AN 961: IO Module Design Example for OPC UA

for Intel® MAX® 10 Devices

Contents

About the IO Module Design Example for Intel® MAX® 10 Devices for OPC UA

The design demonstrates the usability of a low-power FPGA as an IO module, providing compute at the edge. The design shows how FPGAs implement different hardware modules such as triple-speed Ethernet controllers, digital inputs, digital outputs, and Nios II soft processors.

The design is a low-cost IO module that can be part of an industrial network using OPC unified architecture (UA) communication over Ethernet. The design shows you can add small devices such as Intel® MAX® 10 FPGAs to an industrial network to perform basic but necessary tasks like IO supervision and monitoring. For more information on OPC UA, refer to the OPC Foundation website

The design provides an interface to the IO using an OPC UA client and server model and OPC UA PubSub. The design implements all this hardware in the FPGA allowing rapid prototyping and application of changes in the field.

The design's open-source software with FreeRTOS manages the running and scheduling, making the application scalable, reliable, and deterministic.

Figure 1. Basic Setup

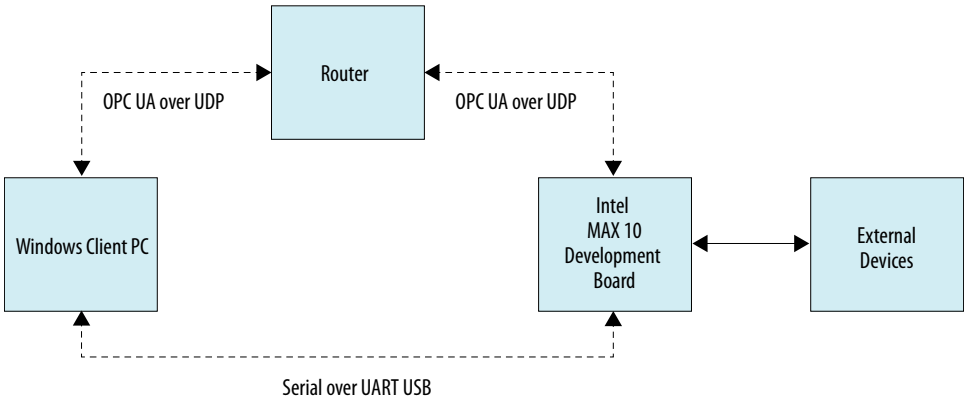
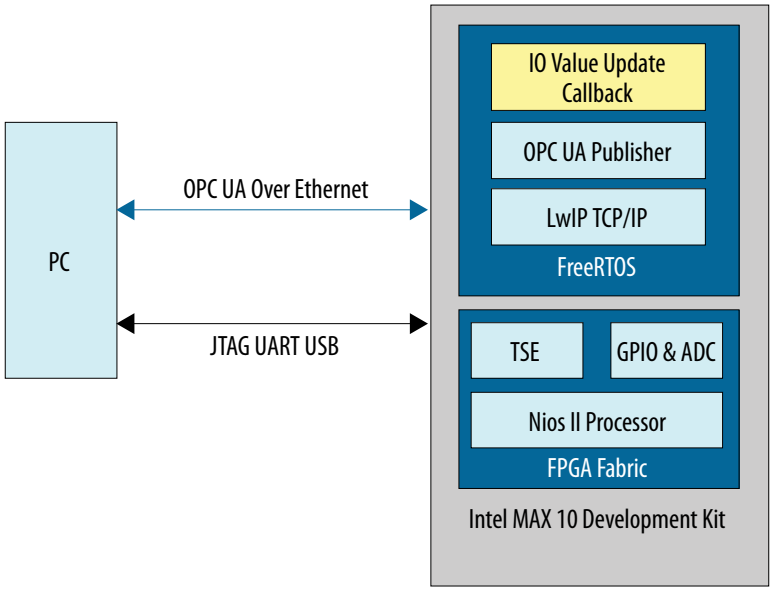


Figure 2. Block diagram

The hardware and software allow you to implement small devices that perform a specific task within an Industry 4.0 network. Industry 4.0 allows workload consolidation at the edge and real-time connectivity between the industrial.



The design:

- Demonstrates industrial networking standards OPC UA Client-Server, OPC UA PubSub, and IO on a real LAN with many affordable and application-specific devices.
- Provides the TCP/IP services required to connect the device to a LAN by a lightweight version of TCP/IP stack known as lightweight IP (lwIP), compatible with FreeRTOS.
- Implements the high-level user application, either OPC UA Client-Server or OPC UA PubSub, using open62541 in amalgamation mode.
- Targets the Intel MAX 10 development board.
- Is compatible with FreeRTOS and lwIP. For more information on FreeRTOS, refer to the FreeRTOS website.

Features

On the Intel MAX 10 Development Board, the design offers:

- An Ethernet interface for onboard IO with an OPC UA server.
- Support for both Ethernet ports.
- Up to 1000 Mbps transfer rate.
- 11 ms minimum OPC UA publishing cycle time (four static and four dynamic datasets).
- ADC: 64 sequencing channels with sample rate of 1 MHz with internal reference 2.5 V.
- Onboard LEDs, push buttons, and on-chip temperature sensing diode.
- Total thermal power dissipation of 754.52 mW.

OPC UA Nodes

OPC UA includes several nodes that correspond to a specific IO on the Intel MAX 10 development board. Two additional nodes measure the delta time between IO data updates and the time difference between each publishing cycle (cycle time). Each of these nodes has a corresponding update function that you can call every publishing cycle either on the read or write to the OPC UA dataset.

Table 1. OPC UA nodes

The configured nodes represent the selected IO and read or write from the corresponding IO pins. Each node is configured with a name, ID, type, access level, and parent node. You can use each of these as filters to select which nodes on the network to input. The design ignores anything not matching.

Name	Node ID	Type	Access	Update Function
LED value	LED-value	`UA_Int32`	Write	`setLED()`
Push button array	Pb-matrix	`UA_Double[4]`	Read	`getPB()`
Onboard temp	Onboard-temp	`UA_Int32`	Read	`getTemp()`
Current time	Current-time-data-source	`UA_DateTime`	Read	`updateCurrentTime()`
Publishing cycle time	Cycle-time	`UA_Int32`	Read	`readCycleTime()`

Related Information

- OPC Foundation
- Intel MAX 10 Development Kit User Guide
- FreeRTOS website
- open 62541 website
- Intel MAX 10 Analog to Digital Converter User Guide
- Using the On-Die Temperature Sensor in MAX10 User Guide

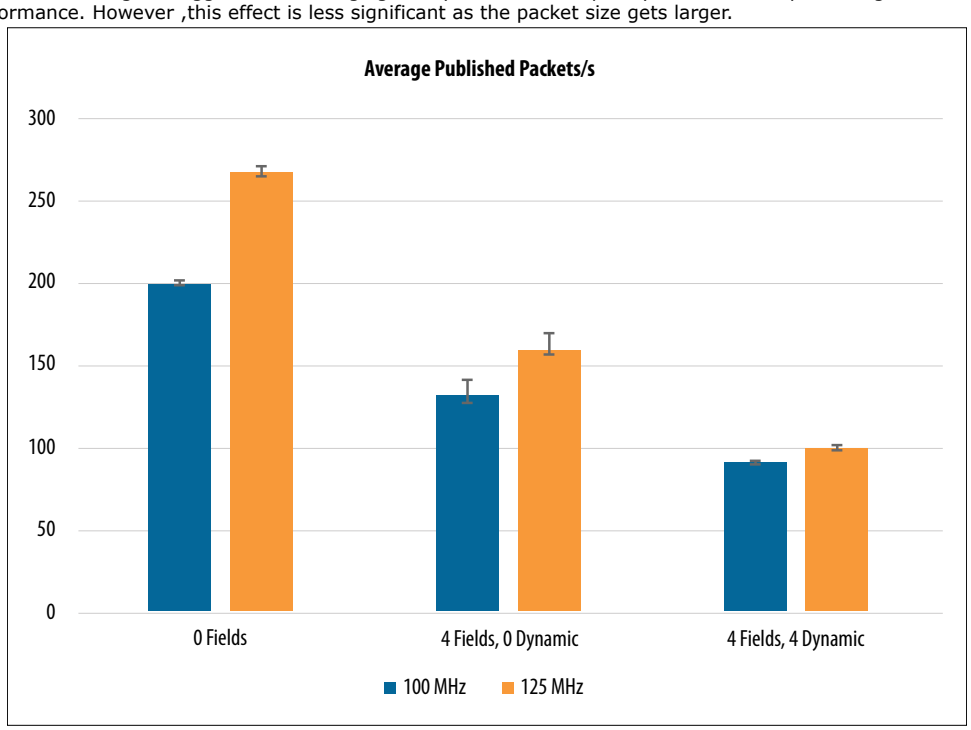
Performance and Resource Utilization for the IO Module Design Example for OPC UA

Intel compared the performance of the IO module with two different Nios II clocks: 100 MHz and 125 MHz, for minimum publishing cycle times. The publishing cycle time determines the rate at which the publisher creates network messages.

Intel used the Wireshark packet inspection tool on a client PC to monitor the incoming packets (filtered to the IP address of the IO module). The tool outputs the average packets per second, which is equivalent to the packet frequency. Intel compared different variations of frame contents: one with an empty dataset, one with 4 fields in the dataset that don't update (static), and one where the four dataset values update on every publishing cycle (dynamic).

Figure 3. Publishing Performance for 100 MHz and 125 MHz

The data in the figure suggests that changing the system clock frequency increases the publishing performance. However ,this effect is less significant as the packet size gets larger.

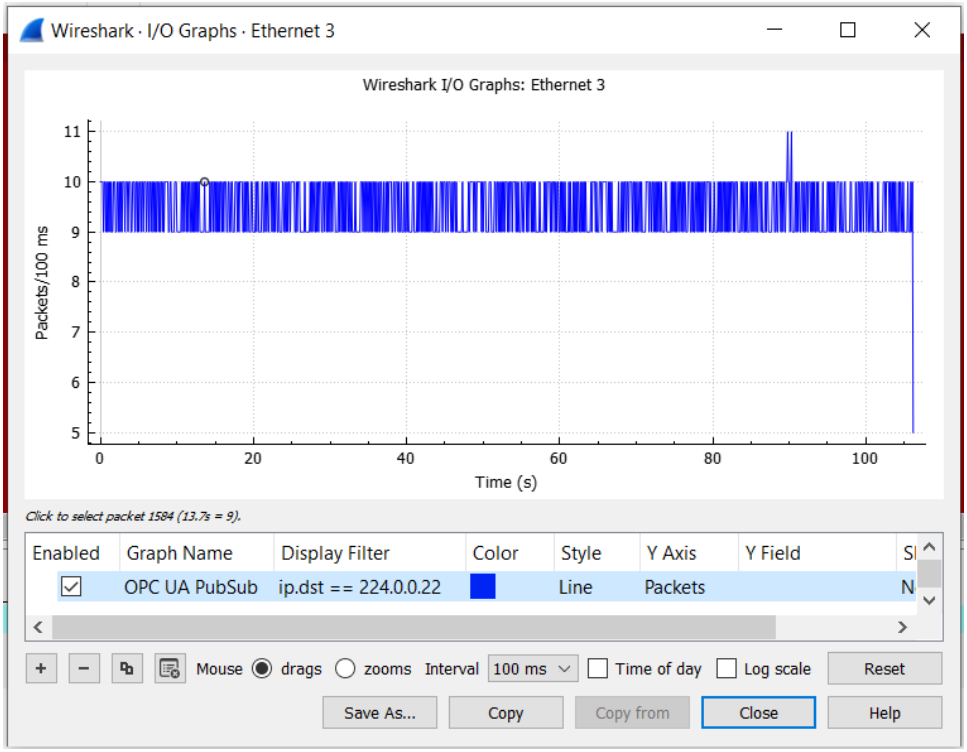


The time difference between static fields and dynamic fields is 3.39 ms (40 packets/s @ 100 MHz) and 3.72 ms (58.9 packets/s @ 125 MHz), which you can attribute to the time it takes to read the IO and update the OPC UA datasets.

The time difference between 4 (static) fields and 0 fields is 2.60 ms (68.3 packets/s @ 100 MHz) and 2.55 ms (108.2 packets/s @ 125 MHz), which is a result of the difference in size of the dataset. A larger dataset takes more time to read from memory and write to the published frames.

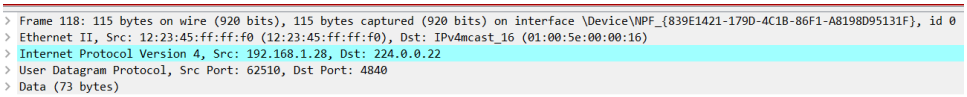
Figure 4. Wireshark Input Displaying Publisher Performance of 91 packets/second.

The figure shows an example output of the Wireshark tool with the system running at 100 MHz and 11 ms publishing cycle time.



This tool also shows that each packet (or frame) has a size of 63 bytes, which is dependent on the complexity of the informational model in the publisher.

Figure 5. Example Packet Details Displayed in Wireshark



Intel optimizes the FPGA resource utilization for timing constraints, port sizes, and memory allocation. The design uses 35% of the FPGA logic, 62% of the on-chip M9K memory blocks, and 20% of the potential IO pins.

Figure 6. Resources

Fitter Resource Usage Summary	
Resource	Usage
Total logic elements	17,532 / 49,760 (35 %)
-- Combinational with no register	5989
-- Register only	4086
-- Combinational with a register	7457
Logic element usage by number of LUT inputs	
-- 4 input functions	7640
-- 3 input functions	3256
-- <=2 input functions	2550
-- Register only	4086
Logic elements by mode	
-- normal mode	11933
-- arithmetic mode	1513
Total registers*	11,655 / 51,509 (23 %)
-- Dedicated logic registers	11,543 / 49,760 (23 %)
-- I/O registers	112 / 1,749 (6 %)
Total LABs: partially or completely used	1,380 / 3,110 (44 %)
Virtual pins	0
I/O pins	72 / 360 (20 %)
-- Clock pins	2 / 8 (25 %)
-- Dedicated input pins	1 / 1 (100 %)
M9Ks	113 / 182 (62 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	1 / 2 (50 %)
Total block memory bits	698,738 / 1,677,312 (42 %)
Total block memory implementation bits	1,041,408 / 1,677,312 (62 %)
Embedded Multiplier 9-bit elements	6 / 288 (2 %)
PLLs	3 / 4 (75 %)
Global signals	20
-- Global clocks	20 / 20 (100 %)
JTAGs	1 / 1 (100 %)
CRC blocks	0 / 1 (0 %)
Remote update blocks	1 / 1 (100 %)
Oscillator blocks	0 / 1 (0 %)
Impedance control blocks	1 / 1 (100 %)
Average interconnect usage (total/H/V)	10.6% / 10.7% / 10.6%
Peak interconnect usage (total/H/V)	36.1% / 36.0% / 36.2%
Maximum fan-out	5636
Highest non-global fan-out	1242
Total fan-out	95103
Average fan-out	3.23

Related Information

Wireshark website

691266 | 2021.10.30

Send Feedback

Getting Started with the IO Module Design Example for OPC UA

Hardware and Software Requirements for the IO Module Design Example for OPC UA

Software

- Intel Quartus® Prime 17.0 standard edition including Intel FPGA Nios II Embedded Design Suite (EDS) v17.0
- FreeRTOS (v10.3.1)
- Modified LwIP TCP/IP stack (v2.1.2)
- Open62541 OPC UA amalgamation (release 1.2.2)
- Unified Automation UAExpert (1.5.1)
- CMake for Windows (3.20.0-rc2) – optional

Hardware

- Intel MAX 10 Development Kit
- MiniUSB for device programming
- Ethernet cable
- Local network router

Downloading the IO Module Design Example for OPC UA

1. In the Intel Design Store, download the relevant design `.par` file for the Intel MAX 10 development kit.
2. In the Quartus II software, click **File ➤ New Project Wizard**.
3. Click **Next**.
4. Enter the path for project working directory and enter a project name.
5. Click **Next**.
6. Select **Project Template**.
7. Click **Next**.
8. Click **Install the design templates**.
9. Browse to select the `.par` file for the design and browse to the destination directory where you want to install it.
10. Click **OK** on the design template installation message.

11. Select the design example.
12. Click **Next**.
13. Click **Finish**.

    The Intel Quartus Prime software expands the archive and sets up the project, which may take some time.

Related Information

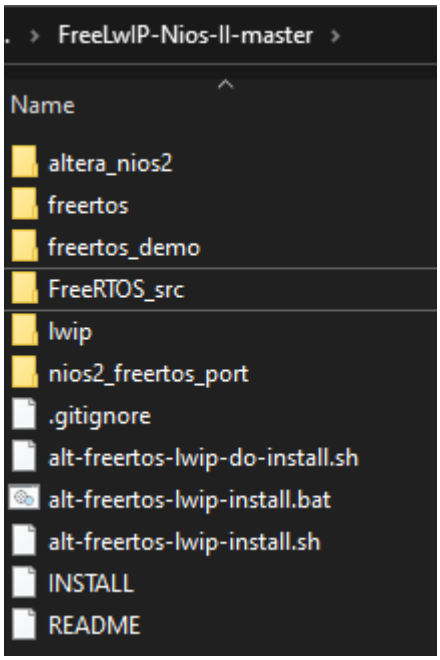
IO Module Design Example

Installing FreeRTOS and LwIP Templates on Nios Eclipse IDE

To correctly generate the `.bsp` file for the IO Module Design Example, run the script that installs FreeRTOS, LwIP, and Intel driver files in the Nios Eclipse IDE directory. Engineering Spirit B.V. provides the script and NetTimeLogic GmbH updates it (refer to fork history of the NetTime Logic Repository).

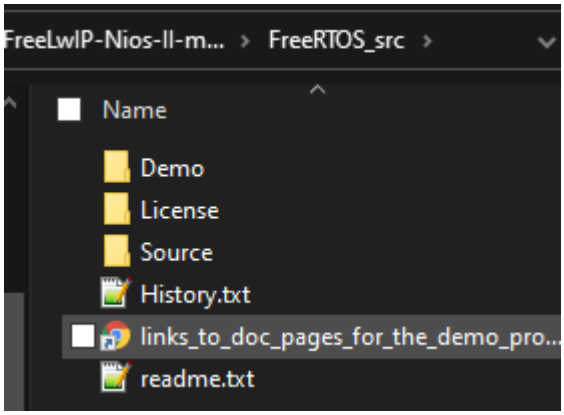
1. Download and extract the NetTimeLogic GmbH installation files:
   - SHA: `4b738bf09a93fcb51a66a684de6e350b78555cf9`
2. Create a new directory: `/FreeLwIP-Nios-II-master/FreeRTOS_src/`.

Figure 7. FreeLwIP-Nios II Master Installation Directory



3. Download FreeRTOS and extract contents of `/FreeRTOSvX.X.X/FreeRTOS` to the newly created directory `/FreeRTOS_src`.

Figure 8. FreeRTOS_src directory with extracted FreeRTOS source code



4. Open `/FreeLwIP-Nios-II/lwip/FreeRTOS/src/arch/altera_avalon_tse.c` and modify line 1277:

   ```
   for (phyadd = 0x00; phyadd < 0x20; phyadd++)
   ```

5. Open `/FreeLwIP-Nios-II/freertos/FreeRTOS/src/tse_ethernet_phys.c` and modify lines 41, 44, 47 to replace `TSE_SYSTEM_EXT_MEM_NO_SHARED_FIFO(X,X,X,X,0,X) input 5 (0)` with `&marvell_cfg_rgmii`.

6. Run `alt-freertos-lwip-install.bat` as indicated in the readme file provided in the Nettime Logic repository.

7. To test theFreeRTOS and lwIP installations, build the FreeRTOS/LwIP template demo:

   a. Open Nios II Software Build Tools for Eclipse.

   b. Click **File ➤ New ➤ Nios II Application and BSP from Template**.

   c. Select from the Template list **FreeRTOS - LwIP Demo**. This template is available in the template list after the successful execution of the `alt-freertos-lwip-install.bat` script.

   d. Select `./<Quartus_Project>/nios.sopcinfo` in **SOPC Information File name** and provide a project name.

   e. Click on **Finish...**

Related Information

- Download NetTime Logic FreeLwIP for Nios II Processor
- Download FreeRTOS

Generating an open62541 OPC UA Amalgamation (optional)

Open62541 provides the alternative for building the library in different modes. In an embedded software project, such as the IO Module Design Example, you can build an amalgamation of the library into a single source and header file (open62541.c and open62541.h) using CMake for Windows and Eclipse.

1. Download and extract open62541.
2. Open `<project directory>/opcua_nios/SdkNios.cmake` and change the UA_ARCH_EXTRA_INCLUDES to point to the project BSP directory.
3. Open CMake, navigate to `open62541` directory for source.
4. Create a `/BUILD_/` directory for the binaries.
5. Click on **Configure**.
6. Select **Eclipse CDT4 - Unix Makefiles** as a generator and **Specify toolchain for cross-compiling**.
7. Specify the toolchain file: `<project directory>/opcua_nios/SdkNios.cmake` and click **Finish**.
8. Click **Generate**.
9. Open Nios II Software Build Tools for Eclipse.
10. Right click **BSP ➤ Import ➤ Existing Projects into Workspace**, browse to `../BUILD_/` directory.
11. Click **Window ➤ Show View ➤ Other ➤ Make ➤ Make Target**.
12. Click **Make Target Window ➤ <open62541> ➤ @BUILD_ ➤ all**.

    Ignore any fail messages.
13. Open `open62541.c` and find the line: `include <task.h>`.
14. In the line above add: `include <FreeRTOS.h>` and save.
15. Copy the generated `open62541.c` and `.h` files to the FreeRTOS project.

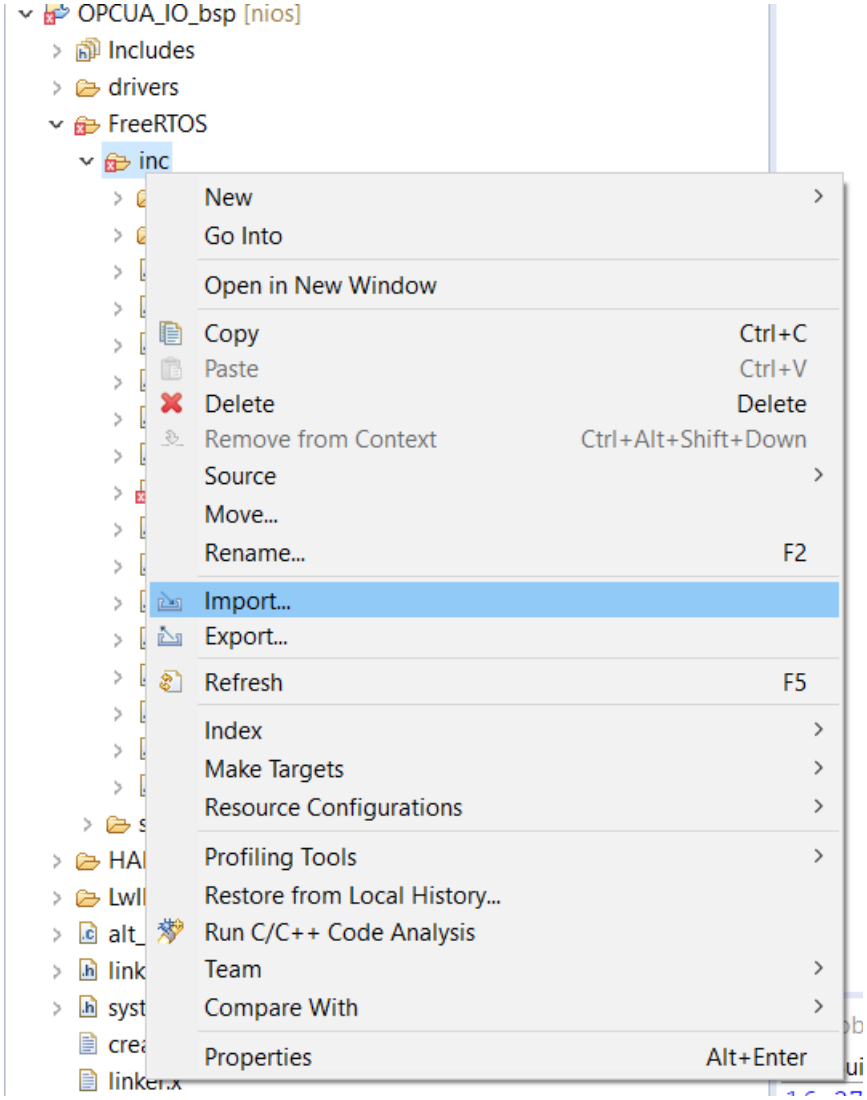
Related Information

Download open62541

Preparing the IO Module Project for Build

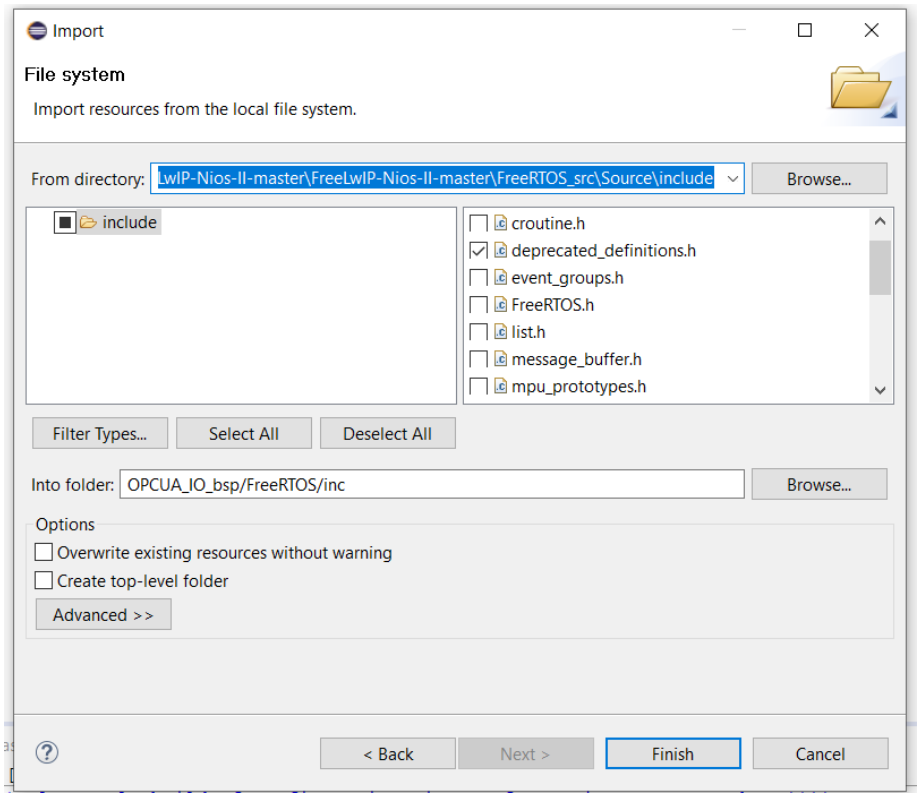
1. Navigate to `FreeRTOS_BSP/FreeRTOS/inc/` in the project pane.
2. Right click **BSP ➤ Import ➤ File System**.

Figure 9. Importing source files to the BSP



3. Navigate to `<source>/FreeLwIP-Nios-II/FreeRTOS_src/Source/include`.
4. Import `deprecated_definitions.h` and `stack_macros.h`.

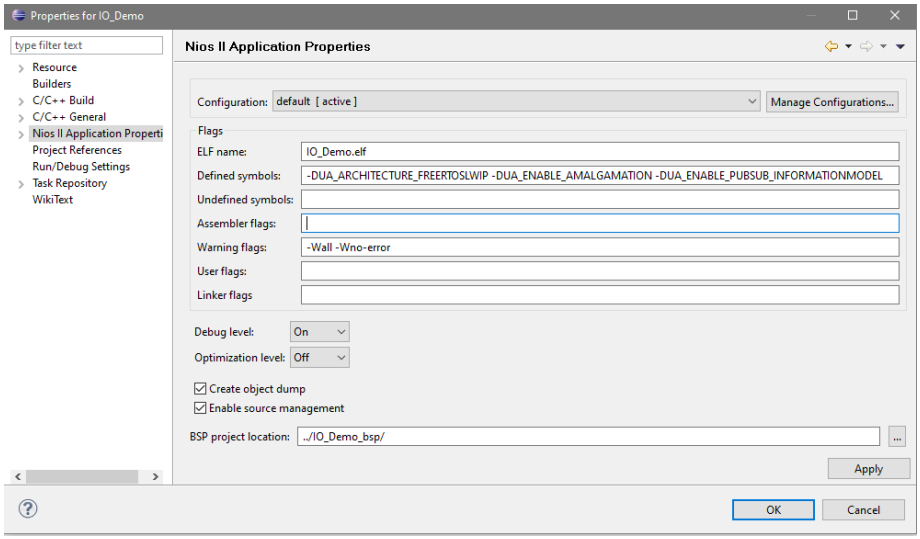
Figure 10. Importing source files to the BSP 2



5. Right click **BSP ➤ Nios II ➤ BSP Editor...**.
6. Go to **Settings**, set **max file descriptors** = 16.
7. Save and generate the BSP.
8. Right-click **Project ➤ Properties ➤ Nios II application properties ➤ .**:

```
Defined symbols = “-DUA_ARCHITECTURE_FREERTOSLWIP -DUA_ENABLE_AMALGAMATION -
DUA_ENABLE_PUBSUB_INFORMATIONMODEL”
Warning flags = “-Wall -Wno-error”
```

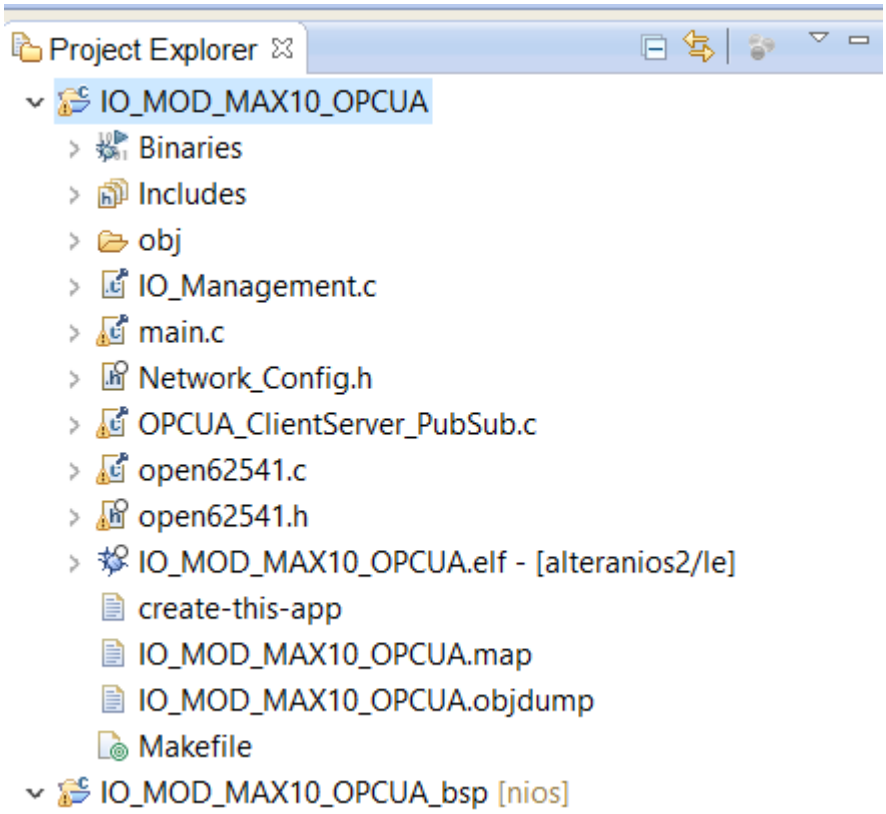
Figure 11. Project Properties



Adding IO Module Software

1. Delete all source (`.c` or `.h`) files in the project.
2. Right-click **Project ➤ Import ➤ General ➤ File System**.
   a. Browse to `/Quartus_Project/software/`.
   b. Import source (`.c`) files.
   c. Browse to `/Quartus_Project/software/open62541`.
   d. Import `open62541.c` and `open62541.h`.
3. Optionally, modify `Network_Conf.h` to apply custom IP settings for the server.
4. Build the project.

Figure 12. Project Files



Running the IO Module Design Example

1. Connect an Ethernet cable to the Intel MAX 10 port A (bottom port) to the same router the windows PC is connected to, and a USB cable between the Intel MAX 10 device and the PC.
2. Program the Intel MAX 10 with ".sof" hardware output file generated during compilation:
   a. Open the Intel Quartus Programmer.
   b. Select **USB blaster**.
   c. Select **Auto Detect** and mark **10M50**.
   d. Select **Change File** and navigate to the `.sof` within `/output_files/` in your Quartus project directory.
   e. Turn on **Program** and **Configure**.
   f. Click **Start** to configure the FPGA on the development board using the `.sof`. The programmer shows **Successful** in the top-right corner.
3. Run the FreeRTOS project in Nios II Eclipse:
   a. Right-click the project.
   b. Click **Run as ➤ Nios II Hardware ➤ ..**

Figure 13. Running a project in Eclipse

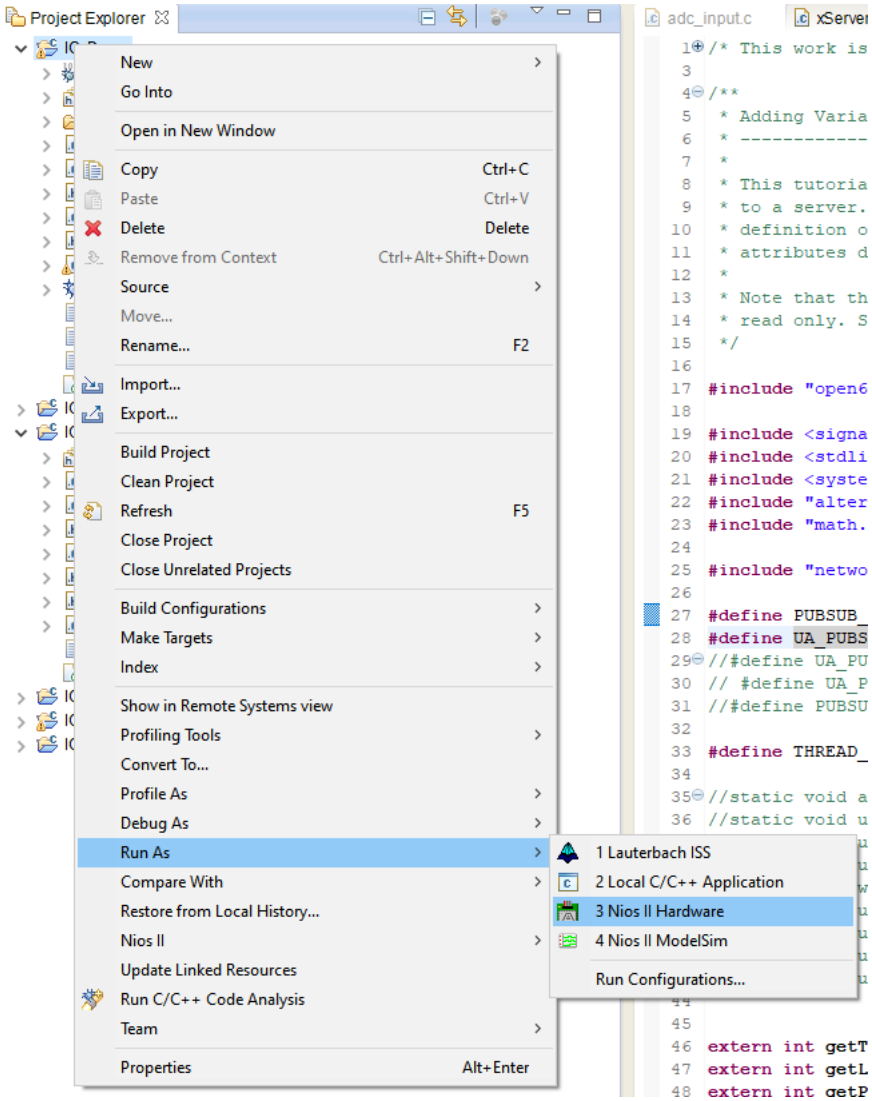
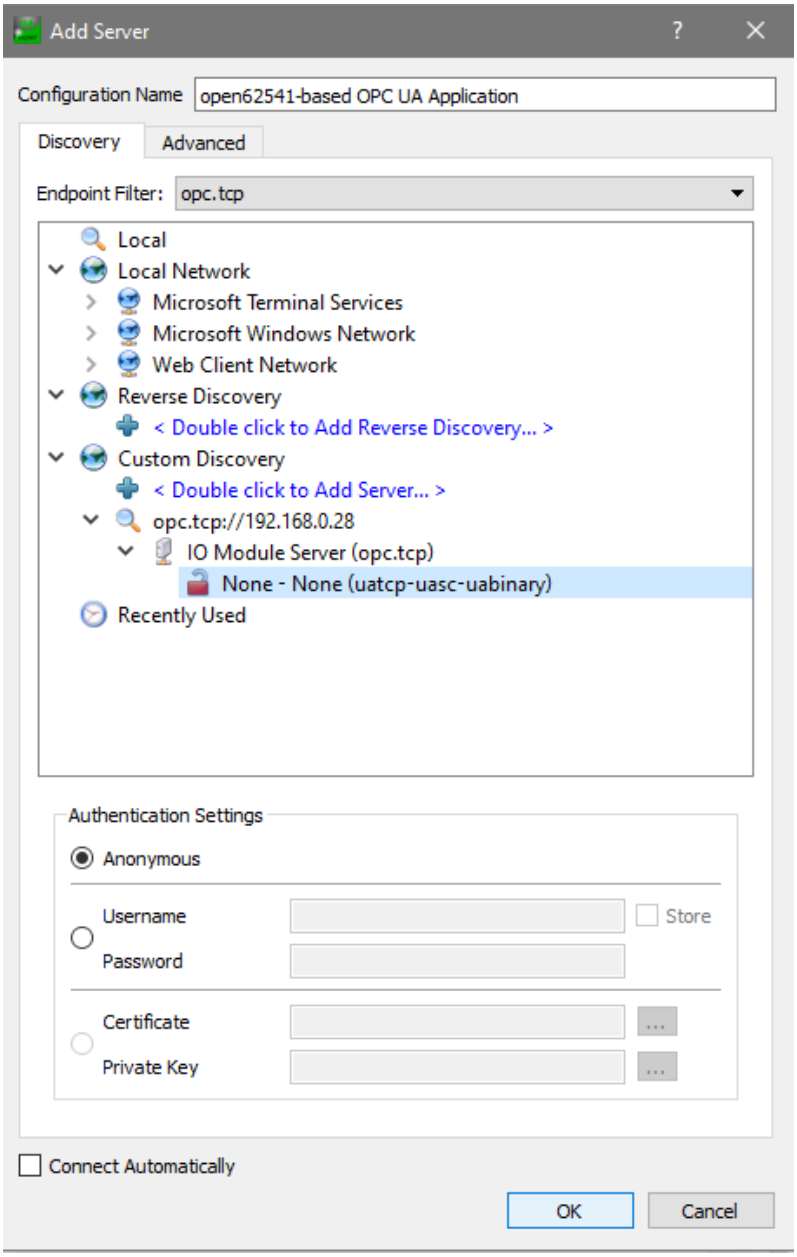


Figure 14. Nios II Eclipse terminal expected printout

```
[ethernet] Acquired IP address via DHCP client for interface: e0
[ethernet] IP address : 192.168.0.28
[ethernet] Subnet     : 255.255.255.0
[ethernet] Gateway    : 192.168.0.1
[lwip_eth0] up
--------- Init Done ---------
[1970-01-01 00:00:13.949 (UTC+0000)] warn/server   AccessControl: Unconfigured AccessControl. Users have all permissions.
[1970-01-01 00:00:13.953 (UTC+0000)] info/server   AccessControl: Anonymous login is enabled
[1970-01-01 00:00:13.957 (UTC+0000)] warn/server   Username/Password configured, but no encrypting SecurityPolicy. This can leak credentials on the network.
[1970-01-01 00:00:13.963 (UTC+0000)] warn/userland AcceptAll Certificate Verification. Any remote certificate will be accepted.
[1970-01-01 00:00:14.101 (UTC+0000)] info/network  TCP network layer listening on opc.tcp://192.168.0.28:4840/
```

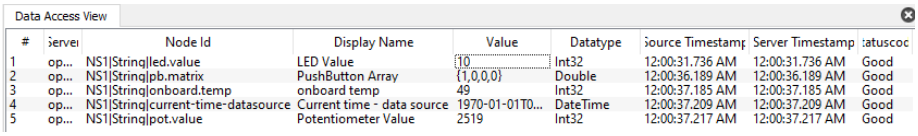
4. Open UAExpert on the Windows PC:
   a. Add server.
   b. Set custom discovery: `opc.tcp://<device IP>`.

Figure 15. Adding the IO module in UA Expert.



5. Connect to the server and add nodes from the left pane (drag and drop).

Figure 16. UAExpert client with example data



6. In the **Value** column try to modify values for the LEDs and press the push buttons, move the potentiometer, and observe the changes in the board and in the UAExpert GUI.
7. Turn off the design:
   a. Close UAexpert connection.
   b. Stop the program in Nios II Eclipse.
   c. Turn off and disconnect the Intel MAX 10 Development Board.

Achieving Timing Closure on a Design Example

Intel Quartus Prime may not achieve full timing closure when it first compiles and fits this designs.

Intel Quartus Prime assume worst-case timing parameters over a wide range of temperature, which is good practice for a commercial design. If you run this design at room temperature, it is unlikely that real timing violations occur. To achieve full timing closure with Intel Quartus Prime, you may include additional pipeline registers. However, the place-and-route process is sensitive to an initial seed value and the result of different seeds is not easy to predict. Before optimizing the design, try the seed sweep function with many different seed values in case the design immediately fits.

1. In Intel Quartus Prime, select **Tools ➤ Launch Design Space Explorer II**.

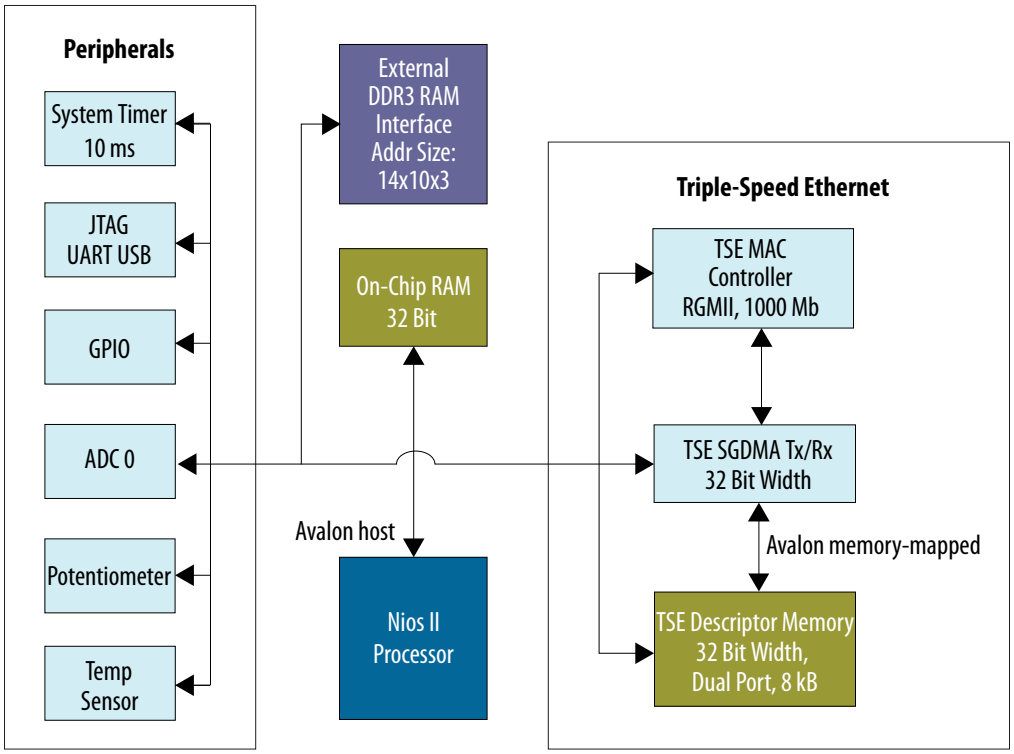
   Design space Explorer opens in a separate window.
2. For a basic seed sweep use the following settings:
   a. In **Setup**, select **Local**.
   b. In **Exploration**, select **Design exploration, exploration mode: Seed Sweep Only, create 10 seeds**.
   c. Click **Start** to run.
   d. Calculate the seed number from the exploration point name. It is the *_number* plus 1. In this example, the best seed is 3(dse1_3) + 1 = 4.
3. Select **Assignments ➤ Settings**.
4. In complier settings, click **Advanced settings (Fitter)...**.
5. Update seed value **Fitter Initial Placement Seed 4**.

Functional Description

The IO Module Design Example for OPC UA targets the Intel MAX 10 development board as the IO module device. This device allows development using on-chip peripherals, CPU, and triple-speed Ethernet controllers.

The design is based on an Intel Ethernet design from the Intel Simple Socket Server design example. It has a top-level subsystem and two subsystems: one peripheral subsystem, and an Ethernet subsystem. This system design uses an Avalon Memory-Mapped interface, a DDR3 RAM, and Nios-II soft processor IP to provide an interface between peripherals, and a triple-speed Ethernet controller. The peripherals include an onboard GPIO with LEDs and pushbuttons, and on-chip IP including timer, ADC, and temperature sensing diode. The design can run the OPC UA PubSub or Client-Server software. The triple-speed Ethernet IP uses the RGMII standard to run at 1000 Mb/s transfer speeds. A clock control subsystem, which determines the suitable clock based on the Ethernet connection type, selects varying clock times. The ADC uses 64 sequencing channels with a sample rate of 1 MHz set by default and an internal reference of 2.5 V.

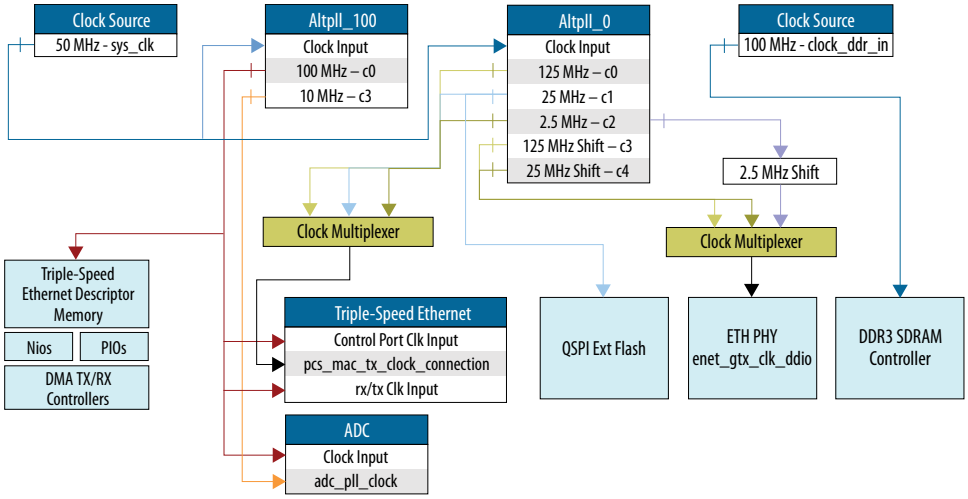
Figure 17. IO Module Design Example Hardware Block Diagram



Clocks

The main system clock source is 50 MHz and is driven by the Intel MAX 10 50 MHz clock pin. This system clock drives the two PLLs: `altpll_0` for TSE RGMII timing and `altpll_100` for the 100 MHz clock supplied to the IO and Nios II processor. An additional 100 MHz clock pin drives the external DDR3 RAM Interface to meet timing constraints.

Figure 18. System clock and PLL dependencies



ADC Sampling

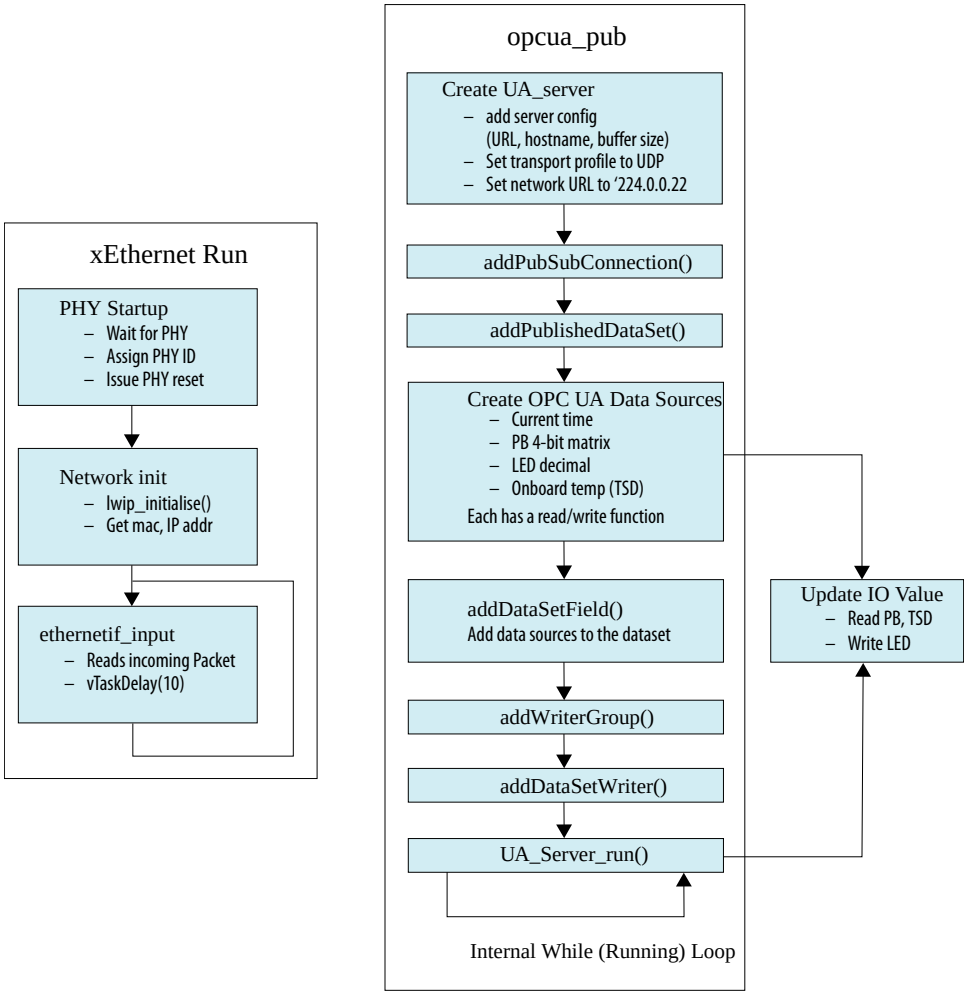
The IO module has ADC sampling using an on-chip temperature sensing diode. The Nios II processor reads these analog inputs interfacing them with an ADC IP that is programmed into the FPGA fabric. The modular ADC IP acts as a soft controller for the two ADC hard IP blocks on the FPGA. The first ADC has 10 channels: 9 input channels and a dedicated channel for the temperature sensing diode. The ADC sample rate automatically changes from 1 megasamples/s on normal channel input mode to 50 kilosamples/s on temperature sensing diode channel mode, which samples at a 64-sample running average method.

The ADC sequencer specifies which channel is sampled at which time and a certain number of samples. The sequencer is built into the modular ADC IP. The handling is based on channel selection, sample rate, ADC clock, and the reference voltage.

Software

The design software runs on the Nios II processor and is primarily programmed in embedded C using the Nios-II Eclipse IDE. The project includes a `.bsp` file, which contains all the FPGA driver files, FreeRTOS source files, and LwIP source files. The software includes the main thread and two sub-threads: `xEthernetRun` for network initialization and `opcua_pub` for the OPC UA publishing.

Figure 19. OPC UA thread flowchart for a PubSub Publisher



To read to or write from GPIOs such as LEDs and push buttons, the design calls a driver command with the corresponding pin. The ADC requires a setup command to begin the sequencer and then you can read as IO. The design converts the input for the temperature sensing diode into a Celsius value using a lookup table provided in the temperature sensing diode design example.

Related Information

- Simple Socket Server Design Example
- OPC Foundation
- FreeRTOS website
- open 62541 website
- Intel MAX 10 Analog to Digital Converter User Guide
- Using the On-Die Temperature Sensor in MAX10 User Guide

Known Issues with the IO Module Design Example

IP address conflicts

A common error causing the UAExpert client to be unable to find the server. To fix the error:

1. Disconnect Intel MAX 10 device from the network Open a Command Prompt and ping the Intel MAX 10 IP address (default: 192.168.0.28),

You should see no response, otherwise:

1. Run `arp -a` command, to show all connected IP addresses.
2. Using the `arp` response, select a new IP address for the Intel MAX 10 device that is not taken and make the modifications in `network_conf.h`,

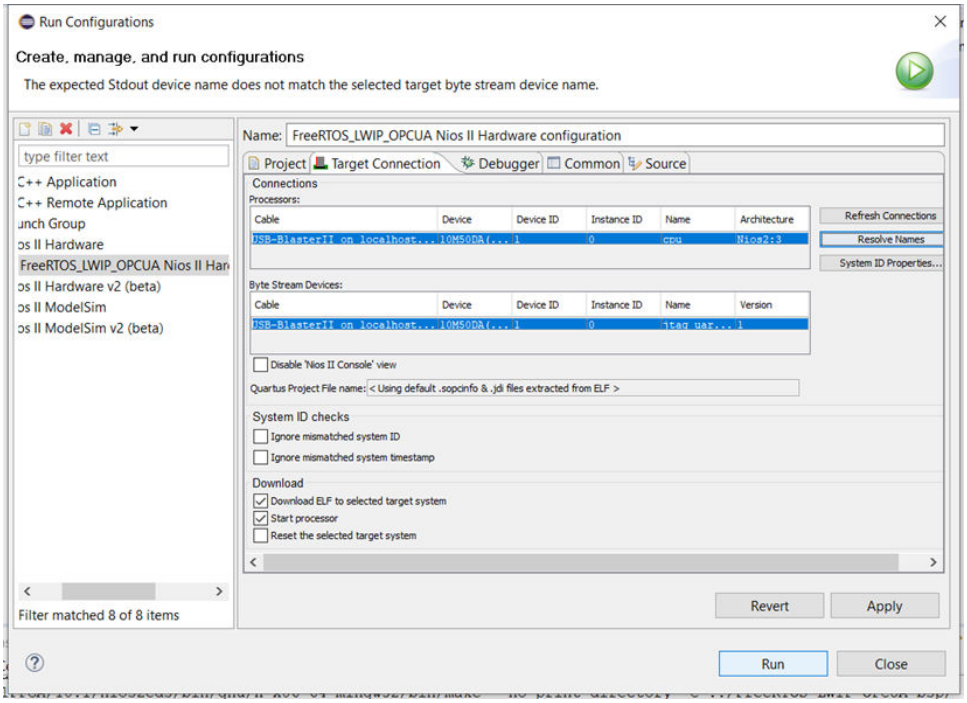
VPN

This design requires you to connect the client PC to the same local network as the Intel MAX 10 development board. Any VPN connections interfere and prevent a connection between the devices.

Errors in Eclipse

The `no run configuration is found` error is usually caused by the run configuration not updating when the Intel MAX® 10 device is connected over USB. To fix this issue, click on **Refresh Connection** and then **Run ➤ Run Configurations ➤ ,**.

Figure 20. Eclipse Run Configurations



Document Revision History for AN 961: IO Module Design Example for Intel MAX 10 Devices for OPC UA

Document Version	Changes
2021.10.30	Initial release.