Instructions for interfacing an FPGA-based control system with GEMS/sGEMS

Andrea Suardi
a.suardi@imperial.ac.uk

August 23, 2013

The aim of this document is to describe step by step how to build an interface for an FPGA-based controller with the Generic EMBOCON Minimal Supervisor (GEMS) [3].

As example, the FPGA source code of the interface as well a MPC controller implementation (Fast Gradient) for controlling the crane experimental setup is available into the git tree folder [4]. Details of its directory listing are reported in Table 1.

The example files provided are compatible with Xilinx evaluation module ML506 [7], Xilinx software ISE V14.6 [9] and MATLAB R2012b [6]. However, the instructions reported below remain valid for any firmware versions any Xilinx evaluation module which supports an Ethernet connection.

Table 1: Directory listing

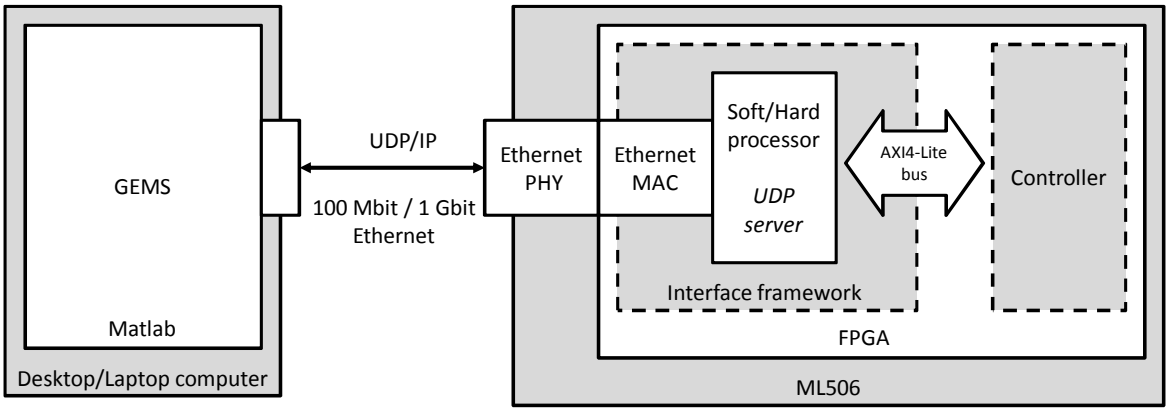
`Main directory`	GEMS files for FPGA communication.
`XPS_FPGA_Hardware`	Xilinx Platform Studio (XPS) [2] project used to build the interface framework module and interfacing it with the controller module.
`XPS_Controller`	Source VHDL code of the controller module. It implements the Fast Gradient algorithm used for controlling the crane setup using fixed-point arithmetic (18 bits words length, 8 bits fraction length).
`SDK_FPGA_Software`	Software Development Kit (SDK) [2] project used to design embedded software application which runs upon the soft processor. Include also the source code for UDP echo server to bridge communication to controller module for the for controlling the crane setup.
`Crane_Standalone`	Standalone Matlab-based interface to the FPGA (it does not requires GEMS)
`Documentation`	This documentation source code.

Contents

1 Introduction

The FPGA-based control system is interfaced with GEMS via an UDP/IP packets based communication infrastructure. Every time GEMS' optimizer function (MAKEOPTSTEP) is called, a sequence of data embedded in the payload of a UDP packet is transmitted over Ethernet to the FPGA. Once the FPGA has completed its computation it returns the control action embedded in another UDP packet.

The figure below



shows a schematic representation of interface.

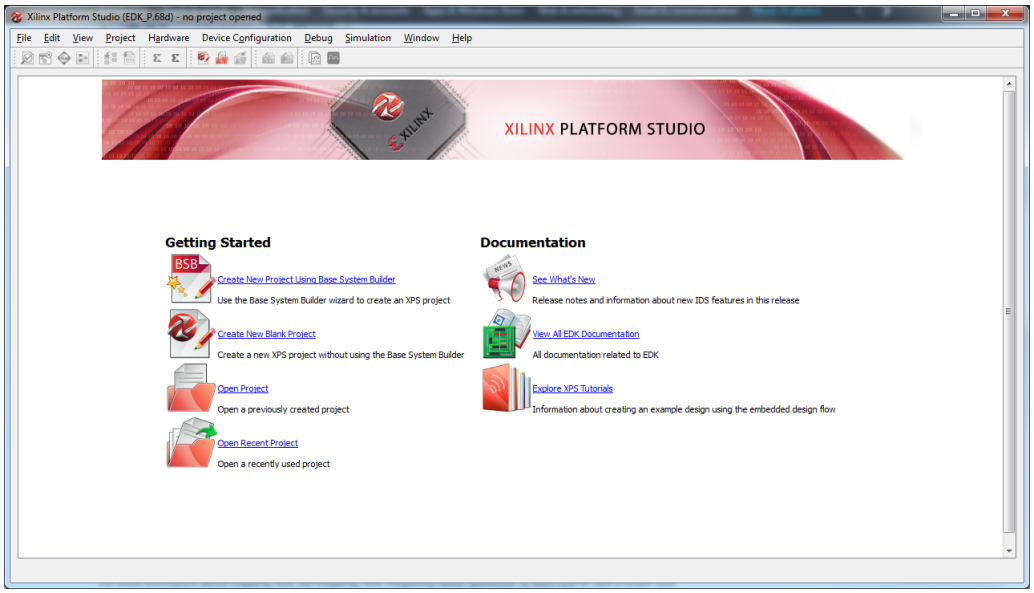
On the FPGA side two hardware modules are required: the interface framework and the controller.
The interface framework is based on Xilinx MicroBlaze soft core processor [10], upon which a software application, based on the *lwIP* UDP server [5], bridges the communication between the Ethernet interface and the controller. The choice of this interface framework architecture has been driven by the fact it provides some system flexibility in comparison to a dedicated custom interface allowing easy portability to other standard interfaces, e.g. SpaceWire, CAN bus, etc.. However this flexibility come with a small increase in FPGA resource usage and communication delay
The controller module is a custom designed circuit implementing the control algorithm. It can be directly coded by hand with a low level Hardware Description Language (HDL), i.e. Verilog or VHDL, or can be coded using an high level language, i.e using C, and translated into hardware using dedicated software such as Vivado HLS [8].
The designed controller module communicates with the processor via an AXI4-Lite bus [1] based on 4 memory registers each of 32 bits size.

The remain part of the document is organized as follow: Section 2 shows how to design the interface framework module from scratch and connect to it a custom designed controller module, Section 3 describes how to build the embedded software application which runs upon the soft processor, Section 4 describes how to communicate with the FPGA and Section 5 shows how to port the provided example design interface to any other application.

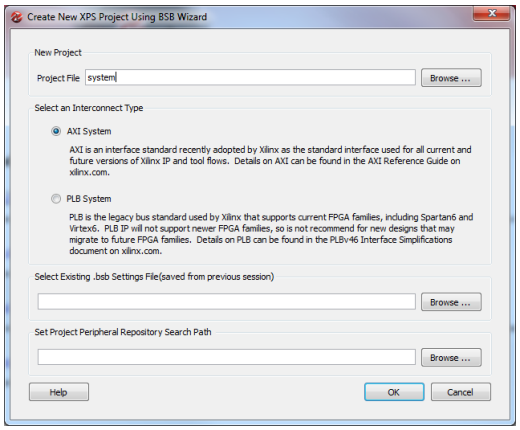
2 XPS: building the interface framework module and interfacing it with the controller module

2.1 Building the interface framework module

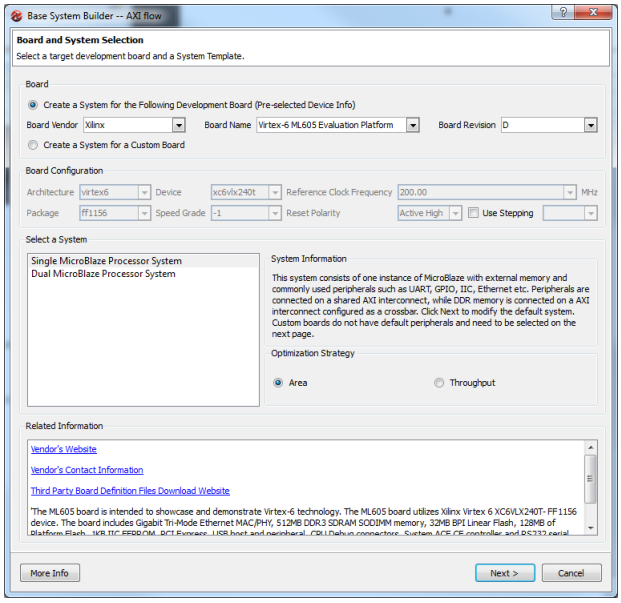
1. Run Xilinx Platform Studio (XPS) tool suite [2] and create a new project using base system builder



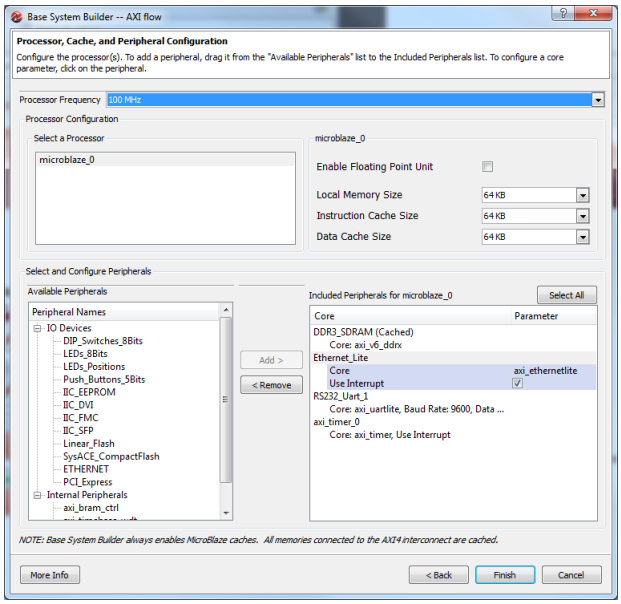
2. Assign a project name and select as interconnect type “AXI System”



3. Choose the development board you have got on available, i.e. ML605, select ”Single MicroBlaze Processor System” and optimization strategy for “Area” or “Throughput”



4. Increase the Local Memory Size , instruction cache size and data cache size to (e.g.) 64 KB, add the “axi_timer” peripheral to the list of included peripherals, and ensure that “axi_timer” and “axi_ethernetlite” use interrupts. Do **NOT** remove the “RS232_Uart” and ” DDR3_SDRAM” peripherals. Press “Finish”.

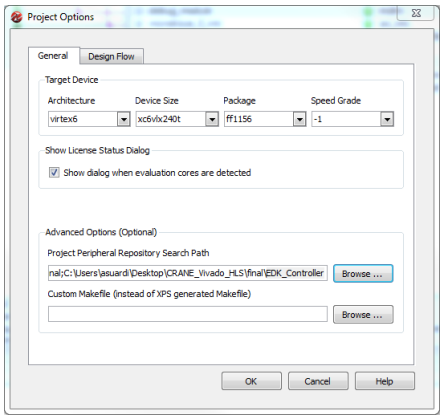


2.2 Interfacing the interface framework module with the controller module

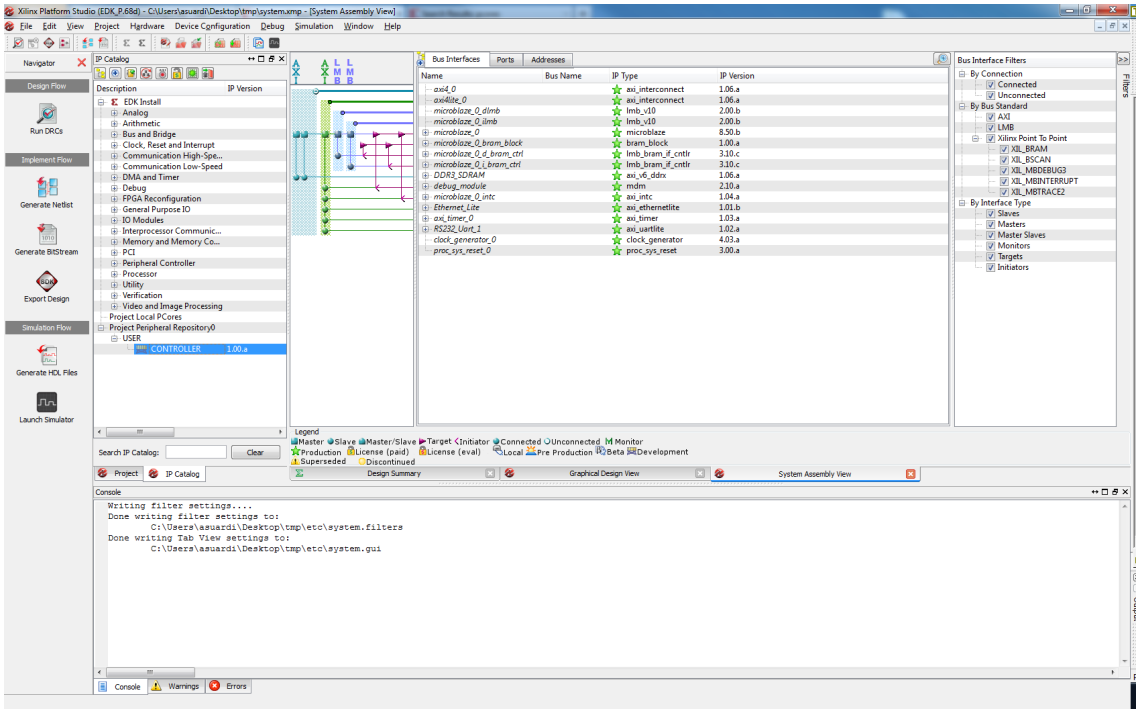
5. In the XPS Project, choose ”Project → Project Options”, and add the “EDK_controller” directory to the “Project Peripheral Repository Search Path”.
Note that the controller module has to be in the Xilinx *pcores* or *processor IPs* [2] format and with a AXI4-Lite bus interface exposing 4 memory registers each of 32 bits size. Their use is summarized in the Table 2.

Table 2: Controller module registers use

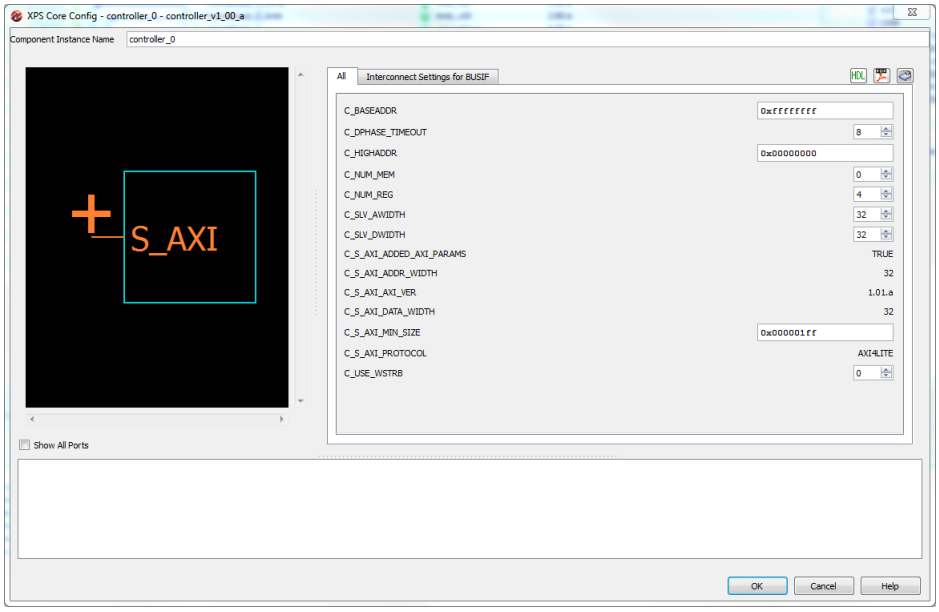
`Reg_0`	When all the data have been passed to the controller module, the processor tells to the controller module to start the computation.
`Reg_1`	The processor writes the data to pass to the controller module one after an other.
`Reg_2`	The controller module communicates to the processor that the computation has been completed, thus the processor can start reading back the results of the computation.
`Reg_3`	The processor reads back results of the computation from the controller module one after an other.



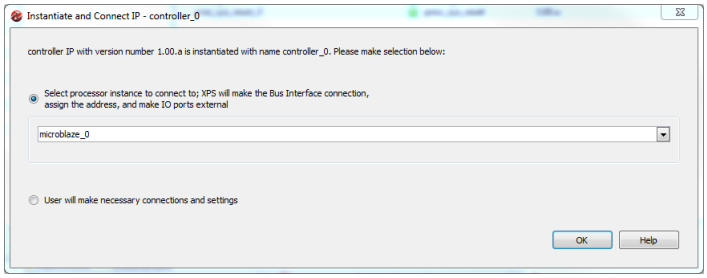
6. Add the controller module from the USER project peripheral repository double clicking on it in the IP catalog



7. Change “C_NUM_MEM” to “0”, and “C_NUM_REG” to “4” and press OK.

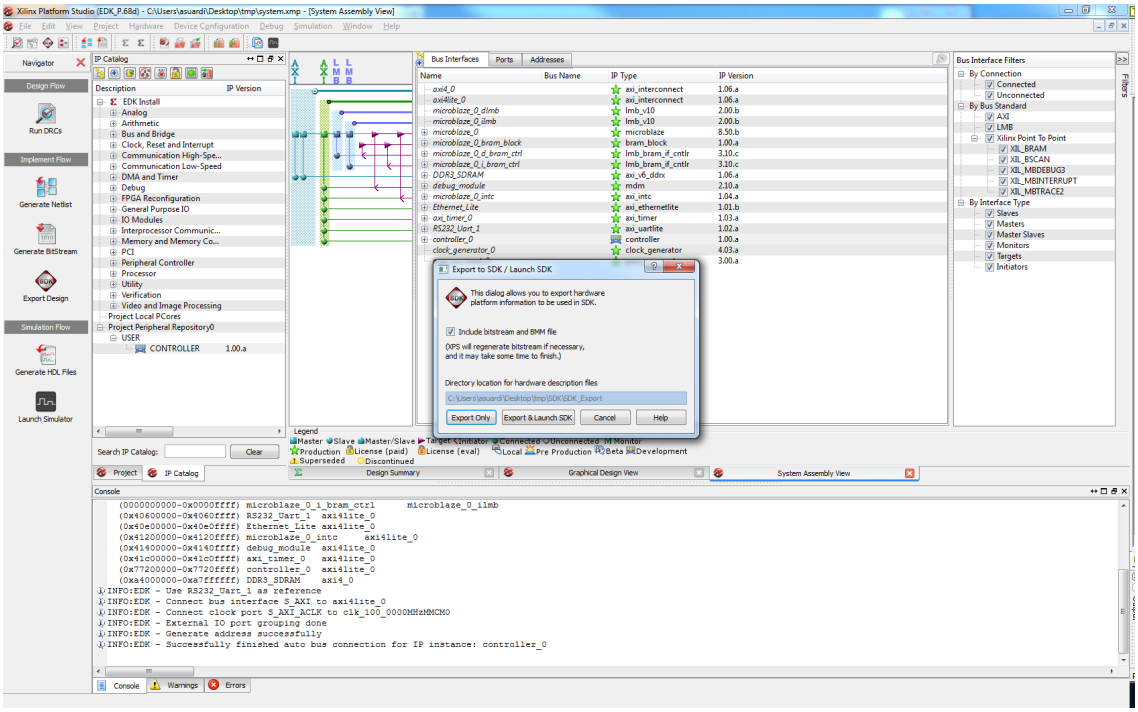


8. Let the XPS connect this to “microblaze_0”.



Now the FPGA circuit architecture has been built, it has to be compiled to generate the FPGA configuration (*bitstream*)

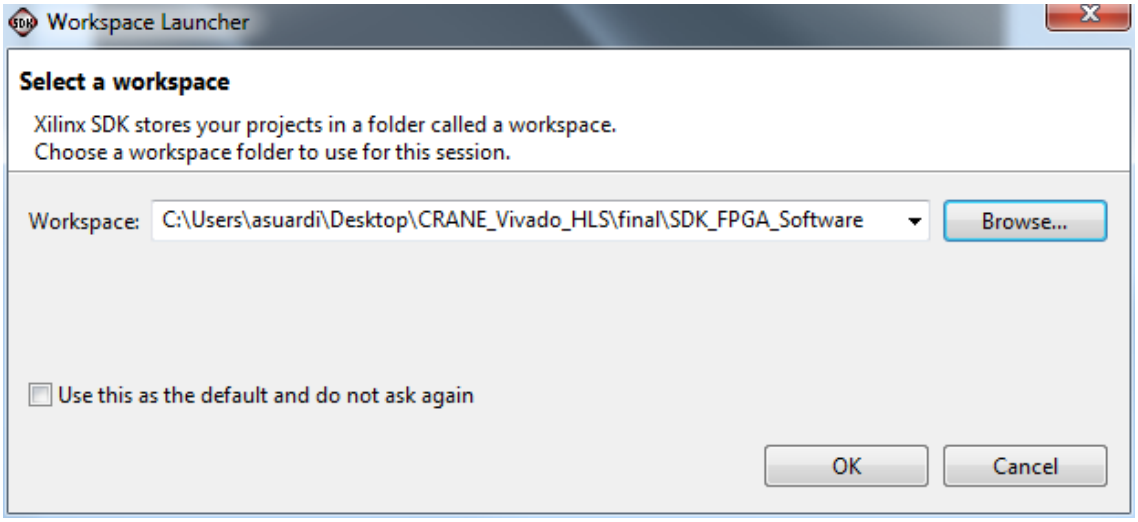
9. In the XPS Project, choose ”Project → Export hardware design to SDK” and select ”Export & launch SDK”. Wait a long time (also a few hours).



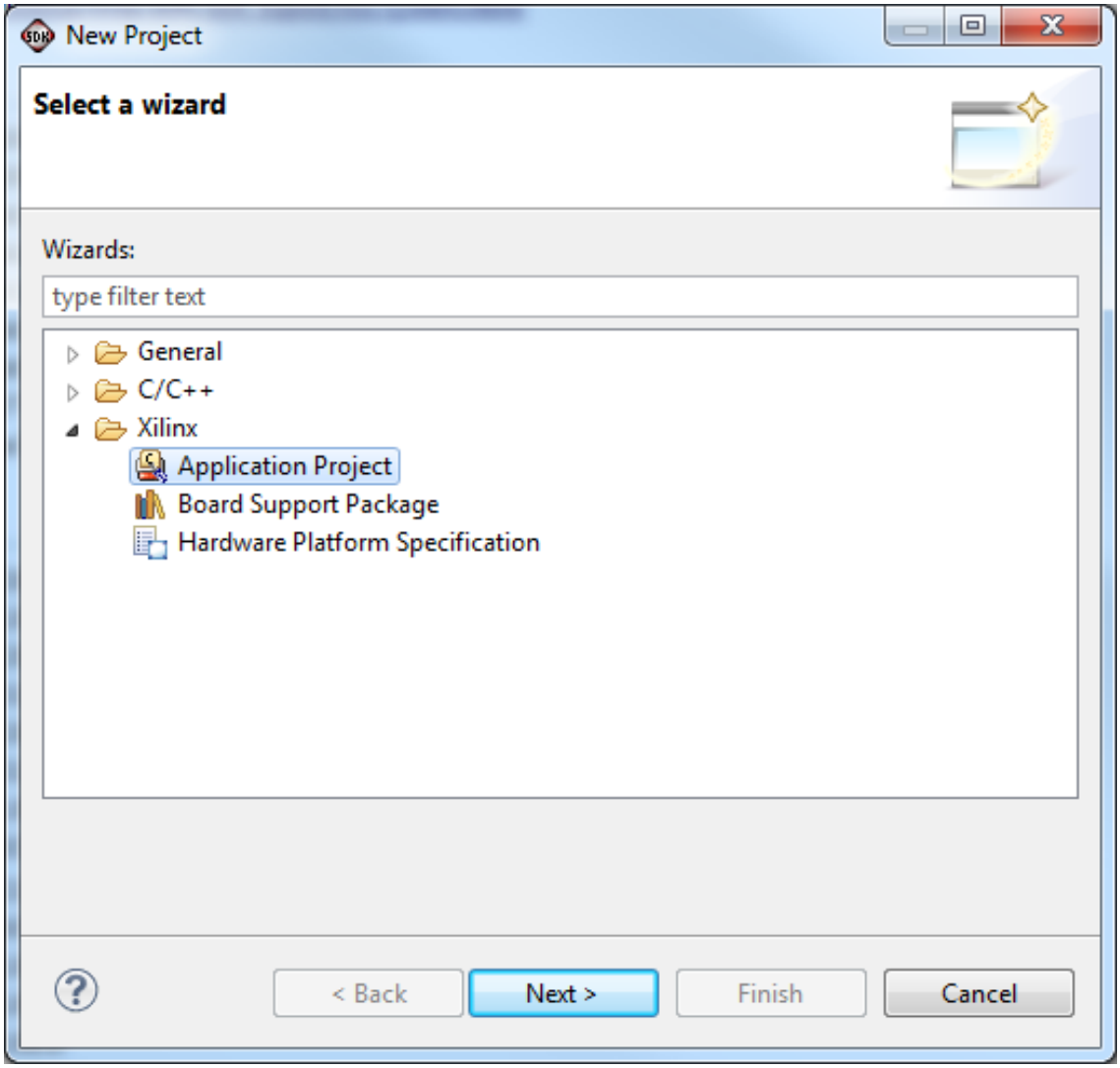
3 SDK: building the embedded software application

The system requires an embedded application to be run on the Xilinx MicroBlaze soft-core to act as a server application, which listens on the Ethernet port of the ML605 evaluation board and brings the data to the controller module.
The application source code is available in the ”SDK_FPGA_Software\controller_server_0\src\echo.c” and ”SDK_FPGA_Software\controller_server_0\src\main.c” files.

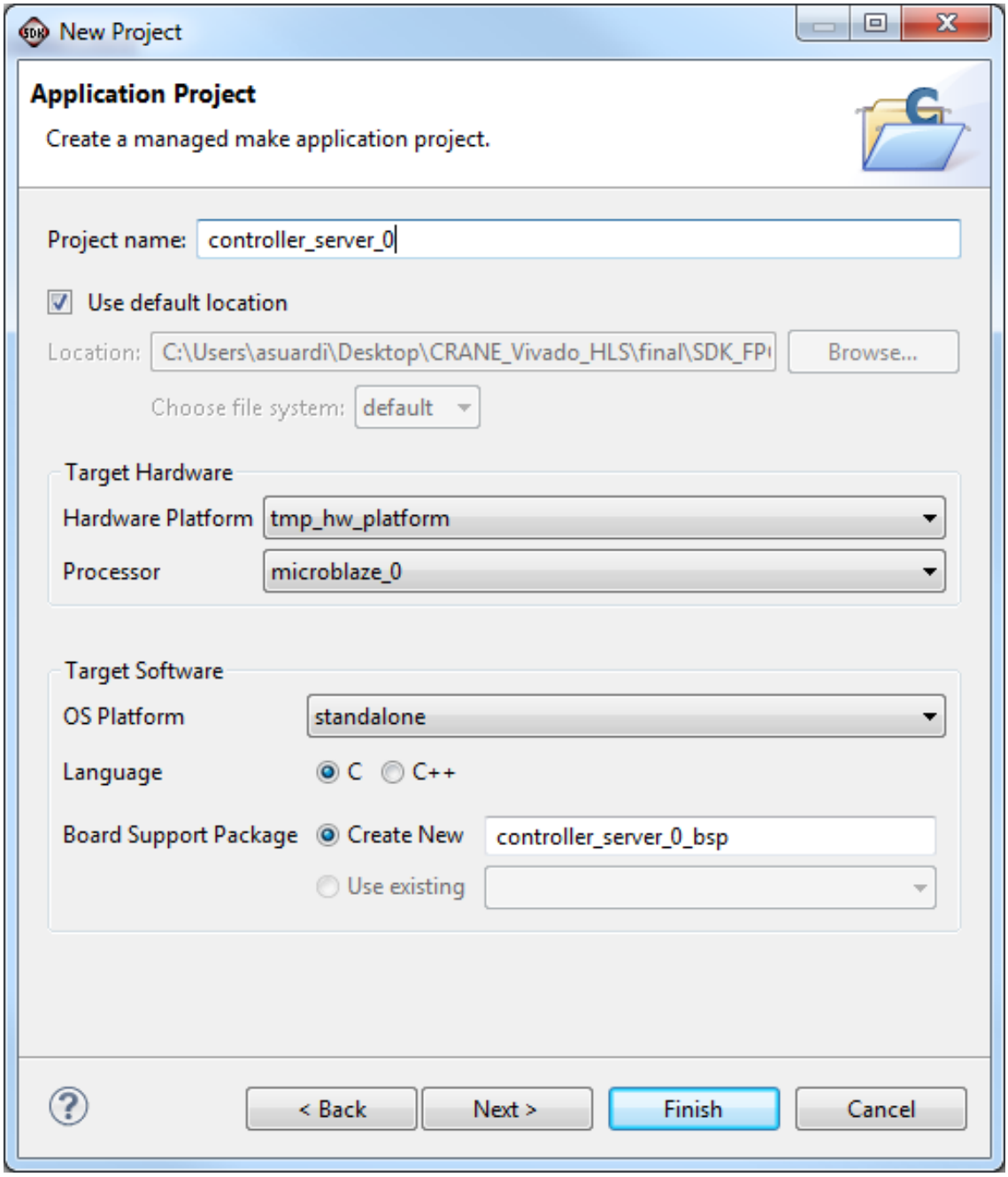
1. In the workspace launcher select the destination folder that is called workspace by SDK



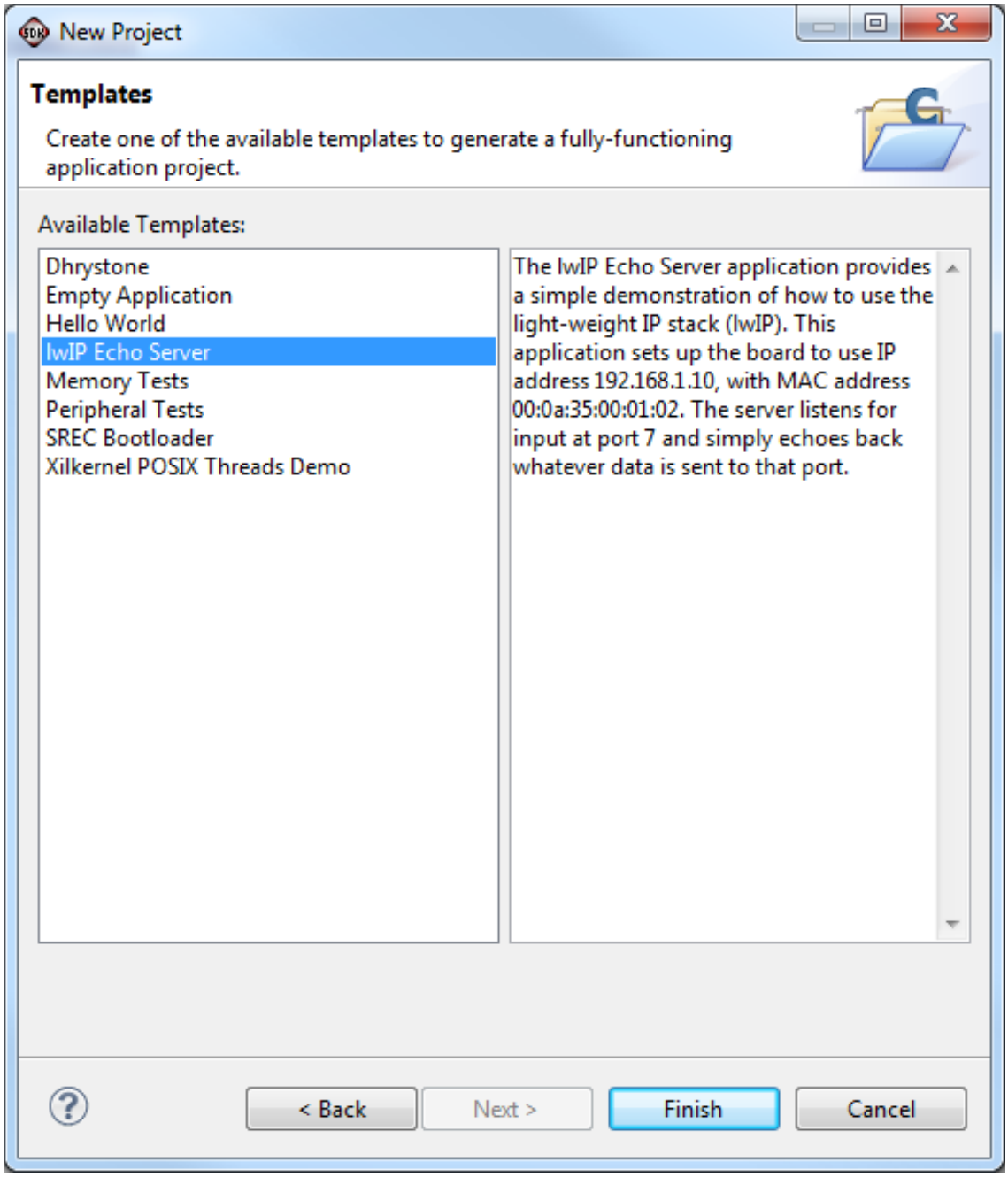
2. In the workspace create a “New Application Project” (File → New → Project → Xilinx → New Application Project).



Named “controller_server_0”.



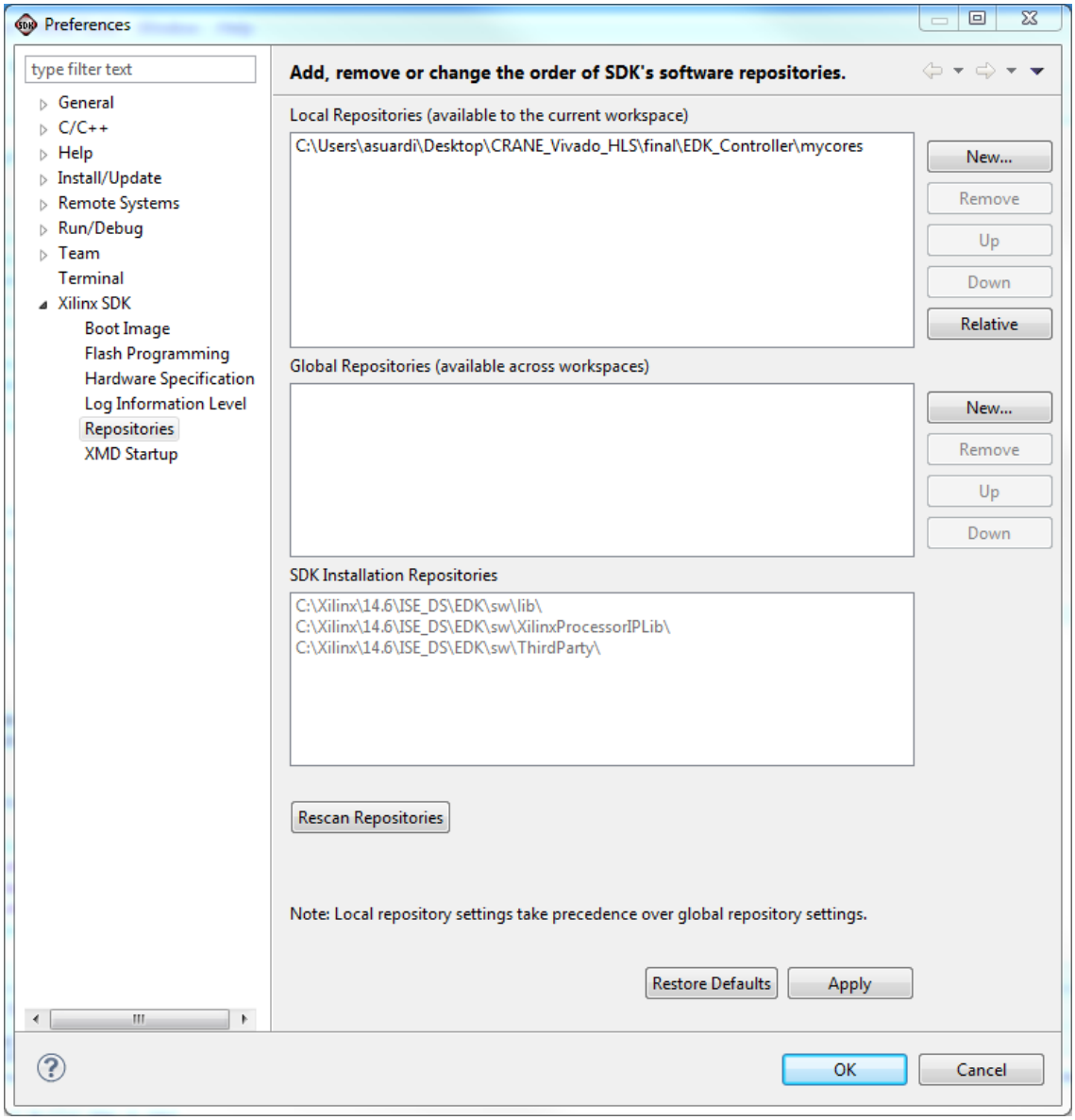
Use the “lwIP Echo Server” template project.



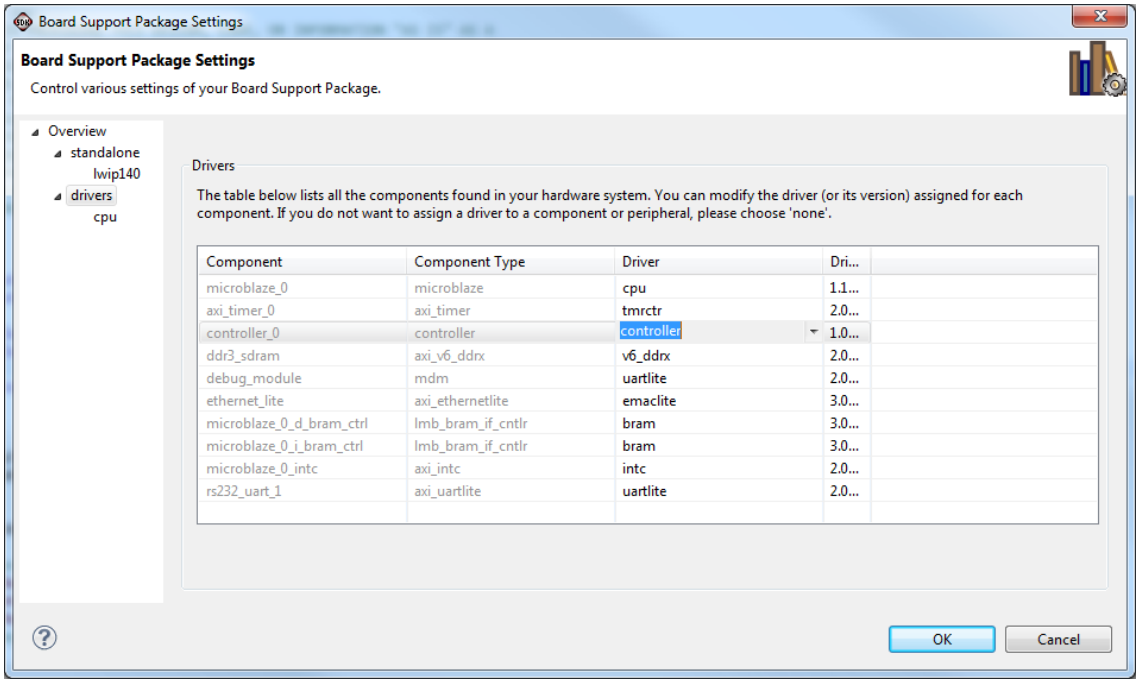
3. Copy “echo.c” and “main.c” from ”SDK_FPGA_Software\controller_server_0\src”’ folder into the project, replacing those automatically generated by the Xilinx SDK.

4. In the workspace run Xilinx Tools → Repositories

   Add the drivers for the controller module core drivers path (”EDK_Controller\mycores) to the local driver repository search path.



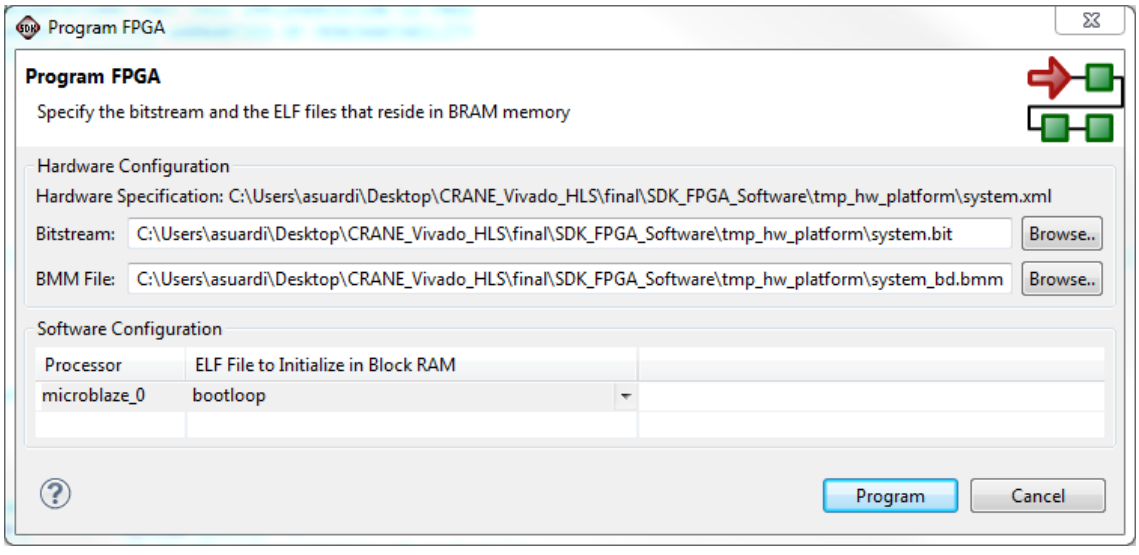
5. In the workspace run Xilinx Tools → Board Support Package Settings → controller_server_0_bsp. Change the drivers for “controller_0” from “generic” to “controller”.



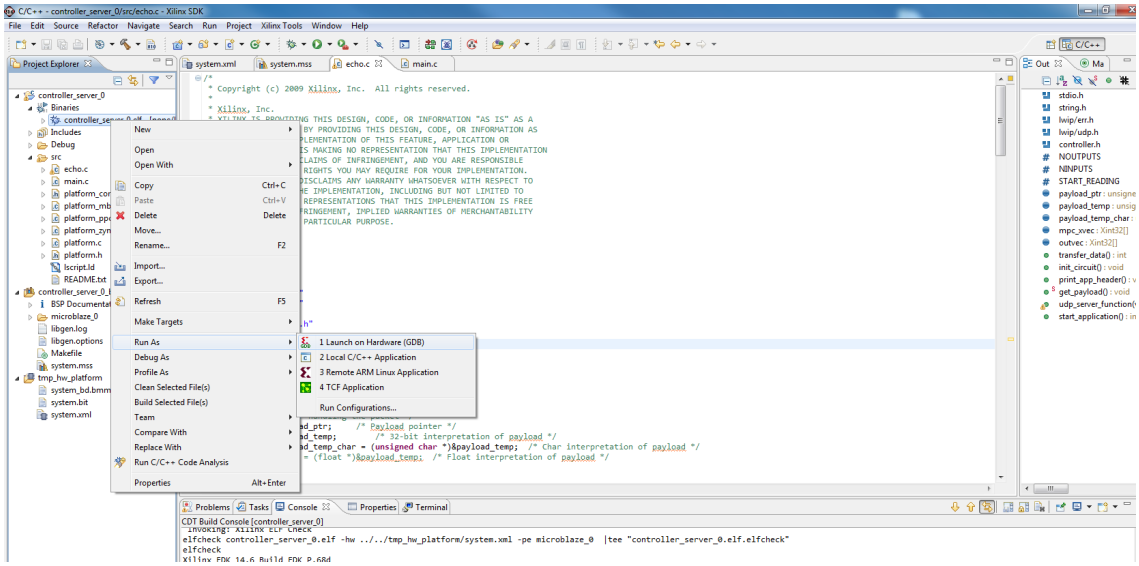
6. The SDK project should now build without errors.

3.1 Running the embedded software application on the FPGA

7. Making sure that the Xilinx evaluation module (ML605) is connected via the UART and JTAG USB cables, run Xilinx Tools → Program FPGA.



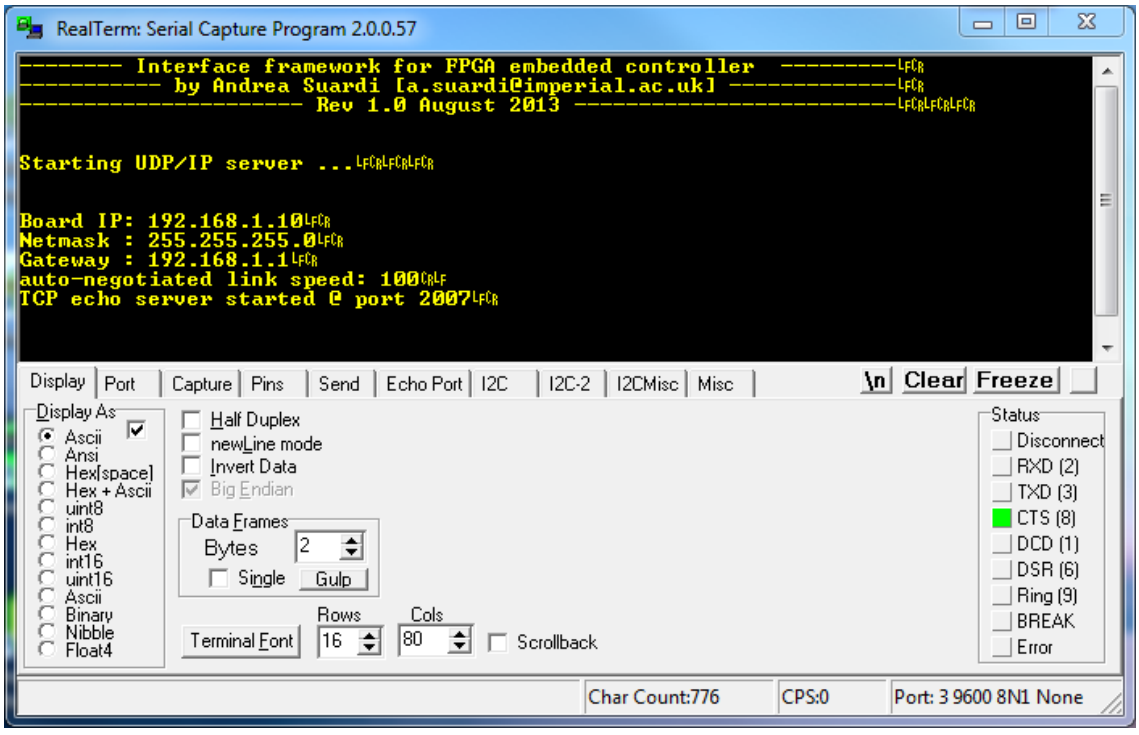
8. Start the software component running on the MicroBlaze.



Now the FPGA is programmed and ready to compute a new the control action every time it receives data from GEMS via the Ethernet interface.

Remarks

1. It is possible to monitor the status of the UDP server that is running on the FPGA by connecting a serial terminal to the serial port connected to the UART on the ML605. Set the Baud Rate to 9600, Parity to None and Data Bits to 8.



2.

4 Interfacing with FPGA

GEMS invokes the controller running on the FPGA by calling the function UDPCLIENT_COMBINED available in the library "UDPclient_int32_library.c".
Every time it is called, the controller's current state XCURR is encapsulated into an UDP/IP packet and sent to the FPGA. The function returns UCURR, the control action.

It should be noticed that the supported communication data formats are the floating-point single precision and any fixed-point precision up to 32 bits word length. The data format should be chosen according to arithmetic used for the FPGA implemented controller.
When a fixed-point precision is selected, the transmitted data is scaled by a factor of $2^{fraction\ length}$ and received data is scaled by $2^{-fraction\ length}$ where $fraction\ length$ is the fixed-point number fraction length.

Remark

A Matlab-based standalone interface is also provided with the FPGA for the crane application in the folder CRANE_STANDALONE.
A user guide called "readme.txt" can be found in the same directory.

5 How to port the provided example interface framework to any other applications

The provided example files have been tailored to control the crane experimental setup with the Fast Gradient algorithm. To reuse the interface framework with any other controller and application type, the following steps should be taken::

- Controller module design should be made according Section 2 - point 5.
- The following changes to the embedded application ("SDK_FPGA_Software\controller_server_0\src\echo.c"), which runs on the embedded microprocessor, have to be done:
  - **#define NINPUTS 17**
    set the number of data inputs the controller module requires
  - **#define NOUTPUTS 2**
    set the number of data outputs the controller module requires
- The following changes to c-based application ("FPGAcontroller_library.h"), which runs on the desktop/laptop computer, have to be done:
  - **#define N_INPUTS 17**
    set the number of data inputs the controller module requires
  - **#define N_OUTPUTS 2**
    set the number of data oputputs the controller module requires
  - **#define FLOAT_FIX 1**
    set according to the controller module arithmetic: **0** if floating-point single precision, **1** if fixed-point (up to 32 bits word length)

- **#define SCALE_INT32 8**
  set the number the fraction length number of bits if fixed-point arithmetic is used

References

[1] Axi interconnect documentation. www.xilinx.com/products/intellectual-property/axi_interconnect.htm.

[2] Embedded system tools reference manual. www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/est_rm.pdf.

[3] GEMS documentation. www.embocon.org/index.php/Category:GEMS.

[4] Github public repository. www.github.com/EMBOCONcs/EMBOCON_Interfaces/tree/master/Examples/sGEMS/crane_fpga.

[5] Lightweight ip (lwip) application examples. www.xilinx.com/support/documentation/application_notes/xapp1026.pdf.

[6] Matlab documentation. www.mathworks.co.uk/.

[7] ML506 evaluation board documentation. www.xilinx.com/products/boards-and-kits/HW-V5-ML506-UNI-G.htm.

[8] Vivado high-level synthesis documentation. www.xilinx.com/products/design-tools/vivado/integration/esl-design/index.htm.

[9] Xilinx ISE design suite (system edition) documentation. www.xilinx.com/products/design-tools/ise-design-suite/index.htm.

[10] Xilinx microblaze soft processor documentation. www.xilinx.com/tools/microblaze.htm.