ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

EPICS MODULE FOR BECKHOFF ADS PROTOCOL

Jernej Varlec*, Jure Varlec, Žiga Oven, Cosylab d.d., Ljubljana, Slovenia

Abstract

With increasing popularity of Beckhoff devices in scientific projects, there is a rising need for their devices to be integrated into EPICS control systems. Our customers often want to use Beckhoff PLCs for applications that must handle many inputs with fast cycle times. How can we connect Beckhoff devices to EPICS control systems without sacrificing this performance?

Beckhoff offers multiple possibilities when it comes to interfacing with their PLCs or industrial PCs, such as Modbus, OPC UA, or ADS protocol. While all of these could be used for the usual use cases, we believe that for more data intensive applications, ADS works best. For this reason, Cosylab developed an EPICS device support module that implements advanced ADS features, such as ADS sum commands, which provide fast read/write capabilities to your IOCs.

INTRODUCTION

When designing EPICS device support, there are two questions one usually considers: how the device support will communicate with the target devices, and which representations of data must be supported.

ADS for Communication

Invented by Beckhoff, *Automation Device Specification* (ADS) [1] is an open protocol that is used for interconnecting various Twincat software modules the company provides, such as event logger, HMI framework, or Twincat PLC runtimes, among others. These modules are considered as being independent virtual devices and form a server/client relationship; an example of this is a field on the HMI screen getting an update from a Twincat Runtime, all via ADS.

Within this relationship, as shown in Fig. 1, servers and clients communicate with ADS request/response messages which need to be, somehow, routed to the correct ADS device. This message routing is the responsibility of the AMS router, which is part of every Beckhoff Twincat device. The router checks the AMS header part of the message and reads the target port and address from it. Beckhoff provides fixed specification, called AMS ports [2], for their internal software modules. For example, typical Twincat 3 runtime uses AMS port 851.

The AMS ports are the first identifier required for communication within the Twincat system, the second part are the *AMS NetIDs*. These look similar to IP addresses with additional octets appended, and often they are: Users may find it easiest to just use their device's IP address, and then append "`.1.1`"[1] to it. Note that the AMS NetID can be anything, as long as it is unique.

In general, there are two types of ADS communication, asynchronous and notification. Asynchronous is exactly what it says on the tin: the client sends the request message to the server, then continues to operate normally until the server provides the client with the response, be it success or error. Notification-based communication, on the other hand, allows the clients to register themselves to the server, which then autonomously provides updates when values client registered change.

Beckhoff provides the ADS client functions in a C++ library published on Github [3].

IEC 61131-3 Data Types

Another thing to consider is what data types are used by a Twincat runtime. Twincat supports all the typical data types one would expect, such as signed an unsigned integer types up to 64-bit width, 32- and 64-bit floating point numbers, support for arrays and strings, pointer and reference types, and structures. But, since it conforms to the IEC 61131-3 standard [4], it also supports some more 'exotic' data types, such as 32-bit *DATE*, *DATE_AND_TIME*, *TIME_OF_DAY*, and their 64-bit versions, and a generic type, called *ANY*.

The list of supported types is quite long and supporting them all could prove to be a challenge. Another thing we must consider is the data types that EPICS records support. For example, *longin* record supports 64-bit signed integers, which means support for writing 64-bit unsigned types into this record type could prove to be problematic.

Of course, speed and stability should be considered as well. One might not think about it much as long their use case requires mere hundreds of reads or writes at a time,

[1] `.1.1.` is used here as a typical example

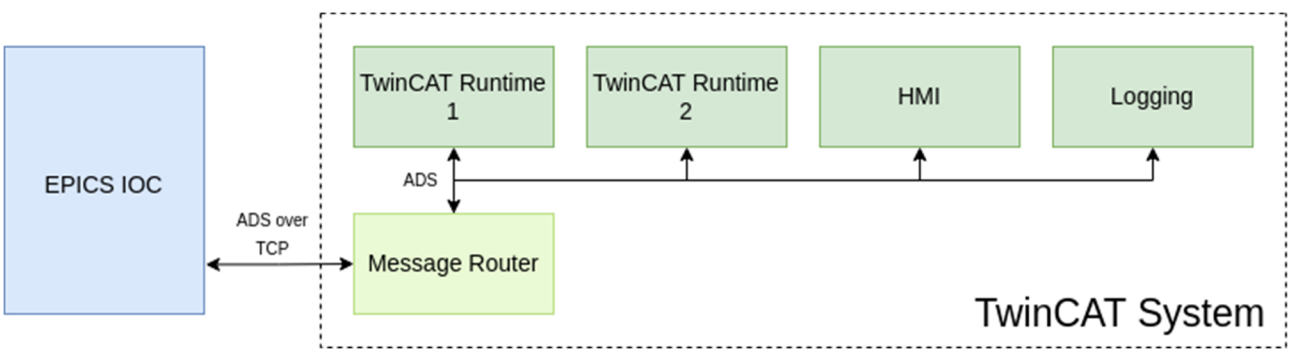

Figure 1: ADS communication via the Message router.

* jernej.varlec@cosylab.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

but the problem quickly escalates if the number of simultaneous reads jumps into four digits; the PLC has to dispatch a large number of responses, the message router must distribute them back to the client, and let's not forget the messages are distributed using TCP/IP. The overheads combined can result in severe performance impact, which can, in turn, result in EPICS scan threads being over-run, which is not desirable to say the least. And, it turns out, that speed and stability can go hand in hand.

ADS DEVICE SUPPORT

Autoparam

We solve the problem of supporting a large amount of data types with AutoparamDriver [5]. This adds a layer of abstraction above Asyn [6], which simplifies the device support a bit: one no longer has to write read and write functions for each Asyn interface. Instead, read and write function are defined for data types. The ADS device support only needs to implement the read and write functions for:

- Integers
- Digital I/O
- Floating point numbers
- Arrays

Data types surrounding time and date are not supported directly. Inside a Twincat system, these are represented by *DWORD* 32-bit unsigned integer type. There is a possibility of this being implemented at some point when a use case is identified.

Unsigned 64-bit integers, e.g., *LWORD* or *ULINT*, are also not supported. EPICS records, that would be likely targets for those types, only support 64-bit signed types.

Sum Commands

ADS C++ library developed by Beckhoff provides standard read and write methods, which are fine as long as there aren't too many requests sent at once. Because the ability to read thousands of variables and being stable is a requirement for the ADS device support, plain read and write methods are not good enough. Thankfully, ADS provides another option named 'Sum commands' [7], which allow us to pack many variables into a single ADS message. This is similar to what the EPICS Modbus module does, except it is not limited to 125 16-bit registers. It can theoretically read any data type and does not have an upper limit to how many variables in a chunk it can read. Practically, there are two limitations:

1. AMS message router can only handle 2048kB of data in a single message.
2. PLC cannot start another cycle until it resolves all ADS requests. This means that the PLC CPU can be stalled if one requests a lot of variables in a single chunk.

For the above reason, Beckhoff recommends no more than 500 variables per read, and this is also the default for the ADS device support. If the EPICS developer knows that they can afford slower PLC cycle times, they can increase this number at IOC init.

ADS device support implements sum reads as a default reading option; the device support automatically organizes all requests from records into chunks, and starts a scan thread, which continuously retrieves fresh values from the target PLC. Sum writes are not supported at the moment.

Using the Device Support

In order to use ADS device support, EPICS integrator needs to know how to interface through EPICS records and how to initialize the device support through EPICS start-up script.

EPICS interface, what we could also call *Address format*, or Asyn *reason*, is comprised of the:

1. **Data type:**
   specifies one of the supported PLC data types, e.g., *USINT, LREAL, BOOL*, etc. If the target variable is an array, append the '[]' to the datatype, except for strings, e.g., *USINT[], LREAL[], STRING*.
2. **Number of element (if requesting arrays):**
   is used to specify number of elements for array access, as well as to specify the length of *STRING* PLC variables, e.g., N=25.
3. **Operation (ADS command):**
   specifies if the PLC variable is read (R) or written (W).
4. **AMS port:**
   port in string or numerical format, e.g., *P=PLC_TC3*
5. **ADS variable:**
   ADS variable name in string format, e.g., *V=Main.temperature.*

An example, in which one wishes to read a *BOOL* variable named *switchStatus*, would be: *field(INP, "@asyn(test 0 0) **BOOL R P=PLC_TC3 V=switchStatus**")*

CONCLUSION

When testing, we wanted to measure the time between each ADS read and if scan threads are being over-run. We were testing the ADS device support in the following setting:

- PLC and IOC were in the same network.
- PLC cycle time was set to 10 ms.
- PLC variables being transferred were 64-bit *LREAL*.
- Number of variables was between 1000 and 10000.
- Around 500 samples were gathered for each SCAN rate.

As shown in Fig. 2, the device support proved to be stable at all SCAN rates (tested from .1 to 1 second) for any number of 64-bit variables up to 10000. Some more time was required for *I/O Intr* scan rate, but that appears to be due to comparing new value with the old one in order to detect value change.

REFERENCES

[1] Beckhoff ADS, https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html&id=6446904803799887467

[2] AMS Ports, https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html&id=6446904803799887467

[3] Beckhoff ADS C++ Library, https://github.com/Beckhoff/ADS

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

Number of records	Record scan rate	Average [s]	Minimum [s]	Maximum [s]	Expected [s]	Number of records	Record scan rate	Average [s]	Minimum [s]	Maximum [s]	Expected [s]
1000	.1 second	0,100	0,100	0,100	0,100	5000	.1 second	0,100	0,100	0,100	0,100
1000	.2 second	0,200	0,200	0,200	0,200	5000	.2 second	0,200	0,200	0,200	0,200
1000	1 second	1,000	1,000	1,000	1,000	5000	1 second	1,000	1,000	1,000	1,000
1000	I/O Intr	0,010	0,006	0,020	nil	5000	I/O Intr	0,326	0,302	0,372	nil
2500	.1 second	0,100	0,100	0,100	0,100	10000	.1 second	0,100	0,100	0,100	0,100
2500	.2 second	0,200	0,200	0,200	0,200	10000	.2 second	0,200	0,200	0,200	0,200
2500	1 second	1,000	1,000	1,000	1,000	10000	1 second	1,000	0,991	1,013	1,000
2500	I/O Intr	0,050	0,044	0,071	nil	10000	I/O Intr	1,362	1,282	1,443	nil

Figure 2: ADS device support test results.

[4] IEC 61131-3, https://webstore.iec.ch/publication/4552

[5] AutoparamDriver, https://epics.cosylab.com/documentation/autoparamDriver/

[6] Asyn driver, https://epics-modules.github.io/master/asyn/

[7] ADS sum commands, https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_adsdll2/124835083.html&id=