XAPP378 (v1.0) June 27, 2002

Using CoolRunner-II Advanced Features

Summary

This application note describes how to implement the CoolRunner™-II advanced features in the Xilinx software. These features include the DualEDGE triggered registers, clock divider, CoolCLOCK, DataGATE, Schmitt trigger inputs, and I/O termination types. HDL code examples are available for download, see **Code Examples Download**, page 11.

CoolRunner-II

Xilinx CoolRunner-II CPLDs combine performance and low power in a single device. For more information on the architecture of CoolRunner-II CPLDs, see **References**, page 11. CoolRunner-II CPLDs feature enhanced clocking flexibility and provide design capabilities that significantly reduce power consumption.

All design features discussed in this application note are supported in Foundation™ ISE, WebFITTER™ and WebPACK™ ISE software from Xilinx. For more information on Xilinx software, see **References**, page 11.

Software Attributes

There are two main methods of attribute entry, each with their own advantages. A User Constraint File (UCF) has the advantage of being easily edited while being separate from the design source files. It has the benefit of allowing changes without needing to re-synthesize the source code. For more information on entering constraints with the UCF, see **References**, page 11.

Attribute entry within the source has the advantage of not needing to maintain a separate file for the design constraints. All CoolRunner-II VHDL and Verilog attribute examples are only applicable to the XST synthesis tool.

The attributes that are user definable and specific to the CoolRunner-II CPLD include: CoolCLOCK, DataGATE, Schmitt trigger input, keeper or pullup I/O termination, I/O standards, VREF, and open drain outputs. Some design constraints from this list are software selectable via the ISE Project Navigator GUI. Individual signal constraints must be explicity defined in the methods described above.

Please note that several syntax and help guides for attributes exist within the Xilinx ISE Project Navigator. These help files are updated with each release of software and provide a reliable source of information. The help menu can be located from within ISE under the menu option **Help | ISE Help Contents | CPLD Attributes**.

DualEDGE Registers

CoolRunner-II DualEDGE triggered registers allows designers to reach unprecedented performance levels. CoolRunner-II CPLDs can double system performance by creating DualEDGE triggered (DET) registers. CoolRunner-II DET registers allow data to be registered on both the rising and falling edge of a clock.

CoolRunner-II DET registers can be used for logic functions that include shift registers, counters, comparators, and state machines. Designers must evaluate the desired performance of the CPLD logic to determine use of DET registers.

The DET register can be inferred in any ABEL, HDL, or schematic design. Table 1 lists the inference methods available to create a DET register in CoolRunner-II.

Table 1: **DET Register Inference Summary**

Design Entry	Instantiation Method
ABEL	Use the following syntax: QOUT:=data; QOUT.DEC=clock;
VHDL/Verilog	Infer a dual edge triggered register.
Schematic	Instantiate a FDDn[S][R][E] component.

Examples

A designer can infer a single edge triggered (SET) register in any HDL design. A SET register active on the rising edge of the input clock would require the following VHDL or Verilog syntax.

```
VHDL: if (clock'event) and (clock = '1') then
Verilog: always @ (posedge clock)
```

The required syntax to infer a CoolRunner-II DET register requires the register be active on both the rising and falling edge of the clock. The following VHDL syntax would be used to infer a CoolRunner DET register.

```
process (clock)
begin
  if (clock'event) then
    ...
  end if;
end process;
```

The following Verilog syntax would be used to infer a DET register in CoolRunner-II.

```
always @ (negedge clock or posedge clock)
  ...
```

The DET register is available with all macrocells in all devices of the CoolRunner-II family.

Clock Divider

CoolRunner-II CPLDs provide additional clocking flexibility to the DET register feature with the clock divider. The CoolRunner-II clock divider provides the capability to divide an incoming clock and globally distribute the divided clock to all macrocells. The clock divider provides additional power savings by reducing the toggle frequency of the internal clock network.

The CoolRunner-II clock divider is available on global clock, GCK2, and can divide the incoming clock by 2, 4, 6, 8, 10, 12, 14, and 16. The clock divider creates a 50-50 duty cycle divided clock without affecting T_{CO}. The clock divider output is initialized low by the CPLD power up reset circuitry.

The clock divider circuit includes an active high synchronous reset, referred to as CDRST. When the CDRST signal is asserted, the clock divider output is disabled after the current cycle. When the CDRST signal is de-asserted the clock divider output will become active upon the first edge of GCK2.

Figure 1 illustrates the CoolRunner-II clock divider.

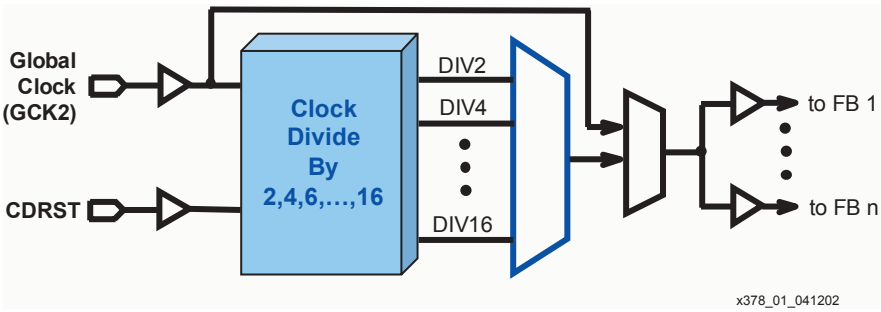

Figure 1: **CoolRunner-II Clock Divider**

The CoolRunner-II clock divider includes a built in delay circuit. With the delay feature enabled, the output of the clock divider will be delayed for one full count cycle. When used, the clock divider does not output a rising clock edge until after the divider reaches the terminal count value. The delay feature is either enabled or disabled upon configuration. The type of clock divider component instantiated will determine if the delay is enabled or disabled. Figure 2 illustrates a timing waveform of the CoolRunner-II clock divider with the delay enabled and disabled.

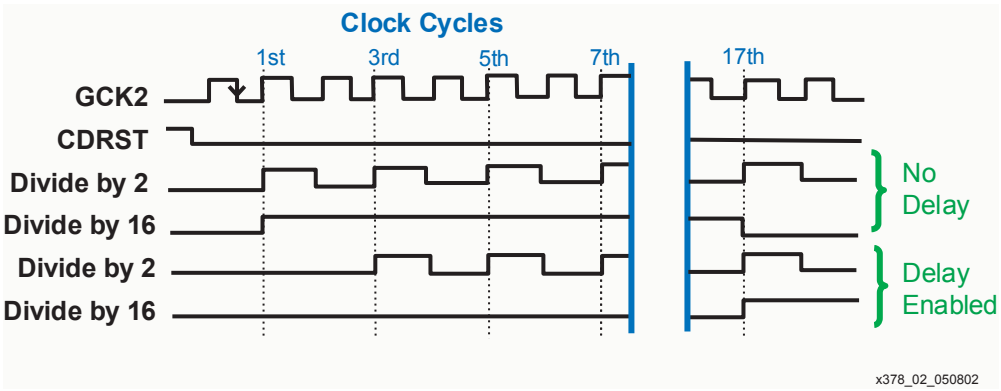

Figure 2: **Clock Divider Waveform**

Xilinx Synthesis Technology (XST) allows a clock divider component to be instantiated directly in the HDL source code. Table 2 lists the available clock divider components that can be instantiated in any ABEL, HDL, or schematic design.

Table 2: **Clock Divider Library Components**

Component	Description
CLK_DIVn	Global Clock Divider Component. No support of the synchronous reset or start delay features. Available: CLK_DIV2, 4, 6, 8, 10, 12, 14, 16
CLK_DIVnR	Global Clock Divider with Synchronous Reset. No support of the start delay feature. Available: CLK_DIV2, 4, 6, 8, 10, 12, 14, 16R
CLK_DIVnSD	Global Clock Divider with Start Delay. No support of the synchronous reset. Available: CLK_DIV2, 4, 6, 8, 10, 12, 14, 16SD
CLK_DIVnRSD	Global Clock Divider with Synchronous Reset and Start Delay. Available: CLK_DIV2, 4, 6, 8, 10, 12, 14, 16RSD

VHDL Example

To design with the CoolRunner-II clock divider in VHDL requires both a component declaration and component instantiation. The component declaration declares the name and interface of the clock divider unit. The VHDL component declaration syntax for using a clock divide by 2, the CLK_DIV2 component is shown here.

```
component CLK_DIV2 is
port (
  CLKIN : in STD_LOGIC;
  CLKDV : out STD_LOGIC );
end component;
```

The component instantiation associates signals with the ports of the clock divider component. If a clock divide by 2 is desired, the CLK_DIV2 component must be instantiated. The incoming clock signal, *clk*, is declared on the CLKIN port and the clock divider output signal, *clk_div_by_2*, is declared on the CLKDV output port. The VHDL syntax is shown here for instantiating the CLK_DIV2 component.

```
U1: CLK_DIV2
port map(
  CLKIN => clk,
  CLKDV => clk_div_by_2 );
```

If a clock divide by 16 with a synchronous reset and start delay is desired, the CLK_DIV16RSD component must be declared and instantiated. The VHDL syntax for the component declaration is shown here.

```
component CLK_DIV16RSD is
port (
  CLKIN : in STD_LOGIC;
  CDRST : in STD_LOGIC;
  CLKDV : out STD_LOGIC );
end component;
```

The component instantiation assigns the port signals. The input clock signal, *clk*, is declared on the CLKIN port. The clock divider reset signal, *clk_div_rst*, is declared on the CDRST port. The clock divider output, *clk_div_by_16*, is declared on the CLKDV port. This component will also enable the start delay circuitry in the CoolRunner-II clock divider. The syntax to instantiate the CLK_DIV16RSD component in VHDL is shown here.

```
U1: CLK_DIV16RSD
port map (
  CLKIN => clk,
  CDRST => clk_div_rst,
  CLKDV => clk_div_by_16 );
```

Verilog Example

Verilog design entry with XST does not require a component declaration; only the component instantiation is necessary. The Verilog syntax to instantiate the CLK_DIV16RSD component is shown here. The input clock signal, *clk*, is assigned to the CLKIN port. The clock divider reset signal, *clk_div_rst*, is assigned to the CDRST port. The clock divider output, *clk_div_by_16*, is assigned to the CLKDV port.

```
CLK_DIV16RSD U1 (
  .CLKIN (clk),
  .CDRST (clk_div_rst),
  .CLKDV (clk_div_by_16) );
```

ABEL Example

Designing with clock dividers in CoolRunner-II requires both the component declaration and component instantiation. The clock divider must be declared as an external component in an ABEL design as shown here.

```
CLK_DIV2R external (CLKIN, CDRST -> CLKDV);
```

Component instantiation in an ABEL design occurs by assigning an identifier to the clock divider component. In this example, U1 is the identier assigned to the clock divider component, CLK_DIV2R, using the functional_block ABEL keyword.

```
U1 functional_block CLK_DIV2R;
```

The following equations illustrate the component port mapping. The input clock, *clk*, is mapped to the CLKIN port. The clock divider reset signal, *clk_div_rst*, is mapped to the CDRST port. The clock divider output signal, *clk_div_by_2*, is assigned to the CLKDV output port.

```
U1.CLKIN = clk;
U1.CDRST = clk_div_rst;
clk_div_by_2 = U1.CLKDV;
```

Note the signal assigned to the CDRST port will automatically be mapped to the CDRST/I/O pin. Note the clock divider is available on CoolRunner-II 128 macrocell devices and larger.

CoolCLOCK

The CoolRunner-II CoolCLOCK feature is the technique of combining the global clock divider and DET registers. Power savings are achieved by dividing the global clock by 2, distributing a lower frequency clock on the internal clock network, and then doubling the clock at each macrocell. Zero clock skew can be acheived due to the zero insertion delay of the clock divider and the DET registers. Figure 3 illustrates the CoolRunner-II CoolCLOCK feature.

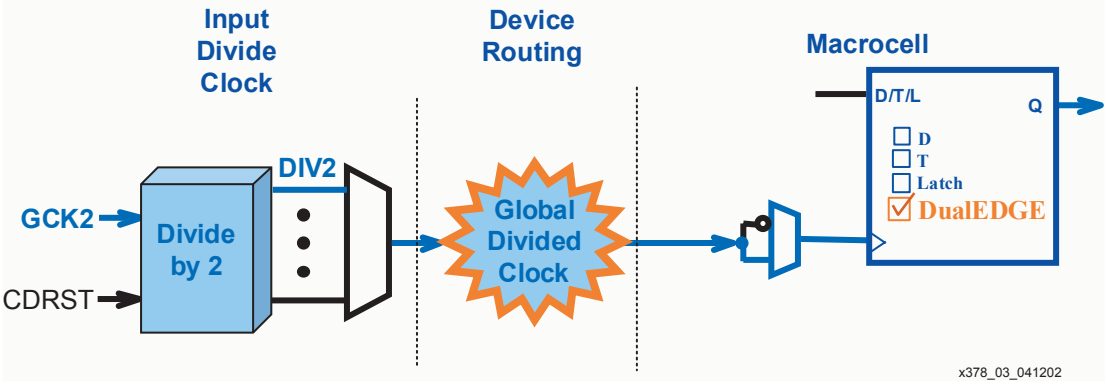

Figure 3: **CoolCLOCK**

Since GCK2 is the only clock network that can be divided, the CoolCLOCK feature is only available on GCK2. The CoolCLOCK feature can be implemented by assigning an attribute to an input clock. The CoolCLOCK attribute replaces the need to instantiate the clock divider and infer DET registers. Table 3 lists the methods available to use the CoolCLOCK attribute.

Table 3: **CoolCLOCK Attribute**

Attribute Format	Syntax	Example
UCF	NET <*clock name*> COOL_CLK;	NET clk COOL_CLK;
ABEL	XILINX PROPERTY 'COOL_CLK <*clock name*>';	XILINX PROPERTY 'COOL_CLK clk';
VHDL	attribute COOL_CLK : string; attribute COOL_CLK of <*clock name*>: signal is "TRUE";	attribute COOL_CLK : string; attribute COOL_CLK of clk : signal is "TRUE";
Verilog	//SYNTHESIS attribute COOL_CLK of <*clock name*>: signal is "TRUE"; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute COOL_CLK of clk : signal is "TRUE";

Note the CoolCLOCK feature is available on CoolRunner-II 128 macrocell devices and larger.

DataGATE

CoolRunner-II designers can block specified inputs under the control of the DataGATE function. By blocking inputs, switching signals do not drive internal chip capacitance and thereby reduce overall power consumption. The last value on the input pin prior to the assertion of the DataGATE rail is latched and used by the CPLD internally. Figure 4 illustrates the DataGATE feature in CoolRunner-II.

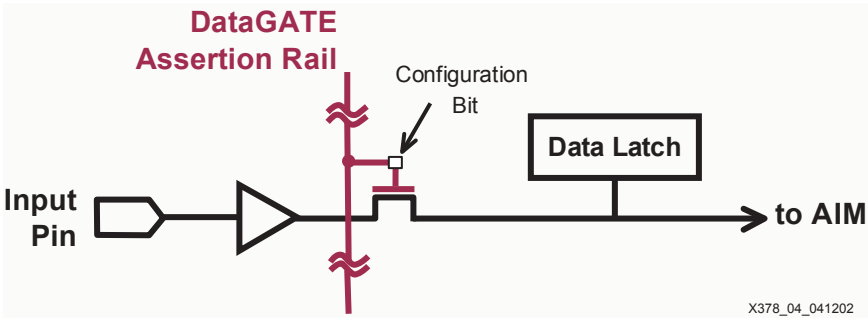

Figure 4: **DataGATE Block Diagram**

There are two attributes associated with the DataGATE feature in CoolRunner-II. The first attribute specifies if an input will be affected by DataGATE and the second designates the DataGATE control signal.

The DataGATE feature is selectable on a per pin basis. Each input pin that uses DataGATE must be assigned a DATA_GATE attribute. Table 4 illustrates the syntax for enabling DataGATE on an input signal.

Table 4: **DataGate Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> DATA_GATE;	NET data_in DATA_GATE;
ABEL	XILINX PROPERTY 'DATA_GATE <*signal name*>';	XILINX PROPERTY 'DATA_GATE data_in';
VHDL	attribute DATA_GATE : STRING; attribute DATA_GATE of <*signal name*>: signal is "TRUE"; Note: The string attribute need only be declared once for all DATA_GATE attributes.	attribute DATA_GATE : STRING; attribute DATA_GATE of data_in : signal is "TRUE";
Verilog	//SYNTHESIS attribute DATA_GATE of <*signal name*>: signal is "TRUE"; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute DATA_GATE of data_in : signal is "TRUE";

The DataGATE assertion rail can be driven from either an I/O pin or internal logic. The DataGATE enable signal is a dedicated DGE/I/O pin for each package in CoolRunner-II. Upon implementation, the software recongnizes a design using DataGATE and automatically assigns this I/O pin to the DataGATE enable control function, DGE. Internally generated DataGATE

contol logic can be assigned to this I/O pin with the BUFG=DATA_GATE attribute. The methods of assigning the DataGATE enable signal are shown in Table 5.

Table 5: **DataGate Control Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> BUFG=DATA_GATE;	NET dg_en BUFG=DATA_GATE;
ABEL	XILINX PROPERTY 'BUFG=DATA_GATE <*signal name*>';	XILINX PROPERTY 'BUFG=DATA_GATE dg_en';
VHDL	attribute BUFG : STRING; attribute BUFG of <*signal name*>: signal is "DATA_GATE"; Note: The string attribute need only be declared once for all BUFG attributes.	attribute BUFG : STRING; attribute BUFG of dg_en : signal is "DATA_GATE";
Verilog	//SYNTHESIS attribute BUFG of <*signal name*>: signal is "DATA_GATE"; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute BUFG of dg_en : signal is "DATA_GATE";

Schmitt Trigger

Each CoolRunner-II I/O has multiple input buffers used for various I/O standard configurations. One of these input buffers behaves as a Schmitt trigger input and is enabled upon CPLD configuration. The Schmitt trigger input allows the board designer the flexibility to utilize the CoolRunner-II with both high speed signals as well as slow switching signals on the same device. Slowly switching signals can cause havoc on digital systems by causing double clocking or glitches on a CMOS input, however this can be virtually eliminated by using the Schmitt trigger input. The Schmitt trigger input use is only at the cost of a few nanosecond delay (refer to the CoolRunner-II datasheet, see **References**, page 11).

Table 6 illustrates the attribute syntax to assign the Schmitt trigger input buffer to a specific signal.

Table 6: **Schmitt Trigger Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> SCHMITT_TRIGGER;	NET data_in SCHMITT_TRIGGER; NET clock SCHMITT_TRIGGER;
ABEL	XILINX PROPERTY 'SCHMITT_TRIGGER <*signal name*>';	XILINX PROPERTY 'SCHMITT_TRIGGER data_in'; XILINX PROPERTY 'SCHMITT_TRIGGER clock';
VHDL	attribute SCHMITT_TRIGGER : STRING; attribute SCHMITT_TRIGGER of <*signal name*>: signal is "TRUE"; Note: The string attribute need only be declared once for all SCHMITT_TRIGGER attributes.	attribute SCHMITT_TRIGGER : STRING; attribute SCHMITT_TRIGGER of data_in: signal is "TRUE"; attribute SCHMITT_TRIGGER of clock: signal is "TRUE";
Verilog	//SYNTHESIS attribute SCHMITT_TRIGGER of <*signal name*>; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute SCHMITT_TRIGGER of data_in; //SYNTHESIS attribute SCHMITT_TRIGGER of clock;

I/O Termination

CoolRunner-II pins may be terminated in the following ways: keeper (also referred to as bushold) and pullup. Usage of the keeper and the pullup circuitry is exclusive on a global basis. When one of these two (keeper and pullup) termination modes is selected for any number of signals, the other termination mode is no longer available to any other signal.

Keeper

The keeper circuitry provides the ability to hold the last known value on an I/O pin using weak pullup/down resistors. If an unterminated I/O pin was in high-impedence and floating, this would cause excessive leakage current. The keeper circuitry eliminates the need for external termination that would resolve this. Table 7 illustrates the attribute syntax for specifying the keeper termination on any I/O pin.

Table 7: **Keeper Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> KEEPER;	NET data_in KEEPER; NET clock KEEPER;
ABEL	XILINX PROPERTY 'KEEPER <*signal name*>';	XILINX PROPERTY 'KEEPER data_in'; XILINX PROPERTY 'KEEPER clock';
VHDL	attribute KEEPER : STRING; attribute KEEPER of <*signal name*>: signal is "TRUE"; Note: The string attribute need only be declared once for all KEEPER attributes.	attribute KEEPER : STRING; attribute KEEPER of data_in: signal is "TRUE"; attribute KEEPER of clock: signal is "TRUE";
Verilog	//SYNTHESIS attribute KEEPER of <*signal name*>; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute KEEPER of data_in; //SYNTHESIS attribute KEEPER of clock;

Pullup

The internal pullup allows the designer to eliminate external pullup resistors on the board, thereby reducing cost and simplifying board layout. Table 8 illustrates the attribute syntax for specifying the pullup I/O termination.

Table 8: **Pullup Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> PULLUP;	NET data_in PULLUP; NET clock PULLUP;
ABEL	XILINX PROPERTY 'PULLUP <*signal name*>';	XILINX PROPERTY 'PULLUP data_in'; XILINX PROPERTY 'PULLUP clock';
VHDL	attribute PULLUP : STRING; attribute PULLUP of <*signal name*>: signal is "TRUE"; Note: The string attribute need only be declared once for all PULLUP attributes.	attribute PULLUP : STRING; attribute PULLUP of data_in: signal is "TRUE"; attribute PULLUP of clock: signal is "TRUE";
Verilog	//SYNTHESIS attribute PULLUP of <*signal name*>; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute PULLUP of data_in; //SYNTHESIS attribute PULLUP of clock;

I/O Configuration

CoolRunner-II devices (128 macrocell and greater) support multiple I/O banks in a single device, allowing for easy interfacing to different voltage standards on one chip. A device can support one I/O standard per bank (i.e. XC2C128 has two banks and can therefore support up to two I/O standards). Regardless of which I/O voltage standard is selected, any pin may be configured as an open-drain output.

I/O Standards

Table 9 lists the supported I/O standards on CoolRunner-II devices. Note that all standards are not supported in every density.

Table 9: **CoolRunner-II Supported I/O Standards**

	XC2C32	XC2C64	XC2C128	XC2C256	XC2C384	XC2C512
I/O Banks	1	1	2	2	4	4
LVTTL	Yes	Yes	Yes	Yes	Yes	Yes
LVCMOS33, LVCMOS25, & LVCMOS18	Yes	Yes	Yes	Yes	Yes	Yes
1.5V I/Os	Yes	Yes	Yes	Yes	Yes	Yes
SSTL2-1 & SSTL3-1	No	No	Yes	Yes	Yes	Yes
HSTL-1	No	No	Yes	Yes	Yes	Yes

Figure 5 illustrates how to specify the default I/O standard for all pins in a design. The I/O standard can be selected in the Implement Design Process Properties window under the Basic Tab.

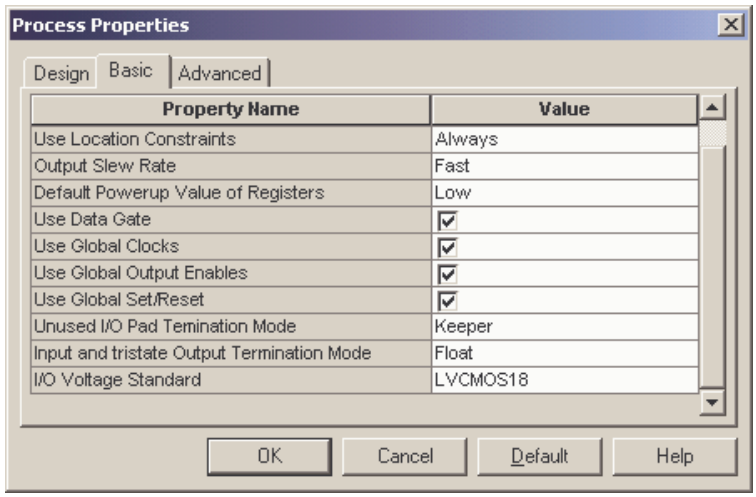

Figure 5: **Global I/O Standard Selection**

If a design requires multiple I/O standards on the same device, each pin must be manually declared with the appropriate I/O standard attribute. Table 10 illustrates the available I/O standard attributes that can be declared.

Table 10: **I/O Standard Attributes**

I/O Standard	Attribute Name
LVTTL	LVTTL
LVCMOS 3.3V	LVCMOS33
LVCMOS 2.5V	LVCMOS25
LVCMOS 1.8V	LVCMOS18
1.5V I/O	LVCMOS15
SSTL 2-1	SSTL2_I
SSTL 3-1	SSTL3_I
HSTL-1	HSTL_I

Table 11 illustrates the syntax for specifying an I/O standard attribute. The examples shown in Table 11 are for specifying the LVCMOS18 I/O standard. For other standards, LVCMOS18 can be replaced with the appropriate attribute name shown in Table 10.

Table 11: **I/O Standard Attribute Syntax**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> <*I/O standard attribute name*>;	NET data_in IOSTANDARD=LVCMOS18; NET clock IOSTANDARD=LVCMOS18;
ABEL	XILINX PROPERTY 'IOSTANDARD=LVCMOS18 <*signal name*>';	XILINX PROPERTY 'IOSTANDARD=LVCMOS18 data_in'; XILINX PROPERTY 'IOSTANDARD=LVCMOS18 clock';
VHDL	attribute IOSTANDARD : STRING; attribute IOSTANDARD of <*signal name*>: signal is "<*I/O standard attribute name*>"; Note: The string attribute need only be declared once for all IOSTANDARD attributes.	attribute IOSTANDARD : STRING; attribute IOSTANDARD of data_in: signal is "LVCMOS18"; attribute IOSTANDARD of clock: signal is "LVCMOS18";
Verilog	//SYNTHESIS attribute IOSTANDARD of <*signal name*> is "<*I/O standard attribute name*>"; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute IOSTANDARD of data_in is "LVCMOS18"; //SYNTHESIS attribute IOSTANDARD of clock is "LVCMOS18";

Open Drain

A signal that is grounded when "false" and is in high-impedence when "true" is considered an open-drain signal. An output on CoolRunner-II can be configured as open drain by simply

declaring the OPEN_DRAIN attribute in the Xilinx software. Table 12 illustrates the syntax for specifing an open drain output with the OPEN_DRAIN attribute.

Table 12: **Open Drain Attribute**

Attribute Format	Syntax	Example
UCF	NET <*signal name*> OPEN_DRAIN;	NET data_out OPEN_DRAIN;
ABEL	XILINX PROPERTY 'OPEN_DRAIN <*signal name*>';	XILINX PROPERTY 'OPEN_DRAIN data_out';
VHDL	attribute OPEN_DRAIN : STRING; attribute OPEN_DRAIN of <*signal name*>: signal is "TRUE"; Note: The string attribute need only be declared once for all OPEN_DRAIN attributes.	attribute OPEN_DRAIN : STRING; attribute OPEN_DRAIN of data_out: signal is "TRUE";
Verilog	//SYNTHESIS attribute OPEN_DRAIN of <*signal name*>; Note: The comment delimiters are intentional and necessary for XST.	//SYNTHESIS attribute OPEN_DRAIN of data_out;

Code Examples Download

Example source code is available for download. Both VHDL and Verilog code with test benches are available for using the CoolRunner-II advanced features.

THE DESIGNS ARE PROVIDED TO YOU "AS IS". XILINX MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. These are only example designs, not fully functional cores. XILINX does not warrant the performance, functionality, or operation of these designs will meet your requirements, or that the operation of the designs will be uninterrupted or error free, or that defects in the designs will be corrected. Furthermore, XILINX does not warrant or make any representations regarding use or the results of the use of the designs in terms of correctness, accuracy, reliability or otherwise.

XAPP378 - http://www.xilinx.com/products/xaw/coolvhdlq.htm

Conclusion

Performance and low power have finally come together with CoolRunner-II CPLDs. The available advanced features furthur reduce power consumption and provide advanced clocking management options. For additional assistance with the CoolRunner-II advanced features, please contact the Xilinx hotline or refer to the Xilinx web support (http://support.xilinx.com/).

References

1. CoolRunner-II family data sheet
2. Xilinx ISE design entry software
3. Application Note: XAPP352: Utilizing a UCF for CoolRunner XPLA3 CPLDs

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/27/02	1.0	Initial Xilinx release.