Hello World – Using Microblaze Softcore

In this tutorial we will create a Vivado project and add a MicroBlaze softcore. For the programming of the softcore we will use Vitis. Later on we will load the program to the flash of the FPGA. If we are successful, we will receive “Hello World” on our terminal program.

Tools we need:

- Alchitry AU
- USB-C Cable
- Vivado 2022.2
- Vitis 2022.2
- Alchitry Loader 1.0
- A terminal program of your choice

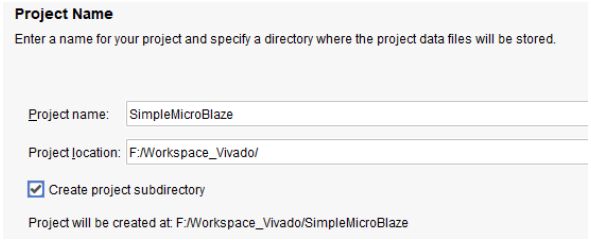
Content

Part one: Creating the Block Design in Vivado

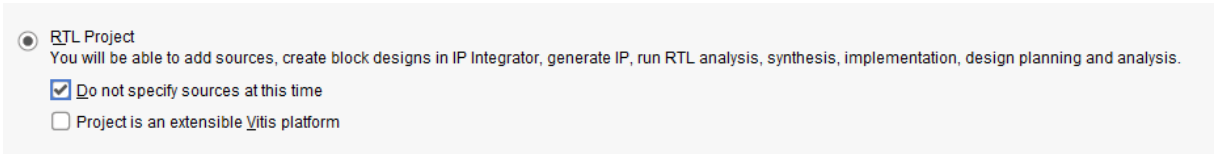
Create a new Project

Open VIVADO and create a new Project by clicking on „Create Project“

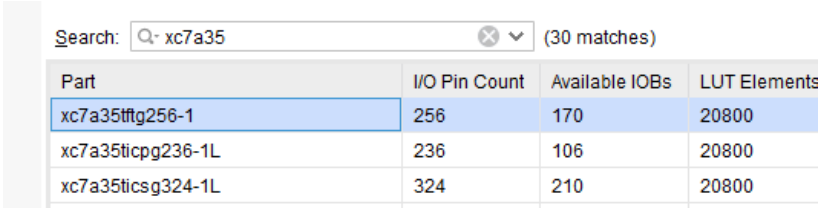
On the next page specify project name and location.



Next we select RTL Project and „Do not specify sources at this time“, as we have no source file at this moment.

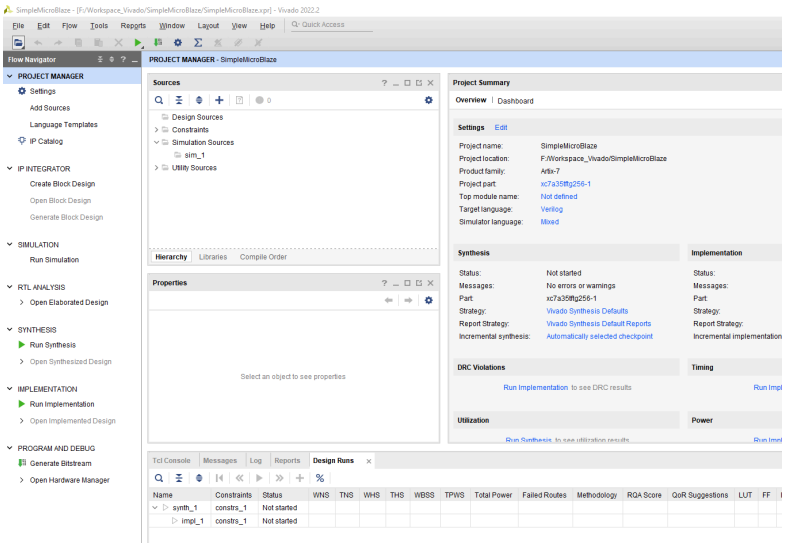


Than we have to select our FPGA. The Alchitry Au–Board is using the „Xc7a35tftg256-1“.



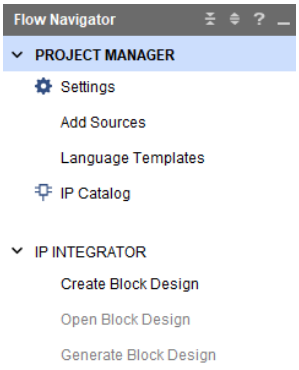
Click on „finish“.

After some seconds you should see the Vivado develop environment.



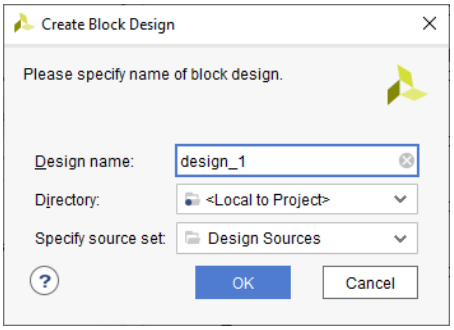
Creating the Block Design

In Vivado the block design is used to create the top file by using graphic blocks. In the Project Manager we click on „Create Block Design“.



.

On the next dialog we just click on „OK“.



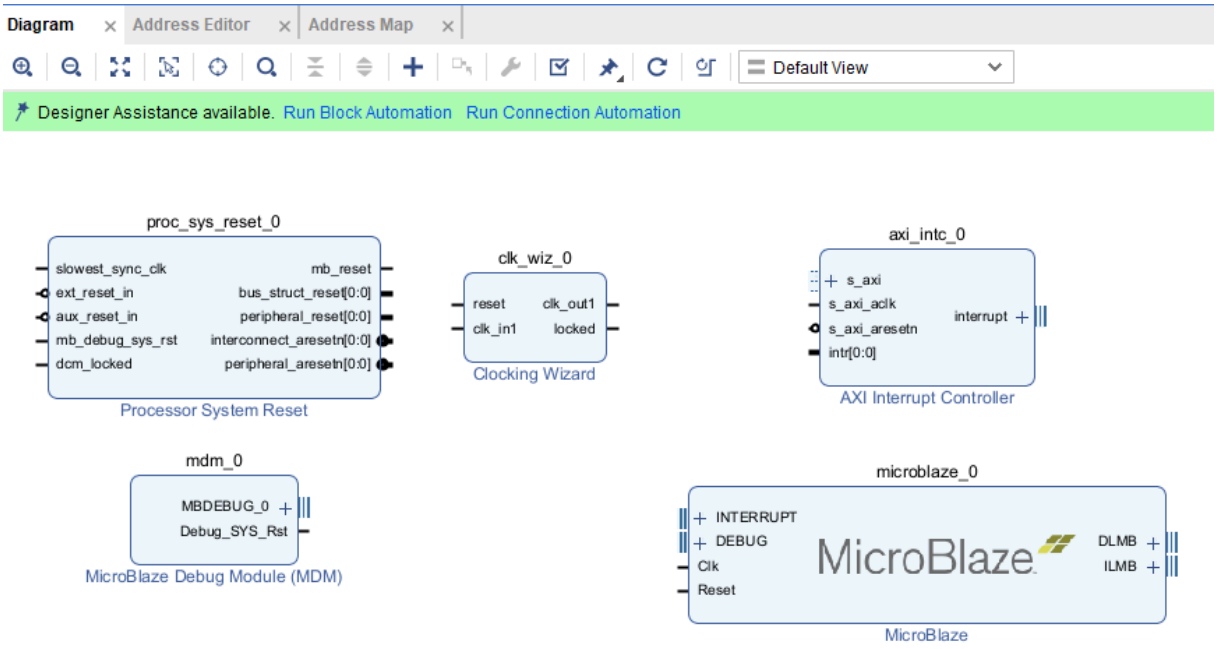
Now you see the empty diagram.

As Vivado tells us, you can add IP Blocks by clicking on the plus “Add IP”. For our design we need:

- MicroBlaze
- MicroBlaze Debug Module (MDM)
- AXI Interrupt Controller
- Clocking Wizard
- Processor System Reset
- AXI Uartlite

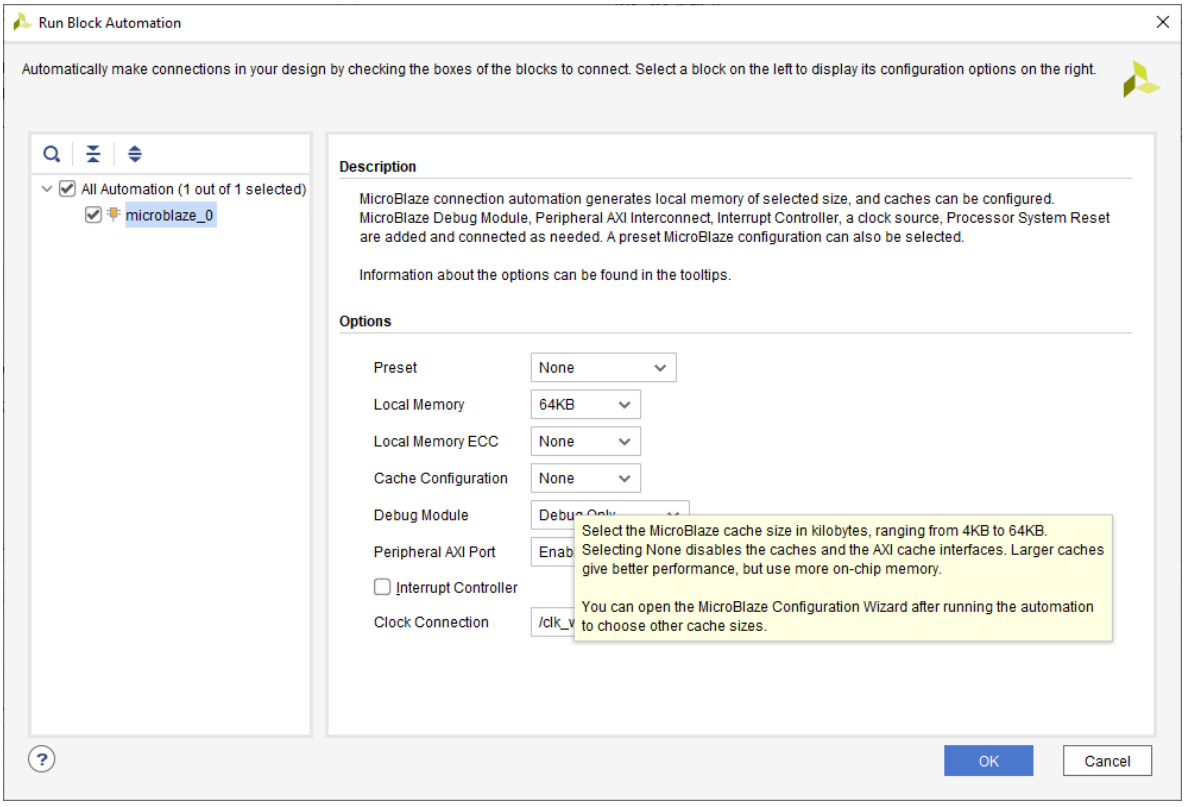
Add them all.

Afterwards there are all blocks lying around without any connection.



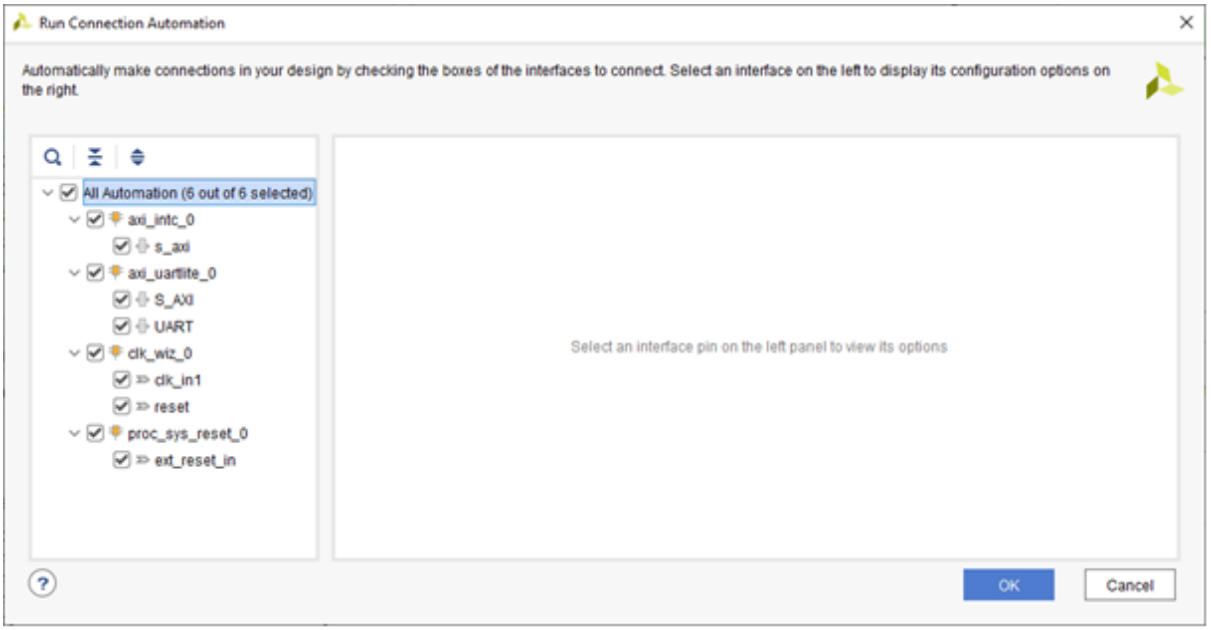
Some of these blocks need configurations to make. Luckily Vivado helps us here. Click on “Run Block Automation”.

You can see, our MicroBlaze needs some configuration. Please give it 64KB of Local Memory. The preselected 8KB are not enough for the “printf” command which we need later. Even the size optimized “xil_printf” will not fit in 8KB of local memory.

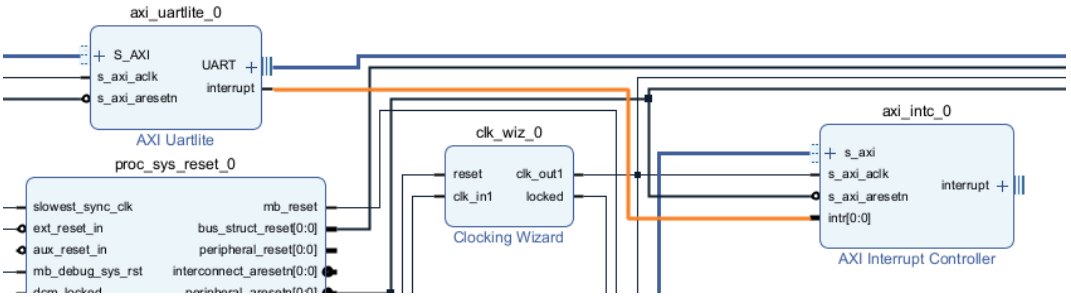


After clicking on “OK” you can see that Vivado created some connections.

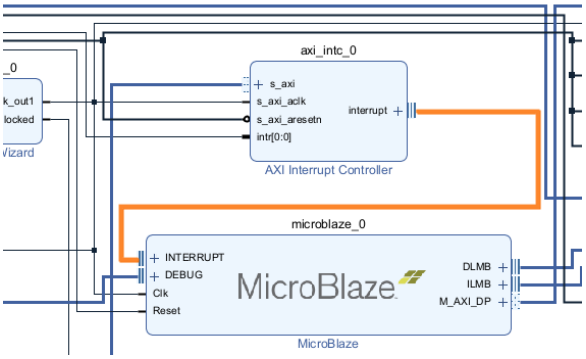
Next click on “Run Connection Automation” and select **all** and quit with “OK”.



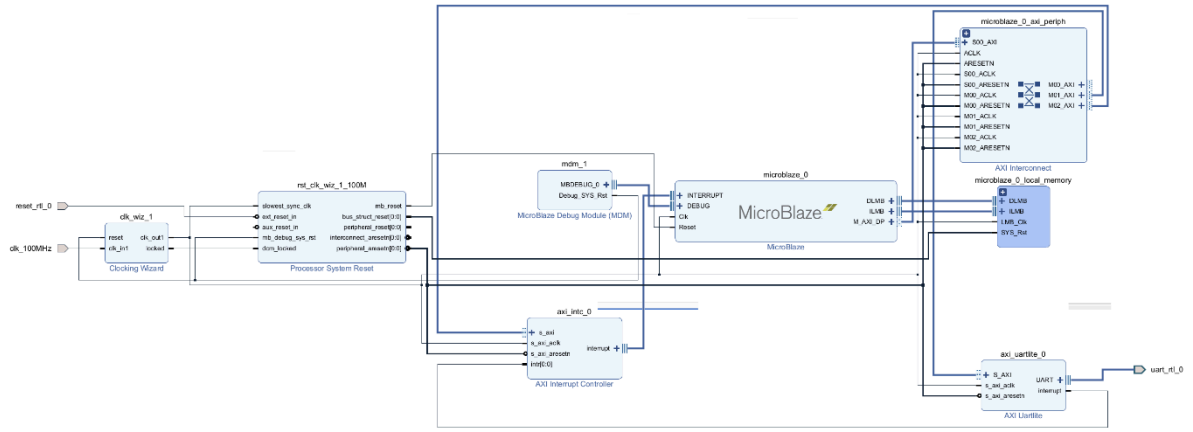
Vivado did a great job! Almost everything is now connected. However, our interrupt line is missing. If the AXI Uartlite receives a signal it will generate an interrupt flag. We are not using the receiving function in this tutorial. However, I am sure you will use it later.
So, connect the interrupt output of the AXI Uartlite with the AXI Interrupt Controller:



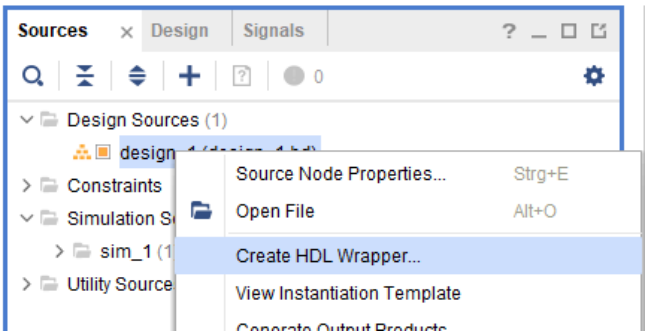
Connect the AXI Interrupt Controller with the MicroBlaze itself.



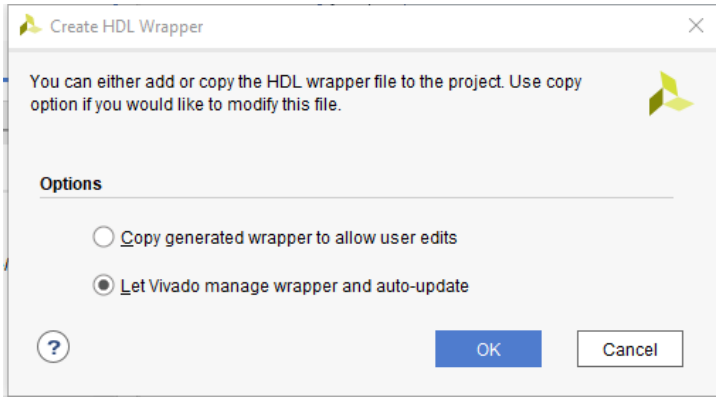
Our diagram is finish and should look like this:



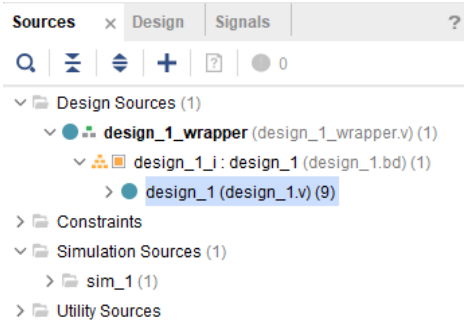
Now, that we have our diagram, we need one more step to generate our top file. Therefore right click on **Design_1->Create HDL Wrapper…**



This wrapper translates the diagram to the top file with its Verilog design and automaticly keeps it up to date.



In the Sources-Window you can see why the call it wrapper:



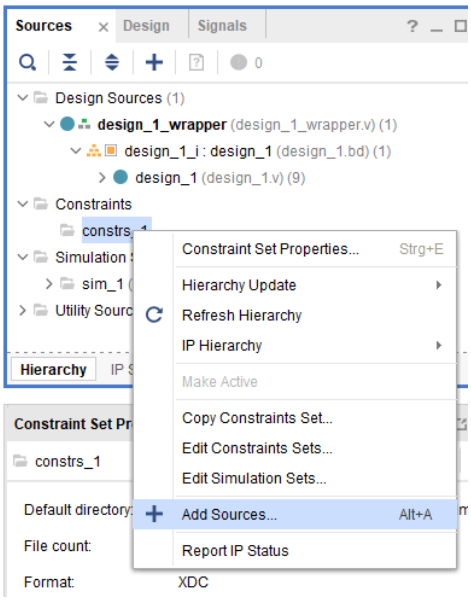
If you open the design_1.v, you can see the typical Verilog design of the top file:

```
13  module design_1
14     (clk_100MHz,
15      reset_rtl_0,
16      uart_rtl_0_rxd,
17      uart_rtl_0_txd);
18    (* X_INTERFACE_INFO = "xilinx.com:signal:clock:1.0
19    (* X_INTERFACE_INFO = "xilinx.com:signal:reset:1.0
20    (* X_INTERFACE_INFO = "xilinx.com:interface:uart:1.
21    (* X_INTERFACE_INFO = "xilinx.com:interface:uart:1.
22
23    wire axi_intc_0_interrupt_INTERRUPT;
24    wire axi_uartlite_0_UART_RxD;
25    wire axi_uartlite_0_UART_TxD;
26    wire axi_uartlite_0_interrupt;
```

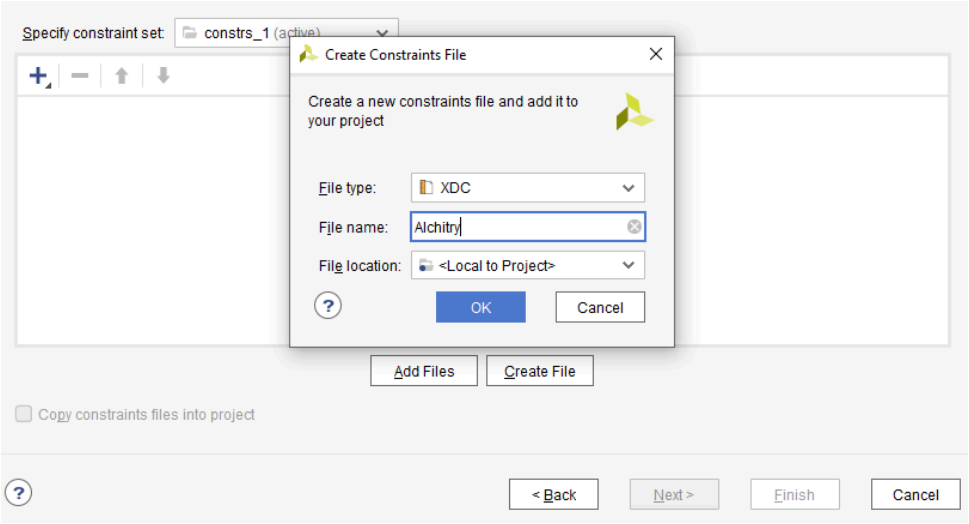
Adding Constraints

Now that our block diagram and top file are finished, we need a constraint file to specify our inputs and outputs.

Right click on **Constr_1->Add Sources…**



On the next windows please create a new file.



Open Alchitry.xdc and add these lines:

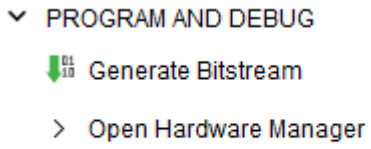
```
set_property PACKAGE_PIN N14 [get_ports clk_100MHz]
set_property IOSTANDARD LVCMOS33 [get_ports clk_100MHz]
set_property PACKAGE_PIN P6 [get_ports reset_rtl_0]
set_property IOSTANDARD LVCMOS33 [get_ports reset_rtl_0]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

set_property PACKAGE_PIN P15 [get_ports uart_rtl_0_rxd]
set_property PACKAGE_PIN P16 [get_ports uart_rtl_0_txd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_0_txd]
```

I guess these lines are not unfamiliar to you …

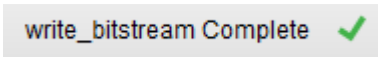
And done!

Generate Bitstream

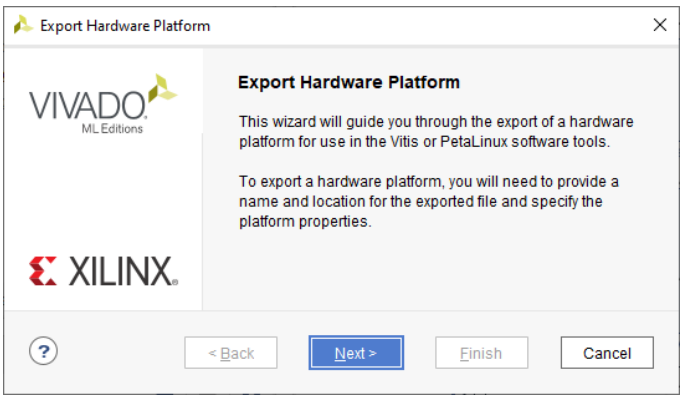
The last step in Vivado is to generate a bitstream. In the Flow Navigator click on “Generate Bitstream”



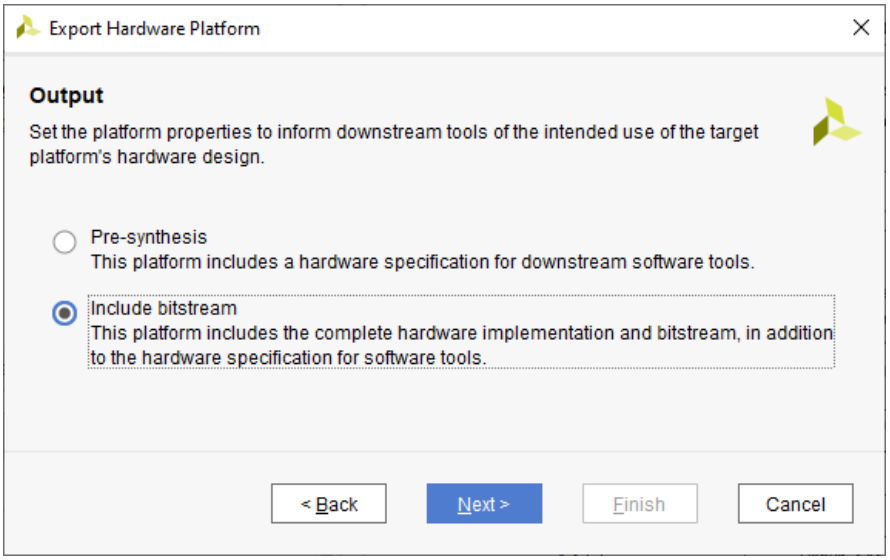
In the top right corner you can see what Vivado is doing at the moment. Synthesis, Implementation, … and after some minutes it will state “write_bitstream Complete”.



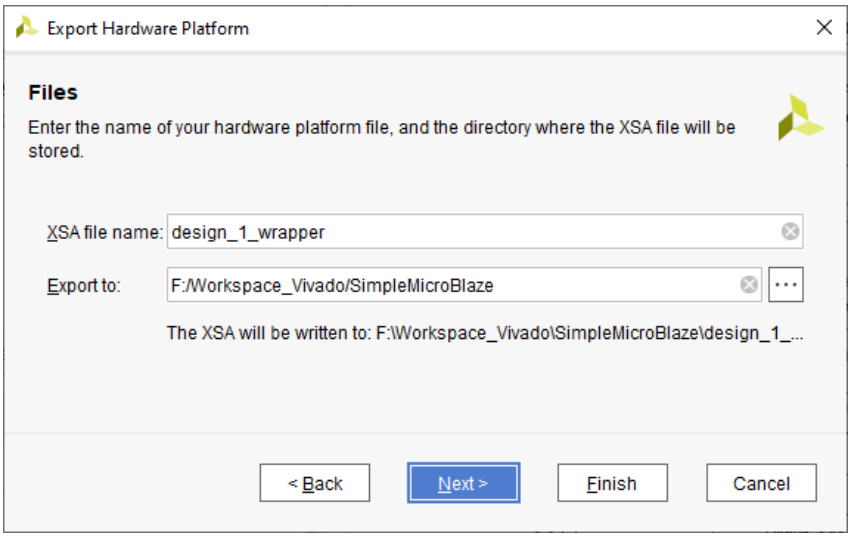
Now, we have to export our hardware design, so that we can use it in VITIS. Click on **File->Export->Export Hardware Platform**.



Select “Include bitstream”.



On the next page you can see were the XSA file will be stored.



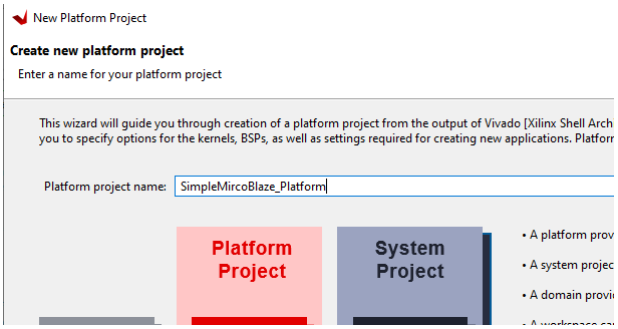
Click on Finish

Done!

Part two: Writing the Program in VITIS

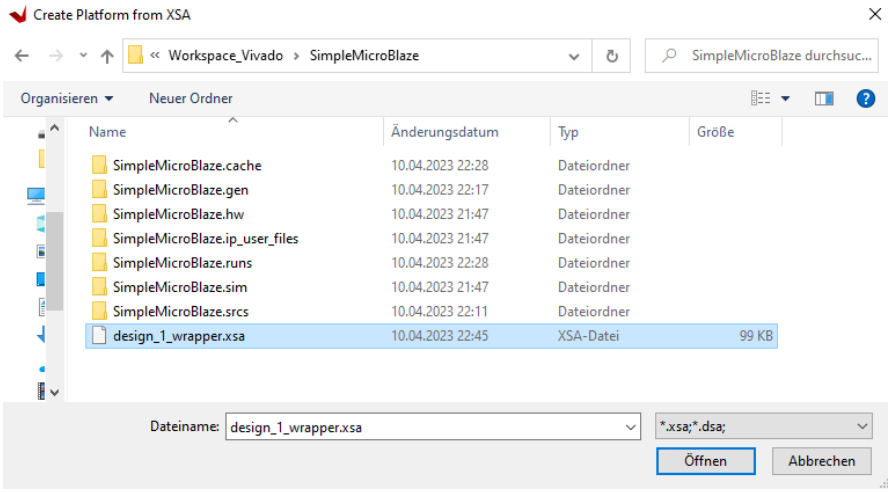
Create a Platform Project

First we have to create a Platform Project. Open Vitis and click on **File->New->Platform Project…**



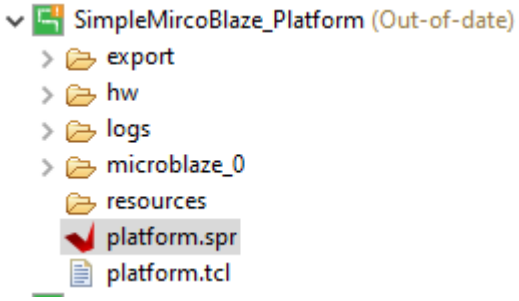
Click Next

On the next dialog we click on “Browse” and navigate to our XSA-File



Click on “Finish”

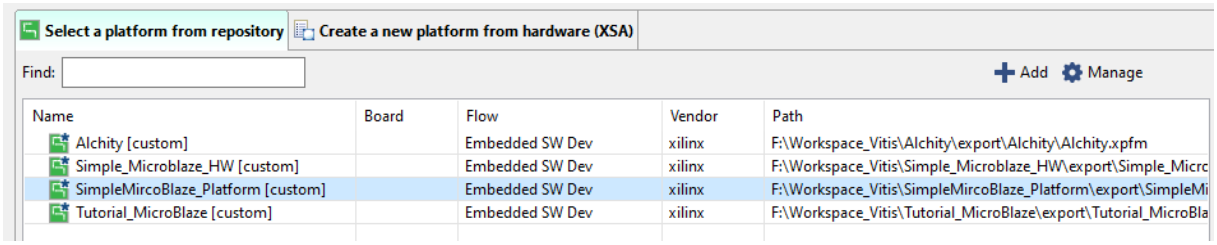
In the Explorer section, we can see our Platform project.



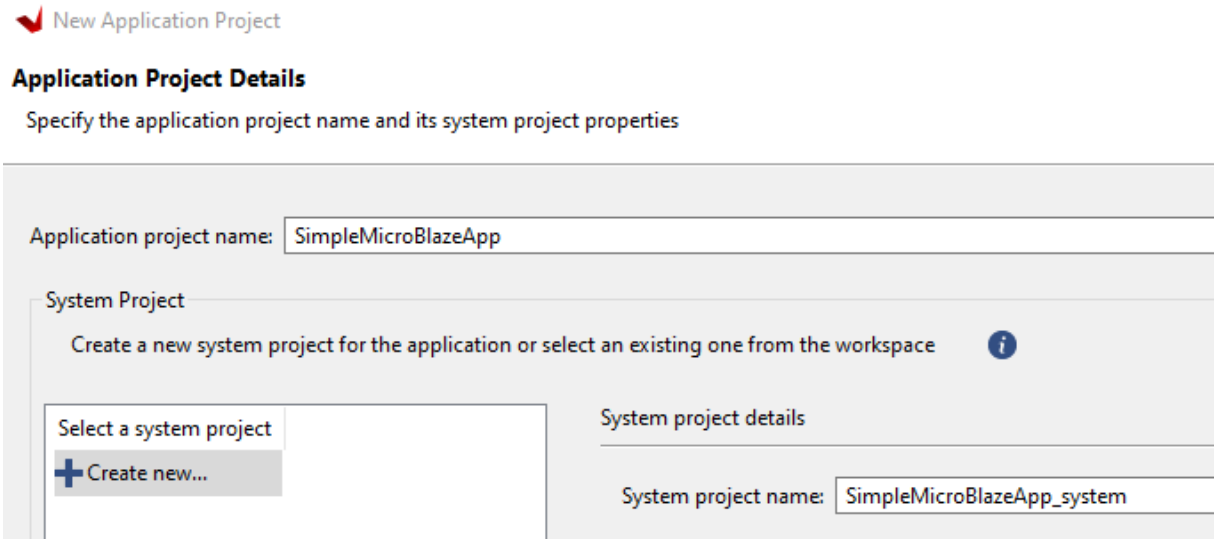
It is out of date. So select it, and built it. **Project->Build Project**.

Create an Application Project

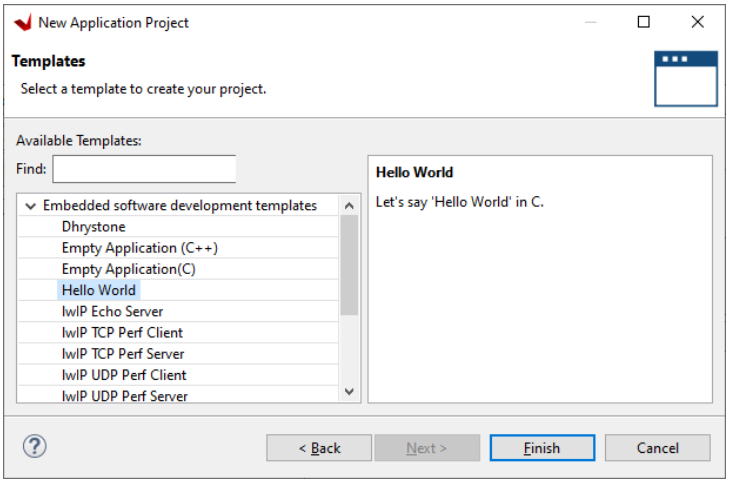
Second, we need our application project. Click on **File->New->Application Project…**
On the second dialog, we select our Hardware Platform:



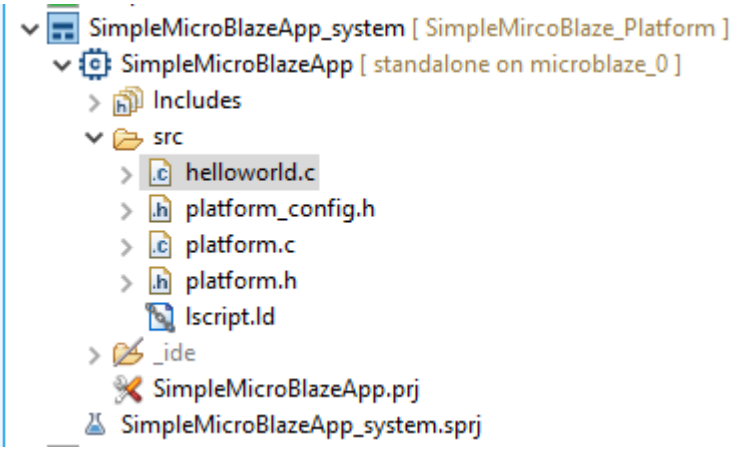
On the next dialog, we give it a name:



On the last Dialog, we select the template: “Hello World” and click on Finish.



The helloworld.c file is located under src:



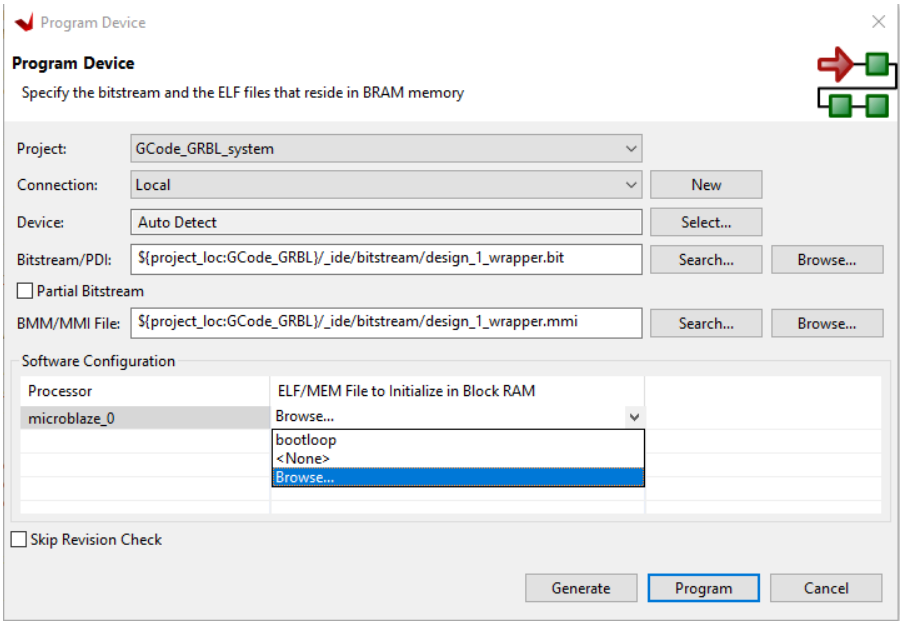
As you can see this is already a working program.

```
47
48 #include <stdio.h>
49 #include "platform.h"
50 #include "xil_printf.h"
51
52
53 int main()
54 {
55     init_platform();
56
57     print("Hello World\n\r");
58     print("Successfully ran Hello World application");
59     cleanup_platform();
60     return 0;
61 }
62
```

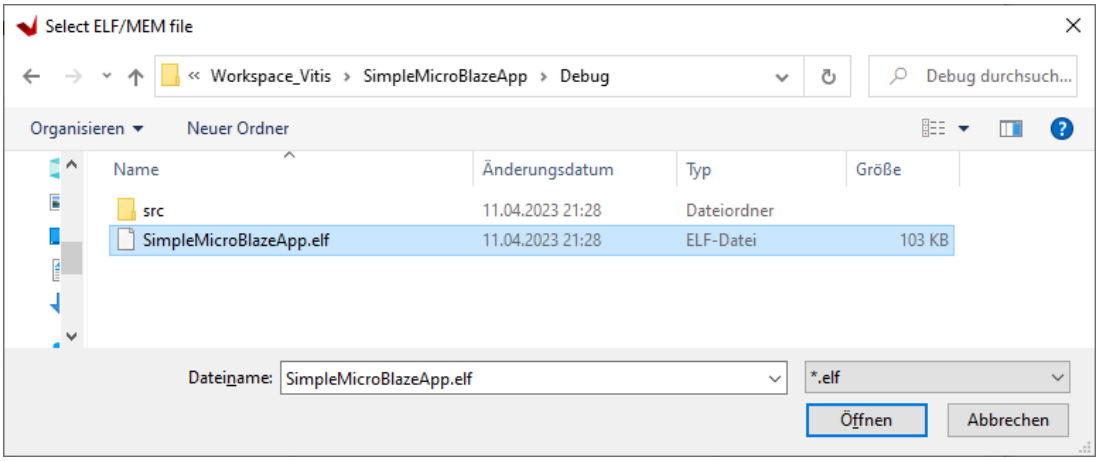
Built it.

Part three: Loading the design and the program into the flash

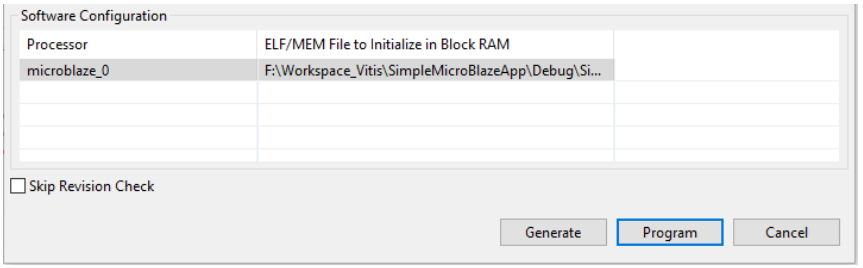
Now that we have our hardware design and our application software, we have to create the binary file which we can upload to our FPGA.
Still in Vitis and still our application project selected click on **Xilinx->Program Device.**
A dialog opens. In the Software Configuration-Section, open the dropdown menu and click on "Browse…" (Sometimes there is a double click necessary, otherwise the browse window will not appear ... a bug?)



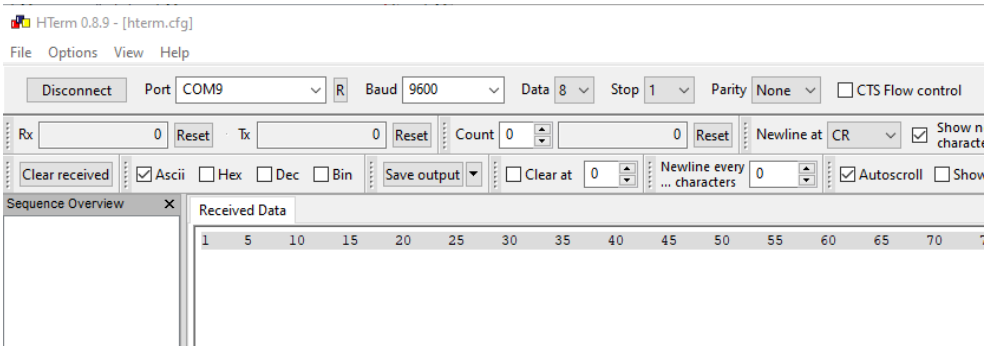
Navigate to the elf-File and open it:



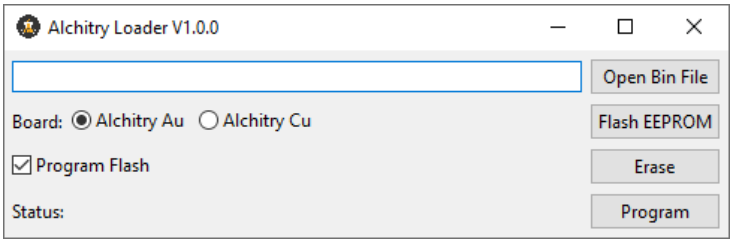
Click on "Generate"



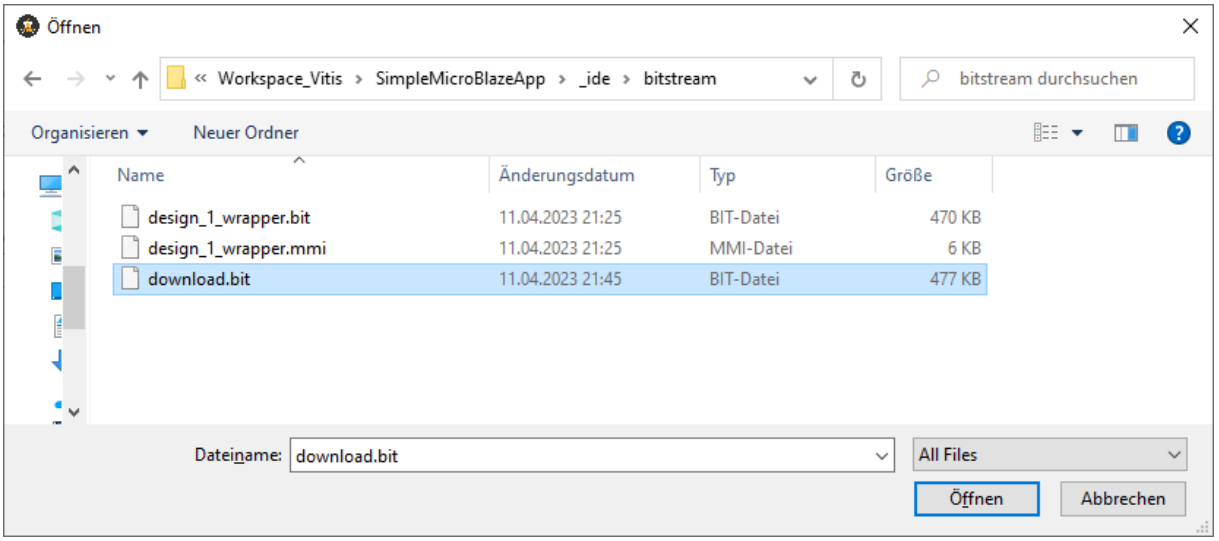
Now our bitstream is ready to upload into our FPGA. So connect your Alchitry-Board to your computer. But before we upload it, let us start our terminal program and connect to our Alchitry-Board. As you have seen in helloworld.c it will only send once "Hello World". I do want to miss it.



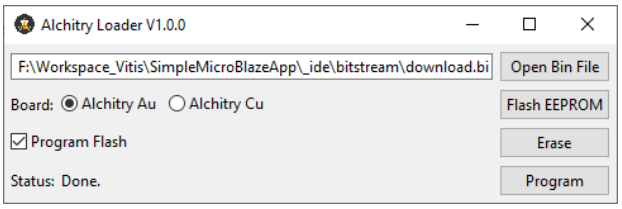
Now we are prepared to see the “Hello World”. Open Alchitry Loader.



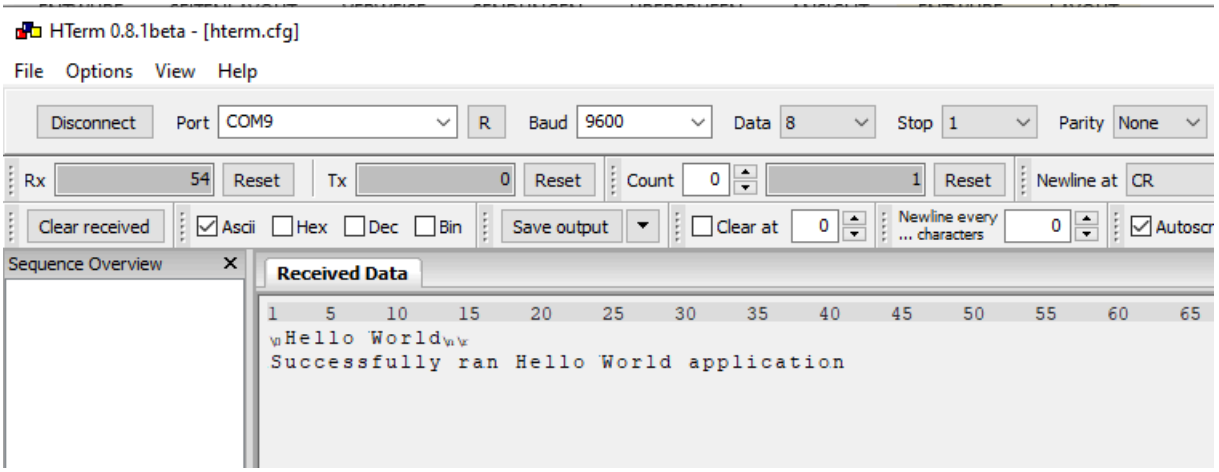
Click on “Open Bin File” change to “All Files” and navigate to download.bit.



Click on Program:



And …..


HTerm 0.8.1beta - [hterm.cfg]
File Options View Help
Disconnect Port COM9 R Baud 9600 Data 8 Stop 1 Parity None
Rx 54 Reset Tx 0 Reset Count 0 1 Reset Newline at CR
Clear received Ascii Hex Dec Bin Save output Clear at 0 Newline every ... characters 0 Autoscr
Sequence Overview
Received Data
1 5 10 15 20 25 30 35 40 45 50 55 60 65
\n\rHello World\n\r
Successfully ran Hello World application

Success!!