User's Manual

RZ/T2M Group, RZ/T2L Group, RZ/N2L Group

CN032 AC Servo Solution Firmware Manual

Rev.4.00 Feb, 2023

Renesas Electronics
www.renesas.com

R11UM0169EJ0400

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

1. Purpose and Target Readers

This manual is designed to provide the user with an understanding of the hardware functions and electrical characteristics of the MCU. It is intended for users designing application systems incorporating the MCU. A basic knowledge of electric circuits, logical circuits, and MCUs is necessary in order to use this manual.

The manual comprises an overview of the product; descriptions of the CPU, system control functions, peripheral functions, and electrical characteristics; and usage notes.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

2. List of Abbreviations and Acronyms

Abbreviation	Full Form
bps	bits per second
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
Hi-Z	High Impedance
I/O	Input / Output
LSB	Least Significant Bit
MSB	Most Significant Bit
NC	Non-Connect
PWM	Pulse Width Modulation
SFR	Special Function Register
UART	Universal Asynchronous Receiver/Transmitter

3. List of related documents

- CN032 AC Servo Solution Hardware Manual (for RZ/T2M, RZ/N2L)
- CN032 AC Servo Solution Hardware Manual (for RZ/T2L)
- CN032 AC Servo Solution Firmware Manual (this manual)
- CN032 AC Servo Solution Startup Guide (for Motion Control Utility)
- CN032 AC Servo Solution Startup Guide (for EtherCAT)

- RZ/T2M Group User's Manual: Hardware
- RZ/T2L Group User's Manual: Hardware
- RZ/N2L Group User's Manual: Hardware

TABLE OF CONTENTS

1. Introduction

1.1 Summary

CN032 AC Servo Solution Firmware is an embedded application that implements the functions of a full featured industrial servo controller. The reference code demonstrates the motion control capabilities of the Renesas RZ/T2M, RZ/T2L or RZ/N2L device including its high-performance deterministic CPU core, flexible absolute encoder interface and variety of connectivity options.

1.2 Function

The firmware implements the following main functions:

- Initialization of the RZ/T2M cores, RZ/T2L core or RZ/N2L core and its peripherals:

  Executable code is transferred from the QSPI Flash memory to the device RAM. The configurable hardware is initialized to support the preferred absolute encoder interface in the RZ/T2M. Whereas, the RZ/T2L and the RZ/N2L realizes the absolute encoder interface using SCI UART. The ADCs and the Timers are configured to interface with the Inverter. PWM Timer Interrupt handler is setup to invoke the Real-time control functions. The SCI Interrupt handlers are setup to respond to host commands. The GPIO pins are initialized to interact with the different digital inputs and outputs of the Controller board.

- Processing of the host commands

  The firmware supports two communication protocols concurrently – ASCII Command Protocol and Binary Packet Protocol. The command interpreter detects the type of the commands and invokes the appropriate dispatcher. The firmware recognizes over 100 commands providing access to all control parameters and algorithms. The host can obtain information periodically to track the status of each motor and control the execution of motion requests. Alternatively, the host can configure the collection of samples from different variables that can be buffered on the devices and analyzed later.

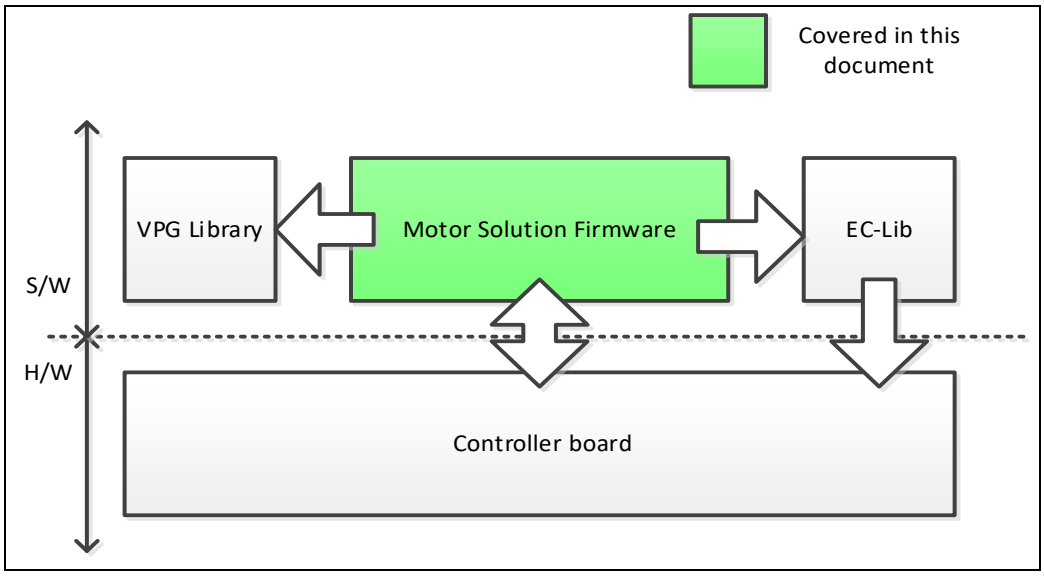
- Processing of the control loops algorithms in real-time

  The control algorithms are invoked from the context of the timer that generates the PWM timing (62.5us). This ensures deterministic – real-time performance of the control function. The real-time tasks include obtaining the current position, executing the position control loop, executing the current control loop (Field Oriented Control), generating the duty cycle for the next PWM period, and finally collecting data for future diagnostics.

1.3 Firmware Configuration

CN032 AC Servo Solution Firmware integrates the core motion control functionality with the features provided by two additional libraries. The ECL library is utilized to initialize and interface with different encoder protocols. The Velocity Profile Generation (VPG) library is used to generate a series of set points needed by facilitate point-to-point motion.

The figure below shows the CN032 AC Servo Solution Firmware place in relation to the rest of the firmware components.



2. Operating Environment

The library documented in this manual operates in the following environment:

Table 1 Operating Environment

Item	Description		
	RZ/T2M version	**RZ/T2L version**	**RZ/N2L version**
Microcomputer	RZ/T2M (Cortex®-R52 Dual)	RZ/T2L (Cortex®-R52 Single)	RZ/N2L (Cortex®-R52 Single)
Operating Frequency	RZ/T2M: 800 MHz	RZ/T2L: 800MHz	RZ/N2L: 400MHz
Operating Voltage	3.3V		
Development Environment	IAR Embedded Workbench® for ARM		
	Version: 9.320.1	Version: 9.32.1(**)	Version: 9.30.1
	Renesas Electronics e2studio		
	Version: 2022-04	Version: 2023-01	Version: 2022-07
	Renesas Electronics FSPSC(*)		
	Version: 2022-04	Version: 2023-01	Version: 2022-07
Flexible Support Package (FSP)	RZT FSP v1.0.0	RZT FSP v1.2.0	RZN FSP v1.0.0

(*) FSPSC (FSP Smart Configurator) is a code generation tool for IAR Embedded Workbench.

(**) After install EWARM 9.32.1, apply patch file for RZ/T2L. The patch file is available from URL below.
https://www.renesas.com/jp/ja/document/sws/rzt-fsp-packs-v120?r=25412341

3. File Configuration

3.1 AC Servo Solution Kit (RZ/T2M)

The below table shows the file configuration of the CN032 AC Servo Solution Firmware for AC Servo Solution Kit (RZ/T2M). Table 3-1 shows the file configuration of firmware for RZ/T2M.

Table 3-1 File configuration of firmware for RZ/T2M

File	Description
Common	
cg_src	They initialize the different RZ/T2M peripherals involved in the firmware operation.
ethercat\ application/ecat	The files in the directory are code specific for EtherCAT CiA402 communication
ethercat\inc\r_ecat_config.h	The file is header file for EtherCAT module device driver
ethercat\src\r_ecat\r_ecat_setting_rzt2.c	The file is code for EtherCAT module device driver
ethercat\src\r_ecat\hal	The files are header file and code for the hardware access layer of EtherCAT
ethercat\src\r_ecat\phy	The files are header file and code for EtherCAT PHY device driver
rzt\fsp\r_ecat\utilities\batch_file\apply_patch.bat	The batch file to apply a patch file on the Slave Stack Code
ethercat\src\r_ecat\utilities\batch_file\ CN032_AC_Servo_Solution_CiA402.patch	The patch file to be applied on the Slave Stack Code
ethercat\src\r_ecat\utilities\esi\Renesas_CN032_AC_Servo_Solution_CiA402.xml	EtherCAT Slave Information file
ethercat\src\r_ecat\utilities\ssc_config\CN032 AC Servo Solution EtherCAT CiA402.esp	The project file to execute SSC Tool

inc\apl\m_common.h	CN032 AC Servo Solution Firmware header file. Includes the motor data structure and signatures of all global functions.
inc\common\platform.h	Common type definitions header file
inc\dsm	The files in this directory are header files for delta-sigma interface driver
inc\qspi	The files in this directory are header files for QSPI memory access driver
inc\shm	The files in this directory are header files for shared memory access driver
lib\ecl\r_ecl_rzt2_if.h	Macros and function definitions of encoder interface library
lib\ecl\Config_Fa_Coder_V1.0.dat	Configuration file for the FA-Coder absolute encoder communication protocol
lib\ecl\RZT2M_pinmux_v1.0.bin	Pinmux file for the FA-Coder absolute encoder communication protocol
lib\gcc\r_ecl_rzt2_gcc.a	The Configurable hardware initialization library for GCC. It is used to facilitate the loading of configuration that matches the specifics of the selected absolute encoder interface protocol
lib\gcc\libVPG.a	The Velocity Profile Generation Library for GCC
lib\r_ecl_rzt2_iar.a	The Configurable hardware initialization library for IAR. It is used to facilitate the loading of configuration that matches the specifics of the selected absolute encoder interface protocol
lib\r_vpg.a	The Velocity Profile Generation Library for IAR
src\apl\m_commands.c	The code for all host commands that can be invoked
src\apl\m_commutation.c	The code for the motor commutation algorithms such as Space Vector Modulation, Hall-based Trapezoidal Commutation
src\apl\m_control.c	The real-time algorithms control execution branches dependent on different state and configuration options
src\apl\m_homing.c	The state machine implementing the homing algorithm
src\apl\m_interlocks.c	The functions checking various interlock conditions.
src\apl\m_interpreter.c	The command parser for the ASCII commands and the command decoder for the binary packets.
src\apl\m_phasing.c	The functions implementing the different phasing algorithms.
src\apl\m_pid_calc.c	The Position control loop algorithm implementation
src\apl\m_pos_read.c	The encoder position reading control algorithm

src\apl\m_recorder.c	The data collection functions and the start / stop triggers evaluation
src\apl\m_vpg_trap.c	The Trapezoidal Velocity Profile Generator
src\drv\m_biplane.h	Macro definitions specific for the hardware of the Solution and the RZ/T2M device
src\drv\m_rzt.c	Code specific for the hardware on the Solution and the RZ/T2M device
src\drv\qspi	The files in this directory are code specific for QSPI flash access
src\drv\dsm	The files in this directory are code specific for delta-sigma modulated interface
src\drv\shm	The file in this directory is code specific for shared memory access driver
src\encoder\FACoder	The files in this directory are dedicated to the implementation of the interface to the FA-Coder absolute encoder interface protocol
src\sharedmemory	The files in this directory are code for shared memory access related to motor control

Project	
gcc\CN032_AC_Servo_Solution_CPU0	RZ/T2M CPU0 project of CN032 AC Servo Solution firmware for RAM debugging with GCC
gcc\CN032_AC_Servo_Solution_CPU0_serialboot	RZ/T2M CPU0 project of CN032 AC Servo Solution firmware project for serial boot with GCC
gcc\CN032_AC_Servo_Solution_CPU0_serialboot\CPU1_boot_bin	RZ/T2M CPU1 application binary file copied from CPU1 project when building the project
gcc\CN032_AC_Servo_Solution_CPU1	RZ/T2M CPU1 project of CN032 AC Servo Solution firmware for RAM debugging with GCC
iccarm\CPU0	RZ/T2M CPU0 project of CN032 AC Servo Solution firmware for RAM debugging with IAR
iccarm\CPU0_serialboot	RZ/T2M CPU0 project of CN032 AC Servo Solution firmware project for serial boot with IAR
iccarm\CPU0_serialboot\CPU1_boot_bin	RZ/T2M CPU1 application binary file copied from CPU1 project when building the project
iccarm\CPU0_serialboot\Flashloader_AT	The files in this folder provide bootstrapping and code transfer from the SPI Flash to the device memory for IAR.
iccarm\CPU1	RZ/T2M CPU1 project of CN032 AC Servo Solution firmware for RAM debugging with IAR

– “FA-Coder is a trademark of Tamagawa-seiki Corporation."

3.2 AC Servo Solution Kit (RZ/T2L)

The below table shows the file configuration of the CN032 AC Servo Solution Firmware for AC Servo Solution Kit (RZ/T2L). Table 3-3 shows the file configuration of firmware for RZ/T2L.

Table 3-2 File configuration of firmware for RZ/N2L

File	Description
Common	
cg_src	They initialize the different RZ/T2L peripherals involved in the firmware operation.
ethercat\ application/ecat	The files in the directory are code specific for EtherCAT CiA402 communication
ethercat\inc\r_ecat_config.h	The file is header file for EtherCAT module device driver
ethercat\src\r_ecat\r_ecat_setting_rzt2.c	The file is code for EtherCAT module device driver
ethercat\src\r_ecat\hal	The files are header file and code for the hardware access layer of EtherCAT
ethercat\src\r_ecat\phy	The files are header file and code for EtherCAT PHY device driver
rzt\fsp\r_ecat\utilities\batch_file\apply_patch.bat	The batch file to apply a patch file on the Slave Stack Code
ethercat\src\r_ecat\utilities\batch_file\ CN032_AC_Servo_Solution_CiA402.patch	The patch file to be applied on the Slave Stack Code
ethercat\src\r_ecat\utilities\esi\Renesas_CN032_AC_Servo_Solution_CiA402.xml	EtherCAT Slave Information file
ethercat\src\r_ecat\utilities\ssc_config\CN032 AC Servo Solution EtherCAT CiA402.esp	The project file to execute SSC Tool

inc\apl\m_common.h	CN032 AC Servo Solution Firmware header file. Includes the motor data structure and signatures of all global functions.
inc\common\platform.h	Common type definitions header file
inc\dsm	The files in this directory are header files for delta-sigma interface driver
inc\encoder	The files in this directory are header files specific for SCI UART to communicate with Tamagawa encoder
inc\qspi	The files in this directory are header files for QSPI memory access driver
inc\shm	The files in this directory are header files for shared memory access driver
lib\gcc\libVPG.a	The Velocity Profile Generation Library for GCC
lib\r_vpg.a	The Velocity Profile Generation Library for IAR
src\apl\m_commands.c	The code for all host commands that can be invoked
src\apl\m_commutation.c	The code for the motor commutation algorithms such as Space Vector Modulation, Hall-based Trapezoidal Commutation
src\apl\m_control.c	The real-time algorithms control execution branches dependent on different state and configuration options
src\apl\m_homing.c	The state machine implementing the homing algorithm
src\apl\m_interlocks.c	The functions checking various interlock conditions.
src\apl\m_interpreter.c	The command parser for the ASCII commands and the command decoder for the binary packets.
src\apl\m_phasing.c	The functions implementing the different phasing algorithms.
src\apl\m_pid_calc.c	The Position control loop algorithm implementation
src\apl\m_pos_read.c	The encoder position reading control algorithm
src\apl\m_recorder.c	The data collection functions and the start / stop triggers evaluation
src\apl\m_vpg_trap.c	The Trapezoidal Velocity Profile Generator
src\drv\m_biplane.h	Macro definitions specific for the hardware of the Solution and the RZ/T2L device
src\drv\m_rzt2l.c	Code specific for the hardware on the Solution and the RZ/T2L device
src\drv\qspi	The files in this directory are code specific for QSPI flash access
src\drv\dsm	The files in this directory are code specific for delta-sigma modulated interface
src\drv\shm	The file in this directory is code specific for shared memory access driver
src\encoder	The files in this directory are code specific for SCI UART to communicate with Tamagawa encoder
src\sharedmemory	The files in this directory are code for memory access shared between motor control processing and EtherCAT communication processing

Project	
gcc\CN032_AC_Servo_Solution	RZ/T2L project of CN032 AC Servo Solution firmware for RAM debugging with GCC
gcc\CN032_AC_Servo_Solution_serialboot	RZ/T2L project of CN032 AC Servo Solution firmware project for serial boot with GCC
iccarm\ram_exe	RZ/T2L project of CN032 AC Servo Solution firmware for RAM debugging with IAR
iccarm\flash_boot	RZ/T2L project of CN032 AC Servo Solution firmware project for serial boot with IAR
iccarm\flash_boot\Flashloader_AT	The files in this folder provide bootstrapping and code transfer from the SPI Flash to the device memory for IAR.

3.3 AC Servo Solution Kit (RZ/N2L)

The below table shows the file configuration of the CN032 AC Servo Solution Firmware for AC Servo Solution Kit (RZ/N2L). Table 3-3 shows the file configuration of firmware for RZ/N2L.

Table 3-3 File configuration of firmware for RZ/N2L

File	Description
Common	
cg_src	They initialize the different RZ/N2L peripherals involved in the firmware operation.
ethercat\ application/ecat	The files in the directory are code specific for EtherCAT CiA402 communication
ethercat\inc\r_ecat_config.h	The file is header file for EtherCAT module device driver
ethercat\src\r_ecat\r_ecat_setting_rzt2.c	The file is code for EtherCAT module device driver
ethercat\src\r_ecat\hal	The files are header file and code for the hardware access layer of EtherCAT
ethercat\src\r_ecat\phy	The files are header file and code for EtherCAT PHY device driver
rzt\fsp\r_ecat\utilities\batch_file\apply_patch.bat	The batch file to apply a patch file on the Slave Stack Code
ethercat\src\r_ecat\utilities\batch_file\ CN032_AC_Servo_Solution_CiA402.patch	The patch file to be applied on the Slave Stack Code
ethercat\src\r_ecat\utilities\esi\Renesas_CN032_AC_Servo_Solution_CiA402.xml	EtherCAT Slave Information file
ethercat\src\r_ecat\utilities\ssc_config\CN032 AC Servo Solution EtherCAT CiA402.esp	The project file to execute SSC Tool

inc\apl\m_common.h	CN032 AC Servo Solution Firmware header file. Includes the motor data structure and signatures of all global functions.
inc\common\platform.h	Common type definitions header file
inc\dsm	The files in this directory are header files for delta-sigma interface driver
inc\encoder	The files in this directory are header files specific for SCI UART to communicate with Tamagawa encoder
inc\qspi	The files in this directory are header files for QSPI memory access driver
inc\shm	The files in this directory are header files for shared memory access driver
lib\gcc\libVPG.a	The Velocity Profile Generation Library for GCC
lib\r_vpg.a	The Velocity Profile Generation Library for IAR
src\apl\m_commands.c	The code for all host commands that can be invoked
src\apl\m_commutation.c	The code for the motor commutation algorithms such as Space Vector Modulation, Hall-based Trapezoidal Commutation
src\apl\m_control.c	The real-time algorithms control execution branches dependent on different state and configuration options
src\apl\m_homing.c	The state machine implementing the homing algorithm
src\apl\m_interlocks.c	The functions checking various interlock conditions.
src\apl\m_interpreter.c	The command parser for the ASCII commands and the command decoder for the binary packets.
src\apl\m_phasing.c	The functions implementing the different phasing algorithms.
src\apl\m_pid_calc.c	The Position control loop algorithm implementation
src\apl\m_pos_read.c	The encoder position reading control algorithm
src\apl\m_recorder.c	The data collection functions and the start / stop triggers evaluation
src\apl\m_vpg_trap.c	The Trapezoidal Velocity Profile Generator
src\drv\m_biplane.h	Macro definitions specific for the hardware of the Solution and the RZ/N2L device
src\drv\m_rzn.c	Code specific for the hardware on the Solution and the RZ/N2L device
src\drv\qspi	The files in this directory are code specific for QSPI flash access
src\drv\dsm	The files in this directory are code specific for delta-sigma modulated interface
src\drv\shm	The file in this directory is code specific for shared memory access driver
src\encoder	The files in this directory are code specific for SCI UART to communicate with Tamagawa encoder
src\sharedmemory	The files in this directory are code for memory access shared between motor control processing and EtherCAT communication processing

Project	
gcc\CN032_AC_Servo_Solution	RZ/N2L project of CN032 AC Servo Solution firmware for RAM debugging with GCC
gcc\CN032_AC_Servo_Solution_serialboot	RZ/N2L project of CN032 AC Servo Solution firmware project for serial boot with GCC
iccarm\ram_exe	RZ/N2L project of CN032 AC Servo Solution firmware for RAM debugging with IAR
iccarm\flash_boot	RZ/N2L project of CN032 AC Servo Solution firmware project for serial boot with IAR
iccarm\flash_boot\Flashloader_AT	The files in this folder provide bootstrapping and code transfer from the SPI Flash to the device memory for IAR.

4. Firmware Architecture

4.1 Overview

CN032 AC Servo Solution Firmware architecture is designed as a set of modules with specific purpose, operating on data structures passed as arguments. The use of global variables is avoided whenever possible. The functions are invoked in a strictly defined priority and context. This is intended to guarantee the deterministic and robust operation of the control algorithms.

The firmware operations are partitioned in different domains based on their need for deterministic behavior. The figure bellow illustrates these domains:

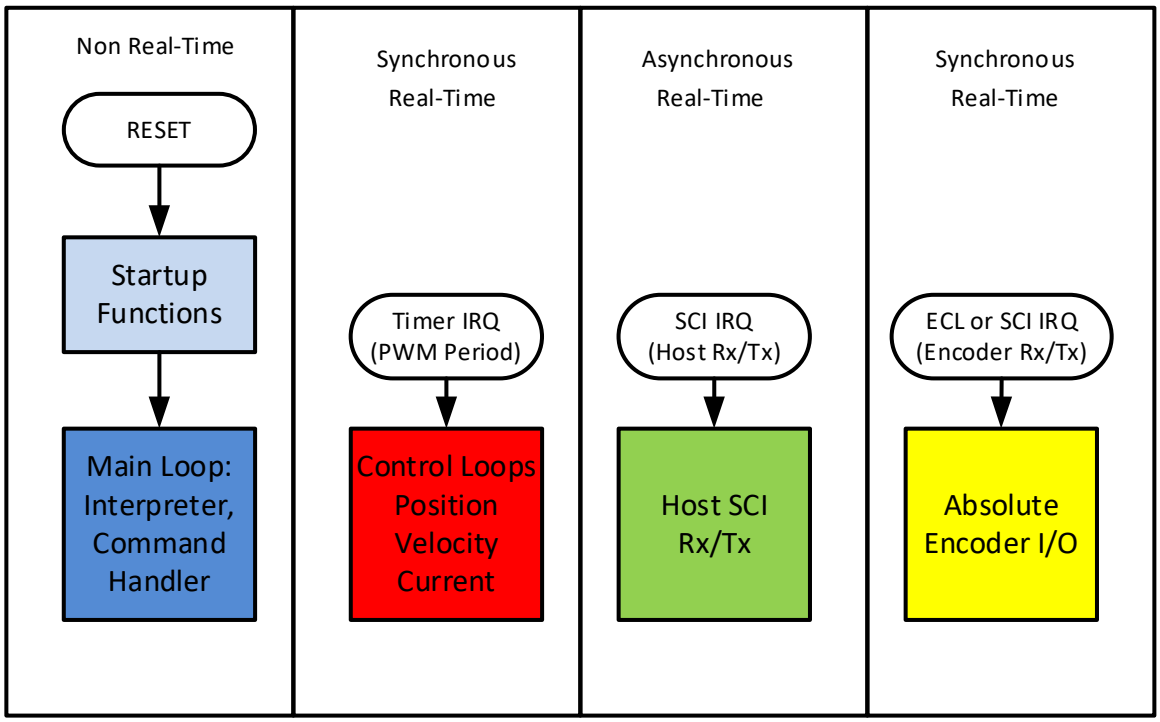


Figure 1 Firmware code execution domains.

The operation of the above functions at run-time can be presented in a time diagram that shows their execution flow. The communication tasks overlap with the others because the RZ/T2M, RZ/T2L or RZ/N2L device employs a dedicated hardware block or SCI communication to interface with the absolute encoders and a FIFO buffer to serve the SCI communications with minimum CPU participation.

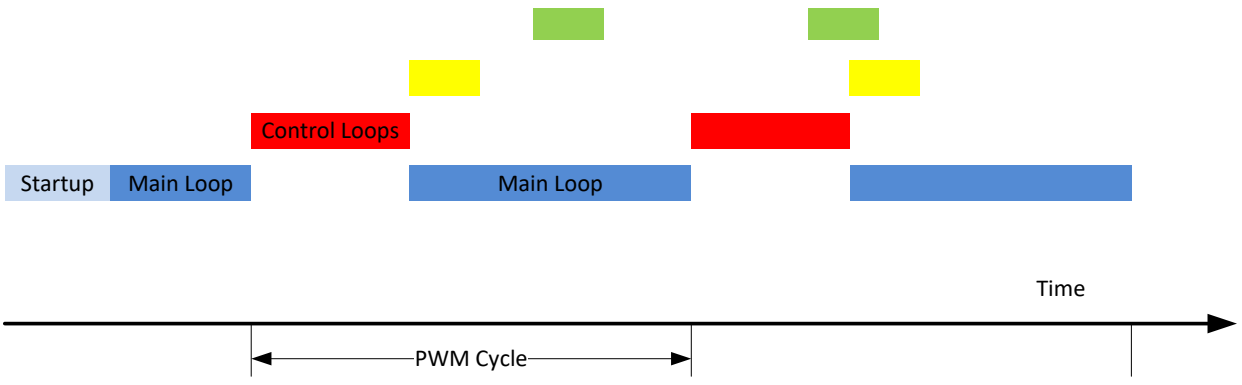


Figure 2 Scheduling of firmware tasks (not to scale)

The coordination between the different functional blocks is implemented with the use of shared data structures that encapsulate the information specific for each motor and each communication interface. In the figure below, the arrows represent the data flow between the functional blocks (in rectangles) and the data structures that they share (in ovals).

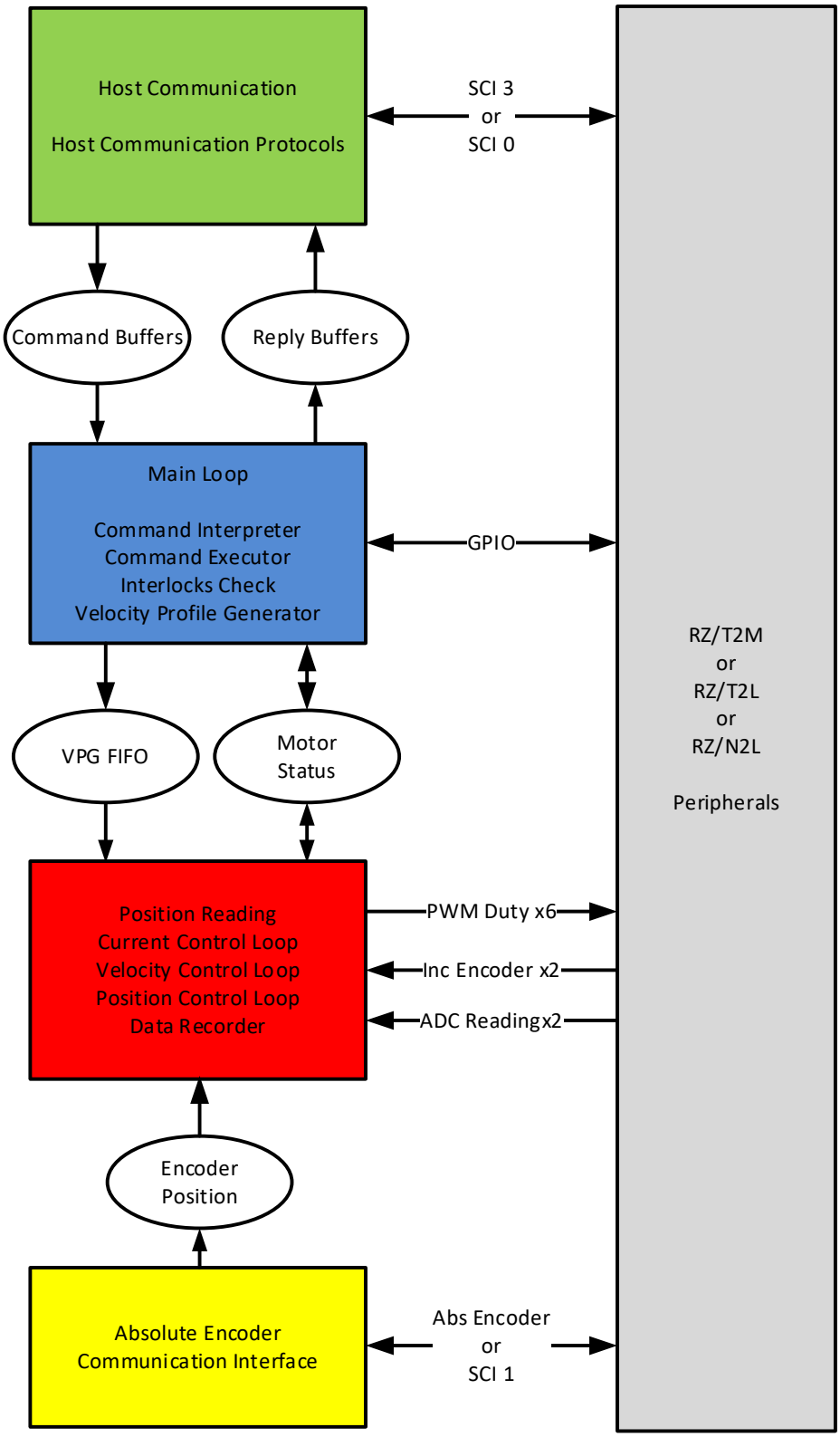


Figure 3 Data flow

4.1.1 Startup Functions

The Startup functions are executed upon reset of the device. They are executed only once and are not dependent on any timing constraints. The main phases of the RZ/T2M, RZ/T2L or RZ/N2L device initialization and the related functions are described below:

Startup Operation	Description
Peripheral Initialization	The second phase of device initialization invokes the functions that configure the peripherals needed by the motion control firmware. The generated files are stored in the folder **cg_src**.
Data Structures Initialization **m_startup()**	The firmware initialization consists of setting up default values to the static data structures, initializing the pointers to the hardware registers to be used by the control algorithms, setting the default state of the GPIOs. The initialization phase completes by loading the programmable encoder protocol configuration based on the selected type. This initialization phase is implemented in the function **m_startup()** in the **src\m_rzt.c** file

4.1.2 Non-Real-Time Functions

The non-real time functions are executed from the context of an infinite loop that begins execution after the firmware initialization is completed. The functions executed are described in the table below:

Non-Real-Time Function	Description
Interptet and Execute Host Command **m_interpreter()**	The interpreter function looks up the host ASCII command in the command dictionary and finds the appropriate command function that needs to be invoked. Alternatively, it directly sets or gets a value when the host command is only requesting parameter change or report.
Run Velocity Profile Generation **vpg_update()**	The Velocity Profile Generator is invoked periodically to generate set-points required for the execution of a point-to-point motion. Since the execution time of the algorithm depends on the phase of the motion profile, the results are buffered and the real-time task that invokes the position control loop function gets the data through dedicated FIFO buffer.
Trigger Data Recorder **m_rec_begin()**	This function checks for a condition that will start or stop the data recording functionality. The condition can be selected from a set of options available to the user.
Check Interlocks **interlocks()**	The interlocks are conditions reflecting errors that indicate faulty hardware or underperforming actuators. They are evaluated periodically and if any erroneous condition is detected the servo control operation is shutoff.
Run Homing State Machine **fsm_homing()**	The Homing State machine executes a series of steps that move a servo axis to known position defined by the location where a designated input (home flag) is triggered and / or the position where the encoder index is captured.

All of the functions described above are invoked from the function **m_foreground()** defined in the file **m_control.c**.

4.1.3 Periodic, Real-Time Functions

The periodic, Real-Time functions are the one executing the motion control algorithms and their determinism is directly dependent on the quality of the motion. For example: any jitter in the position reading timing translates to inaccurate calculation of the derivative component of the position loop affecting the accuracy of the error compensation result.

The real-time operations are executed in the context of the IRQ Handler triggered by the Timer that generates the PWM cycle (62.5 us).

Real-Time Function	Description
Read Encoder Position **pos_read()**	The position reading function is supports different types of encoder interfaces. In case an incremental encoder is configured this function reads the dedicated timer counter registers. In case an absolute encoder type is selected, the function reads the position from a memory location where the last polling request stored the result. Once the absolute encoder position is obtained, the function initiates another polling transaction so its results will be available for the next time slice. The **pos_read()** function is implemented in the file **m_pos_read.c**
Position Control Loop **pos_loop()**	The position control loop implements a classic PID regulator with enhancements such as Velocity and Acceleration Feed Forward, Output Bias and Output Limit control. The **pos_loop()** function is implemented in the file **m_control.c**
Velocity Control Loop **vel_loop()**	The speed control loop performs PID control from the difference between the target speed and the current speed based on the output result of the position control loop. The **vel_loop()** function is implemented at the end of **pos_loop()**
Read ADC Values **crnt_read()**	The current feedback is intended for implementation of the current loop control algorithm as well as the evaluation of the interlock that tracks the motor overload and the amplifier overload. The reading of the ADC values is hardwired to start at the end of each PWM cycle. This is needed to eliminate the switching noise impact on the ADC operation. The **crnt_read()** function is implemented in the file **m_rzt.c**
Current Control Loop **crnt_loop()**	The current loop control loop uses the information from the ADC feedback and the encoder position to calculate the direct and quadrature currents creating torque generating magnetic flux. The current PI regulators operation can be disabled, this algorithm also invokes the Space Vector Modulation function that produces the duty cycles for each of the motor phases. The function operates in different modes depending on the selected mode of motor phasing and the phasing status. The **crnt_loop()** function is implemented in the file **m_control.c**
Recorder **m_recorder()**	The recorder function is an important feature that enables the real-time data collection for the purpose of tuning motion control parameters or troubleshooting dynamic performance of the firmware. The recorder stores the current values of up to four variables in circular buffers. The start and stop of the data recorder are configurable by the host computer.

The **m_recorder()** function is implemented in the file **m_recorder.c**

4.1.4 Communication Functions

The first type of communication functions is executed in response to received commands from the host computer. They are invoked by interrupt handler signaling the reception of a new data from the host or signaling ability to send more data to the host computer.

The second type of communication functions handles the interface with the absolute encoders. Typically, they originate request to get the current position periodically.

Communication Function	Description
Data Reception from Host **m_rx_interrupt()**	The data reception function is serving the interrupt requests from the SCI and handles the specifics of the host communication protocol. This function recognizes the type of the command and the decoding algorithm required. In case of processing binary packet protocols, it also handles the checksum calculation and error handling. This function handles concurrent command requests by maintaining individual buffers for each physical interface. The command interpreter is invoked along with pointer to the command data structure that includes command request and reply to buffers. This enables the concurrent support of different host interfaces. The data reception function is implemented in the file **m_rzt.c**
Data Transmission to Host **m_tx_interrupt()**	The Data Transmission function communicates the result of the command request back to the host. It utilizes the SCI FIFO buffers to minimize the CPU participation in the communication task. The data transmission function is implemented in the file **m_rzt.c**
Polling Absolute Encoder Data	The encoder polling is intended to provide up to date position feedback to the control algorithms. This operation is facilitated by dedicated and configurable hardware block in the RZ/T2M device and does not involve the main CPU. The polling is initiated by the real-time control task, The result of the encoder position polling is stored in a shared memory to be used on the next time slice. The encoder interface functions for RZ/T2M are implemented in the files with the names corresponding to the encoder communication protocol under the folder **src\encoder**. The encoder interface functions for RZ/T2L and RZ/N2Lare implemented in the files with the names corresponding to the encoder communication using SCI UART under the foloder **src\drv\sci_encoder**.

4.2 Data Types

CN032 AC Servo Solution Firmware defines all data types in the file **m_common.h**. The table below describes the most important data types and their purpose:

Data type	Description
TReg32	This type represents 32-bit value as individually accessible two 16-bit values and as four 8-bit values.
TMotionParams	This type encapsulates all motion parameters required for the definition of a point-to-point motion such as target position, velocity, acceleration, jerk, and velocity profile mode
TMotionProfile	This type keeps the motion profile state at specific time and includes snapshot of the velocity profile generation state, position, velocity, acceleration and stopping distance.
TPosVel	This data structure keeps a pair of position and velocity used by the streaming of velocity profile from the host computer (PVT streaming)
t_motor_pars	This type combines all persistent motor parameters.
t_motor	This is main data structure used by the firmware to access all persistent and run-time parameters required for the control of one servo motor. It simplifies the control of multiple motors by instantiating multiple data structure of this type. All motion control functions operate on this data structure by receiving a pointer to specific motor instance as a first parameter.
t_trace	This data structure encapsulates all settings that control the operation of the data recording algorithms.
t_console	This data structure is intended to encapsulate the communication link between the host computer and each communication interface on the Biplane board. This data structure includes command buffer, reply buffer and the pointers that the interrupt handlers use to access the buffers.
t_command	This data type is defined in the m_interpreter.c file where the commands are defined, and the interpreter code is implemented. The structure associate's pointer to variable or function with the name of the ASCII command used to expose them to the host computer.

4.3 Data Structures and Variables

The data structures used by the firmware are instances of the types described in the previous chapter. The table below describes the specific variable instances and their purpose in the firmware:

Data structure instance	Description
t_motor m1	The data structure represents the specific settings for the servo motor supported by the CN032 AC Servo Solution Firmware.
t_console con2	The variable is dedicated for the communication interface providing serial connection to a host computer.
short g_counter	This variable retains the most recent value of the PWM Timer counter at the end of the real-time control algorithms operation. The value useful to monitor the CPU utilization for real-time tasks.
long g_tick	This variable is incremented every time slice. It is used to coordinate the operation between the real-time tasks and the main loop.
short g_suspend	This flag is intended for temporary preventing the real-time functions from executing. Its purpose is to enable time-sensitive operations such as writing flash memory from being affected by the real-time functions.
t_command Commands[]	This is an array of command data structures where all host commands are defined. They consist of ASCII name, type and pointer to either function that executes the command or variable that holds the referenced parameter.

4.4 Enumerations, Macros and Constants

The table below lists all enumerations defined in the CN032 AC Servo Solution Firmware and the description of the individual values:

Enumeration	Description
ETYPE	Defines the different encoder types supported
VPG_STATE	Defines the states of the Velocity Profile Generator (completed, acceleration, deceleration, plateau, etc.)
VPGMode	Defines the different VPG modes – trapezoidal, spline, Bezier
HOMING_STATES	Defines the state machine of the Homing algorithm
CommutationModes	Defines the different operation of the current loop algorithm
ParserStates	Defines the states of the binary protocol parser
ProtocolTypeRequest	Defines the type of host message (ASCII, Packet)
PacketCode	Defines the type of the packet received
PacketError	Defines the possible errors reported by the packet protocol interpreter

5. Initialization and Startup Functions

5.1 Bootloader

The bootloader is intended to initialize the QSPI interface and load the executable code from the flash memory to the RZ/T2M, RZ/T2L or RZ/N2L RAM. Once the data transfer is completed, the execution flow is directed to the **main()** function defined in the file **main.c**.

The files implementing this function are stored in the project folder **FlashLoader_AT.**

5.2 Peripherals Initialization

All peripherals used in the CN032 AC Servo Solution Firmware are initialized the source files are stored in folder **cg_src** and described in the table below:

File Name	Peripheral Description
r_cg_mtu3.c	Configure MTU timers as position decoders for the Incremental Encoder phases
r_cg_s12ad.c	Configure the ADC for the current feedback
r_cg_scifa.c	Configure the SCI communication interfaces
r_cg_poe3.c	Configure the POE3 unit for overcurrent detection

5.3 Firmware Initialization

The firmware initialization is intended to start the operation of the peripherals, initialize the internal data structures, restore the motor parameters from the persistent storage and configure the programmable encoder interface block. This sequence is implemented by the function **m_startup()** defined in the file **m_rzt.c**.

The firmware defines a data structure that defines the motor specific parameters – m1. First, the startup function initializes the members to point at the registers of the timers assigned to the channel.

Next, the startup algorithm invokes the function **setup_motor()** for motor. This function sets the default parameters of all motor structure data members.

Once the data structures are initialized, the startup function starts the timers generating PWM output – **R_MTU3_C3_4_6_7_Start()** and the timers processing the incremental encoder counting – **R_MTU3_Cx_Start()**.

The startup routine continues with re-assigning the SCI3 or SCI0 interrupt vectors to new handlers. This is intended to unify their implementation. The function **setup_scif()** is invoked to configure the data send/receive buffer pointer for SCI interface as well as the data structure (con2) of type **t_console**, that enable its operation. Finally, the SCI interface is enabled by invocation of the function **R_SCI3_Start() or R_SCI0_Start()**.

The startup procedure outputs the character “R” to the SCI3 or SCI0 interface signaling any host connected to the serial interface (UART or RS485), that the module is ready.

Next the startup initializes the data structures required for supporting the data recording operation. This is done by invoking the function **m_TraceSetup()**.

As a last step of the startup function, the algorithm initializes the configurable encoder interface hardware in case the RZ/T2M or SCI1 interface in case the RZ/T2L or the RZ/N2L. This is operation is conditional on the encoder type set to the motor data structures. If it is not incremental type, then the algorithm waits for 150ms and then invokes the function **setup_encoder()**.

With this the firmware initialization is completed and the execution flow is returned to the main loop that repeatedly invokes the function **m_foreground()**.

6. Servo Control Operation

The servo control loop is designed to maintain a desired motor position. This is accomplished by periodically evaluating the difference between the desired and the current position and calculating a compensation to be executed by the motor.

When the servo control loop is first enabled, the desired position is set equal to the current position of the motor. In this state, the algorithm maintains the current position in one place by compensating external disturbances such as gravity, voltage sag or mechanical forces.

When a motion command is issued, a dedicated algorithm (Velocity Profile Generator) calculates the desired of position for each time the servo control loop is executed. The ability of the servo control loop to follow the desired positions creates a motion with programmed and velocity and acceleration.

The operation of the servo control loop and the main functions taking part of it is shown on the figure below:

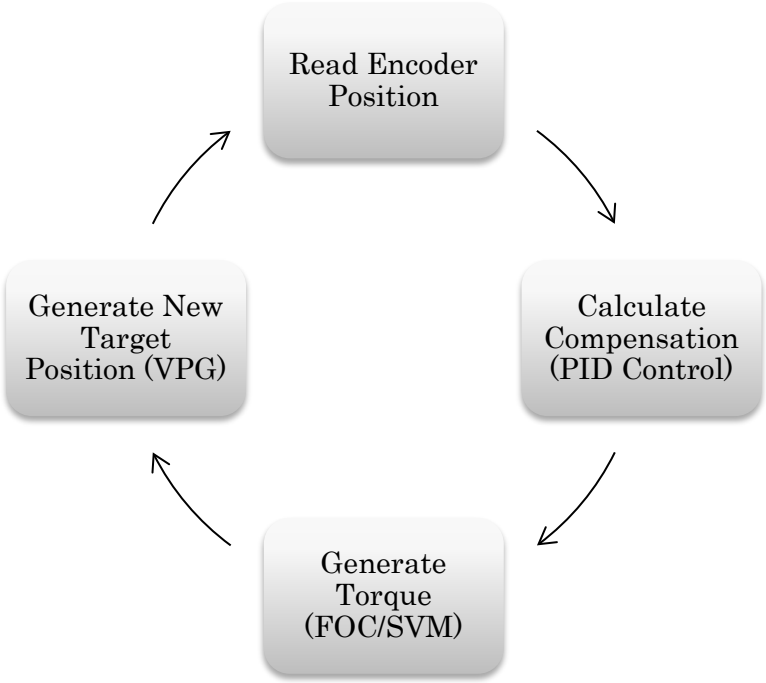


Figure 4 Servo Control Loop

The servo control loop is executed continuously with programmable period. The accuracy of the time between each run is critical for the accuracy of the position error compensation calculations and the ability of the controller to minimize the settle time at the end of any motion.

6.1 Motor Position - Encoder Interface

The purpose of the position feedback is to provide accurate information about the motor position. It is used as an input parameter to the motor control commutation as well as to the position control described below.

CN032 AC Servo Solution Firmware includes variety of encoder feedback options. The possible encoder options are described in the enumeration **ETYPE**. The currently configured encoder type is stored in the **t_motor** data member **encoder_type**. The table below describes the valid settings for this variable:

ETYPE Enumeration	Encoder Type
ETYPE_INCREMENTAL = 0	Incremental encoder – the position change is represented by two phased pulse sequence where the phase between them indicates the direction of motion. A dedicated mode of the timer operation is decoding direction and counting the pulses representing position change. The software reads the current position from the timer counter. It has 16-bit resolution, so the software expands the range to 32 bits.
ETYPE_APE_ENDAT = 1	Absolute encoder with EnDat 2.2 communication protocol.
ETYPE_APE_BISS = 2	Absolute encoder with BiSS communication protocol.
ETYPE_APE_FACODER = 3	Absolute encoder with FA-Coder communication protocol.
ETYPE_APE_AFORMAT = 4	Absolute encoder with A-format communication protocol.
ETYPE_APE_HIPERFACE_DSL = 5	Absolute encoder with Hiperface DSL communication protocol

The incremental encoder feedback position is lost after power cycle, and this requires the execution of algorithms such as Phasing for rotor position identification and Homing – for machine position identification. Another incremental encoder specific feature is the presence of Index pulse – once every revolution. The RZ/T2M timer can use the encoder index as a trigger to capture the position where it has occurred. This operation is hardware defined and does not depend on the speed of motor rotation or any software latency. The index capture mechanism is used for precise initialization of the motor coordinate system as part of the homing procedure.

The absolute encoders eliminate the need for phasing and homing, but require additional configuration parameters (bit rate, bus delay compensation). Depending on the absolute encoder technology they also may require battery backed multi-turn counter and the software must monitor their status for possible battery fault condition. The RZ/T2M hardware blocks perform all interaction with the absolute encoders without the need of CPU involvement.

The absolute encoders offer the ability to store application specific information in an internal EEPROM memory. But the CN032 AC Servo Solution Firmware does not use the methods that access the EEPROM memory of the connected absolute encoder.

Important Notice!

The initialization of the RZ/T2M encoder interface hardware can only be executed once per power cycle. For this reason, the change from one absolute encoder type to another requires intermediate switch to incremental encoder type. Example of switching from BiSS to EnDat encoder:

(Assuming the current encoder type is ETYPE_APE_BISS)

1. **Set the encoder to ETYPE_INCREMENTAL**
2. **Save the motion parameters to Flash memory**
3. **Restart firmware**
4. **Set the encoder to ETYPE_APE_ENDAT**
5. **Save the motion parameters to Flash memory**

The change between absolute and incremental encoder does not require power cycle.

6.2 Motor Control - Torque Generator

CN032 AC Servo Solution Firmware includes functions supporting only main category of motors –Brushless DC Motors or AC Servo Motors. The type of the motor being controlled is defined by the motor control structure data member **motor_type**. The valid setting is described in the table below:

motor_type setting	Motor Type
3	BLDC / PMSM 3-Phase Motor

The motor control algorithms are intended to generate motor torque proportional to the input parameter.

The Brushless DC (BLDC) motor category includes Permanent Magnet Synchronous Motors (PMSM) as well as Linear Motors and AC Servo Motors. All of them share the same principle of torque generation created by three-phase stator windings and permanent magnet rotor with different number of pole pairs.

The three-phase motors are controlled by creating a three-phase voltage that produces magnetic flux with desired magnitude and orientation. This flux can be represented as Space Vector and the process of its calculation is called Space Vector Modulation. It takes the two flux components – magnitude and angle and produces three phase PWM duty cycle numbers stored in the Timer registers to generate the designed phase voltage.

The torque generating flux angle is always dependent on the current position of the rotor. When the controller is restarted, the rotor position is not known (assuming an incremental encoder is used). For this reason, a Phasing algorithm (described in the following chapter) needs to be executed. Until it completes, the motor control can only rely on the rotor position feedback provided by the Hall Sensors (when they are available). In this mode, the motor control is implemented by generating voltage vector in one of the six possible angles identified by the Hall sensors. The figure below represents the structure of the motor control algorithm in this mode:

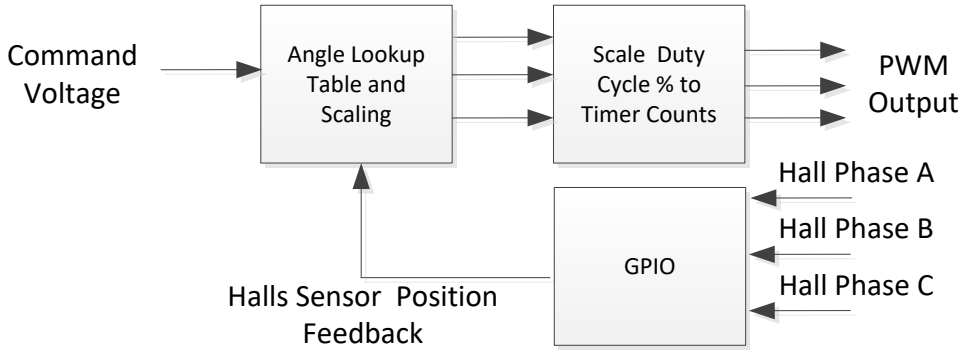


Figure 5 Hall Sensors Based Voltage Control

The maximum torque is generated when the flux is oriented at 90 deg. with respect to the rotor N-S poles. For this reason, the angle of the flux is taken from the current position of the rotor within one electrical cycle. The electrical cycle is equal to the number of encoders counts per mechanical revolution divided by the number of the rotor pole pairs.

The input value for the torque generating algorithm can be interpreted as desired voltage or current. The use of input voltage is simple because the only information required for its calculation is the motor position. This is approach can be called open loop voltage control – it is shown on the picture below:

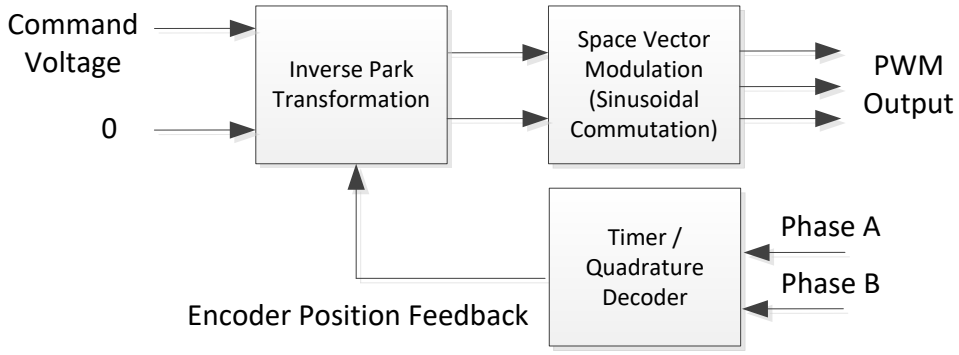


Figure 6 Encoder Based Voltage Control

The voltage control does not reflect the actual torque created because every motor produces Back Electromotive Force (BEMF). The BEMF voltage is proportional to the speed of the motor and the number of the windings of each phase. As a result, the effective voltage applied to the motor windings is the difference between the controller PWM output voltage and the BEMF generated voltage. Subsequently, the current flowing through the motor windings is dependent on the speed of the motor.

The Field Oriented Control (FOC) algorithm is developed to eliminate the significance of the motor speed to the generated torque. This is accomplished by implementing current feedback that provides measure for the actual current (and torque) being produced. The FOC algorithm structure is described in the diagram below:

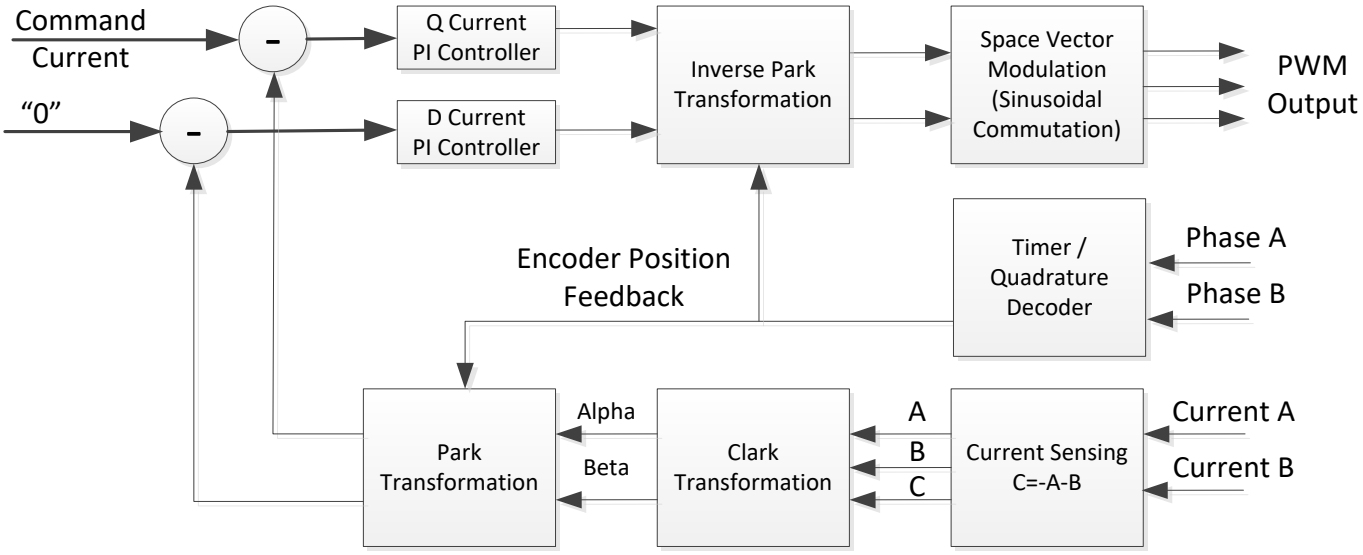


Figure 7 Field Oriented Control Structure

The currents of two of the three phases are sampled continuously by the controller ADCs. The third phase current is reconstructed (Ia + Ib + Ic = 0). The Clark and Park transformations are calculating the components of the torque vector as two orthogonal vectors of representing two current currents:

- Quadrature current – it represents the torque generating flux, perpendicular to the N-S poles of the rotor.
- Direct current – it represents the heat generating force – it should be always 0.

Each of the two currents is compared with it set points and the error is compensated by Proportional – Integral (PI) regulators. The described FOC algorithm is implemented by the function **commutate_foc()** in the file **m_control.c**. The algorithm uses the following motor parameters as input values:

t_motor data member	Description
counts2rad	Coefficient representing the resolution of the encoder per one electrical cycle.
phase_angle	This parameter represents the orientation of the flux vector with respect to the rotor North pole.
p_iu	Pointer to the variable holding the U-phase sampled current
p_iv	Pointer to the variable holding the V-phase sampled current
foc_id	Calculated direct current
foc_iq	Calculated quadrature current
foc_id_err	Calculated direct current error
foc_iq_err	Calculated quadrature current error
foc_vd	Calculated direct voltage
foc_vq	Calculated quadrature voltage
foc_alpha	Alpha component of the voltage vector
foc_beta	Beta component of the voltage vector

The final step of the motor control algorithm execution is the space vector modulation. It is implemented by the function **commutate_svm()** in the file **m_commutation.c**. This function takes the Alpha and Beta voltages calculated by the FOC algorithm and returns the corresponding duty cycle for each phase.

The algorithm for torque generation is configurable run-time by the setting of the motor variable **commutation_mode**.

commutation_mode setting	Description
0 (CM_SVM)	Voltage Control with Space Vector Modulation
1 (CM_FOC)	Current Loop with Field Oriented Control
2 (CM_HALLS)	Hall Sensor Based Control
3 (CM_FORCED)	External / User Defined Phase Voltage Setting
4 (CM_ENC_AND_DSM)	Sinusoidal Vector Control with Encoder and Delta Sigma Modulator
5 (CM_ENC_AND_CT)	Sinusoidal Vector Control with Encoder and Current Transducer
6 (CM_CT)	Sinusoidal Vector Control with Current Transducer

6.3 Position Control – PID Regulator

The position control algorithm takes the position error as an input and calculates output result intended to counter this error. The position loop control function is called **pid_calc()** and is implemented in the file **m_pid_calc.c**.

The structure of the PID regulator is shown on the picture below:

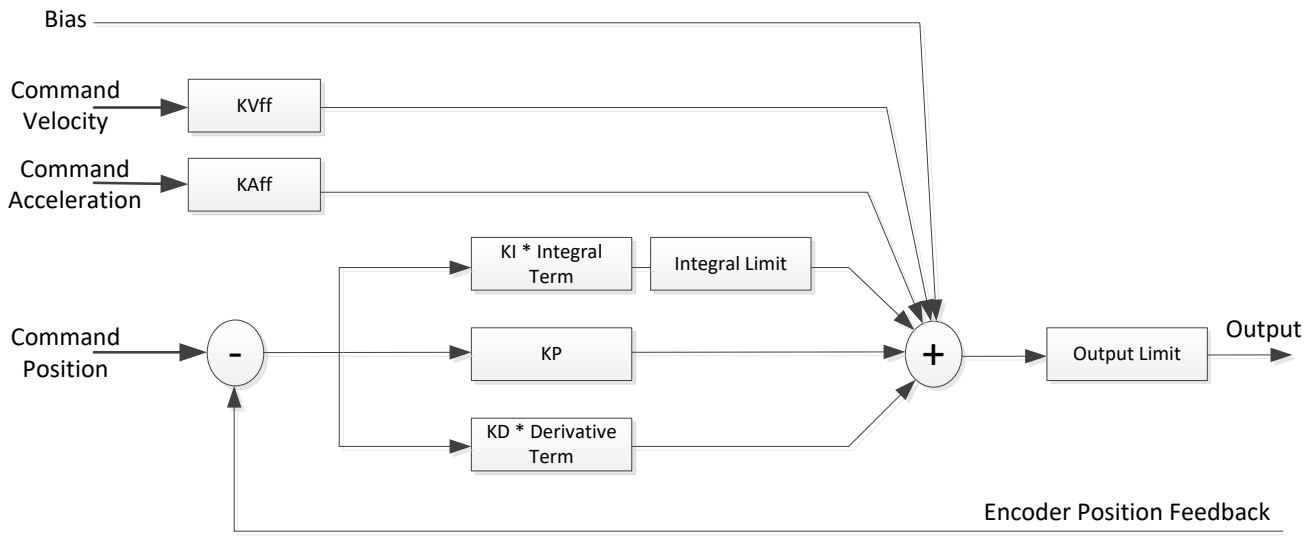


Figure 8 PID Regulator Structure

The position error is calculated and passed as an input parameter to the **pid_calc()** function. The position error is represented in encoder counts.

The configuration parameters for the position control function are stored in the **t_motor** data structure. They are described in the detail in the table below:

t_motor Data Member	Description
crnt_kp	Proportional gain (0 – 32767)
crnt_ki	Integral gain (0 – 32767)
crnt_kd	Differential gain (0 – 32767)
integral_limit	Integration limit (0 – 32767)
crnt_kvff	Velocity feed forward (velocity gain)
crnt_kaff	Acceleration feed forward (acceleration gain)
crnt_bias	Bias – added directly to the output result
cmd_vel	Command velocity
cmd_acc	Command acceleration
pos_loop_limit	Output limit (0 – 32767)

	Temporary variables:
pos_error	Stores the position error from the last invocation of the pid_calc() function.
derivative_err	Stores the calculated derivative of the position error (the difference between the last and the current position error)
integral_err	Stores the calculated integral of position error

The PID control parameters are not directly accessible by the host computer. They are buffered and copied to the variables used by the pid_calc() function only when the servo control is turned on, a new motion is started or an existing motion is stopped. The parameters are updated inside the function m_update() that gets invoked by the functions described above. The picture below shows the parameter buffering mechanism and the related functions.

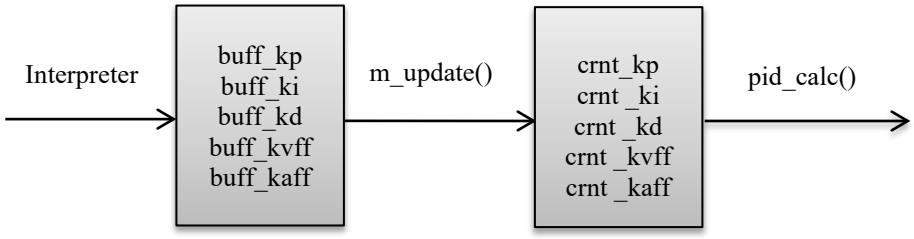


Figure 9 Buffering of Position Loop Parameters

The purpose of the PID gains buffering is intended to update them synchronously and all at the same time. Setting the parameters one at a time or allowing the update to be interrupted may lead to glitch in the output of the control function. This would introduce undesired position error.

The function pid_calc() returns a new output value to be fed as an input to the motor commutation algorithms described in the previous chapter. The output value range is in the range of +/-32767.

6.4 Motion Planning - Velocity Profile Generator

The Motion Planning function defines how the motor will execute a motion from one position to another. First, the algorithm uses the current and the target positions to calculate the desired travel distance. Then it uses the defined maximum acceleration, velocity, and deceleration to calculate the length of the acceleration and deceleration motion phases. The figure below presents typical trapezoidal velocity profile that includes all motion phases:

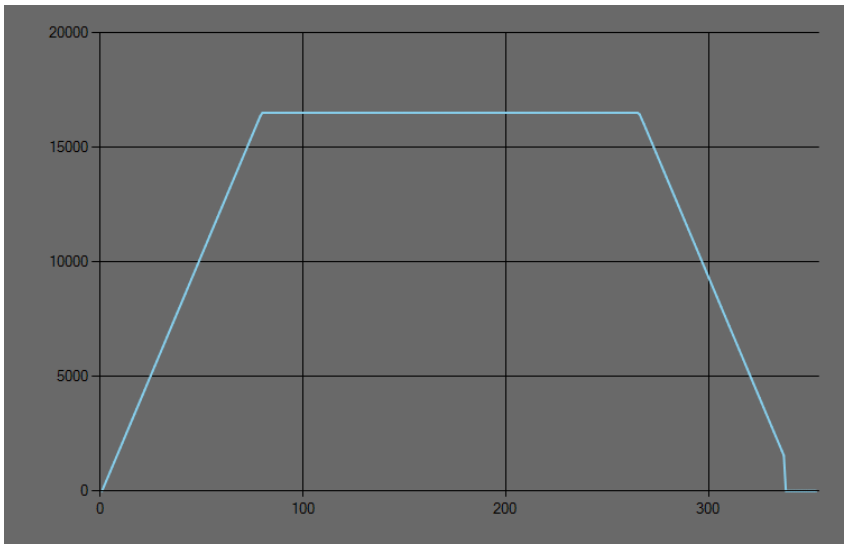


Figure 10 Trapezoidal Velocity Profile

Based on the travel distance and the motion parameters, the algorithm also determines the length of the constant velocity motion phase. In case the travel distance is too short, the motion may never reach the maximum desired velocity.

The diagrams below present an example of a short move where the motion never reaches the maximum velocity.

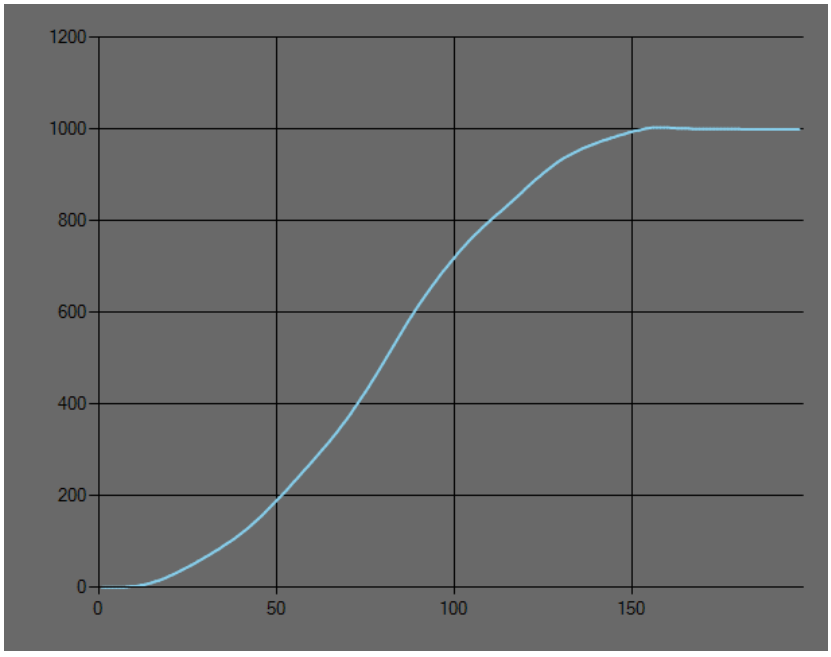


Figure 11 Position Profile for short move to 1000

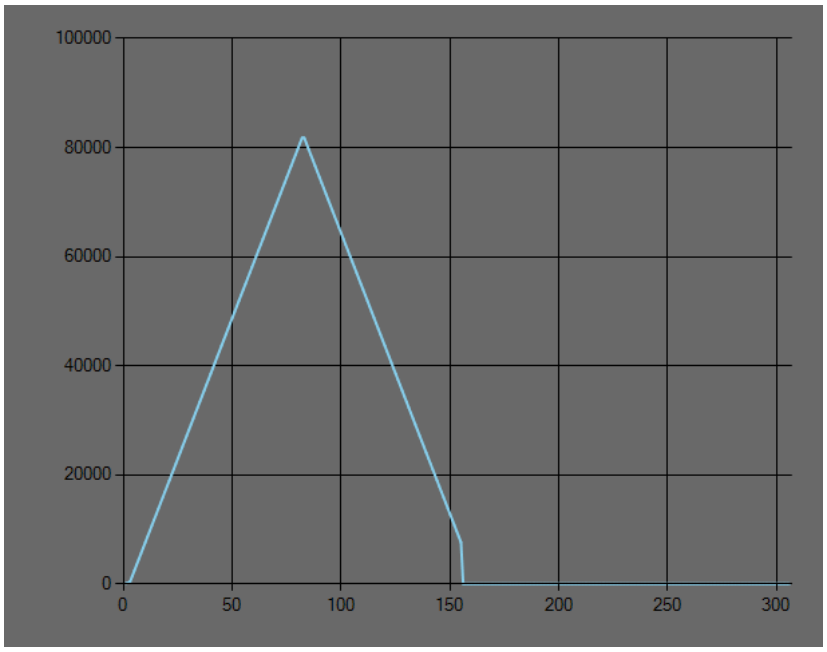


Figure 12 Trapezoidal Profile for a short move

CN032 AC Servo Solution Firmware includes several velocity profile generation algorithms that use different math functions to shape the velocity profile such that it will meet the application specific requirements (*). The figure below shows the velocity profile calculated with the help of a Spline function:

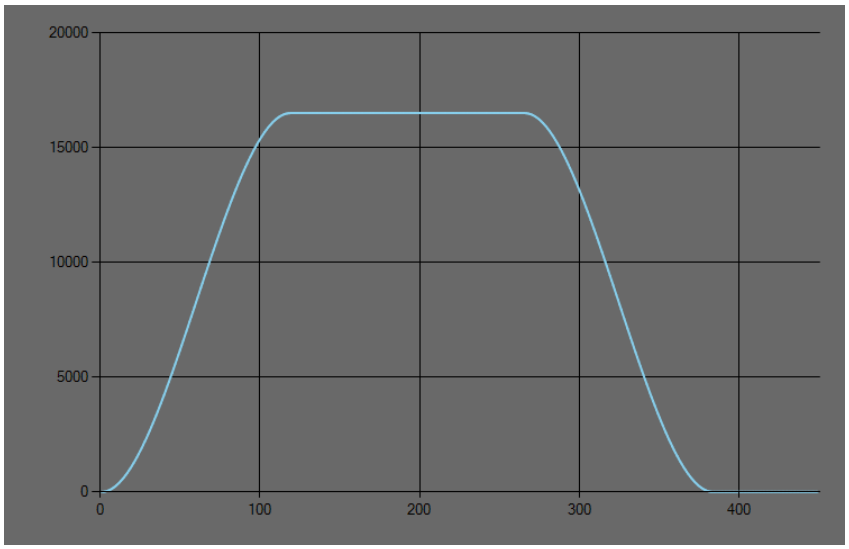


Figure 13 Spline-based Velocity Profile

The smoother velocity profile reduces the vibrations inherent to the trapezoidal profile, but this comes at the expense of longer time to reach a target position. The selection of specific velocity profile is a tradeoff between the time to reach a target position and the settle time at the end of the motion.

The figure below shows a velocity profile generated with the help of a Bezier-curve which enables the individual definition not only of the acceleration and deceleration settings, but of their derivatives (jerk) as well.

(*) CN032 Servo Solution supports the Trapezoidal Velocity Profile only, but not the Spline-based Velocity Profile and Bezier-curve based Velocity Profile.

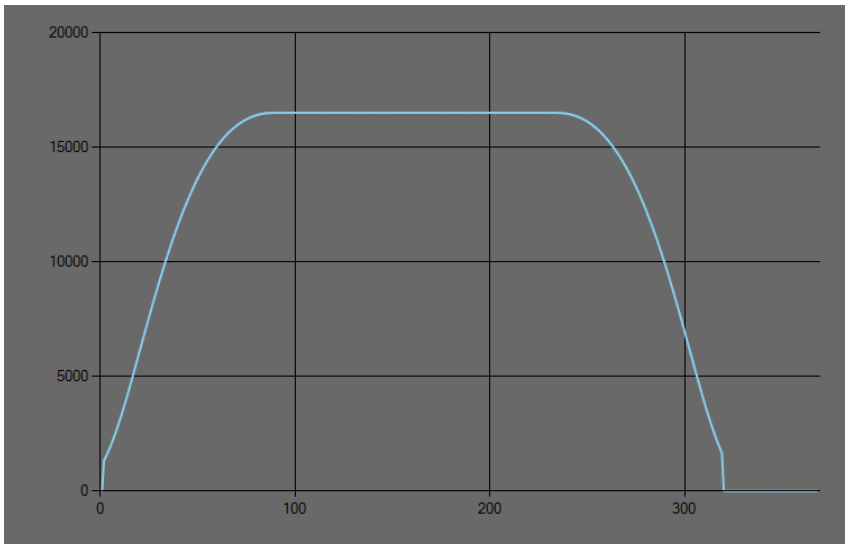


Figure 14 Bezier-curve based Velocity Profile

All the velocity profiles described above only execute a single motion from one position to another. The velocity is zero at the beginning and at the end of this operation. This mechanism is not sufficient for the control of complex mechanism such as robots, gantry stages or CNC machines. These applications require that the host computer generates the complex velocity profiles of the motor. Since the communication bandwidth between the host and the controller is inherently limited, the velocity profiles are presented as sets of Position and Velocity over a fixed time slices (typically 5ms to 20ms). Hence the name Position-Velocity-Time for these profile time. The PVT points are streamed to the controller which in turn execute interpolation algorithm to generate the desired position and velocity set-points each 50 microseconds.

The Velocity Profile Generator operation is defined by state machine with the following states:

State	Value	Description
Idle / Motion Completed	0	Default / Final state.
Acceleration	1	The motion is accelerating
Deceleration	2	The motion is decelerating
Plateau	3	Constant velocity
Streaming / PVT Interpolation	4	Interpolation

The state of the velocity profile generator is stored in the **t_motor** data member **vpg_state**.

When the motion is started all motion preparation is handled by the function **update_ctrl()**. It in turn invokes the startup function of the currently selected velocity profile generator that returns the new VPG state. From this point on the VPG is invoked periodically to produce position and velocity set-points for every cycle of the position control loop. In addition, the VPG returns the new state. Once the motion profile is completed, the state is set to Idle and the periodic VPG function is not invoked anymore.

The table below describes the startup and the periodically invoked functions for each velocity profile type:

VPG Type	Startup VPG Function update_ctrl()	Periodic VPG Function vpg_update()
Trapezoidal	vpg_trap_start()	vpg_trap_next()
Spline-Curve	vpg_spline_start()	vpg_spline_next()
Bezier-Curve	vpg_bezier_start()	vpg_bezier_next()
PVT Interpolation	vpg_pvt_start()	vpg_pvt_next()

The data produced by the VPG is not passed directly to the Position Control Loop. Instead, they are buffered in a dedicated FIFO buffer. This mechanism allows the asynchronous execution of the VPG functions with respect to the position control loop operation. This separation serves two goals:

1. The generation of the velocity profile can be very complex and may require calculation ahead of time. The asynchronous approach avoids the need to “oversize” computational budget of the real-time position and current control loops.
2. The separation of the VPG calculation from the real-time control loop allows for easy redesign where the VPG generator is running on another CPU core or even on another network device.

6.5 Motion Control Parameters

The Motion Control Parameters include the settings that define where the motor is going to and how it is supposed to get there. The motion to be executed also depends on the current state of the motion controller – its current position, position error, current velocity, and current state.

6.5.1 Target Position

The target position is expressed in encoder counts. It can be defined explicitly with a request for Absolute Target Position. (ASCII command ABS).

Alternatively, the target position can be defined as a distance relative to the current position (ASCII command REL). This way of defining the target obviously depends on the motor position at the moment the motion is started. Note that the between specifying the relative distance and the moment the motion is started, the current position may change. This in turn will lead to change of the expected target position.

6.5.2 Maximum Velocity

The maximum velocity parameter defines a limit that may or may not be reached depending on the other motion parameters. The velocity parameter units are defined as “Encoder Counts per Position Loop interval”. Since the position loop interval can be very short time (as little as 62.5 microseconds), the velocity value is communicated as a fixed-point number in 16.16-bit format after being multiplied by 65536.

The conversion of the units from Encoder Counts per Second to the CN032 AC Servo Solution Firmware format, the number must be multiplied by the position loop time slice and then multiplied by 65536. For example:

If the position loop is running at 125 microseconds then 5000 enc.counts per second is converted as:

5000 * 0.000125 * 65536 = 40960

The maximum velocity value corresponding to 5000 enc.counts per second is 40960.

6.5.3 Maximum Acceleration and Deceleration

The maximum acceleration and deceleration parameters define a velocity profile slope that may or may not be reached depending on the capabilities of the control hardware and the specific motor load. The acceleration parameter units are defined as “Encoder Counts per Position Loop interval squared”. Since the position loop interval can be very short time (as little as 62.5 microseconds), the acceleration and deceleration values are communicated as a fixed-point number in 16.16 bit format after being multiplied by 65536.

The conversion of the units from Encoder Counts per Second to the CN032 AC Servo Solution Firmware format, the number must be multiplied by the position loop time slice squared and then multiplied by 65536. For example:

If the position loop is running at 125 microseconds then 30000 enc.counts per second squared is converted as:

30000 * 0.000125 * 0.000125 * 65536 = 31

The maximum acceleration value corresponding to 30000 enc.counts per second is 31.

6.5.4 Maximum Acceleration and Deceleration Jerk

The maximum acceleration and deceleration jerk parameters define a velocity profile slope only when a Bezier-curve velocity profile is being used. The jerk units are dimensionless because they define the Bezier-curve tangent orientation as a ratio. The Jerk value is expressed as integer between 0 and 1000.

6.5.5 Motion Start Modes

The point-to-point motion is started with a single function (ASCII command GO). It invokes the function update_ctrl() which disables the interrupts and copies all motion parameters and position loop parameters from their buffered locations to the variables used by the control loop algorithms. This is intended to update all control parameters simultaneously because of their interdependencies.

The trapezoidal velocity profile allows change of its motion parameters on-the-fly. This allows a new motion to be started even before the last one is completed. The new motion may have some or all of its parameters changed.

Alternatively, the PVT streaming mode can initiate a motion after the PVT buffer is filled to a certain level.

6.5.6 Motion Stop Modes

The Motion Stop modes allow the motion to be stopped in a controlled manner even if the target position is not reached. There are three modes of stopping available:

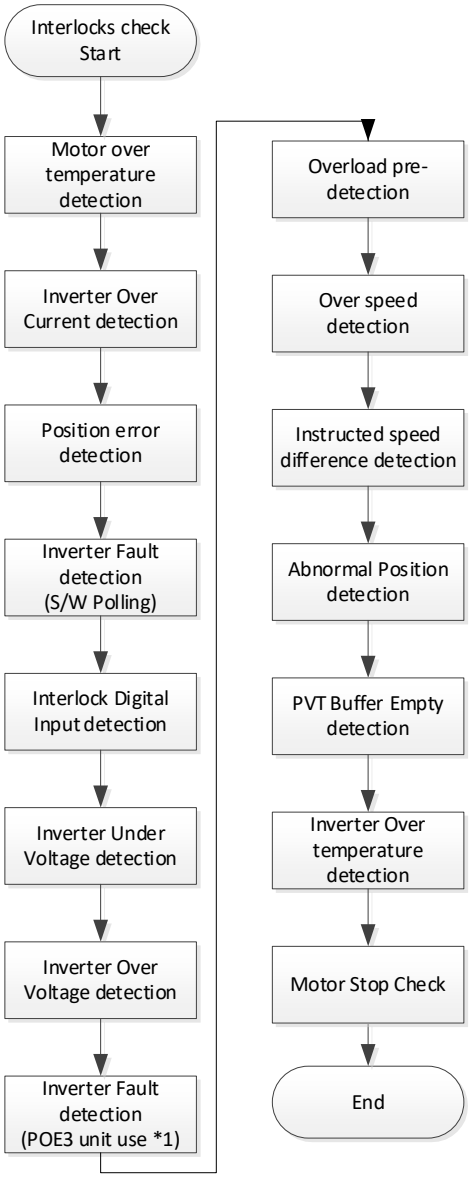
1. Smooth Stop - in this mode the controller executes graceful completion of the current motion. This is accomplished by switching the current velocity profile generator to Trapezoidal and calculating target position based on the programmed maximum deceleration.
2. Abrupt Stop – in this mode the controller uses the deceleration 32 times the maximum defined value. This mode is intended for emergency situations.
3. Servo Off Stop – in this mode the servo control is turned off and the motor windings are shorted by the inverter so that it operates in dynamic braking mode – using the Back EMF as a stopping force.

7. System Control Functions

7.1 Interlocks

The interlocks are hardware defined or software calculated conditions indicating abnormal state of the control system. They are implemented in the function **interlocks()** which is in the file **m_interlock.c**.

The figure below summarizes the algorithm implemented by the **interlocks()** function.



*1 RZ/T2M or RZ/T2L or RZ/N2L Port Output Enable 3 Unit

Figure 15 Interlocks Evaluation

1. Motor over temperature detection

The first interlock is tracking the energy consumed and compares it with the threshold value reflecting the ability of the motor to dissipate heat resulting from its efficiency. The algorithm calculates the square of the measured current consumption. Then it subtracts the nominal current consumption parameter (**I2t_nominal**). The result is integrated in the variable **I2t_integral** and then the integral is compared to configurable limit (**I2t_limit**). Since the overheating is relatively slow process, when the limit is exceeded, the only action is raising a flag in the activity state (**ACT_MtOverTemp**). When the integral value drops below the limit, the flag is cleared.

2. Inverter Over Current detection

The next interlock is comparing the total current consumption to a limit (**tc_limit**) that is exceeded more than a certain time (**tc_limit_time**). The overcurrent interlock is intended to prevent overloading the motor or the machine connected to it in case of mechanical obstruction or other disturbances. When the overload condition is detected, the interlock function sets a flag in the motor activity state (**ACT_OverCurrent**) and turns off the servo control loop. Subsequently, any motion is stopped as well.

3. Inverter Fault detection (S/W Polling)

(This fault detection is disabled in the CN032 AC Servo Solution Firmware)

The PWM amplifier has hardware protection against overcurrent due to faulty cable (shortage) or component. When the Amplifier Fault occurs, the hardware protection turns off the bridge instantly and signals a dedicated digital input. When the interlock function detects that this input is set, it raises a flag in the motor activity state (**ACT_AmpFault**) and turns off the servo control loop.

4. Position error detection

The Position Error interlock tracks the absolute value of the position error. If the position error exceeds the limit (**pos_error_limit**) then a dedicated timer (**pos_error_timer**) starts measuring the duration of the error condition. If it exceeds the configured time (**pos_error_time**), the interlock function stops any motion. Depending on the value of the variable **auto_stop_mode** it may also turn off the servo control. This interlock condition is indicated in the motor activity state by raising the flag **ACT_PosError**.

5. Interlock Digital Input detection

The interlock function can be configured to treat any of the available digital inputs as triggers for an Interlock condition. The inputs to be evaluated are defined as a bitmask in the variable **dinputs_err_mask**. The triggering of the interlock is instantaneous after the AND operation between the inputs byte and the mask is evaluated is non-zero. In response the interlock function turns off the servo control, and the power amplifier output is disabled. The motor activity state is also updated by raising the flag **ACT_Inhibit**.

6. Inverter Under Voltage detection

Set the error detection flag when the inverter bus voltage falls below the threshold value.

7. Inverter Over Voltage detection

Set the error detection flag when the inverter bus voltage exceeds the threshold value.

8. Inverter Fault detection (POE3 unit use)

The Fault signal of the inverter board is monitored with POE3 Unit, the PWM output is automatically switched to high impedance when the Fault signal is detected, and the error detection flag is set.

9. Overload pre-detection

Set the error detection flag when the inverter current value exceeds the threshold value. By setting a lower threshold than "Inverter Overcurrent detection" and masking the servo OFF in this item, it is possible to detect the alarm before the servo turns off due to overcurrent.

10. Over speed detection

Set the error detection flag when the motor speed exceeds the threshold value.

11. Instructed speed difference detection

Set the error detection flag when the change in motor rotation speed exceeds the threshold for 5 seconds.

12. Abnormal Position detection

Set the error detection flag when the position information of the motor deviates from the upper limit threshold or the lower limit threshold range.

13. PVT Buffer Empty detection

Checks the FIFO state of the Velocity Profile Generator every 1.25ms and sets the error detection flag when the state of Empty exceeds the threshold number of times.

14. Inverter Over temperature detection

(This fault detection is disabled in the CN032 AC Servo Solution Firmware)

Set the error detection flag when the temperature of the inverter exceeds the threshold value.

15. Motor Stop Check

Checks the status of each error detection flag and mask setting and stops the motor when the condition is satisfied.

7.2 Data Recording

The Data Recording functions are intended to enable the analysis of the system behavior in real time. It is indispensable tool for analyzing the performance of the different control loops, their configuration parameters, and their efficiency in application specific contest. The data recorder stores up to four user defined parameters in buffers during system operation. The support functions and configuration parameters enable the selection of rate of recording, which variables are recorded, when the recording is started and when it is supposed to end.

The length of the recording as number of samples is defined by the macro **TRACE_BUFFER_SIZE**. By default, it is set to 512. This value can be increased when the application requires longer records and the RAM memory is available. All buffers are combined into a single array named **traceData**[], The data type of the array is short – 16-bit integer. For this reason, when a 32-bit variable is being recorded, its value is split between the fist and the fourth buffers. When the data is reported, it is combined appropriately.

The host specifies the data to be recorded for each channel by using the commands CH1, CH2, CH3 and CH4. These commands are used to set a code representing the data of interest. The first 8 codes are reserved for 32-bit variables. The rest are referencing 16-bit data. The table below defines the correspondence between different data variables and the codes that need to be set for their recording:

CH1, CH2, CH3, CH4 codes	Referenced motor variable
0	Motor position
1	Commanded velocity
2	Commanded acceleration
3	I2t integral
4 - 7	Reserved
8	Position error
9	PID regulator output value
10	Reserved
11	Direct current (heat generating)
12	Quadrature current (torque generating)
13	Direct current error
14	Quadrature current error
15	Raw current A
16	Raw current B
17	PVT FIFO buffer depth
18	FOC voltage output D
19	FOC voltage output Q

20	Real time task timing
21	Phase angle
22	Raw current C
23	Input captured position
24	Position error 2
25	Position control integral error 2
26	Velocity error 2
27	Velocity control integral error 2
28	Direct current control integral error 2
29	Quadrature current control integral error 2
30	Torque estimate
31	Motor electric angle

The start and stop conditions of the data recorder are configured by the ASCII command TRACE (variable **trace.Trigger**). The table below describes the possible settings representing different trigger conditions:

TRACE codes	Trigger Start Condition	Trigger Stop Condition
0	N/A	Stop data recording
1	Start recording immediately.	Stop when the motion is completed. This trigger is useful for examining the end of a motion.
2	Start recording immediately.	Stop when the buffer is full. This trigger is useful for examining the beginning of a motion.
3	Start recording on start of motion.	Stop on end of motion.
4	Start recording immediately.	Stop on input change. The input bit mask is defined in the **trace.Level** variable.
5	Start recording immediately.	Stop on value exceeding the threshold defined in **trace.Level** variable.
6	Start recording immediately.	Stop on value below the threshold defined in **trace.Level** variable.
7	Start on PWM output change.	Stop when the buffer is full.
8	Start recording on input change. Input mask is defined in the **trace.Level** variable.	Stop when the buffer is full.

9	Start on value exceeding the threshold defined in **trace.Level** variable.	Stop when the buffer is full.
10	Start on value below the threshold defined in **trace.Level** variable.	Stop when the buffer is full.

The recorder operates synchronously to the real-time task that executes every 50us. The rate of the recorder can be expressed as multiples of this time interval. The multiple factor is set by the ASCII command TRATE and stored in the variable **trace.RateMult**. For example, if the desired rate of the data recorder is 1ms then the TRATE should be set to 20.

Another variable evaluated during some of the trigger conditions is the ASCII command TLEVEL (variable **trace.Level**). The value of this variable is the threshold that the recorded variable is compared against. Depending on the trigger code, the condition to start recording can test for value either bigger or smaller than the threshold.

The function invoked periodically to test the start trigger condition is **m_rec_begin()**.

The function invoked periodically to test the stop trigger condition and perform the recording is **m_recorder()**.

The recorder mode of operation is reported by the ASCII command TMODE (variable **trace.Mode**). The meaning of the codes stored is described in the table below:

TMODE codes	Modes of operation / status
0	Idle, stops recording if started
1	Recorder is armed – ready to start on beginning of motion
2	Recording in ongoing. Stop once the buffer is full.
3	Recording in circular buffer. Stop once the motion completes.

Once the data recording is completed, the host can use the command PLAY to get content of the recording buffers. The function implementing this request is **m_Play()**. It also formats binary packet response if the invocation context is packet command handler.

7.3 Motor Phasing

The Phasing procedure is intended to identify the absolute angle of the rotor of a brushless motor. This is needed by the control algorithms that calculate the orientation of the torque generating magnetic flux. The execution of the phasing procedure is required after each power cycle when incremental encoder feedback is used. The absolute encoders eliminate the need for such procedure. Still – they need at least once configuration of their phase offset – value that defines the rotor angle within the single turn encoder position.

The selected phasing mode is defined by the value in the motor variable phasing_mode.

The phasing algorithm to be used depends on different factors. The summary below describes the different phasing options as well as the pros and cons of each algorithm.

Phasing Procedure	Description
Forced Phasing **phasing_mode == 0**	In this mode the firmware forms a voltage vector a known angle. It is formed by applying appropriate PWM duty cycle to each of the three phase outputs. The voltage is applied with magnitude defined by the motor variable **motor_power** for a duration defined in the motor variable **phasing_time**. These two variables must be configured so that they will cause the motor to rotate its rotor such that it is oriented along the orientation of the magnetic flux. Once the time expires, the algorithm stores the current position and sets the phase origin 90degrees back from it. This procedure is implemented in the function **forced_phasing()** in the file **m_phasing.c** The pros of this function are its simplicity and robustness. The cons are the small move in random direction the motor would make during the procedure execution. Another disadvantage is that the motor should have no static friction or gravity load that would affect the proper rotor orientation.
Hall Sensor Based Phasing **phasing_mode == 1**	In this mode the phasing depends on scanning the transitions of the hall sensors from one configuration to another. On startup the phase origin is set to one of the 6 possible angles defined by the steady state hall sensor. This allows the motor operation although with less efficiency and noticeable “cogging”. It is because the hall sensors only provide 60 degrees accuracy in determining the steady state rotor orientation. Once the hall sensors report transition from one configuration to another the algorithm infers the rotor position by analyzing the old and the new hall sensor combination. Once this is done the motor status is set to “phased” and the motor commutation is switched to sinusoidal. The algorithm is implemented by the function **hall_phasing()** in the file **m_phasing.c** The pros of this approach are the robustness with regards to any mechanical loads and the lack of unwanted moves before the motion is started for the first time. The biggest cons are the need of hall sensors and the related wiring, cost, and reliability issues.

Phasing Procedure	Description
Dithering Based Phasing **phasing_mode == 2**	The dithering algorithm is derived from the Forced phasing algorithm – identifying the rotor position by observing its position after known flux is applied for a certain time. Unlike the Forced phasing algorithm, the Dithering algorithm does not wait for a certain time – instead it monitors the position change of the rotor. Once the motion direction is detected, the flux orientation is changed so that it causes change in the opposite direction. The magnitude of the flux angle changes is gradually reduced until the motion is no longer detected. The result is motor phasing that only includes small motor vibrations for a short time as part of the phasing. This algorithm has the benefits of the Forced Phasing algorithm but without the drawback of unwanted motion. The cons are the need of carefully tuning the algorithm parameter to match the dynamic characteristics of the mechanical system the motor is attached to. The presence of static friction and gravity load are also undesired. The algorithm is implemented by the function **dither_phasing()** in the file **m_phasing.c**

7.4 Motor Homing

The homing procedure is required in applications that do not have absolute encoders or when the position of the absolute encoder is not calibrated. The execution of the home procedure executes series of moves and reads the feedback from external sensors to establish the origin of the coordinate system of the controlled axis.

The homing procedure can be executed in three different modes depending on the position feedback and home sensor available. They are described in the table below:

Homing Procedure	Description
Index Based	This home procedure is started if the home_mask motor parameter is set to 0. This homing mode is used when a device uses a rotational axis and its coordinate system need to identify the angle where the orientation of the motor shaft is at 0 degree. The use of the index pulse as home position reference is only possible with incremental encoders with index pulse output. Normally it is labeled with Z+/Z-. Note that some encoders do not have differential index output. In this case the Z- input must be biased with 3.3V at the connector. The homing procedure starts a motion and arms the position capture hardware so that it would be triggered as soon as the index pulse is registered. Once the index position is captured, it is subtracted from the current position along with user defined home offset. This effectively adjusts the position offset and establishes the index position (plus the home offset) as new origin (zero) of the axis coordinate system. Once this is done, the axis target position is set to zero and a motion is started.
Hard Stop Based	This home procedure is executed when the **home_mask** motor parameter is set to 3. The hard stop based home procedure is polling the position error continuously. Once it exceeds 100 encoder counts, the current position is considered origin of the axis coordinate system. Similar to the Index based homing, user defined home offset is subtracted from the current position. Once this is done, the axis target position is set to zero and a motion is started.
Home Sensor Based	This homing procedure is executed when the **home_mask** is a non-zero value different than 3. The **home_mask** value is applied to logical AND operation with the digital inputs value. This effectively allows selection of the digital input to be considered “home switch”. The execution of the home procedure starts with motion in direction dependent on the home switch status. The objective of the motion is to cause change in the input state. The triggering of the home switch is the indication of detecting vicinity of the coordinate system origin. Due to the lag in sensing the home sensor trigger (it is polled every 100 microseconds) the accuracy of the home sensor position is not sufficient in high-precision applications. For this reason, the homing continues with the Index Based procedure.

The state diagram shows the states and the transitions between them. The execution flow is defined by the motor variable **home_mask** as described above.

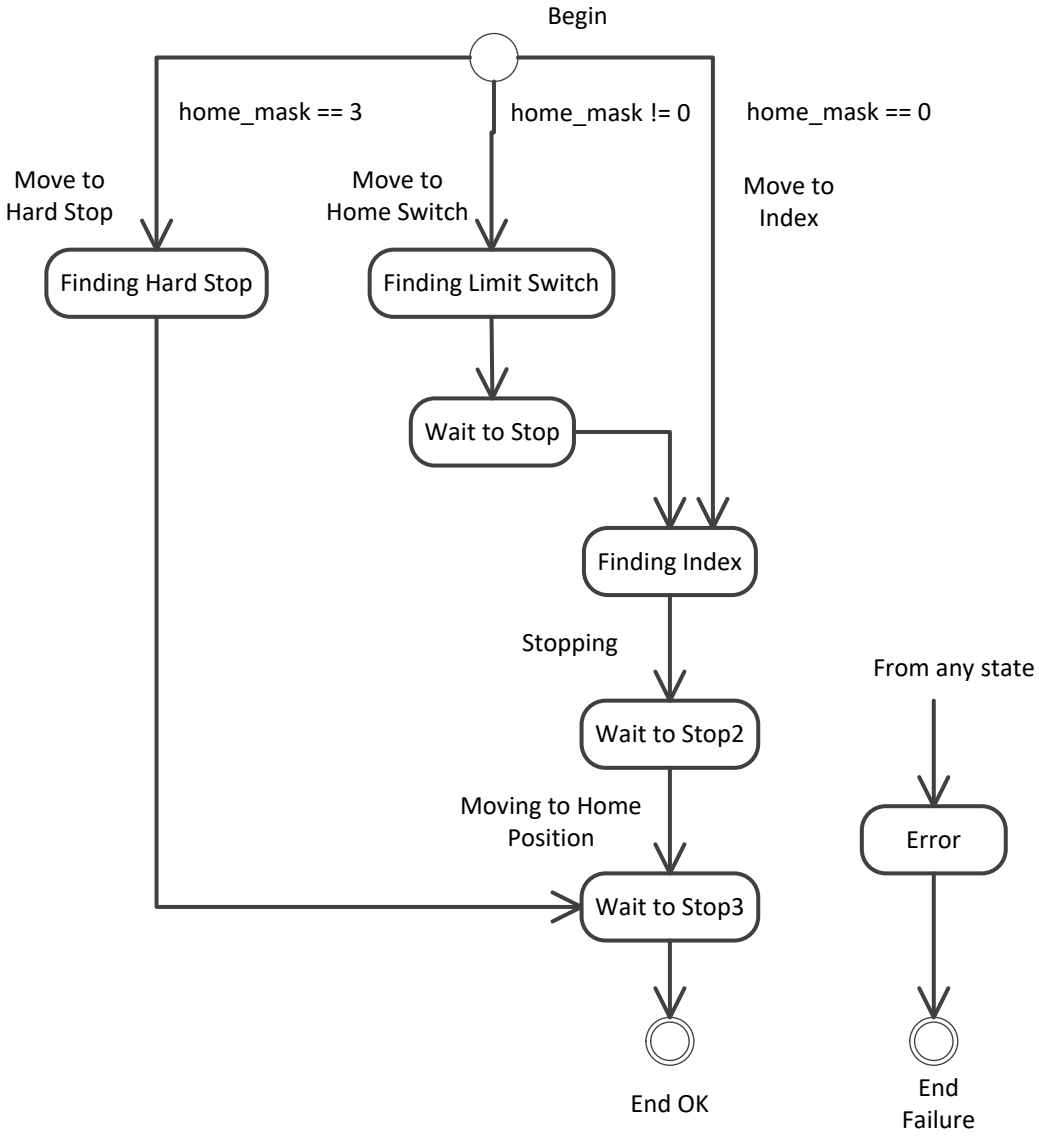


Figure 16 Homing Procedure State Machine

8. Host Communication

8.1 ASCII Communication Protocol

The ASCII protocol is based on commands consisting of ASCII characters terminated with Carriage Return (CR, ASCII 13). The controller responds with an optional data string followed by a prompt.

The ASCII protocols uses the following communication parameters:

115,200 bps, 8 data bits, 1 stop bit, no parity

When the command is accepted the prompt consist of CR, Line Feed (LF, ASCII 10) and Greater Than sign (>). When the command is rejected, the prompt has Question Mark (?) instead of the Greater Than sign.

Example:

POS ; Host command terminated by CR

120 ; Reply Data String

> ; Reply Prompt

The variable names entered at the command prompt report the value of the referenced variable. If a parameter follows the name it is interpreted as a request to set the variable to a new value. Some variables are read-only. An attempt to set a value to them will be reported as invalid command. Examples:

>POS ; Request value of the variable POS

2100

>POS 2000 ; Set POS to a new value

>POS ; Report the new value

2000

>

The full set of ASCII commands is described in the Appendix A to this application note.

8.2 Binary Packet Communication Protocol

Binary Packet Communication Protocol is not supported.

9. Resources

9.1 Hardware

The CN032 AC Servo Solution Firmware is designed to work on the Controller board. It depends on its hardware resources.

9.2 Operating System

The CN032 AC Servo Solution Firmware does not depend on any operating system.

9.3 Memory

The firmware memory blocks occupied for code, constants and uninitialized data are described in the map file.

Appendix A: ASCII Communication Protocol Commands

The table below represents unified enumeration of all configuration parameters, status variables and functions in the Biplane firmware.

R/O = Read Only, R/W = Read / Write, Cmd – Command

Par ID	Name	Byte Size	Access	Description
0	STA	4	R/O	Status word (reported as 4 digit Hex word). The bit flags are documented below.
1	ERR	2	R/O	Position error – the difference between the actual and the expected encoder position. The value is meaningful only when servo control is on.
2	ADC1	2	R/O	Current reading as reported by the ADC channel 1. The correspondence between the number reported and the actual current is hardware dependent.
3	ADC2	2	R/O	Current reading as reported by the ADC channel 2
4	TC	2	R/O	Total current – calculated as a sum of the modulus of ADC1 and ADC2 currents
5	CV	2	R/O	Current velocity as encoder counts per sample interval (the position loop cycle time)
6	PVT	2	R/O	Number of points in the PVT buffer
7	ITIME	2	R/W	Time interval between consecutive PVT points
8	VEL	4	R/W	Command Velocity – (encoder counts per sample interval) * 65536
9	ACC		R/W	Command Acceleration – (encoder counts per sample interval squared) * 65536
10	DEC	4	R/W	Command Deceleration – (encoder counts per sample interval squared) * 65536. If the value is zero then the Acceleration value is used in its place.
11	AJERK	4	R/W	Acceleration Jerk (0 – 1000)
12	DJERK	4	R/W	Deceleration Jerk (0 – 1000)
13	PRO	2	R/W	Velocity profile mode
14	KP	2	R/W	Proportional Gain in the position control loop algorithm (0 – 32767)
15	KI	2	R/W	Integral Gain in the position control loop algorithm (0 – 32767)
16	KD	2	R/W	Differential Gain in the position control loop algorithm (0 – 32767)
17	IL	2	R/W	Integral Limit in the position control loop algorithm (0 – 32767)
18	VFF	2	R/W	Velocity Feed Forward in the position control loop algorithm (0 – 32767)
19	AFF	2	R/W	Acceleration Feed Forward in the position control loop algorithm (0 – 32767)
20	MAX	2	R/W	Maximum position error (0 – 32767)

Par ID	Name	Byte Size	Access	Description
21	ETIME	2	R/W	Maximum position error time (in sample intervals)
22	DS	2	R/W	Position Loop Derivative Sample Interval (sample interval) in multiples of 50us. Setting of 2 indicates 100 us sample interval.
23	MLIMIT	2	R/W	Motor output limit from the position loop PID regulator (0 – 32767)
24	BIAS	4	R/W	Value to be added to the output of the PID regulator continuously
25	ASTOP	2	R/W	Automatic stop mode. Determines the action after position error is exceeded: 0 – stop the motion, 1 – stop the servo control in addition to stopping the motion.
26	PIMODE	2	R/W	Phase initialization mode: 0 – Forced, 1 – Hall Based, 2 – Dithering Based
27	PITIME	2	R/W	Phase initialization time [sample intervals]
28	PIOUT	2	R/W	Phase initialization output (32767 = 100%)
29	PMAP	2	R/W	Phase mapping to PWM output channels. Allows software defined wiring between the controller outputs and the motor windings. Values should range from 0 to 5.
30	PORIGIN	4	R/W	Phase origin – the encoder position within the current motor revolution where the flux is at 0 degree.
31	PCMODE	2	R/W	Phase commutation mode: 0 = voltage controlled Space Vector Modulation, 1 = Field Oriented Control, 2 = Hall based commutation, 3 = Host defined phase angle,
32	PVECTOR	4	R/W	Phase vector orientation times 60 degree. Values should range from 0 to 5.
33	PPAIRS	2	R/W	Pole pairs – this parameter reflects motor rotor construction.
34	PCOUNTS	4	R/W	Encoder counts per electrical cycle (encoder counts per rev divided by the number of pole pairs)
35	ECPR	4	R/W	Encoder counts per revolution – encoder resolution
36	PIOFFS	4	R/W	Phase Initialization Offset – position offset added at the end of the phase initialization procedure
37	PANGLE	4	R/O	The current angle of the magnetic flux
38	PADV	4	R/W	Phase advance gain (not used)
39	VCOMP	4	R/W	Velocity compensation (not used)
40	CLIMIT	2	R/W	Current limit threshold
41	CTIME	2	R/W	Current limit time (in sample intervals)
42	IDM	4	R/O	Direct (heat generating) current
43	IQM	4	R/O	Quadrature (torque generating) current
44	IQERR	4	R/O	Quadrature current error

Par ID	Name	Byte Size	Access	Description
45	QKP	2	R/W	Proportional gain in the Quadrature current control loop
46	QKI	2	R/W	Integral gain in the Quadrature current control loop
47	PCT	2	R/W	Position capture mode (only index capture supported)
48	HMASK	2	R/W	Home switch mask – defines the type of the home algorithm and the specific input wired to home sensor
49	INVERT	2	R/W	Home switch invert mask – defines the inversion of the home switch input so that it will be searched in the correct direction.
50	HINVERT	4	R/W	Hall sensor signal inversion: 0 – no inversion, 1 – inverted
51	HSHIFT	4	R/W	Hall sensor position shift (not used)
52	HPOS	4	R/W	Hall sensor position change – the last position where the hall sensors changed their status
53	EMASK	2	R/W	Digital inputs error mask – defines digital input as an interlock that can trigger motion stop if triggered.
54	ECP	4	R/W	Command position where the following error exceeded the maximum threshold
55	ECV	4	R/W	Velocity at which the following error exceeded the maximum threshold
56	EPO	4	R/W	Actual position where the following error exceeded the maximum threshold
57	U	4	R/W	The output voltage of phase U. (32767 = 100% duty cycle)
58	V	4	R/W	The output voltage of phase V. (32767 = 100% duty cycle)
59	W	4	R/W	The output voltage of phase W. (32767 = 100% duty cycle)
60	TYPE	2	R/W	Module type: 1 = single channel, 2 = dual channel with electronic gearing, 3 = dual channel / two independent motors
61	HTYPE	2	R/W	Hall sensors type (only parallel type supported)
62	HOFFS	4	R/W	Home offset – value added to the zero coordinate at the end of the home procedure.
63	DATA0	4	R/W	User defined data – no specific use in the firmware
64	DATA1	4	R/W	
65	DATA2	4	R/W	
66	DATA3	4	R/W	
67	DATA4	4	R/W	
68	DATA5	4	R/W	
69	DATA6	4	R/W	

Par ID	Name	Byte Size	Access	Description
70	DATA7	4	R/W	
71	PHASES	2	R/W	Defines the motor type: 2 – DC Brush type, 3 – Brushless DC / PMSM motor type
72	BRAKE	2	R/W	Brake control mode (not used)
73	TMODE	2	R/W	Trace mode: 0 - Idle, 1 = Armed – waiting for trigger event, 2 = Start now, Stop on buffer full , 3 = Start now, Stop on end of motion
74	TRATE	2	R/W	Data recorder rate (in 50us intervals)
75	TLEVEL	4	R/W	Data recorder threshold level
76	SIM	2	R/W	Enables (1) or disables(0 – default) encoder simulation
77	TIMER	2	R/W	Timer register that generates the PWM carrier frequency.
78	ADDP	4	R/W	Add Position set point for PVT streaming
79	ADDV	4	R/W	Add Velocity set point for PVT streaming
80	ABS	4	R/W	Defines absolute target position
81	REL	4	R/W	Defines target position relative to the current position
82	POS	4	R/W	Current encoder position
83	INP	2	R/O	Reports the state of the digital inputs as 4 byte hex number.
84	IND	4	R/O	Index position captured
85	GO	0	Cmd	Start motion to the defined target position
86	FWD	0	Cmd	Start jogging forward (positive direction)
87	REV	0	Cmd	Start jogging in reverse (negative direction)
88	RESET	0	Cmd	Resets the firmware / soft restart
89	ON	0	Cmd	Enables the servo control
90	OFF	0	Cmd	Disables the servo control
91	ENABLE	0	Cmd	Enables the PWM amplifier (inverter)
92	DISABLE	0	Cmd	Disables the PWM amplifier (inverter)
93	STOP	0	Cmd	Stop motion smoothly (with the programmed deceleration)
94	ABORT	0	Cmd	Stops motion abruptly (with the maximum deceleration)
95	HOME	0	Cmd	Starts the execution of the Home procedure.
96	ALIGN	0	Cmd	Starts the execution of the Phasing procedure
97	VER	4	R/O	Reports the current firmware version

Par ID	Name	Byte Size	Access	Description
98	OUT1	2	R/W	Controls output #1
99	OUT2	2	R/W	Controls output #2
100	PWM	2	R/W	Output voltage set as PWM duty cycle (32767 = 100%). Requires that the servo control is turned off.
101	IQPKT	2	R/W	Output voltage to generate the Quadrature current component of the motor flux.
102	IDPKT	2	R/W	Output voltage to generate the Direct current component of the motor flux.
103	CH1	2	R/W	Specifies data to be recorded on channel #1 of the data recorder
104	CH2	2	R/W	Specifies data to be recorded on channel #2 of the data recorder
105	CH3	2	R/W	Specifies data to be recorded on channel #3 of the data recorder
106	CH4	2	R/W	Specifies data to be recorded on channel #4 of the data recorder
107	TRACE	2	R/W	Initialize new data recording session
108	PLAY	2	R/W	Reports recorded data
109	PLIMIT	2	R/W	I2T Protection Threshold Limit
110	PTIME	2	R/W	I2T Protection Time Span
111	GEARIN	2	R/W	Specifies the input number of a gear box transmission ratio that defines the electronic gearing ratio.
112	GEAROUT	2	R/W	Specifies the output number of a gear box transmission ratio that defines the electronic gearing ratio.
113	SETUP	2	R/W	Starts procedure to identify the proper mapping of the motor windings and the hall sensors.
114	PINVERT	2	R/W	Specifies if the encoder position feedback should be inverted: 0 – no inversion, 1 = inverted
115	SAVE		Cmd	Saves the persistent parameters to the external Flash memory
116	RESTORE		Cmd	Restores the persistent parameters from the external Flash memory
117	ETYPE	2	R/W	Encoder type: 0 = incremental, 1 = EnDat, 2 = BiSS, 3 = FA-Coder, 4 = A-format
118	EID	4	R/W	Encoder ID:
119	EADDR	2	R/W	Encoder EEPROM address
120	EDATA	2	R/W	Encoder EEPROM data to be stored at or retrieved from the address defined by the EADDR variable
121	EBAUDRATE	2	R/W	Encoder communication baud rate
122	ESTATUS	2	R/W	Absolute Encoder status

Par ID	Name	Byte Size	Access	Description
123	ADDR	2	R/W	Module address used by the packet host communication protocol
124	GROUP	2	R/W	Module group used by the packet host communication protocol
125	TBSIZE	2	R/O	Trace buffer size. Informs the host for the maximum number of samples that can be reported by the data recorder.
126	WMARK	2	R/W	PVT Buffer Watermark – defines the number of slots in the PVT which need to be occupied before watermark warning flag is raised. This is required to properly synchronize the streaming of new PVT points by the host computer.
127	QKD	2	R/W	Proportional Gain in the Quadrature current control loop
128	VKP	2	R/W	Proportional Gain in the velocity velocity loop
129	VKI	2	R/W	Integral Gain in the velocity control loop
130	VKD	2	R/W	Differential Gain in the velocity control loop
131	ELVOLT	4	R/W	Inverter bus voltage Low voltage detection threshold. value is 12bit AD value.
132	EHVOLT	4	R/W	Inverter bus voltage overvoltage detection threshold. value is 12bit AD value.
133	EWPOSMIN	4	R/W	Position abnormality Min threshold
134	EWPOSMAX	4	R/W	Position abnormality Max threshold
135	EOVS	4	R/W	Over speed threshold. value is (Position*65535)/100us.
136	EWOVS	4	R/W	Instructed speed difference threshold. value is (Position*65535)/100us. make it abnormal after continuous detection time (fixed 5 sec).
137	EEMP	4	R/W	PVT Buffer EMPTY Threshold. value is the number of times.
138	EOVTEMP	4	R/W	Temperature anomaly threshold. value is 12bit AD value.
139	ERRMASK	4	R/W	Abnormal state mask. value of ErrMsk in hexadecimal.
140	EVOLT	4	R/O	Bus voltage display. value is 12bit AD value.
141	EQUERY	4	R/O	Abnormal status indication. value of ErrSts in hexadecimal.
142	ERESET	0	Cmd	Abnormal status reset.
143	EOVC	4	R/W	Overload (current) threshold. value is 12bit AD value.
144	ETEMP	4	R/O	Current temperature. value is 12bit AD value.

Revision History	CN032 AC Servo Solution Firmware Manual

Rev.	Date	Description	
		Page	Summary
1.00	Jun.7, 2022	—	First Edition issued
2.00	Aug.9, 2022		
3.00	Sep.30, 2022	11-16	Fix folder structure.
4.00	Feb.28, 2023	8,10, 14-16, 21,27, 53	Description for AC Servo Solution (RZ/T2L) added.

CN032 AC Servo Solution Firmware Manual

Publication Date: Rev.4.00 Feb, 2023

Published by: Renesas Electronics Corporation

SALES OFFICES http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130
Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004
Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900
Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327
Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679
Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999
Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022
Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670
Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300
Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510
Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777
Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

RZ/T2M Group, RZ/T2L Group, RZ/N2L Group

Renesas Electronics Corporation

R11UM0169EJ0400