

Simulink to Silicon

ECE 501- Project in lieu of thesis
VIKAS YELAGONDANAHALLI

Summer 2007

Advisor: Dr. Don Bouldin
*Electrical and Computer Engineering
University of Tennessee, Knoxville*

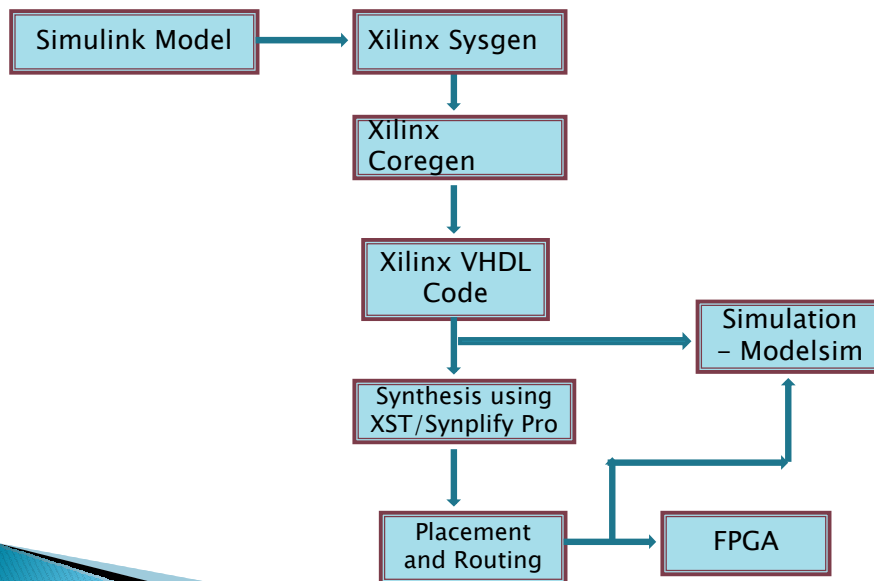
Date: 07/19/07

Goal of the Project:

- ▶ Implementation of a Simulink Design on an ASIC using the IBM7RF 180nm process.
- ▶ Using the Xilinx System Generator (Sysgen) tool to generate the HDL Code.

To implement the Simulink design on an ASIC, there are several steps that need to be followed. The Xilinx Sysgen tool is used to generate a VHDL code from the Simulink Design. This code, which uses Xilinx Core Libraries has to be modified to a generic VHDL code which can be used on any platform. The Generic VHDL code is then synthesized using Synopsys Design Compiler and then implemented on an ASIC using the IBM 405-S PPC core as the CPU.

FPGA Design Flow

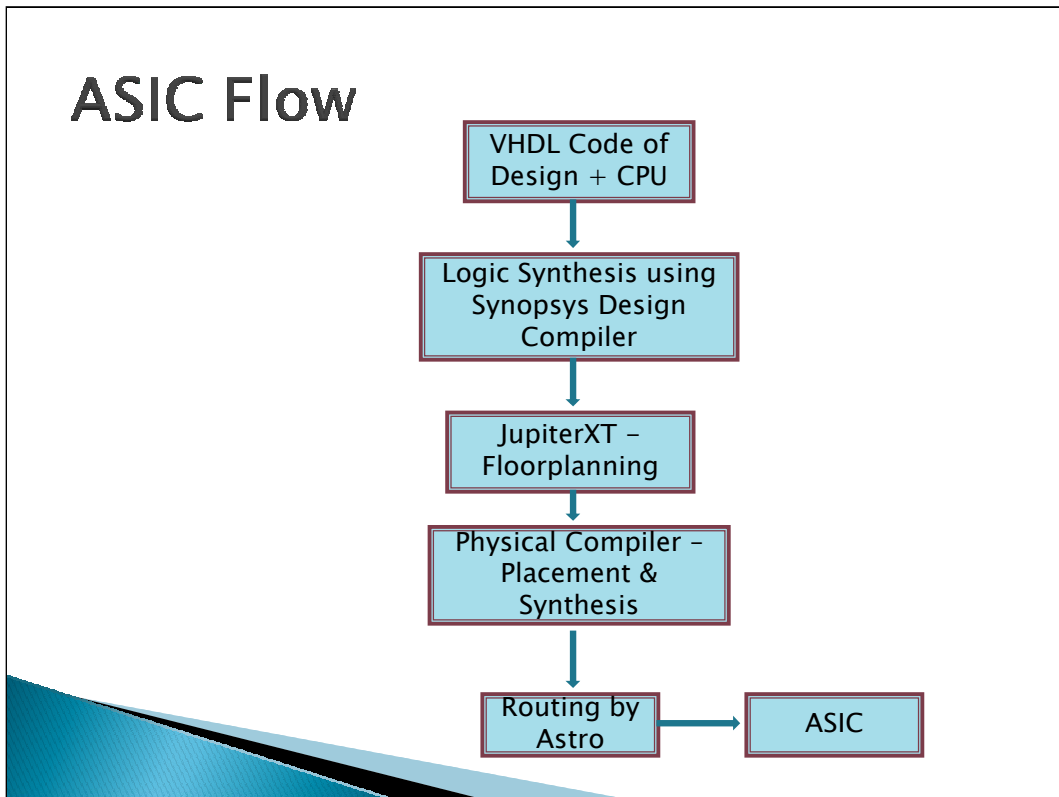


The Simulink Model can be targeted onto an FPGA using the provided flowchart. Once the Simulink model is developed in Matlab, the System Generator tool generates the VHDL code equivalent of the design and targets a particular family of Xilinx FPGAs. This VHDL code is then simulated after compiling the Xilinx Libraries. If Simulator results are satisfactory, the design is synthesized using Xilinx XST or Synplify Pro, and PAR is performed on the synthesized design before implementing the design on the FPGA .

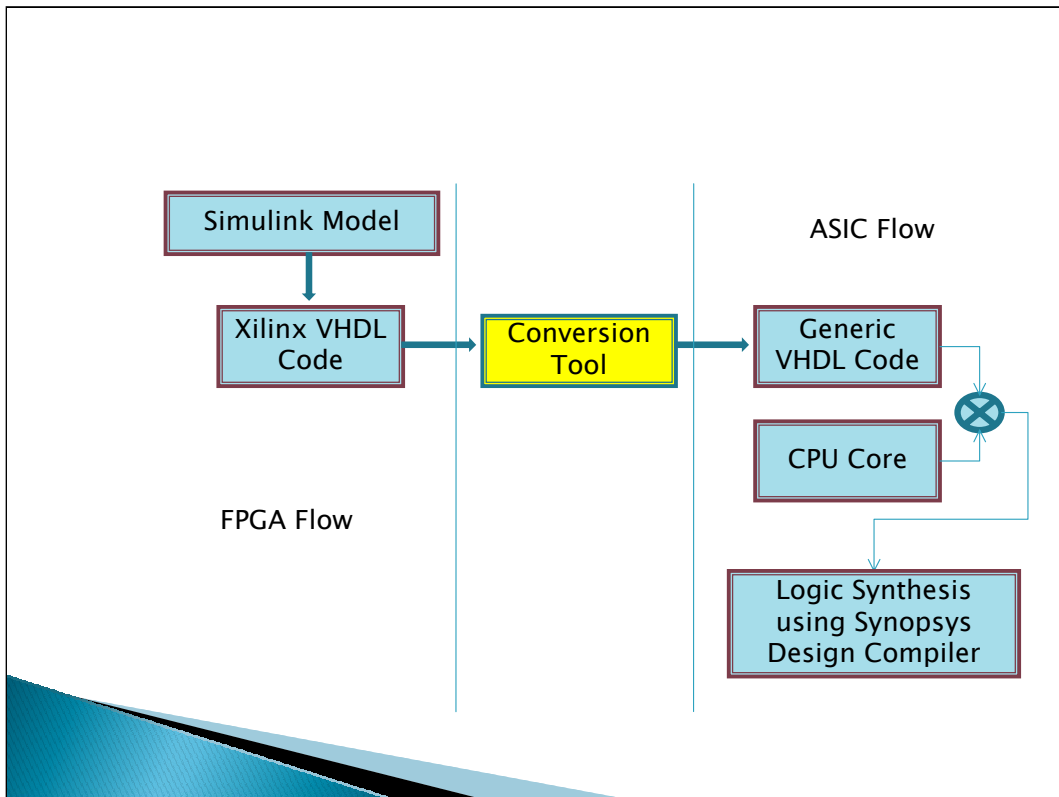
Modelsim by default does not have the Xilinx core libraries installed. These libraries, namely Unisim, Simprim and XilinxCoreLib have to be compiled and mapped so that their paths are added in the modelsim.ini file. The following link provides some information about installing these libraries:

http://www.xilinx.com/xlnx/xil_ans_display.jsp?getPagePath=15338

ASIC Flow



In the design flow of an ASIC, we take the VHDL Code along with a CPU core, like Leon or IBM 405-S PPC and perform synthesis using Synopsys Design Compiler. The JupiterXT tool by Synopsys is used for performing Placement, followed by Routing by Astro. Finally, the layout is implemented onto an ASIC.



When performing Synthesis with Synopsys DC, some of the Xilinx specific components are not synthesized as they are not recognized by Synopsys. To fix this problem, the Xilinx specific code has to be converted to a generic VHDL code which can be used by any tool. Currently, I am working on this tool to automate the process of converting Xilinx specific VHDL to Generic VHDL.

Xilinx System Generator

- ▶ System Generator tool is used for designing high performance DSP systems for Xilinx FPGAs.
- ▶ Is provided as a plug-in to Simulink in Matlab.
- ▶ Generates synthesizable VHDL code which are mapped to Xilinx pre-optimized algorithms.

The Xilinx System Generator tool enables designers to develop DSP systems for Xilinx FPGAs. Designers can design and simulate a system using MATLAB, Simulink, and Xilinx library of bit/cycle-true models. The generated HDL design can be synthesized for implementation in Virtex Platform FPGAs and Spartan family FPGAs. As a result, designers can define an abstract representation of a system-level design and easily transform this single source code into a gate-level representation. Additionally, it provides automatic generation of a HDL testbench, which enables design verification upon implementation.

Other approaches considered

- ▶ SynplifyDSP: A synthesis tool by Synplicity that performs high level DSP optimizations from a Simulink model.
 - Very expensive.
- ▶ AccelDSP: Matlab language based tool for designing DSP blocks for FPGAs and ASICs.
 - Does not use Simulink models but uses Matlab 'm' files.
- ▶ Insecta Project by University of Berkeley.
 - <http://bwrc.eecs.berkeley.edu/Research/Insecta/default.htm>
 - License Restrictions

In SynplifyDSP, the implementations are synthesizable and not vendor dependent like Sysgen. But since the cost of SynplifyDSP is very high (around \$18000 compared to \$2000 for Sysgen), the idea had to be dropped.

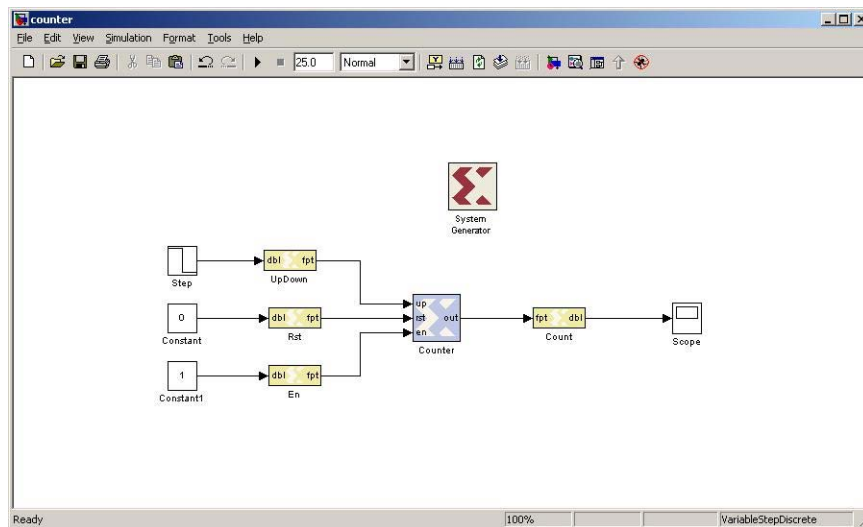
For AccelDSP, though ASICs could be implemented from different vendors, it could not be pursued since the input file was a Matlab 'm' file, and there are no tools to convert Simulink models to Matlab 'm' files.

The Insecta project by Berkeley also uses Sysgen to convert Simulink models to synthesizable HDL and has developed a tool to convert Xilinx specific VHDL to generic VHDL. We are trying to implement the same thing but could not use the Berkeley model due to license restrictions.

System Generator Design

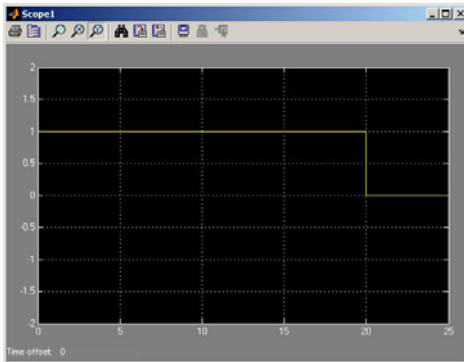
- ▶ 2 Designs were created in Simulink using System Generator and implemented on an FPGA.
- ▶ The 1st Design is an Up/Down Counter and the 2nd design is an FIR Filter.

Up/Down Counter

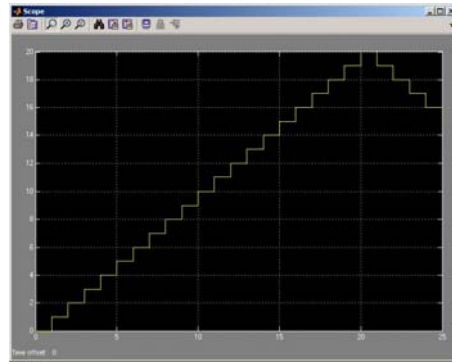


The Counter Design consists of an Up/Down select, an reset pin which has been set to 0 and an enable pin which is set to 1. For the Up/Down Select, I am using a step signal which is high for 20ns and then goes low. So the counter counts from 0 to 20 before it starts to count down. The Gateway In and Gateway Out blocks are used to convert double values to fixed point values and back to double. All the blocks which needs to be synthesized have to be selected from the Xilinx blockset and these blocks will have the Xilinx symbol in them, as seen from the colored blocks in the design.

Simulation results for Counter



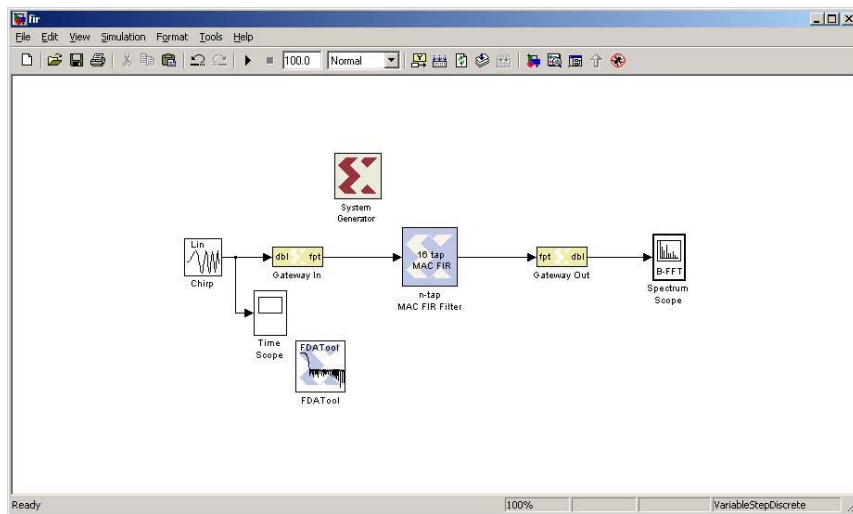
Counter Input



Counter Output

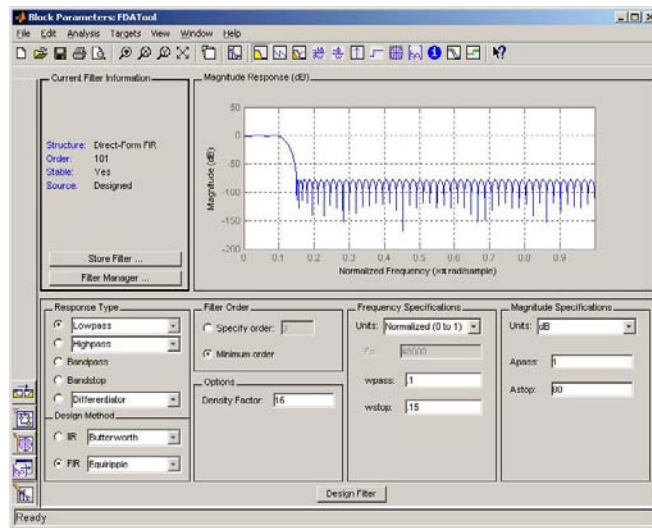
From the graph, we can see that the input step signal is high for 20ns and then goes low for 5 ns. The output is counting up for 20ns and then counts down for 5ns.

FIR Filter



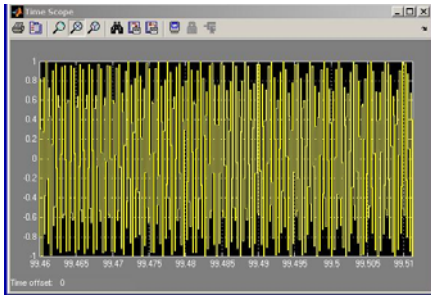
For this design, I am passing a Chirp signal as an input to an FIR Equiripple Filter where the filter parameters are specified by the FDATool. The output spectrum is viewed through a Hamming window. We can change the options of the Spectrum Scope for viewing through different windows.

FDATool

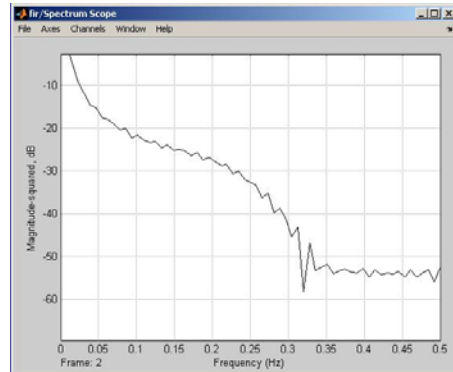


The FDATool allows to design a filter using desired parameters. We can choose between the various options provided and design the filter. Once we do that, the Magnitude response graph on the FDATool is updated and displays the response of the designed filter.

Simulation results for FIR Filter

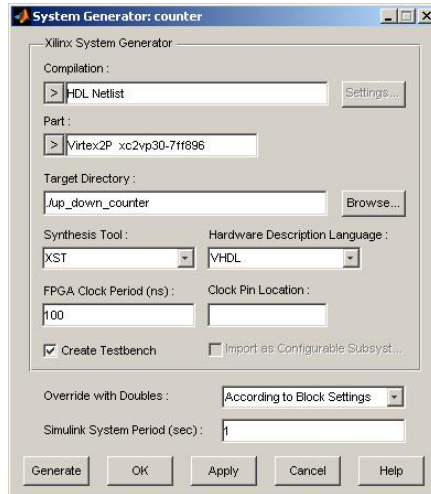


Input Chirp Signal



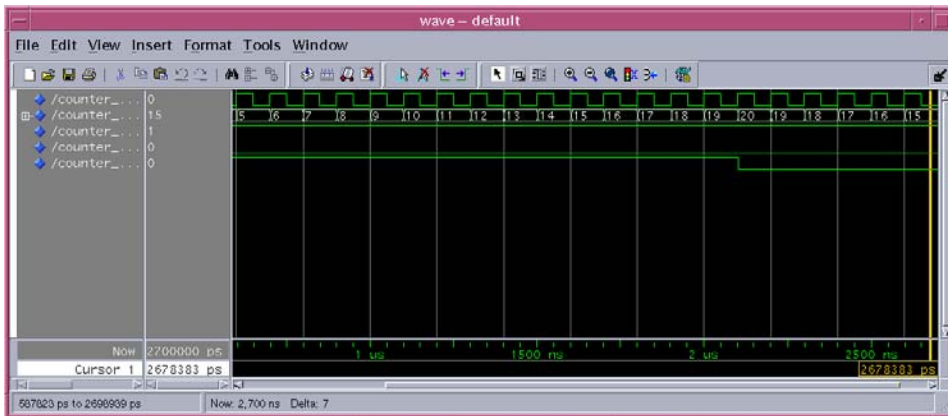
Output Spectrum

The System Generator Block

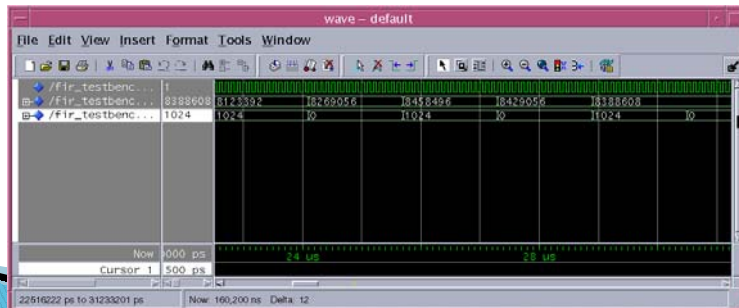
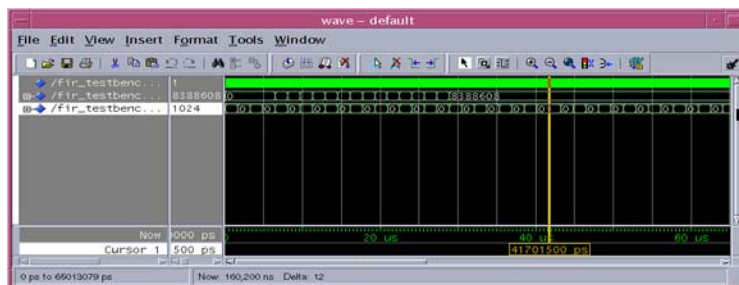


The System Generator block is used to generate the HDL code for the Simulink design. In this block, we specify the family of FPGAs to be targeted, the target directory, the Synthesis tool to be used (Can be XST, Synplify Pro or Leonardo Spectrum) and the HDL. It also creates a testbench by selecting the option. Once we make the selections and hit the generate button, the VHDL files are generated.

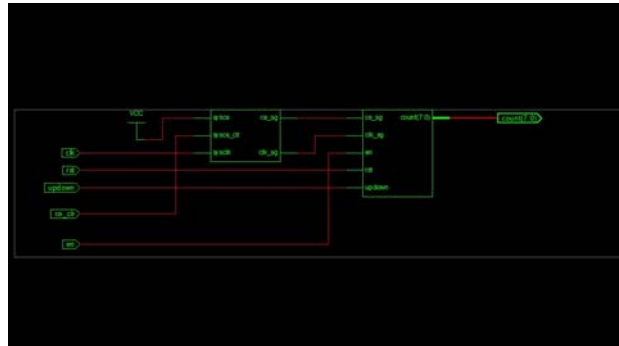
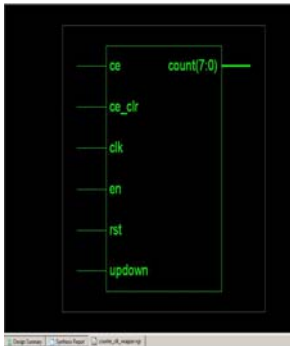
Pre-Synthesis simulation results



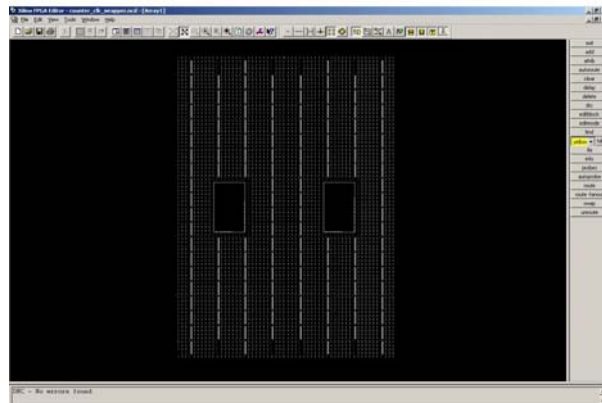
Pre-Synthesis simulation results



Synthesis result



Place and Route Result



Converting Xilinx VHDL to Generic VHDL:

- ▶ The Xilinx VHDL code uses the Xilinx Core Libraries, Unisim and XilinxCoreLib.
- ▶ We need to automate the process for converting the Xilinx VHDL to Generic VHDL.
- ▶ A script can be created in Perl or Awk which reads the Xilinx VHDL code and looks for different Xilinx specific components.
- ▶ For each component, it scans through the Library source files where the entities are defined and searches for the required component entity.
- ▶ After finding the desired entity, the contents of the entity are copied onto another VHDL file which also has the entities of other components parsed by the script.

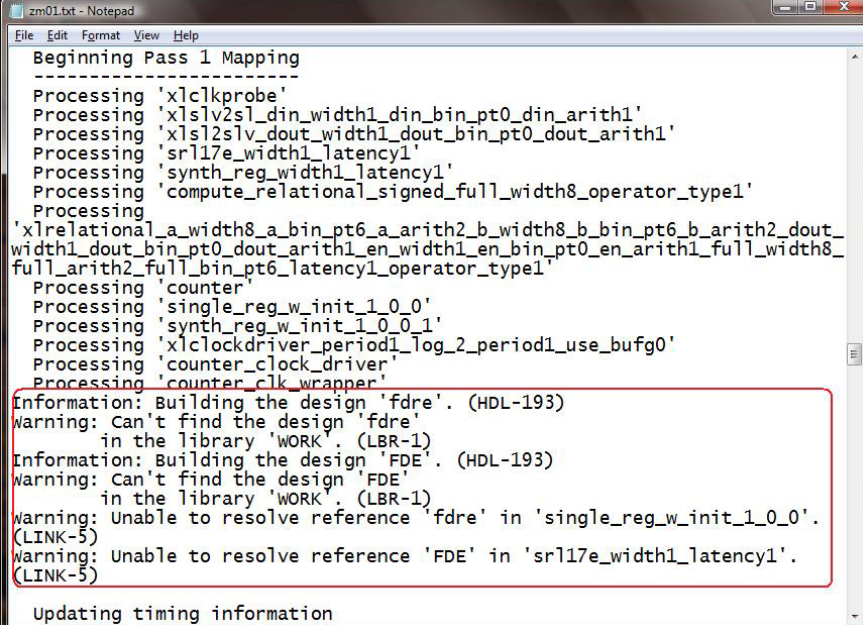
All the Xilinx specific components which are in the Xilinx VHDL code are present in these 2 libraries. So, by scanning and copying the component entities into another file, we can compile that VHDL file which uses only standard libraries. And by adding that VHDL file to the synthesizing list, we can eliminate the use of the library and in that way, the code will be generic.

Using the Manual Process

- ▶ Since the design I am using is small, I was able to manually add the components onto a different file and then add the file to the synthesis list.
- ▶ This will ensure that the code is not using the Xilinx Libraries but just using the generic equivalent entities stored in a different file. These entities depend on only standard libraries like IEEE.

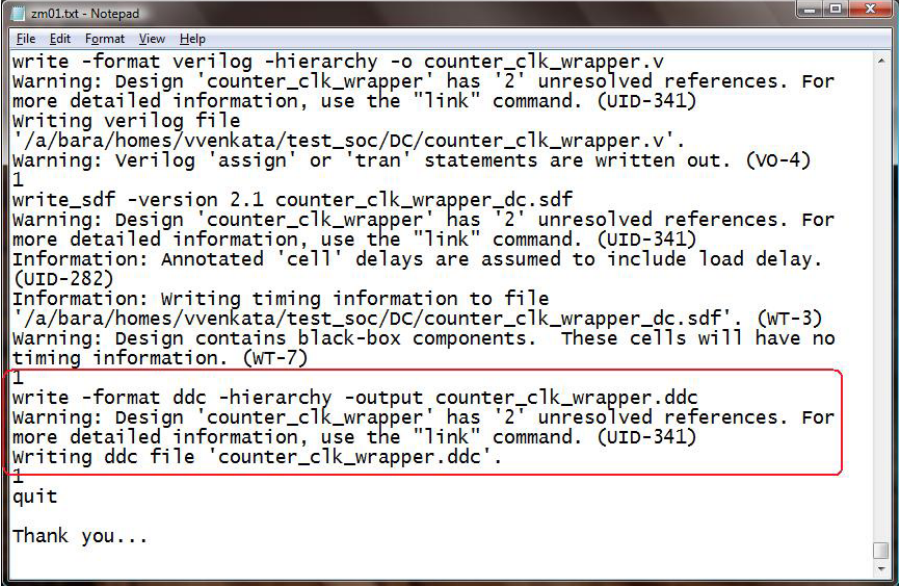
In the manual process, I am taking the Library source file and copying the entity whose component is instantiated in the main code into a different VHDL file. In the counter example which I am using, there are some components which belong the Unisim library, like FDRE and FDE which are D Flip Flops with and without reset respectively. I copied these entities onto a different file and compiled that file and added it to the synthesis list. This resulted in a change in the synthesis result and it was able to recognize and process these components.

Synthesis result before performing manual conversion



```
zm01.txt - Notepad
File Edit Format View Help
-----
Beginning Pass 1 Mapping
-----
Processing 'xclkprobe'
Processing 'x1s1v2s1_din_width1_din_bin_pt0_din_arith1'
Processing 'x1s12s1v_dout_width1_dout_bin_pt0_dout_arith1'
Processing 'sr117e_width1_latency1'
Processing 'synth_reg_width1_latency1'
Processing 'compute_relational_signed_full_width8_operator_type1'
Processing
'xlrelational_a_width8_a_bin_pt6_a_arith2_b_width8_b_bin_pt6_b_arith2_dout_
width1_dout_bin_pt0_dout_arith1_en_width1_en_bin_pt0_en_arith1_full_width8_
full_arith2_full_bin_pt6_latency1_operator_type1'
Processing 'counter'
Processing 'single_reg_w_init_1_0_0'
Processing 'synth_reg_w_init_1_0_0_1'
Processing 'x1clockdriver_period1_log_2_period1_use_bufg0'
Processing 'counter_clock_driver'
Processing 'counter_clk_wrapper'
Information: Building the design 'fdre'. (HDL-193)
warning: Can't find the design 'fdre'
in the library 'WORK'. (LBR-1)
Information: Building the design 'fde'. (HDL-193)
warning: Can't find the design 'fde'
in the library 'WORK'. (LBR-1)
warning: Unable to resolve reference 'fdre' in 'single_reg_w_init_1_0_0'.
(LINK-5)
warning: Unable to resolve reference 'fde' in 'sr117e_width1_latency1'.
(LINK-5)
Updating timing information
```

From the figure, we can see that, on performing synthesis, the Design Compiler could not find the design FDRE and FDE which belong to the Unisim library. Though this appears as a warning message, it can cause potential problems at later stages.



```
zm01.txt - Notepad
File Edit Format View Help
write -format verilog -hierarchy -o counter_clk_wrapper.v
Warning: Design 'counter_clk_wrapper' has '2' unresolved references. For
more detailed information, use the "link" command. (UID-341)
writing verilog file
'/a/bara/homes/vvenkata/test_soc/DC/counter_clk_wrapper.v'.
Warning: Verilog 'assign' or 'tran' statements are written out. (VO-4)
1
write_sdf -version 2.1 counter_clk_wrapper_dc.sdf
Warning: Design 'counter_clk_wrapper' has '2' unresolved references. For
more detailed information, use the "link" command. (UID-341)
Information: Annotated 'cell' delays are assumed to include load delay.
(UID-282)
Information: Writing timing information to file
'/a/bara/homes/vvenkata/test_soc/DC/counter_clk_wrapper_dc.sdf'. (WT-3)
Warning: Design contains black-box components. These cells will have no
timing information. (WT-7)
1
write -format ddc -hierarchy -output counter_clk_wrapper.ddc
Warning: Design 'counter_clk_wrapper' has '2' unresolved references. For
more detailed information, use the "link" command. (UID-341)
writing ddc file 'counter_clk_wrapper.ddc'.
1
quit
Thank you...
```

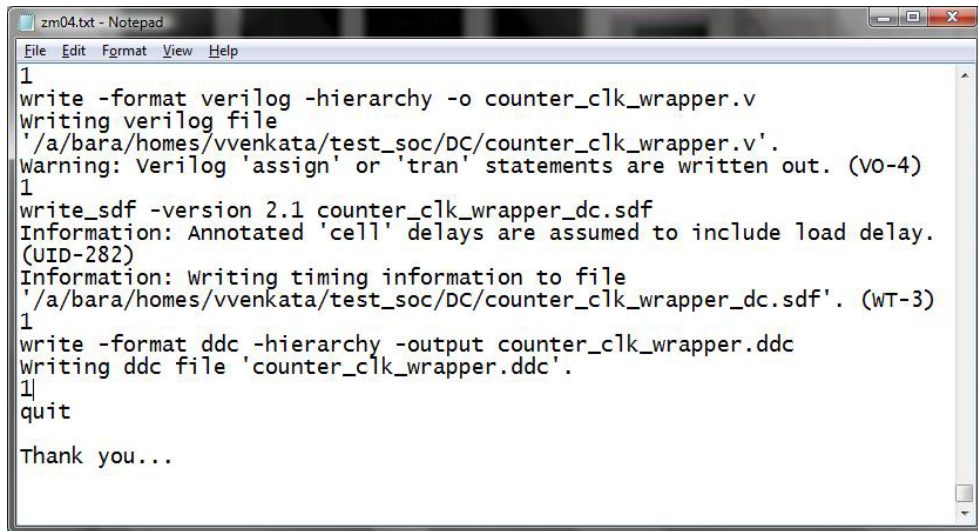
The warning message about 2 unresolved references refer to the FDRE and FDE components.

Synthesis result after performing manual conversion

```
zm04.txt - Notepad
File Edit Format View Help
Beginning Pass 1 Mapping
-----
Processing 'xlclkprobe'
Processing 'x1s1v2s1_din_width1_din_bin_pt0_din_arith1'
Processing 'x1s12s1v_dout_width1_dout_bin_pt0_dout_arith1'
Processing 'FDE'
Processing 'sr117e_width1_latency1'
Processing 'synth_reg_width1_latency1'
Processing 'compute_relational_signed_full_width8_operator_type1'
Processing
'xlrelational_a_width8_a_bin_pt6_a_arith2_b_width8_b_bin_pt6_b_arith2_dout_width1_dout_bin_pt0_dout_arith1_en_width1_en_bin_pt0_en_arith1_full_width8_full_arith2_full_bin_pt6_latency1_operator_type1'
Processing 'counter'
Processing 'FDRE'
Processing 'single_reg_w_init_1_0_0'
Processing 'synth_reg_w_init_1_0_0_1'
Processing 'xlclockdriver_period1_log_2_period1_use_bufg0'
Processing 'counter_clock_driver'
Processing 'counter_clk_wrapper'

Updating timing information
```

After performing manual conversion, we can see that the FDRE and FDE components were processed without any warning messages.



```
zm04.txt - Notepad
File Edit Format View Help
1
write -format verilog -hierarchy -o counter_clk_wrapper.v
writing verilog file
'/a/bara/homes/vvenkata/test_soc/DC/counter_clk_wrapper.v'.
Warning: Verilog 'assign' or 'tran' statements are written out. (VO-4)
1
write_sdf -version 2.1 counter_clk_wrapper_dc.sdf
Information: Annotated 'cell' delays are assumed to include load delay.
(UID-282)
Information: Writing timing information to file
'/a/bara/homes/vvenkata/test_soc/DC/counter_clk_wrapper_dc.sdf'. (WT-3)
1
write -format ddc -hierarchy -output counter_clk_wrapper.ddc
writing ddc file 'counter_clk_wrapper.ddc'.
1
quit

Thank you...
```

We are also not getting any warning messages about unresolved references in the end. This code is therefore generic compared to using the Xilinx core libraries.

Automating the process for conversion

- ▶ To automate the process of converting Xilinx components to generic components, we need to parse the files and look for specific components and replace them with their generic equivalent.
- ▶ A PHP script can be written to parse a file and search for a component and replace with its equivalent generic component.
- ▶ The components are stored in an SQLITE database which is opened from the PHP script.

The PHP script *parsefile.php* contains a function which opens an SQLITE database, executes the query and returns the result as a hash array which is a key-value pair. This hash array is looped through and all keys are stored in one array and its equivalent values in another array. So if the 1st component of the 1st array is a certain Xilinx component name, the 1st component of the 2nd array will be its equivalent generic component name.

Once the names are retrieved and stored in arrays, the “**fopen**” function of PHP can be used to open a file. We can either open the file in read mode and replace the components are store in a new file which is in write mode, or we can open a file in read + write mode. The “**preg_replace**” function can be used to replace the Xilinx component names with the generic names and “**fclose**” closes the file after all names are replaced.

As we discover more Xilinx components in the future, we can just add them to the database and the SQL query fetches them to the arrays. This avoids manually adding the names to the arrays. A User Interface can be created to insert the names to the database and replace the components.

What is SQLITE?

- ▶ SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine. Its transactions are ACID compliant even after system crashes and power failures.
- ▶ A complete database can be stored in a single disk file which makes it very portable.
- ▶ Faster than popular client/server database engines for most common operations.

ACID stands for atomic, consistent, isolated, and durable. A good Database Management System must be ACID compliant.

An **atomic transaction** is a series of database operations which either *all* occur, or *all* do not occur ("fail", although failure is not considered catastrophic). A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright.

A **consistent** transaction is one that does not violate any integrity constraints during its execution. If a transaction leaves the database in an illegal state, it is aborted and an error is reported.

Isolation refers to the ability of the application to make operations in a transaction appear isolated from all other operations. This means that no operation outside the transaction can ever see the data in an intermediate state.

Durability property guarantees that transactions that are successfully committed will survive permanently and will not be undone by system failure.

SQLITE can be used for supporting terabyte-sized databases and gigabyte-sized strings and blobs but cannot be used when lot of concurrent write transactions can occur as entire database is locked for a transaction.

Some of the functions used in the script for SQLITE operations

- ▶ `$db=sqlite_open("xilinxcomponentdb", &$errmsg)`
→ Opening a database.
- ▶ `$result = sqlite_query($db,$query)` → Executes an SQL query stored as a string in `$query` variable.
- ▶ `$output = sqlite_fetch_all($result, SQLITE_ASSOC)`
→ Fetches the result in an array. The result is an array of key-value pairs of Xilinx and Generic components.

These are included in a function called `get_names`. The function code is provided below:

```
function get_names() {
    // Retrieve the Generic Component Information based on the Xilinx name generated
    // Open the database
    $db=sqlite_open("xilinxcomponentdb", &$errmsg);
    if (!$db) {
        echo("Coudn't connect to database <br />");
        echo $errmsg;
    }
    // The SQL query fetches all components. This query can be changed //for specifically
    fetching other components using a WHERE clause.
    $query = "SELECT * FROM xilinxcomponent d";
    $result = sqlite_query($db,$query);
        if (!$result) {
            echo "Error with Query! <br />";
            echo "$query <br />";
            echo sqlite_error_string(sqlite_last_error($db));
        }
    $output = sqlite_fetch_all($result, SQLITE_ASSOC);
    return $output;
}
```

Some of the functions used in the script for fileoperations

- ▶ `$fh = fopen("fir_files.vhd", "r")` → Opens the file `fir_files.vhd` in read mode.
- ▶ `$line = fgets($fh)` → Used in a loop, parses the line and stores in a variable.
- ▶ `$line1 = preg_replace($orignames, $genericnames, $line)` → replaces names in the files present in 1st array with the equivalent names from 2nd array.
- ▶ `fwrite($fh1, $line1)` → write to a different file.
- ▶ `fclose($fh)` → Close the file

The code of parsefile.php script starting from the function call is provided below. The function code should also be included in the main code within the `<?php` and `?>` tags.

```
//Parsefile.php parses a file, searches for a component from an array //populated from an SQLITE database and replaces it with an equivalent //component.
<?php
//Call the get_names function
$output = get_names();
//Create 2 arrays for original names and generic name
$orignames = array();
$genericnames = array();
foreach ($output as $row) {
//Add each xilinx component to 1st array and generic component to 2nd array
    $orignames[] = $row['xilinx_name'];
    $genericnames[] = $row['generic_name'];
    echo "Xilinx Name is " . $row['xilinx_name'] . " and Generic name is " . $row['generic_name'] . "<br />";
}
foreach ($orignames as &$value) {
    $value = " ".$value." ";
}
// file open in read mode and new file in write mode
$fh = fopen("fir_files.vhd", "r") or exit("Unable to open file!");
$fh1 = fopen("fir_files_new.vhd", "w") or exit("Unable to open file!");
while(!feof($fh))
{
    $line = fgets($fh);
    // replace each component from 1st array with equivalent //component from 2nd array
    $line1 = preg_replace($orignames, $genericnames, $line);
    fwrite($fh1, $line1);
}
fclose($fh);
?>
```

Implementing the Generic VHDL on an ASIC:

- ▶ Once the VHDL code is converted to generic code, it can be implemented onto an ASIC using an IBM 405-S PPC core as CPU.
- ▶ This core can be synthesized using a tool called coreConsultant
- ▶ Due to licensing issues and some installation problems, the core has not been synthesized yet.

The IBM PowerPC® 405-S core is a fully synthesizable implementation of the PowerPC 405 32-bit RISC CPU core employing the scalable and flexible Power ISA™ 2.03 and optimized for embedded applications. The PowerPC 405-S core's performance, power specifications, and design attributes make it an ideal solution for emerging wired communications, storage and pervasive computing applications.

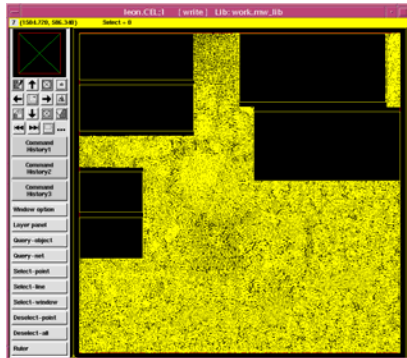
- ▶ The ASIC development flow was illustrated on Slide 4.
- ▶ Logical Synthesis is performed by the Synopsys DC tool which creates the netlists and writes an SDF file.
- ▶ In the next step, floorplanning and power-planning are performed by JupiterXT tool by Synopsys. This generates an output floorplan in the form of a def file.
- ▶ The def file is read by the Physical compiler which performs Physical Synthesis.
- ▶ Following this, the Astro tool by Synopsys performs routing.
- ▶ The design is finally implemented on an ASIC after post-synthesis simulation.

HW-1 in 652 course provides a tutorial for ASIC Development flow. The link can be found here:

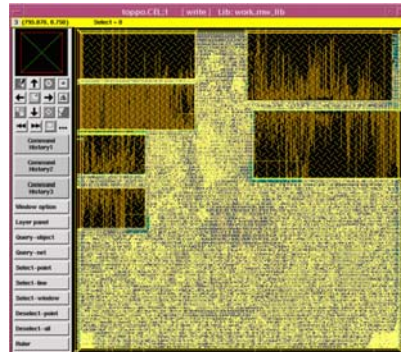
http://www-ece.engr.utk.edu/ece/bouldin_courses/private_html/652-hw1/index.html

Hw-9 in 652 course provides a tutorial for floorplanning and power-planning using JupiterXT

Using JupiterXT & Astro for Placement and Routing



Floorplan after
Placement by Jupiter
XT

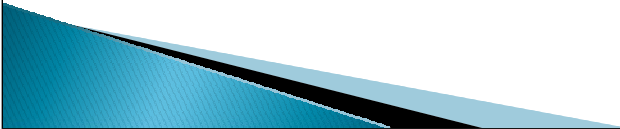


Routed view of
the Floorplan after
running Astro

These images have been taken by running 652 Homework 1.

Further Development

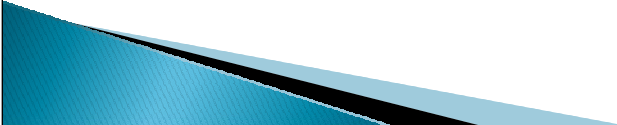
- ▶ Implementing an ASIC with the IBM 405-S PPC core once the license issues are resolved.



References:

- ▶ Insecta group at University of Berkeley:
<http://bwrc.eecs.berkeley.edu/Research/Insecta/default.htm>
- ▶ Xilinx System Generator:
http://www.xilinx.com/ise/optional_prod/system_generator.htm
- ▶ CoreConsultant user guide:
www.synopsys.com/designware/docs/doc/coretools/latest/ccug.pdf
- ▶ 652 HW-1:
http://www-ece.engr.utk.edu/ece/bouldin_courses/private_html/652-hw1/index.html
- ▶ 552 HW-4:
http://www-ce.engr.utk.edu/ece/bouldin_courses/552/homework_4.html

Acknowledgements

- ▶ I would like to thank Dr. Don Bouldin, my MS Advisor for providing continuous support during this project.
 - ▶ Dr. Mark Buckner from ORNL for his helpful suggestions.
 - ▶ Dr. Itamar ElHanany and Dr. Syed Islam for being part of the committee.
 - ▶ Jimmy Webb from IT Services for helping with the installations.
- 

THANK YOU!!

