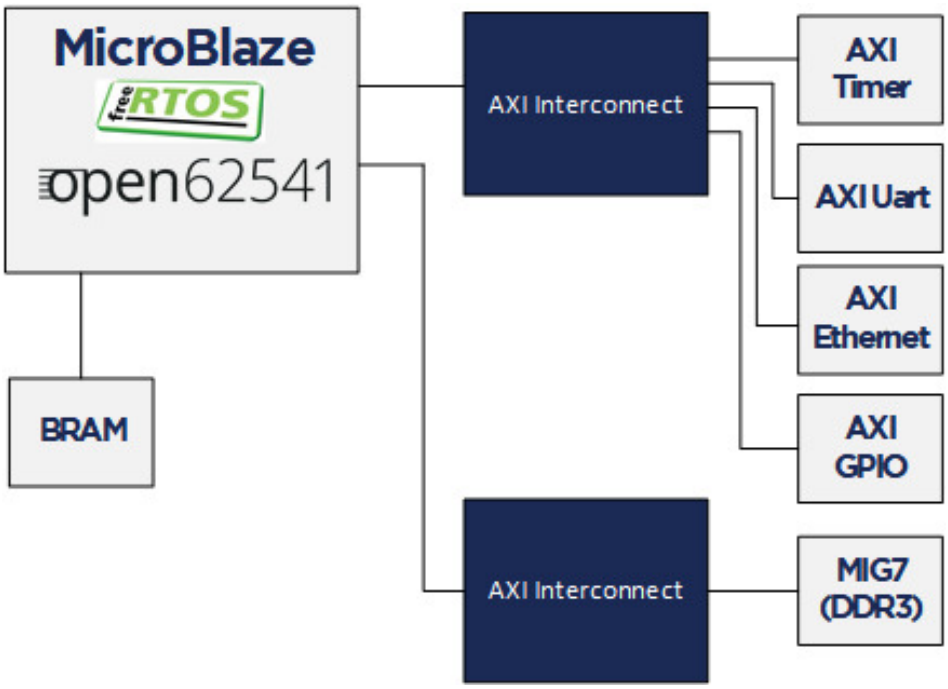
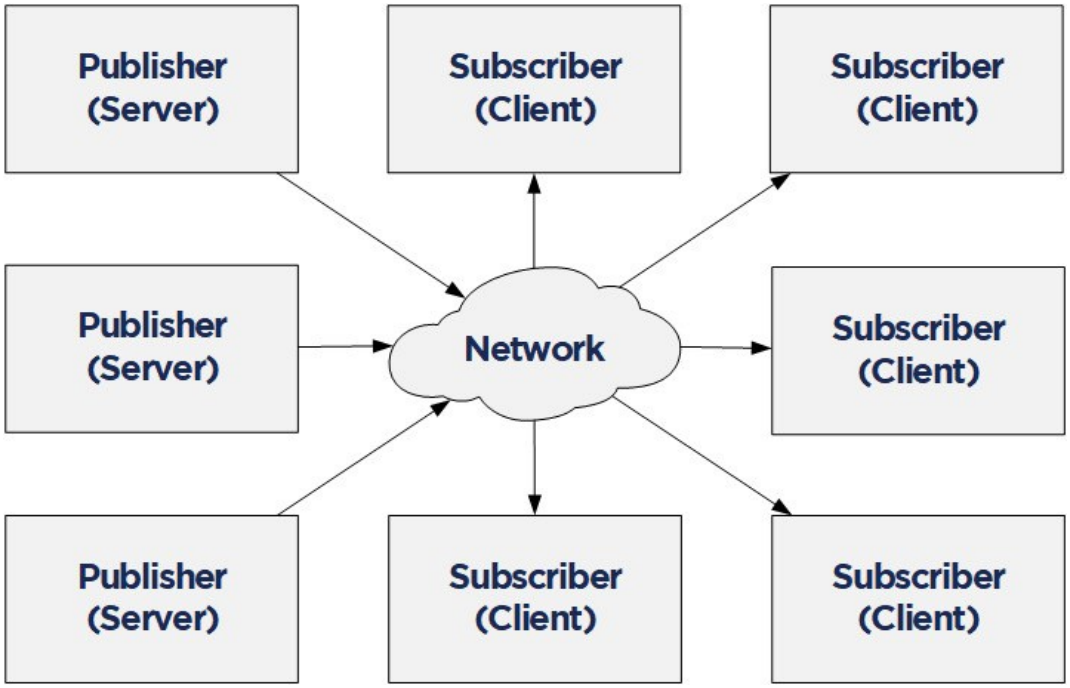


OPC UA PubSub on a FPGA using open62541

2. Oktober 2019

In a first step a simple OPC UA server was set up on a FPGA (see our previous blog post: OPC UA Server on a FPGA using open62541). As a starting point it would be good to begin with this example since the PubSub description builds up on the basic OPC UA server.
Compared to the Client/Server mechanism, the Publish/Subscribe model is even more interesting in the context of Time Sensitive Networking (TSN). PubSub is defined in Part 14 of the OPC Unified Architecture specification and it allows one-to-many or many-to-many connections. In combination with a TSN sub-layer it can fulfill the real-time requirements for the industry.



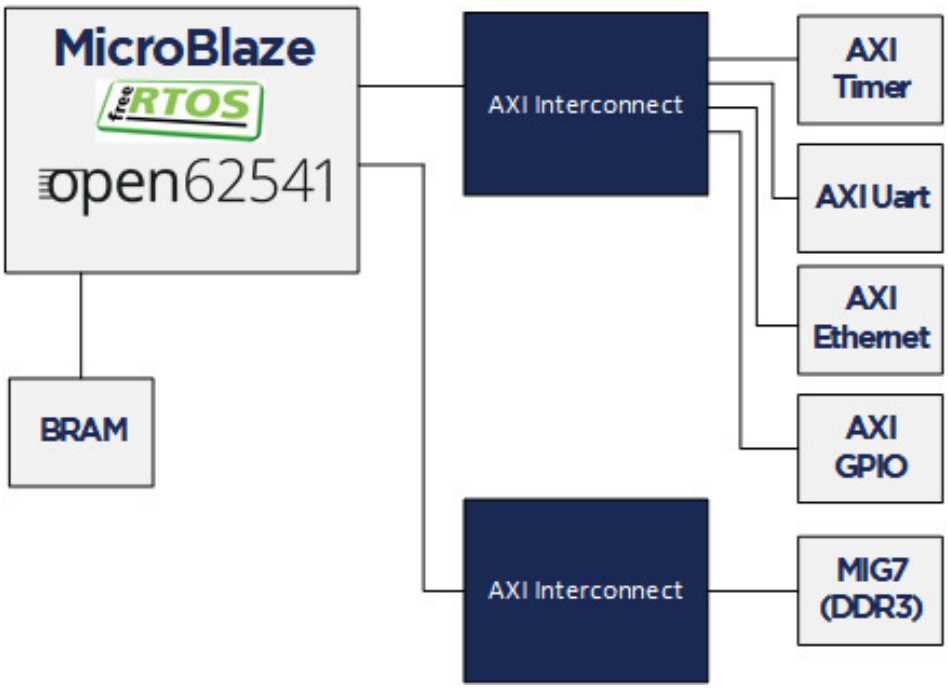
Together with NetTimeLogic's TSN products or the TSN IIC® Plugfest Application (Talker/Listener) an open62541 PubSub application in a MicroBlaze Softcore can be easily combined. For the future we are targeting to realize the TSN Testbed Interoperability Application with the open62541 OPC UA stack and using NetTimeLogic's TSN End Node IP core as realtime sub-layer.

The example FPGA project and the application are available here:
https://github.com/NetTimeLogic/opcua/tree/PubSub_example

The open62541 implementation is available here (v1.0rc5):
https://github.com/open62541/open62541/tree/v1.0-rc5

Introduction

Compared to the Client/Server example no changes in the MicroBlaze FPGA design are needed. However, some adjustments in the CMake and BSP for lwip are required.
The following implementation is based on the open62541 documentation which describes how to build the library and how to work with Publish/Subscribe. The application creates an OPC UA server thread which is publishing a Dataset. It runs with FreeRTOS and lwip. The FPGA use a MicroBlaze softcore with DDR3, Axi Ethernet Lite, Axi Uart Lite AXI GPIO and AXI Timer. As hardware the same Arty A7-100T development board from DIGILENT as before is used.



Required tools

To build the full project, the following tools are required:

To build the full project following tools are required:

- Xilinx Vivado 2019.1
- Xilinx SDK 2019.1
- CMAKE (Python 2.7.x or 3.x)
- UA Expert
- Wireshark

BSP adjustments for LWIP

For the simple OPC UA server some adjustments were needed in the lwip BSP of Xilinx SDK.

See Line 10-19: https://github.com/open62541/open62541/blob/master/arch/common/ua_lwip.h

The Pub/Sub functionality need some more adjustments of the BSP. It should be enough to enable the LWIP_IGMP. Nevertheless, it was not possible to generate successfully the BSP again with this option enabled. As a workaround the additional needed defines are added to the already created (in the previous post) open62541 section in the lwip211.tcl (**bold**) file. This allows to use the standard compilation flow afterwards .

1. Go to:

C:\Xilinx\SDK\2019.1\data\embeddedsw\ThirdParty\sw_services\lwip211_v1_0\data

2. Open the lwip211.tcl
3. Search the proc generate_lwip_opts {libhandle} and go to the end of this procedure
4. Before the line puts $lwipopts_fd "\#endif" add the following code:

```
#OPEN62541 implementation
set open62541_impl  [expr [common::get_property CONFIG.open62541_impl $libhandle] == true]
if {$open62541_impl} {
      puts $lwipopts_fd "\#define LWIP_COMPAT_SOCKETS 0"
      puts $lwipopts_fd "\#define LWIP_SOCKET 1"
      puts $lwipopts_fd "\#define LWIP_DNS 1"
      puts $lwipopts_fd "\#define SO_REUSE 1"
      puts $lwipopts_fd "\#define LWIP_TIMEVAL_PRIVATE 0"
      puts $lwipopts_fd "\#define LWIP_IGMP 1"
      puts $lwipopts_fd "\#define LWIP_MULTICAST_TX_OPTIONS 1"
      puts $lwipopts_fd ""
}
```

5. Save the file

After this change and a restart of Xilinx SDK the new option will be visible in the BSP settings GUI of the lwip stack.

Design preparation

For the detailed design preparation steps please check the previous post OPC UA Server on a FPGA using open62541.

Custom information models

In the basic OPC UA server example the default value "reduced" is used as UA_NAMESPACE_ZERO option. For the UA_ENABLE_PUBSUB option it will compile an additional nodeset and datatype file into the name space zero generated file. Depending on what information will be published this might be not enough. To be on the safe side UA_NAMESPACE_ZERO = "FULL" would be the easiest solution. Since the MicroBlaze CPU is not a very powerful, it is not recommended to use the full namespace. This example would take up to 30 minutes, until the server

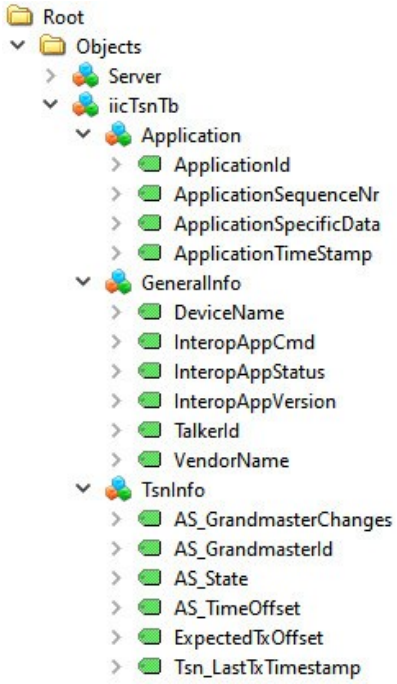
is up and running! It is highly recommended to use an optimized/customized nodeset for such an application.
https://opcua.rocks/custom-information-models/

XML Nodeset Compiler

Most probably in a final application all the variables/objects etc. are not defined manually in the code. There are different tools (commercial but also open source) available to create this information in a GUI. Out from these tools an XML with the OPC UA Nodeset schema can be exported.
Open62541 provides a compiler which creates C code from the XML Nodeset. This code creates then all the object instances as defined.
The complete documentation can be found here:
https://open62541.org/doc/1.0/nodeset_compiler.html
The compiler result for iicNs.c/h are available in git.

Nodeset in this example

Since this example is targeting for the IIC TSN Testbed application, the Nodeset from there is used. It has the following structure with different types of variables:



For this Information Model the minimal Nodeset with PubSub is not sufficient, therefore a customized one was created. This can be done as described above, or even simpler, just by using the already precompiled open62541.c/h files

CMAKE

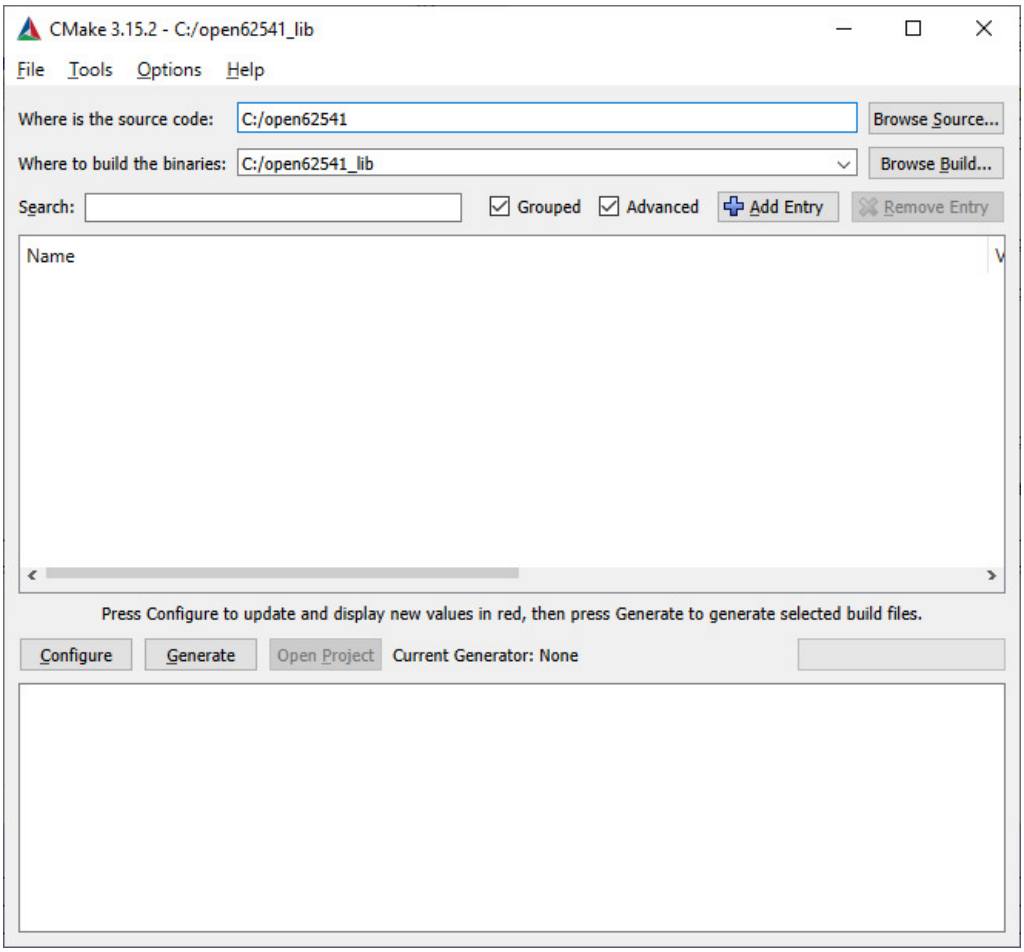
For this example, the open62541 tag v1.0rc5 was used:
https://github.com/open62541/open62541/tree/v1.0-rc5
The easiest way is to work with the CMake GUI. Later it can be used in Xilinx SDK.
If the CMake library build is already available only two adjustments are needed:

```
UA_ENABLE_PUBSUB = ON
UA_ENABLE_PUBSUB_INFORMATIONMODEL = ON
```

If a new build is created, CMake for open62541 is used with the following adjustment:

```
UA_ENABLE_AMALGAMATION = ON
UA_ENABLE_HARDENING = OFF
UA_ENABLE_PUBSUB = ON
UA_ENABLE_PUBSUB_INFORMATIONMODEL = ON
UA_ARCH_EXTRA_INCLUDES = <path to microblaze/include>
UA_ARCH_REMOVE_FLAGS = -Wpedantic -Wno-static-in-inline -Wre-
dundant-decls
CMAKE_C_FLAGS = -Wno-error=format= -mlittle-endian -Dcon-
figUSE_PORT_OPTIMISED_TASK_SELECTION=0 -DconfigAPPLICATION_AL-
LOCATED_HEAP=3 -DUA_ARCHITECTURE_FREERTOSLWIP
UA_LOGLEVEL = 100 (optional for debugging)
```

1. Start the CMake GUI
2. Select the correct source code path where the open62541 GIT repository is located and the path where the binaries were built last time:



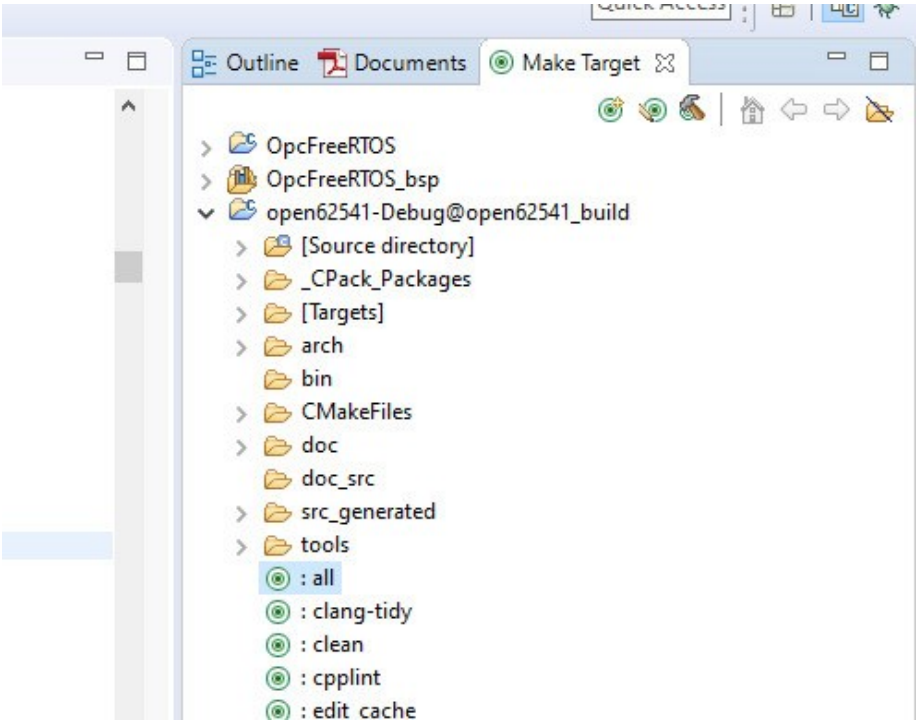
3. Click Configure
4. Change the two parameters:

```
UA_ENABLE_PUBSUB = ON
```

```
UA_ENABLE_PUBSUB_INFORMATIONMODEL = ON
```

5. Click again on Configure and after that on Generate
6. Generate again the open62541.c/h file in Xilinx SDK.

Make Target->all



7. The amalgamation files open62541.c/h should have now the PubSub feature included

A pre-generated version of the open62541.c/h files is available on git.

Creating the OPC UA server application

The complete SDK workspace is available on git.

C/C++ Build Settings

For the build there are no new adjustments required. Please take the same build settings as in the previous post for the simple OPC UA Server.

Linker script

The linker script setting for the memory must be increased, In the example we use now:

- Heap Size: 20MB
- Stack Size: 20MB

OPC UA Server PubSub application

In the Xilinx SDK, the available OpcServer.c can be imported to the OpcServer application project.

In the basic server the thread stack size was defined with 4096. This is not enough anymore and the application will report with the hook functions a StackOverFlow. Therefore, the THREAD_STACKSIZE was increased to 16384.
In a first step the network initialization and the basic OPC UA Server configuration is done. Before the server starts, the PubSub specific setup is needed. The application is targeting to be compatible with the Pub/Sub format for the IIC TSN Testbed interoperability application

Define the PubSub connection

In the PubSub connection mainly the transport profile and the multicast network are defined. For this case we used following settings:
transportProfile: http://opcfoundation.org/UA-Profile/Transport/pubsub-udp-uadp
Network Address URL: opc.udp://224.0.0.22:4840

Add a Publishing dataset

This is the collection of the published fields. All PubSub items are linked to this one. As PublishedDataSetType the following configuration is used:
publishedDataSetType: UA_PUBSUB_DATASET_PUBLISHEDITEMS

Add fields (variables) to the dataset

Here the variables are added by their NodeIds to the Published data set. Depending on the configuration the order of adding the variables has an impact how the published data will look like.
Additionally, a value is set. It is important that the variables have a value (not NULL). If a variable is empty there is just no content for the DataMessage to publish in the PubSub frame.

Add the writer group

The writer group is the important part when it comes to how the message looks like. The whole configuration for the NetworkMessage Header (Extended) is done here (OPC UA Part 14 Chapter 7.2.2.2).
Open62541 allows the specific configuration with the networkMessageContentMask configuration.
For the IIC TSN Testbed interoperability application following settings will be used:

```
writerGroupMessage->networkMessageContentMask = (UA_UADPNETWORKMES-
SAGECONTENTMASK_PUBLISHERID
|                                                                  UA_UA
DPNETWORKMESSAGECONTENTMASK_GROUPHEADER
```

```
| UA_UADPNETWORKMESSAGECONTENTMASK_WRITERGROUPID
| UA_UADPNETWORKMESSAGECONTENTMASK_GROUPVERSION
| UA_UADPNETWORKMESSAGECONTENTMASK_NETWORKMESSAGENUMBER
| UA_UADPNETWORKMESSAGECONTENTMASK_SEQUENCENUMBER
| UA_UADPNETWORKMESSAGECONTENTMASK_PAYLOADHEADER
| UA_UADPNETWORKMESSAGECONTENTMASK_TIMESTAMP);
```

Beside the NetworkMessage Header also settings like the publishing interval or the encoding MimeType are done here.

Add the dataset writer

This part is the second important part and defines how the DataSetMessage Header looks like (OPC UA Part 14 Chapter 7.2.2.3.4).
With the dataSetMessageContentMask and the dataSetFieldContentMask this can be configured.
For the IIC TSN Testbed interoperability application all this additional information is disabled:

```
dataSetWriterMessage->dataSetMessageContentMask = UA_UADPDATASETMESSAGECONTENTMASK_NONE;
dataSetWriterConfig.dataSetFieldContentMask = UA_DATASETFIELDCONTENTMASK_NONE;
```

Start the Server

After all the setup for the variables and the PubSub data set has been done the server is ready to start.
Starting the server takes quite some time with this example. After about two minutes the OPC UA Server starts publishing.

Listen to the OPC UA publisher

If there is no Subscriber available there are other options to understand a bit how the variables are published. Either the UaExpert can give some information or via Wireshark the real PubSub Frame can be analyzed.

Before a connection to the OPC UA server is possible the application needs to be compiled and started on the FPGA. After a successful start you should see the following printout on in the Terminal:

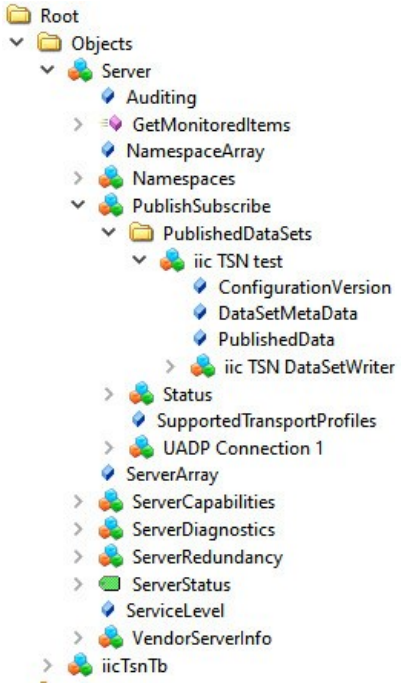
```
-----------------------------------------------------
--------- Starting OPC UA Server application ---------
-----------------------------------------------------
--------- open62541 example created for a    ---------
--------- MicroBlaze design on a Artix7 FPGA ---------
-----------------------------------------------------
--------- NetTImeLogic GmbH, Switzerland     ---------
--------- contact@nettimelogic.com           ---------
-----------------------------------------------------


--------- Init Network ---------
auto-negotiated link speed: 100
auto negotiation done
--------- Init OPC UA Server (pub/sub) ---------
--------- Get Server Config---------
--------- Set Server Config ---------
[1970-01-01 00:01:36.600 (UTC+0000)] error/network      Could not get the hostname
[1970-01-01 00:01:36.640 (UTC+0000)] warn/server        Username/Password configured, but no encrypting SecurityPolicy. This can leak credentials on the network.
--------- Calloc PubSubConnection ---------
--------- Add PubSubConnection ---------
[1970-01-01 00:01:57.330 (UTC+0000)] info/userland      PubSub channel requested
--------- Publish PubSubConnection ---------
--------- Data set field PubSubConnection  ---------
--------- Write Group ---------
[1970-01-01 00:02:03.480 (UTC+0000)] info/network       TCP network layer listening on opc.tcp://192.168.1.10:4840/
```

UA Expert

UA Expert is also a helpful tool to check some stuff about PubSub. With the option UA_ENABLE_PUBSUB_INFORMATIONMODEL the published dataset information is available.

All the configured nodes from the previous steps are now visible (UADP Connection, PublishedDataSets, DataSetWriter etc.) as a structure directly from the server.

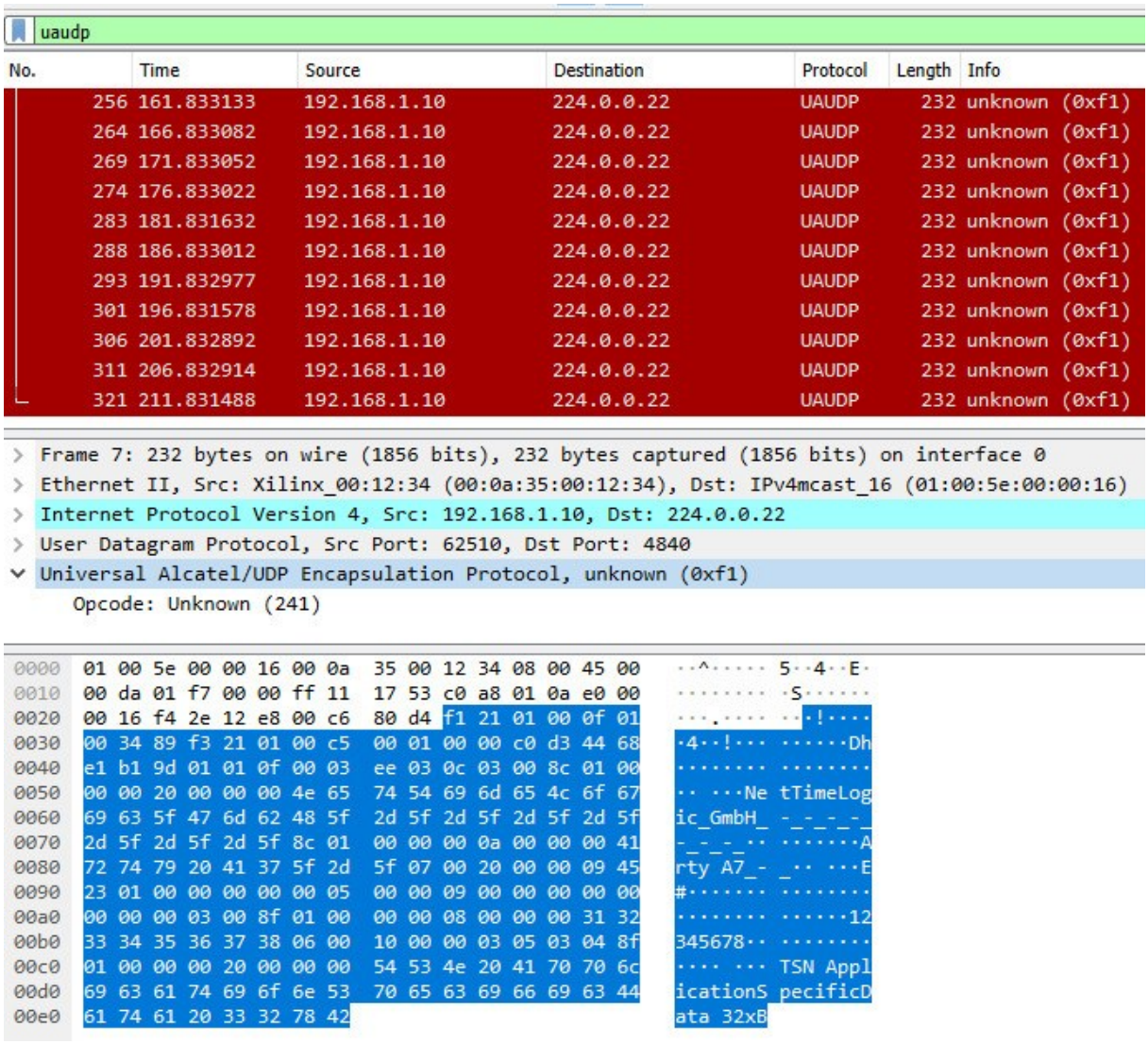


In the Attribute of the object PublishedData all the published variables are visible.

Attribute	Value
▾ NodeId	i=335877163 []
NamespaceIndex	0
IdentifierType	Numeric
Identifier	335877163 []
NodeClass	Variable
BrowseName	0, "PublishedData"
DisplayName	"", "PublishedData"
Description	"", ""
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeIdInvalid (0x80350000)
UserRolePermissions	BadAttributeIdInvalid (0x80350000)
AccessRestrictions	BadAttributeIdInvalid (0x80350000)
▾ Value	
SourceTimestamp	01.01.1970 01:12:18.780
SourcePicoseconds	0
ServerTimestamp	01.01.1970 01:12:18.780
ServerPicoseconds	0
StatusCode	Good (0x00000000)
▾ Value	PublishedVariableDataType Array[15]
▾ [0]	PublishedVariableDataType
▾ PublishedVari...	NodeId
Namespa...	2
Identifier...	Numeric
Identifier	6051
AttributeId	13
SamplingInte...	0
DeadbandType	0
DeadbandVal...	0
IndexRange	
SubstituteValue	Null
MetaDataPro...	QualifiedName Array[0]
› [1]	PublishedVariableDataType
› [2]	PublishedVariableDataType
› [3]	PublishedVariableDataType
› [4]	PublishedVariableDataType
› [5]	PublishedVariableDataType
› [6]	PublishedVariableDataType
› [7]	PublishedVariableDataType
› [8]	PublishedVariableDataType
› [9]	PublishedVariableDataType
› [10]	PublishedVariableDataType
› [11]	PublishedVariableDataType
› [12]	PublishedVariableDataType
› [13]	PublishedVariableDataType
› [14]	PublishedVariableDataType

Wireshark

To check if the content of the published frame manually, Wireshark is the simplest way. This sample application uses a publishing interval of 5000 ms, so every 5s a published frame is received in Wireshark.



Looking into the encoding of the frame above the following information is published:

NetworkMessageHeader				
Version/Flags	Extended Flags1	PubId		
f1	21	01 00		
GroupHeader				
Group Flags	WriterGroupId	GroupVersion	NetworkMessageNumber	SeqNr
0f	01 00	34 89 f3 21	01 00	c5 00
PayloadHeader				
MsgCnt	WriterId			
1	00 00			
Extended NetworkMessageHeader				
TimeStamp				
c0 d3 44 68 e1 b1 9d 01				
DataSetMessageHeader				
Flags1				
1				
DataSetMessage payload				
Field Count				
0f 00				
UA Variable Type	(Array Dimension)	(Array Length)	Variable Data	
03			ee	
03			0c	
03			00	
8c	01 00 00 00	20 00 00 00	4e 65 74 54 69 6d 65 4c 6f 67 69 63 5f 47 6d 62 48 5f 2d 5f 2d 5f 2d 5f 2d 5f 2d 5f 2d 5f 2d 5f	
8c	01 00 00 00	0a 00 00 00	41 72 74 79 20 41 37 5f 2d 5f	
01			00 00 00	
07			00 20 00 00	
09			45 23 01 00 00 00 00 00	
05			00 00	
09			00 00 00 00 00 00 00 00	
03			00	
8f	01 00 00 00	08 00 00 00	31 32 33 34 35 36 37 38	
06			00 10 00 00	
03			05	
03			04	
8f	01 00 00 00	20 00 00 00	54 53 4e 20 41 70 70 6c 69 63 61 74 69 6f 6e 53 70 65 63 69 66 69 63 44 61 74 61 20 33 32 78 42	

Summary

The used version of open62541 (v1.0rc5) allows working with Pub/Sub and allows to do most of the configurations for the header information. However, there were some adaptations required to use it for the IIC TSN Testbed interoperability application.

In our test we saw some problems with some header information.

GroupHeader:

- GroupVersion was assigned
- SequenceNumber was not assigned

Extended NetworkMessageHeader:

- Timestamp was empty

As a workaround we have made some adaptations in the file src/pubsub/ua_pubsub.c by adding some assignments after the following code line:

```
nm.groupHeader.writerGroupId = wg->config.writerGroupId;
```

add:

```
nm.groupHeader.groupVersion = wgm->groupVersion;
nm.groupHeader.sequenceNumber = sequenceNumber;
nm.timestamp = UA_DateTime_now();
```

With these adjustments it was possible to create the frame as shown in the Wireshark capture.
In a next step we will try to be fully compatible with the IIC TSN Testbed interoperability application and combine it with our TSN core for real time publishing.