

Porting EDIF Netlists to the Viva®¹ Environment for Integrated Custom Computing Applications

Sreesa Akella, Heather E. Wake, James P. Davis and Duncan A. Buell

Department of Computer Science and Engineering
University of South Carolina
Columbia, South Carolina 29208
akella@engr.sc.edu

Abstract

We describe a procedure to import an EDIF netlist to the Star Bridge Systems Viva® design environment. The Viva tool is used for modeling and programming the Star Bridge Systems HC 36m Hypercomputer system. The EDIF netlist is generated by synthesizing a functionally working VHDL model of a design, using standard commercial synthesis tools. Importing EDIF into Viva would provide the ability to implement designs previously modeled in VHDL and to obtain reference points with regard to Viva-versus-VHDL performance.

1 Introduction

The HDL-based design methodology is time-tested and provides a very stable design flow. A design can be modeled using any of the hardware description languages like VHDL or Verilog; for our purposes we look at designs modeled in VHDL. The model can be tested for correct functionality using simulation tools like ModelSim. Once the design functionality has been verified, the design can then be synthesized and an EDIF [3] netlist generated. The Star Bridge Systems Viva environment provides us the ability directly to implement this design on the HC 36m reconfigurable hypercomputer by importing the EDIF netlist. This provides us the opportunity to compare the performance of VHDL-modeled designs against designs modeled directly in Viva and also the capability to implement designs previously modeled in VHDL directly on hardware. This also allows us to avoid rewriting working VHDL code for the parts of a computation that do not have a critical effect on the time or space utilization of resources. We describe here a procedure to import successfully an EDIF netlist into the VIVA environment.

2 Process Flow for importing EDIF to Viva

The process flow for importing EDIF to Viva is given in Figure 1. The design is modeled in VHDL. It is better if we have a hierarchical and modular description of a design instead of a single architecture describing the entire design, since a single architecture will lead to greater

¹ Viva®, Hypercomputer® and Star Bridge® are registered trademarks of Star Bridge Systems, Inc.

complexity in the EDIF netlist generated. The model is then verified for its functionality using the ModelSim simulation tool.

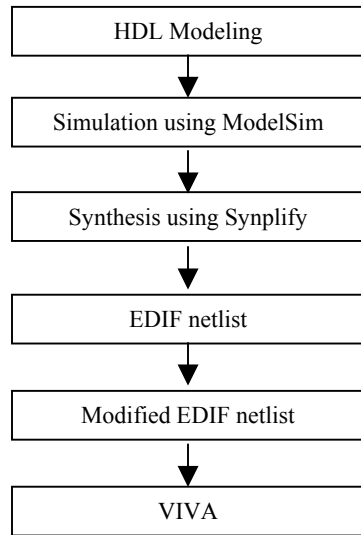


Figure 1. Process Flow for importing EDIF into VIVA

2.1 Synthesis

The verified design can then be synthesized using any of several FPGA synthesis tools; we have used both the Synopsys FPGA Compiler II and the Synplicity Synplify Pro tools [1], [2]. Our target applications make heavy use of multiplication, and we note a compatibility problem with the use of the hardware multipliers on the FPGAs. The Synplify Pro tool, when synthesizing multiplier modules that map to the MULT18x18 blocks of the Virtex II XC2V6000 chips, aligns the MSB of the bit vectors with bit 0 of the MULT18x18 block. This apparently is the case when a ‘rename’ is used to declare a bus. Also, the input and output vectors are defined as arrays which are always defined with a ‘rename’. Invariably, then, the MSB of the bit vectors is mapped to bit 0 of the MULT18x18s, something that is incompatible with what Viva expects. Viva assumes the MSB to be bit 17 of the MULT18x18 instead of bit 0. Since the MULT18x18 blocks use bit 17 as a sign bit and Synplify maps the bit in the opposite order with LSB being bit 17, the performance of the multiplier is affected. The Synopsys FPGA Compiler tool maps the multiplier bits in the correct order and does not use the ‘array’ and ‘rename’ for port definitions. For these reasons the FPGA Compiler tool has been principally employed for synthesis purposes in the work we present here.

The synthesis is one of the important steps in the process, since it requires a certain tweaking of the synthesis options. The options that need to be set are given below.

Device	:	Virtex II XC2V6000
Speed	:	-4
Package	:	FF1152
I/O insertion	:	Disable

The last option of I/O buffer insertion is important. If this option is not disabled, the synthesis tool would insert I/O buffers into the EDIF netlist, and these buffers would then have to be manually removed before the EDIF netlist could be imported into Viva. If this is not done, the Viva environment will generate Xilinx tool errors when the design is implemented. The Viva environment synthesizes its designs using cells defined in its default library, which is placed within a file, named as FPGAStrings.txt. Upon compilation the tool places input and output buffers for each input and output of the design. Thus, if our design netlist already has input and output buffers defined for its ports, the Xilinx place and route tools would generate multiple driver errors during the implementation phase. To avoid this, the I/O buffer insertion option is disabled during synthesis.

2.2 EDIF Modifications

The EDIF netlist generated by this process cannot be directly imported into Viva; several modifications have to be made before the EDIF netlist becomes compatible with what Viva expects. The EDIF netlist consists of library and cell definitions along with the main cell definition and associated sub cell definitions. We do not need all of the EDIF netlist, so stepwise modifications have to be performed. These modifications are done as given below:

- Pick up the main cell definition along with associated sub cell definitions and LUTs not already defined in the FPGAStrings file
- Modify all library references to Active_lib, as this is the default library referenced by the tool
- Modify the 'view' references to 'net'
- Modify the port references to match the port names present in FPGAStrings file

2.3 Importing EDIF into Viva

After performing the modifications, the netlist could be imported into Viva in three different ways. These are:

- Place the modified cell definitions in the FPGAStrings file. This option is not good as it requires maintenance of multiple versions of this file. The Viva tool always references only one FPGAStrings file and any changes made to this file if not correct would cause Xilinx tool errors. Thus any changes necessitate version control.
- Place the modified cell definitions in a file and have the Viva object's 'EDIFFile' attribute reference it. This option is a better one
- The third option is to mix the cells within the FPGAStrings file and our cells and place them in a file. Then we can have the Viva tool reference this file for its library components. This is the best option since it is the least cumbersome and avoids the need for version control.

2.4 Creating the Viva object

Once we have done this, we need to create the Viva object and set some of its attributes. The following steps list the procedure for creating the object:

- Open the Viva tool and create the signature of our component as shown in Figure 2.
- An important detail in this is that bit strings or bit vectors are not permitted as input or output; the input and output ports of the component must be only single bit data types.
- Have the same names for the ports as defined in the EDIF netlist

- This component could then be converted into a primitive object by setting certain attributes as given below:
 - System : PE5 or PE6 or PE7 or PE8
 - Resource : CLB
 - LibRef : top module name

The System attribute refers to the FPGA system on which the design is to be implemented. The system could be any of the four PEs present on the Star Bridge systems HC 36m hypercomputer.

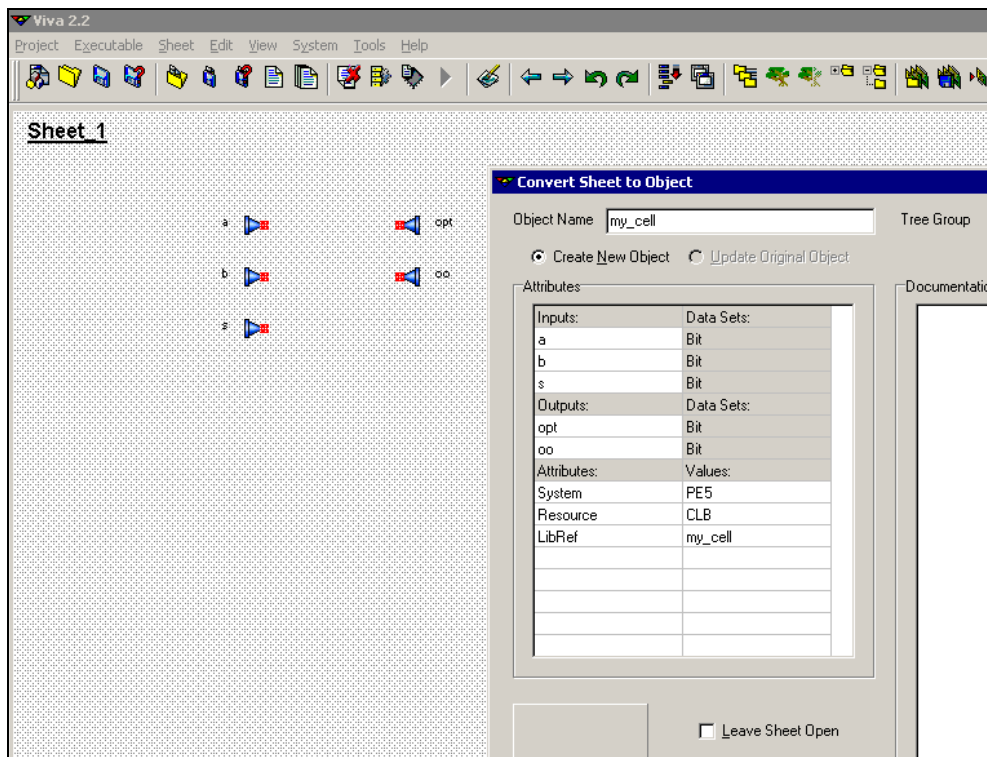


Figure 2. Creating the Viva Object

2.5 Instantiating and testing the object

Once the object is created in Viva, it could then be instantiated as shown in Figure 3. It can then be synthesized and its functionality checked. For instantiating the object we need to follow the following steps:

- Obtain the object from the object tree of the PE system selected after its creation. Then, by selecting the object we can drag it onto the Viva tools graphical editor window.
- Tie the input and output pins of the object, and synthesize.
- Provide test values to the widget form generated by the graphical tool and check the functionality of the design. The graphical tool generates a widget form, which could be used to provide test values for the design and check its functionality.

Finally, when the design has successfully been imported, the resulting module can be used as a library component for other designs just as if it were an object created entirely in Viva.

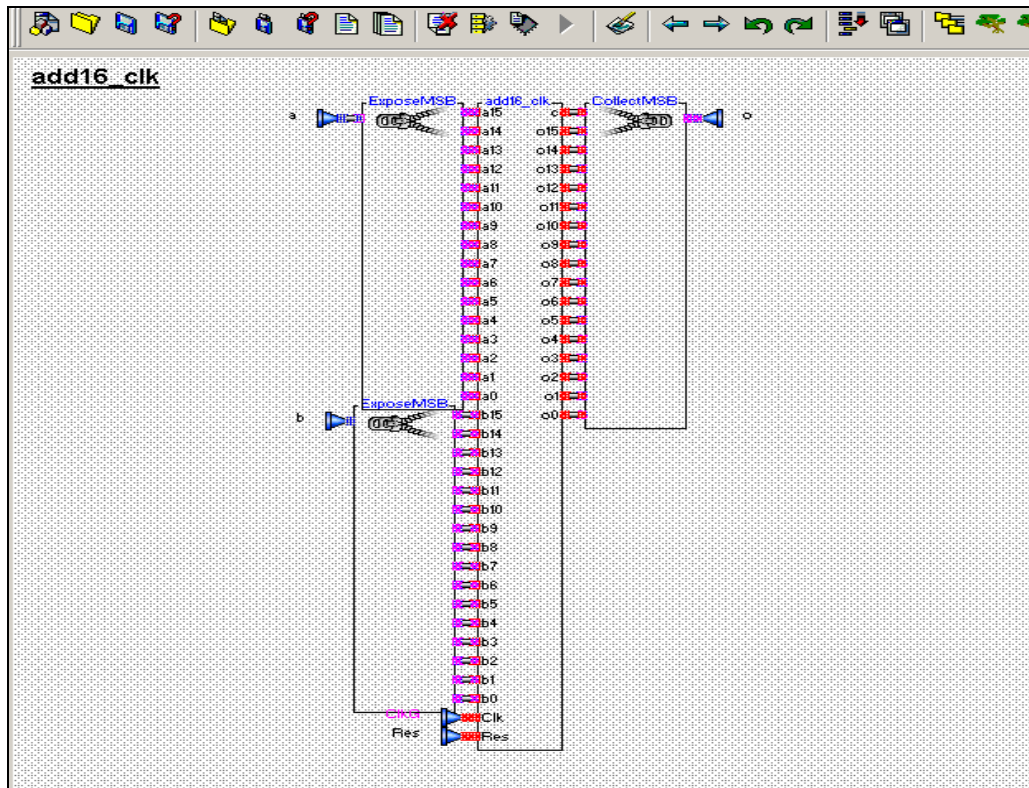


Figure 3. Instantiate the object in Viva graphical editor tool

3 Automating EDIF modifications

Though the EDIF porting of small designs, such as a 16-bit multiplier, is not a difficult task, larger designs pose problems because a great many modifications have to be made before the EDIF netlist is compatible with Viva. A better approach is to break up a hierarchical design, generate netlists piece by piece, import these individual sub module netlists into Viva, and then generate the top design using these sub module objects. Even then, for large hierarchical designs, a number of individual netlists must be modified, and this number will increase as the complexity of the design increases. We are developing an automated process for modifying EDIF netlists using scripts, thus eliminating the laborious process of manually modifying the netlists. These scripts written in Perl [4] will take the EDIF netlist and the FPGAStrings file as inputs and generate the modified EDIF netlist. The scripts will need to perform the following modifications:

- Eliminate unnecessary library definitions
- Eliminate cells already present in the FPGAStrings file
- Modify cell, library, view and port references

The script generate the following three files

- The modified top cell netlist file
- The library file with referenced cells
- The mixed FPGAStrings file

Using the above files we could directly import the EDIF netlist into the Viva environment. The scripts make the porting of large designs into Viva much easier and EDIF porting in general much simpler.

4 Design Experiments and Implementation

The Elliptic Curve Cryptographic addition was chosen as the main test example for our design experiments. The Elliptic curve addition design was imported in two different ways. One was to build bottom up, which required importing several base level cells such as such as the Divide and Conquer multiplier, the Montgomery multiplier, adders, and subtractors of varying bit widths. The other was to import the whole design as a single piece. Similarly, the Divide and Conquer and Montgomery multipliers themselves were built bottom up by importing their base level cells and also by importing them as a single piece. The imported models were then implemented in Viva and their slice usage and timing data was collected. The same designs were modeled using the Viva library objects and corresponding slice usage and timing data collected. A performance comparison was done and results reported.

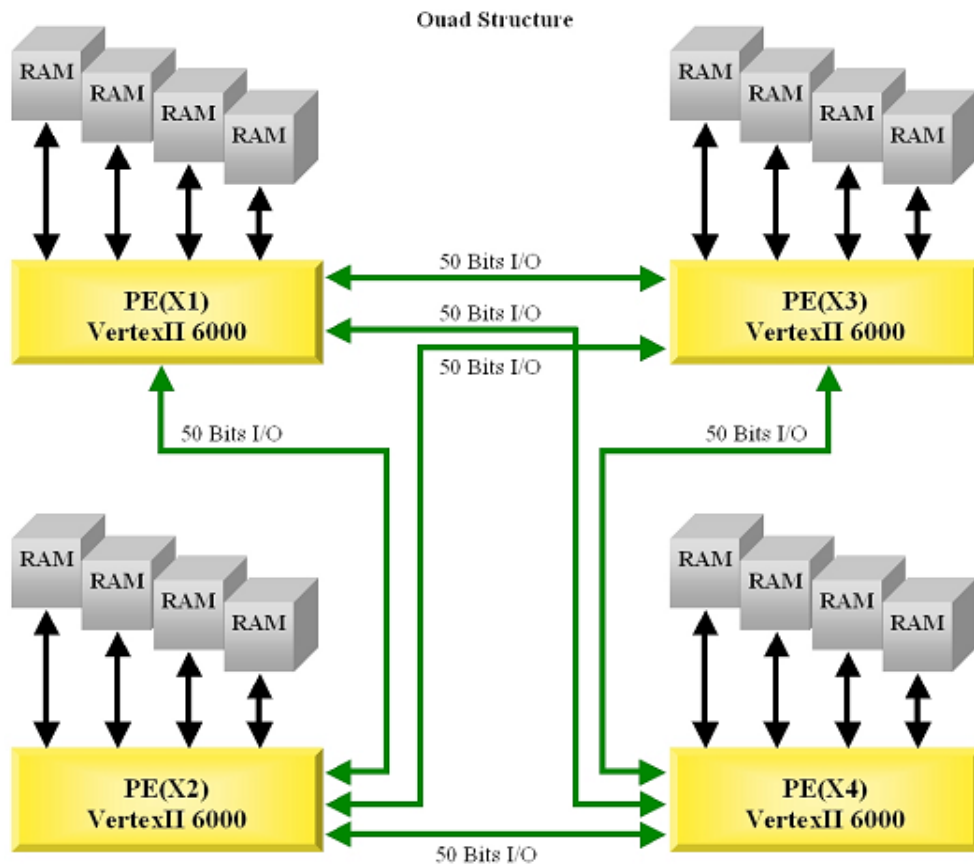


Figure 4. The Star Bridge Systems HC 36m architecture [5]

The Star Bridge Systems HC 36m reconfigurable computer was used as the target platform for experimental implementations. The HC 36m architecture is given in Figure 4. It has a quad structure with four Virtex II XC2V6000 FPGA processing elements and two Virtex II XC2V4000 chips. The processing elements form the building blocks of the architecture.

5 Results and Analysis

The slice usage data for the design examples selected is given in Table 1. The table has three columns for each design. The 'Imported Main Cell' column shows the slice usage data for models imported into Viva and the 'Viva model' column shows slice usage for Viva models. Base cells such as adders, subtractors, and multipliers, which are not designed hierarchically, have only data points, i.e. the 'Imported' and 'Viva' implementations. Hierarchical designs have an additional third data point that gives slice usage data of the hierarchical model built in Viva using the imported sub modules of the design.

Object	Slice Usage		
	Imported Main Cell	Imported Base Cells	Viva Model
16-bit Add	30	-	63
32-bit Add	87	-	151
34-bit Add	92	-	161
64-bit Add	169	-	328
32-bit Subtract	88	-	151
33-bit Subtract	88	-	156
34-bit Subtract	92	-	161
16-bit Multiplier	69	-	105
17-bit Multiplier	73	-	113
32-bit D&Q	269	317	585
32-bit Montgomery	643	723	1674
32-bit ECC Add	7497	7512	~25000

Table 1. Slice Usage results

From Figure 5 we can clearly see that the designs implemented using the imported models take as little as half the number of slices as the designs implemented using Viva models. This difference increases considerably as the size and the complexity of the design increases, as can be seen from the slice usage data collected for the 32-bit ECC Add design. The Viva model uses as much as three times the number of slices used by the imported main cell model. Figure 6 illustrates this fact very clearly. The timing data collected indicated no loss in speed with all the designs running under the default clock cycle speed of 66 Mhz of the HC 36m hypercomputer system.

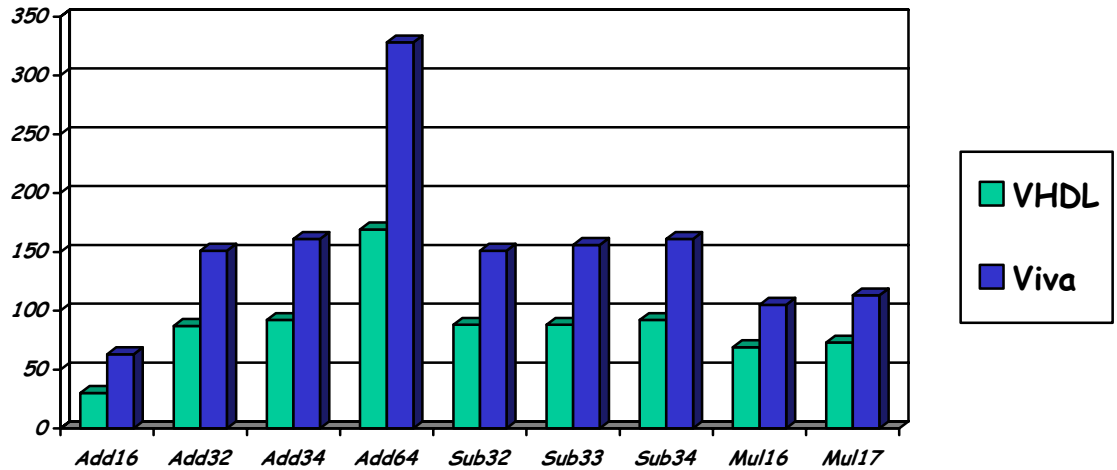


Figure 5. Base units slice usage comparison

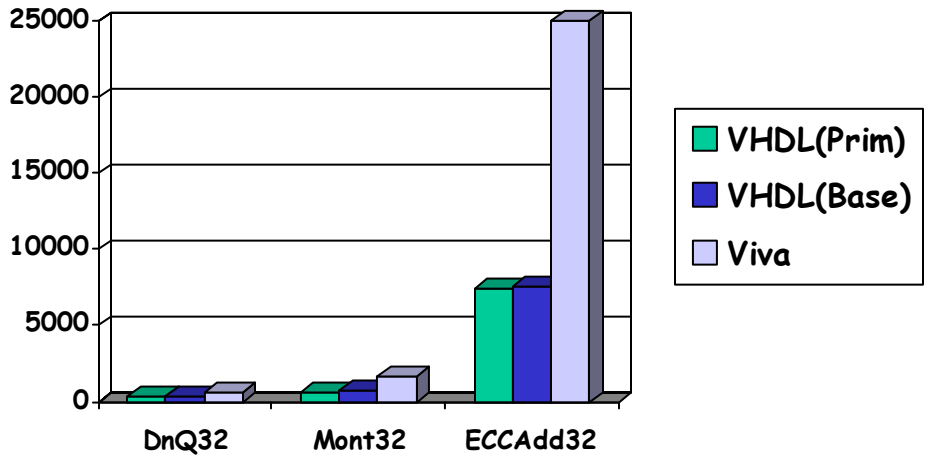


Figure 6. Hierarchical designs slice usage comparison

6 Conclusions and Future work

The HDL-based design methodology provides us with a very stable environment to model and test our designs. The Viva graphical user interface tool provides us the capability to directly implement designs on reconfigurable hardware. Thus importing an EDIF netlist into Viva provides us with the ability to implement designs previously modeled in VHDL, eliminating the need to re-model the same design again in Viva. At the same these imported designs would seem to use fewer resources without any degradation in speed.

The slice usage data indicates that the larger the module imported into Viva the better. However, larger designs have to be imported and the performance data collected for us to ascertain what would be the largest cell that could be imported without any considerable loss in performance.

7 References

- [1] Synopsys FPGA Compiler II™ User Guide. Version 2001.08-FC3.7, January 2002
- [2] Synplify Pro™ Reference Manual, October 2001.
- [3] Introduction to EDIF. www.edif.org.
- [4] Randal L. Schwartz, Tom Christiansen. *Learning Perl*. Second edition, O'Reilly publications, July 1997.
- [5] Star Bridge Systems HC 36 m Hypercomputer hardware reference manual, 2003.