■ Introduction

This application note describes how to calculate the value of the velocity feed forward parameter (FFVEL) in the SyncPos motion controller. The formula for the calculation is presented and a small SyncPos program for automatic calculation of FFVEL is included.

■ Calculation of velocity feed forward

When calculating velocity feed forward, the following formula is used:

FFVEL = (62914560000 * #104) / (V_{NOM} * E_{RES} * Tsample * #205).

#104:	denotes the value of the VLT5000 #104 parameter (nameplate motor frequency)
#205:	denotes the value of the VLT5000 #205 parameter (maximum reference)
V_{NOM}:	is the nominal encoder velocity at nominal motor frequency. The unit is RPM.
E_{RES}:	is the number of quad-counts per encoder revolution.
Tsample:	is the PID controller sampling interval (TIMER).

NB!
In SYNCV (velocity synchronizing) mode, commanded velocity is calculated each msec. independent of TIMER. Therefore "tsample" must always be 1 when calculating FFVEL in SYNCV mode.

<u>**Example**</u>:

Motor:	nominal frequency 50 Hz / nominal speed 1380 RPM
Encoder:	incremental encoder (4096 pulses per revolution) mounted directly on motor shaft
Maximum ref.:	VLT5000 parameter # 205 is set to 70 Hz.

SyncPos PID controller sampling interval at default value (1 ms).

FFVEL = (62914560000 * **50Hz**) / (**1380 RPM** * **4*4096 * 1 * 70 Hz**) = <u>1988</u>.

Now the sample interval is changed to 5 ms. The correct value of FFVEL is now.

FFVEL = (62914560000 * **50Hz**) / (**1380 RPM** * **4*4096 * 5 * 70 Hz**) = <u>398</u>.

■ FFVEL calculation program

The following SyncPos program will automatically calculate FFVEL:

```
LINKGPAR 131 710 “ENCODER RPM” 0 1000000 1

main:

        fnom = GETVLT 104
        refmax = GETVLT 205 % 1000
        vnom = GET 131
        IF (GET ENCODERTYPE == 0) THEN
             eres = (4 * GET ENCODER)
        ELSE
             eres = (GET ENCODER)
        ENDIF
        tsample = (GET TIMER)
        FeedForwardVel = (6291456 * fnom % refmax
        % vnom *10000 % eres % tsample)
        SET FFVEL FeedForwardVel

GOTO main
```

Introduction

When programming time critical tasks in SyncPos, it is important to know the execution times of the individual instructions. This note investigates the command execution times under different operating conditions of the SyncPos, and contains an experimental evaluation of the maximum sampling frequency possible with the ON PERIOD command. Execution times of commonly used arithmetic and logic expressions are also investigated.

SyncPos Command Execution Times

Command	ms	Command	ms.	Command	ms.	Command	ms.	Command	ms.
ACC	1,041	ERRCLR	0,711	LINKGPAR	20,91	PID	0,540	SYNCSTAT	0,416
APOS	0,395	ERRNO	0,341	MAPOS	0,345	POSA	1,659	SYNCV	1,051
AVEL	0,855	GET	0,340	MAVEL	0,800	POSR	2,131	TIME	0,358
AVEL (100ms)	0,858	GOSUB	0,322	MAVEL (100ms)	0,808	PRINT	6,358	TRACKERR	0,432
AXEND	0,366	GOTO	0,165	MIPOS	0,345	PULSACC	0,631	VEL	1,070
COMOPTGET	0,507	IF..THEN..ENDIF (min)	0,466	MOTOR OFF	0,658	PULSVEL	0,842	_GETVEL 50	50,71
COMOPTSEND	0,541	IF..THEN..ENDIF (std.)	1,346	MOTOR ON	0,658	RST ORIGIN	0,208		
CONTINUE	0,236	IN	0,545	MOTOR STOP	0,658	SAVEPROM	3764		
CPOS	0,395	INAD 53	0,534	LOOP	0,363	SET	1,394		
CSTART	0,658	INAD 60	0,536	NOWAIT	0,179	SETVLT	20,91		
CSTOP	0,711	INB 0	0,534	OUT	0,553	SET ORIGIN	0,413		
CVEL	0,989	INB 1	0,531	OUTAN	0,326	STAT	0,519		
DEC	1,034	INKEY (-1)	0,516	OUTB 0	0,539	SYNCERR	0,424		
DEFORIGIN	0,711	IPOS	0,395	OUTB 1	0,541	SYNCM	1,496		
DELAY 10	9,658	LINKAXPAR	20,88	OUTDA	0,547	SYNCP	1,079		

Enabling CVEL (trajectory and PID calculations) increases command execution time by 13% compared with the MOTOR OFF situation. POSA, POSR, SYNCP, SYNCM increases command execution time by 52-53%. Enabling the VIRTUAL MASTER increases command execution time by 5-6%.

The test shows that there is a small difference in the command execution times on different SyncPos option cards. The test results in the table above are measured under “MOTOR ON” conditions, therefore the execution time will be approximately 13% faster in the "MOTOR OFF" situation.

A variable assignment takes 338µs. Arithmetic and bit-wise operations generally take 335µs. It does not affect the execution time whether the operands are constants or variables.

The test shows that the declaration and administration itself of ON PERIOD subroutines does not take up measurable CPU resources.

Test design

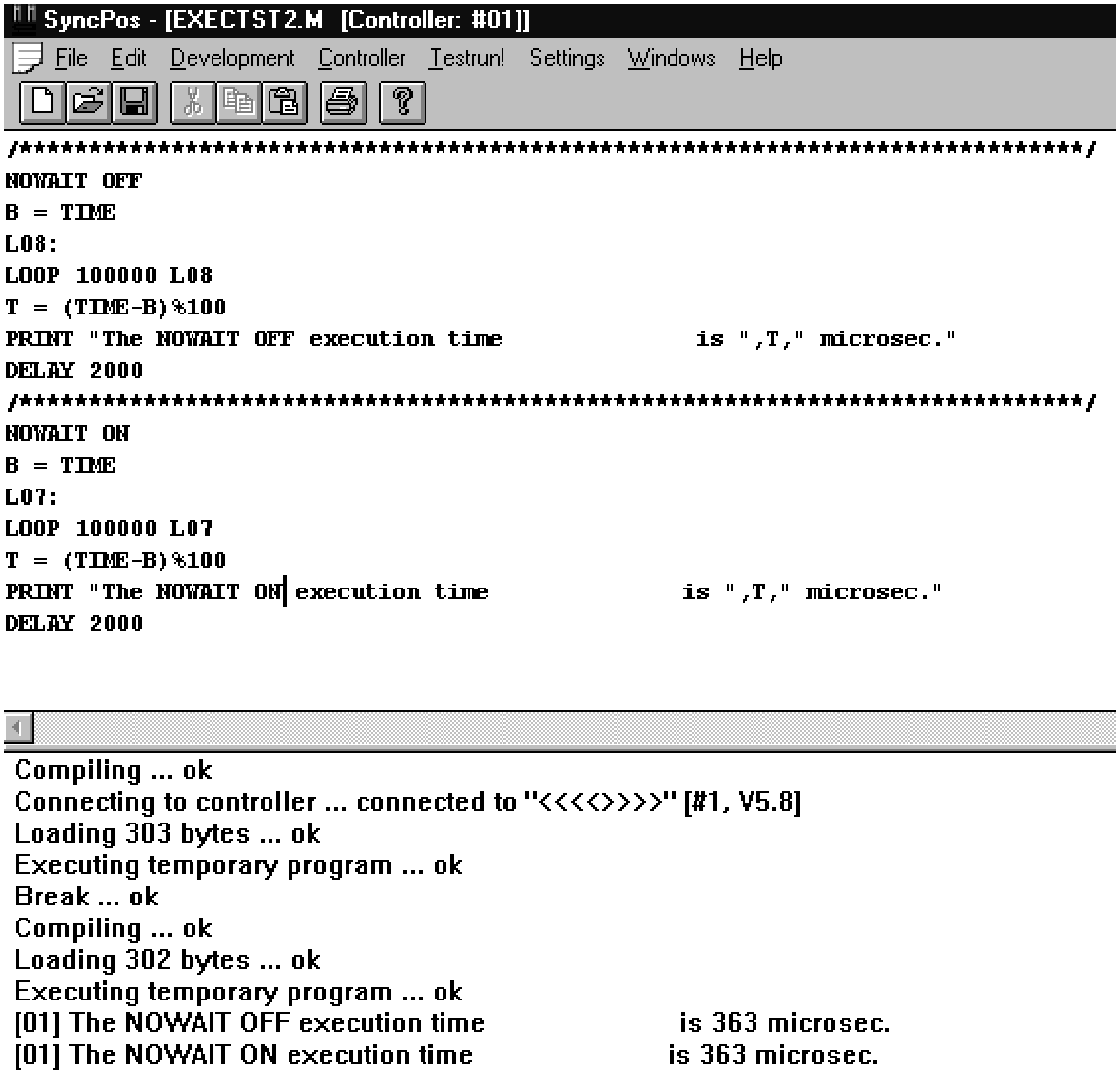
The test is carried out using the TIME function since this is the only way to get a precise measurement of the execution times. The time function answers the questions:
1) When does one command start ?
2) When is it finished ?
3) What is the time difference between these two events ?

Each command is executed several times in order to increase the resolution of the measurements. This is done with the use of the LOOP instruction (see the test program below). However the LOOP function itself also takes some time to execute and so does the TIME instructions, so the real execution time is a little shorter than the mere difference between the start time and the end time. The test program below was used to test this time delay.

A time delay of 363 micro seconds are thus subtracted from the execution time of every instruction in the main test program.

Several conditions exist where the SyncPos will operate with different latencies. The tested conditions are:

- Operation with MOTOR OFF
- Operation with CVEL
- POSR
- POSA
- Operation with SYNCP
- Operation with SYNCM
- Operation with SYNCP w/ VIRTUAL MASTER
- Operation with SYNCM w/ VIRTUAL MASTER


SyncPos - [EXECTST2.M [Controller: #01]]
File Edit Development Controller Testrun! Settings Windows Help

```
/*********************************************************************************/
NOWAIT OFF
B = TIME
L08:
LOOP 100000 L08
T = (TIME-B)%100
PRINT "The NOWAIT OFF execution time              is ",T," microsec."
DELAY 2000
/*********************************************************************************/
NOWAIT ON
B = TIME
L07:
LOOP 100000 L07
T = (TIME-B)%100
PRINT "The NOWAIT ON execution time               is ",T," microsec."
DELAY 2000
```

```
Compiling ... ok
Connecting to controller ... connected to "<<<<>>>>" [#1, V5.8]
Loading 303 bytes ... ok
Executing temporary program ... ok
Break ... ok
Compiling ... ok
Loading 302 bytes ... ok
Executing temporary program ... ok
[01] The NOWAIT OFF execution time              is 363 microsec.
[01] The NOWAIT ON execution time              is 363 microsec.
```

Some commands are not tested, either because it is not possible to test them with the TIME function or because they are usually only executed once during startup and thus are not a time critical part of a program. These commands are listed below:

Command name	Reason for exclusion from test
DIM	Only executed once
EXIT	Only executed once + terminates test program
HOME	Execution time is application dependent
ON ERROR GOSUB …	Not repeatable for testing
ON TIME GOSUB …	Not repeatable for testing
SUBMAINPROG .. ENDPROG	Only executed once
INDEX	Stops program execution
SUBPROG name .. RETURN	Only used by the compiler
WAITAX	Execution time defined by command parameters or external circumstances
WAITI	Execution time defined by command parameters or external circumstances
WAITNDX	Execution time defined by command parameters or external circumstances
WAITP	Execution time defined by command parameters or external circumstances
WAITT	Execution time defined by command parameters or external circumstances

Test results

Instruction execution time

Command	ms	Command	ms.	Command	ms.	Command	ms.	Command	ms.
ACC	1,041	ERRCLR	0,711	LINKGPAR	20,91	PID	0,540	SYNCSTAT	0,416
APOS	0,395	ERRNO	0,341	MAPOS	0,345	POSA	1,659	SYNCV	1,051
AVEL	0,855	GET	0,340	MAVEL	0,800	POSR	2,131	TIME	0,358
AVEL (100ms)	0,858	GOSUB	0,322	MAVEL (100ms)	0,808	PRINT	6,358	TRACKERR	0,432
AXEND	0,366	GOTO	0,165	MIPOS	0,345	PULSACC	0,631	VEL	1,070
COMOPTGET	0,507	IF..THEN..ENDIF (min)	0,466	MOTOR OFF	0,658	PULSVEL	0,842	_GETVEL 50	50,71
COMOPTSEND	0,541	IF..THEN..ENDIF (std.)	1,346	MOTOR ON	0,658	RST ORIGIN	0,208		
CONTINUE	0,236	IN	0,545	MOTOR STOP	0,658	SAVEPROM	3764		
CPOS	0,395	INAD 53	0,534	LOOP	0,363	SET	1,394		
CSTART	0,658	INAD 60	0,536	NOWAIT	0,179	SETVLT	20,91		
CSTOP	0,711	INB 0	0,534	OUT	0,553	SET ORIGIN	0,413		
CVEL	0,989	INB 1	0,531	OUTAN	0,326	STAT	0,519		
DEC	1,034	INKEY (-1)	0,516	OUTB 0	0,539	SYNCERR	0,424		
DEFORIGIN	0,711	IPOS	0,395	OUTB 1	0,541	SYNCM	1,496		
DELAY 10	9,658	LINKAXPAR	20,88	OUTDA	0,547	SYNCP	1,079		

Arithmetic operator execution time

	Operation	Time (ms)	OPS
Assigment	VAR = CONST	0,338	2959
	VAR = VAR	0,338	2959
Arithmetic	VAR = VAR+CONST	0,672	1488
	VAR = VAR+VAR	0,672	1488
	VAR = VAR - CONST	0,672	1488
	VAR = VAR - VAR	0,673	1486
	VAR = VAR * CONST	0,676	1479
	VAR = VAR * VAR	0,676	1479
	VAR = VAR % CONST	0,677	1477
	VAR = VAR % VAR	0,677	1477
Bit-wise ops.	VAR = VAR & CONST	0,672	1488
	VAR = VAR & VAR	0,673	1486
	VAR = VAR | CONST	0,672	1486
	VAR = VAR | VAR	0,673	1486
	-CONST	0,338	2959
	-VAR	0,497	2012
	VAR = VAR << CONST	0,675	1481
	VAR = VAR << VAR	0,675	1481
	VAR = VAR >> CONST	0,675	1481
	VAR = VAR >> VAR	0,675	1481

Benchmark test

The benchmark test is a series of commonly used instructions. The purpose of the test is to examine the general change in execution time depending on the operation condition of the tested SyncPos option.

The program was executed on three different VLT's with the following results:

	Absolute values			Nominal values			
Test condition	VLT #1	VLT #2	VLT #3	VLT #1	VLT #2	VLT #3	AVG.
MOTOR OFF	2969	2960	2966	100%	100%	100%	100%
CVEL	3447	3438	3444	86%	86%	86%	86%
POSA	6329	6329	6345	47%	47%	47%	47%
POSR	6329	6329	6346	47%	47%	47%	47%
SYNCP	6168	6064	6197	48%	49%	48%	48%
SYNCM	6324	6227	6350	47%	48%	47%	47%
SYNCP w/ VIRTUAL M.	7022	6991	7022	42%	42%	42%	42%
SYNCM w/ VIRTUAL M.	7313	7180	7951	41%	41%	37%	40%
TOTAL EXECUTION TIME	45901	45518	46621	100%	101%	98%	100%

Enabling CVEL (trajectory and PID calculations) increases command execution time by 13% compared with the MOTOR OFF situation. POSA, POSR, SYNCP, SYNCM increases command execution time by 52-53%. Enabling the VIRTUAL MASTER increases command execution time by 5-6%.

The test shows that there is a small difference in the command execution times on different SyncPos option cards.

ON PERIOD benchmark test

The ON PERIOD benchmark program is designed to measure the actual latency when keeping track of 5 different ON PERIOD subprograms. The program only investigates the extent of the "administration" time not the actual subroutine calling time. The calling time has previously been established to be 322µs (the GOSUB command).

	Absolute values		Nominal values	
Test condition	No ON PERIOD's	5 ON PERIOD's	No ON PERIOD's	5 ON PERIOD's
MOTOR OFF	2969	2969	100%	100%
CVEL	3447	3450	86%	86%
POSA	6329	6329	47%	47%
POSR	6329	6329	47%	47%
SYNCP	6168	6177	48%	48%
SYNCM	6324	6333	47%	47%
SYNCP w/ VIRTUAL M.	7022	7026	42%	42%
SYNCM w/ VIRTUAL M.	7313	7326	41%	41%

The test shows that the declaration and administration itself of ON PERIOD subroutines does not take up measurable CPU resources.

■ Conclusion

SAVEPROM is the slowest command with an execution time of 3.8 seconds. CONTINUE is the fastest command with an execution time of 236µs.

The SET command (1.4ms) is a lot faster than the SETVLT command (21 ms), approximately a factor 15 in difference.

Enabling CVEL (trajectory and PID calculations) increases command execution time by 13% compared with the MOTOR OFF situation. SYNCP, SYNCM increases command execution time by 52% and 53% respectively. Enabling the VIRTUAL MASTER increases command execution time by 5-6%.

A variable assignment takes 338µs. Arithmetic and bit-wise operations generally take 673µs - 338µs = 335µs. It does not affect the execution time whether the operands are constants or variables.

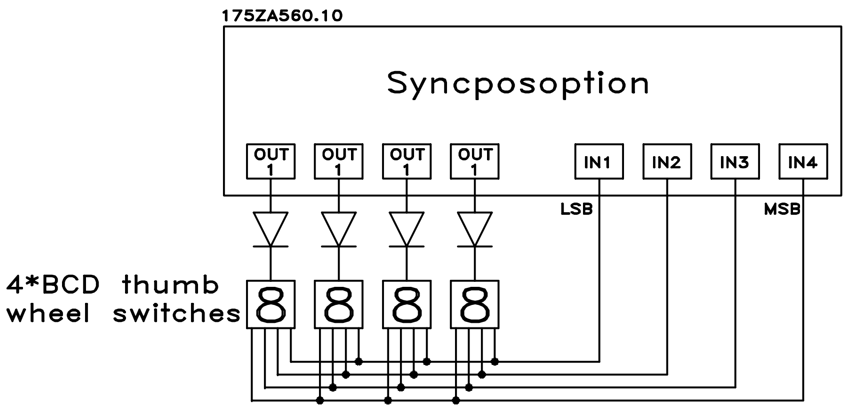
The test shows that the declaration and administration itself of ON PERIOD subroutines does not take up measurable CPU resources.

Introduction

This note gives examples of the use of SyncPos in two different applications with BCD thumb wheel switches and overflow handling.

SyncPos and BCD Thumb Wheel Switches

In a given application four thumb wheel switches are used for setting a value. This could be a length or time setting. In some applications a PLC is used to handle the large number of inputs and outputs required to handle thumb wheel switches. SyncPos eliminates the use of a PLC in these situations. Basically the signals are multiplexed and therefore less inputs are needed.
Below is a drawing of the interfacing:



Below is a syntax example that will handle the above BCD thumb wheel switches.

```
/*Insert this where the cutting length is to be defined*/
  OUT 1 1                                    /*Enable reading of 1000s*/
  WAITT 10
  dist= ((INB 0 & 15)*1000)                  /*Reading of 1000s (last 4 bit only)*/
  OUT 1 0
  WAITT 10
  OUT 2 1                                    /*Enable reading of 100s*/
  WAITT 10
  dist=dist+((INB 0 & 15)*100)               /*Reading of 100s (last 4 bit only)*/
  OUT 2 0
  WAITT 10
  OUT 3 1                                    /*Enable reading of 10s*/
  WAITT 10
  dist=dist+((INB 0 & 15)*10)                /*Reading of 10s (last 4 bit only)*/
  OUT 3 0
  WAITT 10
  OUT 4 1                                    /*Enable reading of 1s*/

  WAITT 10
  dist=dist+(INB 0 & 15)                     /*Reading of 1s (last 4 bit only)*/
  OUT 4 0
  WAITT 1000
```

SyncPos and Overflow Handling

In a cut to length application it may be useful to keep track of the length travelled. This could, for example, be used for warning the operator that the rollers need cleaning. The track record keeper must be able to handle overflow as the machine may run long distances (more than 2147483647 quad counts). Below is a cut to length application example utilizing thumb wheel switches and overflow handling.

```
/*   Cut to length demo program using BCD counters and overflow handling    */

  ON ERROR GOSUB errhandle
  OUTB 0 0
  dist=0                                                           /* Setting up variables */
  calc=0                                                                 /* for use in this */
  oldapos=APOS                                                              /* program */
--------------------------------------------------------------------------------------------
  LINKAXPAR ENCODER 710 "Encoder resol." 0 10000 1     /* For ease of commisioning several internal */
  LINKGPAR 130 711 "Length" 0 1000000000 0            /*parameters have been linked to the LCP, as */
  LINKGPAR 131 712 "Cutting Delay" 0 10000 1          /*well as timers, delays and scaling factors. */
  LINKGPAR 132 713 "Cutting Time" 0 10000 1
  LINKGPAR 133 714 "Length FactorN" 0 1000000000 1
  LINKGPAR 134 715 "Length FactorD" 0 1000000000 1
  LINKGPAR 135 716 "ERROR" -10000 10000 0
  LINKGPAR 136 717 „Break Interval" -2147483648 2147483647 1
--------------------------------------------------------------------------------------------
  VEL 20
  ACC 100
  DEC 100
  DEFORIGIN
--------------------------------------------------------------------------------------------
  MAIN:
  IF IN 5 & calc THEN                   /* Wait for an enable signal and BCD counter to finish counting*/
    travel=(dist*(GET 133 % GET 134))    /* Calculate travel distance and take user factors into account*/
    POSR travel                                        /* The drive will travel the specified distance*/
    WAITT GET 131                                                        /* Precutting delay */
    OUT 5 1                                                                 /* Enable knife */
    WAITT GET 132                                                      /* Cutting time signal */
    OUT 5 0
    SET 135 TRACKERR                                                    /* Set cutting error */
  ENDIF
  IF IN 6 THEN                                  /* If input 6 is high calculate new travel distance */
    GOSUB distance
    travel=(dist*(GET 133 % GET 134))
    SET 130 travel                                      /* Display set distance in parameter 711 */
  ENDIF
  IF IN 7 THEN                                                  /* If input 7 is high save settings */
    SAVEPROM
  ENDIF
  IF(APOS-oldapos<0) THEN                                                /* Handling of overflow */
    oldapos=(oldapos-(2*0xFFFFFFF))
  ENDIF
```

```
IF (APOS-oldapos>=GET 136) THEN        /* Set output when drive has travelled the distance specified
                                       in parameter 717 */
   OUT 8 1
   WAITI 8 1
   OUT 8 0
   oldapos=APOS
ENDIF
GOTO MAIN
```

SUBPROGRAMS:

```
SUBMAINPROG
SUBPROG distance
  calc=0
  OUT 1 1                                              /*Enable reading of 1000s */
  WAITT 10
  dist= ((INB 0 & 15)*1000)                   /*Reading of 1000s (last 4 bit only) */
  OUT 1 0
  WAITT 10
  OUT 2 1                                               /*Enable reading of 100s */
  WAITT 10
  dist=dist+((INB 0 & 15)*100)                  /*Reading of 100s (last 4 bit only)*/
  OUT 2 0
  WAITT 10
  OUT 3 1                                                 /*Enable reading of 10s*/
  WAITT 10
  dist=dist+((INB 0 & 15)*10)                    /*Reading of 10s (last 4 bit only)*/
  OUT 3 0
  WAITT 10
  OUT 4 1                                                  /*Enable reading of 1s*/

  WAITT 10
  dist=dist+(INB 0 & 15)                          /*Reading of 1s (last 4 bit only)*/
  OUT 4 0
  WAITT 10
  calc=1
RETURN

SUBPROG errhandle
  errclr
  calc=0
RETURN

ENDPROG
```

END OF PROGRAM

■ Introduction

A common way of sending instructions to the SyncPos option and to receive status information from the SyncPos option is by means of the digital inputs and outputs. This approach works fine in applications where the necessary communication can be transmitted as a limited number of signals (START, STOP, ALARM etc.)

However in many applications there is a need to transfer process data such as the real-time position of a lift system or the real-time position deviance of a synchronisation application.

To relay this kind of information a Fieldbus option can be used. Using a Fieldbus option can also help to free up the digital inputs and outputs for other tasks.

This note will provide the reader with a series of examples on how to use this flexible interface between a SyncPos option and an external control device (typically a PLC).

■ Working with PCD's

Communication with the SyncPos option is done by exchanging a number of PCD's or "process data words" (a two-byte integer value).
The number of PCD's range from 1 to 8 depending on the Fieldbus type:

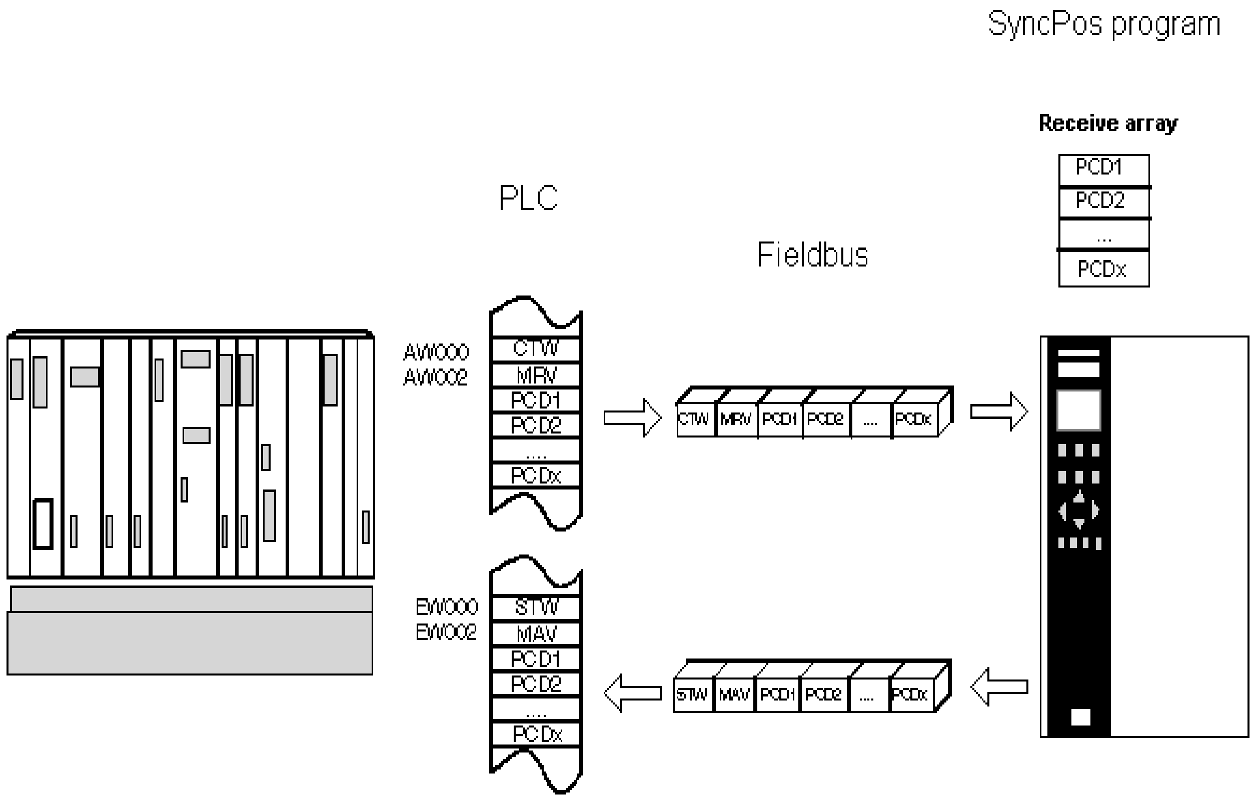
Fieldbus	Number of PCD words
PROFIBUS	PPO type 2 & 4 : 4 PCD's PPO type 5 : 8 PCD's
DeviceNet	Instance 101/151 : 2 PCD's Instance 102/152 : 4 PCD's
Interbus	1 - 7 PCD's

When working with SyncPos the PCD's are not assigned a parameter number by parameter 915 and 916 as is typically done when using a fieldbus option. The SyncPos uses PCD's as free data which can be structured as desired by the programmer.

■ SyncPos communication commands

The SyncPos option program language includes commands to send and receive PCD's. The command is used to read the contents of the Fieldbus, receive register and transfer the data to a SyncPos array and transfer the data to the Fieldbus send register.
The command COMOPTSEND is used to read from a SyncPos array and transfer the data to the Fieldbus send register.

Be aware that when using an Instance/PPO type where the CTW and MRV are included, these words will be ignored. The SyncPos option must maintain control of both the CTW and the MRV in order to regulate position or synchronization. The VLT will still be able to send status word (STW) and main actual value (MAV).



Example 1: Sending and receiving PCD's

```
/ * Program for sending and receiving 8 bytes of data via a PROFIBUS option using PPO type 2 * /

/ * Definition of arrays * /

DIM send [4], receive [4]

/ * Definition of application parameters * /

LINKGPAR 133 710 "SEND DATA WORD 1" 0 65535 0
LINKGPAR 134 711 "SEND DATA WORD 2" 0 65535 0
LINKGPAR 135 712 "SEND DATA WORD 3" 0 65535 0
LINKGPAR 136 713 "SEND DATA WORD 4" 0 65535 0                / * Initialize arrays (all elements = 0) * /
i = 1

WHILE (i<=4) DO
    receive [ i ] = 0
    i = i + 1
ENDWHILE
j = 1

WHILE (j<=4) DO
    send [ j ] = 0
    j = j +1
ENDWHILE

/ * Main program loop * /

main:
    send [ 1 ] = GET 133                             / * send array, element 1 = value of par. 710 */
    send [ 2 ] = GET 134                             / * send array, element 2 = value of par. 711 */
    send [ 3 ] = GET 135                             / * send array, element 3 = value of par. 712 */
    send [ 4 ] = GET 136                             / * send array, element 4 = value of par. 713 */
    COMOPTGET 4 receive                    / * Copy 4 words from comm. Option to receive array * /
    COMOPTSEND 4 send                 / * Copy 4 words from send array to communication Option * /
    / * Print data of receive array * /
    print  "RECEIVED (4 words)", " ", receive [ 1 ], " ", receive [ 2 ], " ", receive [ 3 ], " ", receive [ 4 ]
    DELAY 2000
GOTO main

/ * End of program * /
```

■ Types of information exchange

Basically there are two ways of relaying information through the PCD's, byte-wise or bit-wise. Examples of both are given below.

■ Byte-wise communication

Example 2: Relaying commands to the program

Byte-wise communication is well suited for relaying commands that can be performed one at a time.

"COMMANDWORD" PCD #1	DECIMAL VALUE
STOP	0
JOG+	1
JOG-	2
GO TO POSITION 15000	3

```
DIM Receive [ 4 ]
NOWAIT ON                          / * Let the program cycle through the list of instructions * /

Main:
COMOPTGET 4 Receive                / * copy 4 words from comm. option to receive array * /
IF Receive [ 1 ]  == 0 THEN        / * Execute a full stop as ordered by the PLC * /
    MOTOR  STOP
ENDIF
IF Receive [ 1 ] == 1 THEN         / * Execute a positive motion as ordered by the PLC   * /
    CVEL 100
    CSTART
ENDIF
IF Receive [ 1 ] == 2 THEN         / * Execute a negative motion as ordered by the PLC * /
    CVEL -100
    CSTART
ENDIF
IF Receive [ 1 ] == 3 THEN         / * Go to position 15000 as ordered by the PLC  * /
    POSA 15000
ENDIF
GOTO Main                          / * End of program * /
```

Example 3: Sending data to the SyncPos option

The program in example 2 is expanded to let the user go to other positions than position 15000 and with other ramps and velocities. This example will illustrate how to receive and use data words.

"TRAJECTORY" PCD's	DECIMAL VALUE
POSITION (PCD #2)	15000
VELOCITY (PCD #3)	50
RAMP (PCD #4)	90

```
DIM Receive [ 4 ]
NOWAIT ON                      / * Let the program cycle through the list of instructions * /
Main:
COMOPTGET 4 receive            / * Copy four words from comm. option to receive array * /
ACC Receive [ 4 ]              / * Set the acceleration as specified by the PLC * /
DEC Receive                    / * Set the deceleration as specified by the PLC * /
IF Receive [ 1 ] == 0 THEN     / * Execute a full stop as ordered by the PLC * /
    MOTOR  STOP
ENDIF
IF Receive [ 1 ] == 1 THEN     / * Execute positive motion as ordered by the PLC * /
    CVEL Receive [ 3 ]         / * Set the velocity as specified by the PLC * /
    CSTART
ENDIF
IF Receive [ 1 ] == 2 THEN     / * Execute a negative motion as ordered by the PLC * /
    CVEL -Receive [ 3 ]        / * Set the velocity as specified by the PLC * /
    CSTART
ENDIF
IF Receive [ 1 ] == 3 THEN
    VEL Receive [ 3 ]          / * Set the velocity as specified by the PLC * /
    POSA Receive [ 2 ]         / * Go to the position specified by the PLC * /
ENDIF
GOTO Main                      / * End of program * /
```

■ Bit-wise communication

Bit-wise communication is used if you want to communicate several things through the same PCD at the same time. This is often the case with status information, an example of this is given below.

Example 4: Relaying status information

This example is a further development of example 3. Apart from *receiving* commands and data from the communication option, this time the program will *send* status information to the communication option. The table below shows a list of what status information will be sent and on which bits.

Status word (PCD #1)	**Status bits**											
Status flag	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**	**15**	**16**
Motor magnetized	1	X	X	X	X	X	X	X	X	X	X	X
Motor moving	X	1	X	X	X	X	X	X	X	X	X	X
Position reached	X	X	1	X	X	X	X	X	X	X	X	X

```
DIM Receive [ 4 ], Send [ 4 ]
LastCPOS = CPOS                                                    / * Initialize variable * /
NOWAIT ON                             / * Let the program cycle through the list of instructions * /
Main:
COMOPTGET 4 Receive                    / * Copy 4 words from comm. option to receive array * /
ACC Receive [ 4 ]                          / * Set the acceleration as specified by the PLC * /
IF Receive [ 1 ] == 0 THEN                   / * Execute a full stop as ordered by the PLC * /
    MOTOR STOP
ENDIF
IF Receive [ 1 ] == 1 THEN            / * Execute a positive motion as ordered by the PLC * /
    CVEL Receive [ 3 ]                        / * Set the velocity as specified by the PLC * /
    CSTART
ENDIF
IF Receive [ 1 ] == 2 THEN            / * Execute a negative motion as ordered by the PLC * /
    CVEL -Receive [ 3 ]                       / * Set the velocity as specified by the PLC * /
    CSTART
ENDIF
IF Receive [ 1 ] == 3 THEN
    VEL Receive [ 3 ]                        / * Set the velocity as  specified by the PLC * /
    POSA Receive [ 2 ]                        / * Go to the position specified by the PLC * /
ENDIF
Temp = GETVLT 534                                                  / * Read VLT status word * /
fMotorMagnetized = (Temp & 2048) AND (Temp & 512)                / * Calculate status flags * /
fMotorMoving = (LastCPOS ! = CPOS)
LastCPOS = CPOS
fPositionReached = (AXEND & 1)                                   / * Calculate status word * /
SEND [ 1 ] = fMotorMagnetized +  fMotorMoving*2 + fPositionReached*4
COMOPTSEND 4 Send                                                   / * Send status  word * /
GOTO Main                                                            / * End of program * /
```

Example 5: Creating a "WATCHDOG"

This example is a further development of example 4. To let the external controller (typically a PLC) know that the SyncPos is active, that program execution is not "frozen" at any point and that communication lines are still open between the fieldbus and PLC, it is customary to add a WATCHDOG bit in the status word. The WATCHDOG bit is essentially just a bit that continuously toggles on and off.

Note that with this kind of information, it is important that the program execution passes through the "WATCHDOG toggle" section on a regular basis. The easiest way of doing that is usually to use the NOWAIT ON instruction in the beginning of the program.

Status word (PDC #1)	**Status bits**											
Status flag	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**	**15**	**16**
Motor magnetized	1	X	X	X	X	X	X	X	X	X	X	X
Motor moving	X	1	X	X	X	X	X	X	X	X	X	X
Position reached	X	X	1	X	X	X	X	X	X	X	X	X
WATCHDOG	X	X	X	1	X	X	X	X	X	X	X	X

```
DIM Receive [ 4 ], Send [ 4 ]
LastCPOS = CPOS                                              / * Initialize variable * /
NOWAIT ON                       / * Let the program cycle through the  list of instructions * /

Main:
COMOPTGET 4 receive                  / * Copy 4 words from comm. option to receive array * /
ACC Receive [ 4 ]                       / * Set the acceleration as specified by the PLC * /
IF Receive [ 1 ] ==  0 THEN               / * Execute a full stop as ordered by the PLC * /
   MOTOR STOP
ENDIF
IF Receive [ 1 ] == 1 THEN           / * Execute a positive motion as ordered by the PLC * /
   CVEL Receive [ 3 ]                       / * Set the velocity as specified by the PLC * /
   CSTART
ENDIF
IF Receive [ 1 ] == 2 THEN           / * Execute a negative motion as ordered by the PLC * /
   CVEL -Receive [ 3 ]                      / * Set the velocity as specified by the PLC * /
   CSTART
ENDIF
IF Receive [ 1 ] == 3 THEN
   VEL Receive [ 3 ]                        / * Set the velocity as specified by the PLC * /
   POSA Receive [ 2 ]                      / * Go to the position specified by the PLC * /
ENDIF

/ * Read VLT status word * /

Temp = GETVLT 534                                             / * Read VLT status word * /
fMotorMagnetized = (Temp & 2048) AND (Temp & 512)              / * Calculate status flags * /
fWatchDogBit = NOT fWatchDogBit
fMotorMoving = (LastCPOS ! =CPOS)
LastCPOS = CPOS
fPositionReached = (AXEND & 1)                                 / * Calculate status word * /

Send [ 1 ]=fMotorMagnetized + fMotorMoving*2 + fPositionReached*4 + fWathcDogBit *8
COMOPTSEND 4 Send                                                 / * Send status word * /
GOTO Main
/ * End of program * /
```

■ Conclusion

It should be apparent from the examples provided that using a fieldbus along with the SyncPos option can be a very powerful and flexible means of transferring information between the drive and the system controller. The benefits gained from using a fieldbus and the functional capabilities of the SyncPos option provide a very attractive solution for many drive applications.