Tutorial

XFT35-02

Hardware implementation of fuzzy controllers with xfsg

(ISE 14.7)
©IMSE-CNM 2018

Copyright (c) 2018, Instituto de Microelectrónica de Sevilla (IMSE-CNM)

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the IMSE-CNM nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AUTHOR:

Santiago Sánchez Solano santiago@imse-cnm.csic.es
Instituto de Microelectrónica de Sevilla (IMSE-CNM)
C/ Américo Vespucio s/n, PCT Cartuja.
41092-SEVILLA, SPAIN
# Table of Contents

- **Introduction**
  - Configuration ................................................................. 7
  - FPGAS and development boards ........................................... 7
  - Files .................................................................................. 8
  - Design example...................................................................... 9

- **Xfsg tool**
  - Fuzzy system synthesis ....................................................... 11

- **Verification in MATLAB /Simulink environment**
  - Input/output behavior .......................................................... 15
  - Closed loop simulation .......................................................... 18
  - Fuzzy controller implementation ............................................ 20
  - Hardware co-simulation ......................................................... 23
    - Notes for ZedBoard and Zybo .............................................. 26
    - Simulation on a remote board .............................................. 27

- **Conversion into IP-module**
  - Exporting an IP-module from SysGen ..................................... 29
    - Notes for ZedBoard .......................................................... 40
  - Inclusion of MicroBlaze into a Simulink model ....................... 41
Introduction

The synthesis tool xfs (Xfuzzy to System Generator) included in the Xfuzzy environment allows to automatically convert the specification of a fuzzy system into a Simulink model that can be verified through simulation in the MATLAB environment and implemented in an FPGA using Xilinx’s DSP tools. Thus, the combined use of Xfuzzy, MATLAB modeling and simulation facilities, and Xilinx’s synthesis and implementation tools allows to apply a model-based design flow to accelerate the development of fuzzy controllers on FPGAs.
According to this design flow, the development of a control system is performed in two stages with different levels of abstraction. The first stage, which aims at the description and functional verification of the control system, is carried out with the help of various Xfuzzy facilities. The fuzzy control system is usually described as a hierarchical XFL3 specification, which can combine fuzzy modules (for function approximation or decision making tasks) and crisp modules (that implement general purpose arithmetic and logical operators). Knowledge bases are generated directly from the experience of an expert (xfedit) or by using numerical training data and identification (xfdm) and supervised learning tools (xfsl). Functional verification is carried out by analyzing the input/output relationship of the system (xfplot) and simulating its closed-loop behavior when combined with a model of the plant coded in Java (xfsim). Once the XFL3 control system specification has been validated, the hardware synthesis tool xfsg generates the files necessary to start the second stage.

The second stage of development consists in the execution, in the MATLAB environment, of the Xilinx tools that facilitate synthesis and implementation of the fuzzy control system on an FPGA. At this design stage the system is described by a Simulink model that combines different components of the XfuzzyLib library. The model of the control system can be built from scratch or using the files (.mdl and .txt) generated by xfsg. The use of diverse elements for generating and displaying signals that provide different Simulink toolboxes facilitates system verification at logical level, allowing analyzing in detail aspects such as the number of bits of precision or the type of arithmetic used in different system components. The inclusion of a Simulink model of the plant under control also allows verifying the closed-loop operation of the control system with the same level of detail. In both cases, the functionality of the fuzzy system is defined by the (.m) file generated by xfsg.

Once validated the model, the synthesis and implementation phase begins by selecting the desired option by means of the GUI of the System Generator block included in the model. Possible options include: translation into HDL code, generation of the configuration file for the FPGA (bitstream), and settings of the environment to perform hardware/software co-simulation. In the latter case, the final verification of the control system can be carried out by integrating, in a closed loop of co-simulation, the software model of the plant described in MATLAB and the controller hardware implementation on the FPGA. Implementation options also include alternatives to encapsulate the fuzzy controller as an intellectual property
module (IP-module) connectable to the MicroBlaze or ARM processors available in the different families of Xilinx FPGAs.

This tutorial describes the implementation of fuzzy inference systems on FPGAs following the previous design flow. The tutorial is divided into seven parts covering the following aspects of the process:

- Synthesis of fuzzy system by xfsg
- Verification of input/output behavior
- Closed-loop simulation
- Fuzzy controller implementation
- Hardware co-simulation
- Exporting IP modules from SysGen
- Inclusion of MicroBlaze in a Simulink model

Configuration

The figures and data that appear in the text correspond to the 14.7 version of ISE and R2011b and R2012b versions of MATLAB. The design flow in earlier versions of Xilinx tools is similar but results may be slightly different.

FPGAs and Development Boards

Designs included in this tutorial can be implemented on different families of Xilinx FPGAs. In particular, operation has been verified in the development boards shown in the following table.

<table>
<thead>
<tr>
<th>Board</th>
<th>FPGA Family</th>
<th>Device</th>
<th>Board-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan-3A Starter Kit</td>
<td>Spartan3A</td>
<td>xc3s700a-4fg484</td>
<td>S3A</td>
</tr>
<tr>
<td>Spartan-3E Starter Board</td>
<td>Spartan3E</td>
<td>xc3s500e-4fg320</td>
<td>S3E</td>
</tr>
<tr>
<td>Spartan-3A DSP 1800A Starter Board</td>
<td>Spartan-3A DSP</td>
<td>xc3sd1800a-4fg676</td>
<td>S3ADSP</td>
</tr>
<tr>
<td>Avnet Spartan-6 LX9 MicroBoard</td>
<td>Spartan6</td>
<td>xc6slx9-2csg324</td>
<td>LX9</td>
</tr>
<tr>
<td>Digilent Spartan-6 Atlys¹</td>
<td>Spartan6</td>
<td>xc6slx45-3csg324</td>
<td>Atlys</td>
</tr>
<tr>
<td>Spartan-6 SP605 Evaluation Platform</td>
<td>Spartan6</td>
<td>xc6slx45T-3fgg484</td>
<td>SP605</td>
</tr>
<tr>
<td>ZedBoard Zynq Evaluation and Development Kit</td>
<td>Zynq-7000</td>
<td>xc7z020-1clg484</td>
<td>ZB</td>
</tr>
<tr>
<td>Zybo Zynq-7000 ARM/FPGA SoC Trainer Board</td>
<td>Zynq-7000</td>
<td>xc7z010-1clg400</td>
<td>ZYBO</td>
</tr>
</tbody>
</table>

¹ To use the Atlys board in XPS it is necessary to define the Project repository:
C:\Xilinx\Atlys_BSB_Support_v_3_7\Atlys_AXI_BSB_Support\lib
Files

The tutorial provides the following directory structure:

C:\<Xfuzzy_Tutorials>\Tutorial_xfsg\Romeo_BW_H\Xfuzzy
  - BW_H.xfl
  - RomeoModelBack.class
  - config<i>.cfg
  - Backward.xfl
C:\<Xfuzzy_Tutorials>\Tutorial_xfsg\Romeo_BW_H\SysGen
  - Backward_tb_2011b.mdl
  - Backward_tb_2012b.slx
  - StopFcn_BW_H.m
  - Romeo4.mdl
  - Romeo4.jpg
  - StopFcn_R4.m
  - Closed_loop_2011b.mdl
  - Closed_loop_2012b.slx
  - StopFcn_CL.m
  - Zybo.zip
C:\<Xfuzzy_Tutorials>\Tutorial_xfsg\Romeo_BW_H\EDK
  - Backward_IP_2011b.mdl
  - Backward_IP_2012b.slx
  - Test_Backward.c
  - ZYBO_zynq_def.xml
C:\<Xfuzzy_Tutorials>\Tutorial_xfhdl\Romeo_BW_H\HDL
  - Backward_IP_2011b.mdl
  - Backward_IP_2012b.slx
  - Test_Backward.c

Configuration of the Zybo board for hardware co-simulation with SysGen:
- unzip the file in C:\Xilinx\14.7\ISE_DS\ISE\sysgen

Configuration of the Zybo board for use in XPS (New Blank Project).
Design Example

To illustrate the development flow of fuzzy systems with xfsf, a controller for reverse parking of an autonomous vehicle will be used as an example. The purpose of this controller is that the vehicle Romeo-4R reach the parking place (with the desired final configuration) based on optimal trajectories composed of arcs of minimum radius and line segments with the proper orientation, as it is shown in the figure.

![Diagram of parking paths](image)

The curvature provided by the inference module depends, in this case, not only on the variables $x$ and $\Phi$, but also on the variable $y$. The paths shown in the figure are difficult to obtain from the translation of heuristic knowledge of an expert driver. For this reason, the approach followed for obtaining then exploits the geometrical considerations of the problem to generate a set of training data to identify and adjust the rules of the inference module. From a geometric analysis of the maneuvers, it is concluded that there is a value of angle $\alpha$, dependent on $x$ and $y$, which determines the abrupt changes in the values of the curvature. It can be seen that this angle is what defines the orientation of the line segment containing the point $(x, y)$ of the initial position of the vehicle and is tangential to the arc of the circle which reaches the end point of the path. According to this idea, the controller has the structure shown in the next figure; it consists of two rule bases connected in series with a crisp block. System inputs are the position $(x, y)$ and orientation $(\Phi)$ of the vehicle, and its output is the value of the curvature of the trajectory of the vehicle in reverse displacement $(bw)$. The rulebase interpolation provides the approximate value of angle $\alpha$ depending on the input variables $x$ and $y$. smoothing rulebase provides the value of the curvature $(bw)$ depending on the difference $(\Phi-\alpha)$. This rulebase provides smoother transitions in path curvature to obtain

---

continuous curvature paths that can be followed by the vehicle without having to stop and without violating the dynamic constraints.

As shown in the next figure, the specification of the fuzzy system in the Xfuzzy environment uses two fuzzy inference modules and one subtraction operator to define the control strategy. It also uses another crisp block (delay) to synchronize the input phi with the output a of the interpolation block. The block corresponding to the rulebase interpolation is a first-order Takagi-Sugeno module with two inputs. The rulebase smoothing uses a single input (subtraction = Ф-α) and employs FuzzyMean as defuzzification method.

File ‘Backward.xml’ contains the specification of this fuzzy system.
A) Synthesis of the fuzzy system by means of xfsg

We will use as starting point for carrying out this exercise the specification file ‘BW_H.xfl’ located in directory ‘...\Romeo_BW_H\Xfuzzy’.

1) Open in Xfuzzy the file ‘BW_H.xfl’ (File → Load System) and analyze the different components (operators, types, rulebases and crisp blocks) using both graphical and textual description facilities provided by Xfuzzy.

2) Use xfplot to obtain the input/output graphics of the interpolation rulebase (alpha vs. x, y) and the control surface of the entire system (bw vs. x, phi).

3) Using xfsim and the model of Romeo-4R provided in ‘RomeoModelBack.class’ file, verify controller operation for the following initial conditions:
   - Path1: x= 0, y= 7, phi= 170, gamma= 0 (config1.cfg)
   - Path2: x= 6, y= 6, phi= -45, gamma= 0 (config2.cfg)
   - Path3: x= -10, y= 3, phi= 0, gamma= 0 (config3.cfg)
In describing and modeling the vehicle Romeo-4R and its navigation control system, ranges of universes of discourse of the linguistic variables (distances, angles, curvatures, etc.) take "true" values. To simplify microelectronic implementation, however, value ranges are normalized in the interval [0, 1]. For most inference systems this normalization is independent of the adjustment step. That is, the inference system can be adjusted using a training file with real data and inputs and outputs may be then normalized by applying corresponding scale factors. An exception to this rule is the case of first-order Takagi-Sugeno systems because, in this case, the values of the inputs appear explicitly in the calculation of the system output. As a result, prior to the synthesis of the control system it will be necessary to readjust the interpolation rulebase with a training file containing normalized values. (The result of this adjustment process is available in the file 'Backward.xfl' located in the same directory).

4) Load Backward specification and analyze changes made with respect to BW_H. In particular, check that the control surface of the entire system (bw vs. x, phi) is similar to that obtained in step 2, but now all variables are normalized. Generate the graphical input/output of interpolation block and observe that it also coincides with that obtained in step 2. Also check that functions from the xfsg package have been used in Backward specification. Using this specific package will ensure the viability of synthesis process of a specification in future versions of xfsg.

5) Select Backward specification and run the xfsg tool. To do this, use the (Synthesis → To Sysgen) menu or use the Compile to Sysgen icon in the Xfuzzy taskbar. Check the prefix and the location of output files that appear in the xfsg window.

- Open the rulebase and crisp block hierarchy to make visible all the components of the control system. Select interpolation rulebase and use the edit fields of the right side of the window to define the size of the buses. Set 10 bits for input and output variables, grades and slopes of the membership functions. Maintain unchanged the remaining parameters. Use the Apply button to store the definitions and verify that the icon at the right of the rulebase turns green.

- Define the values for smoothing rulebase (10, 10, 10 and 10) and subtraction (10) and Delay (10) crisp blocks. When all values have been defined, the icon at the right of the specification turns green and the buttons on the bottom of the window become active.
6) Before launching the tool, use the GUI to also define the following parameters and options:

**Section Files and directory information:**
- **Output directory:** `...\Romeo_BW_H\xfsüz\`

**Section Global Options:**
- Generate txt file
- Generate Simulink Model
- Use Simplified Components

- Use the **Apply** button to store the definitions and the **Save Configuration** button to save the settings in the file ‘Backward.xml’.

7) Press the **Generate Files** button and watch the messages displayed in the *Xfuzzy* main window indicating the generated files and a warning (WARNING 14) alerting that there is no element to implement the crisp block *Delay* in *XfuzzyLib* library.

- Use WordPad or Notepad++ to open ‘Backward.txt’ and verify that this file contains a structural description of the control system including inputs and outputs of each of its components, as well as the blocks of *XfuzzyLib* library implementing the different rulebases and crisp blocks (except the *Delay* block, which is associated with a null value as a component).
- Open 'Backward.m' with WordPad or Notepad++ and verify that it contains the definition, in the language used by MATLAB, of the different parameters that determine the behavior of the FLCs predefined in XfuzzyLib library.

- The file 'Backward_Aux.mdl', containing a Simulink model of the control system, will be used in the following stages of the tutorial.
B) Verification in MATLAB/Simulink environment

The System Generator environment (SysGen) facilitates the design of digital signal processing (DSP) systems on Xilinx’s FPGAs. SysGen is a software tool for the description and synthesis of DSP systems based on Simulink (the interactive tool for modeling, simulation and analysis of dynamic systems integrated in MATLAB). By the fact of having powerful tools of numerical analysis, MATLAB environment can handle a high level of abstraction for designing DSP and further provides a graphical environment that facilitates the description and simulation at different levels of the model to be implemented. Environment capabilities can be expanded through specific Toolboxes for different application domains. SysGen provides one of these specific Toolboxes that, once verified the functionality of the design, allows automating the translation process into a hardware implementation optimized in terms of area and speed. To do this, SysGen includes the Xilinx Blockset library for Simulink models, as well as the software required to convert the model of a system described in Simulink to different representations that facilitate its hardware implementation.

Input/output behavior

Before starting this part of the tutorial, copy ‘Backward.m’ and ‘Backward_aux.mdl’ files to ‘SysGen’ directory.

1) Start System Generator through the Windows menu or the ISE Design Suit icon and change the working directory to ‘...\Romeo_BW_H\SysGen’.

- Check that the MATLAB search path (File → Set Path ...) includes the directory containing XfuzzyLib library (‘C:\Xfuzzy\XfuzzyLib’).

- Click on ‘Backward.m’ file to open the MATLAB editor. Run the file (using Debug → Run Backward.m or F5) and analyze the data structures that are stored in the MATLAB Workspace.

- Open ‘Backward_aux.mdl’ file and verify that it corresponds to a Simulink model of the control system. Click twice on each of the blocks to open its parameter definition mask and check that they match the data stored in the Workspace.

(The Delay block is highlighted in red in the model because it does not correspond to an element of the XfuzzyLib library)
The next step is to create a Simulink model of the fuzzy control system designed with Xfuzzy. To complete the model, it is possible to start from scratch (an empty model) or to use the auxiliary ‘Backward_aux.mdl’ file generated by xfsg. For the development of the tutorial a middle course between these two alternatives will be followed.

‘Backward_tb_2011b.mdl’ and ‘Backward_tb_2012b.slx’ files contain Simulink models (compatible with 2011b and 2012b MATLAB versions, respectively) that incorporate the controller.

2) Open the model corresponding to the used MATLAB version. The file contains a hierarchical model, used to check the input/output controller behavior, which includes the System Generator block, Simulink blocks to generate stimuli and display responses, interface blocks (Gateway In and Gateway Out), and the block corresponding to the Backward control system.

- Open Backward block to complete the model according to the outline of the figure below, including the fuzzy rulebases interpolation and smoothing and the subtraction crisp block, by copying them from the auxiliary file generated by xfsg (‘Backward_aux.mdl’).

- To implement the Delay block, the supplied template includes a delay element available in the Xilinx Blockset library whose latency has been defined to provide a delay equivalent to that of the interpolation block (2 * pipe).
Once the model is completed, go to the top level of the hierarchy and save it in file ‘Backward_tb.mdl’ (‘Backward_tb.slx’ for MATLAB 2012b).

3) Run ‘Backward.m’ to initialize the system parameters. Define an appropriate value for variables $np$ (indicating the number of points on the graph) and $pipe$ (corresponding to the number of clock cycles of pipelined stages) and launch the simulation. Once completed, observe the surfaces that are generated after the execution of ‘StopFcn_BW_H.m’ and compare them with those obtained in the Xfuzzy environment.

4) Use the *File → Model Properties* command to set the following Callback functions:
   - *PreLoadFcn*: to run ‘Backward.m’ (Backward) and define variables $np$ ($np = 40$:) and $pipe$ ($pipe = 10$:).
   - *StopFcn*: to run ‘StopFcn_BW_H.m’ (StopFcn_BW_H).

5) Exchange $y$ and $phi$ inputs on the model and re-launch the simulation to obtain ($bw$ vs. $x$, $y$) and ($phi$ vs. $x$, $y$) surfaces. Compare again with Xfuzzy results to analyze similarities and differences.

   **Note:** Before each simulation is often desirable to clear the Workspace and reinitialize the parameter values.

---

The number of clock cycles required to complete the operation of a module depends on the fuzzy inference mechanism, the defuzzification method, and the options employed in its implementation. For more information see:

*Hardware implementation of embedded fuzzy controllers on FPGAs and ASICs*
Closed loop simulation

In addition to facilitating the verification of input/output behavior of the controller, the combined use of MATLAB/Simulink environment and a kinematic model of Romeo-4R can be used to check its operation when acting in closed-loop with the vehicle under control.

6) Open the ‘Romeo4.mdl’ file containing the vehicle model, two blocks defining the reference values of speed and curvature, and other elements to display and store output variables.

- By clicking twice on Romeo4R-D block, the mask used to define the parameters of this subsystem will be opened. These parameters correspond to the initial state of the system \((x, y, phi, gamma, v)\), the number of clock cycles required for each control cycle \((pipe)\), and the time constants for modeling the inertia of the traction and address motors.

- Use Look Under Mask option to view the contents of Romeo4R-D block. Analyze the model observing the relationship between the parameters assigned to the various elements and those captured through the block mask.

- Run the simulation and observe the outputs provided by the model: temporal evolution of wheel angle \((phi)\), evolution of the variables \(x\) and \(y\) \((x-y)\), and path followed by the vehicle (figure generated by the StopFcn_R4 function associated to the model).

- Perform other simulations for different values of system parameters, reference values, and simulation time.

Files ‘Closed_loop_2011b.mdl’ and ‘Closed_loop_2012b.slx’ in directory ‘SysGen’ contain Simulink models (compatible with MATLAB 2011b and 2012b, respectively) in which incorporate the controller.

7) To complete the closed-loop simulation model, from the model corresponding to the used version of MATLAB:

- Include the vehicle model by dragging the corresponding block from the Romeo4 model and connecting it properly.

- Include the model of Backward fuzzy controller by dragging the block from the ‘Backward_tb.mdl’ (‘Backward_tb.slx’ for MATLAB 2012b) model used in the
previous section. In this case, to properly connect the block will be necessary to first obtain its speculative image about the y axis. To do this, position the cursor on the block, press the right mouse button and select the Format → Flip Block option (for MATLAB 2011b) or the Rotate & Flip → Flip Block option (for MATLAB 2012b).

• Once completed, save the model in file ’Closed_loop.mdl’ (’Closed_loop.slx’ for MATLAB 2012b).

• Simulate the system (using the Simulation → Start command or corresponding icon) for the following initial conditions of the vehicle:
  - x= 0, y= 7, phi= 170, gamma= 0
  - x= 6, y= 6, phi= -45, gamma= 0
  - x= -10, y= 3, phi= 0, gamma= 0

After the simulation, the ’StopFcn_CL’ script defined in the Simulink model callback function will be executed.

• Compare the results with those obtained with Xfuzzy tools.
**Fuzzy controller implementation**

After checking the functionality of the Simulink model, incorporation of the *System Generator* block facilitates hardware synthesis by translating the model into different types of netlists.

8) To obtain the FPGA programming file, open *Backward_tb* model, check that ‘*Backward.m*’ has been executed and that appropriate values for *np* and *pipe* variables have been defined, and launch the simulation to verify that it is carried out without errors.

- Double click on the *System Generator* block. In the window that appears, choose ‘*Bitstream*’ as *Compilation* option, select as *Part* the type of FPGA on which the controller will be implemented (see Table 1; e.g., *spartan6* →  *xc6slx9-2csg324* for LX9 MicroBoard) and define in *Target directory* the location of output files (e.g., *./netlist_LX9*) and the FPGA clock period (e.g., 20).

- Press the *Apply* button to save the settings and then the *Generate* button to start the process.

**Note_1**: The first time the model is synthesized the process takes about 15 or 20 minutes. Much of this time is used by the ISE core generation tool (Core Generator) for obtaining the HDL descriptions of the different system components. These cores are stored in a cache and will not be generated again in subsequent executions of System Generator.

**Nota_2**: System Generator synthesizes only the part of the Simulink model that is between input and output Gateways. For this reason, the result of the synthesis process would have been exactly the same if it had been launched in the *Closed_loop* model rather than in *Backward_tb* (as both models use the same *Backward* block).

**Nota_3**: Specific tasks involved in the execution of System Generator depend on the compilation option chosen. In the case of selecting the ‘*Bitstream*’ option, the following tasks are basically performed:

- Generation of cores (*Core Generator*; ‘*.netlist\_sysgen\coregen\xxxx’)
- Logic synthesis of the model (*Xst*; ‘*.netlist\_\synth\_model’)
- Implementation (*Xflow: ngdbuild, map, par; *.netlist\_\xflow’)
- Generation of ‘.bit’ file (*Xflow: bitgen*).
(Bitstream generation for Zynq-7000 family devices generates an error indicating that the standards used in input/output pins have not been defined)

The final outcome of the implementation process is the generation of the FPGA programming file, `Backward_tb cw.bit`, under `netlist_<Board_ID>` directory.

- Once completed the implementation process, use the **Show Reports** button in the **Compilation status** window to analyze FPGA resources consumed by the system.

9) It is also possible to re-implement the model in the ISE environment in an interactive way. To do this, start the ISE project navigator and use the command (**File → Open Project ...**) to open the project generated by **System Generator**: `...\netlist_<Board_ID>\backward_tb cw.ise`.

- The Xilinx project navigator has different windows. The two on the left show the source files that make up the project (**Sources**) and the tasks or processes that can be applied to each file (**Processes**). The windows on the right provide utilities to display the information generated at various stages. Finally, the window located in the lower part acts as a console where errors or warnings that occur when performing each task appear. Initially, the project appears in **New** state and has not any generated report.

- To synthesize the design, select ‘backward_tb cw’ in **Sources** window and press twice the left mouse button on the **Synthesize-XST** task in **Processes** window, or click on the task with the mouse right button and choose **Run**. Note that, once completed the synthesis, estimated resource consumption of the FPGA as
well as a link to the full report generated during the process (Synthesis Report) appear in the design summary.

- By expanding the Synthesize-XST task, it is possible to access to a series of subtasks or collateral tasks that can be of great interest. For example, running the View RTL Schematic task shows a hierarchical block diagram of the synthesized system. Browse the different levels of this hierarchy for some of the blocks included in the XfuzzyLib library in order to see what FPGA components are used in their implementation.

- Design implementation can be done in one step, using the Implement Design level of the Processes window, or by running independently the different steps involved in this level (translation, technology mapping and place&route). Keeping again selected ‘backward_tb-cw’ in the Sources window, run the three stages of implementation and observe the results obtained in each of them.

- In particular, selecting the Module Level Utilización option in the FPGA Design Summary window, it is possible to obtain information about the resources consumed by the different system components.

- Finally, following a procedure similar to the previous one, execute the programming file generation step.
Hardware Co-Simulación

Using the facilities provided by *System Generator*, it is possible to verify the closed-loop behavior of the controller implementation through hardware/software co-simulation of the control system in combination with a Simulink model of the plant. In this case, the "implementable" part of the model (which is that between the input and output gateways) will be synthesized, implemented and programmed on the FPGA, while the rest of the model will continue running by means of *MATLAB* code. In order to provide interaction between the two parts of the model, *System Generator* includes, in a way transparent to the designer, a number of design elements that enable communication and cooperation between software and hardware components.

10) To carry out hardware/software co-simulation of the Romeo-4R control system:

- Select ‘...\Romeo_BW_H\SysGen’ as working directory for *MATLAB*. Open the file ‘Closed_loop.mdl’ (‘Closed_loop.slx’ for 2012b version), define appropriate values for the variables of Romeo 4R, and launch the simulation to verify that it is performed without errors.

- Open the *System Generator* block. Select as Compilation option *Hardware Co-Simulation → <Board-identifier>* (e.g., LX9 MicroBoard) and change the Target directory to ‘./hwcosim_<Board_ID>’. Note that data related to the FPGA cannot be modified in this case, as they are characteristic features of the selected board.
• Press the **Apply** button to save the settings and then the **Generate** button to start the process. Once completed, the library ‘Closed_loop_hwcosim.lib’, containing the model of a *JTAG Co-sim* block called *Closed_loop_hwcosim*, will be generated in ‘\Hwcosim_<Board-ID>‘ directory.

11) Insert the generated block into the *Closed_loop* model by replacing *Backward* block. Flip the block (using the **Format → Flip Block** command) and resize it to connect it to the other components of the model, taking into account that outputs Out and Out1 appear interchanged with respect to those of *Backward* block. Once completed, save the model with name ‘Closed_loop_ <Board_ID>‘.

• Press twice on *Closed_loop_hwcosim* to open its mask and observe the used parameters and initialization code. Switch to the **cable** tab of the new window and select *Digilent JTAG USB Cable* (LX9 MicroBoard board is not able to auto-detect the type of cable).

• When launching the simulation, if the board is not connected and turned on, it is detected when trying to initialize the JTAG and an error message appears. Otherwise, the co-simulation process will be performed.
12) Set the Romeo 4R model for the same paths that were obtained in step 3 of the first part of the tutorial and compare them with those obtained by simulation with Simulink in step 7.
Notes for ZedBoard and Zybo

The Zedboard development board is able to identify the type of cable and to function normally when a locally connected Xilinx Platform USB cable is used.

When a Digilent USB JTAG cable (JTAG HS1) or the USB cable supplied with the board is used, it is necessary to program the FPGA (with iMPACT) and skip the device configuration step in SysGen.

iMPACT console (JTAG HS1 cable):

GUI --- Auto connect to cable...
INFO:iMPACT - Digilent Plugin: Plugin Version: 2.4.4
INFO:iMPACT - Digilent Plugin: found 1 device(s).
INFO:iMPACT - Digilent Plugin: opening device: "JtagHs1", SN:210205805266
INFO:iMPACT - Digilent Plugin: User Name: JtagHs1
INFO:iMPACT - Digilent Plugin: Product Name: Digilent JTAG-HS1
INFO:iMPACT - Digilent Plugin: Serial Number: 210205805266
INFO:iMPACT - Digilent Plugin: Product ID: 30700150
INFO:iMPACT - Digilent Plugin: Firmware Version: 0108
INFO:iMPACT - Digilent Plugin: JTAG Port Number: 0
INFO:iMPACT - Digilent Plugin: JTAG Clock Frequency: 10000000 Hz
iMPACT console (Cable supplied with the board):

GUI --- Auto connect to cable...
INFO:iMPACT - Digilent Plugin: Plugin Version: 2.4.4
INFO:iMPACT - Digilent Plugin: found 1 device(s).
INFO:iMPACT - Digilent Plugin: User Name: Zed
INFO:iMPACT - Digilent Plugin: Product Name: Digilent Zed
INFO:iMPACT - Digilent Plugin: Serial Number: 210248447660
INFO:iMPACT - Digilent Plugin: Product ID: 00E00153
INFO:iMPACT - Digilent Plugin: Firmware Version: 0108
INFO:iMPACT - Digilent Plugin: JTAG Port Number: 0
INFO:iMPACT - Digilent Plugin: JTAG Clock Frequency: 10000000 Hz

The Zybo board can only be programmed through the standard USB cable, so to carry out the hardware co-simulation it will be necessary to use the same procedure (program the device with iMPACT and mark the Skip device configuration option in SysGen).

**Simulation on a remote board**

From Release 12.2, ISE tools support environment co-simulation with a board connected to a remote system via the CSE server. For this, it is necessary to run the "cse_server" command on the remote system.
In the case of the ZedBoard development board, for remote co-simulation it is necessary to use a Xilinx Platform USB cable and follow the above-described procedure (to program the device with iMPACT and to skip the configuration step in SysGen).

You can learn more about this problem and its solution at the following URL:


There is a known issue with the ZC702 board and to work around this, Please follow the below steps.

1. Generate the hardware co-simulation block and make connections

2. Use impact to configure FPGA. Use the bit file from the location generated from the sysgen.

3. After successful of FPGA configuration. Check skip device configuration option in the Hw cosim block advanced tab. Run the co-simulation
C) Conversion into IP-module

System Generator provides two mechanisms for integrating user logic as peripherals of processors developed with Xilinx Platform Studio. Both mechanisms utilize the EDK Processor element available in the Xilinx Blockset library and make use of shared memory blocks to identify peripheral input and output ports that will be mapped into the memory space of the processor. The first of these alternatives is to add into a model developed with SysGen one EDK processor block to export the design as a peripheral of MicroBlaze or ARM processors.

Exporting an IP-module from SysGen

1) Run Xilinx Platform Studio (XPS) and create a new project using the Base System Builder wizard with the following characteristics:
   - Project File: \Romeo_BW_H\EDK\LX9\system.xmp
   - Interconnect Type: AXI System

Board and System Selection section:
   - Board Vendor: Avnet
   - Board Name: Avnet Spartan-6 LX9 MicroBoard

Peripheral Configuration section:
   - Include cores: MCB3_LPDDR y USB_Uart (115200 baudios, 8 bits, No parity)
Files 'Backward_IP_2011b.mdl' and 'Backward_IP_2012b.slx', located in directory ‘...\Romeo_BW_H\EDK’ contain Simulink models (compatible with MATLAB 2011b and 2012b, respectively) required to export the fuzzy controller as an IP-module.

2) Open the file corresponding to the MATLAB version. Analyze the configuration of the different elements that have been introduced to define the interface ports between the IP-module and the AXI bus.

- To complete the design:
  - Include Backward block by copying it from Backward_tb or closed_loop models developed in the second part of the tutorial (‘SysGen’ directory). Also copy the parameter initialization file 'Backward.m' from the same directory.
  - Include one EDK Processor block from Xilinx Blockset library.
  - Save the model as 'Backward_IP.mdl' ('Backward_IP.slx' for MATLAB 2012b).

- Klick twice on EDK Processor block to open its graphical interface.

**Basic:**
- Include all the available memory using the Add button.
- Set the processor to EDK pcore generation.

**Implementation:**
- Select the type of bus: AXI4.
- Mark the Dual Clocks box.
3) Use the System Generator block to start the export process. Choose Export as a pcore to EDK as compilation option. Set the export options (Settings button) including as destination location the directory where XPS project was created in step 1. Select the target device (e.g., spartan6 xc6slx9-2csg324 for LX9 MicroBoard) and define the destination directory (e.g., './netlist_LX9').

- Press Generate button to complete the process.
Once the process of generating the IP-module is complete, the directory structure under ‘...\EDK\LX9\pcores\backward_ip_axiw_v1_00_a’ contains the files needed to connect and access the new peripheral from EDK.

![Compilation status](image)

After running the **Project → Rescan User Repositories** command, the IP named "backward_ip_axiw" appears, under the **Project Local PCores → USERS** group, in the **IP Catalog** tab of XPS project created in step 1.

4) To incorporate the new module into the system, click on it twice or select it and drag it to the **Bus Interfaces** tab of **System Assembly View**. After clicking OK to accept the default values that appear in the module graphical interface and select "microblaze_0" as the processor to which it will be connected, the core is included in the system and connected to the processor through the AXI bus.

- Use the **Ports** tab of **System Assembly View** to connect `sysgen_clk` port to CLKOUT2 signal of the clock generator.
- To prevent the process is interrupted if the temporary restrictions are not met, before starting the generation of the bitstream for programming the FPGA, it is necessary to unmark *Treat timing closure failure as an error* in the project options window (*Project → Project Options*).

- Then, use the *Hardware → Generate Bitstream* command or the corresponding icon to complete the implementation of the system and to generate the bitstream for programming the FPGA. Observe the messages displayed on the XPS console.

5) Use the *Project → Export Hardware Design to SDK* command or the corresponding icon to export the design to SDK. In the *Export to SDK / Launch SDK* window mark the *Include bitstream and BMM file* option, accept the location of hardware description files, and click the *Export & Launch SDK* button.

- Define ‘...\Romeo_BW_H\EDK\LX9\SDK\workspace’ as SDK workspace location in the *Workspace Launcher* window.

When opening SDK, the hardware platform (*LX9_hw_platform*) is imported and information regarding the design is shown using the C/C++ perspective of the Eclipse environment.

6) In SDK, use the *File → New → Application Project* command to create a new application with the following characteristics:

- *Project name*: Test_Backward
- *Hardware Platform*: LX9_hw_platform
- *OS Platform*: standalone
- *Template*: Empty Application
Pressing the Finish button, the new Board Support Package, called ‘Test_Backward_bsp’, is generated and compiled. Check in the system.mss tab that backward_ip_axiw IP-module is associated with a generic driver. To associate it the driver that was created during the process of generating the IP-module, it is necessary to define the repository location.

7) Run the Xilinx Tools → Repositories command and use the New button to add ‘...\Romeo_BW_H\EDK\LX9’ as a new local repository.

- Then, use the Modify this BSP’s Setting button in system.mss tab or the Xilinx Tools → Board Support Package Settings command to assign the new driver (Overview → drivers → backward_ip_axiw).

If the Project → Build Automatically option is set, the different components of the application are recompiled when pressing OK.

8) To import the source code of the application, click the mouse right button on Test_Backward → src and run the Import command.

- Select General → Select File System in the Select window and use the File System window browser to indicate the position of the tutorial ‘EDK’ directory.

- Mark the ‘Test Backward.c’ file that appears on the right side of the window and click the Finish button.

Upon completion of this action, the specified file is imported and compiled to generate the ‘Test_Backward.elf’ executable (Test_Backward → Binaries).

- Use SDK or a text editor to analyze the application source code.

The file ‘xparameters.h’ contains the definition of the base memory address ( XPAR_BACKWARD_IP_AXIW_0_BASEADDR) and other parameters of the IP-module, while the functions that facilitate its use are defined in the header file ‘backward_ip_axiw.h’.

During the IP-module export process, an HTML file, ‘...\EDK\LX9\pcores\backward_ip_axiw_v1_00_a\doc\html\api\index.html’, was created containing information on the API (Application Programming Interface) provided to facilitate the development of software applications that use the IP-module.
Now, it only remains to verify the correct operation of the application on the development board. SDK is not able to automatically detect the LX9 board (use the Digilent USB Cable option of the Xilinx Tools → Configure JTAG Settings command).

9) Program the FPGA using the Xilinx Tools → Program FPGA command while maintaining hardware and software configuration options that appear by default.

10) The application used to verify the operation of the fuzzy controller writes the results to the standard output (STDIO) of the processing system. To display this data, it is possible to connect STDIO to the PC through the USB to UART Bridge port of the development board and use the SDK console or a serial communications program such as Windows HyperTerminal.
Although the UART was set to 115200 baud when the XPS project was created in step 1, when the LX9 board is used, the HyperTerminal or SDK console must be configured to **38400** baud to work properly.

- To run the application, use the mouse right button to select the file ‘Test_Backward.elf’ under ‘Test_Backward → Binaries’ and execute the **Run as → Launch on hardware (GDB)** command.

- To connect the standard output to the SDK console, edit the configuration (**Run as → Run Configurations ...**) and choose the appropriate port and speed in the **STDIO Connection** tab. Launch the application by clicking the **Run** button.
• Using the HyperTerminal, it is possible to capture in a file the standard output of the application (using the Transfer → Capture Text command) in order to plot the input/output surface of the fuzzy controller.

![HyperTerminal capture text](image)

• Launch the simulation by selecting the file 'Test_Backward.elf' with the mouse right button and executing the Run as → Launch on hardware (GDB) command or by using the Run As ... icon in the SDK taskbar.

![HyperTerminal simulation](image)

11) Use gnuplot to graph the input/output controller surface from data stored in 'sal.txt' file.

Fig 1. (gnuplot> splot 'sal.txt' using 1:2:4 with lines)

Fig 2. (gnuplot> splot 'sal.txt' using 1:2:5 with lines)
Notes for ZedBoard

Step 1: XPS Project
- **Project File:** ...\Romeo_BW_H\EDK\ZB\system.xmp
- **Interconnect Type:** AXI System
- **Board Vendor:** Avnet
- **Board Name:** ZedBoard Zynq Evaluation and Development Kit
- **Peripheral Configuration:**
  - Keep default values

Step 3: Configuring the System Generator block
- **Compilation:** Export as a pcore to EDK
- **Settings – EDK project:** ...\Romeo_BW_H\EDK\ZB\system.xmp
- **Part:** Zynq xc7z020-1clg484
- **Target directory:** ./netlist_ZB

Step 4: Connecting the IP-module (System Assembly View – ports)
- Processor to connect the IP: processing_system7_0
- Connect sysgen_clk to processing_system7_0::FCLK_CLK0

Step 5: Application Development
- **SDK Workspace:** ‘...\Romeo_BW_H\EDK\ZB\SDK\workspace’

Step 9: Programming the FPGA (Xilinx Tools → Configure JTAG Settings)
- **Cable type:** Auto Detect

Step 10: Configuring the Standard Output
- The SDK console or HyperTerminal should use the configuration appropriate to the ARM processor connected to UART (115200 baud)).

---

6 The XFT35-03 tutorial (Hardware Implementation of fuzzy controllers with xfvhdl and System Generator) contains the detailed description of a design flow similar to the previous one using the ZedBoard development board.
Inclusion of MicroBlaze into a Simulink model

As an alternative to the mechanism described in the previous section to connect user logic as peripheral of processors implemented on Xilinx’s FPGAs, System Generator also allows importing a MicroBlaze project into a Simulink model through an EDK processor block, to complete the development in MATLAB/Simulink environment.7

1) Run Xilinx Platform Studio (XPS) and create a new project using the Base System Builder wizard with the following characteristics:

- Project File: ...\Tutorial_xfsg\Romeo_BW_H\HDL\Atlys\system.xmp
- Interconnect Type: PLB System
- Peripheral Repository Search Path:
  C:\\Xilinx\Atlys_BSB_Support_v_3_7\Atlys_PLB_BSB_Support\lib
- Board Vendor: Digilent
- Board Name: Atlys
- Local Memory: 32 KB
- Cores: MCB_DDR2, RS232_Uart_1 (115200 bauds, 8 bits, No parity), dlmb_cntrl, dlmb_cntrl

This option only works properly for PLB bus based systems
Files 'Backward_IP_2011b.mdl' and 'Backward_IP_2012b.slx', located in directory ‘...\Romeo_BW_H\HDL’, contain Simulink models (compatible with MATLAB 2011b and 2012b, respectively) needed to import the processor into the Simulink model.

2) Open the file corresponding to the used MATLAB version. Analyze the configuration of the various elements that have been introduced to define the interface ports between the IP-module and the PLB bus.

- To complete the design:
  - Include the fuzzy controller block into the model by copying it from the Backward model developed in the second part of the tutorial (directory ‘SysGen’) and also copy the ‘Backward.m’ file from the same directory.
  - Also include a Subsystem block from Simulink library (Commonly Used Blocks).

- Edit the subsystem to remove input and output ports and to include the following blocks from the Xilinx Blockset library:
  - a System Generator block (Basic Elements)
  - an EDK Processor block (Control Logic)
- Save the model as ‘Backward IP.mdl’ (whether MATLAB 2011b as MATLAB 2012b is used)\(^8\).

- Klick twice on the **EDK Processor** block to set the following properties in the **Basic** tab:
  - **Memory Map:**
    - Include all the available memory using the *add* button.
  - **Processor Options:**
    - Configure the processor for *HDL netlisting* and select as project to import the one created in step 1.

The import process performs the necessary steps to encapsulate the design as an IP-module for MicroBlaze processor (with the interface located under the ‘...\Romeo_BW_H\HDL\Atlys\cores\sg_plbiface_v1_00_a’ directory), as well as to include and connect it to the XPS project, associating one or more addresses in the memory space of the processor.

---

\(^8\) The script that controls the build process of the hardware co-simulation block expects a file with "*.mdl" extension. (This is the usual extension in MATLAB 2011b, but can be also used in the 2012b version).
Select the **Bus Interfaces, Ports and Addresses** tabs in the **XPS** GUI to check how the IP-module has been included in the project.

The inclusion of a **System Generator** block into the subsystem containing the EDK Processor allows implementing the processing system and the IP-module interface on a FPGA while the module itself is further described by a Simulink model. This hardware co-simulation procedure allows refining the design of the IP-module to facilitate verification of their behavior when interacting with the physical implementation of the processor.
3) For implementing the co-simulation block:

- Open the **System Generator** block located in the subsystem included in the **Backward_IP** model. Select as compile option **Hardware Co-Simulation** → `<Board-identifier>` (e.g., "Atlys – JTAG") and change the ‘target directory’ to `.\netlist_<Board_ID>`.

- Press the **Apply** button to save the settings and then the **Generate** button to start the process. Once completed, the ‘Subsystem_hwcosim_lib’ library, which contains the model of a **JTAG Co-sim** block called **Subsystem hwcosim**, will be generated in the `.\netlist_<Board-ID>` directory.

When this design flow is followed, **System Generator** runs first XPS tools to carry out the compilation of the processing system and then ISE tools for implementation on the FPGA.
Xfuzzy: Fuzzy Logic Design Tool
4) To complete the co-simulation model, include the newly generated block into the Backward_IP model replacing the Subsystem block and save the model with the name ‘Backward_IP_<Board_ID>‘.
- Klick twice on Subsystem hwcosim to open its mask and observe the used parameters and initialization code.

- Then, to start the application development environment, go to the Software tab, uncheck the Enable Co-Debug with Xilinx SDK option, verify the location of the Workspace, and click the Launch Xilinx SDK button.

When opening SDK, the hardware platform (Atlys_hw_platform) is imported and information regarding the design is shown using the C/C++ perspective of the Eclipse environment.

5) In SDK, use the File → New → Application Project command to create a new application with the following characteristics:

- **Project name**: Test_Backward
- **Hardware Platform**: Atlys_hw_platform
- **OS Platform**: standalone
- **Template**: Empty Application

Pressing the Finish button, the new Board Support Package, called 'Test_Backward_bsp', is generated and compiled.

- Verify in the system.mss tab that the sg_plbiface_0 IP-module has already associated the correct driver, created in the process of generating the IP-module.

- Run the Xilinx Tools → Repositories command to verify that the directory ‘...\Romeo_BW_H\HDL\netlist_Atlys\SDK_Export\sysgen_repos’ has been added to the list of local repositories.

6) To import the application source code, click the mouse right button on Test_Backward → src and run the Import command.

- Select General → Select File System in the Select window and use the File System window browser to indicate the position of the tutorial ‘HDL’ directory.

- Mark the ‘Test_Backward.c file that appears on the right side of the window and click the Finish button. By completing this action, if the Project → Build Automatically option is set, pressing OK the specified file is imported and compiled to generate the ‘Test_Backward.elf’ executable (Test_Backward → Binaries).

- Use SDK or a text editor to analyze the application source code. The file ‘xparameters.h’ contains the definition of the base memory address
( XPAR_SG_PLBIFACE_0_BASEADDR ) and other parameters of the IP-module, while the functions that facilitate its use are defined in the header file ‘sg_plbiface.h’.

During the IP-module export process, an HTML file, ‘...\HDL\Atlys\pcore\sg_plbiface_v1_00_a\doc\html\api\index.html’, was also created containing information on the API (Application Programming Interface) provided to facilitate the development of software applications that use the IP-module\(^9\).

7) When the new application is added in SDK, a command file is created under the src directory and used by the linker to assign external memory to the different sections of the executable (deploy the src folder and click twice on ‘lscript.Id’). In this case it is necessary that the executable file resides in the local memory of the FPGA so that the program can be loaded when launching Simulink simulation model.

- Place the mouse over the ‘Test_Backward’ folder and use the menu that appears when you press the right button to run the Generate Linker Script command. In the new window, select ‘ilmb_cntlr_dlmb_cntrl’ as destination for sections of code and data of the executable, as well as for locating Heap and Stack. Press the Generate button and answer yes to the question on overwriting the command file.

---

\(^9\) The name of the function that returns the location in memory of input and output ports of the IP-module is "XC_GetShmem" and not "XC_GetShMem" as indicated in the HTML document.
Once generated the executable file of the application, the verification stage of the IP-module is performed in **MATLAB/Simulink** environment. For do that:

8) Back to model *Backward_IP_Atlys*, click twice on *Subsystem hwcosim* to open its mask and go to the *Software* tab.
   - Indicate the location of the ELF file in the field *Software Initialization*: `.\netlist_Atlys\SDK_Workspace\Test_Backward\Debug\Test_Backward.elf`.

9) Launch Simulink model simulation.
   - As in the case of hardware co-simulation described in Part B of the tutorial, if the board is not connected and turned on when launching the simulation, it will be detected when trying to initialize the JTAG and an error message appears. Otherwise, both the FPGA programming file and the application executable are transferred through JTAG and co-simulation process start.
   - The application used to verify the operation of the fuzzy controller writes the results to the standard output (STDIO) of the processing system. To display this data, it is possible to connect STDIO to the PC through the *USB to UART Bridge* port of the development board and use the SDK console or a serial communications program such as Windows HyperTerminal.

Alternatively, in this case it is also possible to use Simulink utilities to monitor model behavior. To do this, open the *Scope 1* block showing the evolution of *phi* input and *bw* output of the fuzzy controller.
- The example used to illustrate this case does not include any synchronization mechanism between system hardware and software components. For this reason, and in order to take into account the time required to transfer information between the Simulink simulator and the processor implemented in FPGA, the application now includes a delay cycle significantly higher than that associated with the strategy described in the previous section.

- The simulation model stores the output variables in the MATLAB workspace, enabling the use of scripts and/or callback functions for graphical representation of system outputs.
## Document Reviews

<table>
<thead>
<tr>
<th>Fecha</th>
<th>Versión</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>15/06/2016</td>
<td>1.0</td>
<td>Initial release in English</td>
</tr>
<tr>
<td>22/03/2018</td>
<td>1.1</td>
<td>Xfuzzy 3.5; Zybo board in SysGen</td>
</tr>
</tbody>
</table>