A VHDL Modeling Approach to the Xilinx X4000 Series FPGA

Vince Sanders
Microsystems Prototyping Laboratory
MSU/NSF Engineering Research Center
Mississippi State University
Starkville, MS
(601) 325-7560
sanders@erc.msstate.edu

Bob Reese
Electrical and Computer Engineering Department
Mississippi State University
(601) 325-3670
reese@erc.msstate.edu

Summary
As part of the ARPA RASSP\(^1\) program, Mississippi State University (MSU) has developed and released a VHDL model for the Xilinx X4000 FPGA families. The Xilinx X4000 family is a static RAM based FPGA. The basic logic cell is called a Configurable Logic Block (CLB) and contains two 4-input lookup tables, two D flip-flops, and dedicated carry logic. IO is handled via a versatile Input Output Block (IOB). There are several other logic resources on the chip as well – an on-chip oscillator, fast decoders, tri-state buffers, pullups, high drive buffers, startup logic, and boundary scan capability. The current release of the model supports all logic functionality except for boundary scan and startup logic (work is in progress in this area). Timing functionality of all blocks is supported; package and speed grade specific timing information is read from Xilinx-generated data files. The additional functionality provided by the X4000E family in the form of synchronous SRAM blocks is also included. A VHDL structural model is generated via a Perl5 script from the LCA (Logic Cell Array) netlist file which is created as part of the normal development cycle. The LCA file can include back annotated net delays which are incorporated into the generated model. The VHDL structural model is composed of the primitive components found on the Xilinx X4000 FPGA – Configurable Logic Blocks, Input-Output blocks, internal oscillator, etc. The X4000 module library has been tested with both Mentor Quick-VHDL and Vantage VHDL development environments. The X4000 VHDL model is available via the WWW; the LCA to VHDL model generator script is available for remote execution via the WWW.

Introduction

One of the tasks under the RASSP award to the Microsystems Prototyping Laboratory was the development of VHDL models for COTS devices in several categories. One of the categories were Field Programmable Gate Arrays (FPGAs). Our goals in developing our FPGA VHDL model were:

1. Pick an FPGA family of interest to the RASSP community and the designer community at large.
2. Good model performance (both memory and execution speed).
3. Support all logic functionality of the target FPGA with timing.
4. Support as many system level features as possible (startup functionality, boundary scan, in-system programming) without sacrificing model performance.

Finally, the resulting VHDL model must be freely available to the RASSP community and ARPA research communities.

Approach

The Xilinx X4000 family was chosen as the target FPGA because of its popularity in the FPGA market; this FPGA was also used in the RASSP benchmark circuit by the Lockheed-Sanders team. The Xilinx X4000 family is a static RAM based FPGA. The basic logic cell is called a Configurable Logic Block (CLB) and contains two 4-input lookup tables, two D flip-flops, and dedicated carry logic. The lookup tables can implement two separate 4-variable functions; the output of the tables can also be combined to form a third logic function. The lookup tables can also be used as an asynchronous SRAM, synchronous SRAM (X4000E) or dual port SRAM (X4000E). Combinational outputs can be registered via the flip-flops if desired. IO is handled via a versatile Input Output Block (JOB) which have several configuration options such as registered/non-registered on the input or output signals, tri-state output, programmable output slew rate, pullup/pulldown on output, and output polarity. There are several other logic resources on the chip as well – an on-chip oscillator, fast decoders, tri-state buffers, pullups, high-drive buffers, startup logic, and boundary scan capability.

Because the Xilinx X4000 family is a static RAM based FPGA, this means that its logic configuration is determined at system startup via a serial or parallel programming stream from an associated ROM. The FPGA can also be dynamically reconfigured at any time to perform different logic functions. One of the first modeling decisions (and most important) was to determine whether or not to support in-system programming. Supporting in-system programming would have allowed the model to be used in system simulations which take advantage of the dynamic reconfigurability of the X4000 FPGA. However, there are several negatives associated with supporting in-system programming, namely:

1. Supporting in-system programming requires Xilinx–proprietary knowledge about the programming bit-stream and its relation to on-chip resources. This would have required signing a Xilinx non-disclosure agreement, which would have inhibited distribution of the VHDL model in source form.
2. Model memory performance would suffer because all on-chip logic resources would have to be instantiated because it would not be known ahead of time which logic resources are actually going to be used in the simulation. Model execution would also suffer because each programmable routing point would have to be represented as an active element.
3. The number of designs which actually incorporate dynamic reconfigurability is a low percentage of the total number of X4000 designs.

Given these factors, it was decided not to support in-system programming. This decision allowed the modeling methodology to follow the traditional path of generating a structural model of primitive components based on the initial programming of the device. The next decision involved choosing the modeling level of the primitive components. As part of the the normal Xilinx development cycle, two netlist files are generated which describe the programming of the device. One of the files is a Xilinx Netlist
Format (XNF) description which is a gate level netlist (ANDs, ORs, NANDs, DFFs, etc.). The second file is a Logic Cell Array (LCA) description which is a netlist of the actual on-chip logic resources (CLBs, IOBs, fast decoders, on-chip oscillators, etc.). It was decided to use the LCA file as the basis for generating the structural model because:

1. It was felt that it would be easier to support modeling of the boundary scan and system startup (emulation-only) using this netlist.

2. Some of the more exotic configurations of the logic resources can not be represented in the XNF file, only the LCA file (the programming bit file is produced from the LCA file, not the XNF file).

3. All timing information is given relative to the primitive logic resources, not the gate-level representations.

4. We felt that model efficiency, both memory and speed, would benefit from using the LCA representation over the gate level representation.

Fortunately, the LCA file is in a non-encoded ASCII format which makes for simpler processing. Once this decision was made, the implementation involved creating a VHDL library of the primitive components and a tool for converting the LCA file into a VHDL structural netlist using these components.

X4000 VHDL Component Library

The X4000/X4000E VHDL component library consists of:

- x4000clb: the configurable logic block
- x4000iob: the input/output block
- x4000bufgp: primary global buffer to implement high drive nets
- x4000bufgs: secondary global buffer to implement high drive nets
- x4000buf: tristate buffer
- x4000startup: component for startup sequence emulation
- x4000osc: internal oscillator — 8MHz, 500KHz, 16KHz, 490Hz available

There are some additional logic resources (wide decoders, pullups, pulldowns) which are implemented directly in the structural model without reference to a component model. The CLB and IOB components are further subdivided into functional blocks which implement the lookup table/SRAM functionality, D flip flops, dedicated carry logic, and IOB latching capability. VHDL generates blocks are used to implement only the logic which is actually required inside of a CLB or IOB (i.e., if a CLB does not make use of the D flip flops, then the D flip-flop components are not instantiated). All component models are VHDL 87 compliant.

For illustration purposes, a CLB component instantiation is shown in Figure 1 (other types of component instantiations are similar and will not be discussed). The WD generics are wire delays which have been read from the LCA file. The CLB configuration information is passed via the CLBtags generic which is a record type containing the CLB tag values from the LCA. The tag data specifies everything about how the CLB is configured: whether the flip-flops are used, the carry logic function, how different internal muxes are programmed, etc. The CLBtags record has 76 entries. The CLBTiming and RAMTiming generics are records containing timing information; the contents of these records are set from generics specified at the entity level. The CLBTiming record contains 34 entries; RAMTiming contains 45. These timing values vary by package type and speed grade and are read from a Xilinx data file during model generation. The CLBFuncs generic contains the contents of the lookup tables for the CLB.
Figure 1: CLB Component Instantiation

CLB_BH: x4000CLB

GENERIC MAP (  
  WD_G4 => 3.1 ns,  
  WD_G3 => 4.0 ns,  
  WD_G2 => 3.3 ns,  
  WD_G1 => 2.8 ns,  
  WD_F4 => 3.4 ns,  
  WD_F3 => 4.0 ns,  
  WD_F2 => 3.3 ns,  
  WD_F1 => 3.3 ns,  
  WD_C4 => 6.5 ns,  
  WD_K => 1.9 ns,  
  MGeneration => MGeneration,  
  XGeneration => XGeneration,  
  Ref => Ref'::CLB_BH',  
)

-- CDIR:  
  CLBTags => CLBTagsArrayData(BH_blk),  
  CLBTiming => CLBTimingRecordData,  
  RAMtiming => RAMtimingRecordData,  
-- F = ((F1*F4*F3*F2)) , G = (G10*(G3*G2*G4))  
  CLBFunacs => CLBFunacsArrayData(BH_blk)
)

PORT MAP (  
  G4 => N_CORE_FILT_A_ROM_CNT, -- CORE/FLIT_A/ROM_CNT  
  G3 => N_CORE_RADDR_A1, -- CORE/RADDR_A1  
  G2 => N_CORE_RADDR_A0, -- CORE/RADDR_A0  
  G1 => N_CORE_RADDR_A2, -- CORE/RADDR_A2  
  F4 => N_CORE_RADDR_A2, -- CORE/RADDR_A2  
  F3 => N_CORE_RADDR_A1, -- CORE/RADDR_A1  
  F2 => N_CORE_RADDR_A0, -- CORE/RADDR_A0  
  F1 => N_CORE_RADDR_A3, -- CORE/RADDR_A3  
  C1 => open,  
  C2 => open,  
  C3 => open,  
  C4 => N_RESET_1, -- RESET_1  
  K => N_CLK_1, -- CLK_1  
  Y => open,  
  YQ => open,  
  X => N_CORE_ROUT_A8, -- CORE/ROUTE_A8  
  XQ => N_CORE_RADDR_A2, -- CORE/RADDR_A2  
  COUT => open,  
  CIN => open,  
  GSR => GSR_LCA2VHD
);
Figure 2: CLB Records for Tags, Logic Programming

TYPE CLBTagsArray IS ARRAY (CLBS) OF CLBTagRecord;
CONSTANT CLBTagsArrayData : CLBTagsArray := (
-- other blocks deleted
BH_blk =>
  tagX_F => TRUE, tagXQ_XQ => TRUE, tagSR_C4 => TRUE,
  tagDX_G => TRUE, tagFFX_K => TRUE, tagFFX_RESET => TRUE,
  tagFFX_SR => TRUE, tagFFY_RESET => TRUE, tagG2_G2I => TRUE,
  tagG3_G3I => TRUE, tagF4_F4I => TRUE, tagF_F1 => TRUE,
  tagF_F2 => TRUE, tagF_F3 => TRUE, tagF_F4 => TRUE,
  tagG_G1 => TRUE, tagG_G2 => TRUE, tagG_G3 => TRUE,
  tagG_G4 => TRUE, OTHERS => FALSE
),
-- other blocks deleted
);

TYPE CLBFunsArray IS ARRAY (CLBS) OF CLBFuncRecord;
CONSTANT CLBFunsArrayData : CLBFunsArray := (
-- other blocks deleted
BH_blk =>
-- F = ((F1*F4*F3*F2), G = (G1*G3*G2*G4))
  (funcF => "0000000000000000", funcG => "0101010101010101",
   funcH => (OTHERS => '0'), funcC => (OTHERS => '0'))
),
-- other blocks deleted
);

Figure 2 shows the contents of the tag record and the logic function record passed to the CLB component instantiation of Figure 1. The fields of the tag record are boolean values, while the contents of the lookup tables are passed a string of binary values (16 values required for a 4 variable lookup table). The funcH field in the CLBFunsArray is used for the internal H function of the CLB (three variable function) while the funcC field is used for fast carry logic configuration data.

The current model release supports a rudimentary emulation of the startup sequence; current work is in progress on enhancing the startup sequence emulation and adding boundary scan support. There are no plans for supporting simulation of the readback capability of the X4000 (readback allows the internal programming configuration data and the state of certain nodes to be obtained via a serial data stream). Support for the X4000E family was recently added. The most significant functionality addition of the E-series was the addition of a synchronous read/write capability to the CLB lookup tables when used in RAM mode.

LCA to VHDL Structural Model Conversion

A Perl5 script called lca2vhd is used to convert an LCA file to a VHDL structural model. In addition to reading the LCA file, the script also reads package and pin files within the Xilinx software distribution to determine various features which are package and pin dependent. Files read are:

1. partlist.xct  This file defines characteristics for each package supported in the X4000 family. Information read from this file includes the number of rows and columns of CLBs on the target device (the package/part is contained within the LCA file). This information is used to decide whether splitters are on or off for tristate buffer and decoder operations.
Splitters are used to divide certain long nets sections; this improves the timing characteristics of these nets. IOB to primary and secondary buffer mapping information is also read; primary and secondary buffers are associated with fixed IOBs and these net connections are not present in the LCA file.

2. `{package-name}.pkg` Each package type (i.e., `4013EPQ160`) has a file which maps the IOBs to package pins. This information is needed to determine which IOBs are bonded out because this is helpful in inferring which nets need to be included in the entity declaration as formal ports.

In addition to these two system files, a design specific timing file (`{design-name}.spc`) will be read if present; this file can be optionally generated during the FPGA mapping process and contains package and speed grade specific timing information. Figure 3 shows some typical net and block information present in an LCA file:

```
Figure 3: Net, Block Data from LCA File
Addnet CORE/$1N67_1 IF.X DF.F1 HI.F2 ND.F4 MI.G2 MJ.F2 ME.C3
Netdelay CORE/$1N67_1 DF.F1 3.1 HI.F2 2.8 ND.F4 4.1 MI.G2 5.0 MJ.F2 5.3 ME.C3 3.6
Edtblk BH
Base FG
Equate F = ((F1*F4*F3*F2))
Equate G = (G10(G3*G2*G4))
Endtblk
```

The `Addnet` record defines a net, with net delays defined via the `Netdelay` record (delays are in nanoseconds). CLB, IOB and component configuration information is specified via `Edtblk` records. Net names must be converted to legal VHDL name; some typical name conversions are shown in Figure 4. A command line option to lca2vhd is available to set the length of converted names.

```
Figure 4: lca2vhd Netname conversions
LCA net name: INTF_CORE/$1N180_1
converted to: N_INTF_CORE_1N180_1

LCA net name: INTF_CORE/D_PATH/ADDR_FIFO/$1I2/$1I80/P_0_S01_1
converted to: N_ADDR_FIFO_1I2_1I80_P_0_S01_1
(user specified that VHDL name length be limited to 32 char)
```

Because `lca2vhd` cannot infer bussed signal names, a user option allows LCA net and port names to be replaced with user-specified names via a mapping file. The typical use for this option is to produce vectored signals; whenever this option is used `lca2vhd` does not produce a top-level entity because it is assumed that the user will provide one which has the desired index ordering on the vectored port signals.

**Model Performance**

It has been difficult to evaluate model performance; we have not been able to obtain access to other Xilinx X4000 VHDL models for comparison purposes (Logic Modeling has a SMART model which supports the X4000; Xilinx has an xref2vhd tool close to release but not available yet). Some ad hoc measurements were performed comparing execution speed of a Mentor Quick-VHDL simulation versus a Viewlogic gate-level simulation (using the Viewsim's built-in gate primitives) for a medium size X4000 design; we found that the X4000 VHDL model implementation was 2X-3X slower. Of course, some slowdown is expected because one is comparing simulator built-in gate primitives versus compiled VHDL component modules. Reports from alpha-testers have concentrated on fixing logic errors in the X4000 VHDL modules and conversion errors inside of lca2vhd; no complaints have been received concerning model execution speed. One can guess from this that performance is at least adequate.
WWW Interface

All of the MPL VHDL models are available via
  \text{http://www.erc.msstate.edu/mpl/vhdl/html/models}

In addition to model download capability, a WWW interface has been created for the lca2vhdl script for purposes of remote execution. The user fills out a form selecting various \text{lca2vhdl} options, uploads their LCA file, and triggers execution of \text{lca2vhdl}. The user then receives a compressed tar file which contains the \text{lca2vhdl} log file and the generated VHDL model. The WWW interface is handy for users which do not have Perl5 installed; it also gives users easy access to the latest version of \text{lca2vhdl} without having to worry about maintaining a local copy. WWW response time, and proprietary concerns about sensitive LCA files are deciding factors in whether a user downloads a local copy of \text{lca2vhdl} or uses the WWW interface. Figure 5 contains some screen shots from the WWW user entry form; Figure 6 shows a results screen.

Future Work

Current plans are to finish the boundary scan simulation support and startup sequence emulation. It will be interesting to compare this model's performance versus the Xilinx supported \text{xnh2vhdl} tool when it becomes available. A new model which supports in-system programming could be attractive to those users experimenting with reconfigurable computing systems.

Acknowledgments

Scott Calhoun is the PI of the MPL RASSP project; he was responsible for the WWW interface to the \text{lca2vhdl} tool. Scott Bilik at Lockheed–Sanders was the first external user of the model; he has offered many valuable suggestions for \text{lca2vhdl} operation and helped debug the X4000 VHDL model components.

References

1. \textit{The Programmable Logic Data Book}, Xilinx Corporation, San Jose, CA.
MPL VHDL Model Collection

Xilinx LCA Upload

This form allows you to upload an lca file and optionally a map file for the purposes of converting the lca file to a vhdl entity/architecture pair. The form uses a Perl script, lca2vhdl, to perform the conversion. Please feel free to provide us feedback on the LCA upload process or Xilinx model.

- senders@erc.msstate.edu email address (required)
- /home/sanders/Paper/ browse lca file

Options

- /home/sanders/Paper/ browse pinmap file
- /home/sanders/Paper/ browse spc file

Yes No simple net names

32 maximum converted name length

behv architecture name

Send File Reset

Figure 5: User Entry for LCA file upload
LCA Conversion Results

<table>
<thead>
<tr>
<th>Email</th>
<th><a href="mailto:sanders@ERC.MsState.Edu">sanders@ERC.MsState.Edu</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>LCA File</td>
<td>x4ke_ft01.lca</td>
</tr>
<tr>
<td>Pinlist File</td>
<td>x4ke_ft01.map</td>
</tr>
<tr>
<td>SPC File</td>
<td>x4ke_ft01.spc</td>
</tr>
<tr>
<td>Architecture Name</td>
<td>behv</td>
</tr>
<tr>
<td>Simple Names</td>
<td>no</td>
</tr>
<tr>
<td>Name Length</td>
<td>32</td>
</tr>
</tbody>
</table>

Conversion Results

```
loa2vhd[1.24]: Reading x4ke_ft01.loa ...  
loa2vhd[1.24]: Part type is: 4003PC84  
loa2vhd[1.24]: Reading /ecad/xilinx/x4000e/data/partlist.xct ...  
loa2vhd[1.24]: Reading /ecad/xilinx/x4000e/data/4003pc84.pkg ...  
loa2vhd[1.24]: Reading x4ke_ft01.map ...  
loa2vhd[1.24]: Reading x4ke_ft01.spc ...  
loa2vhd[1.24]: Reading x4ke_ft01.loa ...  
loa2vhd[1.24]: creating x4ke_ft01.x4000_behv.vhd ...  
loa2vhd[1.24]: Will NOT create entity because map file: x4ke_ft01.map specific  
loa2vhd[1.24]: Finished!  
loa2vhd -map x4ke_ft01.map -arch behv -name_len 32 x4ke_ft01.loa  
```

DOWNLOAD MODELS

Thank you for visiting our site. Please contact mpl-vhdl@erc.msstate.edu for any comments, question, or suggestions.

Figure 6: Results from WWW lca2vhd conversion