A Guide
To The
FPGA Architect - Xilinx
XC4000/Spartan
# Table of Contents

1.0 Introduction .................................................................................................................. 5

2.0 Installation ..................................................................................................................... 6
   2.1 Installation .................................................................................................................. 6
   2.2 Xilinx Software Requirements ................................................................. 6

3.0 Overview ...................................................................................................................... 7
   3.1 Operational Concept ............................................................................................... 7
   3.2 Xilinx Token Libraries ......................................................................................... 8
   3.3 Xilinx Core Parameter Checker ......................................................................... 10
   3.4 Xilinx Core Parameter Generation ................................................................. 11

4.0 SystemView FIR Design Example ............................................................................. 13

APPENDIX A: Numeric Value versus Bit Position Integer Type Conversion using the DSP Adder ................................................................. 18
1.0 Introduction

This document describes the key features and the use of the FPGA Architect - Xilinx XC4000/Spartan. This software allows DSP system designers to implement their SystemView system designs in Xilinx FPGAs.

SystemView’s optional DSP library has been enhanced to include functional equivalents of Xilinx CORE Generator cores. After DSP algorithms are fully tested and analyzed using SystemView, tokens in the DSP system can be converted to Xilinx core equivalents using Xilinx CORE Generator. This software supports conversion of single tokens, groups of tokens, or entire systems.

For example, engineers can use SystemView’s Linear Systems Filter design package to design FIR filters that can be implemented using Xilinx SDA filters. With the click of a button, parameters such as coefficient values and the computational description of the FIR filter are checked and then passed to Xilinx CORE Generator.
2.0 Installation

2.1 Installation
The FPGA Architect is installed automatically with SystemView. To use this software, the SystemView user must license this option and the SystemView DSP library.

2.2 Xilinx Software Requirements
The FPGA Architect requires Xilinx CoreGEN 1.5i and the Xilinx DSP LogiCOREs. In addition, if you wish to select the “VHDL Instantiation Code” option, you will need the Foundation HDL tools from Xilinx. If you wish to select the “Schematic Symbol” option, you will need the Foundation tools. To obtain any of these tools, please contact your local Xilinx sales office.
3.0 Overview

3.1 Operational Concept
The FPGA Architect is optional software for use with the SystemView Professional Edition. This option enables designers to export DSP function design information from a SystemView simulation into Xilinx tools for programming FPGAs.

An overview of the interaction of SystemView with the Xilinx tools is illustrated in Figure 1. To implement functional blocks in Xilinx FPGAs, the designer must select tokens from SystemView that possess the “FPGA aware” attribute, as shown in figure 2. With SystemView, the designer creates, debugs, and optimizes a complete design that includes these functional blocks. After the design is completed, the designer may transfer the information for these functional blocks to Xilinx CORE Generator. Xilinx CORE Generator takes the design information and produces netlists of application-specific DSP cores. Additional outputs can be produced from Xilinx CORE Generator, including VHDL simulation models, VHDL instantiation code, or schematic symbols. If desired, SystemView can invoke Xilinx Foundation tools such as the HDL Editor or the Schematic editor and direct output of the appropriate Xilinx CORE Generator to them.
3.2 Xilinx Token Libraries

A number of SystemView tokens correspond to FPGA cores in the Xilinx CORE Generator library. Table 1 provides a partial list of these tokens and corresponding Xilinx cores. Please contact Elanix technical support for the latest information.
<table>
<thead>
<tr>
<th>Token</th>
<th>Xilinx LogiCORE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adder</td>
<td>Registered adder core</td>
<td>Generates the sum of all inputs.</td>
</tr>
<tr>
<td>BitSym</td>
<td>Serial to Parallel Converter</td>
<td>Converts serial bits to symbols (words)).</td>
</tr>
<tr>
<td>Bitwise Complement</td>
<td>1’s complementer core</td>
<td>Inverts each input bit value.</td>
</tr>
<tr>
<td>Comb Filter</td>
<td>Comb Filter</td>
<td>Delays the input and subtracts the delayed value from the current input.</td>
</tr>
<tr>
<td>Filter</td>
<td>Parallel Distributed Arithmetic FIR filter</td>
<td>Performs digital filtering of input.</td>
</tr>
<tr>
<td>Gain</td>
<td>Constant Coefficient Multiplier</td>
<td>Multiplies the input by a constant.</td>
</tr>
<tr>
<td>Integrator</td>
<td>Integrator</td>
<td>Accumulates the inputs. A nonzero reset value resets the accumulator to 0.</td>
</tr>
<tr>
<td>LUT</td>
<td>Registered ROM</td>
<td>Performs table look up. If the clear control is high, a zero is output. The table is initialized from a text file.</td>
</tr>
<tr>
<td>Multiplier</td>
<td>(a) Parallel multiplier, area optimized, or (b) Parallel multiplier, speed optimized</td>
<td>Generates the product of two inputs.</td>
</tr>
<tr>
<td>Negate</td>
<td>2's complementer core</td>
<td>Subtracts input from zero.</td>
</tr>
<tr>
<td>SinCos</td>
<td>Sine/Cosine</td>
<td>Performs table look up for sine and cosine.</td>
</tr>
<tr>
<td>Sqrt</td>
<td>Square Root</td>
<td>Calculates the square root of the input. If the input is negative, the square root of the absolute value is negated for output.</td>
</tr>
<tr>
<td>Subtractor</td>
<td>Registered subtractor core</td>
<td>Subtracts input 1 from input 0.</td>
</tr>
<tr>
<td>SymBit</td>
<td>Parallel to Serial Converter</td>
<td>Converts symbols (words) to serial bits.</td>
</tr>
</tbody>
</table>

Table 1: List of Xilinx DSP LogiCORE functions supported in SystemView
The majority of these tokens are found in the SystemView DSP library. One exception is the FIR filter token, which resides in the SystemView Operator library.

For SystemView libraries having tokens that map to Xilinx cores, the dialog box has an “FPGA Aware” indicator. When the pointer is placed over a supported token the indicator is green, otherwise it is red. Also, the token status window, obtained by placing the mouse pointer over the token, will show “FPGA Aware = True” or “FPGA Aware = False”.

3.3 Xilinx Core Parameter Checker

Because Xilinx DSP LogiCOREs have implementation constraints, a token’s characterization must be checked to see if it’s appropriate for the corresponding core, and ensure successful implementation. Such characterization includes some or all of the parameters set in the Edit Parameters dialog box, such as output type, and may include the characterization of the token input(s).

To test the validity of a token’s characterization against the core requirements, the designer may use the Check Specific Token function, from the SystemView menu Tools->Xilinx FPGA… dialog. In addition, it
is also possible to check every connected FPGA aware token in the system using the Check Entire System button from the same menu.

Whenever you attempt to use the FPGA Architect, SystemView automatically validates token characterization properties. However, once you begin optimizing your token specifically for Xilinx DSP LogiCOREs, you may wish to immediately recheck its characterization after entering new parameter value(s) into the token parameter dialogue box. This ensures that you only simulate designs that can be successfully implemented.

### 3.4 Xilinx Core Parameter Generation

Once SystemView is satisfied with the results of the parameter check, the FPGA aware tokens can be converted to Xilinx core files.

![SystemView FPGA](Image)

The Convert Selected Token button is used to generate an FPGA core from an individual token. First, the token is checked for its core compatibility, and then, if the core can be generated, a dialog appears asking what output products (files) should be generated and where to place them. Afterward, the token’s characterization is written into a script.
file for Xilinx CORE Generator, and Xilinx CORE Generator is invoked in the batch mode.

When SystemView completes the translation, the generated outputs can be found in the project folder, and the Foundation Schematic Symbol is placed in the current Foundation project.

The names of the output files are created with the name of the SystemView simulation appended by the token number in the system. The file extension determines the product type:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Xilinx Output Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>vei</td>
<td>Verilog Instantiation Template</td>
</tr>
<tr>
<td>vhd</td>
<td>VHDL Behavioral Simulation Model</td>
</tr>
<tr>
<td>vhi, vhx</td>
<td>VHDL Instantiation Template</td>
</tr>
<tr>
<td>xnf, xsf</td>
<td>XNF Implementation Netlist</td>
</tr>
</tbody>
</table>

FPGA Architect - Xilinx XC4000/Spartan
4.0 SystemView FIR Design Example

Step 1: Start SystemView and set the System Time Specification Window to the default values.

Step 2: In the Operator Library, choose the Linear Sys Filters icon and then click parameters.

Step 3: Once in the SystemView Linear System window (Custom Digital System), press the FIR button within the Design area in the upper right corner, and then press the Lowpass button on the pop-up. Alternatively, the same filter can be specified using the menu sequence Filters->FIR->Lowpass…

Step 4: Enter the parameters indicated in the following text boxes. Clicking on Update Est generates the number of taps.
Step 5: Click Finish, and the SystemView LowPass DSP Filter Design window appears as seen below. By default, coefficients are shown in IEEE double precision, and the impulse response of the filter is graphed. To view the filter gain response, click the Gain button.

![SystemView DSP Filter Design Window](image)

Step 6: Choose the DSP Mode on the menu bar, and then choose Enable DSP Arithmetic Mode. By default, SystemView automatically scales the coefficients. Unchecking the Enable Quantization Auto Scale option can change this. For this example, leave the Auto Scale option on. Choose signed fixed point for the Coefficient Arithmetic Mode and accept the default values. In the Convert Int Types Using section select numeric value. Refer to appendix A for a detailed explanation of numeric value versus bit position fixed-point type to fixed-point type conversion. You should see the following window. Click OK.
Step 7: Once the coefficients have been set, SystemView will prompt you for the Output Arithmetic Mode. Set this mode to Signed Fixed Point Register Size to 31 and the Fraction Size to 24. In the Convert Int Types Using section select numeric value. Refer to appendix A for a detailed explanation of numeric value versus bit position fixed-point type to fixed-point type conversion. The following window will appear. Note that the Gain response has changed. Clicking OK at this point will now complete the filter design. The token may now be incorporated into your system design.
Step 8: Take a sine source from the source library and a converter token from the DSP library. Set the sine source frequency to 5 Hz and the amplitude to 0.9 Volts. Set the parameters of the converter to Signed Fixed Point, Register Size to 10 and Fraction Size to 9. In the Convert Int Types Using section select numeric value. Refer to appendix A for a detailed explanation of numeric value versus bit position fixed-point type to fixed-point type conversion. Connect the source token sine output to the converter token input, the converter token data output to the filter token input, and the filter token data output to an analysis sink. You should have the following picture. Note that the FIR filter has a data output as well as an overflow flag output.
SystemView requires that at least one source and one sink are included in every system before any token can be converted to its Xilinx core equivalent.

Step 9: From the menu bar choose Tools->Xilinx FPGA. The following window will appear:

From this window the choices are evident. In this example, we choose Convert Selected Token and then click OK. Finally, click on the FIR filter (token 0) to select the token.
Step 10: The following window will then appear. In this example, we choose the VHDL Instantiation Template. SystemView will now produce the necessary files for Xilinx CORE Generator. For further explanations of choices, please contact Xilinx or Eagleware-Elanix.

![SystemView Xilinx FPGA Settings](image)

All parameters defined for the filter token have now been transmitted to Xilinx CORE Generator.

APPENDIX A: Numeric Value versus Bit Position
Integer Type Conversion using the DSP Adder
Note that the parameter dialog for the DSP Adder (and in other SystemView DSP Library tokens) gives the engineer the option of selecting Bit Position or Numeric Value for converting fixed-point types. Collectively, the unsigned integer, signed integer and signed fixed point types are referred to as fixed-point types.

Converting a 3.2 Signed Fixed Point to a 6.3 Signed Fixed-Point using Numeric Value, retains the position of the binary point of the number.
Converting a 3.2 Signed Fixed-Point to a 6.3 Signed Fixed-Point using Bit Position, scales the value of the number.

Step 1: Set the System Time Specification Window to the sampling rate of 1Hz and 1 Sample.

Step 2: Take two (2) step sources out of the source library, set one to 0.25 volts and the other to 1.25 volt.

Step 3: From the DSP library, select a DSP converter and set its parameters to Signed Fixed-Point, Register Size to 3 bits and Fraction Size to 2 bits. Connect the 0.25 Volt source to this converter. Get another converter and set its parameters to Signed Fixed-Point, Register Size to 6 bits and Fraction Size to 3 bits. Connect the 1.25 Volt source to this converter. Leave the Convert Int Types Using parameter, at Bit position.

Step 4: From the DSP library select a DSP adder. Set the adder’s output type parameters to Signed Fixed-Point, Register Size to 6 bits and Fraction Size to 3 bits. In the Convert Int Types Using section of the adder parameter window choose Bit Position. You should have the following picture.
Step 5: Take another DSP adder and set its parameters similar to that of the previous adder, but in this adder set the Convert Int Types Using parameter to Numeric Value.

Step 6: Connect the Data outputs of each converter to the adders. Take two current value sinks and connect the adder outputs to these sinks. Run the simulation. You should now have the following diagram.
Note that in one case the addition of 1.25 and 0.25 is 1.375, and in the other case the result is 1.5. Recall that one adder is set to Bit Position while the other adder is set to Numeric Value. In the case of the Bit Position adder, the value 0.01₂ (0.25 in Signed Fixed-Point) was transcribed to 000.001₂ (0.125 in decimal) to conform to the selected output type of the adder. Hence, 1.25 + 0.125 is 1.375. In the case of the Numeric Value adder, the value 0.01₂ is transcribed to 000.010₂ or in other words the decimal point is anchored. Hence, 1.25 + 0.25 is 1.5.

The benefit of setting the parameter to Numeric Value, such as in this example, is that no explicit bit extractors or shifters are needed. The bit extractor or shifter is implicit in this case.