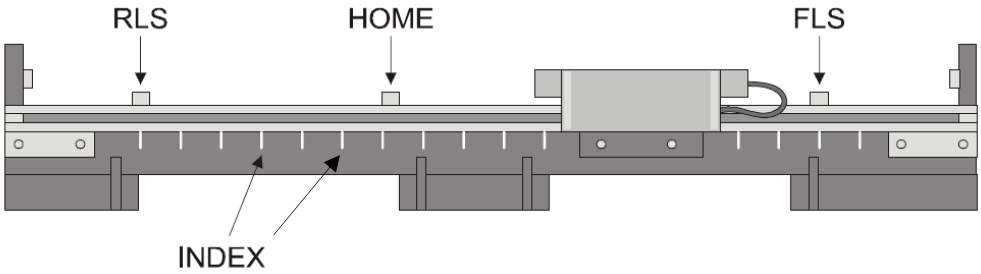


Application Note

Homing in SimplIQ Servo Drives

Rev. 8
June 2010

Elmo
Motion Control
www.elmomc.com

Introduction

Most servo applications use a relative (incremental) encoder as feedback for the controller and servo drive. Incremental encoders are relatively low cost, easy to use and they have high resolution. However, they do not provide any information regarding the absolute position of the system (the position of the moving parts of the machine in relation to the static parts) after power up, even though most applications require this information.

A sequence of predefined motions is normally required in order to locate the system's absolute position after power up. This sequence is generally referred to as *homing*.

The homing phase is carried out by searching for an absolute known sensor along the mechanical travel, and updating the internal's position accordingly. Once the absolute position has been registered in the drive, the incremental (relative) information of the encoder can be used to measure motion relative to this absolute position.

Scope

This application note describes a number of implementation methods for performing the homing sequence with Elmo's digital drives. This Application Note focuses on the special HM[n] command that is related to this procedure, and a number of examples show how to implement it in a user program.

Homing in SimplIQ

SimplIQ servo drives include up to two encoder inputs for a single axis. These inputs have the following purposes:

1. Synchronization control with other axes, such as gearing control, ECAM, etc.
2. Performance improvement by dual loop control algorithms.

Elmo drives use special functions to manipulate the position counters (PX, PY) in real time which ensures that no count is missed when the counter is manipulated. The HM[n] and HY[n] commands are used for changing the main and auxiliary counters respectively.

During initialization, an event (index, limit or home sensor) is configured for capturing the position, and then the motor is moved towards the event. Once the event has been triggered, the drive captures the position of both encoder counters: PX at register HM[7] and PY at HM[8].

Configuring the Home Command HM[n]

Step 1: Configure the Home Trigger Event HM[3]

The Home event can be configured to be triggered in one of the following ways:

1. Immediate: HM[3] = 0
   An immediate event captures PX and PY in registers HM[7] and HM[8] respectively.

2. Upon receiving a hardware input: HM[3] = 1..20
   With this configuration the position counters PX and PY are captured in HM[7], HM[8] according to a defined hardware input as follows:

   - Home Sensor*: HM[3] = 1..2
     Configures the homing function to be triggered on the Home sensor. This option can only be used if the Home sensor has already been defined by the IL[n] command. Note that only input 5 can be defined as the Home Sensor.

   - Encoder Index: HM[3] = 3..4
     Configures the homing function to be triggered by the Index.

   - Limit switch: HM[3] = 5..8
     Configures the homing function to be triggered by the Limit switch (RLS or FLS). This option can only be used if the Limit switch has already been defined by the IL[n] command.

   - General Purpose In: HM[3] = 9..28
     Defines one of the general purpose inputs as the Home event trigger.

   For a detailed table and more information on the HM[n] command, refer to the SimplIQ Command Reference Manual which is available on Elmo's web site.

Step 2: Configure How the PX Counter will be Changed

When the home event has been triggered, there are a number of possibilities regarding the PX position counter. The options are governed by HM[5]:

- HM[5] = 0: The PX counter will be set to a predefined constant number HM[2] (PX = HM[2]) – sets an absolute number.

- HM[5] = 1: The PX counter will be subtracted by a constant number (PX = PX - HM[2]) – sets a relative number.

- HM[5] = 2: The PX counter is not changed.

* In5 or In6 can be assigned as the Home Sensor or Auxiliary Home Sensor by the IL[n] command. Once assigned, the input is no longer referred to as General Purpose, but as a Home Sensor that can be captured in real-time and initiate an AUTO_HM auto routine.

Step 3: Configure the Post Homing Action

Configuring the post homing action is carried out via the HM[4] parameter.

Besides capturing the value of the PX and PY counters, the homing function can be defined to initiate a Stop or to set a general purpose output when triggered.

- HM[4] = 0: Stop motion after homing. This is only applicable in the following modes:
  - Position mode UM = 4, UM = 5
  - Velocity mode UM = 2
  - Stepper mode UM = 3.
- HM[4] = 1: Set digital output equivalent to HM[6] (OP = HM[6]).
- HM[4] = 2: Take no action.

Step 4: Start the Homing Function HM[1]=1

Setting HM[1] will arm the homing function according to the way it was configured in Steps 1–3 above. When the event has been triggered HM[1] is cleared automatically.

Response Time Considerations

The HM[n] command response time can be divided into two cases, depending on the source of the Homing trigger and the method used by the drive to capture the position at the Homing event:

Asynchronous Homing: The main encoder position is captured by hardware immediately, when the Homing event occurs. This is when the trigger is set to Index/Reference mark, or Home sensor (0 < HM[3] < 5).

Synchronous Homing: The capture is performed by the drive's real time software, so it is synchronized with the next position loop sampling. This is when the trigger is set to General Purpose Input, or to RLS/FLS (4 < HM[3] < 29).

Asynchronous Capture (Hardware Based Capturing)

When the Homing trigger is set to Index, or Home sensor, the capture is executed immediately. The only delay is the hardware interface delay, which is less than 5 µs for the Home sensor and negligible for the Index.

Synchronous Capture (Software Based Capturing)

When the homing trigger is set to GP Input, or Limit Switch (RLS/FLS) the total delay is a combination of the following:

1. Hardware interface delay – This is less than 5 µs, which can be negligible.
2. Synchronization delay – The trigger is handled in the next position loop sample, therefore the delay can be anywhere in the range of 0 – 4*TS. The default value of TS is 90, so by default the range is 0 – 360 µs.
3. Input filter delay - a delay due to the programmable filter IF[n].

The capture delay is calculated according to the following formula:

$$IF[N] \le \mathrm{d[ms]} \le (0.004 * \mathrm{TS} + \mathrm{IF[N]})$$

where

d is the input response time delay, in milliseconds

TS is the torque controller sampling time, in microseconds

IF[N] is the input filter width, in milliseconds.

The delay, d, will affect the trigger accuracy by missing a few counts.

The maximum number of missed counts can be estimated by the formula:

$$\mathrm{MissCount[counts]} = \mathrm{VX} * \mathrm{d}/1000$$

where

MissCount is the number of missed counts

VX is the velocity, in counts/second.

An example of calculating the number of missed counts appears below:

TS = 80	The torque controller sampling time, in microseconds
IF[1] = 2	The input filter width for digital input #1, in milliseconds
VX = 20,000	The speed of the main motor main speed, in counts/second.

When homing on digital input #1, the expected number of misses is calculated as follows:

$$d(\max) = 0.004 * 80 + 2 = 2.32ms$$

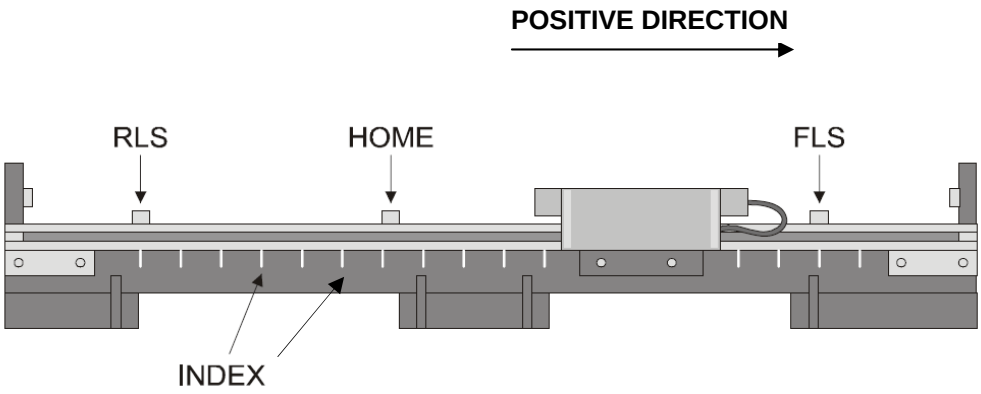
$$MissCount(\max) = \frac{20000 * 2.32}{1000} = 46.4counts$$

Homing Examples

Example 1: Homing with the Home Switch and Index

The following homing algorithm may be used for this very common switch arrangement:

1. Start the motor.
2. Jog back (reverse) until RLS.
3. Jog at forward speed until home.
4. Look for the next index and set the position there to 0.



The position setting is taken by the index because in many applications the index is much more accurate than the Home switch. The main purpose of the Home switch is to resolve index ambiguity, as many index pulses may occur along the path of travel, especially when using rotary motors.

In normal operation the FLS and RLS serve as emergency indicators, and the motor is not expected to reach them. RLS is visited during the homing process in order to determine the direction where the Home switch can be located.

The following user program performs the homing algorithm:

```
function [int status] = homing1(int TimeOut)
/*
Homing routine.
The function goes to RLS, change direction and go to Home sensor, then without
stopping goes to Index and stop.
Input: TimeOut for failure (in mSec)
Output: status=1 if o.k., otherwise a negative error code
Assumptions: RLS,FLS,HOME already programmed by IL[n] command
*/
    int OldMi;

    // Go reverse until limit switch
    OldMi=MI;MI=MI|0x16;            // Prevent operational AUTO_RLS routine
                                    //  while in homing process
    HM[3]=7;HM[4]=0;HM[5]=2;HM[1]=1;// Configure  homing for RLS, stop after
                                    //  homing, don't initialize counter
    MO=1; JV=-10000;BG;             // Go to the reverse
    status = WaitRLS(TimeOut);      // wait for RLS / timeout
    if(status<=0)                   // check for timeout
        ST; goto ##LastLine;        // if timeout error, go to ##LastLine
    end

    // Go towards Home sensor - Do not stop
    HM[3]=1;HM[4]=2;HM[5]=2;HM[1]=1;// configure homing for Home sensor, don't
                                    //  stop, don't change counters
    JV=-JV; BG;                     // move back
    status = WaitHome(TimeOut);     // wait for home sensor / timeout
    if(status<=0)                   // check for timeout
        ST; goto ##LastLine;        // if timeout error, go to ##LastLine
    end

    // Finallyset position by index and stop
    // [note]: Index instance should be in safe distance from Home sensor. this
    //         is due to program execution time (the motor does not stop at Home)
    HM[2]=0;HM[3]=3;HM[4]=0;HM[5]=0;// configure homing for index, stop after
    HM[1]=1;                        //  detection , initialize counter
    status = WaitHome(TimeOut);     // wait for index / timeout
    if(status<=0)                   // check for timeout
        ST; goto ##LastLine;        // if timeout error, go to ##LastLine
    end

    status=1;                       // Success return

    ##LastLine;
    MI=OldMi;                       // Restore AUTO_RLS routine status
    return
```

Click to download the above code: [with annotations] [no annotations, compact file]

Note that this function uses MI to prevent RLS and Home from activating the AUTO_RLS and AUTO_HM routines respectively. Another possible approach would be to use IL[N] to change the functionality of the switch to GPI (general purpose input) and then home on the GPI. However, with the latter approach, the programmer must know which connector pin is programmed as RLS.

The following auxiliary functions were used with the algorithm:

```
function [int status]=WaitRLS(int TimeOut)
/*
The function waits until RLS=1, or until too much time has elapse.
The function also returns termination code (status)
*/
    int StartTime ;

    status = -1;                    // prepare value to be  returned
    StartTime = TM;
    while ( !(IB[11]) )             // IB[11] is the logic value of RLS
        if ( tdif(StartTime) >= TimeOut)// check timeout
            return                  // Timeout elapsed! Terminate the
        end                         //   function
    end
    status = 1 ;                    // Report normal termination
    return

function [int status]=WaitHome(int TimeOut)
/*
The function waits until HM[1]=0 (Homing is finished), or until
 too much time Has elapsed. The function also returns termination
 code (status)
*/
    int StartTime ;

    status = -1;                    // prepare value to be  returned
    StartTime = TM;
    while (HM[1])                   // loop until homing func. finish
        if ( tdif(StartTime) >= TimeOut)// check timeout
            return                  // Timeout elapsed! Terminate the
        end                         //   function
    end
    status = 1 ;                    // Report normal termination
    return
```

Click to download the above code: [with annotations] [no annotations, compact file]

Note: The downloaded file is identical to the file that can be downloaded on page 7 above, as the code for both examples is in the same file. After opening the downloaded file, scroll down to view the code in this example.

Example 2: Double Homing Corrects Backlash Offsets

This example demonstrates homing on the Home switch without using the Index. In many gear systems, the Index signal cannot be used for homing for the following reasons:

- Motor or gear repairs may require tuning of the index position. When a motor or gear are replaced, the absolute position of the index is changed and needs to be "re-learned" in the field.
- Backlash and gear compliance prevent accurate mapping of the motor position to the load.

In order to prevent compliance and timing errors, the position of the Home switch is captured twice, with alternating movement directions. The two captured results are averaged in order to cancel out error sources.

If PX = 10,000 is the absolute position of the middle of the Home switch, the homing formula is:
PX = (PX - 0.5 * (PX at right home edge + PX at left home edge)) + 10,000.

The following user program executes this routine, using the auxiliary functions of the previous example:

```
function [int status] = homing2(int TimeOut)
/*
Homing routine.
Input: TimeOut for failure
Output: status=1 if o.k., otherwise a negative error code
Assumptions:
RLS,FLS,HOME already programmed by IL[n] commands
*/
int OldMi,Pos1;

    /* Go reverse until limit switch */
    OldMi=MI;MI=MI|0x16;              // Prevent operational AUTO_RLS routine
                                      //  while in homing process
    HM[3]=7;HM[4]=0;HM[5]=2;HM[1]=1;// configure homing for RLS, stop after
                                      //  homing, don't initialize counter
    MO=1; JV=-1000; BG;           // Go to the reverse
    status = WaitRLS(TimeOut);        // wait for RLS / timeout
    if(status<=0)
        goto ##LastLine2;             // if timeout error, go to ##LastLine
    end

    /* Go until falling home switch, stop, and capture position*/
    HM[3]=2;HM[4]=0;HM[5]=2;HM[1]=1;// configure homing on Home sensor, stop
                                      //  after motion, don't initialize counter
    JV=-JV;BG;                        // move back
    status = WaitHome(TimeOut);       // wait for home sensor / timeout
    if(status<=0)
        goto ##LastLine2;             // if timeout error, go to ##LastLine2
    end
    Pos1=HM[7];                       // store value of the captured home
                                      //  sensor for later calculation

    /* Sample home switch from other side */
    HM[1]=1; JV=-JV;BG;               // move back; arm home func.
    status = WaitHome(TimeOut);       //
    if(status<=0)
        goto ##LastLine2;
    end

    HM[2]=(HM[7]+Pos1)/2 - 10000;     // Final calculation - set immediate
                                      //  difference correction to PX
    HM[3]=0;HM[4]=2;HM[5]=1;HM[1]=1;
    status=1;                         // Success return

    ##LastLine2
    MI=OldMi;                         // Restore AUTO_RLS routine status
return
```

Click to download the above code: [with annotations] [no annotations, compact file]

On-the-fly Position Counter Updates

Updating a position sensor during the homing process has no effect on motion in UM=1, 2 or 3 because these modes do not use position feedback. The effects of a position sensor update (PY in UM=4 and PX in UM=5) on the homing process depend on the mode.

In PTP motions, the remaining motion to target becomes longer or shorter (refer to the example below). If the home correction is made in a constant speed range of the PTP motion, the redesign of the motion path may be hardly visible. This mechanism enables registration and final motion corrections on-the-fly.

In jog motions, the position command is jumped according to the position feedback, so that the motion is unaffected by the position counter update.

If the software position reference generator stops or has already stopped, the software position command is corrected according to the position feedback. The motion is unaffected by the position counter update.

In PVT or PT motions, an on-the-fly position counter update may lead to an immediate high position error, and the motor may abort with an excessive position tracking error exception.

For UM=5, if the auxiliary encoder counter is modified, and the following are true:

1. The motor tracks the auxiliary encoder with no ECAM table (RM=1, FR[3] is non-zero and EM[1]=0).
2. The software position generator is idle, jogging or running PTP,

then the software position reference will be modified so that the motion is not affected. In PTP mode, the PA parameter will change automatically to reflect the modification.

Example

A PTP motion starts with PA = PX = 0. After setting PA = 1000; BG, a motion 1000 counts long is expected. If at PX = 500 the position has been reset to zero through homing, there should be 1000 counts remaining immediately after the homing. The total length of the motion becomes 1500.

For more information on Elmo:

Elmo Motion Control Ltd.	Elmo Motion Control Inc.	Elmo Motion Control GmbH	Elmo Motion Control APAC
64 Gisin St., P.O. Box 463 Petach Tikva 49103 Israel	42 Technology Way, Nashua NH 03060 USA	Steinkirchring 1 D-78056, Villingen-Schwenningen Germany	#807, Kofomo Building, 16-3 Sunae-dong, Bundang-gu, Sungnam-si, Kyunggido South Korea
Tel: +972 (3) 929-2300 Fax: +972 (3) 929-2322 info-il@elmomc.com	Tel: +1 (603) 821-9979 Fax: +1 (603) 821-9943 info-us@elmomc.com	Tel: +49 (0) 7720-85 77 60 Fax: +49 (0) 7720-85 77 70 info-de@elmomc.com	Tel: (82)-31-698-2010 Fax: (82)-31-98-2013 info-asia@elmomc.com

www.elmomc.com