TIP102/111-SW-42
VxWorks Motion Control Firmware

Version 1.0
Issue 1.0
23. October 1998

TEWS DATENTECHNIK GmbH
Am Bahnhof 7
D-25469 Halstenbek
Germany
Tel.: +49 (0)4101 4058-0
Fax.: +49 (0)4101 4058-19

TIP102/111-SW-42

VxWorks Motion Controller Firmware for motion controller with absolute and incremental interface

TEWS DATEN*TECHNIK* GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS DATEN*TECHNIK* GmbH reserves the right to change the product described in this document at any time without notice.

This product has been designed to operate with IndustryPack® compatible carriers. Connection to incompatible hardware is likely to cause serious damage.

TEWS DATEN*TECHNIK* GmbH is not liable for any damage arising out of the application or use of the device described herein.

Issue	Description	Date
1.0	First Issue	23. October 1998

Table of Contents

Table of Figures

1 Introduction

The TIP102/111 VxWorks Motion Controller Firmware (TMF) can be used to implement a motion controller system up to eight axis on a single MOTOROLA MVME162 CPU. The software supports any number and combinations of TEWS Datentechnik's TIP102 and TIP111 motion controller hardware.

The software package contains several components.

The TMF command server contains the core of the motion control firmware. TMF-Server implements a PID close loop control algorithm for each axis, which can be individually configured by a set of parameter.

The TMF device driver contains the time-base for PID algorithm and callback functionality for the external path control mode.

The optional TMF axis monitor provides an inactive tool to configure, run and monitor the axis. This is most useful during system integration and for diagnostics.

2 Installation

2.1 Install Files

The TIP102/111-SW-42 VxWorks Motion Controller Firmware is delivered on a 3½“ floppy disk:

The floppy has the following contents:

Directory <DRV>:

MCCHIP.H	MVME162 MC-Chip definitions
TMFDRIV.C	TMF device driver source
TMFDRIV.H	C header for application program
TMFDEFS.H	C header for device driver
TMFCONF.H	Device configuration file
MAKEFILE	Makefile to build device driver and driver test application

Directory <TMF>:

<xxx>.C	24 source code modules
TMF.H	C header for TMF-server
TMFSHARE.H	C header for application program
MAKEFILE	Makefile to build the TMF-Server object ‘TMFLIB.O’

Directory <EXAM>:

TMFEXAM.C	simple TMF example application source code modules
MAKEFILE	Makefile to build the TMF example object ‘TMFEXAM.O’

Directory <MONI>:

<xxx>.C	14 source code modules
<xxx>.H	4 C header files
MAKEFILE	Makefile to build the axis monitor object ‘MONILIB.O’

Directory <MONI\VT>:

<xxx>.C	63 source code modules
<xxx>.H	2 C header files
MAKEFILE	Makefile to build the VT-terminal library object ‘VTLIB.O’

Directory <MONI\GRAPH>:

<xxx>.C	27 source code modules
<xxx>.H	1 C header file
MAKEFILE	Makefile to build the Graphic library object ‘GRAPHLIB.O’

For installation you have to copy these files to the desired target directory.

2.2 Configure the Motion Controller Firmware

After installing the files you have to configure the following things:

- select one the four MC-chip tick timer as time-base for the PID close loop control algorithm. (MVME162)
- Setup the PID time-base
- Setup the axis configuration
- Build TMF device driver
- Build TMF server

2.2.1 Select Timer Device

You have to select a free timer as timer device for TMF device driver. This timer will be used to build the time-base for the control algorithm. The delivered device driver is made for the Motorola MVME162 and supports the four timer devices of the MC-Chip. (To use an other timer device, read *2.4 Using other Timer Devices*)

To specify the timer device you have to change the address of the timer device in TMFCONF.H. This is the first value in the data structure, default is 0xfff42016 for the Timer 2 device. Select one of the following timers by their value:

- Timer 1 0xfff42017
- Timer 2 0xfff42016 (default)
- Timer 3 0xfff4201F
- Timer 4 0xfff4201E

You can also change the level of the created interrupt, the default interrupt level is four. To change the interrupt level change the third value in the configuration structure to the value of the needed interrupt level.

The interrupt vector of the MC-Chip timers depends on the MC-Chip interrupt base vector and is not used for the MC-Chip timers, it can be left zero. The interrupt vector will be calculated by the device driver.

2.2.2 Setup the PID Time Base

By default the time-base is set to 4000µs. To change this value you have to change the second value in the configuration structure in TMFCONF.H. You have to specify the value in µs.

The minimum sampling rate depends on the number of processed axis and of the performance of the CPU. Estimated sampling rates on MVME162 system running with 25MHz (with FPU) are 2000µs for up to four axis and 4000µs for up to eight axis.

2.2.3 Axis configuration

The motion controller firmware supports TIP102 and TIP111 controllers mounted on the local IP-slots of the CPU or on external carrier boards. To assign a physical axis to a logical TMF motion controller axis you have to modify the configuration structure in TMFCONF.H. Each entry contains two parameters you have to specify. The first contains the base address of the module and the second specifies the axis controller index. The axis controller index must be '0' for TIP102/111 for one axis type (-1x) and '0' or '1' for TIP102/111 for two axis type (-2x).

For example:

Configure axis 0 on IP mounted to IP-Slot C as motion controller axis 4:

```
...
/* 4 */   0xfff58200    0
...
```

2.3 Using other Carriers

The TIP102/111 Motion Controller Firmware is designed for Motorola MVME162 CPUs. To run it on other systems you have guarantee the following points:

- All TIP102 and TIP111 modules must be accessible
- A timer device must support all control functions of the delivered device driver (Have a look to *2.4 Using other Timer Devices)*
- You have to adapt the makefiles and you have to assign the right processor type. If you are not using a Motorola 68040-CPU with floating-point unit, you have to use software floating-point or you have to add some code for floating register saving in the ISR. *(Have a look to 'tmfIsr()' in TMFDRIV.H)*

2.4 Using other Timer Devices

It may be necessary to use a timer which is not supported by the default device driver. In this case you have to adapt the driver for new timer.

You have to make the following adaptations in the device driver source TMFDRIV.C:

Add initialize code for the new timer

Function '*tmfDevCreate*':

```
switch(par->TimerAddress)
{
case 0xfff42017:
     ...
     break;

case 0xfff42016:
     ...
     break;

case 0xfff4201F:
     ...
     break;

case 0xfff4201E:
     ...
     break;

case 0xffff3000:   /* Timer base address at ffff3000 */
     /* Setup the timer to generate a cyclic interrupt    */
     /* at the specified time     */
     ...

     /* Install the interrupt service function  */
     if (intConnect(...) == ERROR)
     {
          /* Handle error */
     }

     /* enable cyclic interrupts  */

     break;
}
```

Adapt the control function for the new timer

Function '*tmfIoctl*':

- Make sure the functions will work for the new timer like the default functions on the default timers.

Adapt the interrupt service function for the new timer

Function '*tmfIsr*':

- Adapt the part where the Interrupt request is acknowledged at the beginning of the function.

2.5 Starting the Motion Controller Firmware

After creating and loading the object module TMFDRIV.O and TMFLIB.O onto the VxWorks system, it is possible to start the TMF-Server. The TMF-Server should be started as a task with a high priority. This can be done from the shell or from the application software. *(Please refer to the VxWorks Manuals)*

For example start the TMF-Server from the shell:

```
taskSpawn("TMFS", 200, 0, 0x2000, run_tmfserver)
```

3 Application Interface

This chapter describes the interface between the TMF motion control firmware and the application program.

This interface consists of four objects:

- A shared parameter structure to pass the parameters
- A command message queue to activate new commands
- An access semaphore locking the parameter structure and command activation
- Optional completion semaphores for each axis

The shared parameter structure and the access semaphore are created and initialized by the TMF-Server. The completion semaphores have to be created by the application task.

3.1 Shared Parameter Structure

The shared parameter structure contains the axis parameter storage ('*axp[]*') for eight axis and the storage for the common parameters ('*cop*'). The structure is used to pass the parameters between the application and the TMF-Server. The shared parameter structure is created as *SMM_TYPE*, which is defined in the include file TMFSHARE.H, which is part of the distribution. The TMF-Server assigns the base address of the shared parameter structure to a pointer variable '*smm*'.

(All parameters in this structure are described below in chapter 5 Parameter)

3.2 Access Semaphore

The access semaphore is used to protect the parameters of a command, to be changed before the command is started and do not longer need this parameters.

The application must call '*semTake()*' before entering the application part changing parameters. The TMF-Server will call '*semGive()*' after using the parameters for the new command.

The access semaphore is created by the TMF-Server and its ID is stored in '*AccessSemID*', which is imported by TMFSHARE.H.

3.3 Command Message Queue

The command message queue is used to signal a new command needs action from the TMF-Server. This can have two reasons:

- The application wants to start a new command and the parameters have been set up.
- A command has been completed and the TMF-Server has to signal the completion.

The application must call a '*msgQSend()*' and the TMF-Server will wait for a command with a '*msgQReceive()*' call.

The value of the message signals the type of the message:

- New Command is ready to be processed = 1
- A command has been completed = 2

The command message queue is created by the TMF-Server and its ID is stored in '*CmdMsgID*', which is imported by TMFSHARE.H.

3.4 Completion Semaphores

The completion semaphores must be created by the application and they must be empty after creation. The semaphore ID must be stored in '*Completion_SemID*', which is part of the axis parameter block in the parameter structure.

A completion semaphore can be created for every axis, the TMF-Server will signal a completion with a '*semGive()*' call, and the application can wait for the completion with a '*semTake()*' call.

3.5 Communication Interface

The following figure shows the communication interface between the application and the TMF-Server.

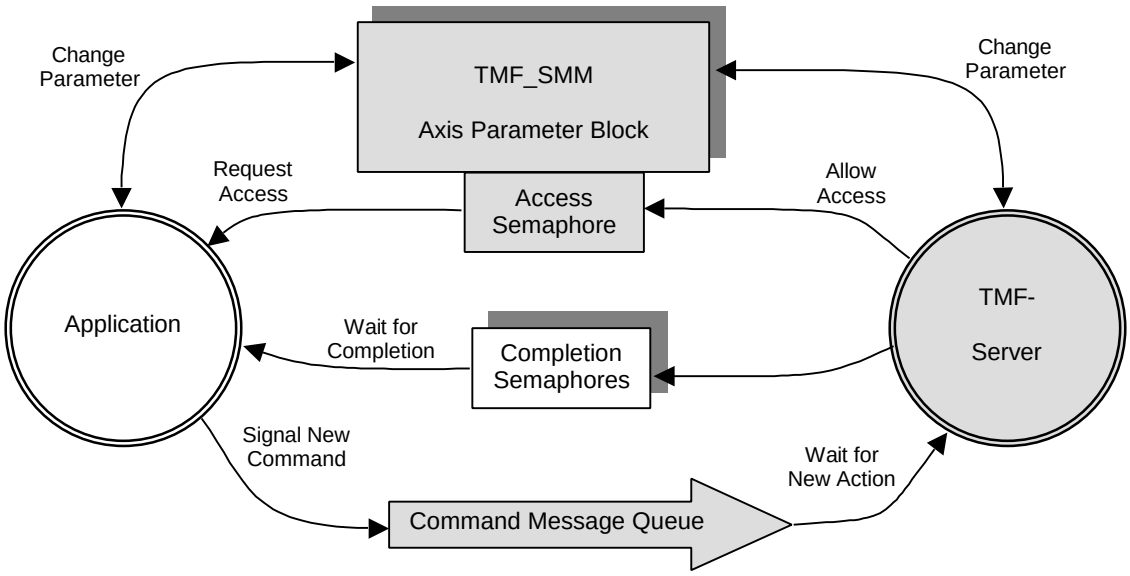

Figure 1: Application Interface

The following figure shows a float chart, demonstrating how to setup a new command

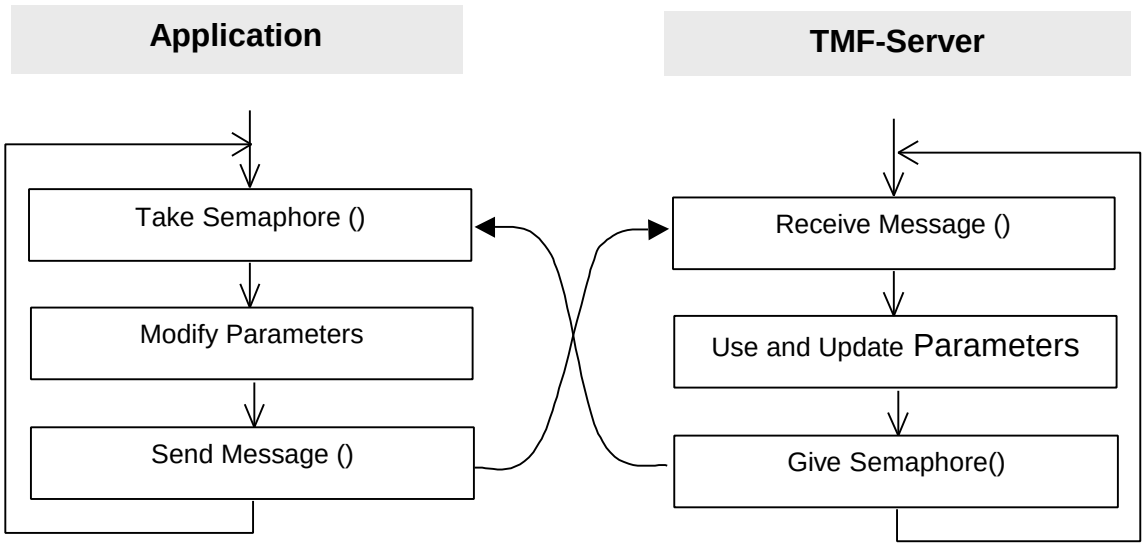

Figure 2: Command Sequence

4 Functions

4.1 Initialization Function

NAME

TMF_INITIAL_FUNCTION – Initialization and configuration of the specified axis.

DESCRIPTION

The initialization function is the first function which must be executed for all axis. This function interprets all user configuration parameters, initializes all internal process data and configures the corresponding axis into the motion controller system.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Initial(int ax)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Initial_Reg              = TMF_AXIS_ENABLE_BIT;
     smm->axp[ax].Reference_Configuration  = TMF_REF_SWITCH_BIT;
     smm->axp[ax].Command_Offset           = 0;
     smm->axp[ax].Kv_Command_Value         = KV_COMMAND;
     smm->axp[ax].Kv_Speed_Value           = KV_SPEED;
     smm->axp[ax].Kp                       = 1.5;
     smm->axp[ax].Ki                       = 0.0;
     smm->axp[ax].Kd                       = 0.0;
          <... and so on, see also example program ...>

     smm->axp[ax].Axis_Command             = TMF_INITIAL_FUNCTION;
     smm->axp[ax].Axis_New_Command         = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);

     return smm->axp[ax].Axis_Error;
}
```

4.2 Reference Function

NAME

TMF_REFERENCE_FUNCTION – Search for the axis reference position.

DESCRIPTION

The Reference Function is used to calibrate the counter of the incremental encoder interface of the TIP102 to an absolute position. For the TIP111 hardware, this function is without operation, because the absolute encoder delivers already absolute positions.

The Reference Function must be called after the Initialization Function was called, but it can be repeated any time later.

To perform the calibration of the incremental measurement system, the Reference Function has to move the axis until the reference condition occurs inside the TIP102 hardware (*see also the TIP102 hardware manual*).

The required sequence to find the reference point depends on the design of the axis mechanical configuration. The bits of the Reference Configuration Register are used, to configure the motion controller firmware for the right hardware configuration.

4.2.1 Configuration with no reference switch

In hardware designs which have no reference switch, only a single zero (index) pulse is allowed within the working range of the axis. When the Reference Function is started the system does not know, if the zero (index) pulse of the encoder will occur when the axis is moved in positive or negative direction. If the *TMF_REF_DIRECTION_BIT* is cleared, the axis is moved in positive direction first, otherwise in negative direction.

If the limit switch becomes active, before the zero (index) pulse was seen, the direction of the movement is reversed. If the zero (index) pulse is seen, the reference condition is true, the counter is loaded with the value of the Reference Position Register (*Reference_Position*) and the axis is stopped.

If the axis hits the opposite limit switch without the zero (index) pulse was seen, the axis is stopped with an error

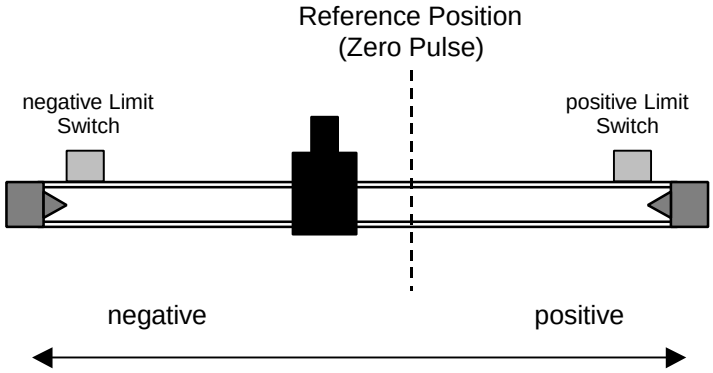

Figure 3: Axis configuration without reference switch

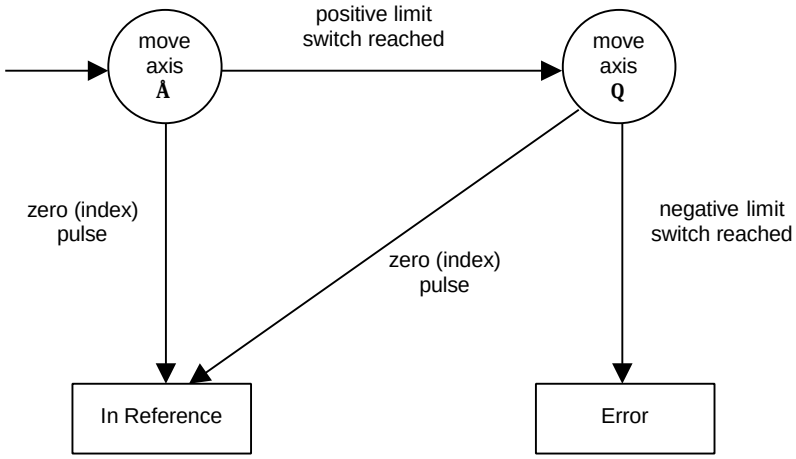

Figure 4: State diagram – reference without reference switch

4.2.2 Configuration with a 'spot' reference switch

In this configuration a reference switch is present, which is active in the range of a single zero (index) pulse of the encoder. The reference switch is used to mask only one zero pulse in the working range of the axis to define the reference position. For this configuration the *TMF_REF_SWITCH_BIT* and the *TMF_REF_SPOT_BIT* must be set in the Reference Configuration Register (*Reference_Configuration*). The search sequence of the reference position is the same as in the configuration without reference switch.

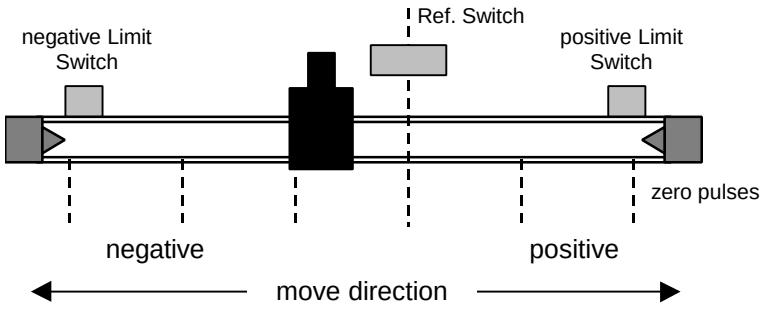

Figure 5: Axis configuration with ‘spot’ reference switch

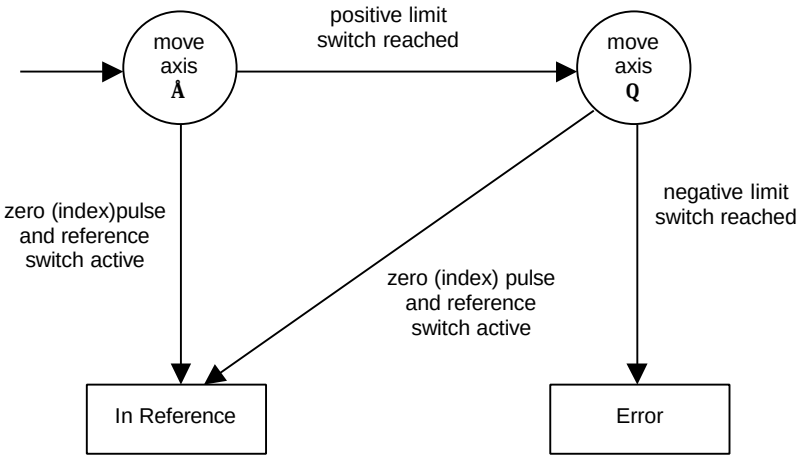

Figure 6: State diagram – reference with ‘spot’ reference switch

4.2.3 Configuration with a 'long' reference switch

In this configuration the reference switch can be active for more than one zero (index) pulse of the encoder. The reference switch must be active for all positions between the reference position and one of the limit switches. In this configuration the reference position is defined by the first zero (index) pulse after an inactive to active transition of the reference switch has been detected. By checking the initial status of the reference switch, the Reference Function knows in which direction the reference position must be searched. So this is the most 'intelligent' reference configuration and it may require much less movements and time to calibrate the measurement system. For this configuration the *TMF_REF_SWICHT_BIT* must be set in the Reference Configuration Register (*Reference_Configuration*).

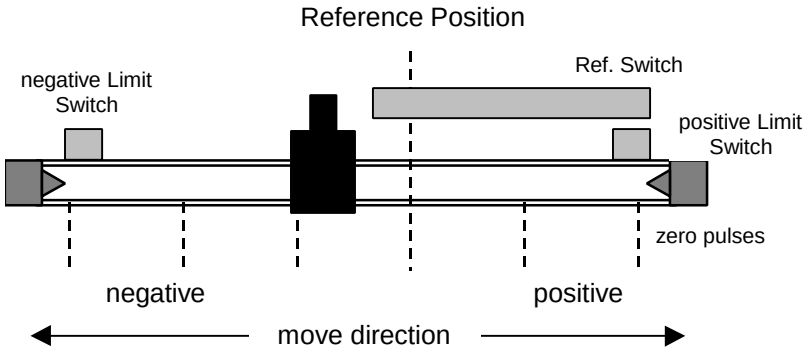

Figure 7: Axis configuration with 'long' reference switch

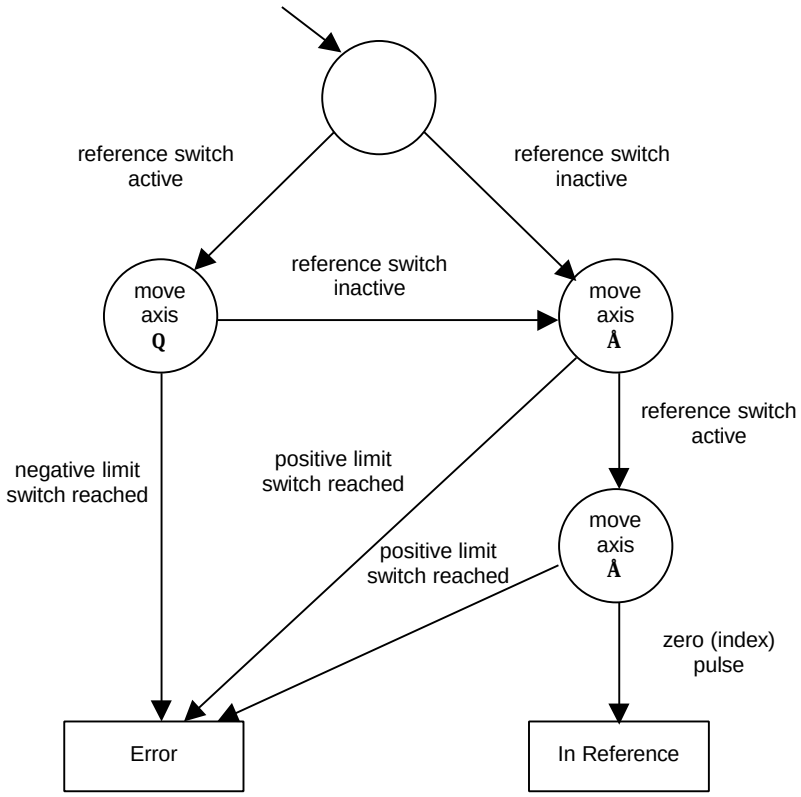

Figure 8: State diagram - reference with 'long' reference switch

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Reference(int ax)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Reference_Configuration  = TMF_REF_SWITCH_BIT;
     smm->axp[ax].Reference_Speed          = 1000;
     smm->axp[ax].Reference_Position       = 0;

     smm->axp[ax].Axis_Command             = TMF_REFERENCE_FUNCTION;
     smm->axp[ax].Axis_New_Command         = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);

     return smm->axp[ax].Axis_Error;
}
```

4.3 Move Function

NAME

TMF_MOVE_FUNCTION – Move the axis to the specified target position.

DESCRIPTION

The Move Function of the motion controller firmware is used to position a axis to a new target position. This function can only be started, if the Reference Function has been executed successfully before (TIP102). The new target position must be loaded into the Target Position Register (*Target_Position*), the start acceleration into the Start Acceleration Register (*Start_Acceleration*) and the stop acceleration into the Stop Acceleration Register (*Stop_Acceleration*). The movement from the current position to a new target position is divided into three sections:

- The acceleration section where the axis is increasing its speed (S_1).
- The section where the axis is moving with its maximum speed (S_2).
- The deceleration section where the axis comes to stop at the target position (S_3).

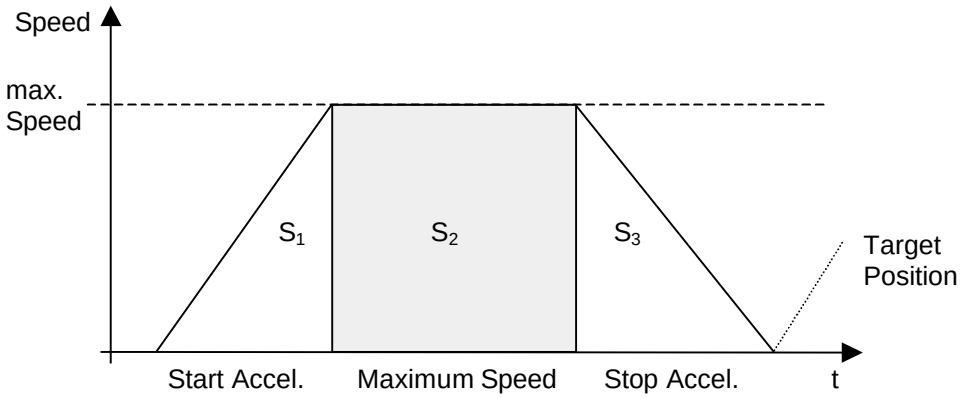

Figure 9: Speed diagram for long movements

If the distance to a target position is too short for the axis being accelerated to maximum speed, section S_1 directly passes control to section S_3. The axis will not reach maximum speed for this type of movement.

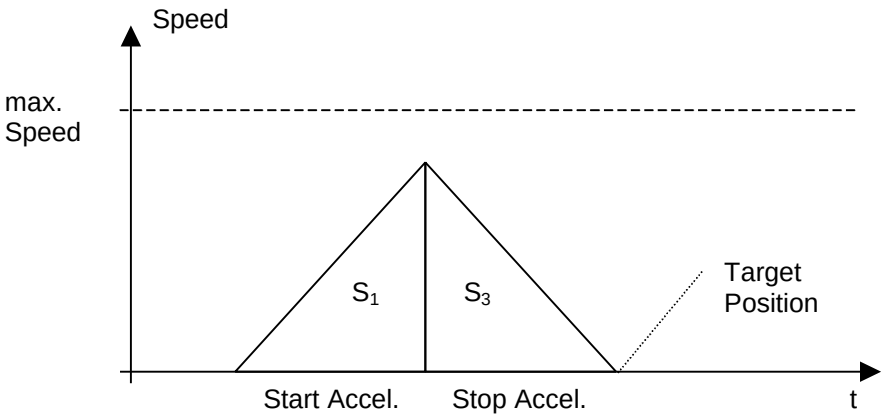

Figure 10: Speed diagram for short movements

The user may restart the Move Function with new parameter for Target Position, Maximum Speed, Start and Stop Acceleration, while the motion controller firmware is still executing an older Move Function for that axis. In this case the motion controller firmware will change to the new parameter when ever possible on the fly, without having the older function finished and having the axis stopped at the old target position.

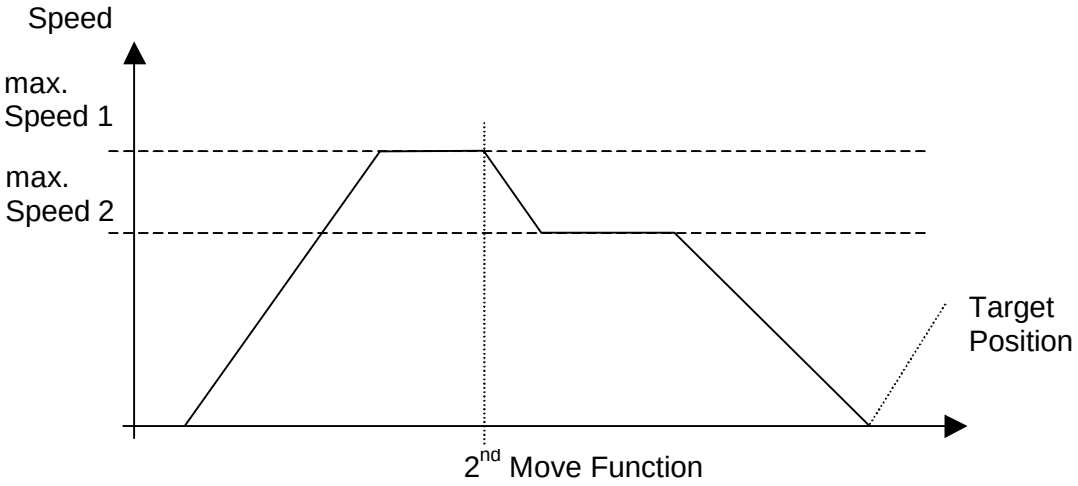

Figure 11: Speed diagram with speed change on the fly

Figure 10 is showing the resulting speed diagram, when the Maximum Speed is reduced with a second Move Function on the fly.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Move(int ax, long target)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Target_Position     = target;

     /* perhaps setup new speed and acceleration values! */
     smm->axp[ax].Axis_Command        = TMF_MOVE_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);

     return smm->axp[ax].Axis_Error;
}
```

4.4 Swing Function

NAME

TMF_SWING_FUNCTION – Move the axis consecutive between two positions.

DESCRIPTION

The Swing Function is a tool for tuning your system. After starting, the axis moves endless (if not explicit stopped) between two target positions (*Target_Position and Target_Pos_2*).

Note

This function reports no completion. It proceeds until explicit aborting by an other command.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Swing(int ax, long target1, long target2)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Target_Position     = target1;
     smm->axp[ax].Target_Pos_2        = target2;

     /* perhaps setup new speed and acceleration values! */
     smm->axp[ax].Axis_Command        = TMF_SWING_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);
}
```

4.5 Stop Function

NAME

TMF_STOP_FUNCTION. -.Stop the axis with the specified deceleration.

DESCRIPTION

The Stop Function terminates every movement of the axis in a controlled way. The Stop Function uses a separate deceleration, which is defined in the Stop Acceleration Register (*Stop_Acceleration*), to bring the axis to a stop. The axis remains in close loop control at the stop position.

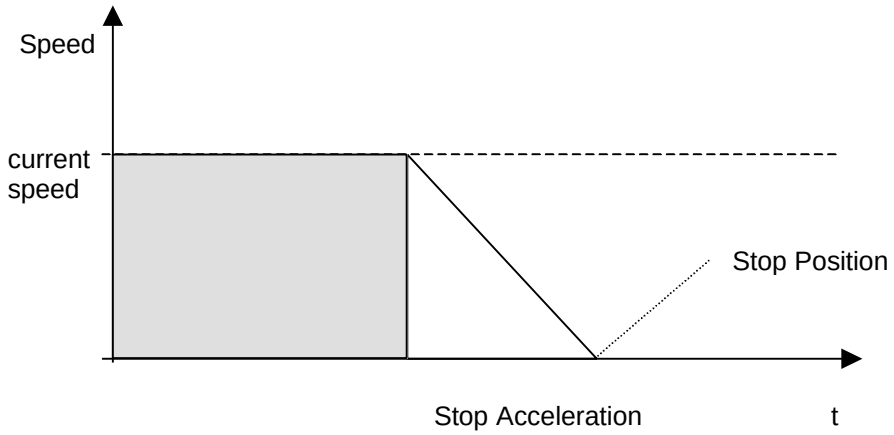

Figure 12: Speed diagram for Stop Function

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Stop(int ax, long accel)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Stop_Acceleration   = accel;

     smm->axp[ax].Axis_Command        = TMF_STOP_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);

     return smm->axp[ax].Axis_Error;
}
```

4.6 Emergency Stop Function

NAME

TMF_EMERGENCY_STOP_FUNCTION Stop the axis with maximum deceleration.

DESCRIPTION

The Emergency Stop Function stops the axis in the shortest possible time. This function will reset the speed command output immediately to '0'. The axis will use the maximum motor current, which can be supplied by the servo amplifier, to stop the motor. If the number of sampling periods, which are specified in the Disable Delay Configuration Register (Disable_Delay), have expired, the Servo Amplifier Enable Output is disabled.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int EmergencyStop(int ax)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Disable_Delay       = DELAY;

     smm->axp[ax].Axis_Command        = TMF_EMERGENCY_STOP_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);

     return smm->axp[ax].Axis_Error;
}
```

4.7 Jog Function

NAME

TMF_JOG_UP_FUNCTION – Move the axis in forward direction.
TMF_JOG_DOWN_FUNCTION – Move the axis in backward direction

DESCRIPTION

The Jog Up/Down Functions moves the axis with a constant velocity in positive or negative direction until a limit switch is reached or the function will be aborted by an other function (for example TMF_STOP_FUNCTION). The velocity must be defined in the Jog Up/Down Speed Register (*Jog_Up_Speed, Jog_Down_Speed*).

Note

This function reports no completion. It proceed until a limit switch is reached or it was explicit aborted by an other command.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int JogUp(int ax, long speed)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Jog_Up_Speed        = speed;

     smm->axp[ax].Axis_Command        = TMF_JOG_UP_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);
}
```

4.8 Go Function

NAME

TMF_GO_UP_FUNCTION – Move the axis in forward direction.
TMF_GO_DOWN_FUNCTION – Move the axis in backward direction

DESCRIPTION

The Go Up/Down Functions moves the axis with a constant command value (without PID control) in positive or negative direction until a limit switch is reached or the function will be aborted by an other function (for example TMF_EMERGENCY_STOP_FUNCTION). The command value (DAC value) must be defined in the Go Up/Down Command Value Register (*Go_Up_Command_Value, Go_Down_Command_Value*).

This function is most useful for the first installation if PID parameter are unknown.

Note

This function reports no completion. It proceed until a limit switch is reached or it was explicit aborted by an other command.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int GoUp(int ax, short value)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Go_Up_Command_Value  = value;

     smm->axp[ax].Axis_Command         = TMF_GO_UP_FUNCTION;
     smm->axp[ax].Axis_New_Command     = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);
}
```

4.9 Joystick Function

NAME

TMF_JOYSTICK_FUNCTION – The motion is controlled by an external analog signal.

DESCRIPTION

After the Joystick Function is active the velocity of the axis movement depends on the analog input signal of the ADC (Joystick). The bandwidth of the analog signal can be divided into three sections.

The 1st section is the idle section, controlled by the parameter *Joystick_Idle_Window*. In this section the velocity is zero and the axis is stopped.

The 2nd section is the linear section, controlled by the parameter *Joystick_Linear_Window*. In this section the velocity is proportional to the input voltage. The gradient of the straight line is defined by the parameter *Joystick_Linear_Speed*.

The velocity of the rest of the analog bandwidth is calculated by an square root build in function. The parameter *Joystick_Maximum_Speed* defines the velocity at full-scale range of the analog input of this section.

Note

This function reports no completion. It proceed until a limit switch is reached or it was explicit aborted by an other command.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

int Joystick(int ax)
{
     unsigned long msg_buffer = MSG_CMD_VALUE;

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Joystick_Idle_Window= 10; /* [%] */
     smm->axp[ax].Joystick_Linear_Window   = 30; /* [%] */
     smm->axp[ax].Joystick_Linear_Speed    = 1000;
     smm->axp[ax].Joystick_Maximum_Speed   = 20000;

     smm->axp[ax].Axis_Command        = TMF_JOYSTICK_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);
}
```

4.10 External Path Function

NAME

TMF_PATH_FUNCTION – Move the axis on an external calculated path

DESCRIPTION

This function starts the external path mode. The axis movement is complete controlled by the application program, that means, the next target position for every sampling period is passed through the parameter *External_Path_Position* to the PID closed loop algorithm.

To synchronize the path calculation function with the sampling period, a callback functionality is implemented in the TMF device driver. After the callback function is installed into the system, it is called at every sampling period until the driver is closed. Inside the callback function you have unrestricted access to the tasks global variable.

ATTENTION! The callback function will be executed in system state during interrupt processing.

Note

This function reports no completion. It proceed until a limit switch is reached or it was explicit aborted by an other command.

EXAMPLE

```
#include <errnoLib.h>
#include <semLib.h>
#include <msgQLib.h>
#include "tmfshare.h"

...

void CalcPath(unsigned long arg)
{
     long NextPosition;

     /* calculate next path position */

     smm->axp[arg].External_Path_Position = NextPosition;
}
```

```
int InitPath(int ax)

{
     tmfFUNC_STRUCT     func_par;
     unsigned long      msg_buffer = MSG_CMD_VALUE;
     int                tmf;

     tmf = open("/tmf", 0, 0);

     /* Install callback function */
     func_par.func      = CalcPath;
     func_par.arg       = ax;

     ioctl(tmf, tmf_INST_CALLFUNC, &func_par);

     /* Get Access to parameter structure */
     semTake(AccessSemID, WAIT_FOREVER);

     /* setup all necessary parameter */
     smm->axp[ax].Axis_Command        = TMF_PATH_FUNCTION;
     smm->axp[ax].Axis_New_Command    = TRUE;

     /* Announce new command is ready to be processed */
     msgQSend( CmdMsgID,
               (char*)&msg_buffer,
               sizeof(msg_buffer),
               NO_WAIT,
               MSG_PRI_NORMAL);
}
```

5 Parameter

The function of the TMF motion controller is controlled by a set of parameters. Some of these parameters are common to all axes, like the *BaseSamplePeriod* for the PID close loop control algorithm. Others are individual to each axis. Some of the parameters are only used at system initialization time to configure the system, others are used with each command which is started, like the Target Position of an axis.

Note

All used symbolic names in this chapter are defined in the include file TMFSHARE.H which is part of the distribution. See also the example application program TMFEXAM.C in the sub directory <EXAM> for further information of using.

5.1 Axis Specific Configuration Parameter

The following section is a description of all parameters which are used to configure the axis at initialization time. Any modification to these parameters will become effective only when the Initialize Function is executed for the corresponding axis.

5.1.1 Initial_Reg

The Initial Configuration Register defines some basic modes of operation for an axis by a set of bits.

The following bits are used:

5.1.1.1 TMF_AXIS_ENABLE_BIT

This is the general enable bit for the axis. Setting this bit allows the motion controller firmware to control this axis.

5.1.1.2 TMF_SINGLE_RES_BIT

The normal resolution of the TIP102 counter is four times the resolution of the incremental encoder by counting each state transition of the encoders two phase signals A and B. This bit forces the counter back to single resolution. This bit is valid only for the TIP102 motion controller hardware.

5.1.1.3 TMF_DOUBLE_RES_BIT

The normal resolution of the TIP102 counter is four times the resolution of the incremental encoder by counting each state transition of the encoders two phase signals A and B. This bit forces the counter back to double resolution. This bit is valid only for the TIP102 motion controller hardware.

5.1.1.4 TMF_SSI_PARITY_BIT

This bit indicates, that the absolute encoder with the synchronous serial interface (SSI) generates a parity bit. This bit is valid only for the TIP111 motion controller hardware.

5.1.1.5 TMF_SSI_GRAY_BIT

This bit indicates, that the absolute encoder with the synchronous serial interface (SSI) generates gray code as position output. This bit is valid only for the TIP111 motion controller hardware.

5.1.1.6 TMF_SIMULATION_BIT

This bit indicates, that the axis shall operate in simulation mode. In this mode the motion controller firmware accepts all commands but it does not actually move the axis. All response to the user task is simulated by the motion controller firmware. This mode is most useful for debugging user task without moving axes.

5.1.1.7 TMF_ENC_DIFF_BIT

This bit indicates, that the inverted input of the encoder phase signals A, B and I shall be checked for error detection. This bit is only valid if the transition modules TIP102-TM-x1/x2/x 3 are used with the TIP102 IP.

5.1.2 Input_Polarity

This register is used to adapt the polarity of digital input signals. The motion control software performs an exclusive OR operation with the Input Polarity and the Input Register before the input signals are interpreted by software (*see also TIP102/111 Hardware User Manual*).

5.1.3 Output_Polarity

This register is used to adapt the polarity of digital output signals. The motion control software performs an exclusive OR operation with the Output Polarity and the Output Register before writing to the physical output register (*see also TIP102/111 Hardware User Manual*).

5.1.4 Sampling_Factor

This parameter can be used to reduce the sampling rate of an individual axis. For example: a sampling factor of two, causes the PID close loop control algorithm to execute every second time of the common sample period. The motion controller firmware assigns at the start up a default value of 1 to this parameter.

5.1.5 SSI_Bit_Count

The SSI Bit Count Configuration Register defines the number of data bits for the absolute encoder with a synchronous serial interface (SSI). This register is valid only for the TIP111 hardware (*see also TIP111 Hardware User Manual*).

5.1.6 SSI_Clock_Rate

The SSI Clock Rate Configuration Register defines the rate of the data clock for the absolute encoder with a synchronous serial interface (SSI). This register is valid only for the TIP111 hardware (*see also TIP111 Hardware User Manual*).

5.1.7 Kv_Command_Value, Kv_Speed_Value

This pair of parameter defines the ratio between the output command value in [mv] (*Kv_Command_Value*) and the resulting velocity of the axis in [inc/s] (*Kv_Speed_Value*). You can determine the parameter at any speed, only the ratio is decisive. The precision of movements depends on the accuracy of these parameter.

5.1.8 Kp, Kp_2

The Kp Configuration Register defines the P-constant of the PID close loop control algorithm. Usually the range of the P-constant is between 1.0 and 3.0. A value of 1.5 is always a good starting point.

5.1.9 Ki, Ki_2

The Ki Configuration Register defines the I–time constant of the PID close loop control algorithm. A good start value to determine the I–constant of the system is 1.0.

5.1.10 Kd, Kd_2

The Kd Configuration Register defines the D–time constant of the PID close loop control algorithm. A good start value to determine the D–constant of the system is 0.01.

5.2 Function Specific Parameter

Some of the parameters are specific to a certain function which can be executed by the motion controller firmware. These parameters may be modified before a single function execution.

5.2.1 Axis_Command

This parameter contains the code of the next axis command to execute.

Valid commands are (symbols defined in TMFSHARE.H):

1	TMF_EMERGENCY_STOP_FUNCTION
2	TMF_MOVE_FUNCTION
3	TMF_SWING_FUNCTION
4	TMF_REFERENCE_FUNCTION
5	TMF_INITIAL_FUNCTION
8	TMF_STOP_FUNCTION
9	TMF_JOG_UP_FUNCTION
10	TMF_JOG_DOWN_FUNCTION
11	TMF_PATH_FUNCTION
15	TMF_JOYSTICK_FUNCTION
16	TMF_GO_UP_FUNCTION
17	TMF_GO_DOWN_FUNCTION

5.2.2 Axis_New_Command

This parameter must be set to TRUE (1) before each command execution to signal the motion control software a new command. Immediately after startup of the new command the motion control firmware set this parameter to FALSE (0).

5.2.3 Axis_Status

The parameter contains the actual execution status of the axis. If this parameter contains a ‘1’ (TMF_BUSY) the axis is busy and executing a command, otherwise this parameter is set to ‘0’ (TMF_READY).

5.2.4 Axis_Error

This parameter contains a ‘0’ (TMF_OK) if execution of the last axis command has been successful or an appropriate error code if the execution has failed. *(see also 6.1 Status and Error Codes)*

5.2.5 Completion_SemID

This parameter contains the semaphore ID of the applications completion semaphore. Every time the motion control software finishes a command execution for the axis, this semaphore will be signaled. *(see also 3.4 Completion Semaphores)*

5.2.6 Axis_Type

This parameter contains the type of the assigned axis.

0	TMF_NO_AXIS	No axis assigned
1	TMF_INC_AXIS	Incremental interface axis (TIP102)
2	TMF_SSI_AXIS	Absolute encoder interface axis (TIP111)

5.2.7 Actual_Position

Actual position of the axis in [inc].

5.2.8 Actual_Speed

Actual speed of the axis in [inc/s]. This parameter is only updated for commands with closed loop control.

5.2.9 Actual_Error_Position

This parameter contains the position difference [inc] between the current position and desired position for the actual sampling cycle.

5.2.10 Control_Reg

This register displays the actual state of execution of the axis.

The bits are defined as follows (see also TMFSHARE.H) :

Bit 0 = 1	Emergency stop function is active
Bit 1 = 1	Reference function is active
Bit 2 = 1	Initial function is active
Bit 3 = 1	Move, Jog or Path function are active
Bit 4 = 1	Joystick function is active
Bit 5 = 1	Stop function is active
Bit 6 = 1	Go function is active

5.2.11 Status_Reg

Only for internal use.

5.2.12 Target_Position

This Parameter defines the target position of a movement in increments [inc]. This parameter must be set before the execution of the Move Function.

5.2.13 Target_Pos_2

This parameter defines the 2nd target position in increments [inc], it is used for the Swing Function.

5.2.14 External_Path_Position

Next target position for the axis, calculated from an external application function. *(see also 4.10 External Path Function)*

5.2.15 Input_Reg

Contents of the TIP102/111 input register after polarity correction by the input polarity register (*see also TIP102/111 Hardware User Manual*).

5.2.16 Output_Reg

Contents of the TIP102/111 output register before polarity correction by the output polarity register (*see also TIP102/111 Hardware User Manual*).

5.2.17 ADC_Input_Reg

Contents of the ADC data register [ADC bits] (*see also TIP102/111 Hardware User Manual*).

5.2.18 Command_Output

Output value for the DAC data register [DAC bits]. This parameter is calculated by the closed loop control algorithm (*see also TIP102/111 Hardware User Manual*).

5.2.19 Reference_Configuration

The Reference Configuration Register defines the modes of the operation of the reference function with set of bits. This register is only valid for the TIP102 hardware.

The following bits are used:

5.2.19.1 TMF_REF_SWITCH_BIT

This bit defines, that a reference switch is connected to the system.

5.2.19.2 TMF_REF_DIRECTION_BIT

This bit defines the direction in which the reference function starts looking for the reference condition.

5.2.19.3 TMF_AUTO_REF_BIT

This bit enables the automatic reference mode.

5.2.19.4 TMF_REF_SPOT_BIT

This bit defines, that the reference switch is in the ‘on-state’ only for a single zero (index) pulse of the encoder.

5.2.19.5 TMF_NO_CLOSE_LOOP_BIT

This bit defines, that the movement of the axis during the reference function shall execute with a constant speed command and not in close loop control with the PID algorithm.

5.2.20 Limit_Switch_Configuration

The Limit Switch Configuration Register defines, how the motion control firmware reacts, if one of the limits switches is activated.

The following bits are used:

5.2.20.1 TMF_LIM_EMER_STOP_BIT

This bit defines, that the Emergency Stop Function shall be activated, if the axis runs into the limit switch.

5.2.20.2 TMF_LIM_STOP_BIT

This bit defines, that the normal Stop Function shall be activated, if the axis runs into the limit switch.

5.2.20.3 TMF_LIM_FAST_STOP_BIT

This bit defines, that the Fast Stop Function shall be activated, if the axis runs into the limit switch.

5.2.20.4 TMF_LIM_JOY_ENABLE_BIT

This bit defines, that the Joystick Function is a valid function, if the axis is in limit switch position.

5.2.20.5 TMF_LIM_JOG_ENABLE_BIT

This bit defines, that the Jog Function is a valid function, if the axis is in limit switch position.

5.2.20.6 TMF_LIM_MOVE_ENABLE_BIT

This bit defines, that the Move Function towards a position within normal working rage is a valid function, if the axis is in limit switch position.

5.2.20.7 TMF_LIM_REFER_ENABLE_BIT

This bit defines, that the Reference Function is a valid function, if the axis is in limit switch position.

5.2.20.8 TMF_LIM_GO_ENABLE_BIT

This bit defines, that the Go Function is a valid function, if the axis is in limit switch position.

5.2.21 Software_Maximum_Position

This parameter in [inc] limits movements in positive direction. If the axis crosses this maximum position an error (*E_TMF_SOFT_LIMIT*) occurs and the axis is stopped

5.2.22 Software_Minimum_Position

This parameter in [inc] limits movements in positive direction. If the axis crosses this minimum position an error (*E_TMF_SOFT_LIMIT*) occurs and the axis is stopped

5.2.23 Window_In_Position

This parameter defines a window around the target position. If the axis reaches this window the motion control software signals the completion semaphore for this axis (the axis is *In-Position*).

Example

```
Window_In_Position [inc] = d
Target_Position [inc]   = X

Window : (X-d) <= X <= (X+d)
```

5.2.24 Reference_Position

The Reference Position Configuration Register is used to define the absolute position which shall be loaded into the counter chip of the TIP102, if the reference condition occurs. This register is valid for the TIP102 hardware only.

5.2.25 Disable_Delay

The Disable Delay Configuration Register defines after how many sampling periods the servo amplifier enable output will be switched off, if the Emergency Stop Function is executed.

5.2.26 Position_Error_Limit

This parameter contains the maximum allowed position error in [inc]. If the axis passes this level the axis is stopped (Emergency Stop) with an error (E_TMF_POS_DIFF). This function is only active, if this parameter contains a value greater than 0.

5.2.27 Command_Offset

This parameter compensates the offset error of the servo drive. To do this you must stop the axis under closed loop control. The resulting command output value *(see also 5.2.18 Command_Output)* is the command offset of this axis. The command offset compensation allows you to hit the target position with high accuracy using a simple P–closed loop algorithm.

5.2.28 AccelComp

Reserved.

5.2.29 Parameter_Set_2_Distance

This parameter defines maximum distance between actual and target position in [inc] to use the velocity and acceleration parameter set 2 instead of set 1.

5.2.30 Start_Acceleration

This parameter defines the acceleration of the start phase in [inc/s^2] for the next Move Function.

5.2.31 Brake_Acceleration

This parameter defines the deceleration of the stop phase in [inc/s^2] for the next Move Function.

5.2.32 Speed_Maximum

This parameter defines the maximum velocity in [inc/s] for the next Move Function.

5.2.33 Start_Accel_2

This parameter defines the acceleration of the start phase in [inc/s^2] for the next Move Function. This parameter is used instead of the parameter *Start_Acceleration*, if the distance of a move is shorter than defined in parameter *Parameter_Set_2_Distance*.

5.2.34 Brake_Accel_2

This parameter defines the deceleration of the stop phase in [inc/s^2] for the next Move Function. This parameter is used instead of the parameter *Brake_Acceleration*, if the distance of a move is shorter than defined in parameter *Parameter_Set_2_Distance*.

5.2.35 Speed_Max_2

This parameter defines the maximum velocity in [inc/s] for the next Move Function. This parameter is used instead of the parameter *Speed_Maximum*, if the distance for a move is shorter than defined in parameter *Parameter_Set_2_Distance*.

5.2.36 Stop_Acceleration

This parameter defines the deceleration in [inc/s^2] used for the next Stop Function.

5.2.37 Reference_Speed

The Reference Speed Configuration Register is used to define the speed in increments per second of the axis, which is used during the execution of the Reference Function. This register is valid only for the TIP102 hardware.

5.2.38 Joystick_Idle_Window

This parameter defines a window around the zero position of ADC input in percent from full-scale range. If the Joystick Function is active and the ADC input value is inside this window the axis is stopped. Valid range 0..100%.

5.2.39 Joystick_Linear_Window

This parameter defines a window around the zero position of the ADC input in percent form full-scale range. Inside this the velocity is linear. Valid range 0..100%

5.2.40 Joystick_Linear_Speed

This parameter defines the maximum speed of the linear window in [inc/s].

5.2.41 Joystick_Maximum_Speed

This parameter defines the maximum speed of the joystick [inc/s].

5.2.42 Jog_Up_Speed

Absolute velocity [inc/s] of the Jog Up Function.

5.2.43 Jog_Down_Speed

Absolute velocity [inc/s] of the Jog Down Function.

5.2.44 Go_Up_Command_Value

Absolute command output value [DAC bits] of the Go Up Function.

5.2.45 Go_Down_Command_Value

Absolute command output value [DAC bits] of the Go Down Function.

5.2.46 PID_Switch_Position

This parameter defines the distance [inc] to the target position, from which the 2 nd PID parameter set is used.

5.3 Axis Common Parameter

5.3.1 BaseSamplePeriod

This parameter contains the sampling rate for the closed loop algorithm in [µs]. *(see also 2.2.2 Setup the PID Time Base)*

5.3.2 version

This parameter contains the actual software version of the motion control firmware as a character string (for example “1.00”)

5.3.3 DebName[]

Only for internal use.

5.3.4 Debwert[]

Only for internal use.

5.3.5 TEST

This parameter is incremented every sampling period.

5.3.6 Protocol_Axis_Number

This parameter defines the axis [1..8] for recording process data.

5.3.7 number_behind

Used for data recording.

5.3.8 counter_behind

Used for data recording.

5.3.9 mark_index

Used for data recording.

5.3.10 tab_index

Used for data recording.

5.3.11 inpos_index

Used for data recording.

5.3.12 tab_count

Used for data recording.

5.3.13 mark_flag

Used for data recording.

5.3.14 init_flag

Used for data recording.

5.3.15 tab[][]

Buffer for recorded process data.

6 Appendix

The following symbols are predefined int the include file TMFSHARE.H

6.1 Status and Error Codes

TMF_OK	0x000	No error
E_TMF_NOAXIS	0x900	No physical axis controller assigned to this axis
E_TMF_ENCODER	0x901	An error in the timing of the encoder phase signals has occurred or the encoder itself has reported an error
E_TMF_REF1	0x902	Hardware limit was switch reached during reference search.
E_TMF_REF2	0x903	No index impulse found before reaching the hardware limit switch (with reference switch).
E_TMF_REF3	0x904	No index impulse found before reaching the hardware limit switch (without reference switch).
E_TMF_OVERLOAD	0x908	The maximum command value was passed longer than 10 sampling cycles.
E_TMF_POS_DIFF	0x909	The actual position error is greater than the maximum allowed. *(see also 5.2.26 Position_Error_Limit)*
E_TMF_LIMIT_SWITCH_FUNC	0x90A	Illegal function at hardware limit switch.
E_TMF_LIMIT_CONFIG	0x90B	Illegal limit switch configuration. *(see also 5.2.20 Limit_Switch_Configuration)*
E_TMF_LIMIT_MOVE_DIR	0x90C	The direction to leave the limit switch is illegal.
E_TMF_HARD_LIMIT	0x90D	The axis has reached the hardware limit switch.
E_TMF_SOFT_LIMIT	0x90E	The actual position is out of range. *(see also 5.2.21 Software_Maximum_Position and 5.2.22 Software_Minimum_Position)*
E_TMF_BAD_PARAM	0x90F	An axis specific parameter used for execution of this axis command is out of range (for example velocity, acceleration, position and so on)
E_TMF_ABORT	0x910	The active function can't be aborted by this function.
E_TMF_PERMIT_1	0x911	Initialize the axis first.
E_TMF_PERMIT_2	0x912	Execute the reference function first.
E_TMF_PERMIT_3	0x913	Axis isn't under closed loop control, the new function is illegal.
E_TMF_PERMIT_4	0x914	Axis is under closed loop control, the new function is illegal.