XAPP123 (v2.0) January 16, 2002

Using Three-State Enable Registers in 4000XLA/XV, and Spartan-XL FPGAs

Summary

The use of the internal IOB three-state control register can significantly improve output enable and disable time. This application note describes how to use hard macros to implement this register in both HDL and schematic based designs.

Introduction

It is common design practice to drive the enable signal of a three-state output or bidirectional I/O with a registered signal. In earlier FPGA devices (e.g., 4000XL), the signal is usually driven by a flip-flop contained within a CLB, whose output must be routed to each IOB that is to be controlled. Output enable and disable times are directly related to the delays associated with the routing from this CLB flip-flop to each IOB. (See Figure 1.)

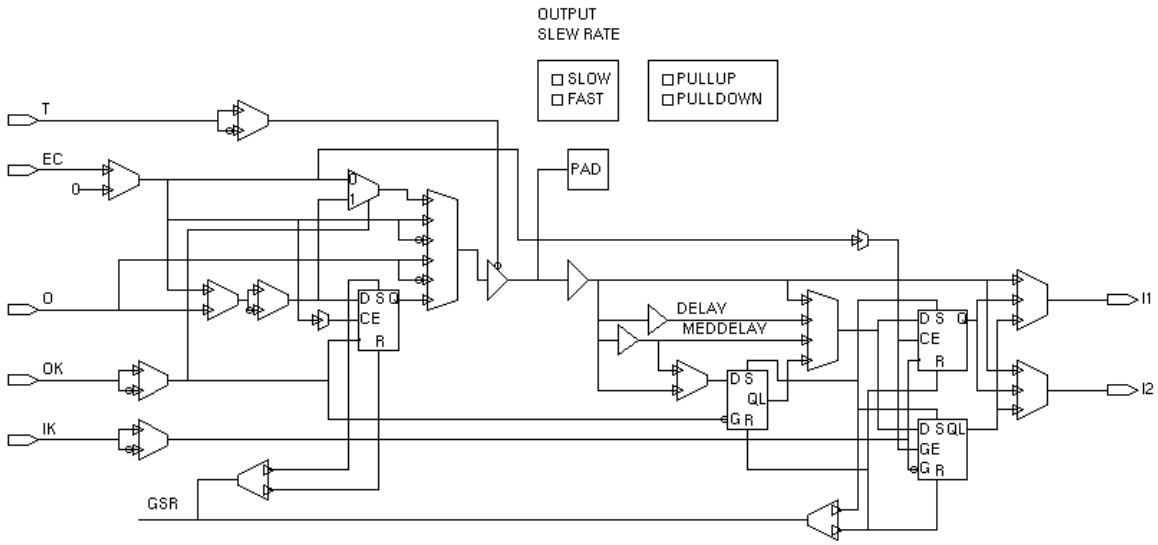

Figure 1: **IOB Configuration of a Spartan-XL Device Without the Three-State Enable Register**

New for the XC4000XLA, XV, Spartan®-XL, and Virtex® devices, Xilinx has added a flip-flop directly inside the IOB that can be utilized to drive the three-state enable signal. Since the flip-flop is local to the IOB, the enable and disable delays are very short. (See Figure 2.)

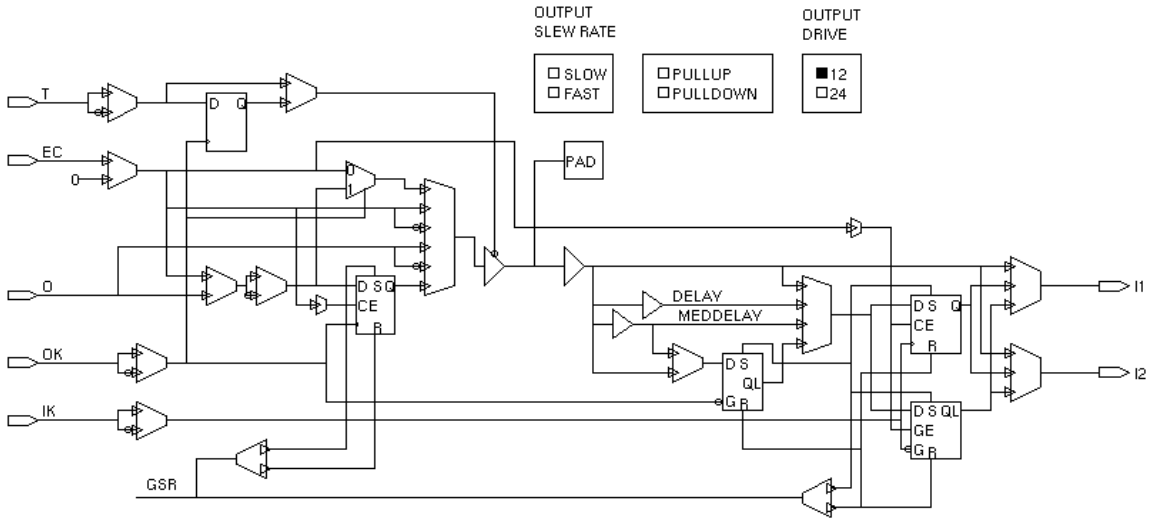

Figure 2: **IOB Configuration of an XLA Device With the Three-State Enable Register**

Process Overview

Under the current implementation software (F3.1i or A3.1i), the mapping algorithms do not automatically utilize this three-state enable register (Virtex Family designs are the exception and do not have this restriction). The best way to utilize this flip-flop is to instantiate a hard macro into the design.

Xilinx provides a PERL script which can be used to help create a hard-macro for each IOB configuration desired. The hard macro can then be instantiated as a “black-box” into the source HDL code (or schematic). The design is then processed as normal through the Xilinx implementation tools, where the hard macro (.nmc file) is automatically integrated during the netlist translation and mapping process.

As is the case with all Xilinx FPGA designs, you can choose to have the implementation tools produce a Verilog or VHDL netlist representation of the implemented design. An associated SDF file is also created, which will contain the delays associated with the three-state enable register. This netlist (and SDF file) can be used within the simulation test bench to verify that the timing paths containing the three-state enable registers are in fact meeting timing goals for the design.

The following sections will explain how to generate the hard macro, and then implement the hard macro into a design.

System Setup

Support for three-state enable flip-flop was added in F1.5i and A1.5i and later releases. This software must be installed for the hard-macro implementation to work.

The `make_macro.pl` script is written using PERL which can be run in the PC or UNIX environment. A version of PERL also comes with the Xilinx tool set so there is no need to install PERL. To verify if PERL has been setup correctly, go to a DOS shell or xterm and type “`xilperl -h`”. The system should return a list of options available to PERL. If the options menu is not returned, verify that the path variable points to $XILINX\bin\nt for the PC and $XILINX/bin/sol for UNIX. If this is correct, verify that the PERL executable is located in the directory just referenced.

Hard Macro Creation

Obtaining the Script

The `make_macro.pl` script is available on the Xilinx FTP site.

For PCs:

ftp://ftp.xilinx.com/pub/utilities/fpga/make_macro.zip

For Unix:

ftp://ftp.xilinx.com/pub/utilities/fpga/make_macro.tar.Z

Both files should be extracted in the $XILINX directory.

Executing the PERL Script

Open a DOS shell or xterm and “cd” to the project directory. Execute the script by typing, `xilperl $XILINX\bin\nt\make_macro.pl`. You will be asked several questions about the type of IOB hard macro you wish to create. When the script has completed it will create two files. The first file is a text file with the same name as the component just created. This file contains information about the component like inputs and outputs, polarity of clocks, clock enables, resets, etc. The second file is the script file to be executed in FPGA Editor. This file will create the hard macro.

Executing the FPGA Editor Script

At the DOS shell or xterm type “fpga_editor”. Once FPGA Editor has come up select "File -> New". Select the “Macro” radio button and select the target "Architecture:", "Device:", "Package:", and "Speed:". The dialog also asks for a Macro File name. This file name is used later in instantiations and black-box symbols. Type in a file name at this time and click on "Ok" to close the dialog. At this point FPGA Editor will continue to load.

In FPGA Editor, select "Tools -> Scripts -> Playback". The "Script Playback" dialog box opens and allows the user to browse for the script file created by the PERL script. Once the file has been selected, click on "Ok" which executes the script file. Now verify that the macro was created correctly, select "Tools -> DRC -> Run", there should be 0 warnings and 0 errors.

Save the macro, click on "File -> Save". This creates a .nmc hard macro file used by NGDBUILD. The .nmc file must be moved to the directory containing the top level .edn or .xnf netlist.

Things to Note About Hard Macros

When the PERL script is executed, one of the questions asked of the user is “Pin Number”. The “Pin Number” is only a reference used for creation of the hard macro. It does not lock the hard macro to the specified pin. This leads to another benefit of the hard macros. If the user has an 8-bit bus, there is no reason to create eight unique macros. One macro can be used multiple times. The only reason for multiple macros is to vary the configuration of the macros.

General Flow

There are two main flows available, HDL and schematic. The HDL flow will discuss Verilog and VHDL using FPGA Express, Symplicity, Exemplar, and Design Compiler. The schematic flow will focus on Foundation.

The main topics of discussion will be coding examples (Figure 3 and Figure 4), schematic macro creation, function simulation, constraints and timing. Timing simulation is not affected by the use of hard macros and will not be discussed. Table 1 explains the input and output pins

possible, but not required, in the hard macro. The pin names must be used exactly when creating black-boxes.

Table 1: **Hard Macro Pin Description and Names**

Pin	Input/Output	Description
T	Input	Tri-State Enable
EC	Input	Clk enable for OFD, IFD, ILFFX
O	Input	Output to Pad
OK	Input	Output Clk and Latch Gate
IK	Input	Input Clk
I2	Output	Input from Pad

HDL Coding Examples

```
VHDL:
  --Declaration of Hard Macro----
    component XXXXX
    port ( IK : in std_logic
    OK:in std_logic
    O:in std_logic
    I2:out std_logic
    T:in std_logic
    EC:in std_logic)
  end component
  -- Instantiation of Hard Macro -----
  -- (port_map_name=>signal_name)
  U1:XXXXX port map (IK=>clk_in, OK=>clk_out, O=>flop_out, I2=>input, T=>tri_en_input, EC=>ff_en)
```

Figure 3: **VHDL Coding Example**

```
Verilog:
  -- Declaration of Hardmacro. Implemented as a separate module -----
  module XXXXX (OK, O, IK, I2, T, EC)
    input OK
    input O
    input IK
    output I2
    input T
    input EC
  -- Instantiation of Hardmacro -----
  XXXXX U1(.IK(clk_in), .OK(clk_out), .O(flop_out), .I2(input), .T(tri_en_input), .EC(flop_enable))
```

Notes:

1. The XXXXX represents the macro name. This is the name of the file created by FPGA Editor (XXXXX.nmc).

Figure 4: **Verilog Coding Example**

When using *Simplicity*, the following attributes need to be added:

For **VHDL** add the following after the component declaration.

```
attribute black_box : boolean;
attribute black_box of XXXXX : component is true;
```

This set of attributes tells the compiler that **XXXXX** is a black box:

```
attribute synthesis_noprune : boolean;
attribute synthesis_noprune of XXXXX : component is true;
```

This set of attributes tells the compiler that XXXXX should not be trimmed. This attribute is only necessary when there are no outputs from the macro.

For **Verilog** add the following on the module declaration.

```
XXXXX U1(.IK(clk_in), .OK(clk_out), .O(flop_out), .I2(input),
.T(tri_en_input),
.EC(flop_enable)) /* synthesis black_box .noprune=1 */
```

When using *Design Compiler*, the following parameter needs to be added to the design script file:

dont_touch “XXXXX”

When using *Exemplar*, only bi-directional macros can be synthesized. Macros that have inputs only will be removed during synthesis. This is a known issue and will be resolved by Exemplar.

HDL Functional Simulation

This section will outline the steps necessary to generate a VHDL or Verilog file for use in a behavioral simulator.

1. Run the design through NGDBUILD. This can be done with the design manager or on the command line This will generate a NGD file, which is a flattened netlist containing all of the levels of design information.
2. Now change directories to the directory containing the NGD file just created. If the Design Manager was used, the file will be in the project directory /xproj/ver#/rev# just run.
3. Run “ngd2vhdl design.ngd” or “ngd2ver design.ngd”. This will create a VHDL or Verilog functional simulation netlist.

   Note: Since the hard macros contain IOB components, the VHDL or Verilog netlist will contain ports specific to the hard macro. They will not be represented as internal nodes.

Schematic Macro Symbol Creation

Open the Foundation Project Manager on the project using the hard macro. From the Schematic Editor menu options, select “Hierarchy -> New Symbol Wizard...”. The Design Wizard dialog box will open. Click on “Next”. Now fill in the Symbol Name, this is the same name as the FPGA Editor hard macro created (.nmc file). The contents section should have “Empty” checked. Click on “Next”.

Click on the “New” button and type in the port name. These are the names in Figure 3, page 4. Enter the port names used in the hard macro you created. Click on “Next”. This will create the symbol and add it to the project library for use.

Schematic Functional Simulation

This section will outline the steps necessary to generate an edif file for functional simulation in Foundation.

1. Implement the design through NGDBUILD.
2. In the Project Manager, select “Tools -> Simulation/Verification -> Checkpoint Gate Simulation Control...”.
3. The Checkpoint Simulation dialog box opens and lists all of the ver#/rev# run for the current project. Select the ver#/rev# just run and click on “Ok”.
4. This will load the Simulator for functional simulation.

Timing and Constraints

There are three issues associated with applying timing constraints to the hard macro.

1. TRCE and Timing Analizer do not evaluate FPGA Editor hard macros.
2. Period constraints created with the Constraints Editor will not be evaluated. (Figure 5)

Here are several suggestions of how to work around these issues. Both of these methods, when used correctly, will report the setup time for the output flop and tri-state flop.

1. For general constraining of paths leading to the hard macro use From-To constraints.

   For Example:

   ```
   timespec TS_clk_out = from (source) to (destination) 20.0ns;
   ```

2. For advanced path constraining, the user must hand edit the PCF file. This text must be added after the schematic start section in the PCF file (Figure 6)

Explanation of the PCF Syntax:

`TIMEGRP “FLOPS” = FFS (“*”);` — Creates a group named FLOPS containing all of the flip-flops in the design.

PIN `“iobenff”` = NET `“tri_en_input”` COMP `“u1/xxxxx”;` — The PIN is defined by the user. The `“tri_en_input”` is the name of the net connected to the D input of the tri-state or output flop. The COMP is the hierarchical name of the IOB macro. The user can create multiple members of a group by keeping the PIN name the same, and varying the NET and COMP names.

`TIMEGRP “sync_1” = PIN “iobenff”;` — Creates a group based upon the `PIN “iobenff”;`

`TS03 = MAXDELAY FROM TIMEGRP “FLOPS” TO TIMEGRP “sync_1” 25;` — Creates a timespec labeled TS03. TS03 specifies the source, destination, and the allowed time.

Pin Locking the Hard Macro.

Pin assignments for the hard macro can easily be done using the following syntax:

```
INST U1 LOC = P14;
```

Simply use the instance name of the macro including the hierarchy. .

```
NET “clk_out” TNM_NET = “clk_out”;
TIMESPEC “TS_clK_OUT” = PERIOD “clk_out” 20.0ns’;
```

Figure 5: **Timing and Constraints Not Evaluated**

```
TIMEGRP “FLOPS” = FFS (“*”);
PIN “iobenff” = NET “tri_en_input” COMP “u1/xxxxx”;
TIMEGRP “sync_1” = PIN “iobenff”;
TS03 = MAXDELAY FROM TIMEGRP “FLOPS” TO TIMEGRP “sync_1” 25;
```

Figure 6: **Advanced Path Constraining**

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/02/99	1.0	Initial Xilinx release.
01/16/02	2.0	Updated.