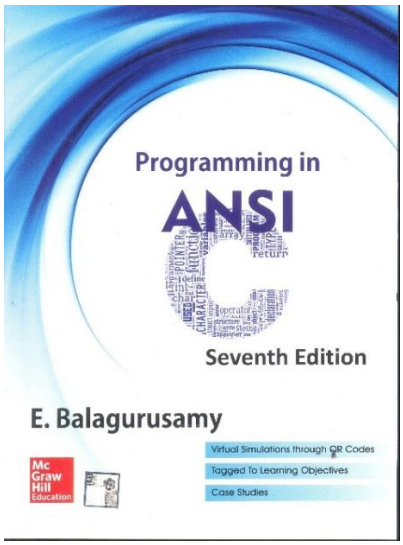
AN100631: Customizing KannMOTION drives, w. ANSI C-code

Introduction

This document is intended to help users to work with KannMOTION drives. KannMOTION allows users to customize drive functions (user sequences) using ANSI-C code and is based on the ISO/IEC 9899:1999 standard, commonly referred to as C99.



KannMOTION's customizing approach allows users to implement their own functionality in a very code and runtime-efficient way. Unlike other common approaches, the user's code does not need to be parsed by a special function such as a PLC. Instead, the user's code is directly converted or compiled into CPU machine code at compile time.

For this purpose, adlos offers two ways to implement own code:

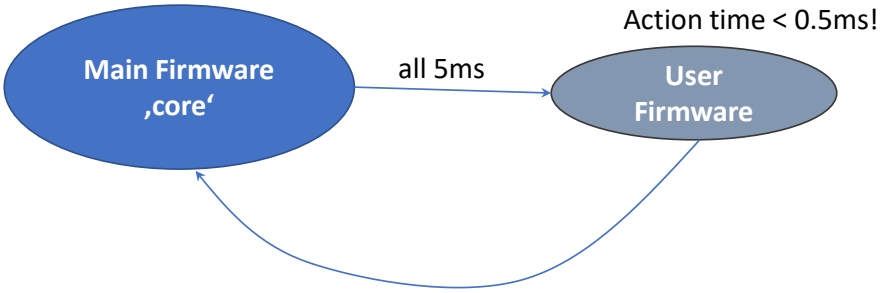
- Graphical block programming (the easier way)
- ANSI-C code programming

How it works

Our KannMOTION controllerboards run on a base firmware that takes care of handling various features such as communication, motor driving, positioning, error management, and I/O scanning and more. This firmware serves as the foundation for all our drives' essential functions.

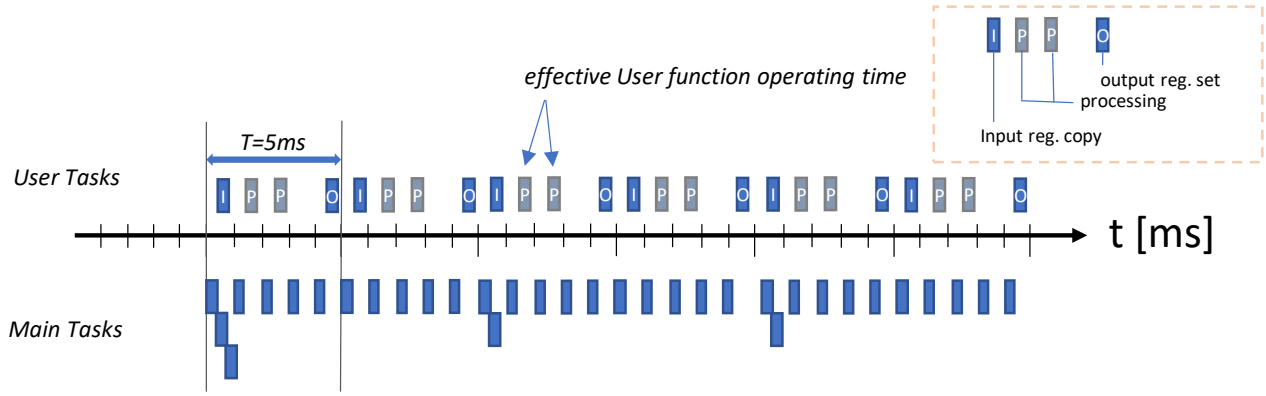
As a distinctive extension, the base (core) firmware can call user code (user sequence) located in the user code section if the checksum and function pointers are valid.

Picture 1: basic call principle



The principle underlying the firmware of our drivers is "cooperative multitasking," which means that no task is permitted to block the operation. Big working loads need to be splitted in smaller working packages!

Picture 2: Timing



Write your code cooperative, means you are niot allowed to write time blocking code !
Your functions are not allowed to exeed more than 1ms operating time, for a stable system it's needed that your operation time/cycle is much less than 1ms, shall be in Range of <100us

```
While (stAppCSPS.SPSUserVar.u8_Din.b.bDI1==1)
{
}
```

Use stepchain Var or merker Var to store, hold, or wait on conditions….

```
if (stAppCSPS.SPSUserVar.u8_Din.b.bDI1==1)
{
    stAppCSPS.SPSUserVar.u8_StepChain[0]=1;
}
```

Working

Using a C-Editor

We suggest using a C-code highlighting editor or IDE for larger projects, instead of the the one in the KannMOTION Manager, because there is no code completion etc. You can use:

- SCiTE http://www.scintilla.org
- STM32CubeIDE
- …

Header Files

Filename	Description	Mode
L0_KannCSPS_exTypes.h	Special user region types	Read Only
L0_KannMotDrvTypes.h	KannMOTION types and enumeration defines	Read Only
L0_TA_datatypes_GCC_ARM.h	Standard data types	Read Only
L2_APPC_SPS_myUserFunctionDefines.h	Helper include to define user functions in short form and provide better documentation for the user	Read Only
L2_APPC_SPS_myUserTypes.h	Helper include to define the structure of user variables	Read Only
L2_APPC_SPS_User.h	User file header	Read Only

Adlos standard types

```
// ******* Unsigned Integer Types, abgeleitet von C99-Standard ******************************
typedef uint8_t   UI_8;                                    //!<  8 Bit -> [0..255]
typedef uint16_t UI_16;                                    //!< 16 Bit -> [0..65535]
typedef uint32_t UI_32;                                    //!< 32 Bit -> [0..4'294'967'295]
typedef uint64_t UI_64;                                    //!< 64 Bit

// ******* signed Integer Types ******************************
typedef int8_t   SI_8;                                     //!<  8 Bit -> [ -128  .. +127]
typedef int16_t SI_16;                                     //!< 16 Bit -> [-32768 .. +32767]
typedef int32_t SI_32;                                     //!< 32 Bit -> [-2147483648 .. +2147483647]
typedef int64_t SI_64;                                     //!< 64 Bit
```

Your User File

Place your code into the following functions.

SPS-USER function	Description
void LOCATEUSER AppCSPS_USER_SEQ_STANDARD_1(void)	Your sequence Part-1 - program here in what you need to do, split it maybe in a second part
void LOCATEUSER AppCSPS_USER_SEQ_STANDARD_2(void)	Your sequence Part-2 - second part maybe needed to meet cooperative multitasking needs
void LOCATEUSER AppCSPS_USER_SEQ_ERROR(void)	Your error Handler Maybe special thinks to do while going intio error mode
void AppCSPS_USER_SEQ_STANDARD_1(void)	TaskHandler1 Program your sequence here, maybe split it into a second part if it's too large or computationally intensive
void AppCSPS_USER_SEQ_STANDARD_2(void)	TaskHandler2 Program your sequence here, second part maybe needed to meet the requirements of cooperative multitasking
void AppCSPS_USER_SEQ_ERROR(void)	ErrorHandler Called when going into error mode
void AppCSPS_USER_SEQ_232_RX_Event(tCSPSCOMRXDATA* pComData)	RS232-RxEvent Called upon receiving the [Df] serial command
void AppCSPS_USER_SEQ_CAN_RX_Event(tCANDATA* pRxData, UI_8 Datalength)	CAN-RXEvent Called upon receiving the CANopen-PDO4 Rx
void AppCSPS_USER_SEQ_CAN_TX_SynxEvent(void)	CAN-TxSyncEvent Called when CANopen-PDO4 Tx (*Sync on Time* OR *Sync CMD reception*)
void AppCSPS_USER_100ms_Event(void)	100ms Regular Event Is called even in error or homing state

Example code: drive to position 1 on digital input 1 rising, and position 2 on digital input 2 rising.
For more examples, check links in ***Additional Information***.

```
#define DIN_POS1          0x01     // Digital Input to go to position 1
#define DIN_POS2          0x02     // Digital Input to go to position 2
#define  POS1_VAL         1800     // Position 1
#define  POS2_VAL         -1800    // Position 2

#define u08_state         UVar.u8_StepChain[0]

// ---- States --------------------
typedef enum
{
   eSTATE__GOTO_POS1      = 1,     // Go to position 1
   eSTATE__GOTO_POS2      = 2,     // Go to position 2
   eSTATE__DEFAULT        = 99
};

/*****************************************************************************************************/
/*!
 * \brief        SPS-USER function / TaskHandler1 (first)
 * \details      Is called while <SPS-RUN> state every 5ms (not while in error-State!).
 *               Your code shall not block (Cooperative Multitask).
 *               Execution time of your block shall be < 50us / max 0.5ms!
 *
 *               Program here, regularly things, e.g. Checking IO's ....
*****************************************************************************************************/
void AppCSPS_USER_SEQ_STANDARD_1(void)
{
    // Check RISING edge on Din 1
    if (((UVar.u8_Din.ucAllBits & DIN_POS1) == DIN_POS1) && (UVar.u8_DinChange.ucAllBits & DIN_POS1))
    {
       u08_state = eSTATE__GOTO_POS1;
    }
    else if (((UVar.u8_Din.ucAllBits & DIN_POS2) == DIN_POS2) && (UVar.u8_DinChange.ucAllBits & DIN_POS2))
    // Check RISING edge on Din 2
    {
       u08_state = eSTATE__GOTO_POS2;
    }

    switch (u08_state)
    {
       case eSTATE__GOTO_POS1:
       {
          // Call go to function
          if(UFu_GotoFuncP(POS1_VAL, eGOTO_um_01deg) == eMS_OK)
          {
             u08_state = eSTATE__DEFAULT;
          }
          break;
       }
       case eSTATE__GOTO_POS2:
       {
          // Call go to function
          if(UFu_GotoFuncP(POS2_VAL, eGOTO_um_01deg) == eMS_OK)
          {
             u08_state = eSTATE__DEFAULT;
          }
          break;
       }
       default:
       {
          // wait/do something
       }
    }
}
```

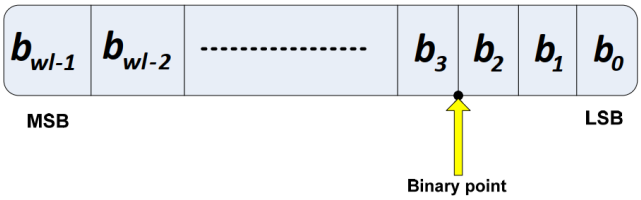
Own functions

⚠	You may define your own function, but make sure to avoid the following: - Passing too many parameters (calling by value), which can lead to excessive heap and stack usage - Making recursive calls, which can lead to a stack overflow
⚠	Keep in mind that the KannMOTION CPU does not have a floating-point unit. Therefore, avoid using floating-point calculations and instead use fixed-point operations. Please refer to the next chapter for more information.
✓	On the other hand, make your functions small, efficient, and fast.

Calculations

To optimize performance, it is recommended to avoid floating-point calculations on the KannMOTION CPU as it does not have a floating-point unit. Instead, use fixed-point operations for better efficiency.

Figure 1: Fix Point representation



See:
https://www.embedded.com/fixed-point-math-in-c/

Flex-User Variables

Here the FlexUser area which has a size of 20 or 128 bytes, depending on the controllerboard (HW, FW version). Except for the reserved bytes for controller-specific variables, this area is available for FlexUser variables and can be used freely. See other tabs for more information on how about to use it.

Find more information in AN100631_SampleCode.zip, see ***Additional Information*** for downloadlink.

Total Flex User Vars (n)	20 / 128 Bytes		
RAM Adress	**Access type**		
	u08 / i08	**u16 / i16**	**u32 / i32**
Base+0	0	0	0
Base+1	1		
Base+2	2	1	
Base+3	3		
Base+4	4	2	1
Base+5	5		
Base+6	6	3	
Base+7	7		
Base+(n-8)	n-8	(n/2)-4	(n/4)-2
Base+(n-7)	n-7		
Base+(n-6)	n-6	(n/2)-3	
Base+(n-5)	n-5		
Base+(n-4)	n-4	(n/2)-2	(n/4)-1
Base+(n-3)	n-3		
Base+(n-2)	n-2	(n/2)-1	
Base+(n-1)	n-1		

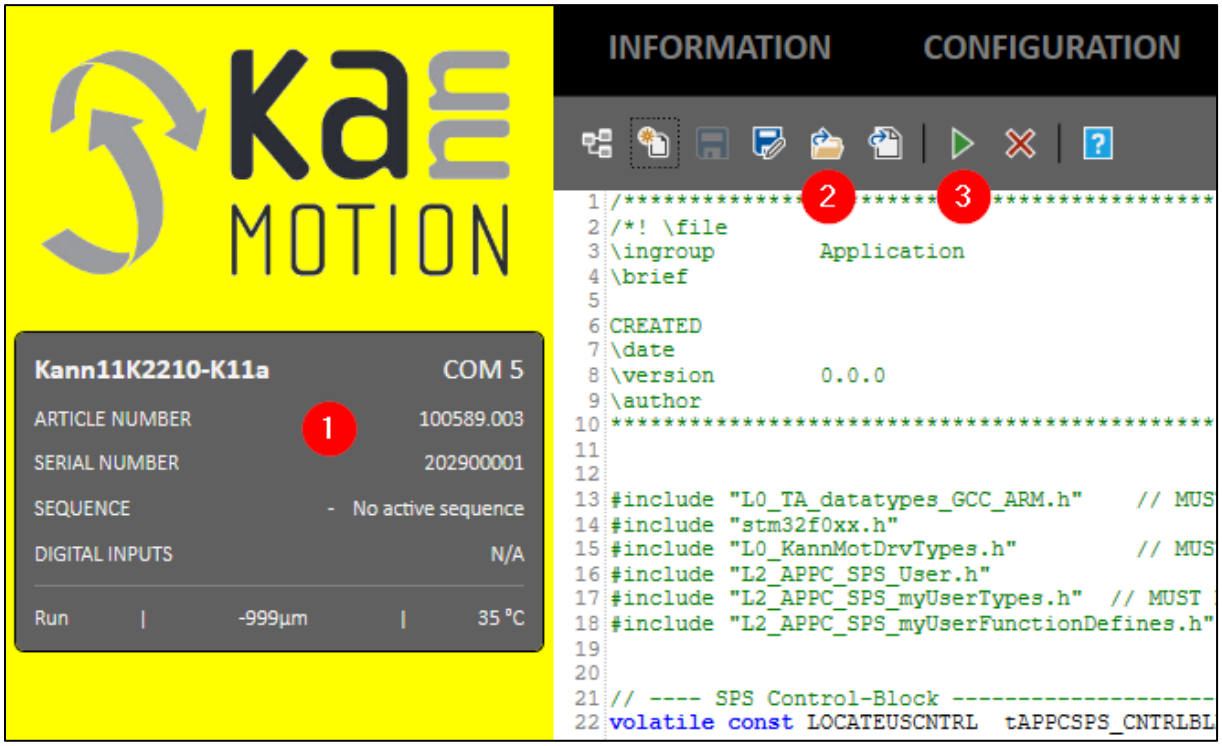
FW Art.No.	*190082*	*190056*	*190069*	*190103*	*190145*
128 Bytes Version (others are 20 bytes)	>= 2.5-003	>= 2.0-002 < 3.0-000 >= 3.0-002	>= 3.1-001	>= 2.6-002 < 2.7-001 >= 2.7-002	>= 0.9-000
					Controller type depe
	K17a	K17c		K17e	K17f
	free to use	**free to use**		**free to use**	**free to use**
		u8_Reserved			i16_Din_SlopeCntp50ms
		u8_CAN_Errors			
		i16_Din_SlopeCntp50ms			u16_Ain4to20mA_0d01mApE
	u16_Ain0to24_8d86mVpE	u16_Ain0to24_8d86mVpE		u16_Ain0to24_8d86mVpE	u8_CAN_Errors
					u8_Reserved

190150	*190097*	*190120*	*190142*
>= 0.9-000	>= 2.5-005	>= 2.5-005	>= 0.9-000
nding pre-allocation			
K17g	K11a	K11b	D17a
free to use	**free to use**	**free to use**	**free to use**
i16_Din_SlopeCntp50ms	u16_Aout4to20mA_0d01mApE	u16_Aout0to10V_1mVpE	
u16_Ain0to10v_1mVpE	u16_AinMagSns_0d806mVpE	u16_AinMagSns_0d806mVpE	u16_Ain0to10v_1mVpE
u8_CAN_Errors	u16_Ain4to20mA_0d01mApE	u16_Ain0to10v_1mVpE	u8_CAN_Errors
u8_Reserved			u8_Reserved

Program your Drive

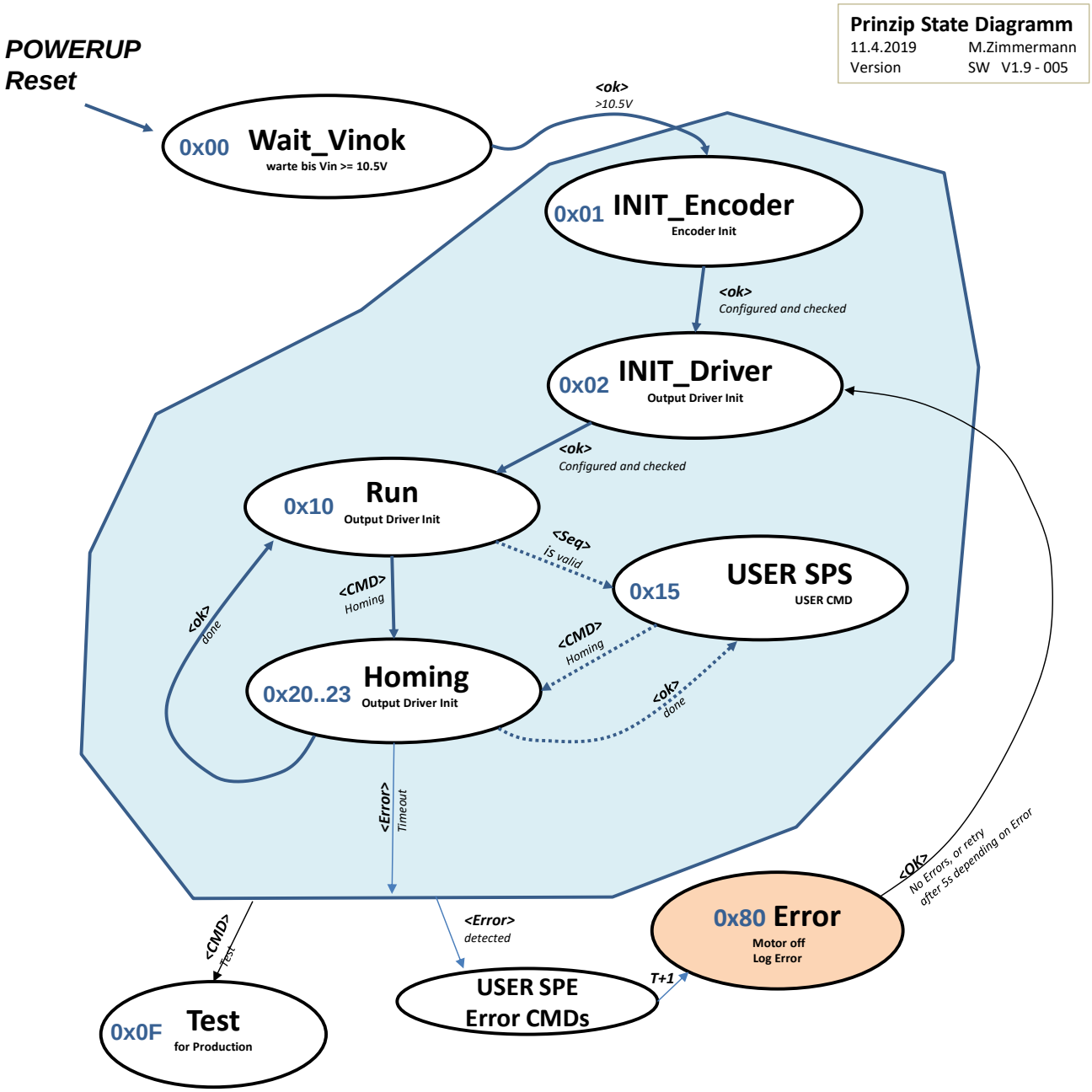
To run your user sequence on the KannMOTION, follow these steps:

1. Start the KannMOTION Manager and select the appropriate drive
2. Load your user sequence (C-file)
3. Compile the program and program it to the drive

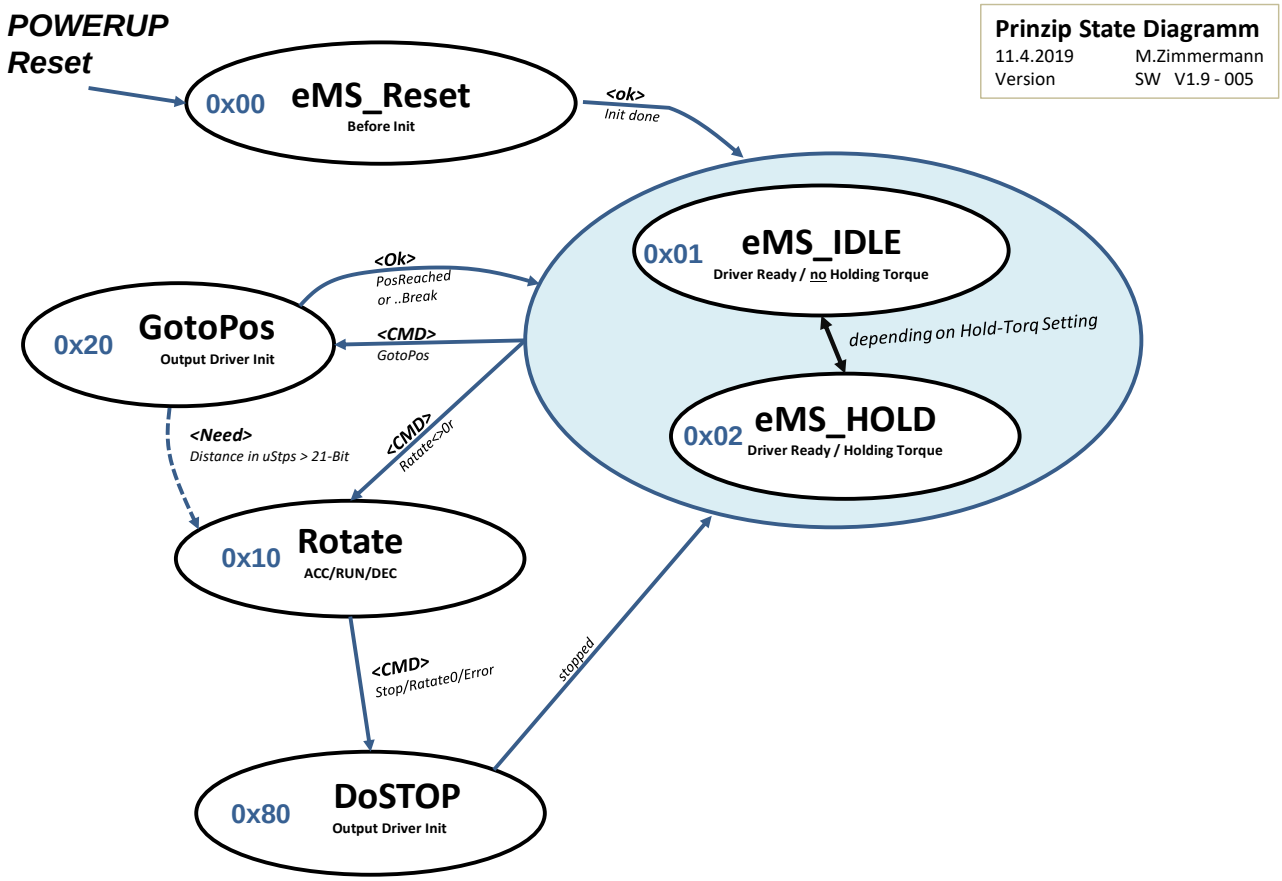


Appendix

Drive Principal State Digram / Main States



Drive Principal State Digram / Drive Substates



Additional Information

Check following links for additional information like manuals, tools and especially the sample codes for the user sequences.

Manuals, Tools, AppNotes	https://www.kannmotion.com/en/downloads/
User Sequence Examples	https://kannmotion.li/download/ANs/100631/AN100631_SampleCode.zip

Contact information

Adlos AG
Föhrenweg 14
FL-9496 Balzers

Thomas Vogt
thomas.vogt@adlos.com
Tel: +423 263 63 63

Countries: CH, A, LI, SK, IT
www.adlos.com

KOCO MOTION GmbH
Niedereschacher Straße 54
D-78083 Dauchingen

Olaf Kämmerling
o.kaemmerling@kocomotion.de
Tel: +49 7720/995858-0

Countries: DE, BE, NL, LU
www.kocomotion.de