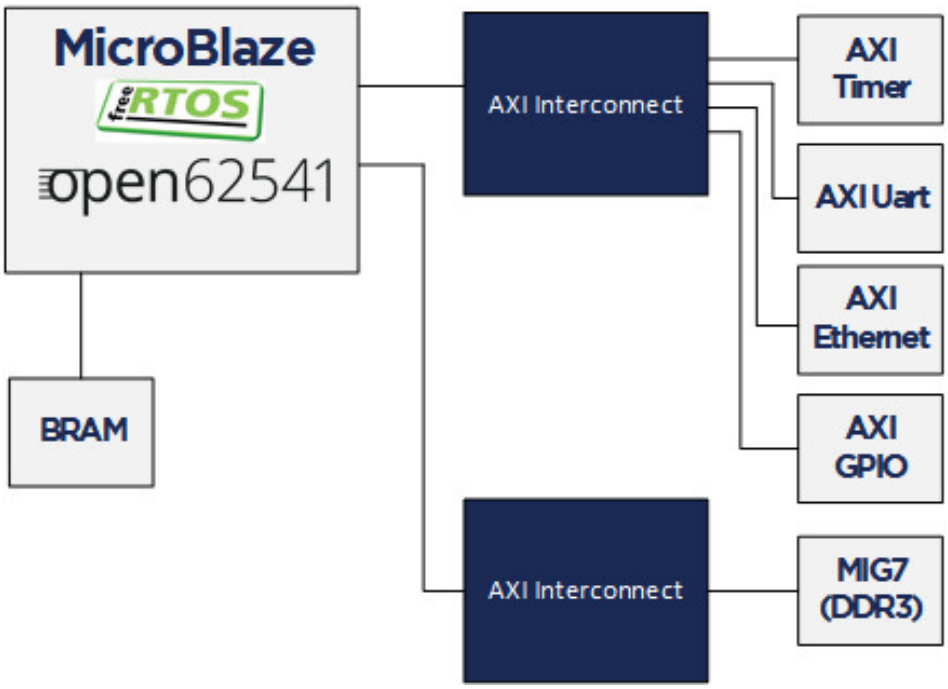


OPC UA server on a FPGA using open62541

30. August 2019

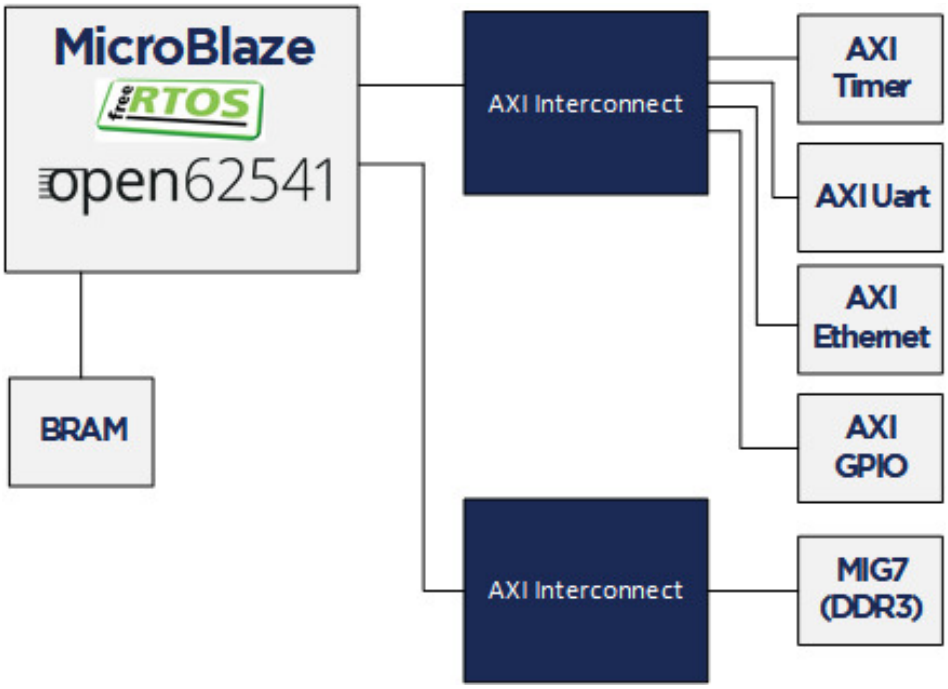
Open Platform Communications Unified Architecture (OPC-UA - IEC62541) is a standardized platform-independent architecture which provides a service-based data exchange. In combination with TSN it allows new possibilities when it comes to high interoperability and deterministic communication.
Based on the open62541 implementation the following steps show how everything has to be setup to get it up and running on a FPGA (Artix7 with MicroBlaze). In combination with NetTimeLogic's complete FPGA based TSN solution you get the full solution for industrial communication 4.0.

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

The open62541 implementation is available here:
https://github.com/open62541/open62541

Introduction

It is not straight forward to get the open62541 OPC UA stack up and running on an embedded system even if FreeRTOS and lwip is supported. The following implementation description is based on the open62541 documentation which describes how to build the library and how to implement a basic OPC UA server. The application creates an OPC UA server thread which is running under FreeRTOS with lwip. The FPGA use a MicroBlaze softcore with DDR3, Axi Ethernet Lite, Axi Uart Lite AXI GPIO and AXI Timer. As hardware an Arty A7-100T development board from DIGILENT is used.



Required tools

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

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

BSP adjustments for LWIP

Open62541 supports “freertosLWIP” as an architecture. In that case it uses the libraries of the target device which are the ones of the BSP in Xilinx SDK.
To be able to compile the open62541 library some adjustments for the lwipopts.h file are needed:

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

Since this file is managed by the BSP in Xilinx SDK, manual modifications are overwritten when the BSP is generated. With the following workaround, it is possible to add the additional defines over the BSP setting GUI.

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. Add before the line *puts $lwipopts_fd "\#endif"* 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 ""
}
```

5. Save the file
6. Open the file lwip211.mld
7. Add the new Parameter e.g. at line 47:

```
PARAM name = open62541_impl, desc = "Used as an open62541 implementation?", type = bool, default = false;}
```

8. Save the file
9. Restart Xilinx SDK

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

Design preparation

Before everything is ready to build the open62541 library, the implemented FPGA design from Xilinx Vivado and a software application project in Xilinx SDK is

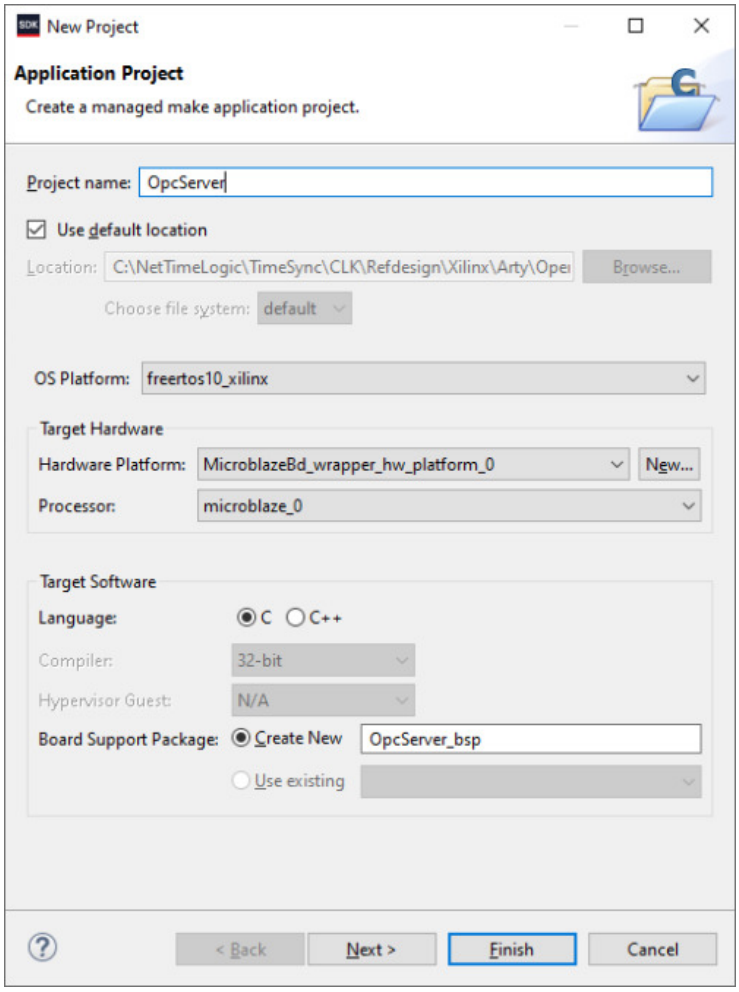
needed. In this example project a MicroBlaze design with DDR3 is used (unfortunately the application does not fit into the available block RAM).

Vivado

The Vivado project can be created with the available tcl script. By running the implementation of the Vivado project the bitstream can be created. With *File->Export->Export Hardware* the hardware definition can be created.
File->Launch SDK starts the SDK.

Xilin SDK

In Xilinx SDK a new empty Application Project with the OS Platform “freertos10_xilinx” can be created.
File->New->Application Project.



fter the project is created some adjustments in the OpcServer_bsp are needed

- Select lwip211 as supported libraries

- Go to the lwip211 and adjust following parameter:
  `api_mode = socket_API`
  `open62541_impl = true`
- Go to the freertos20_xilinx and adjust the following parameters:
  `Check_for_stack_overflow = 1`
  `total_heap_size = 2097152`
- Re-generate BSP sources

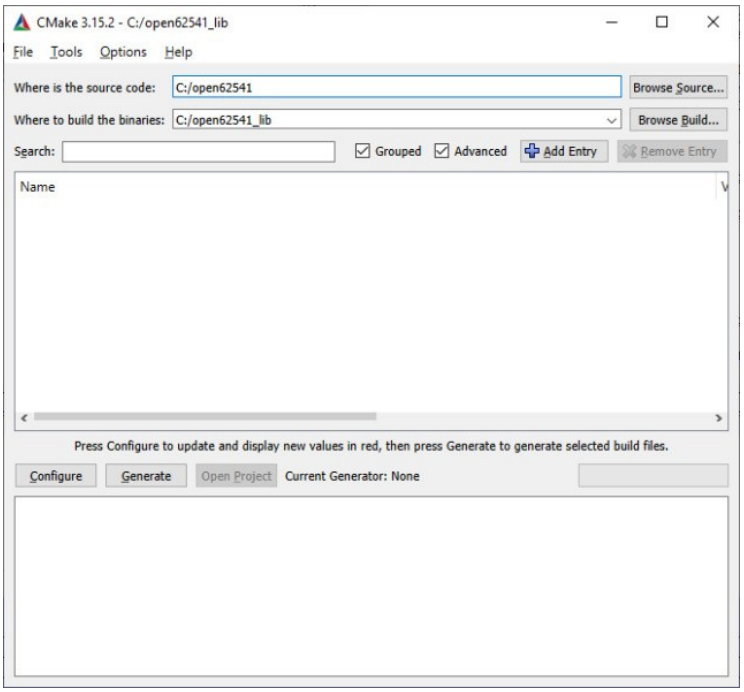
The environment is now ready to start with CMake.

CMake

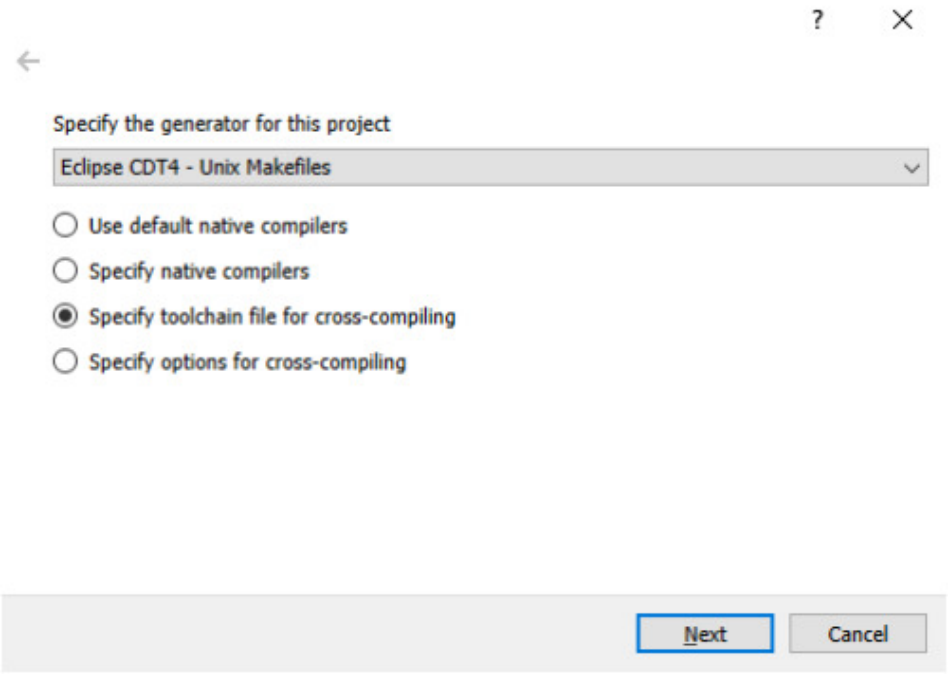
The easiest way is to work with the CMake GUI. Later it can be used in Xilinx SDK. CMake for open62541 is used with following adjustment:

- `UA_ENABLE_AMALGAMATION = ON`
- `UA_ENABLE_HARDENING = OFF`
- `UA_ARCH_EXTRA_INCLUDES = <path to microblaze/include>`
- `UA_ARCH_REMOVE_FLAGS = -Wpedantic -Wno-static-in-inline -Wredundant-decls`
- `CMAKE_C_FLAGS = -Wno-error=format= -mlittle-endian -DconfigUSE_PORT_OPTIMISED_TASK_SELECTION=0 -DconfigAPPLICATION_ALLOCATED_HEAP=3 -DUA_ARCHITECTURE_FREERTOSLWIP`
- `UA_LOGLEVEL = 100 (optional for debugging)`

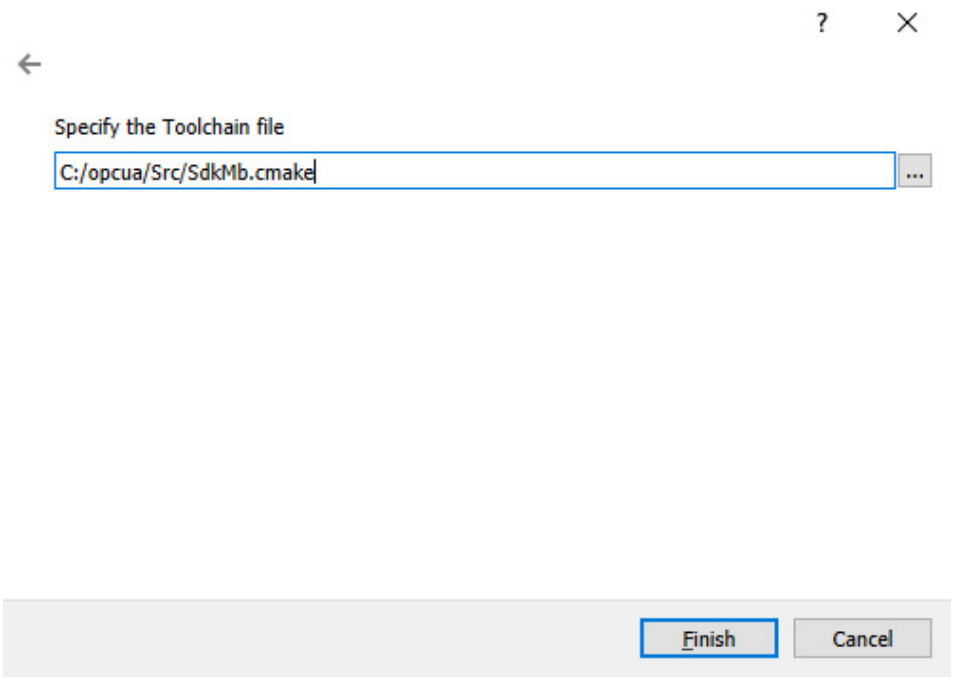
1. Start the CMake GUI
2. Select the correct source code path where the open62541 GIT repository is located and define the path where you want to build the binaries:



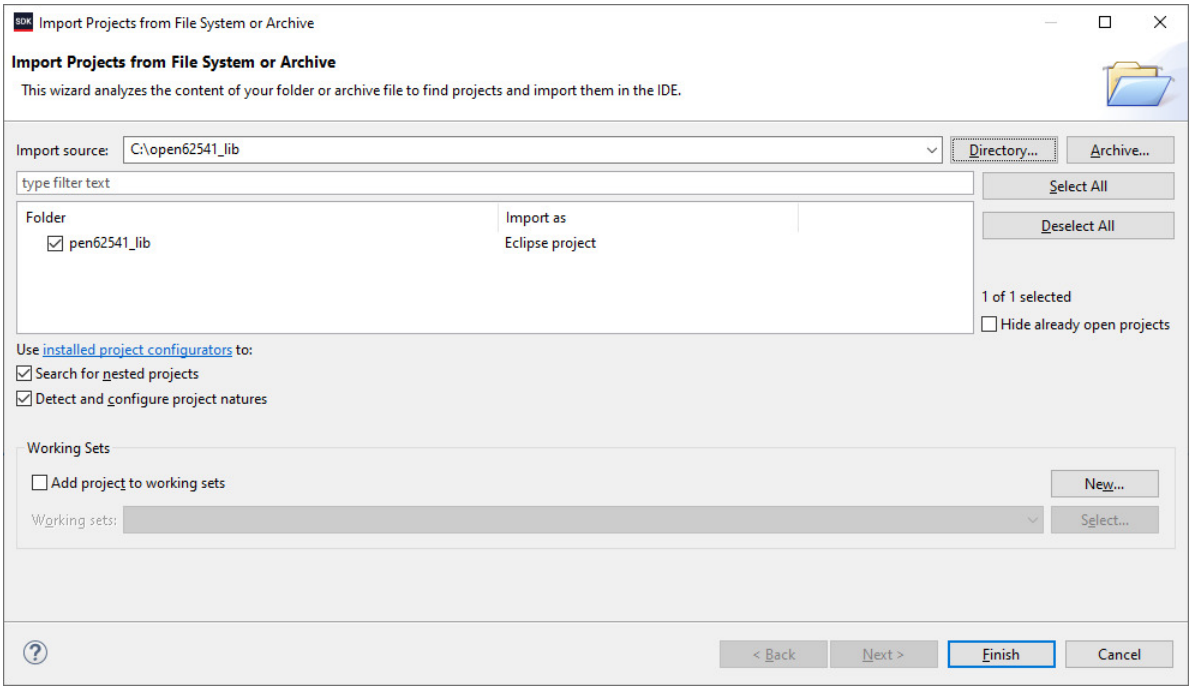
3. Click Configure:



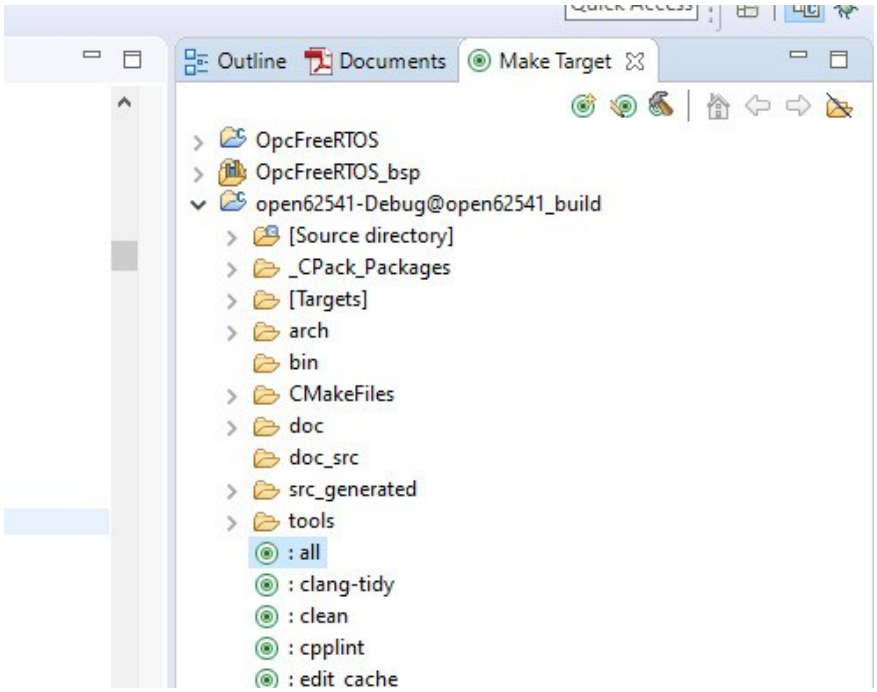
4. Select the CMake file which defines the compilation toolchain and other settings:



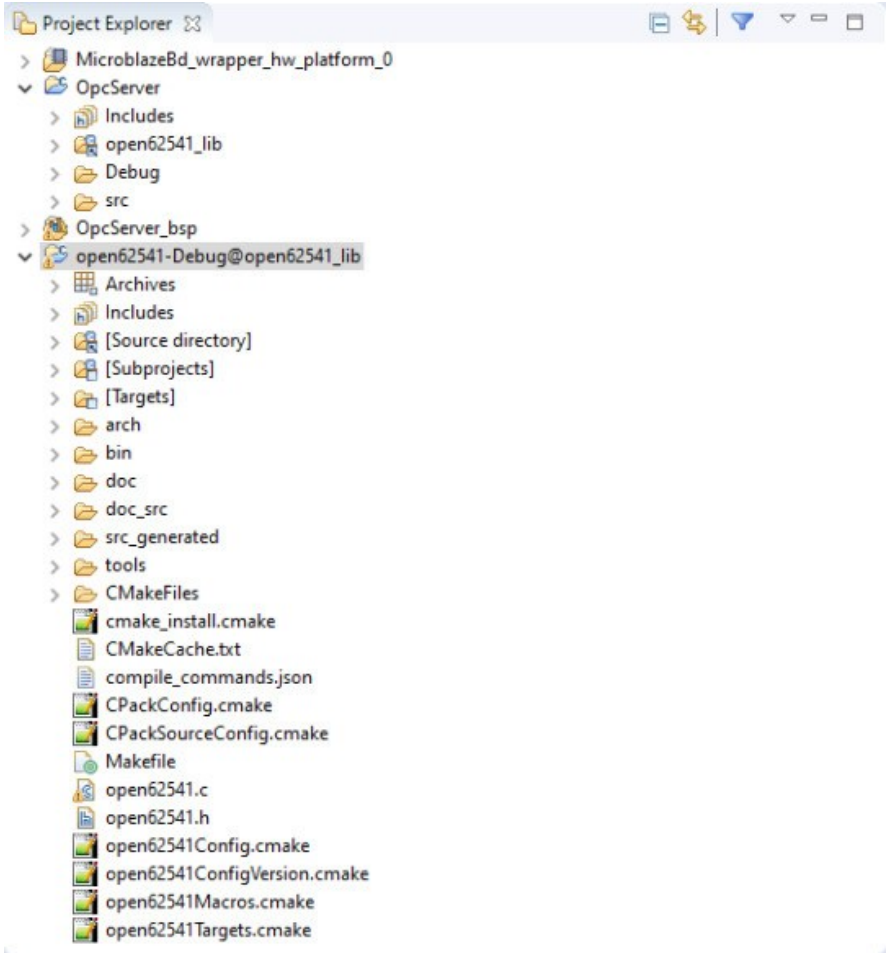
5. Click again on Configure and after that on Generate
6. The Unix Makefiles are now ready and can be added Xilinx SDK workspace: File->Open Projects from File system



7. Now it should be possible to generate the open62541.c/h file in Xilinx SDK. Make Target->all



8. The workspace should have now following structure:

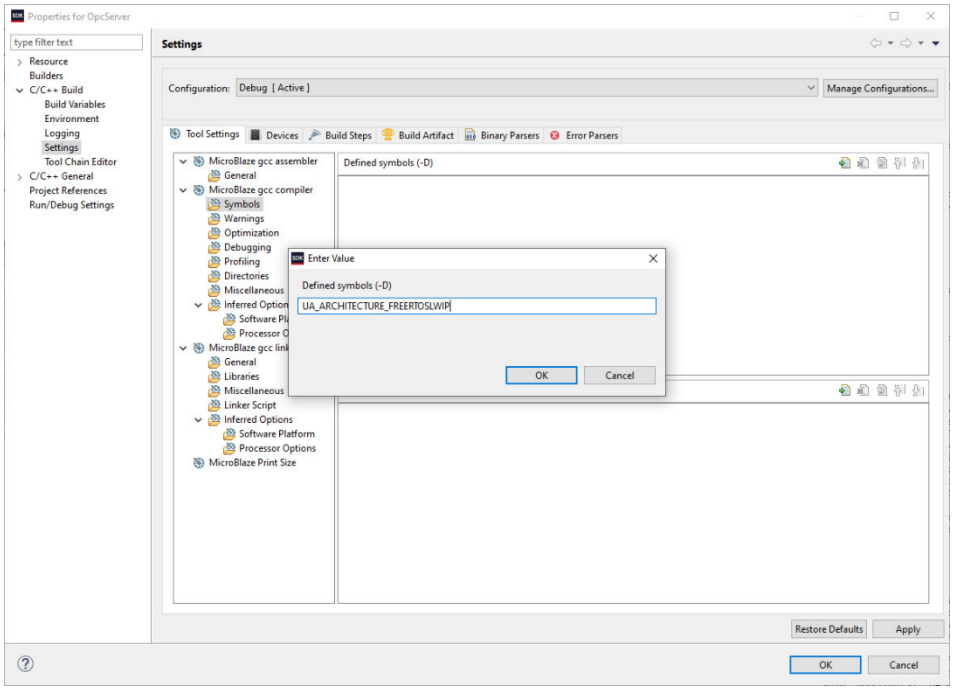


Creating the OPC UA server application

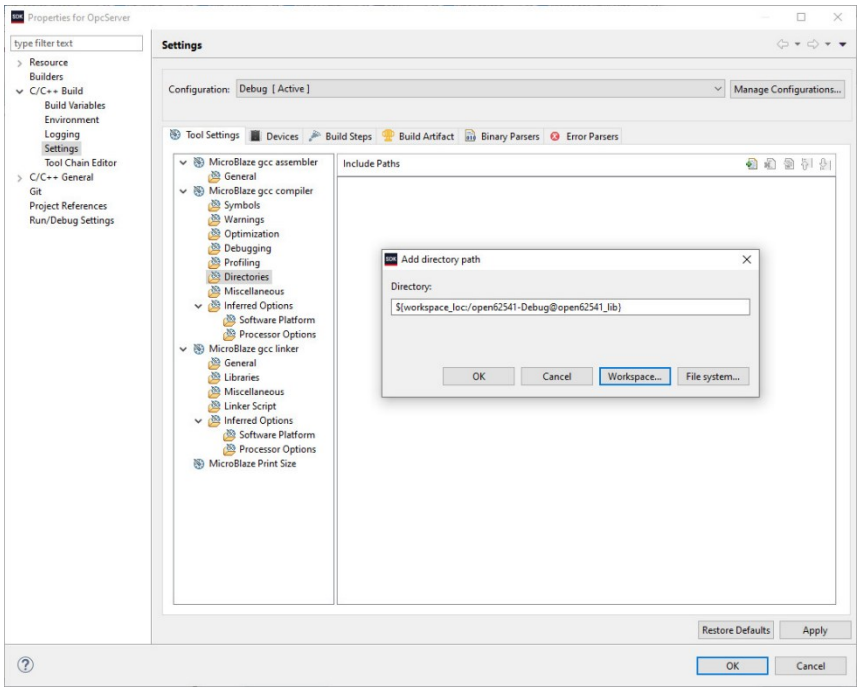
C/C++ Build settings

For a compilation without errors some adjustments in the application project Build settings are required.

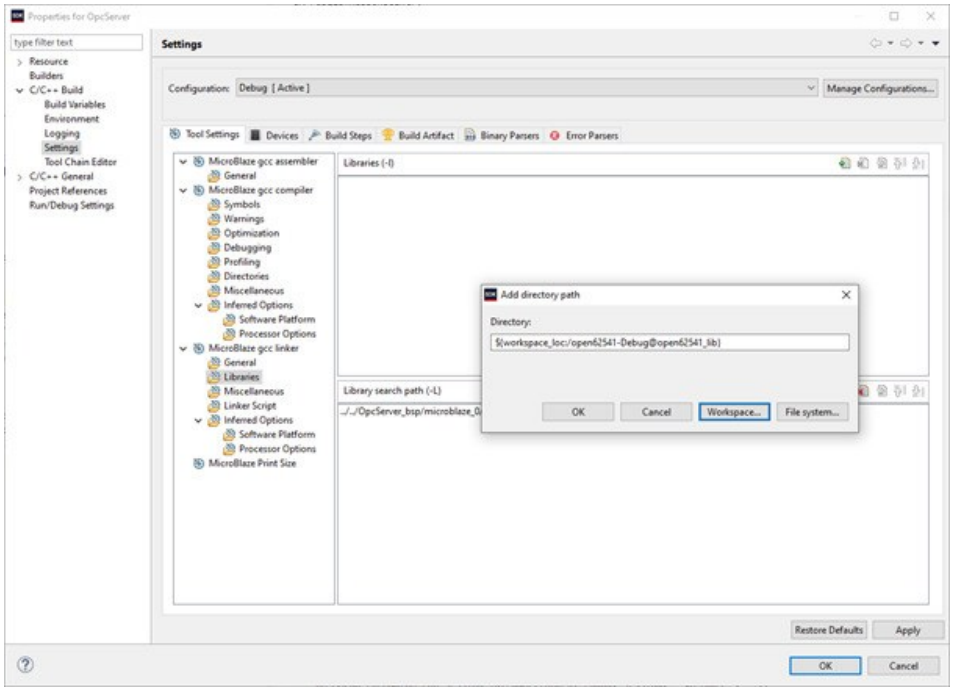
1. Add the symbol for UA_ARCHITECTURE_FREERTOSLWIP



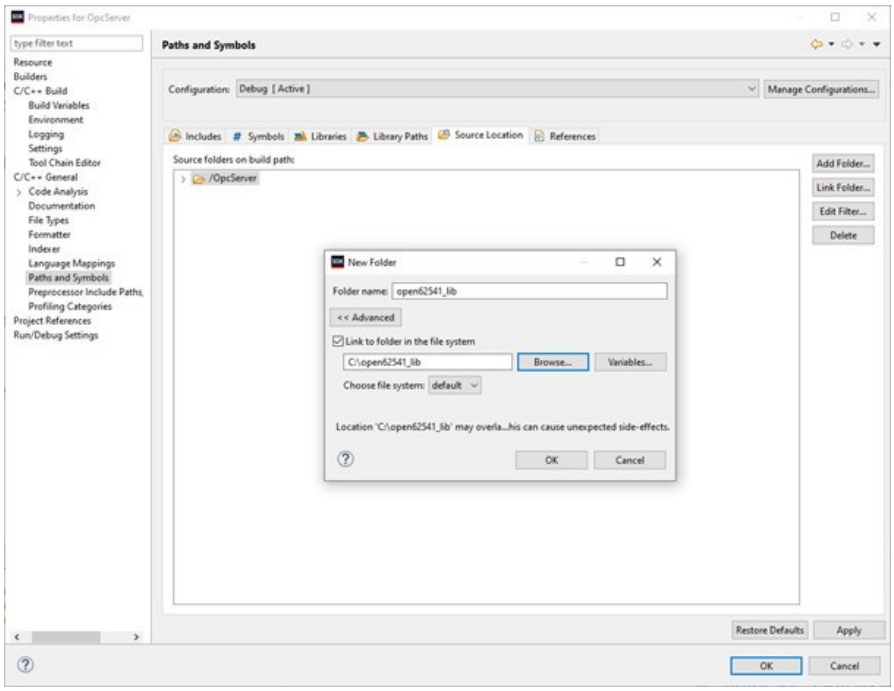
2. Add the open62541 build directory as include path



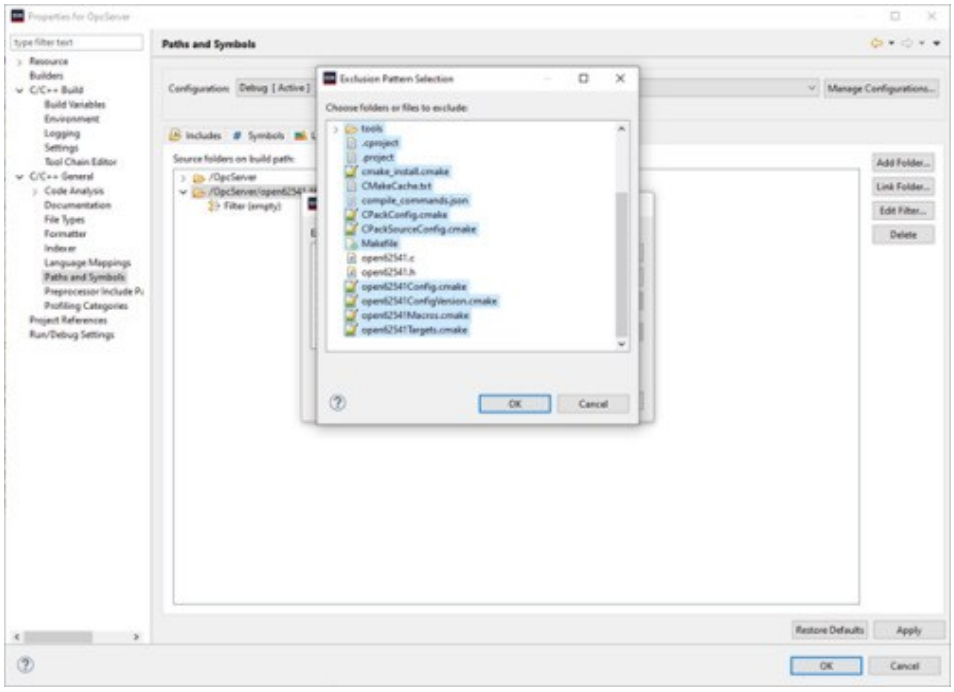
3. Add the open62541 build directory as library search path



4. Link the folder to the source location of open62541.c/h



5. Add an exclusion pattern that only the open62541.c/h are used:

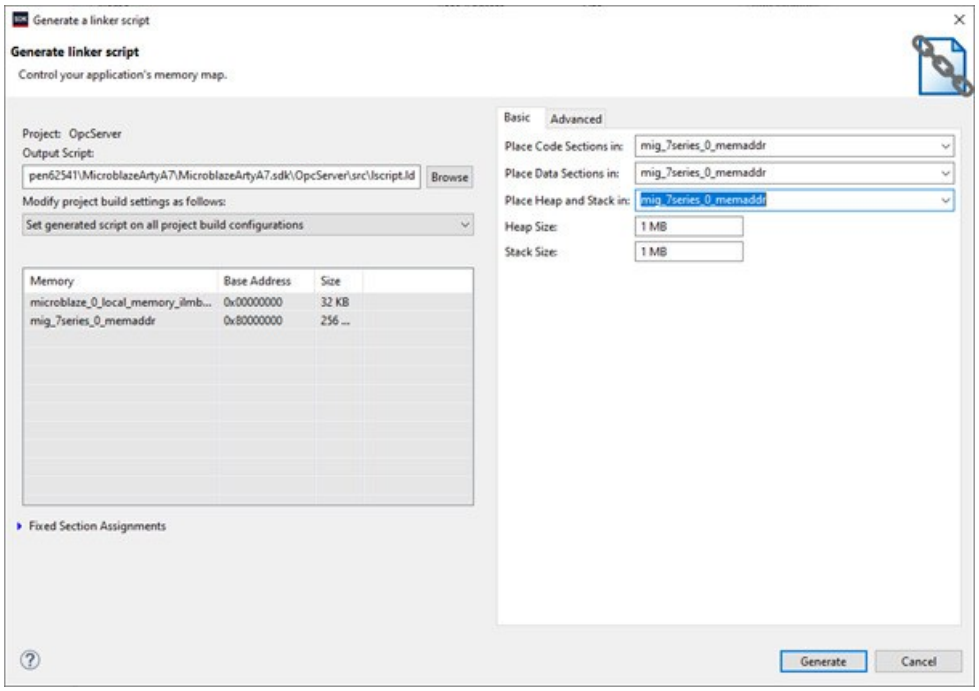


Linker script

The linker script for our OPC UA server application needs some small adjustments. With Xilinx->Generate linker script a new lscript.ld with following settings can be created:

- Heap Size: (min) 1MB

- Stack Size: (min) 1MB



Now the application project should be ready for a successful compilation.

OPC UA Server app

The complate Workspace is here available:
https://github.com/NetTimeLogic/opcua/tree/master/Src/Sdk_workspace
In Xilinx SDK the source file OpcServer.c can be imported to the OpcServer application project.

The thread stack size is defined with 4096 it might be possible that the application is not running properly with other values. However, the hook functions for MallocFailed or StackOverflow might be helpful.
In a first step the network initialization is done. This includes auto negotiation, ip configuration, interface binding and starting the lwip receive thread. After that the opcua thread gets started.
Important for a working server is the configuration and especially the buffer size of the network layer. With the following settings, the server was running without any problems:

```
config->networkLayers->localConnectionConfig.recvBufferSize =
32768;   config->networkLayers-
```

```
>localConnectionConfig.sendBufferSize = 32768;    config->network-
Layers->localConnectionConfig.maxMessageSize = 32768;
```

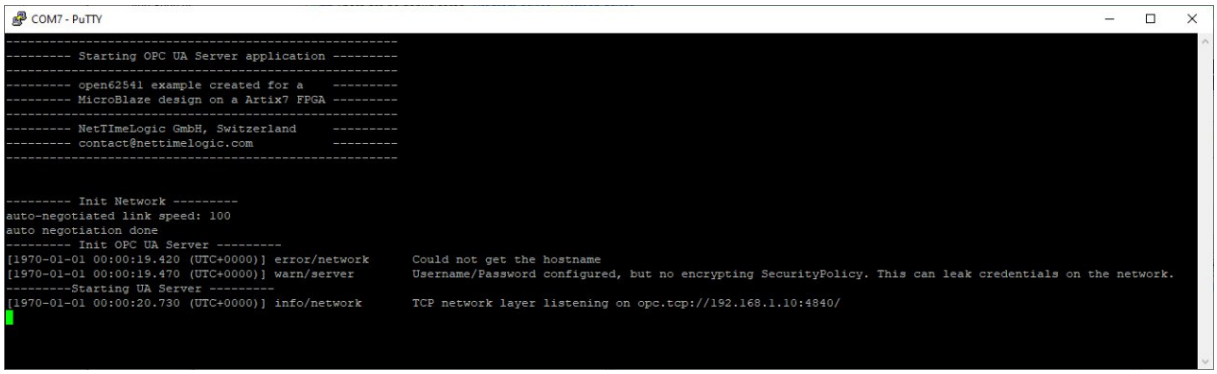
Before the server is started an object and a variable are added to the server. Additionally, a callback for the variable is configured, which allows to control the LEDs on the ArtyA7 board by an OPC client. After that the server gets started and runs until the running variable is set to false (never in this case).

Connecting to the OPC UA Server

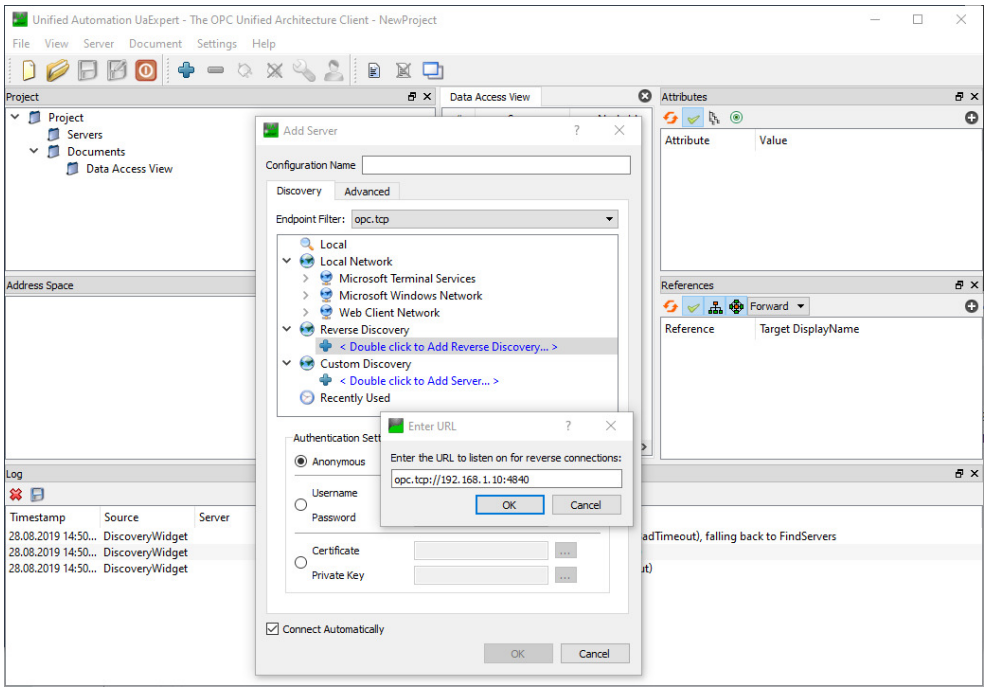
After a successful implementation of the MicroBlaze FPGA design, building the open62541 library and compiling the OPC UA server application everything is ready.

The Arty A7 board should be connected to the PC over USB and the RJ45 is connected to a network interface adapter.

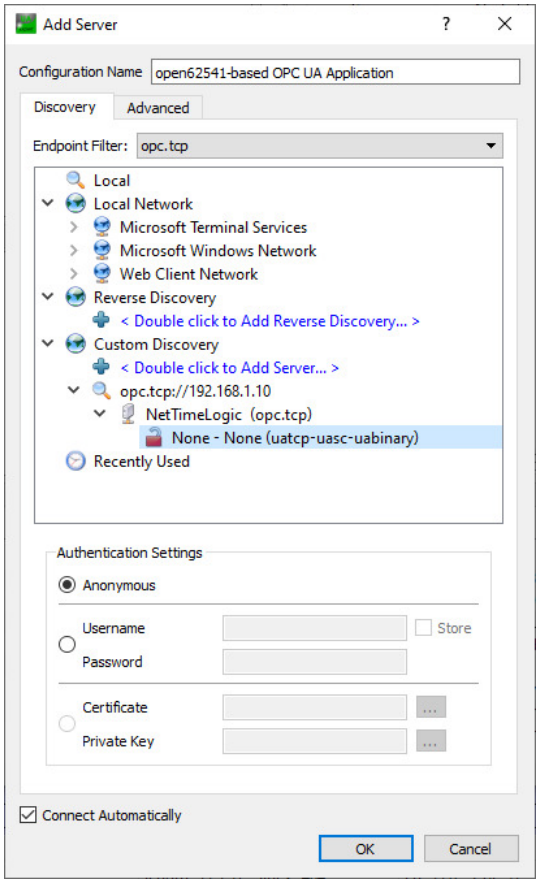
1. Open a serial terminal for the debug print out (baud rate: 115200)
2. Loading the bitstream (from Vivado or SDK)
3. Run the Application (from SDK)
4. If the application has successfully started, in the serial terminal following text is printed out:



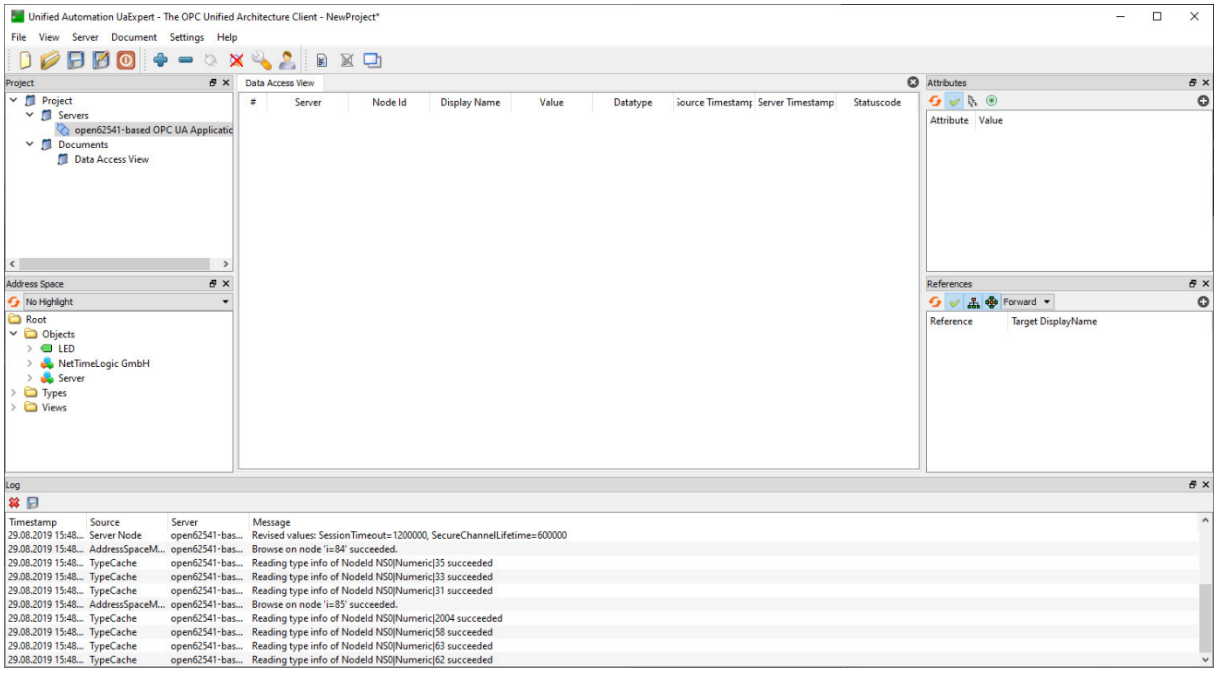
5. Start UaExpert
6. Add the server via “Custom Discovery” with the configured open62541 hostname



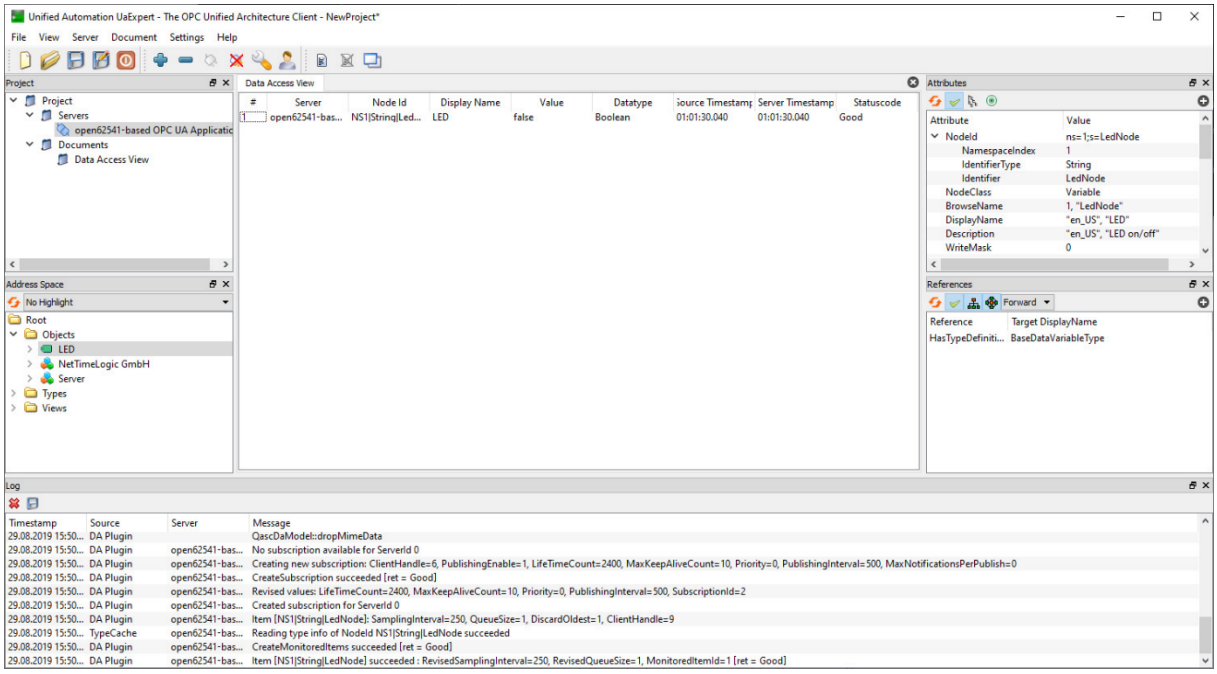
7. Expand the added server to add the connection. In the serial terminal you get already some information that a new connection over TCP was detected



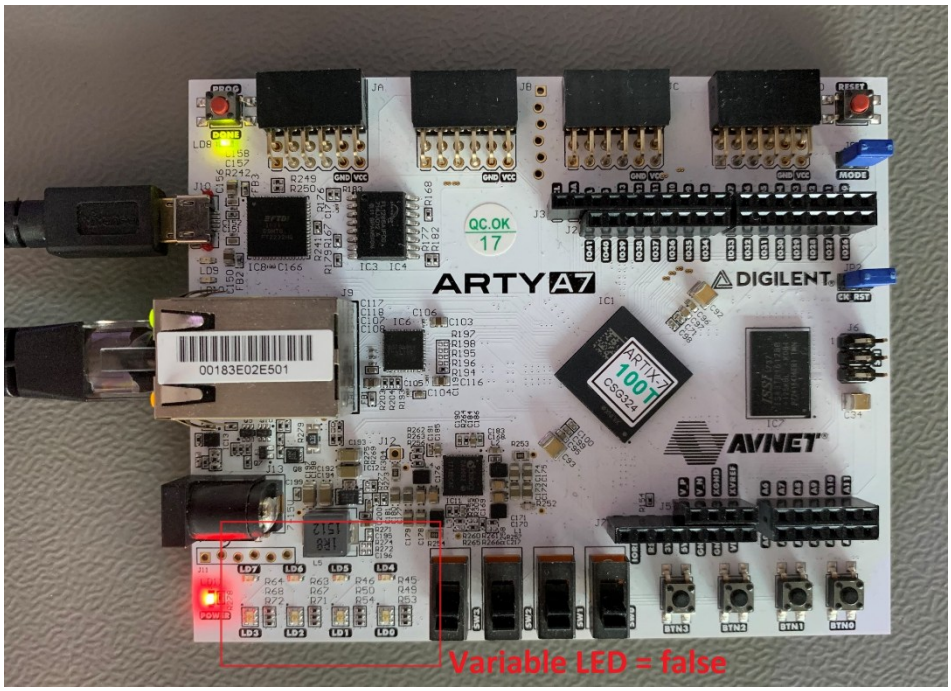
8. After a successful connection in UaExpert the defined object a variable are visible.

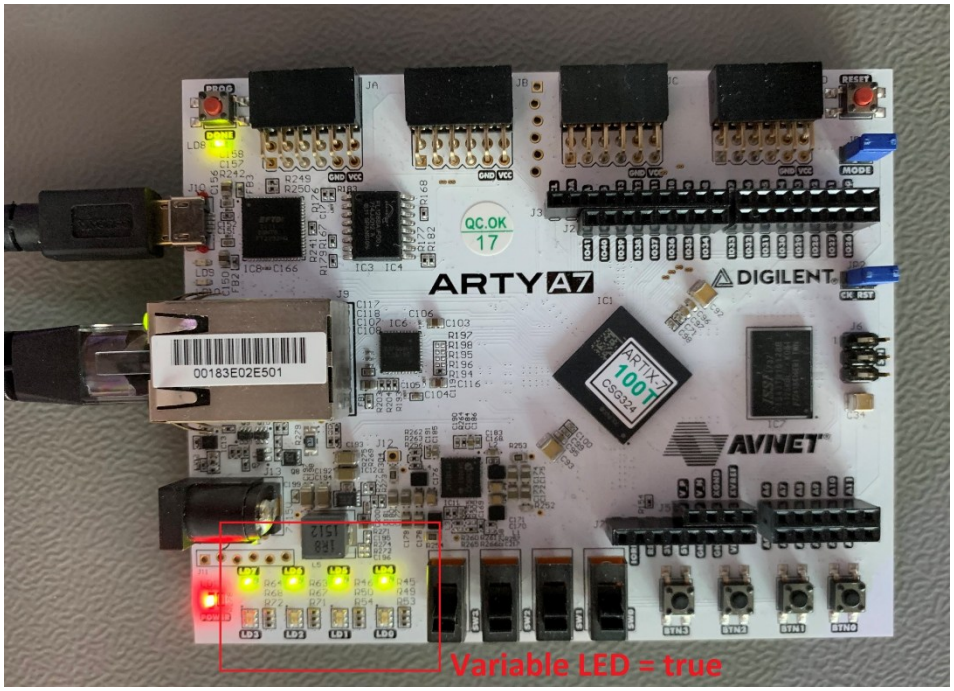


9. The variable LED can now be added to the Data Access View via drag & drop



10. By changing the boolean value of the variable, the LEDs on the ArtyA7 can be switched on /off.


ARTY A7
DIGILENT
AVNET
00183E02E501
Variable LED = false


ARTY A7
DIGILENT
AVNET
00183E02E501
Variable LED = true

Summary

To get an open62541 server running on a MicroBlaze softcore following adjustments are needed:

- Add the defines in the lwip BSP for lwipopts.h:

```
#define LWIP_COMPAT_SOCKETS 0
#define LWIP_SOCKET 1
#define LWIP_DNS 1
#define SO_REUSE 1
#define LWIP_TIMEVAL_PRIVATE 0
```

- Adjust the BSP settings for lwip:

```
api_mode = socket_API
open62541_impl = true
```

- Adjust the BSP settings for FreeRTOS:

```
Check_for_stack_overflow = 1
total_heap_size = 2097152
```

- Adjust CMake options for open62541:

```
UA_ENABLE_AMALGAMATION = ON
UA_ENABLE_HARDENING = OFF
UA_ARCH_EXTRA_INCLUDES = <path to microblaze/include>
UA_ARCH_REMOVE_FLAGS =    -Wpedantic -Wno-static-in-inline
-Wredundant-decls
CMAKE_C_FLAGS =  -Wno-error=format= -mlittle-endian
-DconfigUSE_PORT_OPTIMISED_TASK_SELECTION=0
-DconfigAPPLICATION_ALLOCATED_HEAP=3
-DUA_ARCHITECTURE_FREERTOSLWIP
UA_LOGLEVEL = 100 (optional for debugging)
```

- Generate a linker script with at least: 1MB heap and 1MB stack
- Adjust the C/C++ build settings / include sources/libraries
- Define the thread stack size to 4096
- Adjust the buffer size of the server config:

```
config->networkLayers->localConnectionConfig.recvBufferSize =
32768;
config->networkLayers->localConnectionConfig.sendBufferSize =
32768;
config->networkLayers->localConnectionConfig.maxMessageSize =
32768;
```