic operation, is the sum of the computational load of:

- Angle Tracking Observer single iteration when the function is configured for periodic execution of the Angle Tracking Observer algorithm,
- PWM generation within a period when the function is processing single events,
- PWM generation within a period when the function is processing simultaneous events.

If the RSLV eTPU function operates on requests, the computational load is the sum of performed-on-request activities, which can be: PWM update, PWM reset, or a single Angle Tracking Observer computation.

The summary of all performance figures is shown in Table 2.

Table 2. RSLV Function Performance Figures

Activity	Maximum processing time [eTPU cycles]
Angle Tracking Observer, single iteration	160
PWM generation within a period, single events only	28
PWM generation within a period, simultaneous events	45
PWM update	15
Angle Tracking Observer reset	13

4 C Level API for the Function

The C level API offers a convenient method to access all the features of the RSLV eTPU function. The API functions can be found in the `etpu_rslv.h` and `etpu_rslv.c` files. In order to use the RSLV API, these files need to be compiled and linked with other application source files.

It should be noted that this document does not include all the details of the C Level API. The full reference is included as comments within the `etpu_rslv.c` source code (see [3]).

Use of the API functions needs to follow specific phases of how the RSLV eTPU function operates. Calls to specific API functions are restricted by the RSLV eTPU function operation phase. The phases and the allowed API calls are presented in Figure 8.

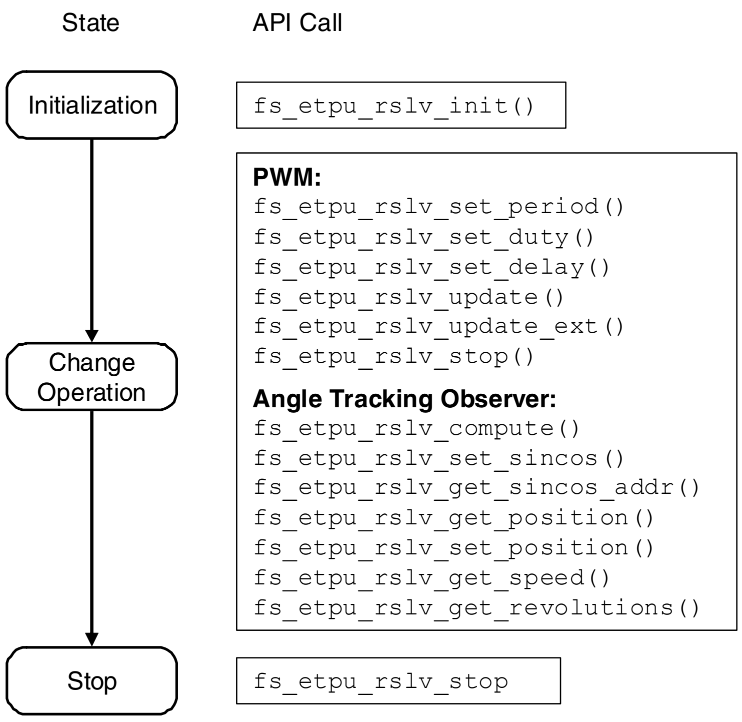


Figure 8 RSLV Function API State Flow

The subsequent sections contain description of individual API calls.

4.1 API Functions Description

4.1.1 int32_t fs_etpu_rslv_init(...)

This routine is used to initialize the eTPU channel for the RSLV function. The following arguments are accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `priority` (`uint8_t`) — This is the priority to assign to the RSLV function. This argument should be assigned a value of:
  - `FS_ETPU_PRIORITY_HIGH`
  - `FS_ETPU_PRIORITY_MIDDLE`
  - `FS_ETPU_PRIORITY_LOW`
  - `FS_ETPU_PRIORITY_DISABLE`
- `mode` (**`uint32_t`**) — This is the channel operating mode. One or more of the following should be ORed, the entries marked as default do not need to be specified:
  - `FS_ETPU_RSLV_MODE_PERIODIC`
  - `FS_ETPU_RSLV_MODE_NOPERIODIC` (default)
  - `FS_ETPU_RSLV_MODE_MASTER`

— `FS_ETPU_RSLV_MODE_SLAVE` (default)
— `FS_ETPU_RSLV_MODE_TCR1` (default)
— `FS_ETPU_RSLV_MODE_TCR2`
— `FS_ETPU_RSLV_MODE_LINK`
— `FS_ETPU_RSLV_MODE_NOLINK` (default)

See also the `etpu_rslv.c` source code and Table 1.

- `period` (`uint32_t`) — This is the period of the left-aligned pulse-width waveform generated by the channel. The period is expressed as a 24-bit unsigned integer in time base ticks in the range of `0` to `7fffff` in hexadecimal notation.
- `duty` (`uint32_t`) — This is the delay between the start of the PWM wave and the start of the Angle Tracking Observer computation. The delay is expressed as a 24-bit unsigned integer in time base ticks. The valid range for the parameter is from `0` to `period - 1`. For the best performance, the parameters `duty` and `delay` should be different.
- `delay` (`uint32_t`) — This is the delay between the start of the PWM wave and the start of the Angle Tracking Observer computation. The delay is expressed as a 24-bit unsigned integer in time base ticks. The valid range for the parameter is from `0` to `period - 1`. For the best performance, the parameters `duty` and `delay` should be different.
- `start_offset` (`int32_t`) — This parameter defines when the generation of the PWM wave is activated. By default, the sum of this parameter and one period specifies a time offset relative to the current value of a time base before the PWM wave is started. The parameter is a signed 24-bit integer and can be positive or negative.
- `scale` (`uint32_t`) — Scaling constant for the signals from the resolver. The input sine and cosine values will be shifted left by the `scale` before being used by the Angle Tracking Observer.
- `dc_offset` (`int32_t`) — The value added to both signals from the resolver after the scaling operation and before being used by the Angle Tracking Observer algorithm.
- `link_chan` (`uint8_t`) — The number of a channel the link is sent to. The number needs to be right-aligned and in line with the format of the eTPU LINK register (see [4]).
- `rate_irq` (`uint32_t`) — The number of Angle Tracking Observer iterations per one interrupt request. 0 (zero) means one iteration per one interrupt request, 1 means two iterations per one interrupt request, and so on.
- `rate_link` (`uint32_t`) — The number of Angle Tracking Observer iterations per one link request. 0 (zero) means one iteration per one link request, 1 means two iterations per one link request, and so on.
- `k1_d` (`int32_t`) — Multiplication factor of the first Angle Tracking Observer coefficient. This is a 24-bit fractional number.
- `k1_scale` (`int32_t`) — Scaling constant of the first Angle Tracking Observer coefficient. This is an unsigned 24-bit integer and must be greater than or equal to 1.
- `k2_d` (`int32_t`) — Multiplication factor of the second Angle Tracking Observer coefficient. This is an unsigned 24-bit integer.
- `k2_scale` (`int32_t`) — Scaling constant of the second Angle Tracking Observer coefficient. This is an unsigned 24-bit integer and must be greater than or equal to 0.

4.1.2 int32_t fs_etpu_rslv_set_period(…)

This function should be used to modify the PWM wave period. This function does not directly cause a change in the PWM wave period. In order to cause a change, the function `fs_etpu_rslv_update()` should be called.

The function accepts two arguments:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. This argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `period` (`uint8_t`) — This is the period of the PWM wave in timer base clock cycles. This is an unsigned 24-bit integer.

4.1.3 int32_t fs_etpu_rslv_set_duty(…)

This function should be used to modify the PWM wave duty cycle. A call to the function does not result directly in a change of the PWM wave duty cycle. In order to cause a change, the function `fs_etpu_rslv_update()` should be called.

The function accepts two arguments:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `duty` (`uint32_t`) — This is the duty cycle of the PWM wave in timer base clock cycles. This is an unsigned 24-bit integer and must be lower than `period`.

4.1.4 int32_t fs_etpu_rslv_set_delay(…)

This function should be used to modify the computational delay of the Angle Tracking Observer against the beginning of the PWM wave period. A call to this function does not result directly in a change of the computational delay. In order to cause an actual change, the function `fs_etpu_rslv_update()` should be called.

The function accepts two arguments:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `delay` (`uint32_t`) — This is the computational delay of the Angle Tracking Observer in timer base clock cycles. This is an unsigned 24-bit integer and must be lower than `period`.

4.1.5 int32_t fs_etpu_rslv_update(...)

This function should be used to make an actual change to the PWM wave parameters. On the next PWM wave period after a call to this function, the modification made by the functions: `fs_etpu_rslv_set_period()`, `fs_etpu_rslv_set_duty()`, `fs_etpu_rslv_set_delay()` will actually take place.

The following arguments are accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

4.1.6 int32_t fs_etpu_rslv_update_ext(...)

This function should be used to make an actual change to the PWM wave parameters. From the next PWM wave period after a call to the function, the function arguments will be put into effect.

The following arguments are accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `period` (`uint32_t`) — This is the period of the PWM wave in timer base clock cycles. This is an unsigned 24-bit integer.
- `duty` (`uint32_t`) — This is the duty cycle of the PWM wave in timer base clock cycles. This is an unsigned 24-bit integer and must be lower than `period`.
- `delay` (`uint32_t`) — This is the computational delay of the Angle Tracking Observer in timer base clock cycles. This is an unsigned 24-bit integer and must be lower than `period`.

4.1.7 int32_t fs_etpu_rslv_compute(...)

This function performs a single iteration of the Angle Tracking Observer algorithm. The computation will take place regardless of the set operating mode. One argument is accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

4.1.8 int32_t fs_etpu_rslv_set_sincos(...)

This function sets the sine and cosine inputs for the Angle Tracking Observer algorithm. The arguments are 32 bits long, however the sine and cosine values need to be provided in the signed 24-bit fractional format, occupying the 24 less significant bits of the 32-bit word. The 8 most significant bits are neglected.

The signed 24-bit fractional format means that the minimum valid value is –1 and the maximum valid value is almost 1 (exactly $1 - 2^{-23}$), which corresponds to `800000` and `7fffff` in hexadecimal notation.

Before taken into calculations by the Angle Tracking Observer, the sine and cosine values are first shifted left by the `scale` parameter and then secondly the parameter `dc_offset` is added. After this operation the sine and cosine value should be still within the signed fractional range of –1 to 1 and should be positioned symmetrically around zero.

The following arguments are accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `sina` (`int32_t`) — The sine signal from a resolver in signed 24-bits fractional format, should occupy the 24 less significant bits.
- `cosa` (`int32_t`) — The cosine signal from a resolver in signed 24-bits fractional format, should occupy the 24 less significant bits.

4.1.9 int32_t* fs_etpu_rslv_get_sincos_addr(...)

This function returns the address of the first 32-bit word holding the sine and cosine parameters for the Angle Tracking Observer algorithm.

One argument is accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

4.1.10 int32_t fs_etpu_rslv_get_position(...)

This function returns the current resolver position.

The position is provided as a value modulo in the form of a 24-bit signed fractional number, with `800000`hex (–1.0) corresponding to $-\pi$ and `7fffff`hex (almost 1.0) corresponding to π.

The function takes into account the Angle Tracking Observer offset by returning the sum of the computed Angle Tracking Observer angle and the Angle Tracking Observer offset. The adjustment by the offset can be used to conveniently set the beginning of the angle axis.

The function `fs_etpu_rslv_set_position()` can be used to set the Angle Tracking Observer offset.

The returned value is the angle within a single 360 degree rotation. The absolute position can be calculated by adding:

- the returned value multiplied by π,
- the number of revolution multiplied by $2 \times \pi$.

One argument is accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B

4.1.11 int32_t fs_etpu_rslv_set_position(...)

This function sets the new resolver position.

The new position needs to be provided as a value modulo in the form of a 24-bit signed fractional number, with `800000`hex (–1.0) corresponding to $-\pi$ and `7fffff`hex (almost 1.0) corresponding to π.

The function sets the Angle Tracking Observer offset, so that the immediate call to the `fs_etpu_rslv_get_position()` will return the new position value.

Two arguments are accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.
- `position` (`int32_t`) — The new resolver position.

4.1.12 int32_t fs_etpu_rslv_get_speed(...)

This function returns the current resolver angular velocity.

The returned value is the digital representation of the speed. In order to calculate the physical value, it is necessary to multiply the returned value by $\pi \times f_s$, where f_s is the sampling frequency.

One argument is accepted:

- `channel (uint8_t)` — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

4.1.13 int32_t fs_etpu_rslv_get_revolutions(...)

This function returns the resolver current revolution number of the resolver.

The number of revolutions is necessary to determine the absolute position of the resolver, as the resolver position is provided within a single 360 degree rotation. The absolute position can be calculated by adding:

- the position returned by the `fs_etpu_rslv_get_position()` function multiplied by π,
- the returned value multiplied by $2 \times \pi$.

One argument is accepted:

- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

4.1.14 int32_t fs_etpu_rslv_stop(...)

This function ends the generation of the PWM wave and also, if applicable, the periodic invocation of the Angle Tracking Observer algorithm.

It should be noted that even after the call to this function, the Angle Tracking Observer algorithm can be accessed through the `fs_etpu_rslv_compute()` function.

One argument is accepted:

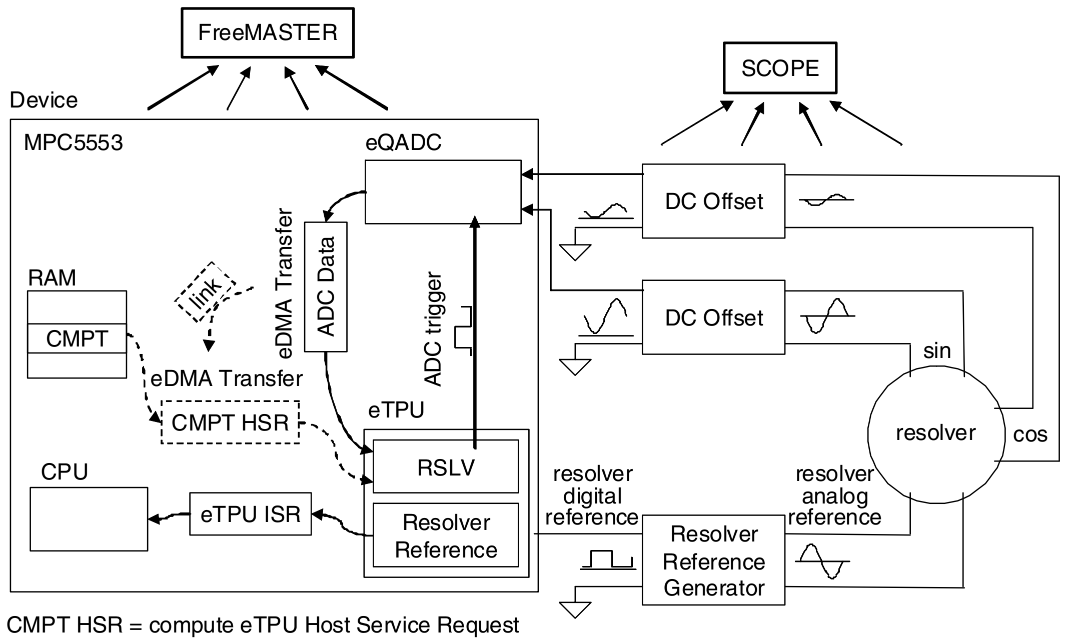
- `channel` (`uint8_t`) — This is the number of the RSLV function channel. The argument should be assigned a value of 0–31 for ETPU_A, and 64–96 for ETPU_B.

5 Example Use of Function

Two application examples of using the resolver driver are presented to show the two basic configurations. Both examples are similar and differ in the way that the resolver eTPU function is configured to call the Angle Tracking Observer routine.

In both examples, one of the eTPU channels generates a simple, left-aligned PWM signal, called the resolver digital reference. This signal is used by the Resolver Reference Generator to generate a sinusoidal wave, called the resolver analog reference, which is supplied to the input of the resolver. According to the shaft position, the resolver generates appropriate signals on its two output windings. Both signals are biased by a DC offset, so that their range remains within the acceptable range of the eQADC inputs.

Both applications are monitored by an electrical scope and FreeMASTER (see Figure 9). A detailed description is included in the next sections.



CMPT HSR = compute eTPU Host Service Request
eTPUISR = eTPU Interrupt Service Request

Figure 9 Signal Flow of Resolver Driver Application Examples

In order to better illustrate the operation of the application examples, a timing diagram is presented. The timing diagram shows all the important signals and events with respect to their time relationships.

In both applications, signals are synchronized in time as follows. The primary signal is the resolver digital reference. The resolver analog reference is shifted by a specific phase shift, which is related to the analog circuitry of the Resolver Reference Generator. The resolver driver PWM wave is shifted in phase against the resolver digital reference so that its rising edge appears exactly at the maximum of the resolver analog reference signal. The resolver driver PWM wave is the source of the A/D conversion, which in turn is the source of other events in the systems. The events triggered by the A/D conversion differ in both examples, which will be described in detail in the subsequent sections.

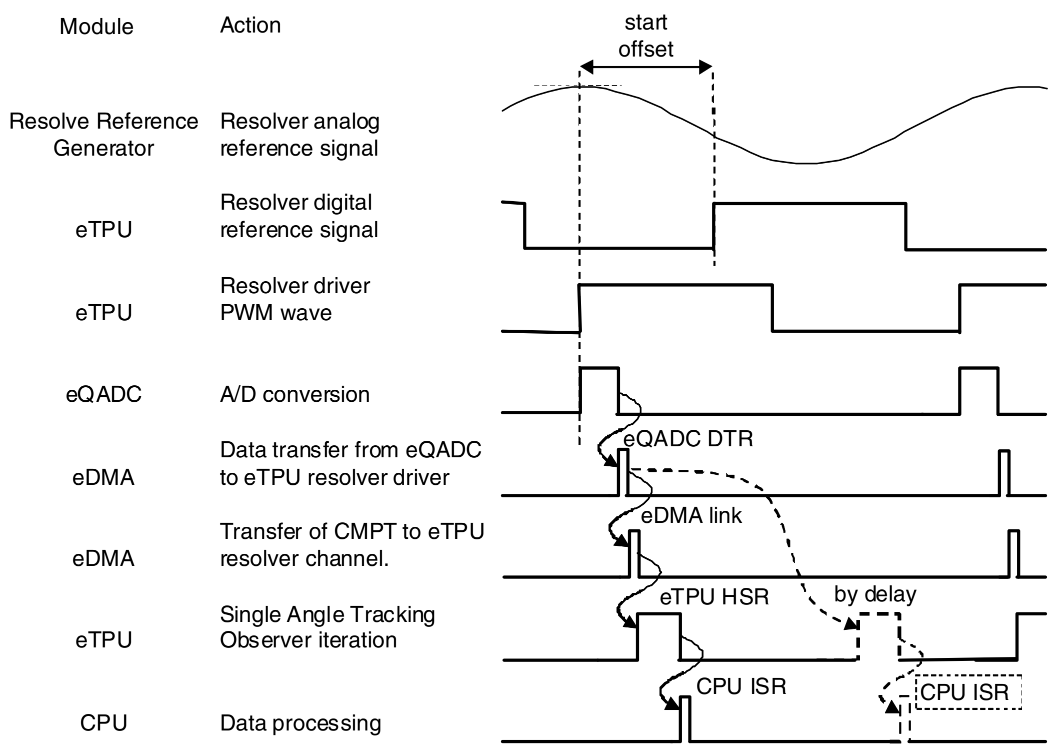

Module
Action
start offset
Resolve Reference Generator
Resolver analog reference signal
eTPU
Resolver digital reference signal
eTPU
Resolver driver PWM wave
eQADC
A/D conversion
eQADC DTR
eDMA
Data transfer from eQADC to eTPU resolver driver
eDMA link
eDMA
Transfer of CMPT to eTPU resolver channel.
eTPU HSR
by delay
eTPU
Single Angle Tracking Observer iteration
CPU ISR
CPU ISR
CPU
Data processing

Figure 10 Timing Diagram of Resolver Driver Application Examples

5.1 Data Synchronization Example

In the first example, the eQADC, eTPU, and the CPU are synchronized by data transfer, which means that each module starts its own task after receiving a piece of data.

In this example, two eDMA channels are used. The first one performs data transfer from the eQADC to the eTPU RSLV function upon completion of an eQADC A/D conversion.

The second eDMA channel is triggered by the first one by the eDMA link, and makes a transfer of the CMPT Host Service Request to the RSLV eTPU channel (see Figure 9, diagram elements in the dotted line). The writing of the CMPT Host Service Request to the RSLV eTPU channel memory causes the invocation of the Angle Tracking Observer routine to be scheduled.

In the first eDMA data transfer from the eQADC to the eTPU, care is taken to correctly adopt the eQADC and eTPU interfaces. Firstly, the eDMA is configured so that the 16-bit data from the A/D conversion is placed in the 16 less significant bits of the 32-bit words in the RSLV eTPU function parameter memory. Secondly, the eQADC data format is appropriately configured and the RSLV eTPU `scale` and `dc_offset` parameters are properly set. In the example, the following configuration is applied:

- the eQADC A/D conversion result is configured to be a signed right-justified fractional value (`FMT` = 1),
- the `scale` RSLV eTPU parameter is set to 10, in order to shift the data from the eQADC in order to appear in the 24-bit fractional format,
- the `dc_offset` RSLV eTPU parameter is set so that the data from the eQADC is positioned symmetrically around zero (see Figure 11).

In the application, events unfold as follows. Upon a trigger from the RSLV eTPU function, the eQADC starts an A/D conversion. At the end of the A/D conversion, an eDMA channel is triggered to transfer data from the eQADC to the RSLV eTPU channel memory. After completing the transfer, the eDMA channel issues a link request to another eDMA channel, which performs a transfer of the CMPT Host Service Request to the RSLV eTPU function. Upon completion of this transfer the eTPU schedules execution of the Angle Tracking Observer routine. At the end of execution of the Angle Tracking Observer routine, a CPU interrupt is issued, which causes execution of an interrupt service routine. In the example, the interrupt service routine toggles a device output pin, showing CPU activity (see Figure 10 and Figure 12).

The following is the actual initialization of the RSLV eTPU function:

```
err_code = fs_etpu_rslv_init (RSLV0_CHANNEL,/* engine: A; channel: 1 */
                                FS_ETPU_PRIORITY_LOW,/* priority: Low */
                                FS_ETPU_RSLV_MODE_MASTER,/* mode: FS_ETPU_RSLV_MODE_MASTER*/
                                10000,/* period: 10000 */
                                5000,/* duty: 5000 */
                                2500,/* delay: 2500 */
                                -3056,/* start_offset: -3056 */
                                10,/* scale: 10 */
                                0x002B851E, /* dc_offset */
                                0,/* link_chan: 0 */
                                0,/* rate_irq: 0 */
                                0,/* rate_link: 0 */
                                0x005E87CE,/* k1_d: 0x004056fe */
                                8,/* k1_scale: 8 */
                                0x00433A20,/* k2_d: 0x00517cc2 */
                                3);/* k2_scale: 3 */
```

Figure 11 Initialization of the RSLV eTPU Function in the Data Synchronization Example

The actual electrical signals from the application are shown in Figure 12. In the figure, Ch4 is the resolver digital reference, Ch3 is the resolver analog reference, Ch2 is the RSLV eTPU function PWM wave, and Ch1 is a signal representing the CPU activity.

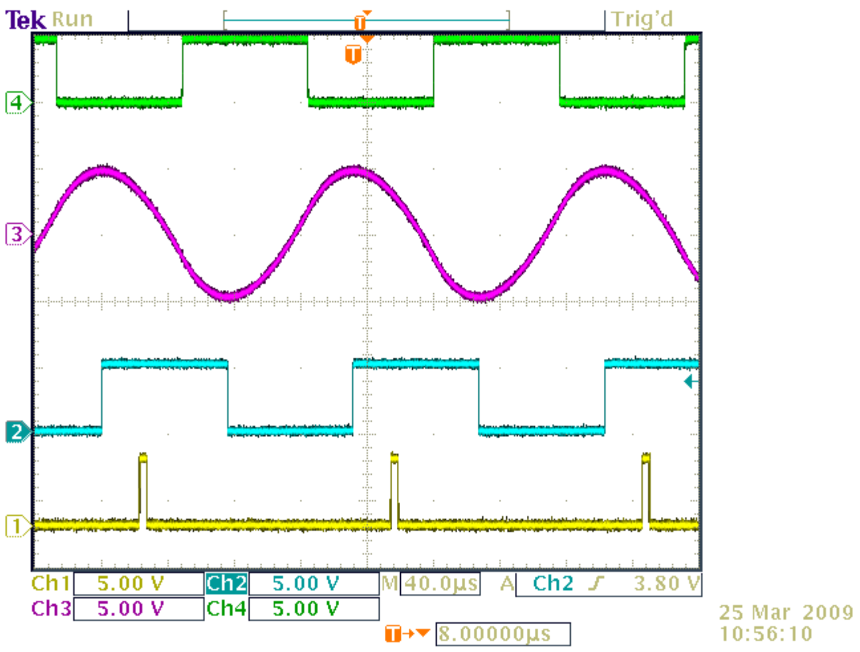


Figure 12 Electrical Signals in Data Synchronization Example

Figure 13 presents an actual screen of the FreeMASTER program monitoring the operation of the Angle Tracking Observer. The shaft of the rotor, where the resolver is mounted, is rotated manually. The figure shows a slow movement of the rotor in two directions.

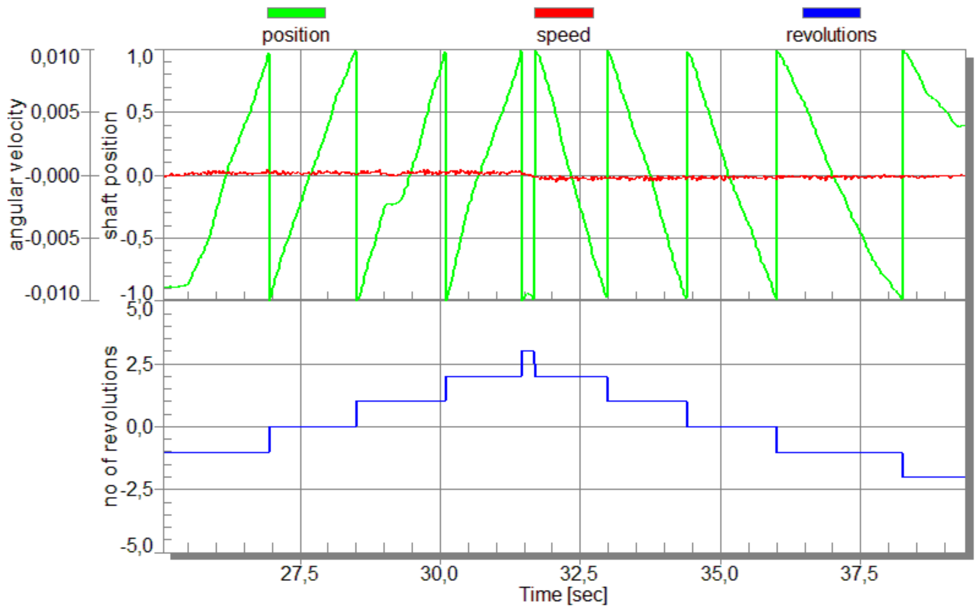


Figure 13 Position and Angular Velocity Computed in the Data Synchronization Example

5.2 Timing Synchronization Example

In the second example the eQADC, eTPU, and the CPU are synchronized by an appropriate definition of the timing constants.

In this example, the eDMA channel transfers the data from the eQADC to the RSLV eTPU channel and stops. After a certain delay, defined at initialization of the RSLV eTPU function, the computation of the Angle Tracking Observer routine is executed. This sequence of events is shown in the dotted line in Figure 10. This method assumes that the computational delay is defined so that the correct input data is available for the Angle Tracking Observer algorithm on execution.

The following is the actual initialization of the RSLV eTPU function:

```
  err_code = fs_etpu_rslv_init (RSLV0_CHANNEL,/* engine: A; channel: 30 */
                                FS_ETPU_PRIORITY_LOW,/* priority: Low */
                                FS_ETPU_RSLV_MODE_MASTER | FS_ETPU_RSLV_MODE_PERIODIC,/*
 mode: FS_ETPU_RSLV_MODE_MASTER | FS_ETPU_RSLV_MODE_PERIODIC */
                                10000,/* period: 10000 */
                                5000,/* duty: 5000 */
                                7500,/* delay: 7500 */
                                -3056,/* start_offset: -3056 */
                                10,/* scale: 10 */
                                0x002B851E,/* dc_offset: 0x002B851E */
                                0,/* link_chan: 0 */
                                1,/* rate_irq: 1 */
                                0,/* rate_link: 0 */
                                0x005E87CE,/* k1_d: 0x004056fe */
                                8,/* k1_scale: 8 */
                                0x00433A20,/* k2_d: 0x00517cc2 */
                                3);/* k2_scale: 3 */
```

Figure 14 Initialization of the RSLV eTPU Function in the Timing Synchronization Example

The actual electrical signals from the application are shown in Figure 15. In the figure, Ch4 is the resolver digital reference, Ch3 is the resolver analog reference, Ch2 is the RSLV eTPU function PWM wave, and Ch1 is a signal representing CPU activity. It should be noted that the rate of the CPU interrupts is set to 1 (see the `rate_irq` argument of the `fs_etpu_rslv_init()` function), which results in issuing an interrupt request once every two iterations of the Angle Tracking Observer algorithm.

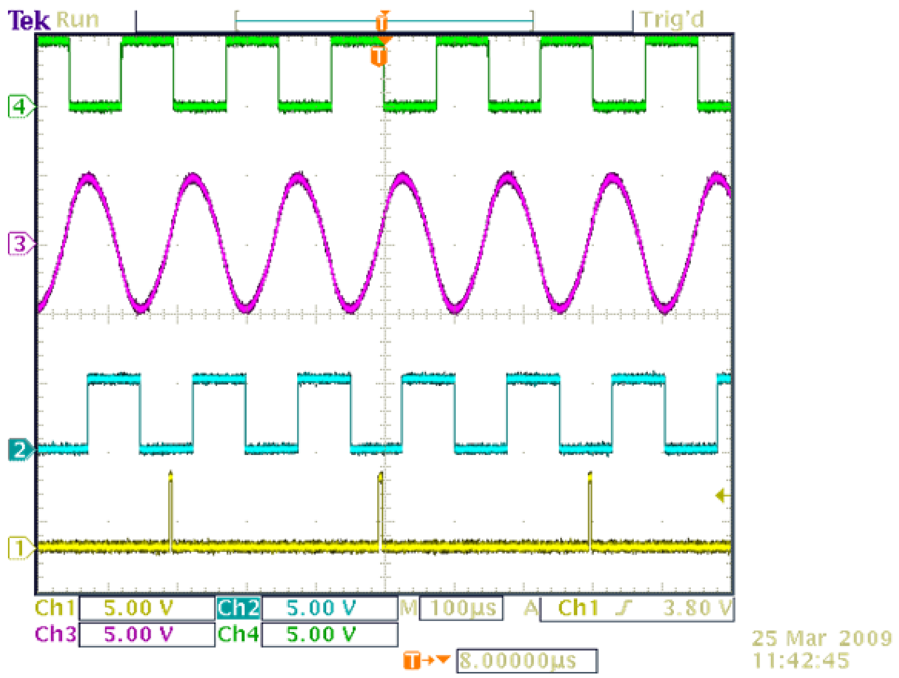


Figure 15 Electrical Signals in Timing Synchronization Example

Figure 15 presents an actual screen of the FreeMASTER program monitoring the operation of the Angle Tracking Observer. The shaft of the rotor, where the resolver is mounted, is rotated manually. The figure shows a slow movement of the rotor in two directions.

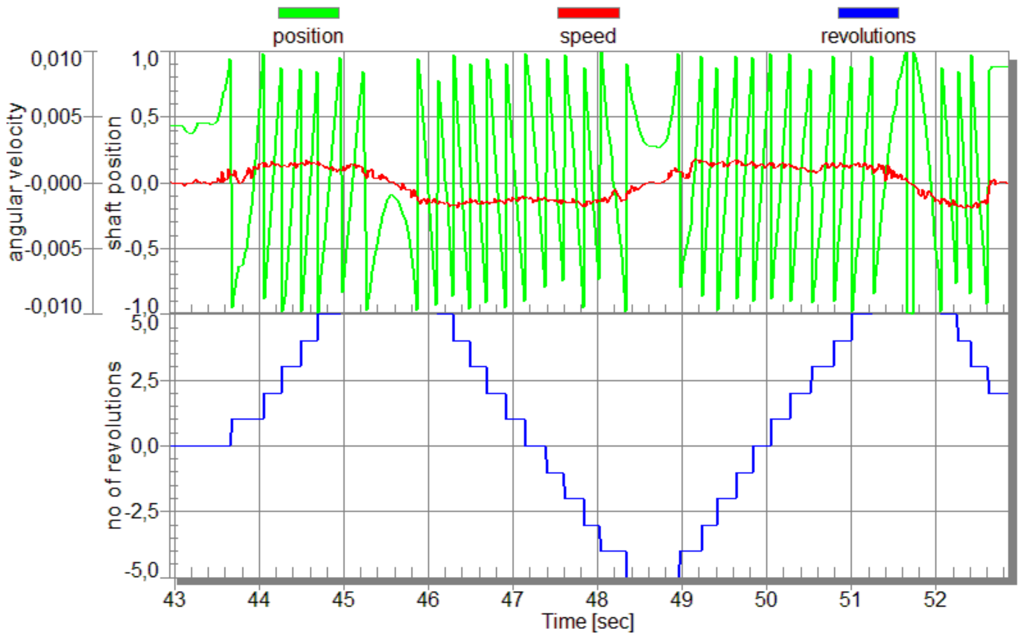


Figure 16 Position and Angular Velocity Computed in Timing Synchronization Example

5.3 Electrical and Mechanical Components

An electrical schematic of the resolver hardware interface is shown in Figure A-1.

In the lower part of the schematic, the Resolver Reference Generator is shown and consists of the U204A and U204B amplifiers and surrounding passive elements. Both amplifiers form a filter, which converts the incoming left-aligned PWM rectangle signal into a sinusoidal one by passing through the main harmonic and filtering out the higher ones. The R224 and C226 protect the output from high frequency oscillations (so-called Boucherot cell). The C223 decouples the output DC component. The overall transfer function is given by:

$$H(s) = \frac{V_{\mathrm{RES\_GEN}}}{V_{\mathrm{RES\_REF}}} = \frac{R_7}{R_6 + R_7} \times \frac{R_2 \times R_3}{R_1 \times R_8} \times \frac{s \times \tau_1}{(1 + s \times \tau_4) \times (1 + s \times \tau_2) \times (1 + s \times \tau_3)}$$
$$\tau_1 = R_1 \times C_1$$
$$\tau_2 = R_2 \times C_2 \qquad \textit{Eqn. 15}$$
$$\tau_3 = R_3 \times C_3$$
$$\tau_4 = R_1 \times C_1 \times (1 + \frac{R_6 \times R_7}{R_6 + R_7} / R_1)$$

where R_6 and R_7 are the actual resistances of the resistive voltage divider R6-R7.

The upper part of the schematic shows a signal amplification circuit with DC Offset circuitry. There are two differential amplifiers with an output level referenced to the virtual ground/DC offset (middle of the V_{CCA} +3.3 V). The virtual ground is created by U200A, set to +1.65 V. The U202A, B amplifiers are rail-to-rail MC33502, or similar, capable of 3.3 V single supply operation. The capacitors C206, C208, C211, C213, C218, and C219 add low-pass filtering to suppress unwanted high-frequency noise, which is often present in systems with power switching units.

All values of the components in the schematic are designed to operate at about 6.5 kHz of the resolver analog reference signal. If the operating frequency changes, the element values need to be adjusted accordingly.

A TGDrives company (see [5]) motor, with an integrated resolver of LTN (see [6]), was used.

6 Summary and Conclusions

This application note describes an eTPU function dedicated to be a software interface between a device with an eTPU module and a resolver hardware. Two application examples are presented.

The presented eTPU function offers a convenient method to initially process resolver signals without any computational load from CPU.

References:

1. DSP56F80x Resolver Driver and Hardware Interface AN1942/D.
2. eTPU Graphical Configuration Tool, http://www.freescale.com/etpu, ETPUGCT.
3. Source code of host API for the RSLV eTPU function `etpu_rslv.h`, `etpu_rslv.c`, http://www.freescale.com/etpu, RSLV.
4. Enhanced Time Processing Unit Reference Manual, ETPURM/D.
5. TGDrives TGT2-0032-30-24/TOPS1KX www.tgdrives.com.
6. LTN www.ltn.de.

Document Number: AN3943
Rev. 0
10/2009

Appendix A

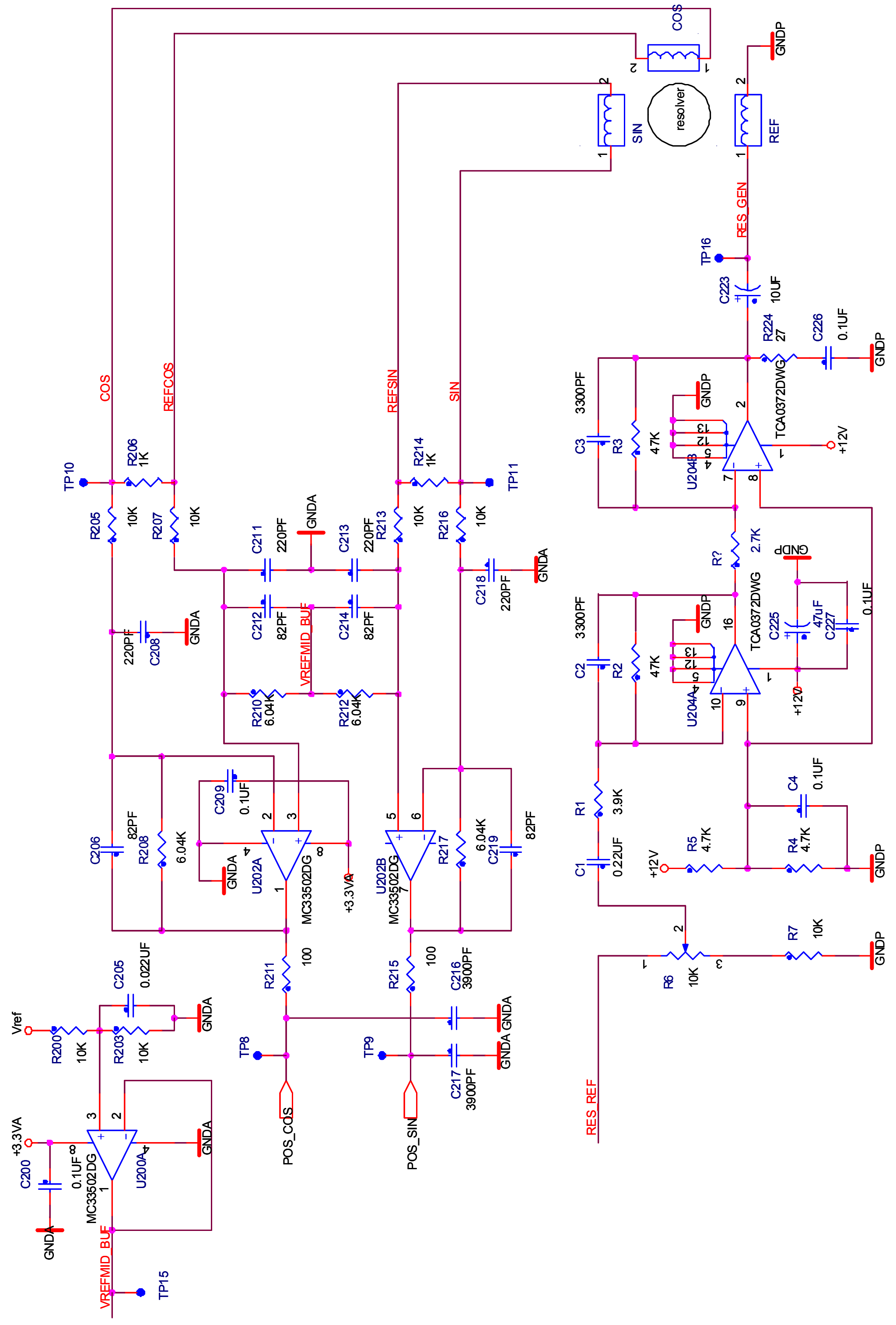


Figure A-1. Resolver Hardware Interface

```
function [K1_D, K2_D, K1_SCALE, K2_SCALE, er1, er2, K1, K2] = trackdempar(F0, D, Fs);
%
% function [K1_D, K2_D, K1_SCALE, K2_SCALE, er1, er2] = trackdempar(F0, D, Fs);
%
% Function calculates parameters of tracking observer.
% Input parameter:
%   F0 - natural frequency of tracking observer [Hz]
%   D  - damping factor [-]
%   Fs - sampling frequency [Hz]
%
% Output parameters are calculated according to these formulas:
%   K1 = (2*pi*F0/Fs)^2 / pi
%   K2 = 2*D/(2*pi*F0/Fs)
%
% Example:
%   F0 = 100; D = 0.2; Fs = 8000;
%   [K1_D, K2_D, K1_SCALE, K2_SCALE, er1, er2, K1, K2] = trackdempar(F0, D, Fs)

format long
Q = 1/(2*D);                                % quality factor
K1 = (2*pi*F0/Fs)^2/pi;% "Omega0^2" = (2*pi*F0/Fs)^2
K2 = 2*D/(2*pi*F0/Fs);% "2*Delta/Omega0" = Delta/(2*pi*F0)*Fs
K1_SCALE = floor(log2(1/K1));
K2_SCALE = ceil(log2(K2));
K1_D = K1*2^K1_SCALE;
K2_D = K2/2^K2_SCALE;
er1 = K1-K1_D/2^K1_SCALE;   % test if calculation is OK, should be 0
er2 = K2-K2_D*2^K2_SCALE;   % test if calculation is OK, should be 0
```

Figure A-2. Matlab Function for Calculation of Angle Tracking Observer Coefficients