epics/ether_ip / file revision Mercurial

ether_ipApp/doc/readme.txt

author	saa@beldar.slac.stanford.edu
	Tue May 10 10:26:01 2011 -0700 (2 months ago)
changeset 201	bad3916fbc01
parent 113	d0295534b5db
permissions	-rw-r--r--

Correct EIP_INVALID_SOCKET typo made in last commit. Remove INADDR_NONE for solaris.

```
 1 -*- outline -*- $Id$
 2
 3 * This document
 4 is part of the EPICS EtherIP package, to be found in
 5 <ether_ip>/ether_ipApp/doc/readme.txt.
 6 This is the "Manual" on how to use the driver.
 7
 8 For details on the underlying protocol, see
 9 "Interfacing the ControlLogix PLC Over EtherNet/IP",
10  K.U. Kasemir, L.R. Dalesio
11  ICALEPCS PSN THAP020
12  LANL E-Print Archive: http://arXiv.org/abs/cs.NI/0110065
13
14 * EtherNet/IP
15 EtherNet/IP, originally called "ControlNet over Ethernet"
16 as defined in the ControlNet Spec, Errata 2, is the protocol
17 used by Allen-Bradley ControlLogix PLCs.
18
19 This software is both a command-line test tool for Win32/Unix
20 and a driver/device for EPICS IOCs.
21
22 * Compilation
23 See the top-level README
24
25 * Command-line tool
26 The ether_ip_test executable (ether_ip_test.exe on Win32)
27 allows for simple communication checks.
```

```
28 The available command line options might change,
29 use the "-?" option for help:
30     >ether_ip_test -?
31     Usage: ether_ip_test <flags> [tag]
32     Options:
33       -v verbosity
34       -i ip  (as 123.456.789.001 or DNS name)
35       -p port
36       -s PLC slot in ControlLogix crate (default: 0)
37       -s slot (default: 0)
38       -t timeout (ms)
39       -a array size
40       -w <double value to write>
41
42 Example:
43 Read tag "REAL" from plc with IP 128.165.160.146,
44 PLC happens to be in slot 6 of the ControlLogix crate:
45     >ether_ip_test -i 128.165.160.146 -s 6 REAL
46     Tag REAL
47     REAL 0.002502
48
49 Add the "-v 10" option to see a dump of all the exchanged
50 EtherIP messages. The included error messages might help
51 to detect why some tag cannot be read.
52
53 * EPICS startup files
54 1) Load Driver & Device Support
55 The ether_ipApp creates a library "ether_ipLib"
56 which contains only the driver code.
57
58 You can load the ether_ipLib itself or use e.g.
59 the makeBaseApp.pl ADE to include this library
60 in your application library.
61
62 2) IP Setup
63 Since the driver uses TCP/IP, the route to the PLC has to defined.
64 Therefore you have to add something like this to your IOC startup file:
65     # Define the DNS name for the PLC, so we can it instead of the
66     # raw IP address
67     hostAdd "snsplc1", "128.165.160.146"
68
69     # *IF* "128.165.160.146" is in a different subnet
70     # that the IOC cannot get to directly, define
71     # a route table entry. In this example, ..254 is the gateway
72     routeAdd "128.165.160.146", "128.165.160.254"
73
74     # Test: See if the IOC can get to "snsplc1":
75     ping "snsplc1", 5
76
77 3) Driver Configuration
78 Before calling iocInit in your IOC startup file, the driver has to
79 be configured. After loading the driver object code either directly
80 or as part of an ADE library, issue the following commands.
81 Note that the IP address (128.165.160.146), the DNS name (snsplc1)
82 and the name that the driver uses (plc1) are all related but different!
```

```
 83
 84     # Initialize EtherIP driver, define PLCs
 85     # -----------------------------------
 86     drvEtherIP_init
 87
 88     # Provide a default for the driver's scan rate in case neither
 89     # the record's SCAN field nor the INP/OUT link
 90     # contain a scan rate that the driver can use.
 91     # Recommendation: Do not use this feature, provide a scan flag
 92     # "S" in the INP/OUT link instead. See manual comments on the "S" flag.
 93     drvEtherIP_default_rate = 0.5
 94
 95     # drvEtherIP_define_PLC <name>, <ip_addr>, <slot>
 96     # The driver/device uses the <name> to indentify the PLC.
 97     #
 98     # <ip_addr> can be an IP address in dot-notation
 99     # or a name that the IOC knows about (defined via hostAdd).
100     # The IP address gets us to the ENET interface.
101     # To get to the PLC itself, we need the slot that
102     # it resides in. The first, left-most slot in the
103     # ControlLogix crate is slot 0.
104     # (When omitting the slot number, the default is also 0)
105     drvEtherIP_define_PLC "plc1", "snsplc1", 0
106
107     # EtherIP driver verbosity, 0=silent, up to 10:
108     EIP_verbosity=4
109
110 NOTE for EPICS R3.14 Soft-IOCs:
111 Replace variable assignments like "EIP_verbosity=4"
112 by function calls like "EIP_verbosity(4)".
113
114 4) Tell EPICS Database about Driver/Device
115 To inform EPICS of this new driver/device, a DBD file is used.
116 ether_ip.dbd looks like this:
117      driver(drvEtherIP)
118      device(ai,         INST_IO, devAiEtherIP,         "EtherIP")
119      device(bi,         INST_IO, devBiEtherIP,         "EtherIP")
120      device(mbbi,       INST_IO, devMbbiEtherIP,       "EtherIP")
121      device(mbbiDirect, INST_IO, devMbbiDirectEtherIP, "EtherIP")
122      device(stringin,   INST_IO, devSiEtherIP,         "EtherIP")
123      device(ao,         INST_IO, devAoEtherIP,         "EtherIP")
124      device(bo,         INST_IO, devBoEtherIP,         "EtherIP")
125      device(mbbo,       INST_IO, devMbboEtherIP,       "EtherIP")
126      device(mbboDirect, INST_IO, devMbboDirectEtherIP, "EtherIP")
127
128 You can load this directly via dbLoadDatabase in the startup script.
129 Usually, however, you would use something like the makeBaseApp.pl ADE,
130 have this:
131        include "base.dbd"
132        include "ether_ip.dbd"
133 in your application DBD file and then in the IOC startup script,
134 "dbLoadDatabase" loads the single application DBD file which
135 includes the EtherIP DBD file.
136
137 * EPICS records: Guidelines
```

```
138 The EtherIP driver was designed in order to optimize the tag
139 transfers. When multiple records are attached to the same tag, the
140 driver will transfer the tag only once, using the highest scan rate of
141 the attached records. When records refer to different elements of an
142 array, the driver will transfer the array as a whole. Tags are
143 arranged according to PLC and scan rate. In order to not disturb
144 processing of the EPICS database, the driver has one separate task per
145 PLC.
146
147 You should try to benefit from the driver optimization by arranging
148 tags in arrays. You can have alias tags on the PLC, so that a
149 meaningful alias like "InputFlow" is used in the ladder logic while
150 the data is also held in an array element "xfer[5]" which the EPICS
151 record can use for the network transfer.
152 Arrays should be one-directional: Use separate "EPICS to PLC" and "PLC
153 to EPICS" arrays. Because of PLC buffer limitations, the array size is
154 unfortunately limited to about BOOL[350] and REAL[40]. While you can
155 define bigger arrays, those cannot be transferred over the network
156 with EtherIP. Consequently you might end up with several transfer arrays.
157
158 You should also understand that the network transfer can be delayed or
159 even fail because of network problems. Consequently you must not
160 depend on "output" records to write to the PLC within milliseconds. If
161 e.g. an output on the PLC has to be "on" for a certain amount of time,
162 have the PLC ladder logic implement this critical timing. The EPICS
163 record can then write to a "start" tag on the PLC, the PLC handles the
164 exact timing in response to the command. When done, the PLC signals
165 success or failure via another "status" tag.
166 This way, network delays in the transfer of "start" and "status" tags
167 will not harm the critical timing.
168
169 * EPICS records: Generic fields
170 ** DTYP: Device type
171 Has to be "EtherIP" as defined in DBD file:
172     field(DTYP, "EtherIP")
173
174 ** SCAN: Scan Field
175 The driver has to know how often it should communicate
176 with the PLC. Per default, it uses the SCAN field
177 of the record:
178     field(SCAN, "1 second")
179     field(SCAN, ".1 second")
180     field(SCAN, "10 second")
181 ...
182 The driver scans the PLC at the same rate.
183 The record simply reads the most recent value.
184 Note: The scan tasks of the driver and the EPICS database
185 are not synchronized.
186
187 *** Note on multiple records attached to the same tag
188 When multiple records refer to the same tag, the driver
189 will scan that tag at the highest scan rate.
190 Example:
191 record(ai, "A")
192 {
```

```
193     field(INP, "@plc1 fred")
194     field(SCAN, "1 second")
195     ...
196 }
197
198 record(ai, "B")
199 {
200     field(INP, "@plc1 fred")
201     field(SCAN, ".1 second")
202     ...
203 }
204
205 -> The driver will scan the tag "fred" at 10 Hz.
206
207 This also applies to arrays:
208 Since requests to array elements my_array[0], my_array[2],
209 my_array[5],... are combined into a SINGLE transfer of the
210 tag my_array, the rate of that transfer is the fastes rate
211 requested for any of the array elements.
212 (Unless you request single element requests with the 'E'
213 flag which you should try to avoid).
214
215 *** SCAN Passive
216 Output records are often passive:
217 They are only processed when the record is accessed via ChannelAccess
218 from an operator screen where someone entered a new value for this
219 record.
220
221 While the driver will only write to a tag when the record is
222 processed, it will still try to read the tag from the PLC in case it is
223 changed from another source (another IOC, PanelView, ...).
224 The section "Keeping things synchronized" gives details on this.
225 Since the driver cannot extract an update rate from the SCAN field
226 when it is set to "Passive", the "S" scan flag has to be used as
227 described in the INP/OUT link section.
228
229 *** SCAN I/O Intr
230 Input records can be configured to use
231     field(SCAN, "I/O Intr")
232 The driver causes the record to be processed as soon as a new value is
233 received. As in the Passive case, the driver needs the "S" scan
234 flag to determine the poll rate.
235
236 ** INP, OUT: Input/Output Link
237 The INP field for input records resp. the OUT field for output records
238 has to match
239    field(INP, "@<plc name> <tag> [flags]")
240    field(OUT, "@<plc name> <tag> [flags]")
241
242 *** <plc name>
243     This is the driver's name for the PLC, defined in the IOC
244     startup script via
245        drvEtherIP_define_PLC <name>, <ip_addr>, <slot>
246     Example:
247        drvEtherIP_define_PLC "plc1", "snsplc1", 0
```

```
248     More detail on this as well as the IP address mapping
249     and routing can be found in the "Installation" section.
250
251 *** <tag>
252     This can be a single tag "fred" that is defined in the "Controller
253     Tags" section of the PLC ladder logic. It can also be an array tag
254     "my_array[5]" as well as a structure element "Local:2:I.Ch0Data".
255     Array elements are indexed beginning with 0.
256     Note: you can use decimals (2, 10, 15), hex numbers (0x0f) and
257     octal numbers (04, 07, 12). This means that 08 is invalid because
258     it is interpreted as an octal number!
259
260     The <tag> has to be a single elementary item (scalar tag, array
261     element, structure element), not a whole array or structure.
262
263 *** <flags>
264     There are record-specific flags that will be explained
265     later. Common flags are:
266
267     "S <scan period>" - Scan flag
268     If the SCAN field does not specify a scan rate as in the case of
269     "Passive" output records or input records with SCAN="I/O Intr",
270     the S flag has to be used to inform the driver of the requested update
271     rate.
272     Note that the behavior of the scan flag is only defined for these
273     cases:
274      Record Type     SCAN
275      AI              I/O Intr
276      BI               I/O Intr
277      MBBI             I/O Intr
278      MBBIDirect       I/O Intr
279      AO               Passive
280      BO               Passive
281      MBBO             Passive
282      MBBODirect       Passive
283
284     In all other cases, the S flag should not be used, instead the
285     SCAN field must provide the needed period (e.g. SCAN=".5 second").
286
287     The time format is in seconds, like the SCAN field, but without "seconds".
288     Examples:
289        field(INP, "@snsioc1 temp S .1")
290        field(INP, "@myplc xyz S 0.5")
291     There has to be a space after the "S"!
292
293     If the record has neither a periodic SCAN rate nor an S flag in
294     the link field, you will get an error message similar to
295
296        devEtherIP (Test_HPRF:Amp_Out:Pwr1_H):
297        cannot decode SCAN field, no scan flag given
298        Device support will use the default of 1 secs,
299        please complete the record config
300
301     In the IOC startup file, you can set the double-typed variable
302     "drvEtherIP_default_rate" to provide a default rate.
```

```
303     If you do that, the warning will vanish.
304     The recommended practice, however, is to provide a per-record
305     "S" flag because then you can recollect the full configuration
306     from the record and avoid ambiguities.
307
308     "E" - force elementary transfer
309     If the tag refers to an array element,
310        field(INP, "@snsioc1 arraytag[5]")
311     the driver will combine all array requests into a single array
312     transfer for this tag. This is meant to reduce network traffic:
313     Records scanning arraytag[0], ... arraytag[5] will result in a single
314     "arraytag" transfer for elements 0 to 5.
315
316     The "E" flag overrides this:
317        field(INP, "@snsioc1 arraytag[5] E")
318     will result into an individual transfer of "arraytag" element 5,
319     not combined with other array elements.
320
321     Reasons for doing this:
322     a) The software can only transfer array elements 0 to N, always
323        beginning at 0. If you need array element 100 and only this element,
324        so there is no point reading the array from 0 to 100.
325     b) You want array elements 401, 402, ... 410. It's not possible
326        for the driver to read 401-410 only, it has to read 0-410. This,
327        however, might be impossible because the send/receive buffers of the
328        PLC can only hold about 512 bytes. So in this case you have to read
329        elements 401-410 one by one with the "E" flag.
330     c) Binary record types (bi, bo, mbbi, ...) with a non-BOOL array
331        element. See the binary record details below.
332
333     Unless you absolutely have to use the "E" flag for these reasons,
334     don't use it.
335     It is no problem to have one "BOOL[352]" tag for IOC->PLC
336     communication and another "BOOL[352]" array for PLC->IOC
337     communication, both at 10Hz. The result is a low and constant
338     network load, the transfers are almost predictable even though
339     Ethernet is not "deterministic". If instead you use several "E"
340     flags, each of those tags ends up being a separate transfer,
341     leading to more network load and possible collisions and delays.
342
343
344 * ai, Analog Input Record
345 By default the tag itself is read:
346
347 PLC Tag type      Action
348 ------------      ------------------------------------------------
349 REAL              VAL field is set (no conversion).
350 INT, DINT, BOOL   RVAL is set, conversions (linear, ...) can be used.
351
352 The analog record cannot be used with BOOL array elements,
353 other arrays (REAL, INT, ...) are allowed.
354
355 ** Statistics Flags
356 The driver holds statistics per Tag which can be accessed with ai
357 records via the flag field. A valid tag is *always* required. For
```

```
358 e.g. "TAG_TRANSFER_TIME" this makes sense because you query per-tag
359 information. In other cases it's used to find the scanlist.
360
361     field(INP, "@$(PLC) $(TAG) PLC_ERRORS")
362     - # of timeouts/errors in communication with PLC [count]
363
364     field(INP, "@$(PLC) $(TAG) PLC_TASK_SLOW")
365     - # times when scan task was slow [count]
366
367     field(INP, "@$(PLC) $(TAG) LIST_TICKS")
368     field(INP, "@$(PLC) $(TAG) LIST_TIME")
369     - vx Ticktime (3.13) or secs since 1990 (3.14) when tag's list was checked.
370       Useful to monitor that the driver is still running.
371
372     field(INP, "@$(PLC) $(TAG) LIST_SCAN_TIME"),
373     field(INP, "@$(PLC) $(TAG) LIST_MIN_SCAN_TIME"),
374     field(INP, "@$(PLC) $(TAG) LIST_MAX_SCAN_TIME"),
375     - Time for handling scanlist [secs]: last, minumum, maximum
376
377     field(INP, "@$(PLC) $(TAG) TAG_TRANSFER_TIME")
378     - Time for last round-trip data request for this tag
379
380 The PLC_TASK_SLOW flag is of less use than anticipated. It's
381 incremented when the scan task is done processing the list and then
382 notices that it's already time to process the list again. Since all
383 delays are specified in vxWorks ticks (3.13) or seconds (3.14), defaulting
384 to 60 ticks per second (3.13) or 1 second (3.14), this scheduling is rather
385 coarse. With all the other task scheduling going on and ethernet delays,
386 PLC_TASK_SLOW might increment every once in a while without a
387 noticeable impact on the data (no time-outs, no old data).
388
389 * ao, Analog Output Record
390 Like analog input, tags of type REAL, INT, DINT, BOOL are supported as
391 well as REAL, INT, DINT arrays (no BOOL arrays). No statistics flags
392 are supported.
393 For REAL tags, the VAL field of the record is written to the tag.
394 Otherwise, the RVAL field is used and you can benefit from
395 the AO record's conversions VAL <-> RVAL.
396
397 If the SCAN field is "Passive", the "S" flag has to be used.
398
399 ** Write Caveats
400
401 *** Keeping things synchronized
402 The problem is that the EPICS IOC does not "own" the PLC. Someone else
403 might write to the PLC's tag (RSLogix, PanelView, another IOC,
404 command-line program). The PLC can also be rebooted independent from
405 the IOC. Therefore the write records cannot just write once they have
406 a new value, they have to reflect the actual value on the PLC.
407
408 In order to learn about changes to the PLC from other sources, the
409 driver scans write tags just like read tags, so it always knows the
410 current value. When the record is processed, it checks if the value to
411 be written is different from what the PLC has. If so, it puts its RVAL
412 into the driver's table and marks it for update
```

```
413 -> the driver writes the new value to the PLC.
414
415 So in the case of output records the driver will still read from the PLC
416 periodically and only switch to write mode once after an output record
417 has been processed and provided a new value.
418
419 Some glue code in the device is called for every value that the driver
420 gets. It checks if this still matches the record's value. If not, the
421 record's RVAL is updated and the record is processed. A user interface
422 tool that looks at the record sees the actual value of the PLC.
423 The record will not write the value that it just received because
424 it can see that RVAL matches what the driver has.
425
426 This fails if two output records are connected to the same tag,
427 especially if one is a binary output that tries to write 0 or 1. In
428 that case the two records each try to write "their" value into the
429 tag, which is likely to make the value fluctuate.
430
431 Another side effect is that when processing an output record,
432 that record will not write immediately. The writing is handled
433 by a separate thread in the driver. The next time the tag is scanned,
434 the driver thread will notice the "update" flag and write to the PLC.
435 Consequently you adjust the write latency when you specify the scan
436 rate of the driver thread.
437
438 *** Output records and arrays
439 When using _input_records_ that reference array tags a[0], a[1],
440 a[9], the driver will read the whole referenced part of the array,
441 that is a[0...9]. While the array might have more elements, the driver
442 reads elements from zero up to the highest element referenced by a
443 record.
444
445 Likewise, when output records reference those array tags,
446 the whole section of the array from 0 to the highest element
447 referenced by a record gets written.
448 When no output record requested a 'write', it is read.
449
450 This is perfect for e.g. limit settings:
451 Most of the time, they are unchanged and the driver efficiently
452 monitors them. Should an operator change one of the limits on the IOC,
453 the whole array is written. Should the operator change a limit via
454 PanelView, the driver on the IOC notices the change and updates
455 the output record for this array entry.
456
457 There are problems when frequently processed records are combined in
458 such a bi-directional array tag.
459
460 Example: A heartbeat record, processed every second, is part of an
461 'output' array. Every second, that record marks the whole array(!) for
462 'write'.
463 If an operator now changes another array element on the IOC, that gets
464 written, too. But when the operator changes a value on the PLC via
465 PanelView, that change is very likely to be lost because the driver
466 doesn't get around to 'read' the tag since the heartbeat record causes
467 it to 'write' all the time. Consequently, most tag changes from
```

```
468 PanelView are almost immediately overwritten by the IOC's value.
469
470 Conclusion:
471 It's impossible to have truly 100% bi-directional communication.
472 If both the record and the tag on the PLC change, one may overrule
473 the other depending on timing (scanning, network).
474
475 Next Best Solution:
476 Bi-directional use of arrays for e.g. limits work well enough
477 if they are infrequently changed from either side.
478 Records that are frequently written should not be combined in such
479 arrays. If they happen to be in the same array, use the 'E' flag
480 in the OUT link of e.g. the heartbeat record. That way, the heartbeat
481 record will only write that single array element and not trigger a
482 write of the whole referenced subsection of the array.
483 One could conclude to add 'E' to every output record, but then you
484 loose all the possible array-transfer optimization.
485
486 Best Solution:
487 Change the driver to
488 - read output record tags as part of an array
489 - but write only the single element
490 That requires some fundamental change to the driver because
491 these two read/write transfers involve technically different
492 tags.
493
494 *** Dropped messages
495 The EtherIP driver uses TCP. So unless the TCP layer reports
496 errors, all messages to the PLC should eventually reach the PLC.
497 And unless the PLC reports an error in response to a write request,
498 the write request should actually change the affected PLC tag.
499 In practice, Herb Strong and Pam Gurd (ORNL) noticed that
500 some write requests do not reach the PLC.
501 I believe this was based on an error in this driver:
502 A read request was sent out. Sometimes, an output record could
503 process before the response to the read arrives.
504 So the output record tries to write a 'new' value but before
505 that new value gets written, the previous read arrives
506 and therefore overwrites the 'new' value to be written with
507 the previous value from the PLC
508 -> When we get around to write, we write the old value.
509 This has been fixed and I could not reproduce any 'dropped write
510 requests' ever since.
511
512 *** "FORCE" Flag
513 Whenever an output record is processed, it will
514 update the driver's copy of a tag and mark it for "write".
515 The next time the driver processes the scan list which
516 contains the tag, it will write the tag to the PLC.
517
518 When the record is not processed, and therefore the tag
519 is not marked for write, the driver will read the
520 tag from the PLC.
521 What happens when the value of the tag differs from
522 the value of the record?
```

```
523
524 Per default, the record is updated to reflect the value of the
525 tag. This way, both the IOC and e.g. a PanelView display can change
526 the same PLC tag. Changes from "one" source are reflected on the
527 respective "other" side.
528 With TPRO, it looks like this:
529      'Test_HPRF:Fil1:WrmRmp_Set': got 8 from driver
530      'Test_HPRF:Fil1:WrmRmp_Set': updated record's value 8
531
532 The "FORCE" flag will change this behavior:
533 When the driver notices a discrepency, it will NOT
534 change the record but simply re-process it.
535 This causes the IOC to write to the tag on the PLC
536 again and again until the tag on the PLC matches
537 the value of the record. The record tries to "force"
538 its value into the tag.
539 With TPRO, it looks like this:
540      'Test_HPRF:Xmtr1:FilOff_Cmd': got 0 from driver
541      'Test_HPRF:Xmtr1:FilOff_Cmd': will re-write record's value 1
542
543 *** Arrays
544 When writing array tags, a single ao record (or bo, mbbo, ...)
545 is connected to a single element of the array.
546 When the record has a new value, it will update that array
547 element and mark the array as "please write to PLC during the
548 next scan cycle of the driver".
549 This is desirable because it allows several output records to
550 specify new values and then the WHOLE ARRAY is written as one unit.
551
552 Writing the values that didn't change doesn't matter because
553 a) the transfer time for a single tag and an array is almost
554    the same. Transferring an array where many items didn't change
555    is not costly, transferring two separate tags that did change
556    would take longer.
557 b) the PLC doesn't care if tags are written. There is no
558    "tag was written" event in the PLC that I know of.
559    Writing the same value again does not upset the ladder logic.
560
561 Possible problem:
562 DO NOT MIX DIRECTIONS within an array.
563 Do use arrays instead of single tags to speed up the transfer,
564 but keep different "EPICS to PLC" and "PLC to EPICS" arrays.
565 If you have to have handshake tags (EPICS writes, PLC uses
566 it and then PLC resets the tag), those bidirectional tags
567 should not be in arrays. They have to be standalone, scalar tags.
568
569 * bi, Binary Input Record
570 Reads a single bit from a tag.
571
572 PLC Tag type      Action
573 ------------      ------------------------------------------------
574 BOOL              VAL field is set to the BOOL value
575 other             converted into UDINT, then bit 0 is read
576
577 BOOL Arrays can be used:
```

```
578    field(INP, "@plc1 BOOLs[52]")
579 will read the 52nd element of the BOOL array.
580
581 INT, DINT arrays are treated as bit arrays:
582    field(INP, "@plc1 DINTs[40]")
583 will *NOT* read array element #40 but bit #40 which is bit # 8 in the
584 second DINT.
585
586 If you want to read the first bit of DINT #40, the "E" flag can be
587 used to make an elementary request for "DINTs[40]". The preferred solution,
588 though, is the Bit flag.
589 The TPRO field (see the section on debugging) is often helpful in
590 analyzing what array element and what bit is used.
591
592 ** "B <bit>": Bit flag
593    field(INP, "@plc1 DINTs[1] B 8")
594 will read bit #8 in the second DINT array element.
595
596 ** write caveats
597 See the ao comments.
598
599 * mbbi, mbbiDirect Multi-bit Binary Input Records
600 These records read multiple consecutive bits, the count is given in
601 the number-of-bits field:
602    field(NOBT, "3")
603
604 The input specification follows the bi description,
605 except that the addressed bit is the first bit.
606
607 When using array elements, the same bit-addressing applies. As a
608 result, the "B <blit>" flag should be used for non-BOOL arrays.
609
610 Note: In the current implementation, the mbbiX records can read across array
611 elements of DINT arrays. This record reads element 4, bit 31 and
612 element 5, bit 1:
613         field(INP, "@$(PLC) DINTs[4] B 31")
614         field(NOBT, "2")
615 But this feature is merely a side effect, it's safer to read
616 within one INT/DINT. Or use BOOL arrays.
617
618 * bo, mbbo, mbboDirect Binary Output Records
619 The output records use the same OUT configurations as the
620 corresponding input records.
621
622 If the SCAN field is "Passive", the "S" flag has to be used.
623
624 Note that if several records read and write different elements of an
625 array tag X, that tag is read once per cycle from element 0 up to the
626 highest element index N that any record refers to. If any output record
627 modifies an entry, the driver will write the array (0..N) in the next
628 cycle since it is marked as changed.
629
630 As a result, it is advisable to keep "read" and "write" arrays
631 separate, because otherwise elements meant for "read" will be written
632 whenever one or more other elements are changed by output records.
```

```
633
634 ** write caveats
635 See the ao comments.
636
637 * stringin String Input Records
638 String input records can be connected to STRING tags
639 on the PLC:
640
641         field(DTYP, "EtherIP")
642         field(INP,  "@$(PLC) text_tag")
643         field(SCAN, "1 second")
644
645 STRING tags seem to have an allowed length of up to 82
646 characters. The stringin record is limited to 40 characters.
647 Since I decided to include the '\0', any STRING tag gets
648 truncated to 39 characters. There is no fault indication for this,
649 just a limited string.
650
651 The stringin record works only with STRING tags. Any other tag
652 type will result in errors.
653 Likewise, only stringin records must be used with STRING tags.
654 Any other record type will fail with STRING tags.
655
656 Note: The STRING tag data type was not documented!
657 To the driver, a STRING tag looks like a "CIP structure" and the
658 location of the string length and character data in there were
659 determined from tests.
660
661 * waveform Array Input Records
662 Waveform records can be connected to REAL or DINT array tags
663 on the PLC:
664         field(DTYP, "EtherIP")
665         field(SCAN, "1 second")
666         field(INP,  "@$(PLC) array_tag")
667         field(NELM, "40")
668  field(FTVL, "DOUBLE")
669 or
670  field(FTVL, "LONG")
671
672 ** Note on tags in INP
673 On the PLC, "array_tag" could be
674       fred = REAL[40]
675 or
676       fred = DINT[80]
677 When specifying the array tag in INP, do not use
678 'fred[0]' or 'fred[any other number]', use only 'fred'.
679 The NELM field defines the number of elements read from the tag.
680 The record will read fred[0] ... fred[NELM-1].
681
682 ** Note on FTVL
683 For REAL[] array tags, FTVL must be DOUBLE.
684 For DINT[] array tags, FTVL must be LONG.
685 That way, the data type sizes match and no conversion
686 is necessary.
687 For other array tags, FTVL==LONG might work
```

```
688 but is not guaranteed to work.
689
690 * Debugging
691 The driver can display information via the usual EPICS dbior call
692 on the IOC console (or a telnet connection to the IOC):
693     dbior "drvEtherIP", 10
694 A direct call to
695     drvEtherIP_report 10
696 yields the same result. Instead of 10, lower levels of verbosity are
697 allowed.
698
699 Hint: It's useful to redirect the output to the host:
700     drvEtherIP_report 10 >/tmp/eip.txt
701 Then, on the Win32 or Unix host, open that file
702 with EMACS. The outline format allows easy browsing.
703
704 drvEtherIP_help shows all user-callable driver routines:
705     -> drvEtherIP_help
706     drvEtherIP V1.1 diagnostics routines:
707       int EIP_verbosity:
708       -  set to 0..10
709       double drvEtherIP_default_rate = <seconds>
710        -  define the default scan rate
711           (if neither SCAN nor INP/OUT provide one)
712       drvEtherIP_define_PLC <name>, <ip_addr>, <slot>
713       -  define a PLC name (used by EPICS records) as IP
714          (DNS name or dot-notation) and slot (0...)
715      drvEtherIP_read_tag <ip>, <slot>, <tag>, <elm.>, <timeout>
716      -  call to test a round-trip single tag read
717         ip: IP address (numbers or name known by IOC)
718         slot: Slot of the PLC controller (not ENET). 0, 1, ...
719         timeout: milliseconds
720       drvEtherIP_report <level>
721       -  level = 0..10
722       drvEtherIP_dump
723       -  dump all tags and values; short version of drvEtherIP_report
724       drvEtherIP_reset_statistics
725       -  reset error counts and min/max scan times
726       drvEtherIP_restart
727       -  in case of communication errors, driver will restart,
728          so calling this one directly shouldn't be necessary
729          but is possible
730
731 A common problem might be that a record does not seem to read/write
732 the PLC tag that it was supposed to be connected to.
733 When setting "TPRO" for a record, EPICS will log a message whenever a
734 record is processed. The EtherIP device support shows some additional
735 info on how it interpreted the INP/OUT link. Use a display manager, a
736 command line channel access tool or
737     dbpf "record.TPRO", "1"
738 in the IOC shell to set TPRO. Set TPRO to "0" to switch this off again.
739
740 Example output for a binary input that addresses "DINTs[40]":
741 process:   snsioc4:biDINTs40
742    link_text  : 'plc1 DINTs[40]'
```

```
743    PLC_name   : 'plc1'
744    string_tag : 'DINTs'
745    element    : 1          <- element 1!
746    mask       : 0x100      <- mask selects bit 8!
747
748 As you see, the BI record is reading bit #8
749 in DINT[1], that's bit #40 when counting from the
750 beginning of the DINT array.
751 If that's what you wanted, OK.
752 If you entered "DINTs[40]" because you wanted bit #0
753 in array element 40, you should have used "DINTs[40] B 0"
754 (See the description of the bi record and the "B" flag)
755
756 ** Checklist
757 ( ) Set the record's TPRO to "1".
758     Does the record get processed when you want it to be processed?
759     Does the link_text make sense?
760     Is it parsed correctly, i.e. is the PLC_name what you
761     meant to use for a PLC name?
762     Does the combination of string_tag, element & mask make sense?
763 ( ) Call "drvEtherIP_report 10", locate the information
764     for the tag that the record uses:
765     *** Tag 'Word2' @ 0x18F0F38:
766       scanlist            : 0x191B5F0
767       compiled tag        : 'Word2'
768       elements            : 29
769       cip_r_request_size  : 12
770       cip_r_response_size : 64
771       cip_w_request_size  : 72
772       cip_w_response_size : 4
773       data_lock ID        : 0x18F0ED8
774       data_size (buffer)  : 60
775       valid_data_size     : 60
776       do_write            : no
777       is_writing          : no
778       data                : INT 4 1 1 1 4 0 0 0 0 0 1 1 1 1\
779                              4 1 1 1 1 1 2 2 1 16 0 0 4 1 1
780       transfer tick-time  : 46 (0.046 secs)
781     If the "...._size" fields in there are zero, the driver
782     could not learn anything about the tag.
783     See if the tag actually exists on the PLC (next step).
784     Note on arrays:
785     Requests are combined. Assume that we are debugging a record
786     that accesses tag FRED[7]. drvEtherIP_report might show
787     that the driver is actually trying to access 10 elements
788     for tag FRED. That means that some other record must
789     try to get FRED[9], so alltogether the driver reaches
790     for FRED[0]...FRED[9] -> 10 elements.
791     Assert that there are at least 10 elements for the tag FRED
792     on the PLC!
793 ( ) Use the ether_ip_test tool, e.g. try
794        ether_ip_test -i 123.45.67 MyTag[12]
795     to see if you can get to the PLC and read the tag.
796
797 * EIP_verbosity
```

```
798 Depending on the value of EIP_verbosity, the driver
799 and device support will report various levels of detail:
800
801  0: Only severe errors are reported
802  1: Show errors and some more information
803 ...
804  9: Show a hex-dump of all the network traffic
805     that's sent and received
806 10: Show byte-by-byte explanation of what's assembled
807     in the send buffer, hex-dump of that buffer,
808     hex-dump of the received data and byte-by-byte
809     explanation of what has been received.
810
811 A LOT of output is generated at levels 9 & 10, resulting
812 in a significant CPU load and - when viewed in a telnet
813 session - network traffic.
814
815
816 * Driver Operation Details
817 Example:
818 Records
819    "fred", 10 seconds
820    "freddy", 10 seconds
821    "jane", 10 seconds
822    "analogs.temp[2].input", 5 seconds
823    "binaries[3] E", 1 Hz
824    "binaries", element 1, 10Hz
825    "binaries", element 5, 10Hz
826    "binaries", element 10, 10Hz
827
828 Scanlist created from this
829    10  Hz: "binaries", elements 0-10
830     1  Hz: "binaries[3]"
831     0.5Hz: "analogs.temp[2].input"
832     0.1Hz: "fred", "freddy", "jane"
833
834 Driver actions
835    One thread and socket per PLC for communication.
836    One TagInfo per tag: name, elements, sizes.
837    ScanTask: runs over scanlists for its PLC.
838    For each scanlist:
839        Figure out how many requests can be combined
840        into one request/response round-trip
841        (~500 byte limit), record in TagInfo.
842
843 The driver simply adds requests from the current scanlist
844 until the buffer limit is reached. The following tags are
845 placed in another transfer. The driver does not try every possible
846 combination of tags from the current scanlist to find the optimal
847 combination to reduce the number of transfers.
848 It does not combine tags from e.g. the 10 second scanlist
849 with tags from the 1 second scanlist every 10th turn.
850
851 * PLC Buffer Limit
852 See ether_ip.h for details on the limit which is about 500 bytes.
```

```
853
854 The driver can only combine read/write requests into one multi-request
855 until either the combined request or the expected response reaches a
856 buffer limit. In practice, this means:
857
858 When reading many INT tags, each with a 4-character tag name,
859 32 read commands can be combined until hitting the request-size limit.
860 The response of 32 * 2 bytes (INT) plus some protocol overhead is much
861 smaller than the request.
862
863 When reading many REAL tags, each with a 1-character tag name, 39 read
864 commands combine into one request. Both the request and the response
865 are close to the limit.
866
867 When reading elements of a REAL array tag, 120 array elements can be read.
868 The request contains the single array tag, asking for 111 elements,
869 the response reaches the transfer buffer limit. Similarly, INT arrays
870 can use up to 240 element.
871
872 The guideline of "limit arrays to 40 elements" allow the driver a lot
873 of flexibility: It can combine three REAL[40] requests into one
874 transfer or add several single-tag requests with 2 x INT[40] requests etc.
875
876 * CIP data details
877 Analog array REALs[40], read "REALs", 2 elements
878 -> REALs[0], REALs[1]
879
880 Binary array BOOLs[352], read "BOOLs", 1 element
881 -> 32bit DINT with bits  0..31
882
883 Access to binaries is translated inside the driver.
884 Assume access to "fred[5]":
885 For analog records, a request to the 5th array element is assumed.
886 For binary records, we assume that the 5th _bit_ should be addressed.
887 Therefore the first element (bits 0-31) is read and the single
888 bit number 5 in there returned.
889
890 * Message '<channel xxx> already writing'
891 This message is a result of how the device & driver support writes
892 to the PLC.
893 Remember that even _output_ records are periodically _read_ by the
894 driver, and in case the value of the tag on the PLC differs from
895 what's in the record, the record gets updated & processed.
896 Most of the time, the tag is thus read, the result matches what's
897 in the record, and nothing else happens.
898 When on the other hand an output record is updated via ChannelAccess
899 or database processing, the device support for this record type
900 deposits the new value to be written in the driver's tag table
901 (the entry for that tag or element of an array tag), and marks the tag
902 to be written.
903 The next time around in the driver scan task, the driver recognizes that
904 the tag should be written instead of read, and writes the tag to the PLC,
905 and resets the 'please write' flag, so the next time around, we're back
906 to reading the tag.
907 If you have various records all associated with elements of an array tag,
```

```
908 and these records get processed at about the same time, the following can happen:
909 Record A processes, updates array element Na of the array tag,
910 and marks the array to be written.
911 If now records B, C, ... process, updating array elements Nb, Nc, ...,
912 (doesn't matter if all the Nx are different or not),
913 the array tag has already been marked for writing, and if the EIP_verbosity
914 is high enough, you get the 'already writing' message.
915 Most of the time, this is not a problem.
916 If the affected records process at about the same time, it's to be expected,
917 and you can simply set EIP_verbosity=5 or lower to hide the message.
918 If, on the other hand, you would have expected the driver to handle the
919 'write' between record processings, this would indicate a problem.
920 Example:
921 The one and only output record with OUT="@plc tagname S 5"
922 configures the driver to scan the 'tagname' every 5 seconds.
923 If you now process the record every second by e.g. entering
924 new value via ChannelAccess, you'll see about 4 'already writing'
925 messages, because the driver will only write every 5 seconds.
926 But if you only process the record every 10 seconds, you should
927 see no message, because the last new value should have been written
928 by the time you enter a new value.
929
930 * Files
931 ether_ip.[ch]    EtherNet/IP protocol
932 dl_list*         Double-linked list, used by the following
933 drvEtherIP*      IOC driver
934 devEtherIP*      EPICS device support
935 ether_ip_test.c  main for Unix/Win32
936
```

epics/ether_ip RSS Atom