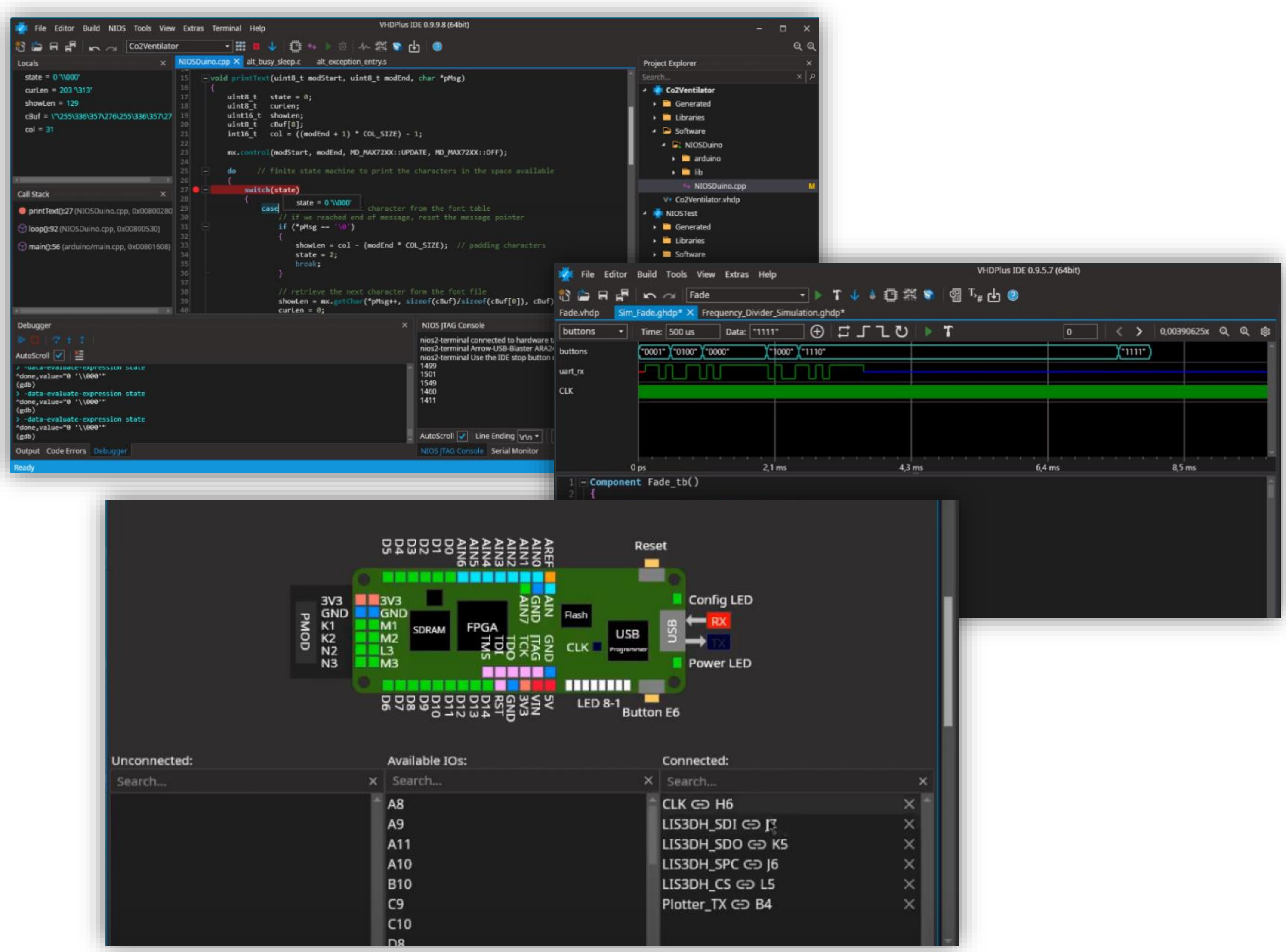
VHDPlus

MAX1000

VHDPlus Overview Lab



Software and hardware requirements to complete all exercises
Software Requirements: Quartus® Prime Lite or Standard Edition version 18.0 or 18.1 & **VHDPlus IDE**
Hardware Requirements: ARROW MAX1000 Board

1. Introduction

This tutorial provides comprehensive information to help you understand how to use the VHDPlus IDE to accelerate and simplify your FPGA development. You will learn how to create a working program for your hardware with a few clicks, visualize your received data, program with VHDL and our language VHDP, simulate the code with our assistant and you will learn how to add a customized NIOS processor to your design and program it inside the VHDPlus IDE.

Contents

2. Getting Started

The first objective is to ensure that you have all the necessary hardware items and software installed so that the lab can be completed successfully. Below is a list of items required to complete this lab:

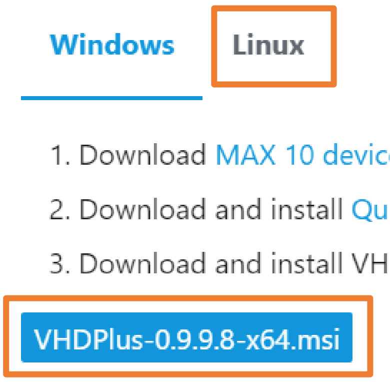
- MAX1000 Board (10M08SAU169C8G)
- USB Cable
- Quartus Prime 18.0 Lite was used for this lab. Previous/newer versions should work (If no Quartus Prime is installed, refer to MAX1000 User Guide for instructions)
- Installed Arrow USB Drivers (If not, refer to MAX1000 User Guide for instructions)
- Personal computer or laptop running 64-bit Linux / Windows 7 or later with at least an Intel i3 core (or equivalent), 4GB RAM and 12 GB of free hard disk space

Finally, the most important part is the VHDPlus IDE. In the following you learn how to download, install and setup the IDE.

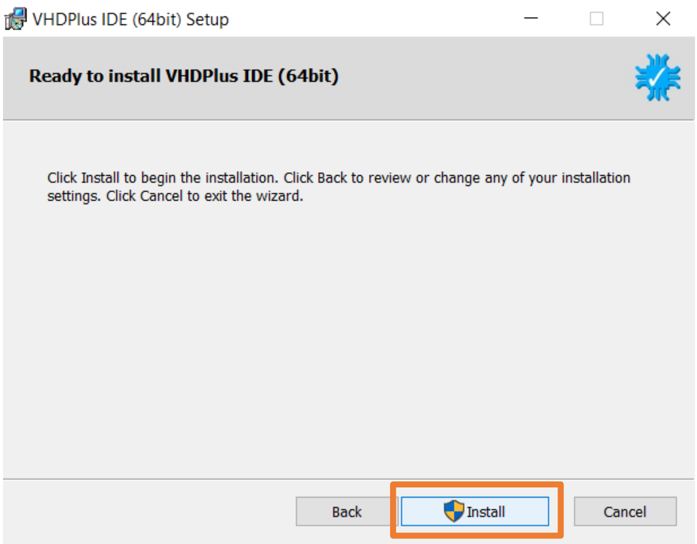
2.1 Install the VHDPlus IDE

2.1.1 Open the Website “vhdp.de” and click on “Download”

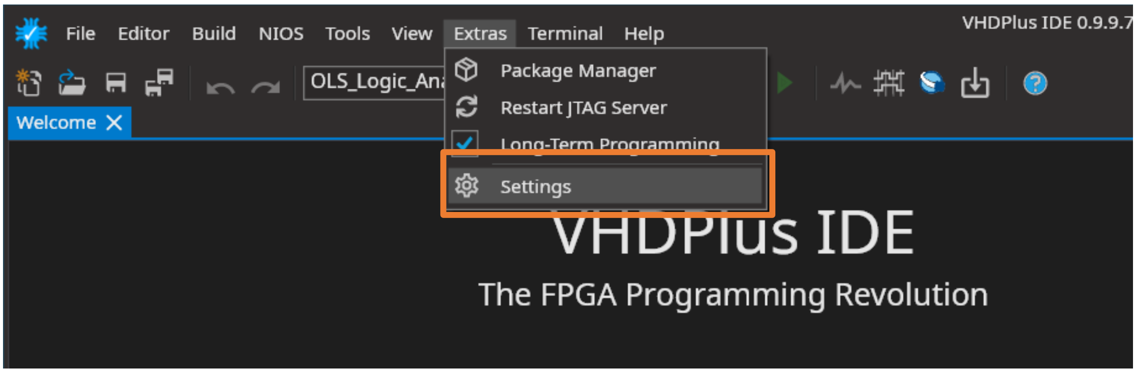
2.1.2 Download the Installer for Windows or click on “Linux” to see how to install the IDE on your distribution



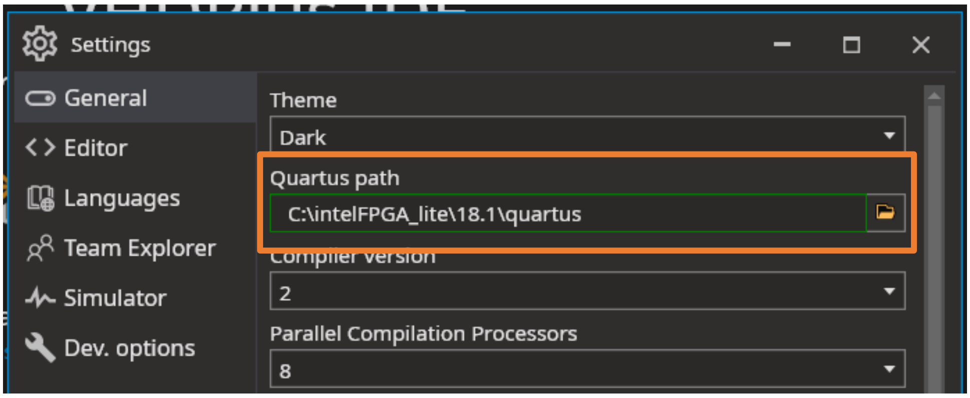
2.1.3 Install the IDE and open the Program



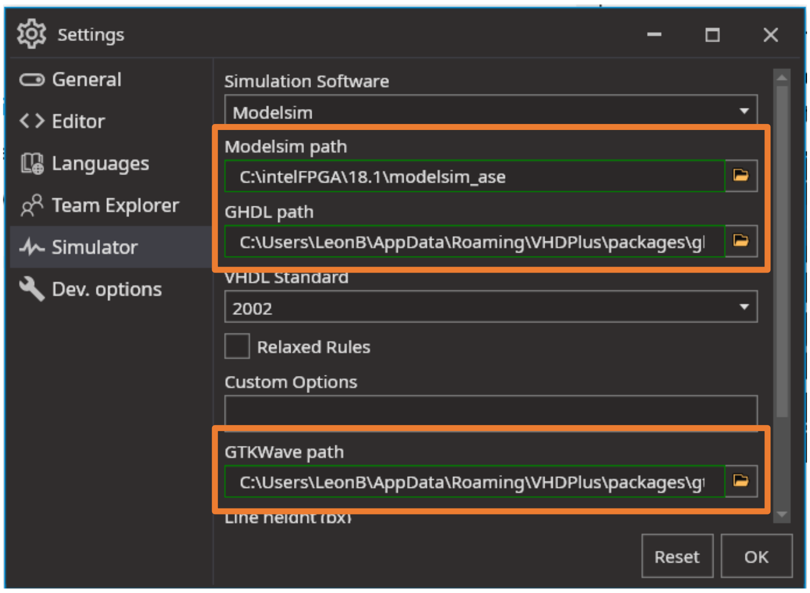
2.1.4 Open settings



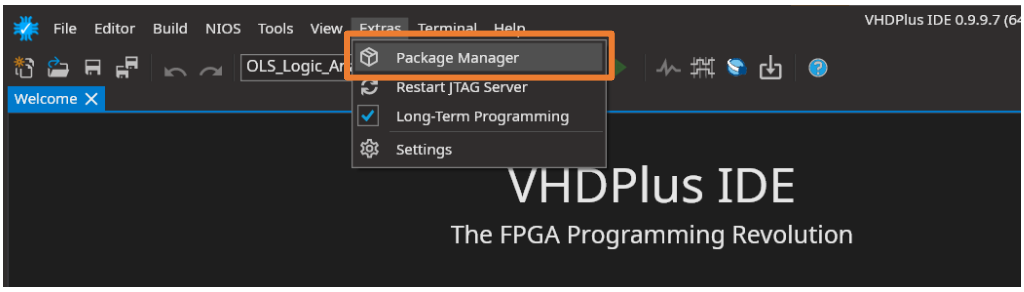
2.1.5 Connect the IDE with Quartus. You must select the “quartus” folder in that you find the application. When the boundary is green of the “Quartus path” text box, the correct path is selected



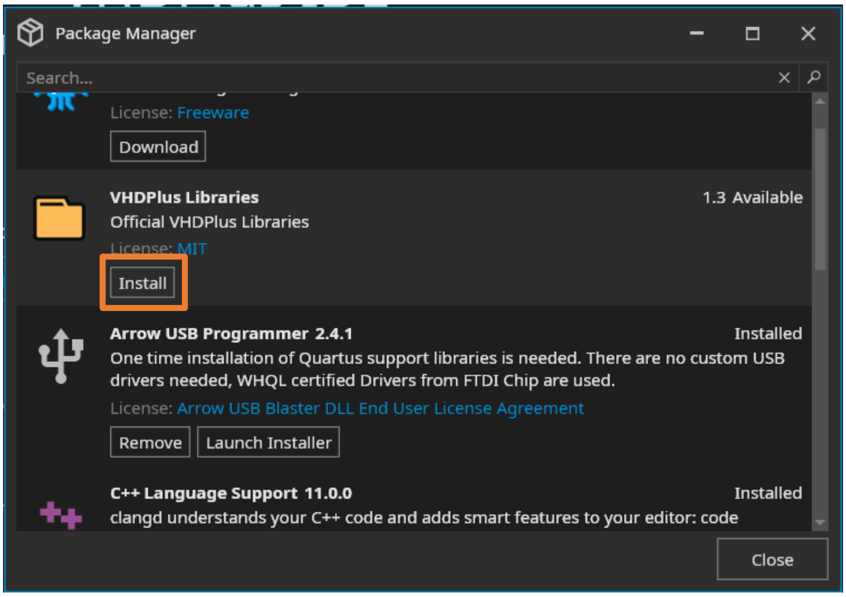
2.1.6 (Optional) Connect your simulation tool. You can either use Modelsim or GHDL



2.1.7 Install the needed packages with the Package Manager



2.1.8 For this lab you need the VHDPlus Libraries, GHDL, GTKWave and the VHDL Language Support, but you can also install other packages



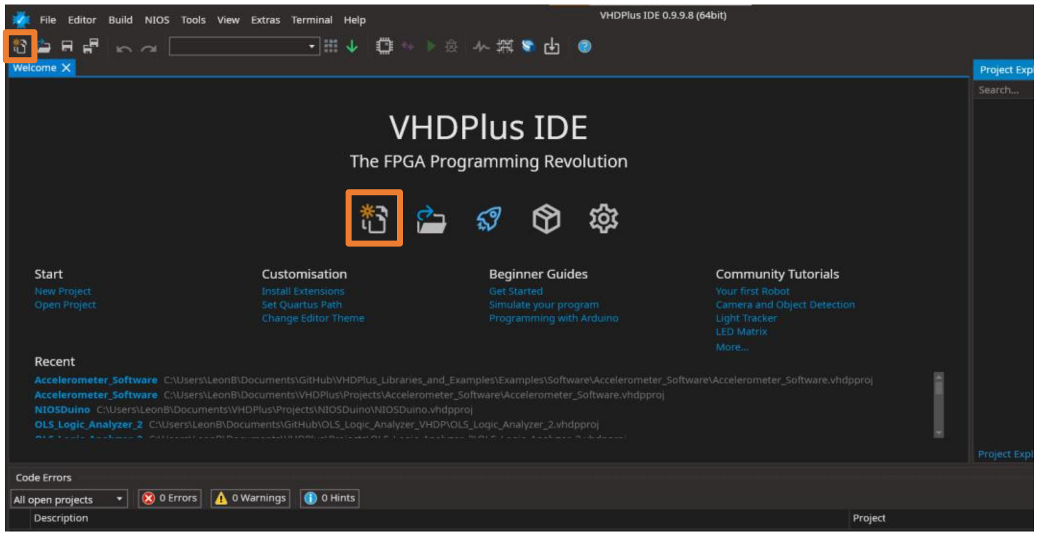
3. Accelerometer

In this part you learn how to use the Accelerometer of your MAX1000 board. You learn how to create a project in the VHDPlus IDE, use the built-in libraries and visualize the received data with the Serial Plotter.

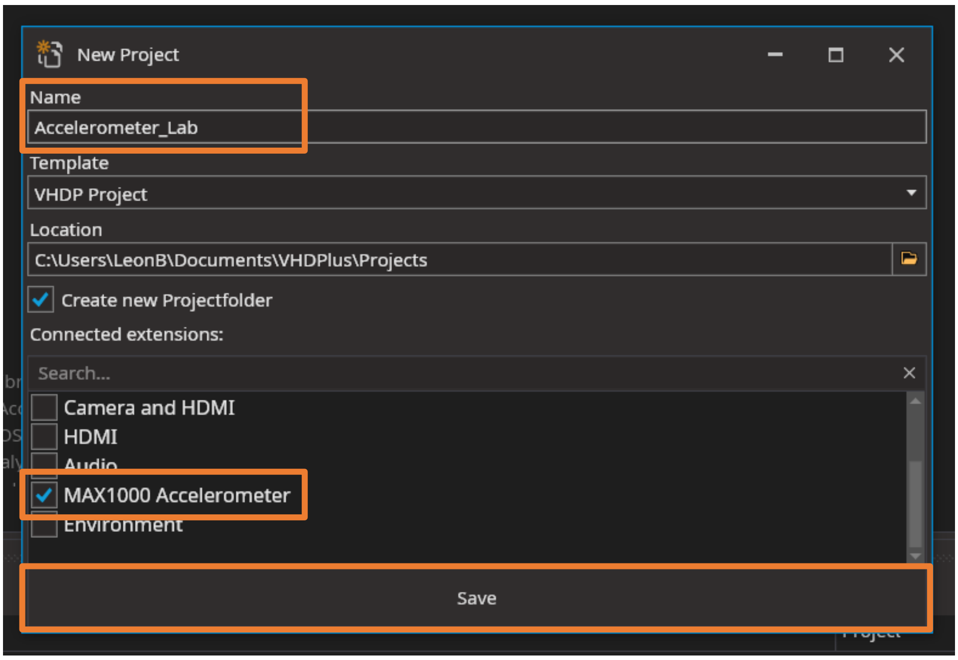
Alternatively to this part 3 you can watch and follow this Video Tutorial!

3.1 Create the project

3.1.1 Create a new project



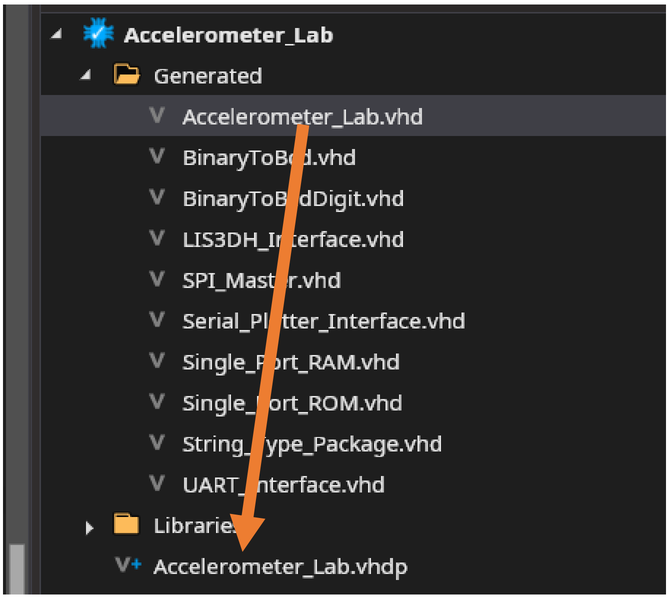
3.1.2 Name the project, select the accelerometer as hardware and create the project



3.1.3 You can look at the code and see what is created:

The code is written in our language VHDP that is based on VHDL but simplifies a lot of aspects. You can find more information here.

(Optional) When you don’t want to use VHDP, you can click on *Build > Create VHDP Code* and drag the “Accelerometer_Lab.vhd” file from the “Generated” folder to your project folder and delete the “Accelerometer_Lab.vhdp” file



3.1.3.1 The UART TX output for plotting the data and the SPI interface pins are declared

```
Main
(

    Plotter_TX : OUT STD_LOGIC; --UART output for plotter

    LIS3DH_SDI : OUT STD_LOGIC; --Accelerometer data in
    LIS3DH_SDO : IN  STD_LOGIC; --Accelerometer data out
    LIS3DH_SPC : OUT STD_LOGIC; --Accelerometer clock
    LIS3DH_CS  : OUT STD_LOGIC; --Accelerometer chip select

    |
)
{
```

3.1.3.2 You can set the UART baud rate and the chars for the data name and unit

```
    --MAX1000 Accelerometer-----------------------------------------------------------------

    CONSTANT Plotter_Baud_Rate   : NATURAL := 9600;
    CONSTANT Plotter_UnitChars   : NATURAL := 1;   --Set to max. number of chars of the unit an
    CONSTANT Plotter_NameChars   : NATURAL := 15; --Set to max. number of chars of the graph n
```

3.1.3.3 The current value and data name are set and sent with the plotter interface

```
    Process ()
    {
        Thread
        {
            Plotter_Value      <= LIS3DH_X_Val;
            Plotter_Graph_Name <= s"Accelerometer X";
            Plotter_Send       <= '1';
            While(Plotter_Busy = '0'){}
            Plotter_Send       <= '0';
            While(Plotter_Busy = '1'){}
```

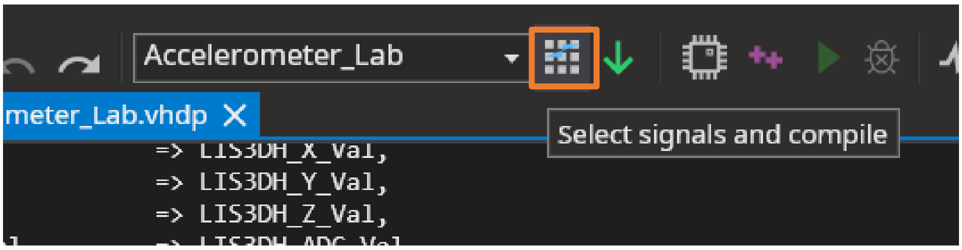
3.1.3.4 The accelerometer interface is configured with a 400Hz data rate and the ADC input is enabled. But the ADC is not used in this example

```
    SIGNAL LIS3DH_X_Val          : INTEGER   range -2048 to 2047 := 0;
    SIGNAL LIS3DH_Y_Val          : INTEGER   range -2048 to 2047 := 0;
    SIGNAL LIS3DH_Z_Val          : INTEGER   range -2048 to 2047 := 0;
    SIGNAL LIS3DH_ADC_Val        : INTEGER   range -512 to 511 := 0;
    NewComponent LIS3DH_Interface
    (
        Data_Rate       => 7,
        Use_ADC         => '1',
        Use_Temp        => '0',

        X_Val           => LIS3DH_X_Val,
        Y_Val           => LIS3DH_Y_Val,
        Z_Val           => LIS3DH_Z_Val,
        ADC_Val         => LIS3DH_ADC_Val,
        SDI             => LIS3DH_SDI,
        SDO             => LIS3DH_SDO,
        SPC             => LIS3DH_SPC,
        CS              => LIS3DH_CS,
    );
```

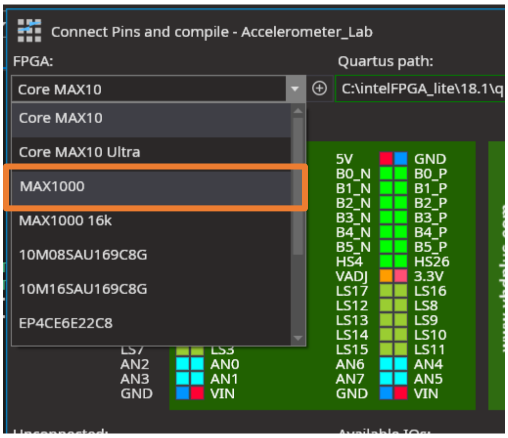
3.1.3.5 The serial plotter interface is used without time or unit input. In the created code only the graph name and the value are set

```
    SIGNAL Plotter_Value         : INTEGER := 0;
    SIGNAL Plotter_Graph_Name    : STD_LOGIC_VECTOR (Plotter_NameChars*8-1 downto 0) := (others => '0');
    SIGNAL Plotter_ValueUnit     : STD_LOGIC_VECTOR (Plotter_UnitChars*8-1 downto 0) := (others => '0');
    SIGNAL Plotter_Send          : STD_LOGIC := '0';
    SIGNAL Plotter_Busy          : STD_LOGIC := '0';
    NewComponent Serial_Plotter_Interface
    (
        UseTime         => false,
        NameChars       => Plotter_NameChars,
        UnitChars       => Plotter_UnitChars,
        Baud_Rate       => Plotter_Baud_Rate,

        Value           => Plotter_Value,
        Graph_Name      => Plotter_Graph_Name,
        ValueUnit       => Plotter_ValueUnit,
        Send            => Plotter_Send,
        Busy            => Plotter_Busy,
        TX              => Plotter_TX,
    );
```

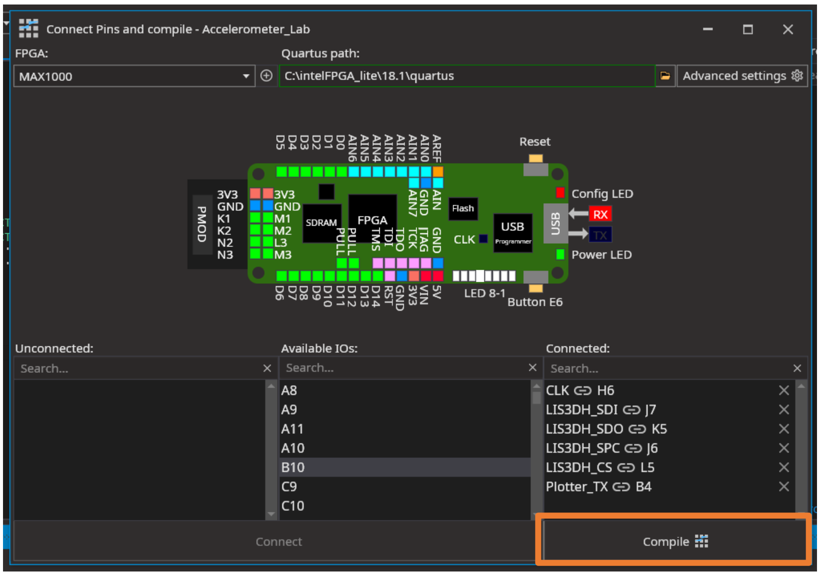
3.1.4 Click on the create button to convert the VHDP code to VHDL and select the pins



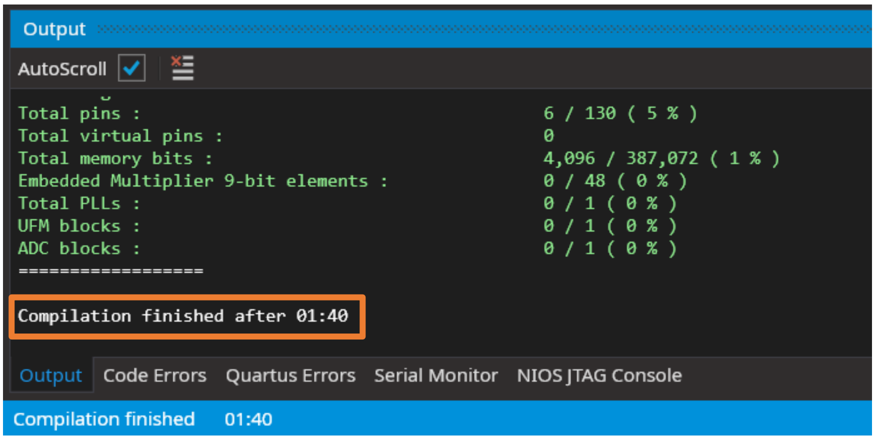
3.1.5 Select the MAX1000 as hardware



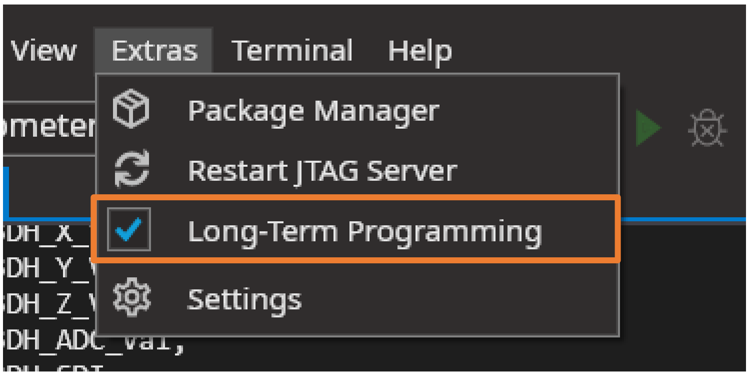
3.1.6 Click on create. The pins of the accelerometer were already assigned in the accelerometer library



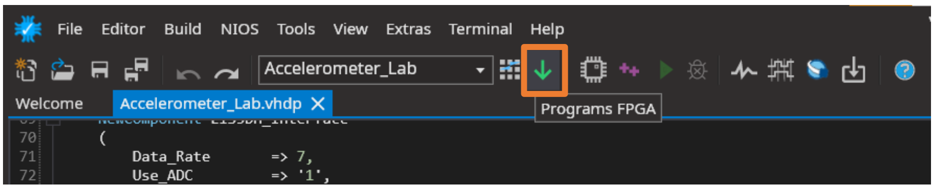
3.1.7 Wait until the compilation is finished



3.1.8 Select long term programming if you want to save the configuration on the FPGA



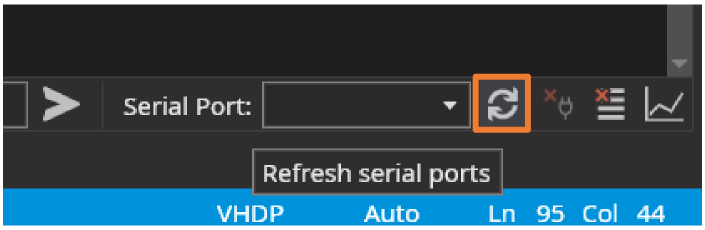
3.1.9 Connect the MAX1000 with an USB cable to your PC and click on the program button



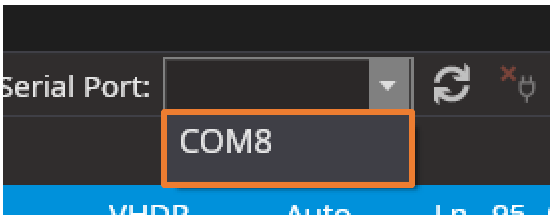
3.2 Try your program

3.2.1 Open the Serial Monitor

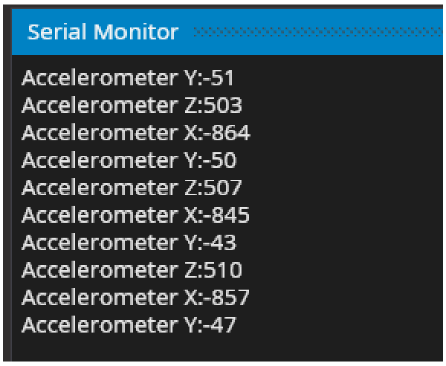
3.2.2 Make sure the baud rate is set to 9600 and refresh the connected devices



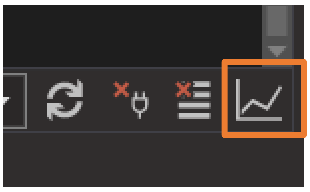
3.2.3 Select you MAX1000 that should appear in the dropdown menu



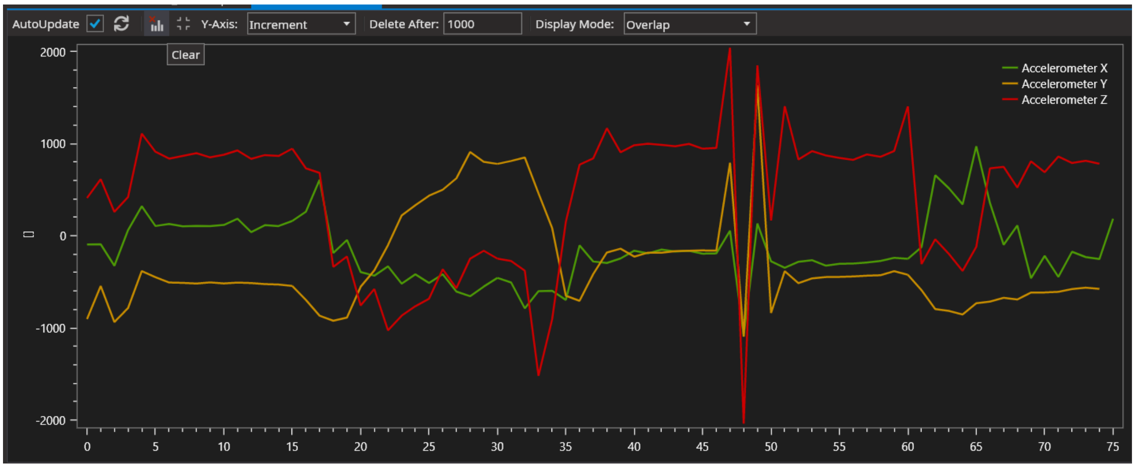
3.2.4 You should already see the data in the serial monitor



3.2.5 Click on the serial plotter button on the right



3.2.6 When you move the MAX100, you should see the values changing

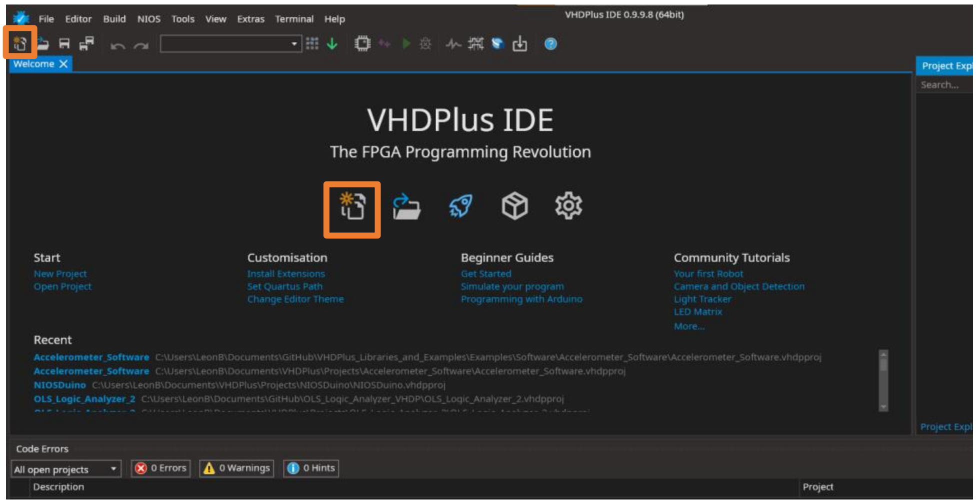


4. VHDL + Simulation

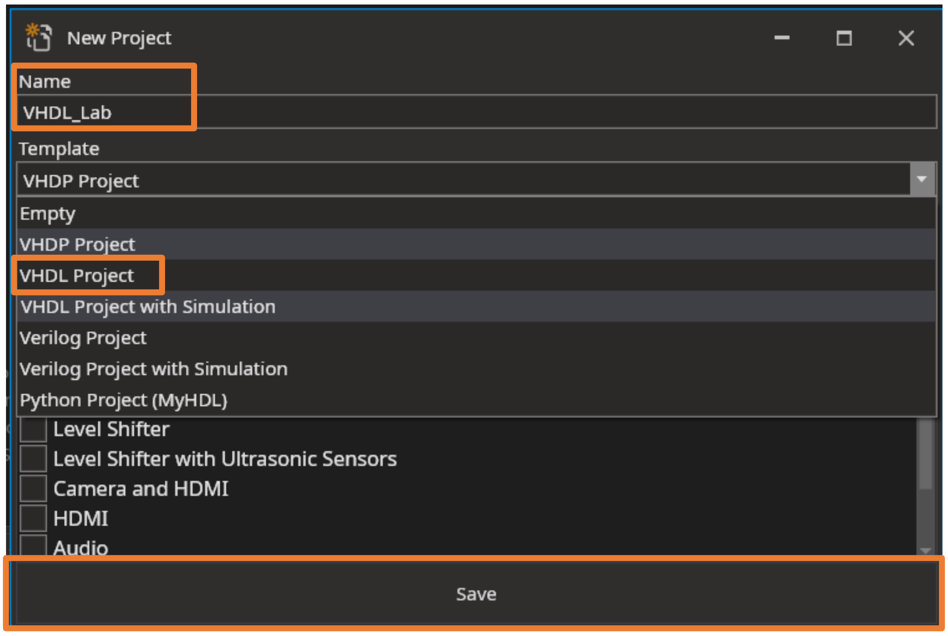
In this part you learn to program with VHDL in the VHDPlus IDE, how to use the simulation assistant and how to use VHDP libraries together with VHDL code. You will use the UART and random number generator library to program a simple random number generator that could be used as dices or a coin flip.

4.1 Create the project

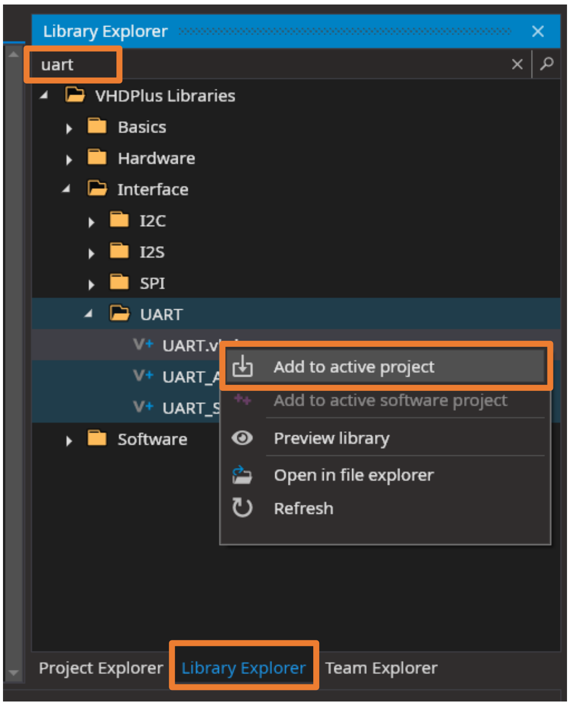
4.1.1 Create a new project



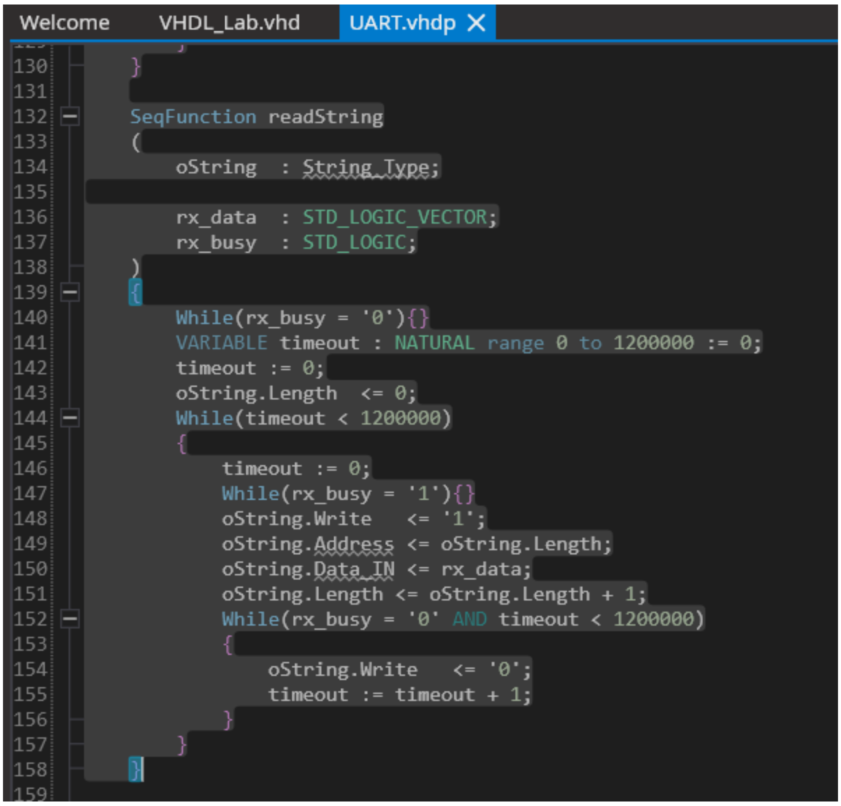
4.1.2 Name the project, select as template a VHDL project and create the project



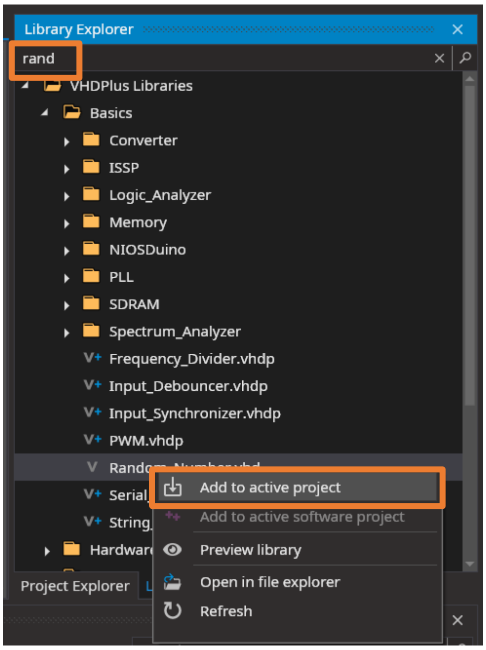
4.1.3 Go to the Library Explorer, search for “UART”, right click the library and add it to your project



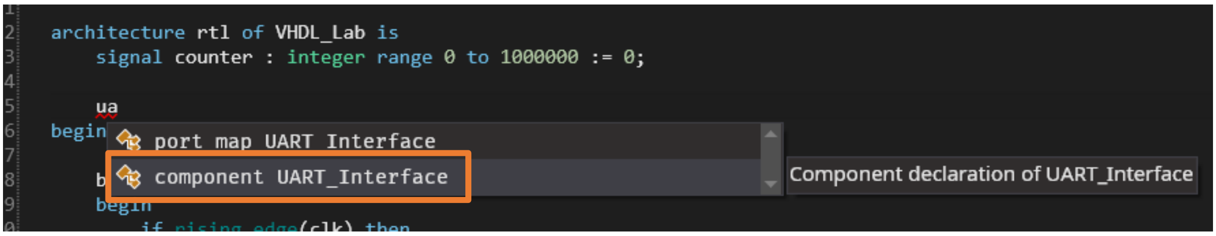
4.1.4 Because we do not use the VHDP string libraries, you get some warnings in the Code Errors tab. Double click one warning and delete the read and printString function in the Libraries/UART.vhdp file

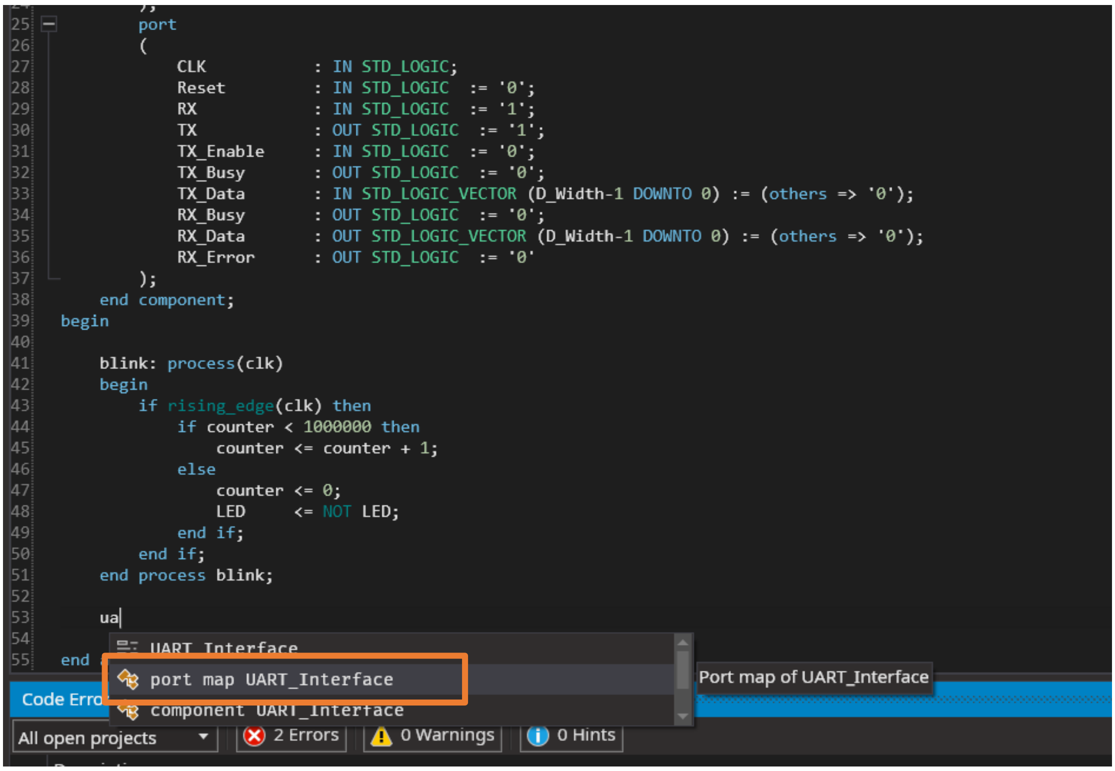


4.1.5 Search for the random number library and it to your project



4.1.6 Add the UART component to your VHDL code





4.1.7 Create and connect the signals

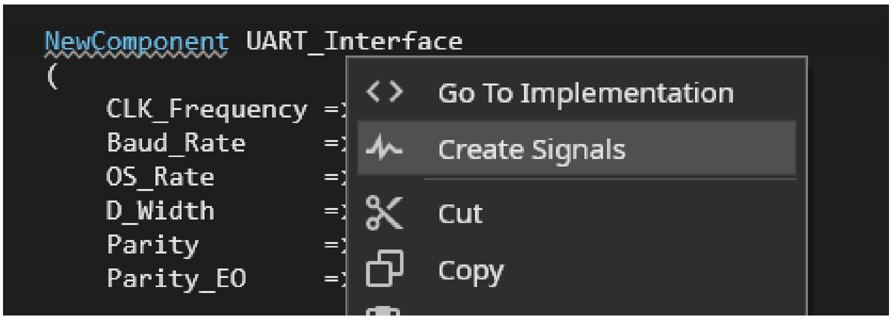
```vhdl
        LED : buffer std_logic := '0';

        RX          : IN STD_LOGIC  := '1';
        TX          : OUT STD_LOGIC  := '1'
    );
end entity VHDL_Lab;

architecture rtl of VHDL_Lab is
    signal counter : integer range 0 to 1000000 := 0;

    signal Reset         : STD_LOGIC  := '0';
    signal TX_Enable     : STD_LOGIC  := '0';
    signal TX_Busy       : STD_LOGIC  := '0';
    signal TX_Data       : STD_LOGIC_VECTOR (7 DOWNTO 0) := (others => '0');
    signal RX_Busy       : STD_LOGIC  := '0';
    signal RX_Data       : STD_LOGIC_VECTOR (7 DOWNTO 0) := (others => '0');
```

```vhdl
uart: UART_Interface
generic map
(
    CLK_Frequency => 12000000,
    Baud_Rate     => 19200,
    OS_Rate       => 16,
    D_Width       => 8,
    Parity        => 0,
    Parity_EO     => '0'
)
port map
(
    CLK           => CLK,
    Reset         => Reset,
    RX            => RX,
    TX            => TX,
    TX_Enable     => TX_Enable,
    TX_Busy       => TX_Busy,
    TX_Data       => TX_Data,
    RX_Busy       => RX_Busy,
    RX_Data       => RX_Data
);
```

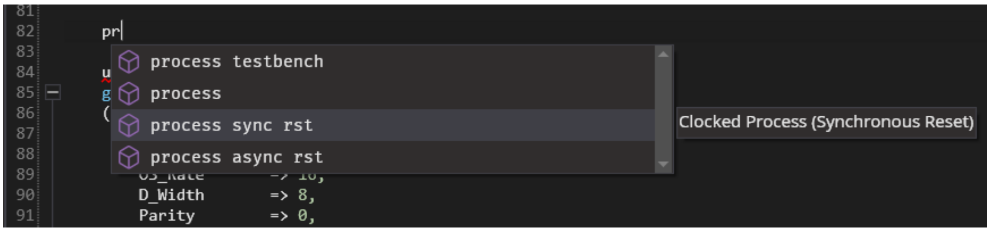
4.1.8 Do the same for the random number generator. But you only need the out_data output

```vhdl
    random: Random_Number
    generic map
    (
        init_seed => x"0123456789abcdef0123456789abcdef",
        pipeline  => true
    )
    port map
    (
        clk       => CLK,
        out_data  => random_data
    );
```

For your Info: When using VHDP, this process only consists of 2 steps:
1. You add the component
2. You right click the component to create and connect the signals



4.1.9 Create a process for our code that receives and sends the UART data



4.1.10 Create a variable for the current state

```
process(clk)
VARIABLE state : NATURAL range 0 to 6 := 0;
begin
    if rising_edge(clk) then
        if state = 0 then

        else

        end if;
    end if;
end process;
```

4.1.11 When the UART interface starts receiving something, go to the next state

```
begin
    if rising_edge(clk) then
        if state = 0 then
            if RX_Busy = '1' then
                state := 1;
            end if;
        else
```

4.1.12 When the interface finished receiving, save the range for the random number. Create a variable for that and you must convert the STD_LOGIC_VECTOR to a NATURAL

```
    VARIABLE state : NATURAL range 0 to 6 := 0;
    VARIABLE number_range : NATURAL range 1 to 10:= 0;
    begin
        if rising_edge(clk) then
            if state = 0 then
                if RX_Busy = '1' then
                    state := 1;
                end if;
            elsif state = 1 then
                if RX_Busy = '0' then
                    number_range := STD
                end if;
            end if;
        end if;
    end process;

    uart: UART_Interface
    generic map
    (
        CLK_Frequency => 12000000,
        Baud_Rate     => 19200,
        OS_Rate       => 16,
        D_Width       => 8,
        Parity        => 0,
        Parity_EO     => '0'
    )
    port map
    (
```

```
std_logic_1164
std_logic_vector range
resize (STD_LOGIC_VECTOR)
STD_LOGIC_VECTOR (BIT_VECTOR)
SIGNED (STD_LOGIC_VECTOR)
BIT (STD_LOGIC)
BIT_VECTOR (STD_LOGIC_VECTOR)
INTEGER (STD_LOGIC_VECTOR)
NATURAL (STD_LOGIC_VECTOR)
UNSIGNED (STD_LOGIC_VECTOR)
STD_LOGIC_VECTOR -> NATURAL
```

4.1.13 When you subtract 47 from the ASCII value for 0-9, you get a number from 1-10. Then you create a variable for the random number, convert 8 bits from the random_data vector to a number, get the remainder of the division with the number range using mod and add 1. Now you have a random number from 0 to your number range - 1.

```
    process(clk)
    VARIABLE state : NATURAL range 0 to 6 := 0;
    VARIABLE number_range  : NATURAL range 1 to 10 := 1;
    VARIABLE random_number : NATURAL range 0 to 9 := 0;
    begin
        if rising_edge(clk) then
            if state = 0 then
                if RX_Busy = '1' then
                    state := 1;
                end if;
            elsif state = 1 then
                if RX_Busy = '0' then
                    number_range  := TO_INTEGER(UNSIGNED(RX_Data)) - 47;
                    random_number := TO_INTEGER(UNSIGNED(random_data(7 downto 0))) mod number_range;
                end if;
            end if;
        end if;
    end process;
```

4.1.14 Set the data to transmit to your random number. Add 48 so you get the ASCII value and convert the NATURAL to a STD_LOGIC_VECTOR. Now set TX_Enable to ‘1’, so the transmission can start and go to the next state

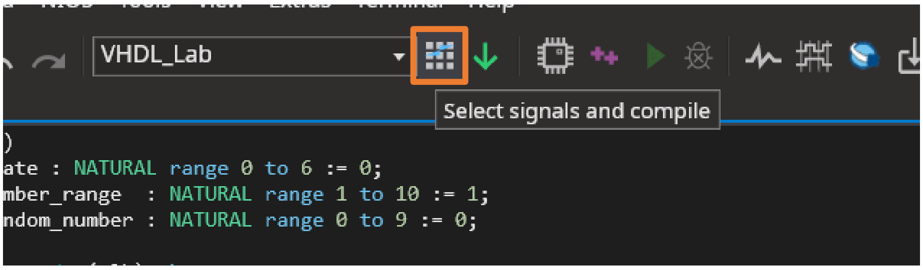
```
                    number_range  := TO_INTEGER(UNSIGNED(RX_Data)) - 47;
                    random_number := TO_INTEGER(UNSIGNED(random_data(7 downto 0))) mod number_range;
                    TX_Data       <= STD_LOGIC_VECTOR(TO_UNSIGNED(random_number + 48, TX_Data'LENGTH));
                    TX_Enable     <= '1';
                    state         := 2;
                end if;
            end if;
```

4.1.15 Wait until the transmission starts, go in the next state and wait until the transmission is finished. Then go to the first state

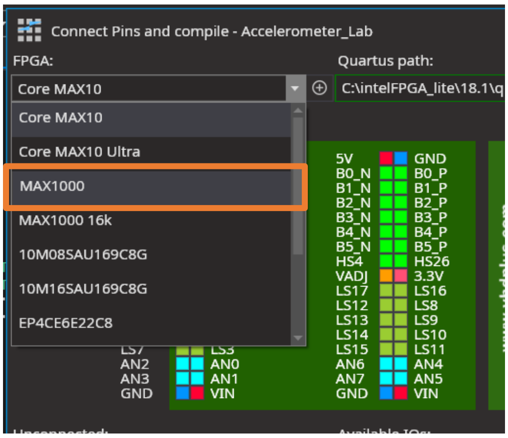
```
        elsif state = 2 then
            if TX_Busy = '1' then
                state := 3;
            end if;
        elsif state = 3 then
            if TX_Busy = '0' then
                state := 0;
            end if;
        end if;
```

4.2 Program the FPGA

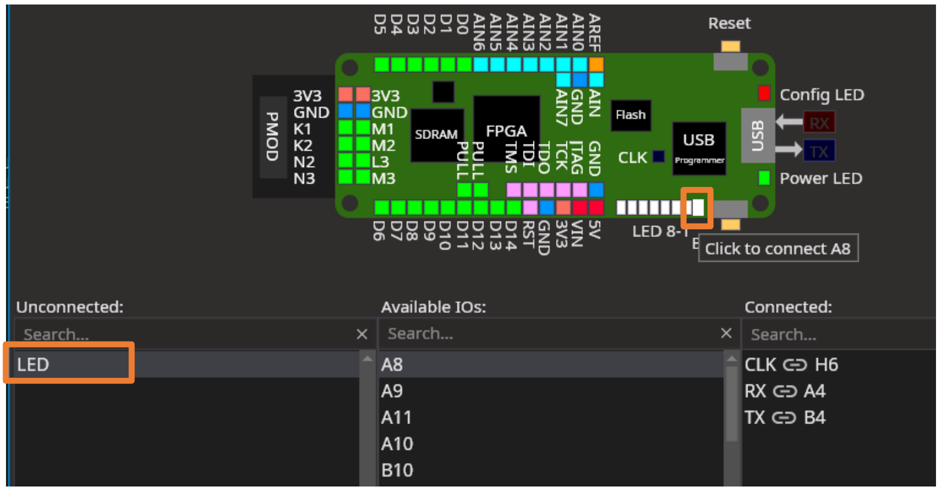
4.2.1 Click on the create button to convert the VHDP code to VHDL and select the pins

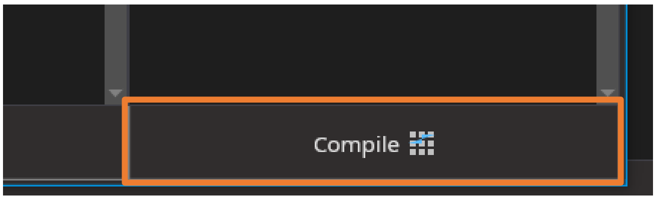


4.2.2 Select the MAX1000 as hardware

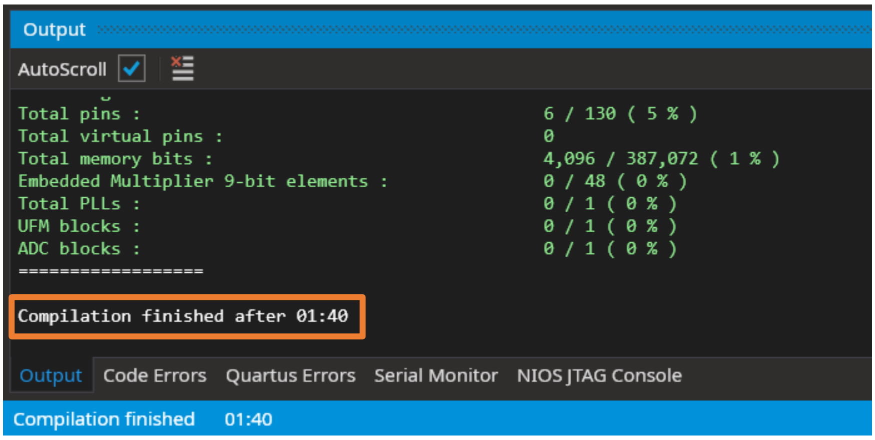


4.2.3 The RX and TX pins are already connected. You can then connect the blinking LED from the default program with one user LED on the MAX1000 board. Just click on the LED on the model and press enter. Then click on compile on the bottom

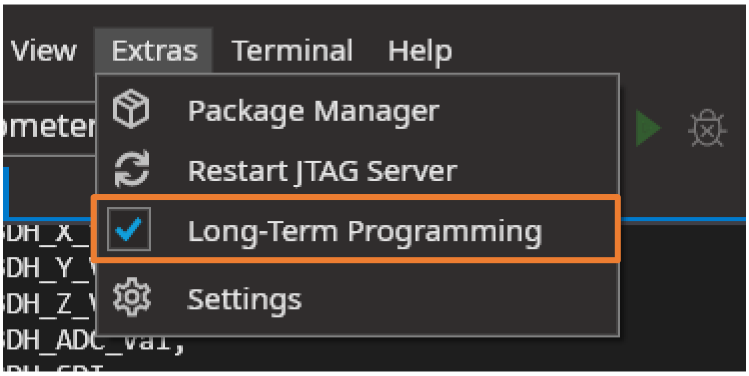




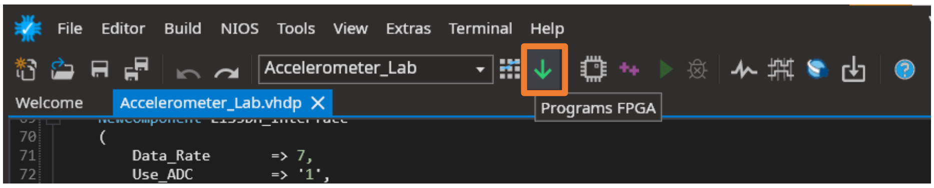
4.2.4 Wait until the compilation is finished



4.2.5 Select long term programming if you want to save the configuration on the FPGA



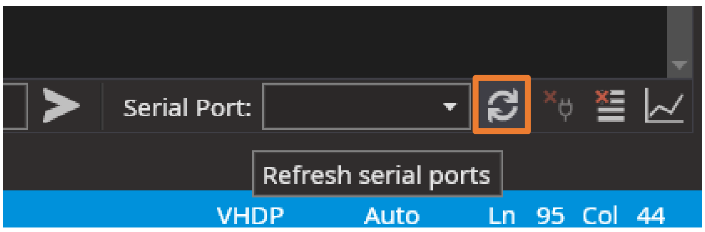
4.2.6 Connect the MAX1000 with an USB cable to your PC and click on the program button



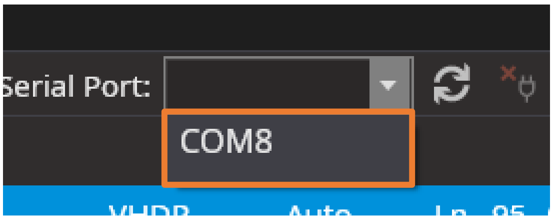
4.3 Try your program

4.3.1 Open the Serial Monitor

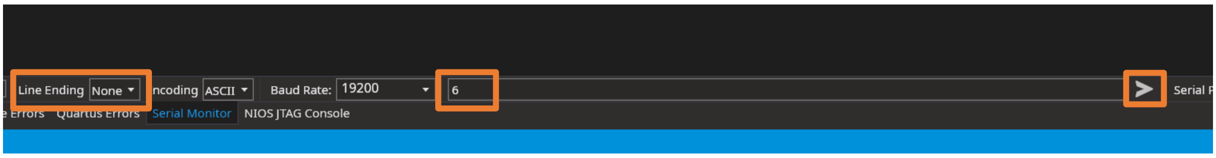
4.3.2 Make sure the baud rate is set to 19200 and refresh the connected devices



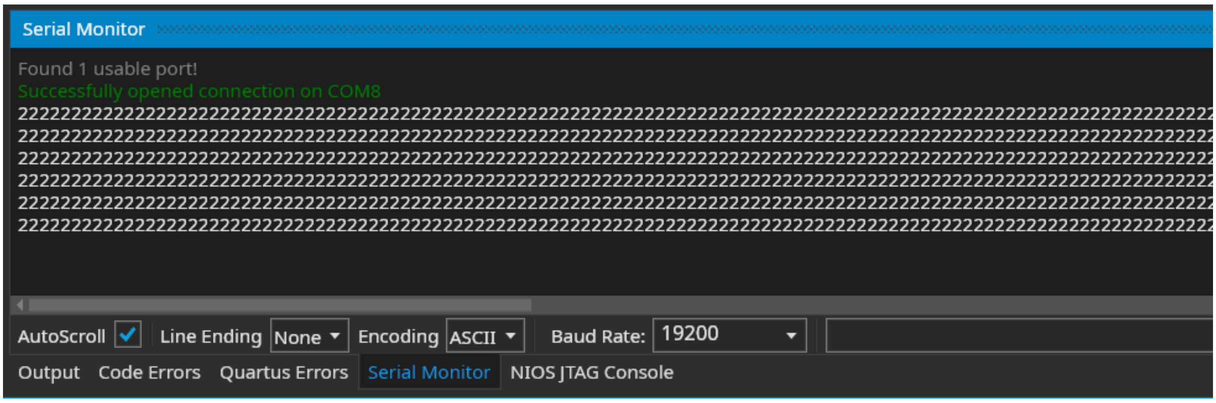
4.3.3 Select you MAX1000 that should appear in the dropdown menu



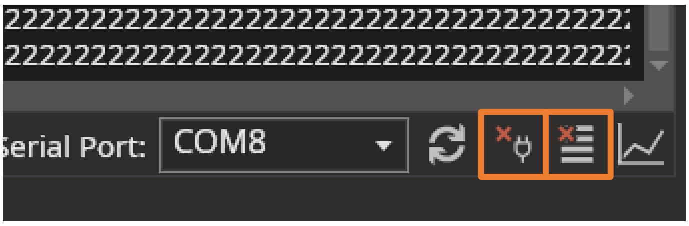
4.3.4 To get a number between 0 and 6, send “6” to the MAX1000. But first you should set the line ending to “None”, so only the char ‘6’ is sent.



4.3.5 You see the random number printed in the console, but something isn’t working correctly, because we only wanted one number as return



4.3.6 Disconnect the USB connection and clean the console

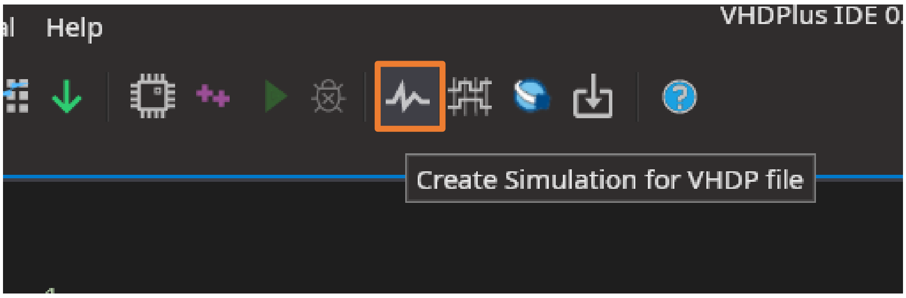


4.4 Simulate your program

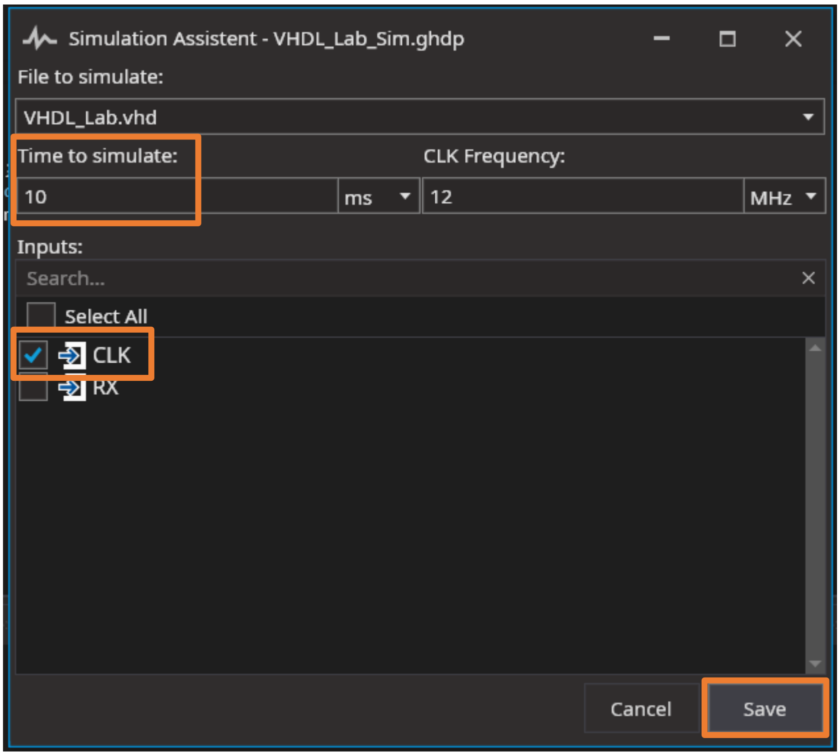
In the following steps you will learn how to find a problem in your code with a simulation. With this you see how the signals change and can find the problem.

For this you first use the VHDP version of a simulation with our simulation assistant. Then you see how to run a simulation with a VHDL testbench

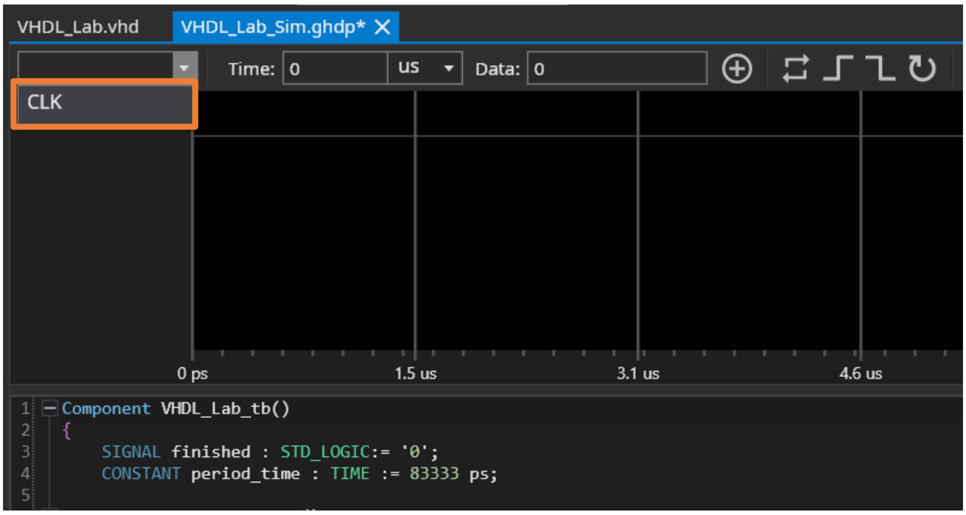
4.4.1 Click on the simulation button to create the simulation file



4.4.2 Set the time to simulate to 10ms and select the CLK input to simulate. Do not select the RX pin, you will learn a different type of simulation with this pin. Then click on save.

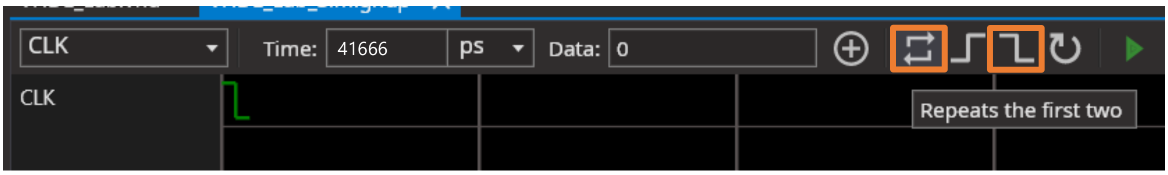


4.4.3 Now simulate the CLK pin. First select the signal

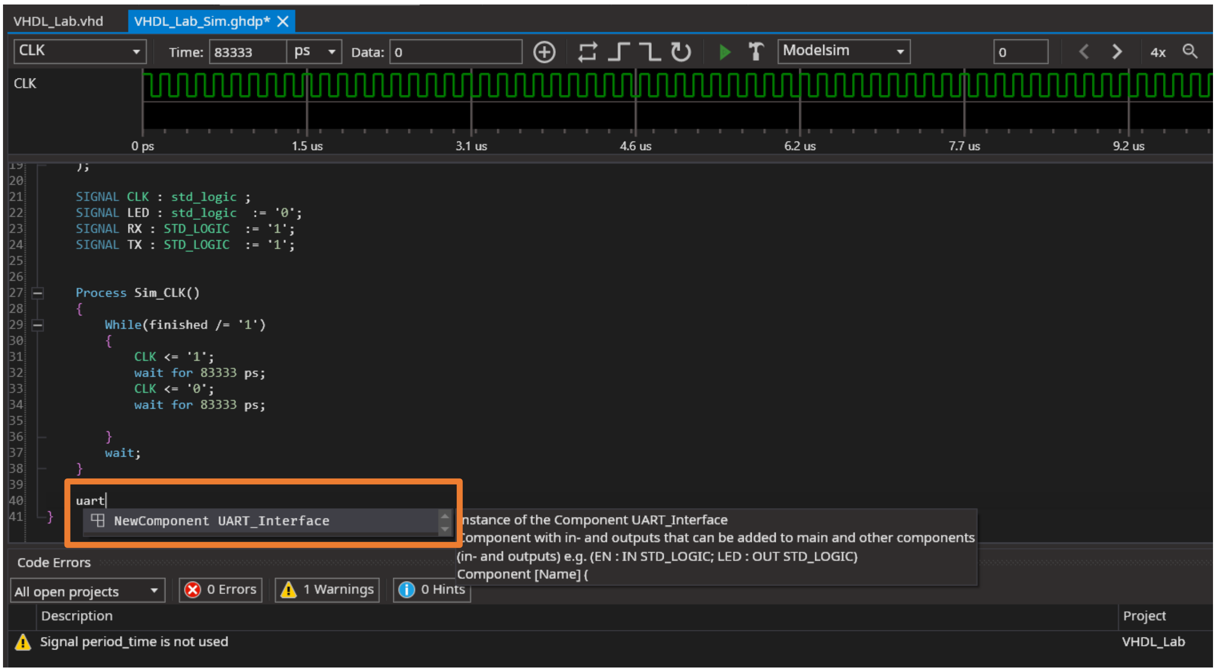


4.4.4 Set the time to 41666 pico seconds to simulate the 12MHz clock of the MAX1000 and press the high button. When using VHDP, the clock signal would be simulated automatically

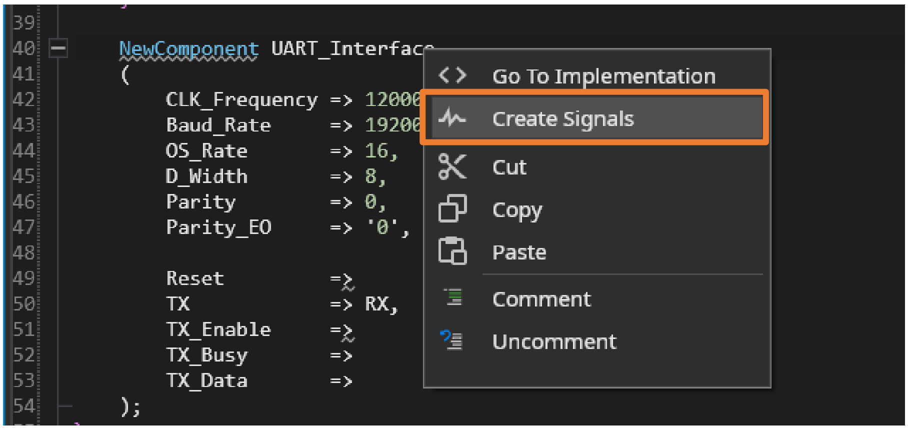
4.4.5 Click on the low button and then on the repeat button



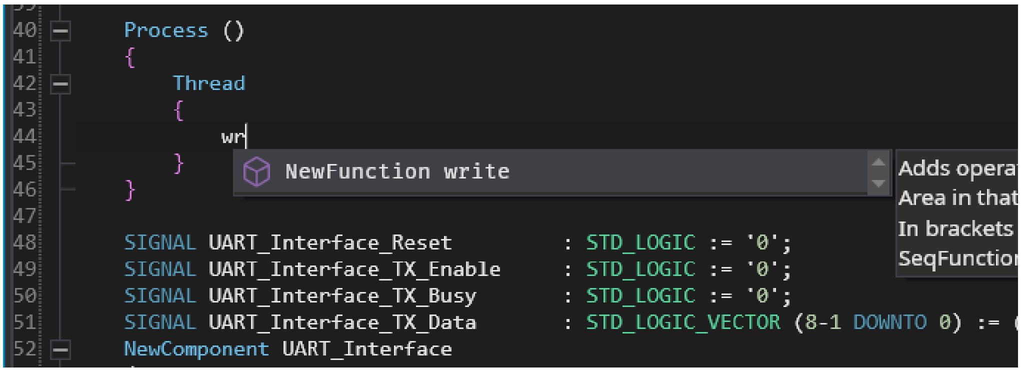
4.4.6 Now you simulate the RX signal. For this, add a UART interface to the testbench



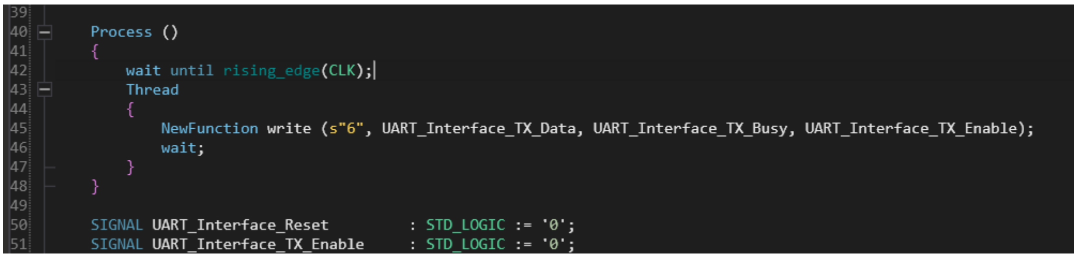
4.4.7 Delete the RX signals, connect TX with the RX signal and right click the component to create the signals



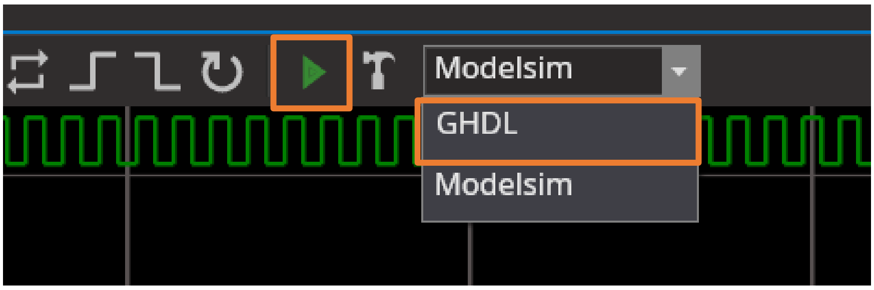
4.4.8 Create a process and a thread for your RX simulation. In the thread you can just use the write function of the UART interface



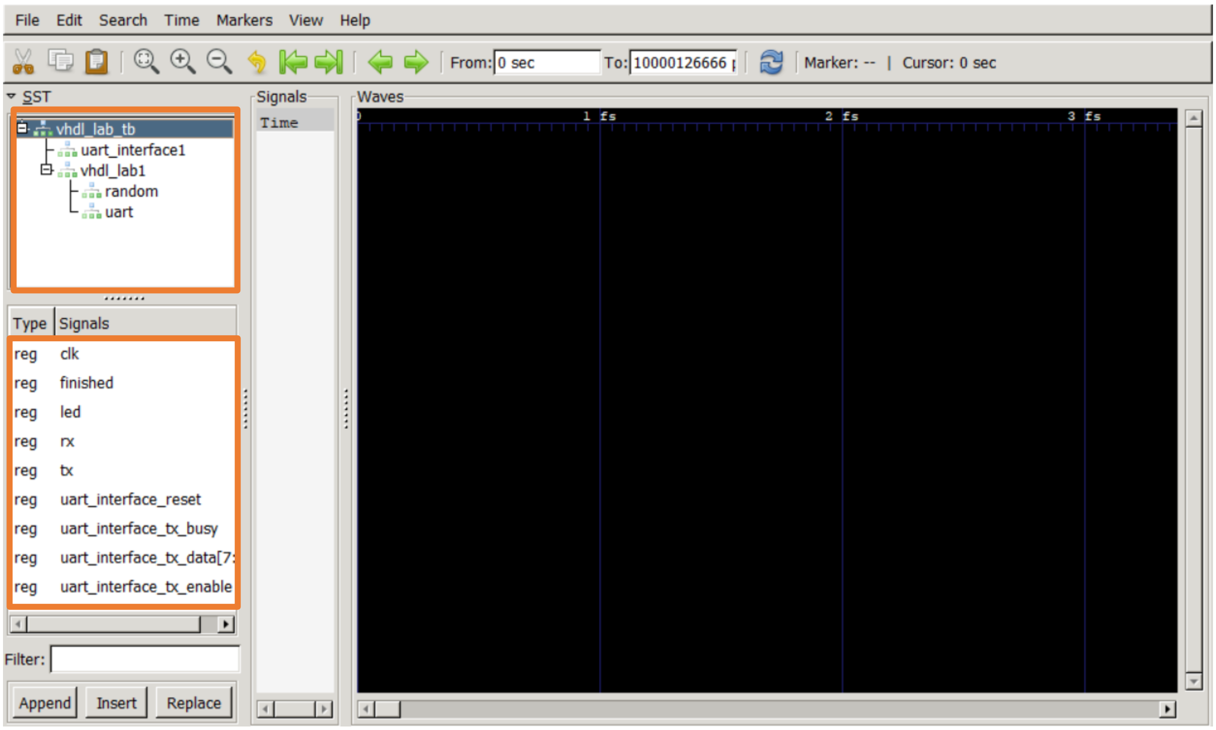
4.4.9 For the simulation you first must wait until the CLK rising edge, so the process runs every clock cycle. Set the char to the ascii value of 6 and connect the signals of your UART interface. At the end, you end this transmission with the statement “wait”



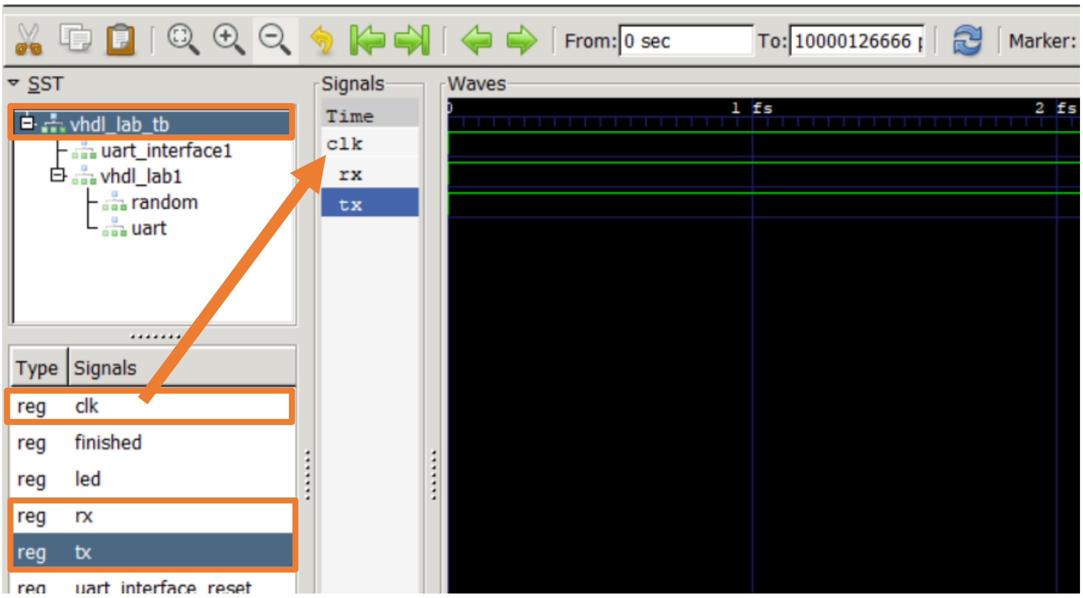
4.4.10 Now select GHDL as simulation program and click on run



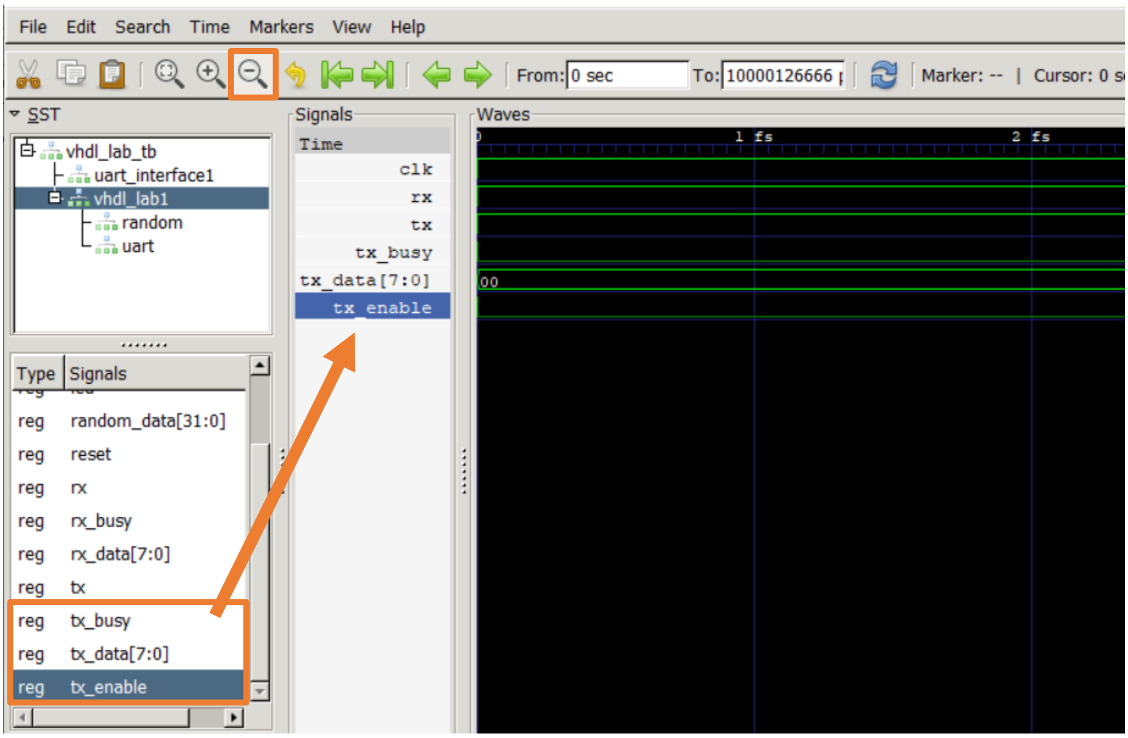
4.4.11 You can see all components in the simulation and see the signals from these



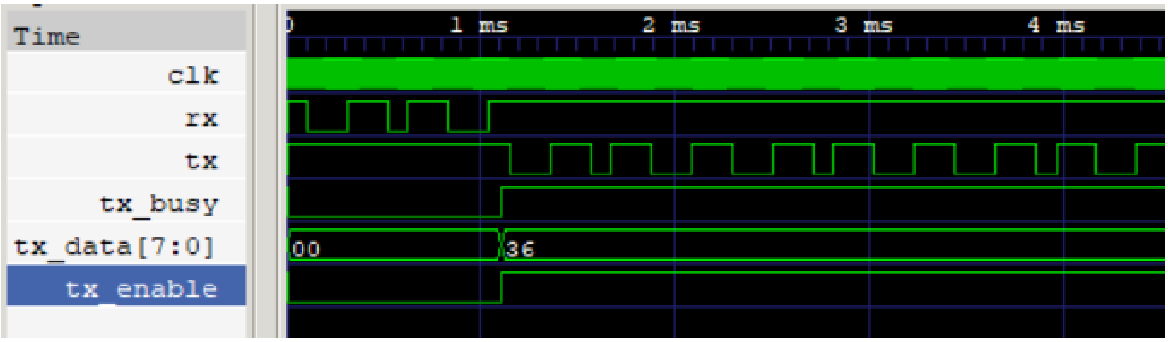
4.4.12 First move the clk, rx and tx signals of the testbench file to the wave



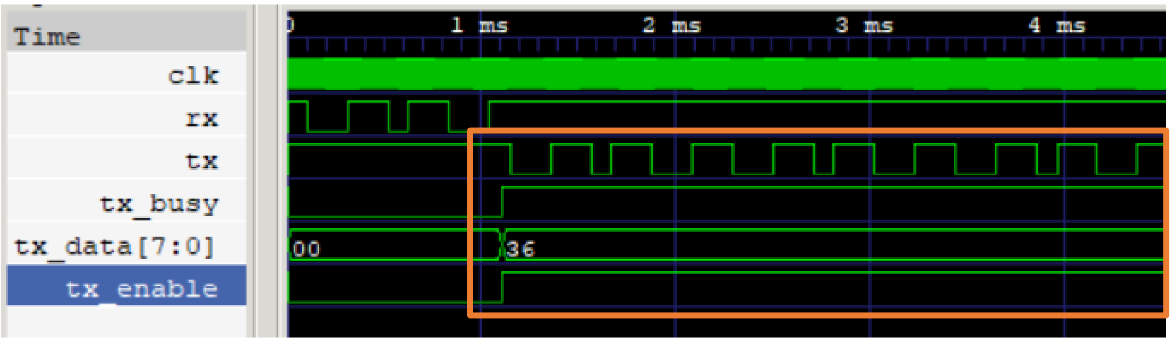
4.4.13 Go to the vhdl_lab1 component to simulate the signals of your code you wrote previously. Add the signals for transmitting with the UART interface to your wave and then zoom out to see more of the simulation



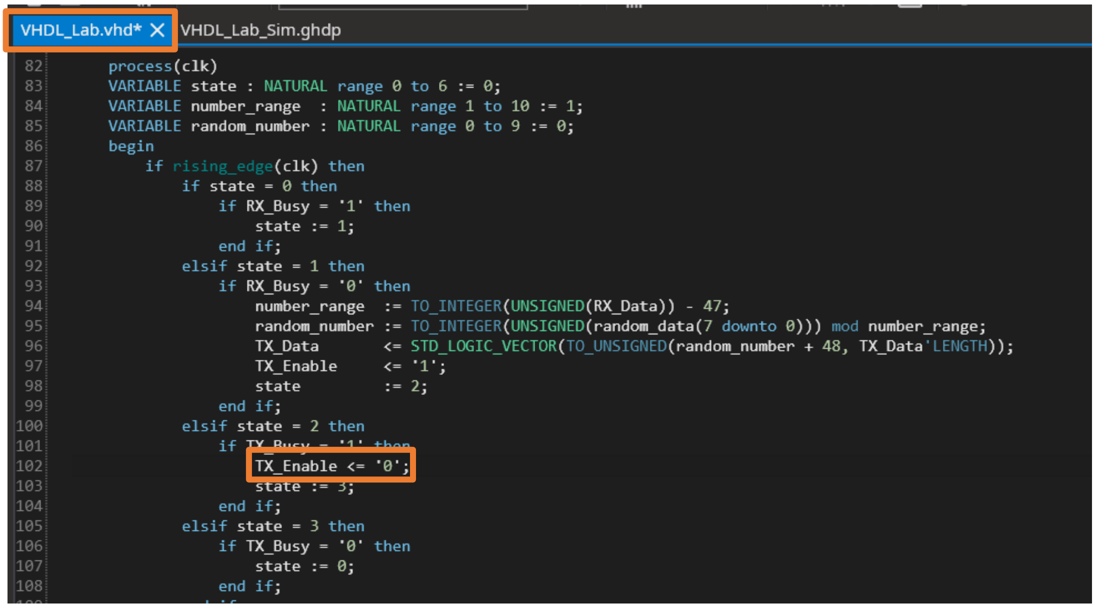
4.4.14 You can see how first your testbench sends the char '6' with the RX signal and then with TX, data is returned



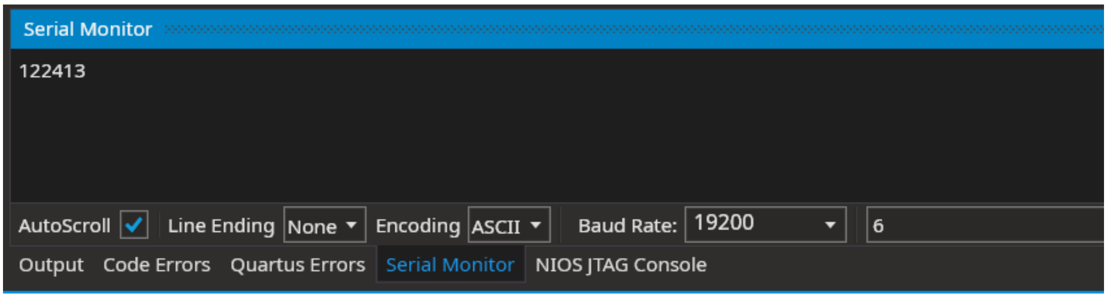
4.4.15 You see that the tx_busy and tx_enable signals stay ‘1’ and the TX data keeps sending. Now you see that the problem is that you don’t set tx_enable back to ‘0’.



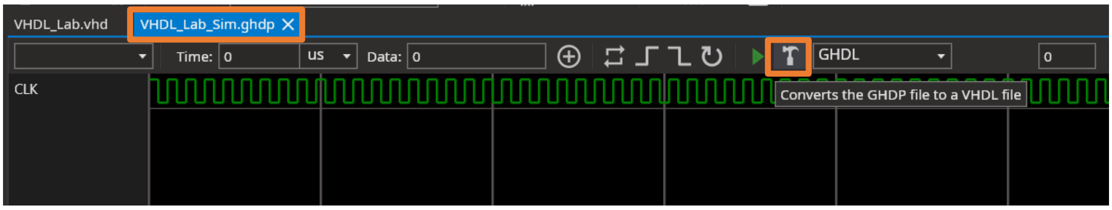
4.4.16 Go back to your code and fix the problem. Just set tx_enable to ‘0’ after the UART Interface starts sending



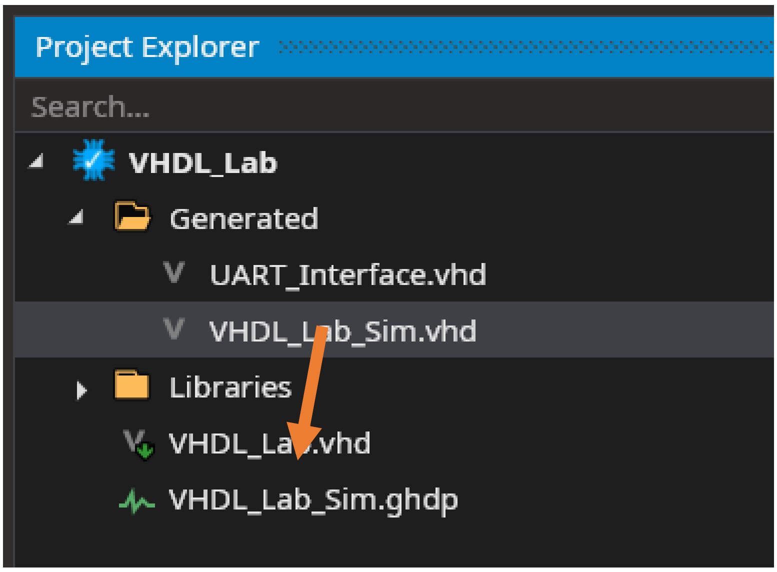
4.4.17 Now follow the steps 3.2.1 to 3.3.4 again to see if your code is now working. Now every time you click on the send button, a random number between 0 and 6 should be printed in the monitor. You can do this with every number between 0 and 9



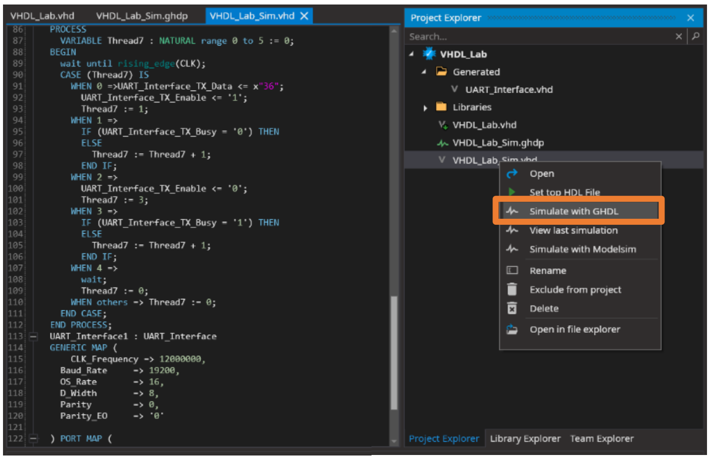
4.4.18 Finally, you can check the simulation again, but now using a VHDL testbench. To use you testbench of the first simulation, go to the VHDL_Lab_Sim.ghdp file and click on the build button



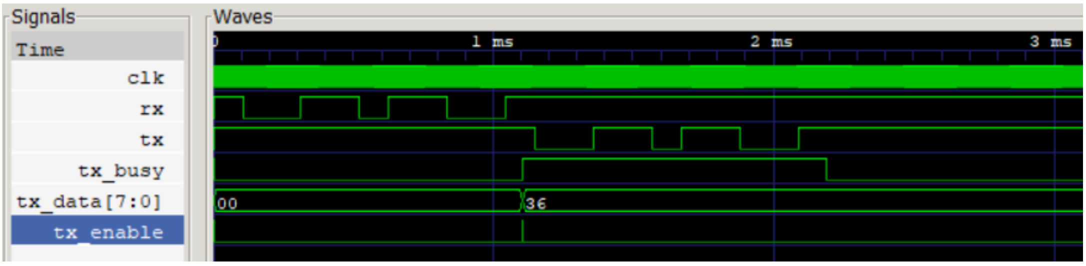
4.4.19 Drag the "VHDL_Lab_Sim.vhd" file from the "Generated" folder to your project folder and delete the VHDL_Lab_Sim.ghdp file



4.4.20 Right click the VHDL testbench and click on “Simulate with GHDL”



4.4.21 Repeat the steps 3.5.11 to 3.5.13 and this time your simulation should look like this

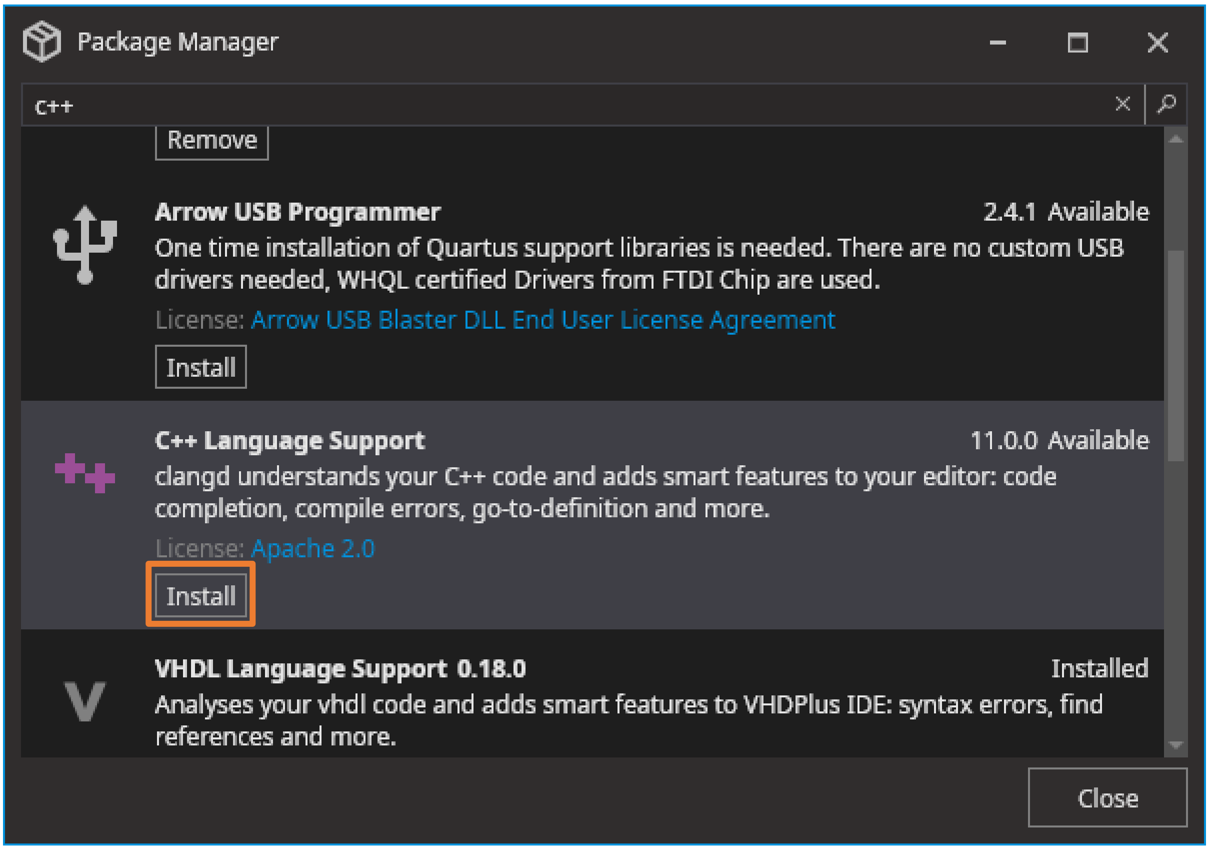


5. NIOS II

In this part you learn how to add and program the NIOS II Softcore processor using VHDPlus IDE.

5.1 Install C++ Support

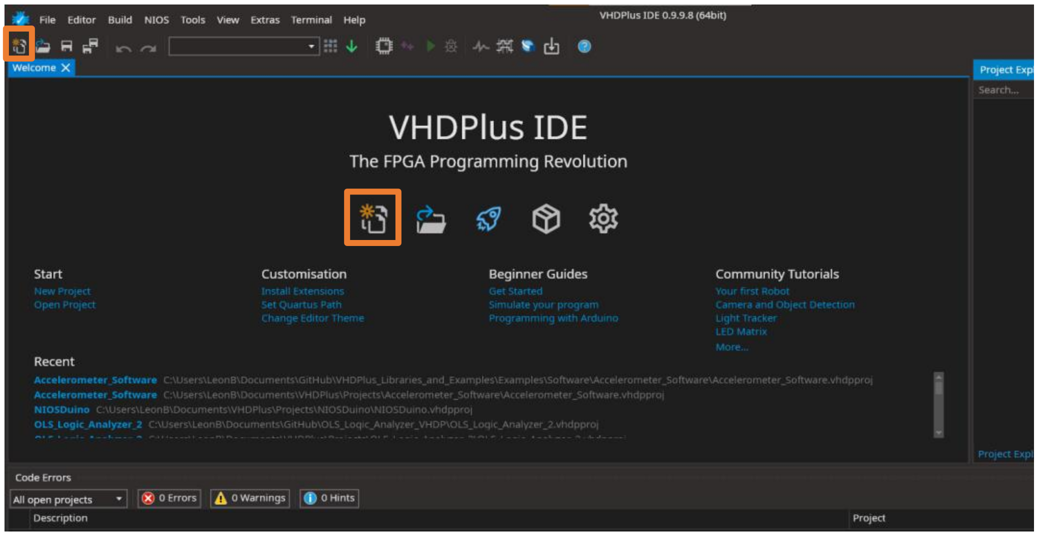
5.1.1 Open the Package Manager (Extras -> Package Manager) and install C++ Language support



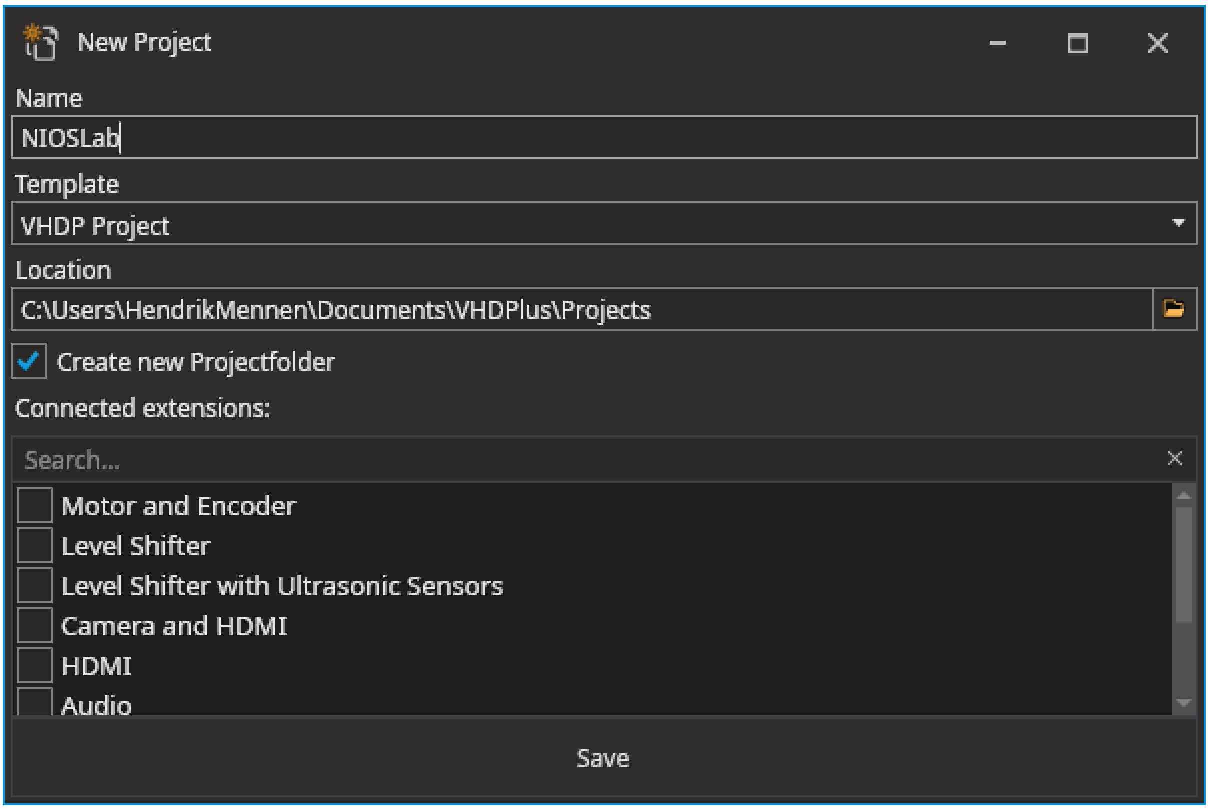
Alternatively to the steps 5.2 to 5.4.6 you can watch and follow this [Video Tutorial](#)!

5.2 Create the project

5.2.1 Create a VHDP Project

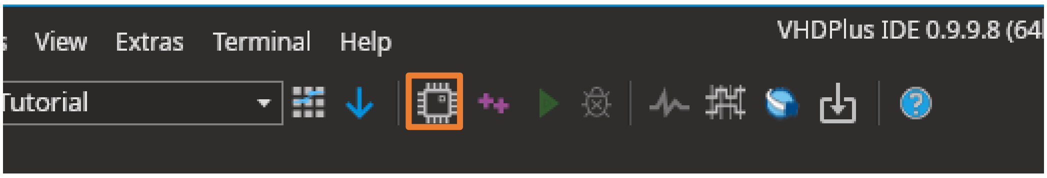


5.2.2 Name the Project, chose VHDP Project as template as click on save.

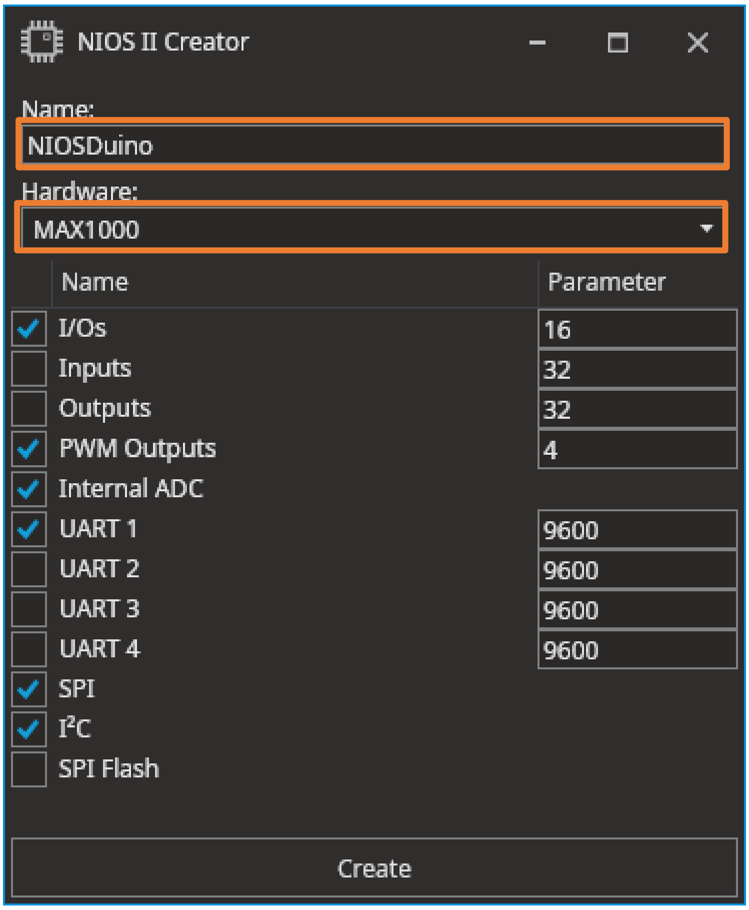


5.3 Create a processor.

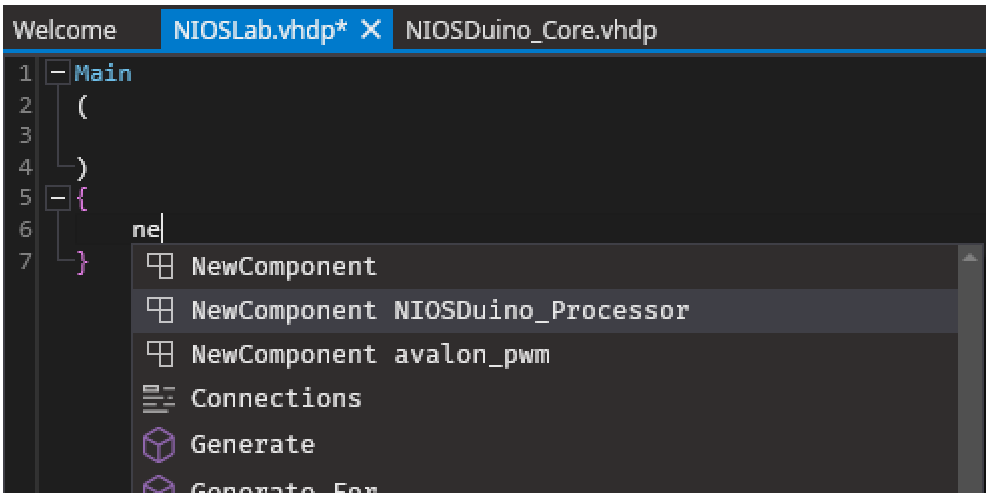
5.3.1 Open the NIOS II Creator



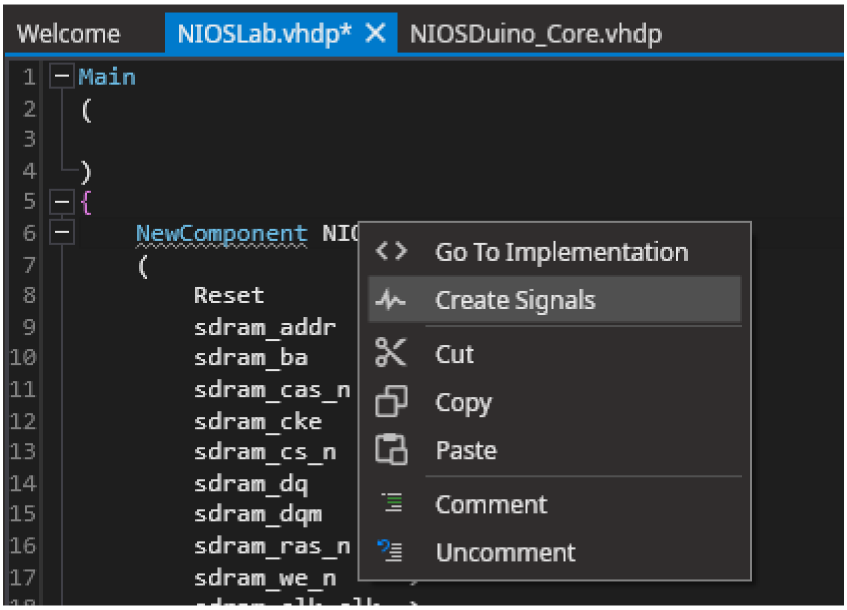
5.3.2 Set a name for your NIOS II Processor, select your hardware, and configure it to your needs. For this example, you don't need any peripherals, so you could unselect all to speed up the compilation.



5.3.3 Open your main file (NIOSLab.vhdp) and add the Processor to your code.

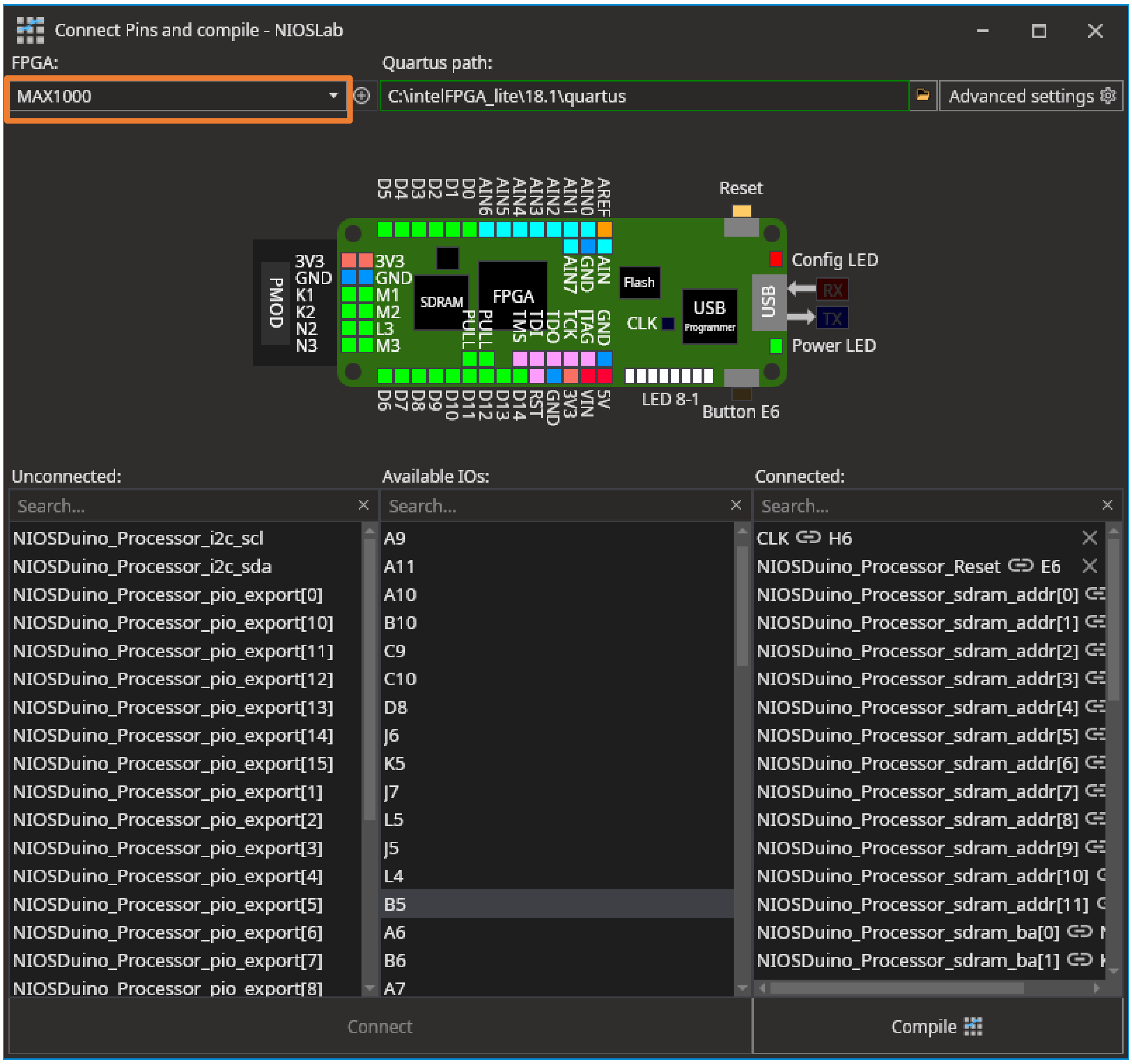


5.3.4 Create the required Signals



5.3.5 Compile and select the MAX1000.

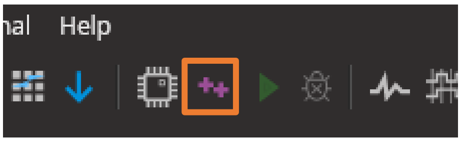
All required Signals are connected automatically. You can now connect optional I/Os to use withing your software project



5.3.6 Wait for the compilation to finish, connect the MAX1000 with an USB cable to your PC and program the design to your MAX1000

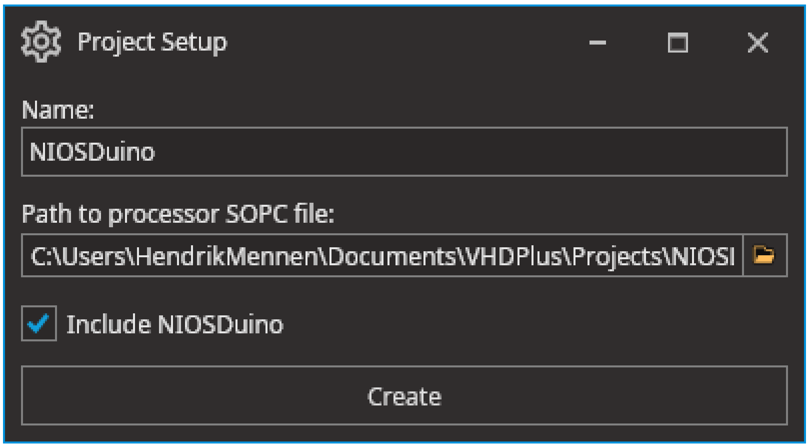
5.4 Create a software project

5.4.1 Open the NIOS II Software creator



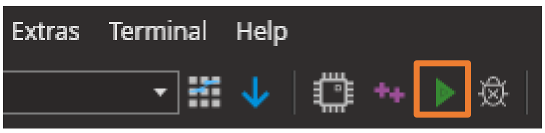
5.4.2 Set a name for the software project and target it to the .SOPCINFO file that got generated during compilation inside your project root folder.
Select “Include NIOSDuino” to start with an Arduino like template and libraries.

Continue by clicking on “Create” and wait a few seconds for the BSP to generate.

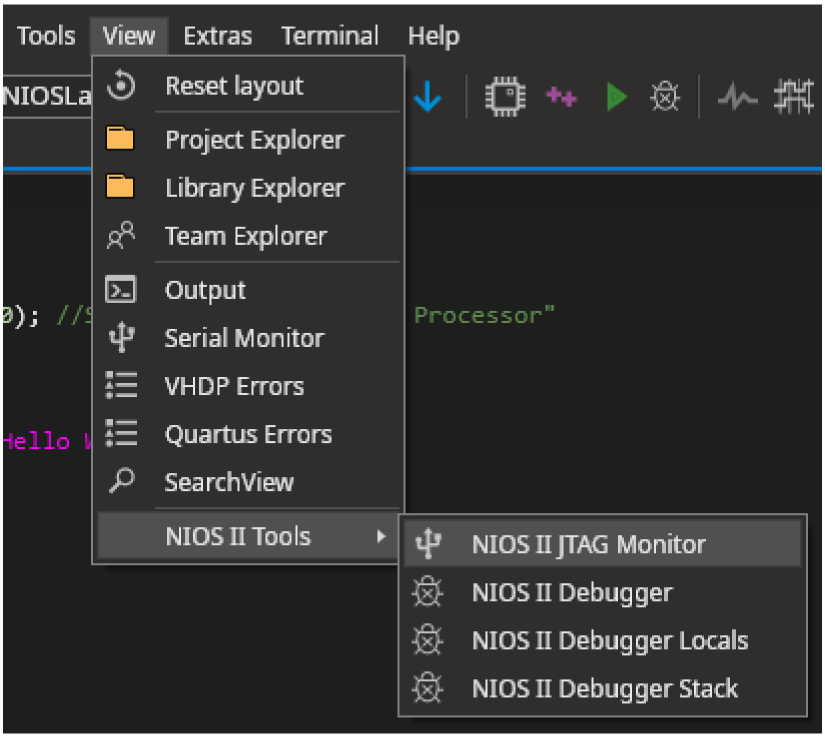


5.4.3 The automatically generated “Hello World” program should look like this:

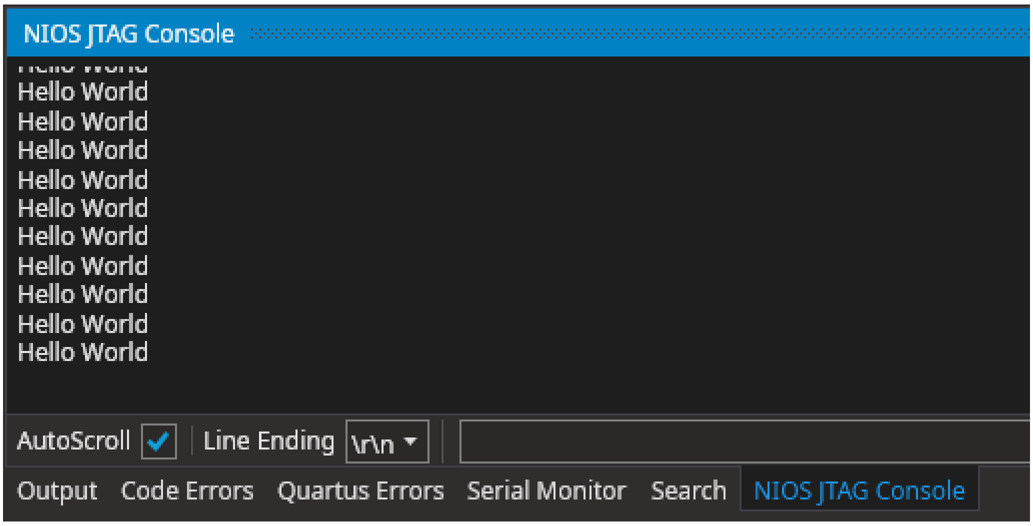
5.4.4 Compile and download the program.



5.4.5 Open the NIOS II JTAG monitor.



5.4.6 Check if the program works.

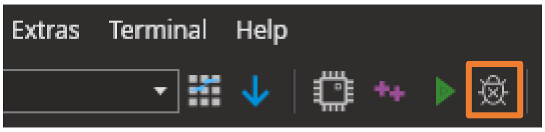


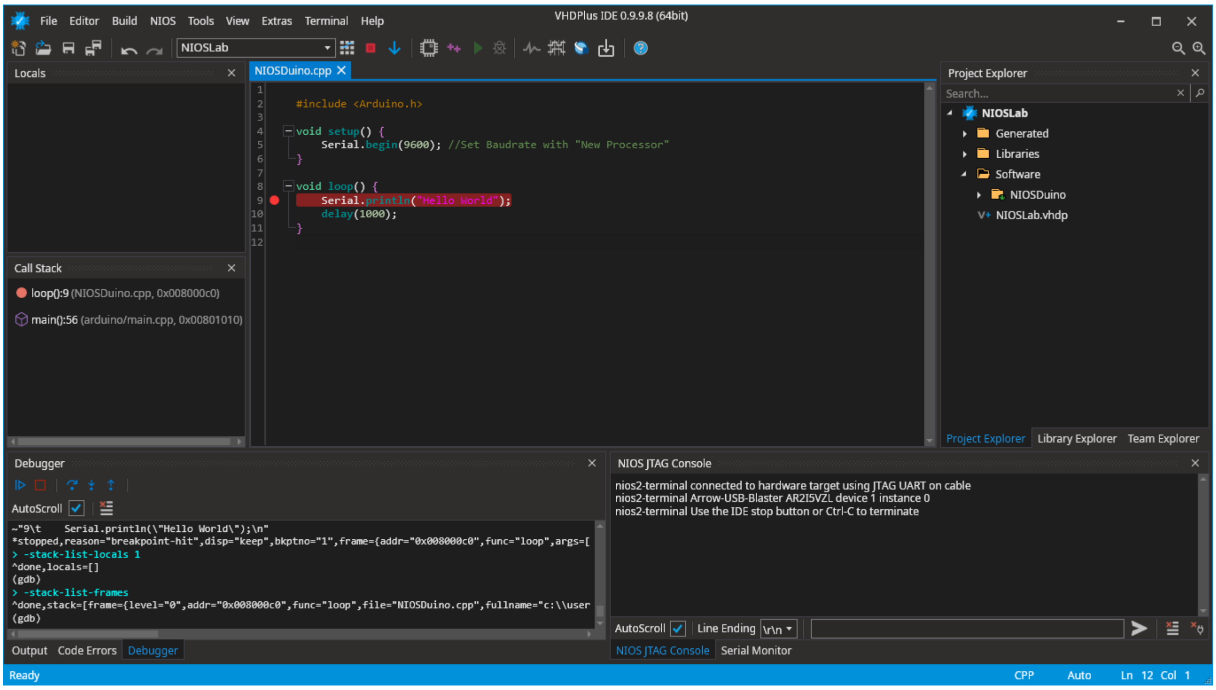
5.5 Debug

Warning: As of VHDPlus IDE 0.9.9.9, the integrated NIOS Debugger is an VHDPlus IDE feature we are still working on. Some features are missing, and bugs are likely.

5.5.1 Set breakpoints by simply clicking on the side next to the line number

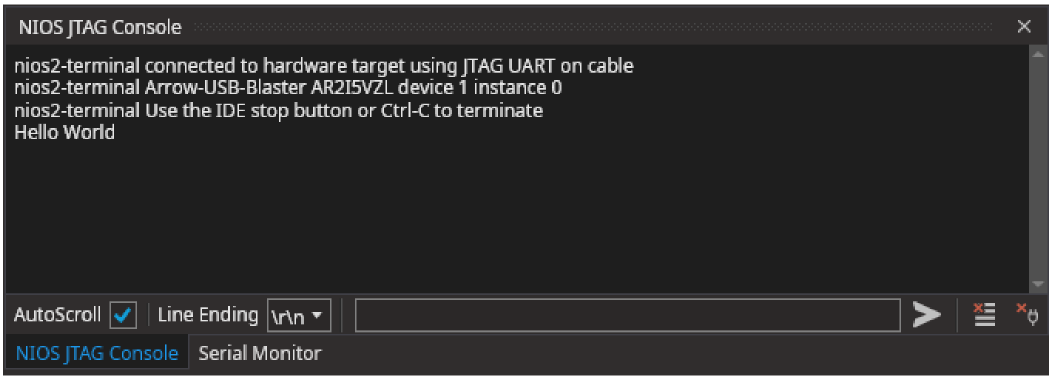
5.5.2 Start the Debugger. This will compile your code, start the debug server, and automatically connects it to your NIOS processor. The program will start executing until it reaches the breakpoint.





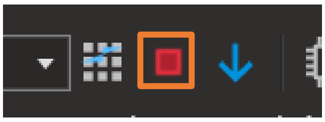
5.5.3 You can control the program execution using the debugger buttons. Click on the “Continue” button to start executing the program until it hits the breakpoint again. Make sure to select the NIOS JTAG Console window to see if the program is still working

5.5.4 You should see “Hello World” in the NIOS JTAG console every time you press on continue.



5.5.5 Examine variables live by hovering over them.

5.5.6 Stop the debugger and disconnect the JTAG Console



CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED THE VHDPLUS OVERVIEW LAB!

6 Legal Disclaimer

ARROW ELECTRONICS

EVALUATION BOARD LICENSE AGREEMENT

By using this evaluation board or kit (together with all related software, firmware, components, and documentation provided by Arrow, “Evaluation Board”), You (“You”) are agreeing to be bound by the terms and conditions of this Evaluation Board License Agreement (“Agreement”). Do not use the Evaluation Board until You have read and agreed to this Agreement. Your use of the Evaluation Board constitutes Your acceptance of this Agreement.

PURPOSE

The purpose of this evaluation board is solely intended for evaluation purposes. Any use of the Board beyond these purposes is on your own risk. Furthermore, according the applicable law, the offering Arrow entity explicitly does not warrant, guarantee or provide any remedies to you with regard to the board.

LICENSE

Arrow grants You a non-exclusive, limited right to use the enclosed Evaluation Board offering limited features only for Your evaluation and testing purposes in a research and development setting. Usage in a live environment is prohibited. The Evaluation Board shall not be, in any case, directly or indirectly assembled as a part in any production of Yours as it is solely developed to serve evaluation purposes and has no direct function and is not a finished product.

EVALUATION BOARD STATUS

The Evaluation Board offers limited features allowing You only to evaluate and test purposes. The Evaluation Board is not intended for consumer or household use. You are not authorized to use the Evaluation Board in any production system, and it may not be offered for sale or lease, or sold, leased or otherwise distributed for commercial purposes.

OWNERSHIP AND COPYRIGHT

Title to the Evaluation Board remains with Arrow and/or its licensors. This Agreement does not involve any transfer of intellectual property rights (“IPR) for evaluation board. You may not remove any copyright or other proprietary rights notices without prior written authorization from Arrow or it licensors.

RESTRICTIONS AND WARNINGS

Before You handle or use the Evaluation Board, You shall comply with all such warnings and other instructions and employ reasonable safety precautions in using the Evaluation Board. Failure to do so may result in death, personal injury, or property damage.

You shall not use the Evaluation Board in any safety critical or functional safety testing, including but not limited to testing of life supporting, military or nuclear applications. Arrow expressly disclaims any responsibility for such usage which shall be made at Your sole risk.

WARRANTY

Arrow warrants that it has the right to provide the evaluation board to you. This warranty is provided by Arrow in lieu of all other warranties, written or oral, statutory, express or implied, including any warranty as to merchantability, non-infringement, fitness for any particular purpose, or uninterrupted or error-free operation, all of which are expressly disclaimed. The evaluation board is provided “as is” without any other rights or warranties, directly or indirectly.

You warrant to Arrow that the evaluation board is used only by electronics experts who understand the dangers of handling and using such items, you assume all responsibility and liability for any improper or unsafe handling or use of the evaluation board by you, your employees, affiliates, contractors, and designees.

LIMITATION OF LIABILITIES

In no event shall Arrow be liable to you, whether in contract, tort (including negligence), strict liability, or any other legal theory, for any direct, indirect, special, consequential, incidental, punitive, or exemplary damages with respect to any matters relating to this agreement. In no event shall arrow’s liability arising out of this agreement in the aggregate exceed the amount paid by you under this agreement for the purchase of the evaluation board.

IDENTIFICATION

You shall, at Your expense, defend Arrow and its Affiliates and Licensors against a claim or action brought by a third party for infringement or misappropriation of any patent, copyright, trade secret or other intellectual property right of a third party to the extent resulting from (1) Your combination of the Evaluation Board with any other component, system, software, or firmware, (2) Your modification of the Evaluation Board, or (3) Your use of the Evaluation Board in a manner not permitted under this Agreement. You shall indemnify Arrow and its Affiliates and Licensors against and pay any resulting costs and damages finally awarded against Arrow and its Affiliates and Licensors or agreed to in any settlement, provided that You have sole control of the defense and settlement of the claim or action, and Arrow cooperates in the defense and furnishes all related evidence under its control at Your expense. Arrow will be entitled to participate in the defense of such claim or action and to employ counsel at its own expense.

RECYCLING

The Evaluation Board is not to be disposed as an urban waste. At the end of its life cycle, differentiated waste collection must be followed, as stated in the directive 2002/96/EC. In all the countries belonging to the European Union (EU Dir. 2002/96/EC) and those following differentiated recycling, the Evaluation Board is subject to differentiated recycling at the end of its life cycle, therefore: It is forbidden to dispose the Evaluation Board as an undifferentiated waste or with other domestic wastes. Consult the local authorities for more information on the proper disposal channels. An incorrect Evaluation Board disposal may cause damage to the environment and is punishable by the law.