iCE40 Design Guidelines

Application Note

FPGA-AN-02109-1.0

January 2026

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
BGA	Ball Grid Array
CRC	Cyclic Redundancy Check
ECC	Error Checking and Correction
GPIO	General Purpose Input Output
I/O	Input Output
LED	Light Emitting Diode
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
LVDS	Low Voltage Differential Signaling
NVCM	Non-Volatile Configuration Memory
PCN	Product Change Notifications
PIO	Programmable Input Output
PLB	Programmable Logic Block
QFN	Quad Flat No-Lead
VQFP	Very-Thin Quad Flat Pack
WLCSP	Wafer Level Chip Scale Package

1. Introduction

The iCE40 Design Handbook is a complementary document for iCE40 devices that use the iCEcube2 software tool. This document contains useful links and information related to iCE40 devices and serves as a guideline for best practices to help you achieve your design goals quickly and efficiently.

1.1. Using This Document

This document provides guidance on iCE40 devices and the implementation of the Lattice Radiant™ and Lattice iCEcube2 software design flow. It includes high-level information, design guidelines, design requirements, and design decision trade-offs. The table below provides a description and overview of each section in the document.

Table 1.1. Document Section Description

Section	Description
About the iCE40 Device Family	Provides an overview of the iCE40 device architecture and best practices for design and implementation
Non-Volatile Configuration Memory (NVCM)	Describes the NVCM programming flow for iCE40 devices.
Board Design Reference Point	Lists available development kits and boards for iCE40 devices.
iCE40 Software Support	Provides software installation requirements, supported operating systems, license setup, installer path, and tips for installing iCEcube2 software on Linux.
Reference Designs for iCE40 Devices	Lists available reference designs for iCE40 devices and describes capabilities of each reference design.
Frequent Asked Questions (FAQs)	Provides guidance on accessing FAQs for iCE40 devices and iCEcube2 topics through the Lattice Semiconductor Knowledge Base.
Product Change Notifications (PCNs)	Describe how to locate PCN for iCE40 devices under the *Affected Devices/Packages* column in the PCN page and find information on discontinued iCE40 devices.

2. About the iCE40 Device Family

2.1. Products Overview

The iCE40 device family offers densities ranging from 384 to 7,680 LUTs. The series includes four main active devices: Lattice iCE40 UltraPlus™, Lattice iCE40 Ultra™, Lattice iCE40 UltraLite™, and Lattice iCE40 HX/LP™.

All devices in the series use an advanced 40nm CMOS low-power process, enabling high speed and high energy efficiency in a compact chip. Refer to the table below for summary of each device's features.

Table 2.1. iCE40 Products Overview and I/O Package Support

Features			iCE40 UltraPlus		iCE40 Ultra			iCE40 UltraLite		iCE40 LP						iCE40 HX		
Device			UP3K	UP5K	LP1K	LP2K	LP4K	UL640	UL1K	LP384	LP640	LP1K	LP4K	LP8K	HX1K	HX4K	HX8K
Logic Cells			2800	5280	1100	2048	3520	640	1248	384	640	1280	3520	7680	1280	3520	7680
EBR Memory (Kb)			80	120	64	80	80	56	56	0	64	54	80	128	64	80	128
SPRAM Block (Kb)			1024	1024	–	–	–	–	–	–	–	–	–	–	–	–	–
PLL Block			1	1	1	1	1	1	1	–	–	1	2	2	1	2	2
Hardened I2C			2	2	2	2	2	2	2	–	–	–	–	–	–	–	–
Hardened SPI			2	2	2	2	2	–	–	–	–	–	–	–	–	–	–
Low Power Oscillator			1	1	1	1	1	1	1	–	–	–	–	–	–	–	–
High Frequency Oscillator			1	1	1	1	1	1	1	–	–	–	–	–	–	–	–
DSP Blocks			4	8	2	4	4	–	–	–	–	–	–	–	–	–	–
24mA Driver			3	3	3	3	3	3	3	3	3	3	–	–	–	–	–
100 mA + 400 mA Driver			–	–	–	–	–	1	1	–	–	–	–	–	–	–	–
500 mA Drive			–	–	1	1	1	–	–	–	–	–	–	–	–	–	–
0.35 mm Spacing			Total I/Os (Dedicated I/Os)														
WLCSP	16	1.4 x 1.4 mm	–	–	–	–	–	10	10	–	–	–	–	–	–	–	–
		1.40 x 1.48 mm	–			–	–	–	–	–	10	10	–	–	–	–	–
	25	1.71 x 1.71 mm	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	36	2.08 x 2.08 mm	–	–	26	26	26	–	–	–	–	–	–	–	–	–	–
0.4 mm Spacing			Total I/Os (Dedicated I/Os)														
WLCSP	30	2.15 x 2.55 mm	21	21	–	–	–	–	–	–	–	–	–	–	–	–	–
ucBGA	36	2.50 x 2.50 mm	–	–	–	–	–	26	26	27	–	27	–	–	–	–	–
	49	3 x 3 mm	–	–	–	–	–	–	–	39	–	37	–	–	–	–	–
	81	4 x 4 mm	–	–	–	–	–	–	–	–	–	65	65	65	–	–	–
	121	5 x 5 mm	–	–	–	–	–	–	–	–	–	97	95	95	–	–	–
	225	7 x 7 mm	–	–	–	–	–	–	–	–	–	–	180	180	–	–	–
ucfBGA	36	2.50 x 2.50 mm	–	–	26	26	26	–	–	–	–	–	–	–	–	–	–
0.5 mm Spacing			Total I/Os (Dedicated I/Os)														
QFN	32	5 x 5 mm	–	–	–	–	–	–	–	23	–	–	–	–	–	–	–
	48	7 x 7 mm	–	39	39	39	39	–	–	–	–	–	–	–	–	–	39
	84	7 x 7 mm	–	–	–	–	–	–	–	–	–	69	–	–	–	–	–
csBGA	81	5 x 5 mm	–	–	–	–	–	–	–	–	–	64	–	–	–	–	–

	121	6 x 6 mm	–	–	–	–	–	–	–	–	–	94	–	–	–	–	–
	132	8 x 8 mm	–	–	–	–	–	–	–	–	–	–	–	–	97	97	97
VQFP	100	14 x 14 mm	–	–	–	–	–	–	–	–	–	–	–	–	74	–	–
TQFP	144	20 x 20 mm	–	–	–	–	–	–	–	–	–	–	–	–	98	109	–
0.8 mm Spacing			**Total I/Os (Dedicated I/Os)**														
caBGA	121	9 x 9 mm	–	–	–	–	–	–	–	–	–	–	–	–	–	95	95
	256	14 x 14 mm	–	–	–	–	–	–	–	–	–	–	–	–	–	–	208

Notes:

- No PLL hard block is available on the 16 WLCSP, 36 ucBGA, 81 csBGA, 84 QFN, and 100 VQFP packages for LP1K
- Only one PLL hard block is available on the 81 ucBGA package for LP4K and LP8K
- PLL hard block is only available on the 36 ucBGA package for UL640
- The 24 mA driver are RGB open drain LED outputs which has configurable sink up current of up to 24 mA
- 100 mA and 400mA drivers are barcode emulator driver and IR LED output accordingly. iCE40 UltraLite can utilize the 100 mA barcode driver to form a 500 mA IR driver. Note that these driver have open drain pins.
- 500 mA driver is high current IR LED driver which has configurable sink up current of up to 500 mA. Note that this driver has open drain pins

For each device package, iCE40 devices support both single-ended programmable I/Os and differential I/Os, which can be utilized as needed. Specific single-ended I/Os are paired or combined to emulate the behavior of a differential I/O. The single-ended programmable I/O that uses PAD A is the positive polarity, while the one that uses PAD B is the negative polarity within the differential pair. In the table above, you can see how many programmable I/Os can be used for each device.

2.2. iCE40 Family Architecture and Performance

2.2.1. iCE40 sysI/O Buffer

The iCE40 devices support I/O standards at various VCCIO levels, from 3.3 V (all iCE40 devices) down to 1.8 V (iCE40 UltraPlus can reach 1.2 V for outputs).

Supported standards include LVCMOS (inputs and outputs) and LVDSE (input only), with performance targets of 250 MHz for LVCMOS33/25, 155 MHz for LVCMOS18, 70 MHz for LVCMOS12, and 250 MHz for LVDSE.

Note that iCE40 devices do not support mixed voltages for these standards. For example, selecting LVCMOS33 requires a 3.3 V bank voltage. In this case, you cannot assign both LVCMOS25 and LVCMOS33 standards to the same bank.

The single-ended (LVCMOS) standard supports configurable pull-up resistors for input signals of 100 kΩ, 10 kΩ, 6.8 kΩ, and 3.3 kΩ. Similarly, the I/O drive strength can be configured to increase the I/O output current.

In general, all iCE40 devices support an internal pull-up resistor, and the pull-up resistance varies depending on the bank voltage. However, this configurable feature is supported only on specific iCE40 devices.

The pull-up resistor is configurable on iCE40 UltraPlus and iCE40 UltraLite devices, while drive strength modification is supported on iCE40 LP/HX devices.

Table 2.2. iCE40 Products Drive Strength Capability

Device	Drive Strength Support			
	3.3 V	2.5 V	1.8 V	1.2 V
iCE40 LP/HX	8, 16, 24	8, 16, 24	8, 16, 24	–
iCE40 Ultra/UltraLite	8	6	4	–
iCE40 UltraPlus	8	6	4	2

Table 2.3. iCE40 Products Internal Pull Up

	Internal Pull Up Support			
	3.3 V	2.5 V	1.8 V	1.2 V
Min	~ 25.7k	~ 34.7k	~ 58k	–
Max	~ 300k	~ 312.5k	~ 600k	–

Note: Internal pull-up support for the I/O blocks is calculated based on VCCIO bank voltage or internal Programmable Input Output (PIO) pull-up current. The internal PIO pull-up current is described in the DC Electrical Characteristics section of each iCE40 device family data sheet. Refer to the References section for the data sheets.

The differential (LVDSE) standard is supported only for input signals. It consists of two single-ended ports (Pad A and Pad B) combined to emulate a differential pair, which limits performance to 250 MHz.

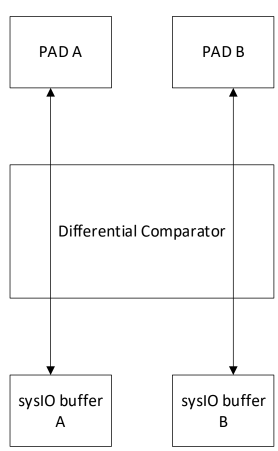


Figure 2.1. iCE40 sysI/O

The features described above can be configured by understanding the I/O architecture and how it is implemented in each software tool (Lattice Radiant or iCEcube2). The following subsections elaborate on the I/O architecture, with a primary focus on implementation in iCEcube2.

2.2.1.1. iCE40 I/O Buffer (SB_IO)

To implement various sysI/O parameters for iCEcube2 supported devices (iCE40 LP/HX, iCE40 Ultra/UltraLite, iCE40 UltraPlus), you may instantiate the SB_IO block or primitive in your design. The architecture and signals or parameters of the block are described below.

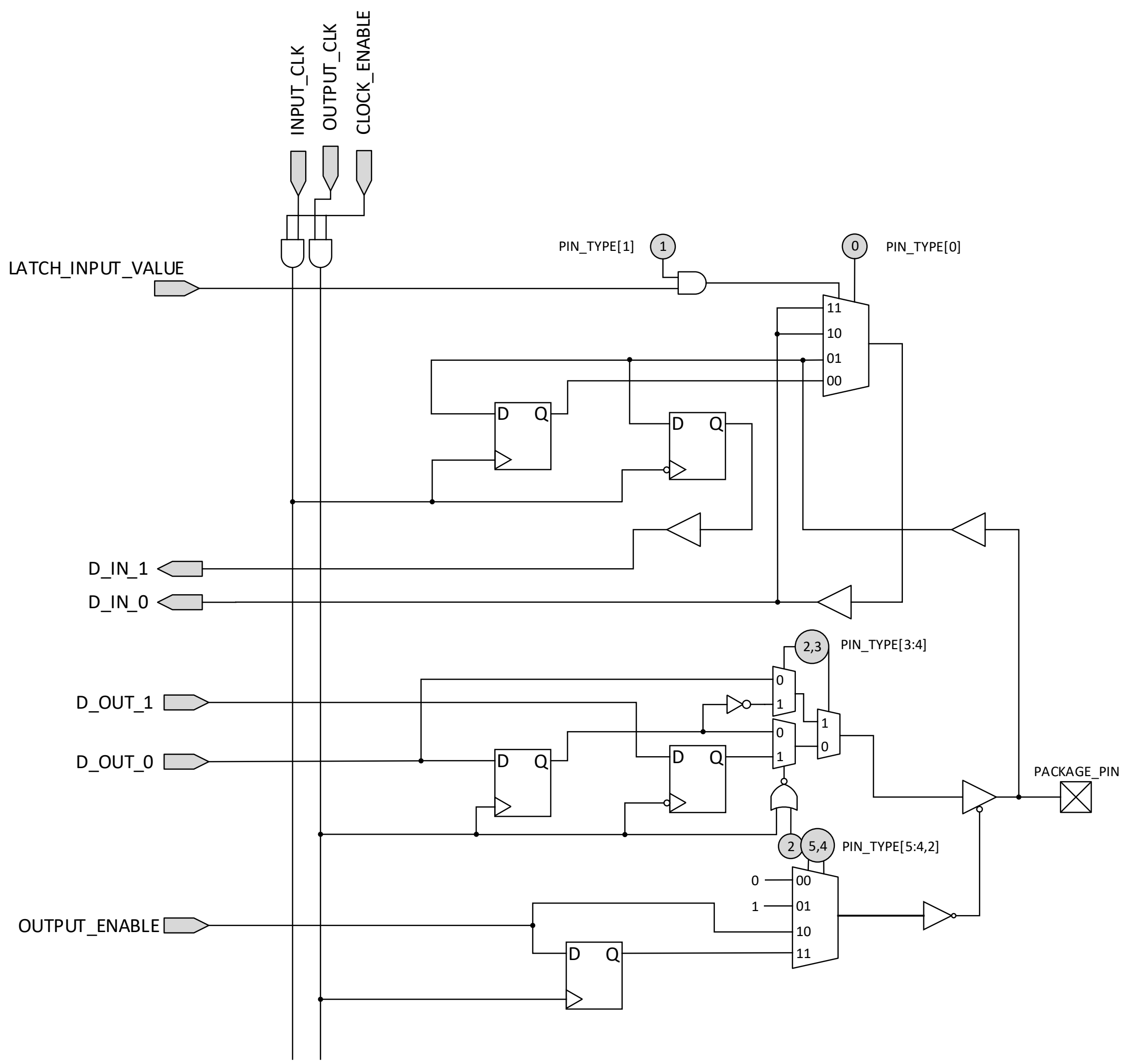


Figure 2.2. iCE40 SB_IO Buffer Structure

Table 2.4. iCE40 SB_IO Block Signal Description

Signal Name	Description
LATCH_INPUT_VALUE	• Control signal used to latch or hold the previous data from the input data pin • This signal is useful for capturing or storing data at a specific time by locking it to a defined state (1 or 0) and at the same time reduces power • Active-high signal
D_IN_0, D_IN_1	• These data signals are captured from the package pin • *D_IN_0* is captured on the rising edge of the clock, and the *D_IN_1* on the falling edge • Use these signals when the I/O buffer is configured as an input or bidirectional block

	• If you are not using a Double Data Rate (DDR) interface or the PIN [1:0] is not equivalent to 00: • By default, it is configured as Single Data Rate (SDR) interface where only *D_IN_0* is used as the data signal • Signal connected to these signals must be wire or net
D_OUT_0, D_OUT_1	• These data signals are sent to the package pin • *D_OUT_0* is sent on the rising edge of the clock, and *D_OUT_1* on the falling edge • Use these signals when the I/O buffer is configured as an input or bidirectional block • If you are not using a DDR interface or the PIN [5:2] is not equivalent to 0100, 1000, or 1100: • By default, it is configured as SDR interface where *D_OUT_0* is used only as the data signal • Signal connected to these signals should be wire or net
OUTPUT_ENABLE	• This signal controls data output from the I/O block to the package pin • Active-high signal. When signal is asserted, the output on the package pin is in tri-state
PACKAGE_PIN	This signal connects to the physical pad of the device.
CLOCK_ENABLE	• Enables the clock signal to pass through the AND gate of the I/O buffer when the I/O block is registered • Setting this signal to low causes the registers to latch or hold the current data
INPUT_CLOCK OUTPUT_CLOCK	Clock pins when input or output are used with the internal registers

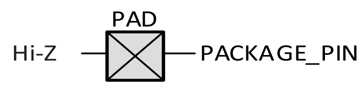
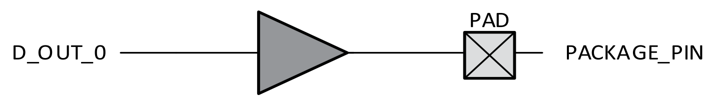
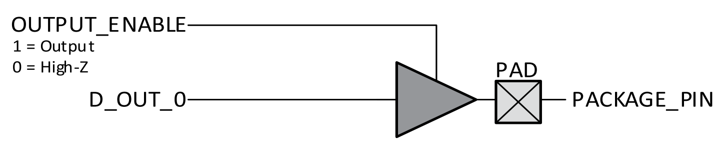
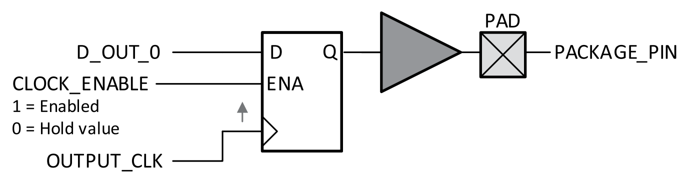
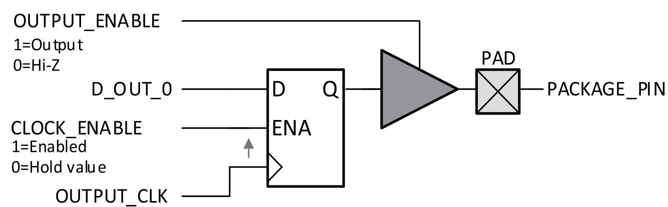
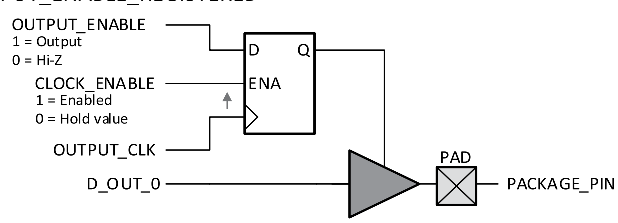
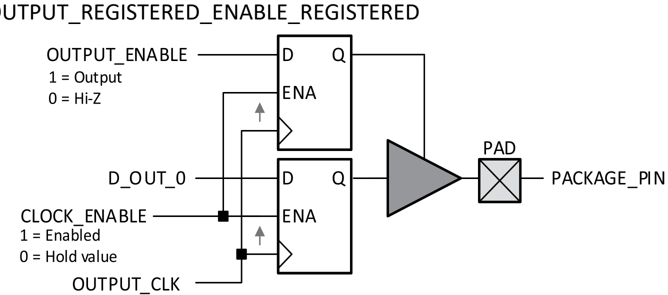
Note: If the input signals above are left unconnected, the iCEcube2 tool assigns these signals to logic *0* except for *CLOCK_ENABLE*.

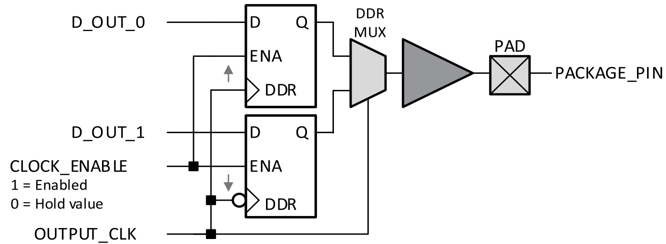
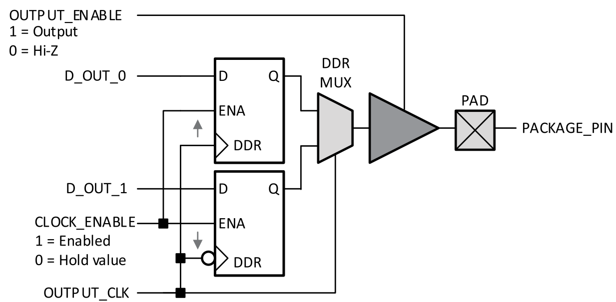
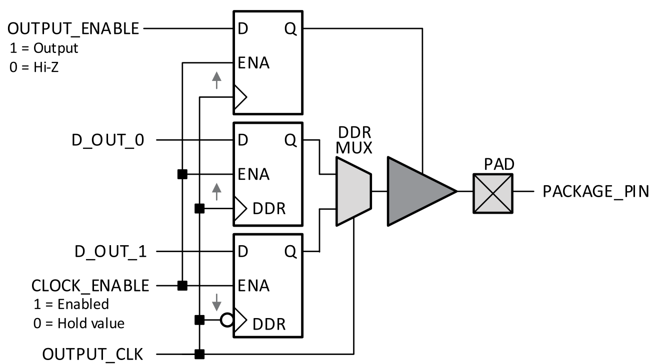
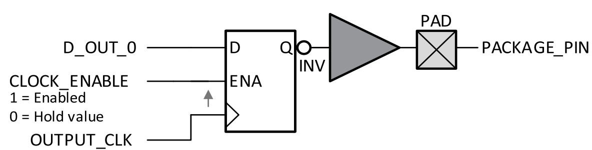
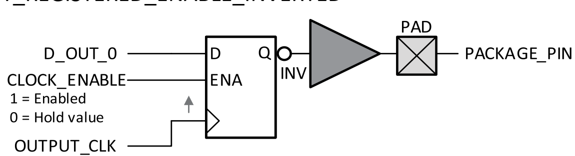
Table 2.5. iCE40 I/O Block Parameter Description (SB_IO)

Parameter Name	Description
PIN_TYPE	• Defines the behavior or function of the I/O block either as an input, output or others • This parameter consists of 6 bits: • [1:0] controls the input function • [5:2] controls the output function • Refer to Table 2.6 and Table 2.7 for full parameter list
PULLUP	Allows you to set the pull-up resistor on the I/O pin.
NEG_TRIGGER	• Defines the polarity of the registers or flip-flops inside the I/O block • When set to 1, data is triggered on the falling edge of the clock. The default value is 0, which indicates a rising edge
IO_STANDARD	Indicates if the I/O standard is SB_LVCMOS or SB_LVDS_INPUT.

The SB_IO block PIN_TYPE can be modified using the values shown in Table 2.6 and Table 2.7. Each PIN_TYPE defines the circuitry for the data flow and control.

Table 2.6. Output Structures of SB_IO block PIN_TYPE Values

Style	Mnemonic (Diagram)	PIN_TYPE [5:2]			
None (output disabled)	PIN_NO_OUTPUT 	0	0	0	0
Non-registered Output	PIN_OUTPUT 	0	1	1	0
	PIN_OUTPUT_TRISTATE 	1	0	1	0
Registered Outputs	PIN_OUTPUT_REGISTERED 	0	1	0	1
	PIN_OUTPUT_REGISTERED_ENABLE 	1	0	0	1
	PIN_OUTPUT_ENABLE_REGISTERED 	1	1	1	0
	PIN_OUTPUT_REGISTERED_ENABLE_REGISTERED 	1	1	0	1
DDR Output	PIN_OUTPUT_DDR	0	1	0	0

<table>
<tr><th>Style</th><th>Mnemonic (Diagram)</th><th colspan="4">PIN_TYPE [5:2]</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>PIN_OUTPUT_DDR_ENABLE
</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td></td><td>PIN_OUTPUT_DDR_ENABLE_REGISTERED
</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>Registered Output, Inverted</td><td>PIN_OUTPUT_REGISTERED_INVERTED
</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>PIN_OUTPUT_REGISTERED_ENABLE_INVERTED
</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td></td><td>PIN_OUTPUT_REGISTERED_ENABLE_REGISTERED_INVERTED</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

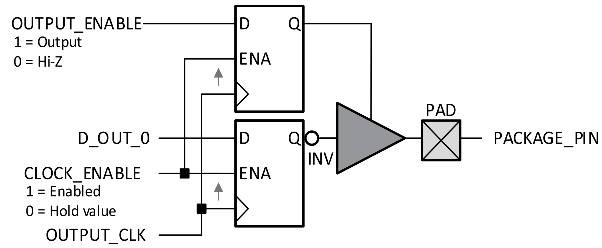
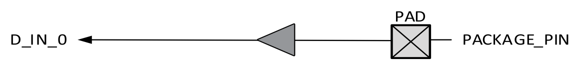
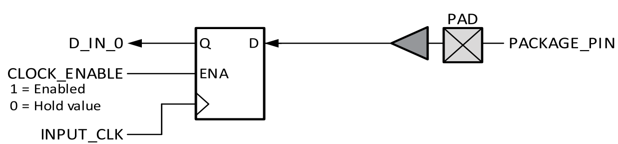
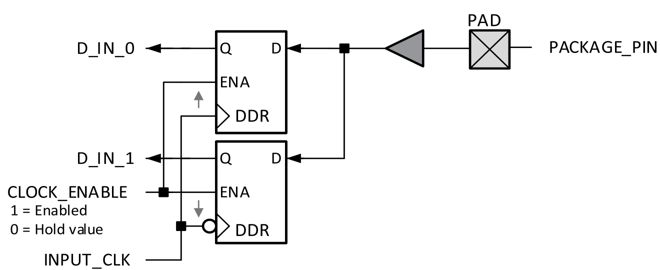
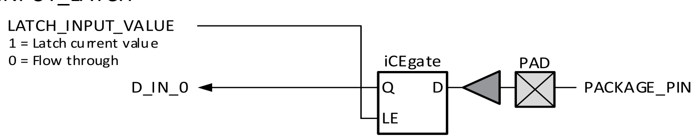
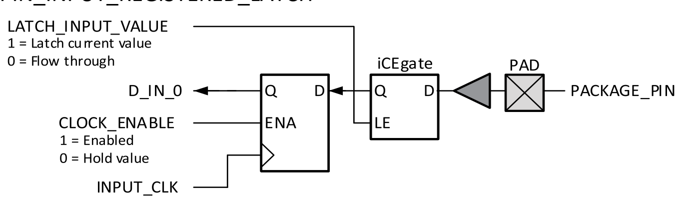
Style	Mnemonic (Diagram)	PIN_TYPE [5:2]			
					

Table 2.7. Input Structures and PIN_TYPE Values (SB_IO)

Style	Mnemonic (Diagram)	PIN_TYPE [1:0]	
Direct	PIN_INPUT 	0	1
Registered	PIN_INPUT_REGISTERED 	0	0
DDR Output	PIN_INPUT_DDR 	0	0
iCEgate Low-Power Latch	PIN_INPUT_LATCH 	1	1
	PIN_INPUT_REGISTERED_LATCH 	1	0

Use Case 1: When using the SB_IO block for a single-ended I/O signal, the block is automatically inferred during design synthesis. You do not need to instantiate it unless the I/O blocks are used beyond their default behavior. By default, the SB_IO blocks are inferred as generic SDR I/O.

Figure 2.3 shows an example of how the SB_IO block and its parameters are used in a design, illustrating data flow within the I/O block. In this example, two SB_IO blocks are used: one as an input (*i_test1*) and one as an output (*o_test_0*). The PIN_TYPE values are defined as 000001 (no output, simple input) and 011000 (simple output, input DDR clocked), respectively.

```
module top (clk, sw, din, dout);

input clk, sw;
input  din;
output  dout;

wire clk_pll;
wire rst;
reg  datareg;
wire  dinw;
wire  doutw;

SB_IO i_test1
(.PACKAGE_PIN (din), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (), .D_OUT_0 (),
.D_OUT_1 (), .D_IN_0 (dinw), .D_IN_1 () );
defparam i_test1.PIN_TYPE = 6'b000001;
defparam i_test1.PULLUP = 1'b1;
defparam i_test1.IO_STANDARD = "SB_LVCMOS";
defparam i_test1.NEG_TRIGGER = 1'b0;

iCE40HandbookTest_pll iCE40HandbookTest_pll_inst(.REFERENCECLK(clk), .PLLOUTCORE(clk_pll), .PLLOUTGLOBAL(), .RESET(sw), .LOCK(rst));

always @(posedge clk_pll)
    begin
        if(rst == 1'b0)
            begin
                datareg <= 2'b0;
            end
        else
            begin
                datareg <= dinw;
            end
    end


assign doutw = datareg;


SB_IO o_test0 (.PACKAGE_PIN (dout), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (),
.D_OUT_0 (doutw), .D_OUT_1 (), .D_IN_0 (), .D_IN_1 () ) ;
defparam o_test0.PIN_TYPE = 6'b011000;
defparam o_test0.PULLUP = 1'b1;
defparam o_test0.IO_STANDARD = "SB_LVCMOS";
defparam o_test0.NEG_TRIGGER = 1'b0;

endmodule
```

Figure 2.3. Sample Verilog Implementation of SB_IO

Figure 2.4 shows the corresponding circuitry and data flow for these PIN_TYPE definitions. The PIN_TYPE parameter determines which parts of the circuitry in the I/O block are active.

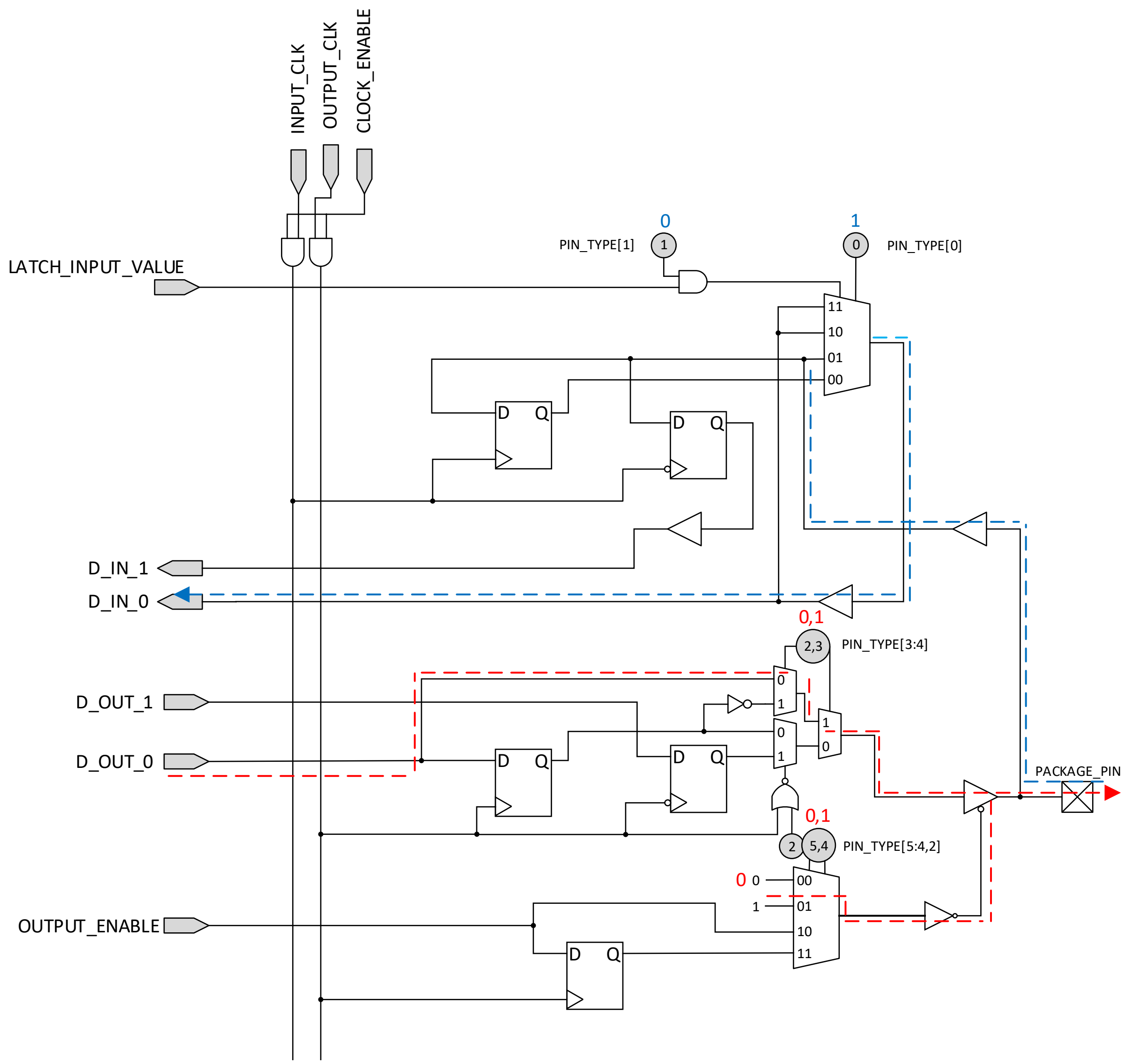


Figure 2.4. Data and Circuit Flow with Defined PIN_TYPE

Use Case 2: When using the SB_IO block for differential I/O signals. Differential inputs are supported for all iCE40 devices. To use differential inputs, you must instantiate the SB_IO primitive and define the IO_STANDARD to the SB_LVDS_INPUT. Defining this IO_STANDARD automatically reserves and uses the differential pair of the selected input signal. The reserved pin is always adjacent to the defined input port.

Figure 2.5 shows an example of how the SB_IO block is used as a differential I/O buffer. In this example, three SB_IO blocks are instantiated in the design. The input and output PIN_TYPE values are the same as those used in Use Case 1. The difference lies in the IO_STANDARD setting. Therefore, the data flow and circuitry for this use case remain the same as shown in Figure 2.4.

```
module top (clk, sw, din, doutp,doutn);

input clk, sw;
input  din;
output  doutp,doutn;

wire clk_pll;
wire rst;
reg  datareg;
wire  dinw;
wire  doutw;

SB_IO i_test1
(.PACKAGE_PIN (din), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (), .D_OUT_0 (),
.D_OUT_1 (), .D_IN_0 (dinw), .D_IN_1 () );
defparam i_test1.PIN_TYPE = 6'b000001;
defparam i_test1.PULLUP = 1'b1;
defparam i_test1 IO_STANDARD = "SB_LVDS_INPUT";
defparam i_test1.NEG_TRIGGER = 1'b0;

iCE40HandbookTest_pll iCE40HandbookTest_pll_inst(.REFERENCECLK(clk), .PLLOUTCORE(clk_pll), .PLLOUTGLOBAL(), .RESET(sw), .LOCK(rst));

always @(posedge clk_pll)
    begin
        if(rst == 1'b0)
            begin
                datareg <= 2'b0;
            end
        else
            begin
                datareg <= dinw;
            end
    end

assign doutw = datareg;

SB_IO op_test0 (.PACKAGE_PIN (doutp), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (),
.D_OUT_0 (doutw), .D_OUT_1 (), .D_IN_0 (), .D_IN_1 () ) ;
defparam o_test0.PIN_TYPE = 6'b011000;
defparam o_test0.PULLUP = 1'b1;
defparam o_test0.IO_STANDARD = "SB_LVCMOS";

SB_IO on_test0 (.PACKAGE_PIN (doutn), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (),
.D_OUT_0 (~doutw), .D_OUT_1 (), .D_IN_0 (), .D_IN_1 () ) ;
defparam o_test0.PIN_TYPE = 6'b011000;
defparam o_test0.PULLUP = 1'b1;
defparam o_test0.IO_STANDARD = "SB_LVCMOS";


endmodule
```

Figure 2.5. Sample Verilog Implementation with SB_IO using LVDS Inputs

For LVDS input, you only need to select the positive polarity port. In this case, based on the Pin Constraint Editor shown in Figure 2.6, setting pin 42 for *din* automatically assigns pin 38 as the negative polarity pin for the LVDS input, as shown in Figure 2.7. This occurs because pin 38 is the differential pair of pin 42 and represents the negative polarity.

	Locked	Object List	Type	Pin Location	Bank	IO Standard	Pull Up	PullUp Resistor	Weak PullUp	Load Cap
1	☐	sw	Input	28	Top					
2	☐	doutp	Output	36	Top	SB_LVCMOS	No			
3	☐	doutn	Output	37	Top	SB_LVCMOS	No			
4	☐	din	Input	42	Top	SB_LVDS_INPUT	Yes	3P3K		
5	☐	clk	Input	32	Top					

Figure 2.6. PIN Constraint Editor for iCEcube2

```
Output | iCE40HandbookTest_pll_inst.v | iCE40HandbookTest_pll.v | top.v | Package View | Pin Constraints Editor | top_sbt.rpt | iCE40HandbookTest.scf | placer.log | top_timing.rpt
* ******************************************************************************

Device, iCE40UP5K
Package, SG48

Pin Number, Pin Type Name, Signal Name, Direction, Global Buffer, IO Standard, Pull Up, Pullup resistor, Weak Pull Up, IO Function, IO Bank, Drive Strength
--------------, --------------, --------------, --------------, --------------, --------------, --------------, --------------, --------------, --------------, --------------, --------------,
1, VCCIO_2, not used, , , , , , bottomBank,
2, PIO, not used, , , , , , bottomBank,
3, PIO, not used, , , , , , bottomBank,
4, PIO, not used, , , , , , bottomBank,
5, VCC, not used, , , , , , UnknownBank,
6, PIO, not used, , , , , , bottomBank,
7, CDONE, not used, , , , , , bottomBank,
8, CRESET_B, not used, , , , , , bottomBank,
9, PIO, not used, , , , , , bottomBank,
10, PIO, not used, , , , , , bottomBank,
11, PIO, not used, , , , , , bottomBank,
12, PIO, not used, , , , , , bottomBank,
13, PIO, not used, , , , , , bottomBank,
14, SPI_SO, not used, , , , , , bottomBank,
15, SPI_SCK, not used, , , , , , bottomBank,
16, SPI_SS_B, not used, , , , , , bottomBank,
17, SPI_SI, not used, , , , , , bottomBank,
18, PIO, not used, , , , , , bottomBank,
19, PIO, not used, , , , , , bottomBank,
20, PIO_GBIN, not used, , , , , , bottomBank,
21, PIO, not used, , , , , , bottomBank,
22, VDDIO_SPI, not used, , , , , , bottomBank,
23, PIO_I3C, not used, , , , , , topBank,
24, VPP, not used, , , , , , topBank,
25, PIO_I3C, not used, , , , , , topBank,
26, PIO, not used, , , , , , topBank,
27, PIO, not used, , , , , , topBank,
28, PIO, sw, Input, , SB_LVCMOS, No Pull Up, , , Simple Input, topBank, x1
29, AVDD_TOP, not used, , , , , , UnknownBank,
30, VCC, not used, , , , , , UnknownBank,
31, PIO, not used, , , , , , topBank,
32, PIO, clk, Input, , SB_LVCMOS, No Pull Up, , , Simple Input, topBank, x1
33, VCCIO_0, not used, , , , , , topBank,
34, PIO, not used, , , , , , topBank,
35, PIO_GBIN, not used, , , , , , topBank,
36, PIO, doutp, Output, , SB_LVCMOS, No Pull Up, , , No Output, topBank, x1
37, PIO_GBIN, doutn, Output, , SB_LVCMOS, No Pull Up, , , No Output, topBank, x1
38, PIO, din (negative), Input, , SB_LVDS_INPUT, Pull Up, 3P3K, , Simple Input, topBank,
39, PIO_RGB, not used, , , , , , topBank,
40, PIO_RGB, not used, , , , , , topBank,
41, PIO_RGB, not used, , , , , , topBank,
42, PIO, din, Input, , SB_LVDS_INPUT, Pull Up, 3P3K, , Simple Input, topBank, x1
43, PIO, not used, , , , , , topBank,
44, PIO_GBIN, not used, , , , , , bottomBank,
45, PIO, not used, , , , , , bottomBank,
46, PIO, not used, , , , , , bottomBank,
47, PIO, not used, , , , , , bottomBank,
48, PIO, not used, , , , , , bottomBank,
```

Figure 2.7. I/O Report for iCECube2

Unlike LVDS input, LVDS output cannot automatically dedicate the pins of the LVDS signal. This is because, architecture-wise, iCE40 devices implement only emulated LVDS output. In this case, you must instantiate two SB_IO blocks: one for the positive polarity signal and one for the negative polarity signal. For this implementation, you manually create an LVDS output signal with two single-ended logic paths, where the signal connected to the D_OUT pin of one SB_IO block is manually negated.

Note that differential resistors and termination are implemented externally when using LVDS with iCE40 devices.

In iCEcube2, you can enable or disable automatic insertion of the I/O buffers in your design. If you do not use buffer primitives directly in your RTL design, you must enable *IO Insertion* during synthesis. This feature adds the I/O primitives automatically. This is enabled through strategy settings under **Tools** > **Tools Option** as shown in Figure 2.9. Without the I/O Buffers, synthesis tool will generate an error as shown in Figure 2.8.

```
Process took 0h:00m:01s realtime, 0h:00m:01s cputime
# Tue Oct 22 09:58:55 2024
###########################################################]
Synthesis exit by 2.
Synthesis failed.
Synthesis batch mode runtime 3 seconds
```

Figure 2.8. Error with Disabled IO Insertion

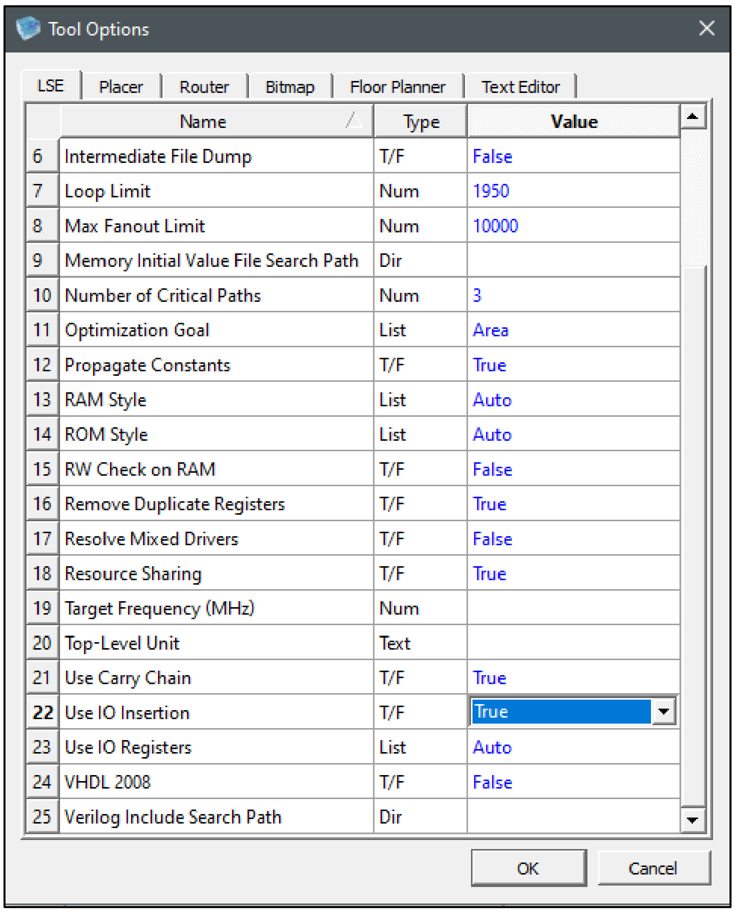


Figure 2.9. Enabled IO Insertion for LSE

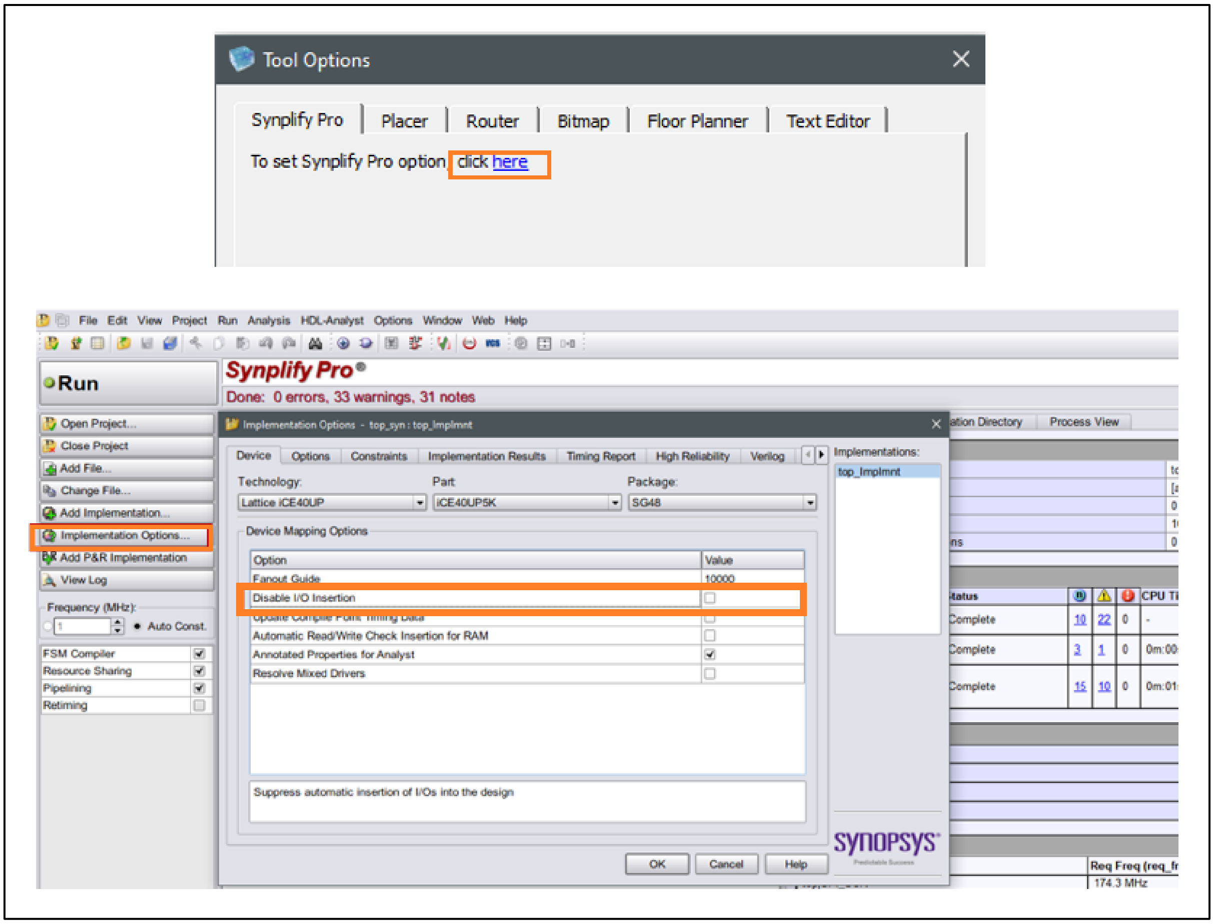


Figure 2.10. Enable or Disable I/O Insertion for Synplify Pro

2.2.1.2. iCE40 Open Drain Buffer (SB_IO_OD)

iCE40 devices such as UltraPlus, Ultra, and UltraLite can use their LED drivers to drive multi-color LEDs, transmit and receive data via infrared LEDs, function as a barcode driver, and more. These LED pins are open-drain. In some cases, these open-drain pins are used as GPIOs instead of LED drivers.

A common mistake when using these pins in iCEcube2 is directly connecting signals from your design to open-drain pins. This is not allowed in iCEcube2. If you do this, you may encounter the error shown below.

```
Error during constrained IO placement
I2723: placment information file is dumped at :
C:/Users/[redacted]/Downloads/carry/top/top_Implmnt\sbt\outputs\placer\top.pcf
I2709: Tool unable to complete IOPlacement for the design
E2792: Instance led_obuf_0 incorrectly constrained at SB_IO_OD location
E2055: Error while doing placement of the design
```

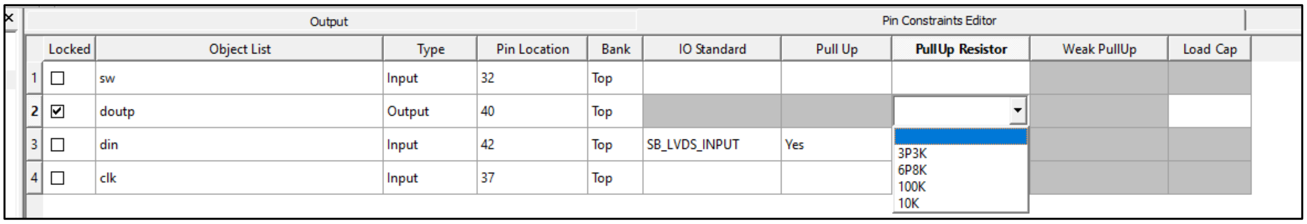
Figure 2.11. Error when Utilizing Open Drain Pins with No SB_IO_OD

The keyword associated with this error is **SB_IO_OD**. In addition to regular I/O buffers such as SB_IO, SB_IO_OD must be used when an open-drain pin is configured as a GPIO pin instead of an LED driver. You also cannot use SB_IO_OD on a regular GPIO. Doing so results in a placement error, as shown in the figure below.

```
SB_IO_OD placed at NON SB_IO_OD Location
Packing failed due to placement violation!
```

Figure 2.12. Error when Utilizing SB_IO_OD On Regular GPIO

Similarly, SB_IO_OD has the same I/O buffer structure as SB_IO. The difference lies in the supported parameters. SB_IO_OD supports only PIN_TYPE and NEG_TRIGGER. Note that the open-drain pins on iCE40 devices do not include internal pull-up resistors, even though the tool may allow you to select one. In these cases, use an external pull-up resistor, as shown in Figure 2.13.



	Output					Pin Constraints Editor				
	Locked	Object List	Type	Pin Location	Bank	IO Standard	Pull Up	PullUp Resistor	Weak PullUp	Load Cap
1	☐	sw	Input	32	Top					
2	☑	doutp	Output	40	Top					
3	☐	din	Input	42	Top	SB_LVDS_INPUT	Yes			
4	☐	clk	Input	37	Top					

Figure 2.13. Setting Internal Pull Up Resistor on Pin Constraints Editor

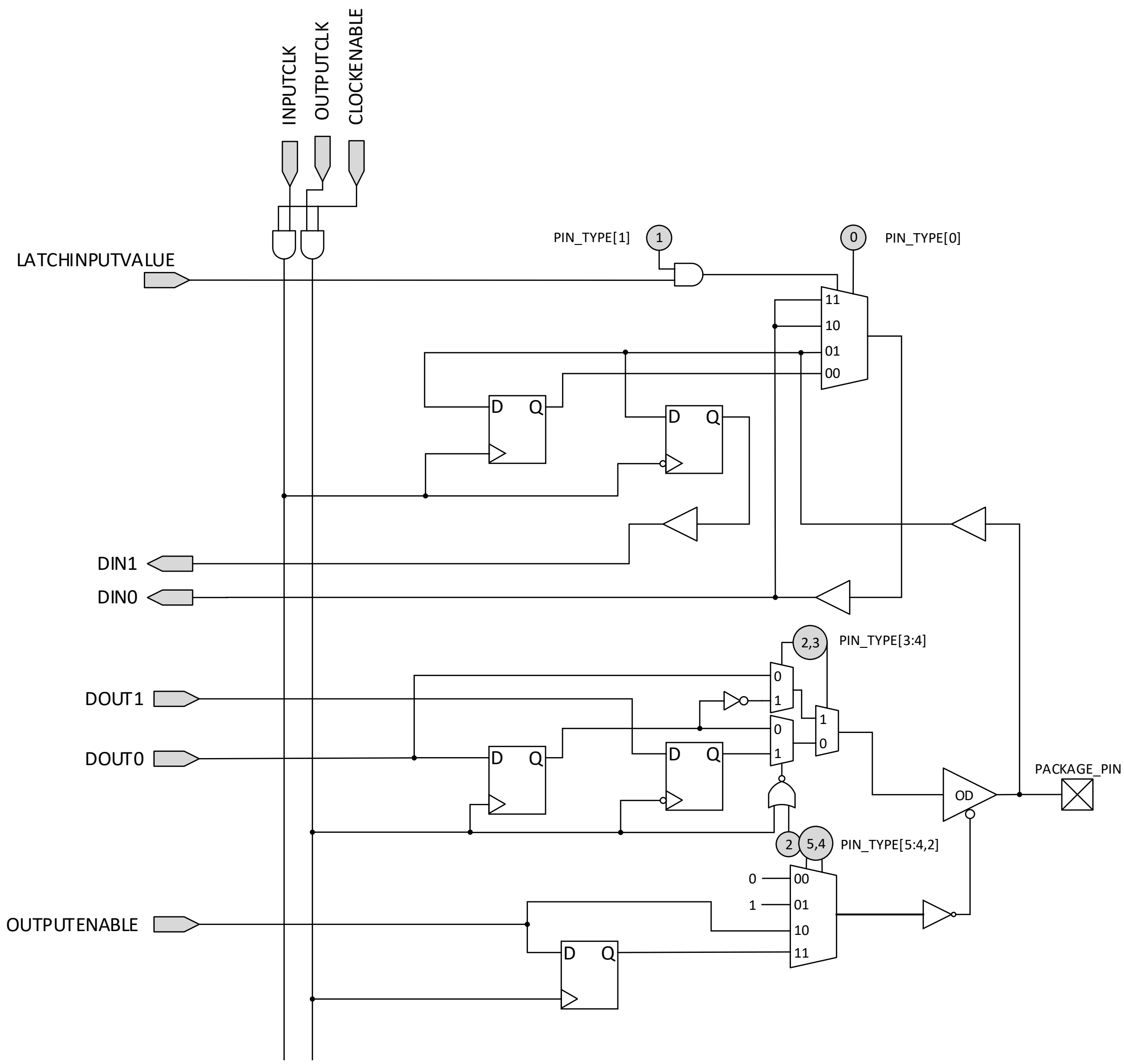


Figure 2.14. iCE40 I/O Open Drain Buffer Structure

Table 2.8. iCE40 SB_IO_OD Block Signal Description

<table>
<tr><th>Signal Name</th><th>Description</th></tr>
<tr><td>LATCHINPUTVALUE</td><td>Control signal used to latch or hold the previous data from the input data pin
This signal is useful for capturing or storing data at a specific time by locking it to a defined state (1 or 0) and at the same time reduces power
Active-high signal</td></tr>
<tr><td>DIN0,
DIN1</td><td>These data signals are captured from the package pin
DIN0 is captured on the rising edge of the clock, and the DIN1 on the falling edge
Use these signals when the I/O buffer is configured as an input or bidirectional block
If you are not using a Double Data Rate (DDR) interface or the PIN [1:0] is not equivalent to 00:
By default, it is configured as Single Data Rate (SDR) interface where only DIN0 is used as the data signal
Signal connected to these signals must be wire or net</td></tr>
<tr><td>DOUT0,
DOUT1</td><td>These data signals are sent to the package pin
DOUT0 is sent on the rising edge of the clock, and DOUT1 on the falling edge
Use these signals when the I/O buffer is configured as an input or bidirectional block
If you are not using a DDR interface or the PIN [5:2] is not equivalent to 0100, 1000, or 1100:
By default, it is configured as SDR interface where DOUT0 is used only as the data signal
Signal connected to these signals should be wire or net</td></tr>
<tr><td>OUTPUTENABLE</td><td>This signal controls data output from the I/O block to the package pin
Active-high signal. When signal is asserted, the output on the package pin is in tri-state</td></tr>
<tr><td>PACKAGEPIN</td><td>This signal connects to the physical pad of the device.</td></tr>
<tr><td>CLOCKENABLE</td><td>Enables the clock signal to pass through the AND gate of the I/O buffer when the I/O block is registered
Setting this signal to low causes the registers to latch or hold the current data</td></tr>
<tr><td>INPUTCLOCK,
OUTPUTCLOCK</td><td>Clock pins when input or output are used with the internal registers.</td></tr>
</table>

Table 2.9. iCE40 IO Block Parameter Description

<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>PIN_TYPE</td><td>Defines the behavior or function of the I/O block either as an input, output or others
This parameter consists of 6 bits:
[1:0] controls the input function
[5:2] controls the output function
Refer to Table 2.7 and Table 2.8 for full parameter list</td></tr>
<tr><td>NEG_TRIGGER</td><td>Defines the polarity of the registers or flip-flops inside the I/O block
When set to 1, data is triggered on the falling edge of the clock. The default value is 0, which indicates a rising edge.</td></tr>
</table>

Table 2.10. iCE40 SB_IO_OD Block Signal Description

Device	UWG30	SG48
iCE40UP3K	C5, B5, A5	–
iCE40UP5K	C5, B5, A5	41, 40, 39

Table 2.11. iCE40 SB_IO_OD Block Signal Description

Device	WLCP16	WLCS36	CM36A	SG48
iCE5LP1K	–			
iCE5LP2K	–	A6, B6, C6, A2	A1, B1, A2, A6	39, 40, 41
iCE5LP4K	–			
iCE40UL640	C4, A4, B4, A1, A2	–	A1, B1, A2, A6, A5	–
iCE40UL1K		–		–

2.2.1.3. iCE40 Global Buffer (SB_GB_IO, SB_GB)

Two global buffer resources are available for iCE40 devices: SB_GB_IO and SB_GB. These buffers allow signals connected to these resources to use the primary routing of the device architecture, enabling dedicated primary routing for easier access to the fabric or Programmable Logic Block (PLB) resources. Both serve the same purpose but are used in different contexts.

SB_GB_IO connects to the package pad or pin, while SB_GB connects to nets or internal signals that are not directly connected to the package pad. For example, internally generated clock signals from user logic can drive fabric or PLB resources.

When using the SB_GB_IO resource, always ensure that you select the global buffer I/O resource. If not, you may encounter error in iCEcube2 as shown in the figure below.

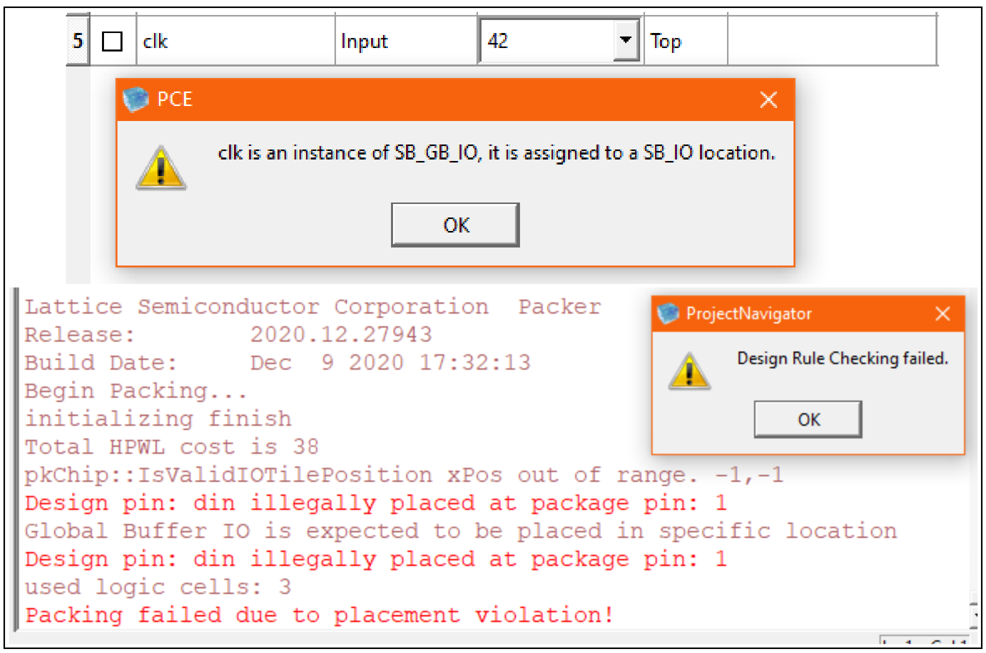


Figure 2.15. iCEcube2 SW DRC Error for Global Buffer IO

Figure below shows how two different resources are used. In this case, the signal connected to SB_GB_IO is the clk pin, which is directly connected to the device pad. SB_GB, on the other hand, connects to internal signals or modules.

```
module top (clk, rst, din, doutp,doutp2);

input clk, rst;
input  din;
output  [1:0] doutp;;
output  [1:0] doutp2;

wire clk_pll;
wire rst;
reg  [1:0] datareg;
reg  [1:0] datareg2;
reg clk_dv;
wire  dinw;
wire  [1:0] doutw;
wire clkgbio , clkgbnet;

SB_GB_IO My_Clock_Buffer_Package_Pin ( .PACKAGE_PIN (clk), .CLOCK_ENABLE (1'b1), .INPUT_CLK (1'b1),.OUTPUT_CLK (),
.OUTPUT_ENABLE (), .D_OUT_0 (), .D_OUT_1 (),.D_IN_0 (),.D_IN_1 (), .GLOBAL_BUFFER_OUTPUT (clkgbio));
defparam My_Clock_Buffer_Package_Pin.PIN_TYPE = 6'b000000;

SB_IO i_test1
(.PACKAGE_PIN (din), .LATCH_INPUT_VALUE (), .CLOCK_ENABLE (), .INPUT_CLK (), .OUTPUT_CLK (), .OUTPUT_ENABLE (), .D_OUT_0 (),
.D_OUT_1 (), .D_IN_0 (dinw), .D_IN_1 () );
defparam i_test1.PIN_TYPE = 6'b000001;
defparam i_test1.PULLUP = 1'b1;
defparam i_test1.IO_STANDARD = "SB_LVDS_INPUT";
defparam i_test1.NEG_TRIGGER = 1'b0;


//iCE40HandbookTest_pll iCE40HandbookTest_pll_inst(.REFERENCECLK(clkgb), .PLLOUTCORE(clk_pll), .PLLOUTGLOBAL(), .RESET(sw), .LOCK(rst));
always @(posedge clkgbio)
    begin
        if(rst == 1'b0)
            begin
                datareg <= 2'b0;
            end
        else
            begin
                datareg <= dinw;
            end
    end

always @(posedge clk)
    begin
        if(rst == 1'b0)
            clk_dv <= 1'b0;
        else
            clk_dv <= ~clk_dv;
    end

SB_GB My_Clock_Net (.USER_SIGNAL_TO_GLOBAL_BUFFER(clk_dv), .GLOBAL_BUFFER_OUTPUT (clkgbnet));

always @(posedge clkgbnet)
    begin
        if(rst == 1'b0)
            begin
                datareg2 <= 2'b1;
            end
        else
            begin
                datareg2 <= dinw + 1'b1;
            end
    end
```

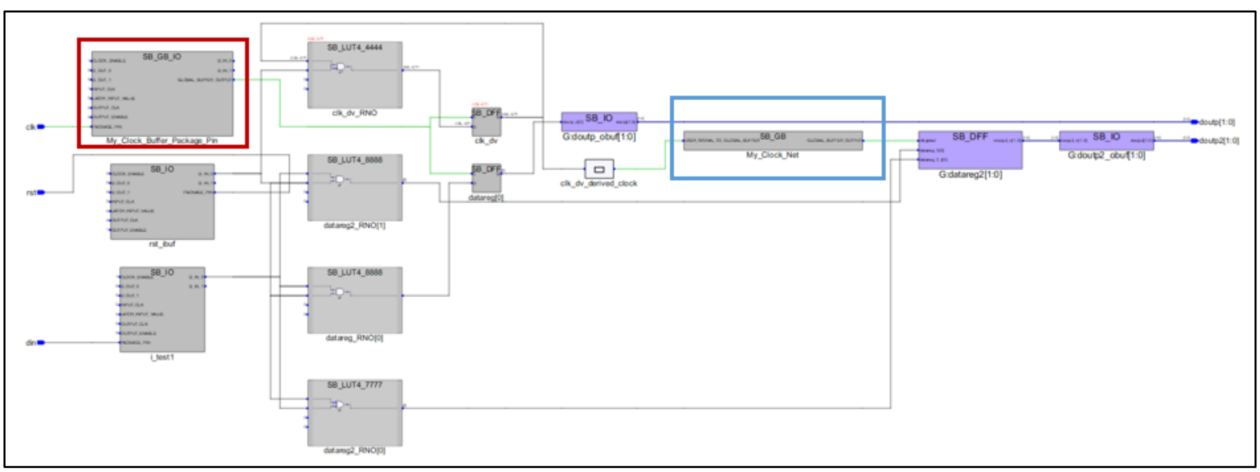


Figure 2.16. iCEcube2 RTL and Generated Netlist Example for Global Buffer Primitives

2.2.1.4. iCE40 Device I/O Behavior

Specific pins within the device exhibit different behaviors depending on the device state. These behaviors can be grouped into three categories:

- When the device is unpowered
- When the device is powered but empty
- When the device is asserted

Table 2.12. I/O Behavior Based on Defined Status

<table>
<tr><th>Pin Type</th><th>Unpowered Device</th><th>Empty Device</th><th>CRESET Asserted</th></tr>
<tr><td>GPIO</td><td rowspan="8">Tristate</td><td>Tristate with Pull Up</td><td>Tristate with Pull Up</td></tr>
<tr><td>RGB</td><td>Tristate</td><td>Tristate</td></tr>
<tr><td>CDONE</td><td>Active Low</td><td>Active Low</td></tr>
<tr><td>GBIN</td><td>Tristate with Pull Up</td><td>Tristate with Pull Up</td></tr>
<tr><td>SS CONFIG</td><td>Active High</td><td>Tristate with Pull Up</td></tr>
<tr><td>SCK CONFIG</td><td>Active High</td><td>Active High</td></tr>
<tr><td>SI CONFIG</td><td>Tristate with Pull Up</td><td>Tristate with Pull Up</td></tr>
<tr><td>SO CONFIG</td><td>Active High</td><td>Tristate with Pull Up</td></tr>
</table>

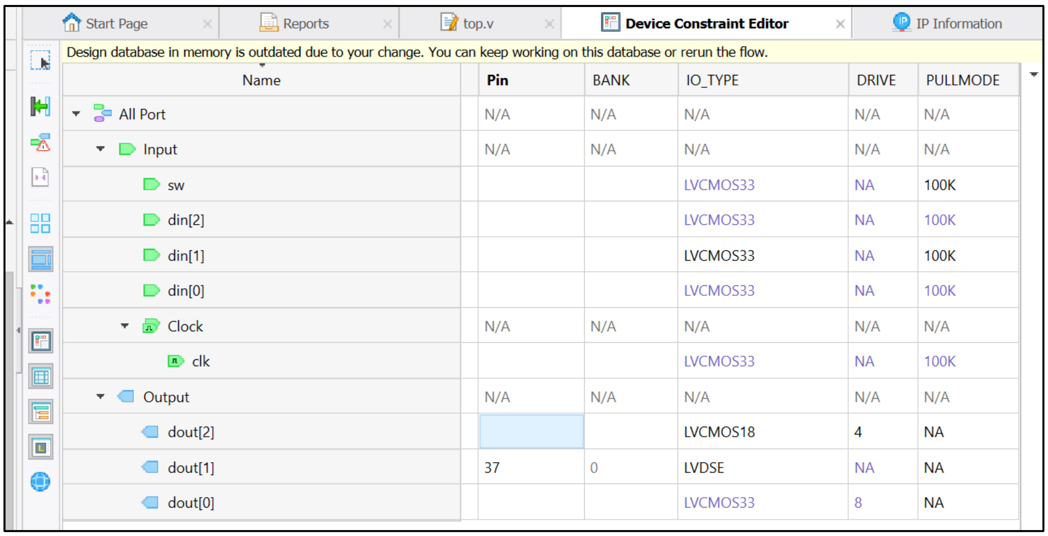
2.2.1.5. iCE40 I/O Implementation on Lattice Radiant

The only iCE40 device supported in Radiant is the iCE40 UltraPlus. The primary difference between using iCE40 devices in Radiant versus iCEcube2 is the ease of use provided by Lattice Radiant software.

In iCEcube2, you must manually include I/O primitives, as described in the previous section. In contrast, Lattice Radiant handles I/O primitives internally, although you still have the option to instantiate equivalent primitives manually.

For example, in Lattice Radiant, to configure I/O parameter for LVDS signal for a certain signal, you can set the parameter directly in Device Constraint Editor.

However, in iCEcube2, you must set the LVDS configuration through SB_IO parameters before declaring it in the Pin Constraints Editor, because the tool assumes LVCMOS by default.



Name	Pin	BANK	IO_TYPE	DRIVE	PULLMODE
All Port	N/A	N/A	N/A	N/A	N/A
Input	N/A	N/A	N/A	N/A	N/A
sw			LVCMOS33	NA	100K
din[2]			LVCMOS33	NA	100K
din[1]			LVCMOS33	NA	100K
din[0]			LVCMOS33	NA	100K
Clock	N/A	N/A	N/A	N/A	N/A
clk			LVCMOS33	NA	100K
Output	N/A	N/A	N/A	N/A	N/A
dout[2]			LVCMOS18	4	NA
dout[1]	37	0	LVDSE	NA	NA
dout[0]			LVCMOS33	8	NA

Figure 2.17. Configure I/O Parameter for LVDS signal in Lattice Radiant

For the exact equivalence of the I/O primitives from iCEcube2 to Radiant, refer to the Migrating iCEcube2 iCE40 UltraPlus Designs to Lattice Radiant Software.

2.2.2. iCE40 sysClock OSC

As discussed in Products Overview section, on-chip oscillators are supported only in iCE40 UltraLite, iCE40 Ultra, and iCE40 UltraPlus devices. These devices support a maximum of 48 MHz for the high-frequency oscillator and 10 kHz for the low-frequency oscillator. In addition to the frequency difference, the 48 MHz oscillator includes a divider that can generate clock frequencies of 24 MHz, 12 MHz, and 6 MHz, depending on the selected divider value.

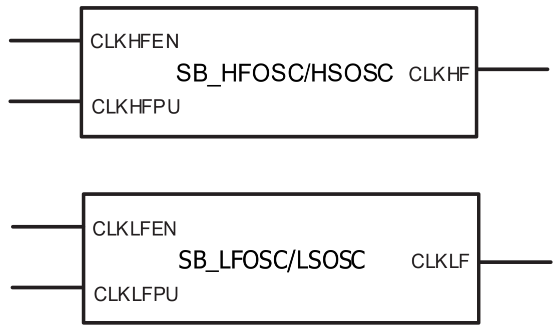


Figure 2.18. On-Chip Oscillator

The oscillator output can be routed to either general routing or primary routing (global routing). By default, it uses primary routing for better performance. Note that the oscillator output stabilizes after 100 µs.

The oscillator has two key inputs: CLKHFPU/CLKLFPU and CLKHFEN/CLKLFEN. CLKHFPU/CLKLFPU powers up the oscillator when set to HIGH, while CLKHFEN/CLKLFEN controls the oscillator output frequency; setting to HIGH generates the output frequency.

The oscillator output requires approximately 100 µs to stabilize after CLKHFPU/CLKLFPU is asserted. From a design perspective, you can tie these pins to HIGH. However, if you plan to control CLKHFPU externally, consider the 100 µs delay required for the output to stabilize to avoid design conflicts..

To use the oscillator in the supported iCE40 devices, you must instantiate the oscillator module. Refer to the Design Entry section in the iCE40 Oscillator User Guide (FPGA-TN-02008).

2.2.3. iCE40 sysClock PLL

Phase-locked loop (PLL) blocks are essential components on iCE40 devices. The PLL support multiple applications such as Clock Frequency Generation through Input Reference clock Multiplication and Division, Clock synchronization, and Frequency filtering.

Depending on the package and device density, iCE40 devices contain at least one PLL block. To identify the number of PLL blocks available for your selected device, refer to Table 2.1. iCE40 Products Overview or Table 1.1 in iCE40 sysCLOCK PLL Technical Note (FPGA-TN-02052).

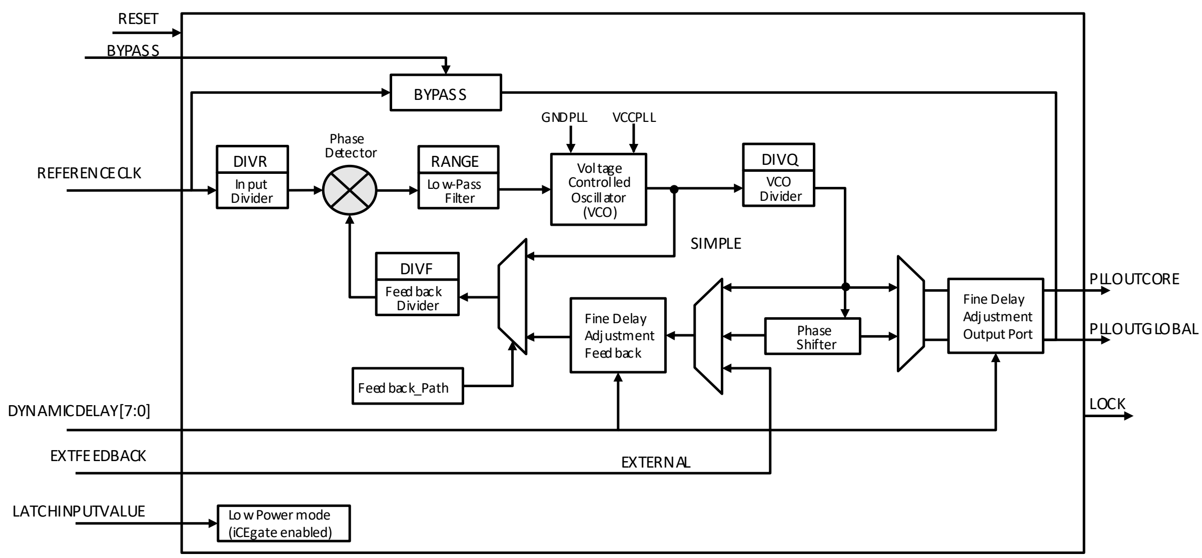


Figure 2.19. sysClock PLL Architecture

iCE40 devices include global routing resources such as low-skew global lines and dedicated routing resources for clock distribution, which are useful for signals with high fanout. It is recommended to use global routing resources, such as the global buffer inputs (GBIN, Gx, PCLK), to drive the input reference clock of the PLL, especially if it is driven by an external source or through the FPGA pad.

When selecting a pad for the PLL reference clock, consider the following:

1. You can use either GBIN/Gx buffers or other DPIO/PIO pins.
2. It is recommended to use a GPLL pin, as it offers the lowest routing delay to the PLL resource.
3. If a PLL resource is used and the GPLL pin is not selected as the input reference clock pad, you cannot use the GPLL pin as an input connection for other pad signals. In the example below, clk is assigned to pin 13 as the input reference clock of the PLL. Attempting to use pin 35 for other input signals results in a tool error due to hardware limitations.

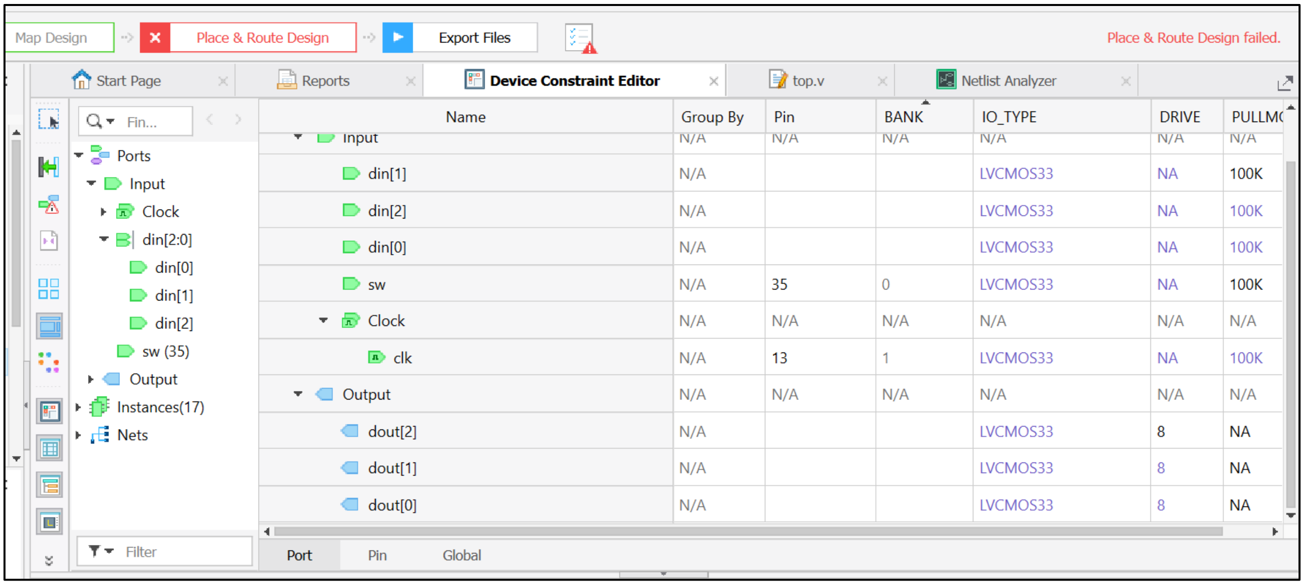


Figure 2.20. Example of Incorrect Usage of Pins when PLL is Utilized

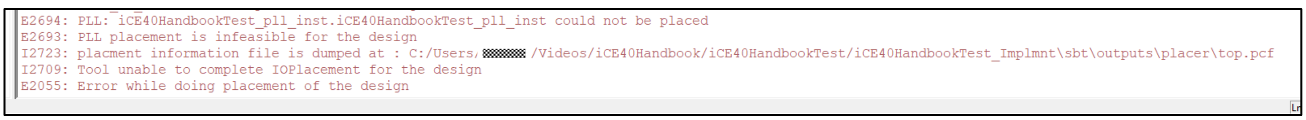


Figure 2.21. iCEcube2 Error Message If Item #3 is Violated

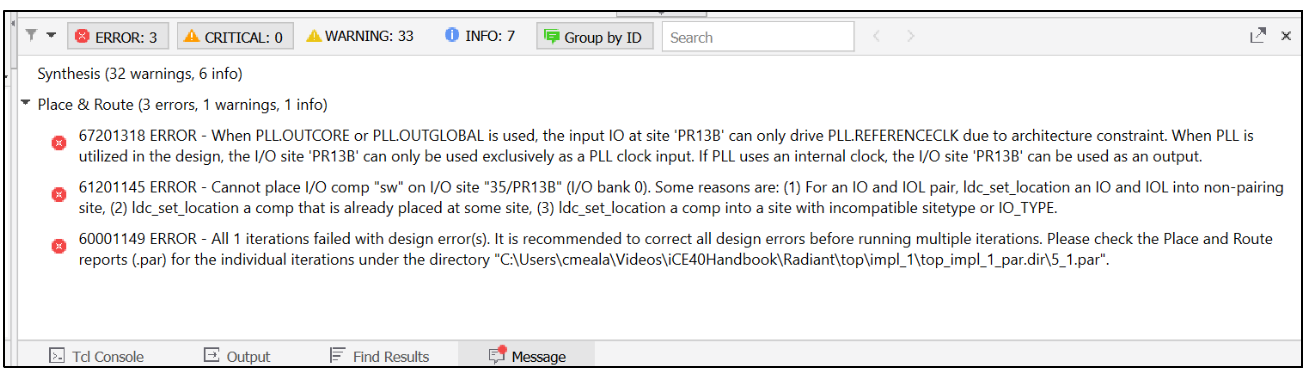


Figure 2.22. Radiant Error Message if #3 is Violated

4. For PLL module generation parameters, if you intend to use the GPLL pin as the PLL input, you must select Dedicated Clock Pad (Single Ended).

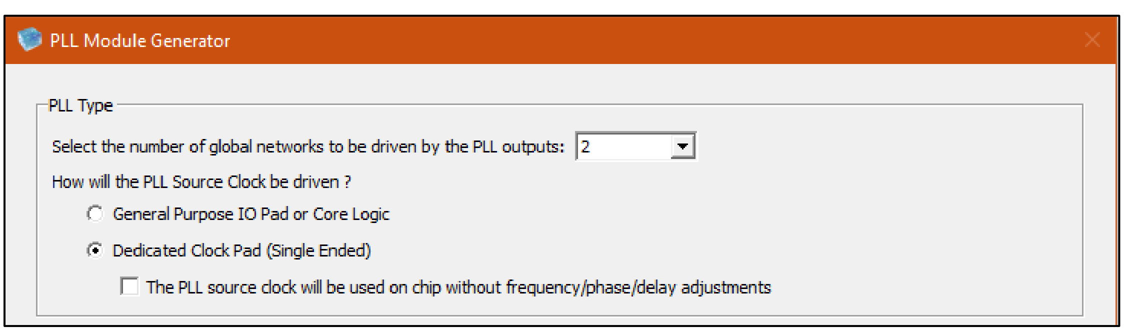


Figure 2.23. PLL Type Definition for Module Generation

iCE40 devices include two output clock ports: OUTGLOBAL and OUTCORE. These ports differ based on the type of load they drive.
OUTGLOBAL connects directly to the primary clock network and should only be used to drive clock loads (for example, clock pins of registers or flip-flops). In the iCE40 fabric, signals routed through the primary clock network (such as OUTGLOBAL) cannot return to general routing to drive data loads. In this case, use OUTCORE, which is designed to drive data loads and uses general routing resources. Use the OUTGLOBAL and OUTCORE output of the PLL as follow:

- Use OUTGLOBAL if you intend to drive an internal clock load (for example, flip-flop clock pins) from the FPGA.
- Use OUTCORE if you intend to drive an output pad or an internal clock load (for example, flip-flop clock pins) from the FPGA.

If you violate these guidelines in Lattice Radiant, the tool blocks you from performing place-and-route (PAR) and displays an error, as shown in Figure 2.24 and Figure 2.25. Note that iCEcube2 may not prevent this invalid usage, so follow the recommended guidelines to avoid design issues.

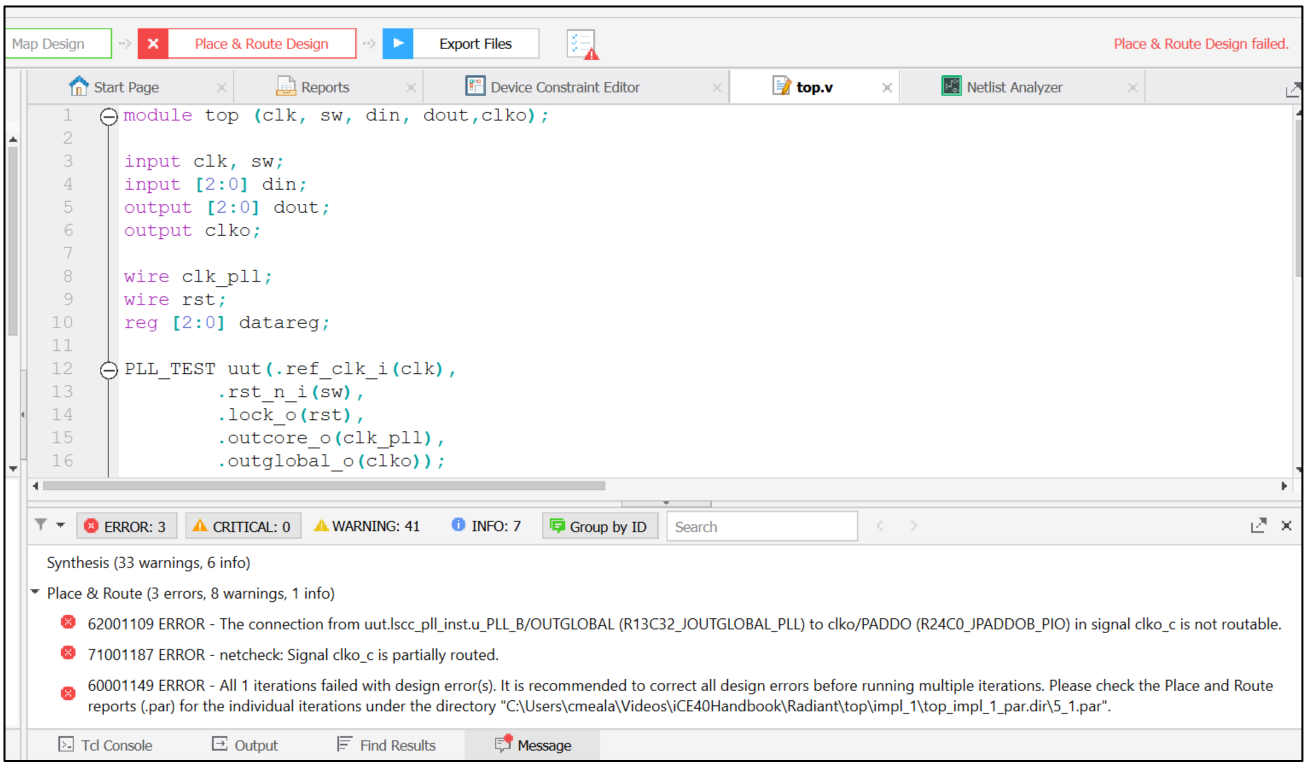


Figure 2.24. Radiant PAR Error Due to Connection of OUTGLOBAL to Output Pad

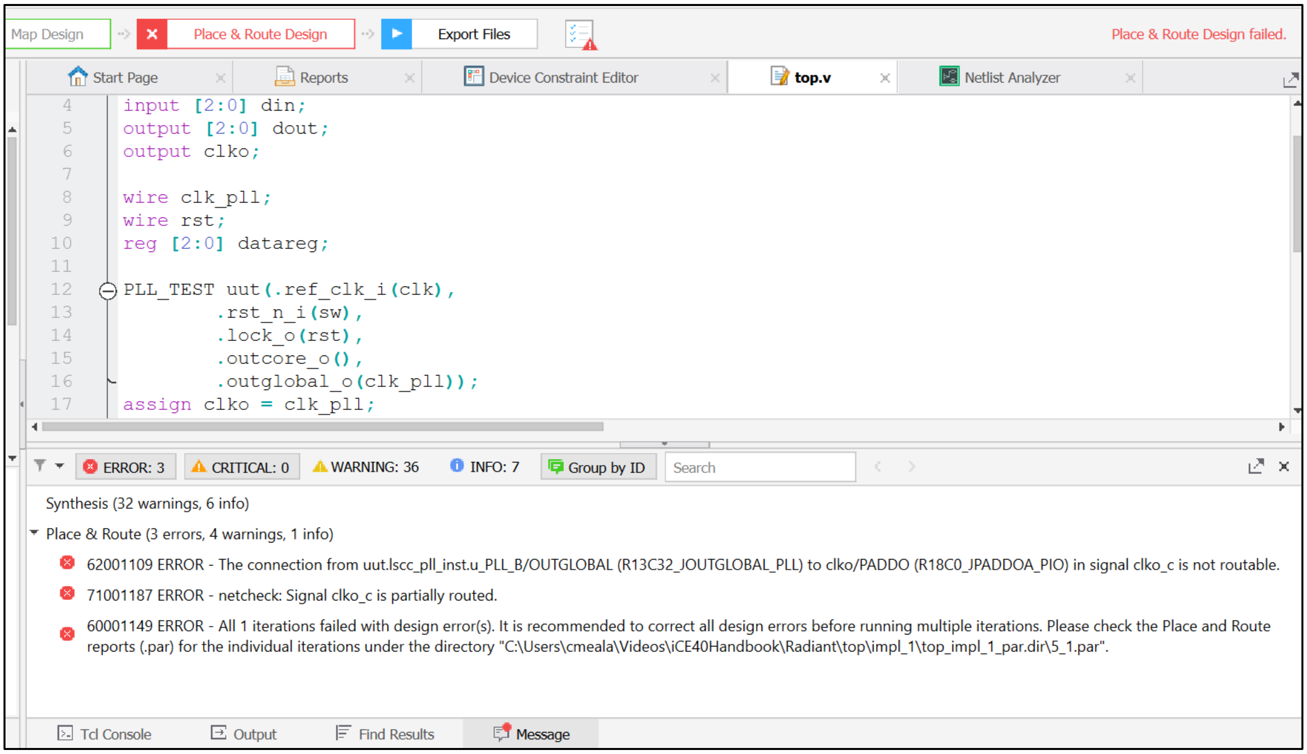


Figure 2.25. Radiant PAR Error Due to Connection of OUTGLOBAL to clock load and output Pad

For details on how PLL output frequencies are calculated and generated in Lattice Radiant and iCEcube2, refer to Sections 3.5 and 4 of iCE40 sysCLOCK PLL Technical Note (FPGA-TN-02052) respectively.

From a design perspective, like the oscillator module, the PLL RESET pin can be fully asserted (tied to HIGH). You do not need to implement a minimum 10 ns delay for the reset pulse width as indicated in the iCE40 sysCLOCK PLL Technical Note (FPGA-TN-02052). The power-on reset of the iCE40 device compensates for this value to completely initialize the PLL block.

3. Non-Volatile Configuration Memory (NVCM)

This section applies to iCE40 LP, iCE40 HX, iCE40 Ultra, iCE40 UltraLite, and iCE40 UltraPlus devices only. All standard iCE40 devices have an internal NVCM large enough to program a complete iCE40 device, including initializing all embedded block RAM.

The NVCM memory also provides a very high programming yield due to extensive error checking and correction (ECC) circuitry. It is ideal for cost-sensitive, high-volume production applications, saving the cost and board space associated with an external configuration PROM.

In addition, the NVCM provides exceptional design security, protecting critical intellectual property (IP). Its contents are entirely contained within the iCE40 device and are not readable once protected by the one-time programmable security bits.

There is no observable difference between a programmed or unprogrammed memory cell using optical or electron microscopy. The NVCM memory has a programming interface similar to a 25-series SPI serial Flash PROM. Consequently, it can be programmed using Diamond Programmer (version 2.2 or later) before or after circuit board assembly or programmed in-system from a microprocessor or other intelligent controller. The NVCM can also be pre-programmed at the factory.

The NVCM can be programmed in the following ways:

- Diamond Programmer
  - Programming with the Diamond Programmer (version 2.0.1 or later) is recommended for prototyping.
  - Programming is supported using the Lattice programming cable. For more information refer to the Diamond Programmer Online Help and the Programming Cables User Guide (FPGA-UG-02042).
- Factory Programming
  - The Lattice factory offers NVCM programming. For more information, contact Technical Support Assistance.
- Embedded Programming
  - The NVCM can be programmed using a processor. For more information, contact Technical Support Assistance.

3.1. Top Level NVCM Programming Flow

The programming of the NVCM requires a series of steps:

1. Power up device, connect to the programmer and enable NVCM programming mode.
2. Busy Status Bit Checking:
   Monitor the busy status register. This operation takes time to complete.
3. Set Up Reading Parameter in Trim Registers:
   The trim parameter registers hold the hardware-specific codes for reading the NVCM memory.
4. Verify chip ID:
   Verify that the silicon signature corresponds to the device number in the NVCM file and matches the selected device in the programmer GUI.
5. Blank Check on the NVCM Trim Parameter OTP block:
   Before programming, check the NVCM for quality control purposes. Perform a blank check to confirm the device has not been previously programmed. This is done by issuing a read command that returns the value of the trim parameter OTP block.
6. (OPTIONAL) Blank Check on the NVCM Arrays:
   Perform a blank check to confirm the NVCM array has not been previously programmed. This is done by issuing a read command that returns the entire NVCM array.
7. Setup Program Parameters in Trim Registers:
   The trim parameter registers hold the hardware-specific codes for programming the NVCM memory.
8. Program the NVCM main Arrays
   Program the data from the .nvcm file into the NVCM. This file includes embedded CRC data. In addition, the configuration logic automatically generates an ECC pattern for each 64-byte page, which is stored in the NVCM.

9. Verify the NVCM main Arrays:
   Correct NVCM programming results in the CDONE pin going high, indicating the CRC check passed. You can also read back the programmed NVCM bits and compare them with the NVCM file. Since the NVCM file contains fuse, command, and address information, only programmed fuse bits can be compared.
10. Program and Verify the Trim Parameter OTP Block:
    After programming the NVCM, program the trim parameter OTP to inform the internal control logic that configuration should now be retrieved from the NVCM during power-up.
11. Confirm Proper Programming:
    After programming the NVCM and trim parameter OTP, each time the device powers up or CRESET_B toggles, the internal state machine performs an ECC check on each page, automatically correcting any incorrect data. Next, the data is transferred to CRAM, and the device compares the CRC value programmed into the NVCM (from the .nvcm file) with the value calculated by the device. If the CRC comparison passes, CDONE changes to true, indicating correct programming. A formal verify pass of the programmed data is not strictly required.
12. (OPTIONAL) Program and Verify of the Security Bits:
    If desired program and verify the security bits, otherwise go to step 13. After verifying the security bits, step 11 may be repeated to activate security and confirm device startup.
13. Program the Silicon Signature:
    After programming the device, program the silicon signature. This non-volatile signature retains the programming status, programmer manufacturer, model, and date.

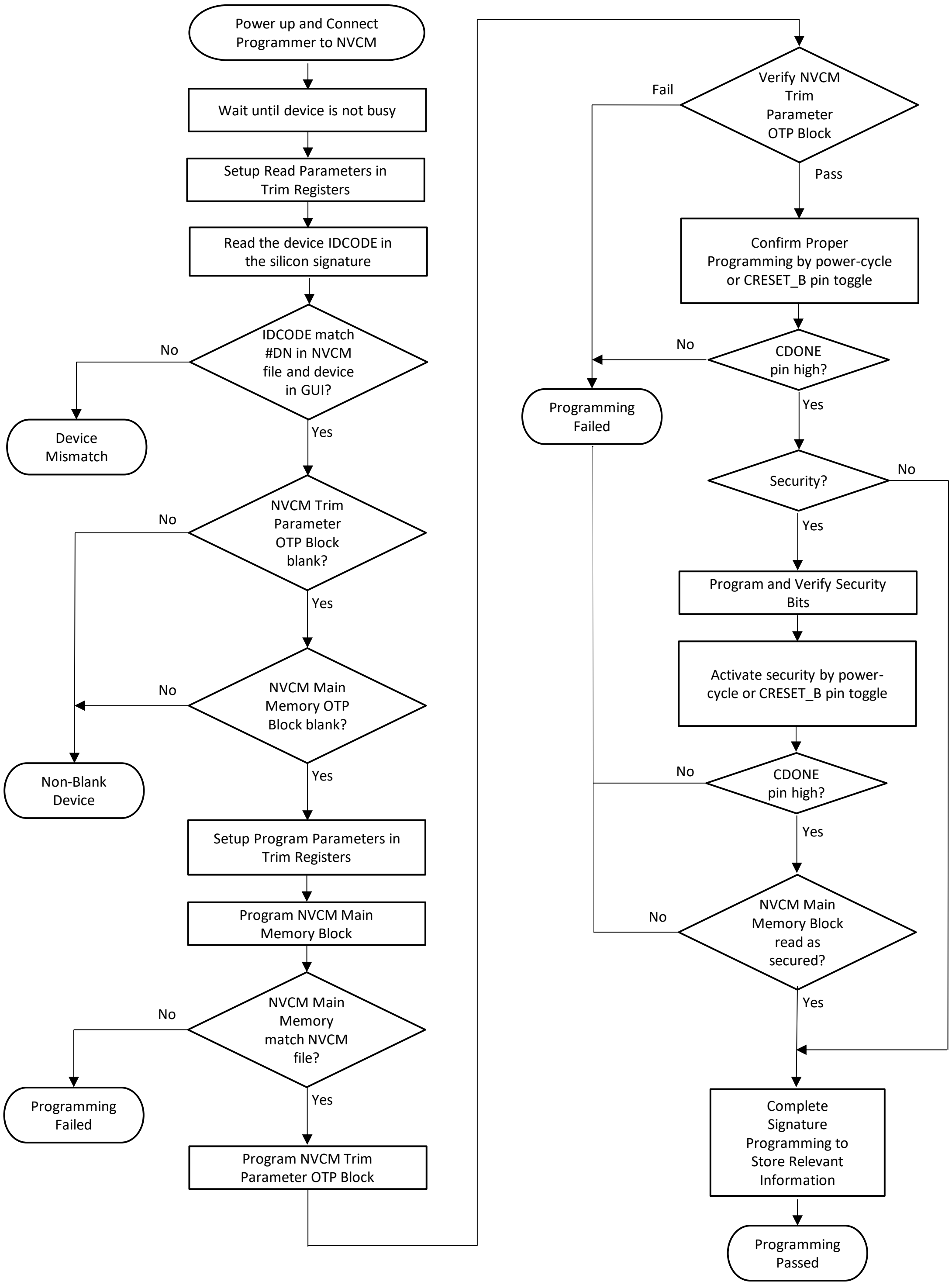


Figure 3.1. Top Level NVCM Programming Flow Chart

4. Board Design Reference Point

Below are the list of development kits and boards that are available for iCE40 device family.

Table 4.1. iCE40 Development Kits and Boards

Devices	Development Kits and Boards
iCE40 UltraPlus	iCE40 UltraPlus Breakout Board
iCE40 UltraLite	iCE40 UltraLite Breakout Board
iCE40 Ultra	iCE40 Ultra Breakout Board
	iCE40 Ultra Wearable Development Platform
iCE40HX	iCEstick Evaluation Kit
	iCE40-HX8K Breakout Board
iCE40LM	iCE40LM4K Sensor Evaluation Kit

5. iCE40 Software Support

iCEcube2 and Lattice Radiant software provide a complete design environment for the iCE40 device family.

The table below lists iCE40 devices supported in iCEcube2 and Lattice Radiant software.

Table 5.1. Supported iCE40 Devices in iCEcube2 and Lattice Radiant Software.

Software	iCE40 UltraPlus	iCE40 Ultra/UltraLite	iCE40 LP/HX
iCEcube2 Design Software	Available	Available	Available
Lattice Radiant Software	Available	–	–

5.1. Software Installation and Licensing Resources

5.1.1. iCEcube2 Design Software

iCEcube2 supports Windows 10 (32-bit and 64-bit) and Red Hat Enterprise Linux WS version 4, 5, and 6. You can download the iCEcube2 installer for Linux and Windows from the links below:

- Linux: iCEcube2 2020-12 for Linux
- Windows: iCEcube2 2020-12 for Windows

5.1.2. Lattice Radiant Software

You can visit the Lattice Radiant Software web page and navigate to the Software Downloads & Documentation section.

Select Downloadable Software to see the latest installer for Linux and Windows. Click on either version to download the software.

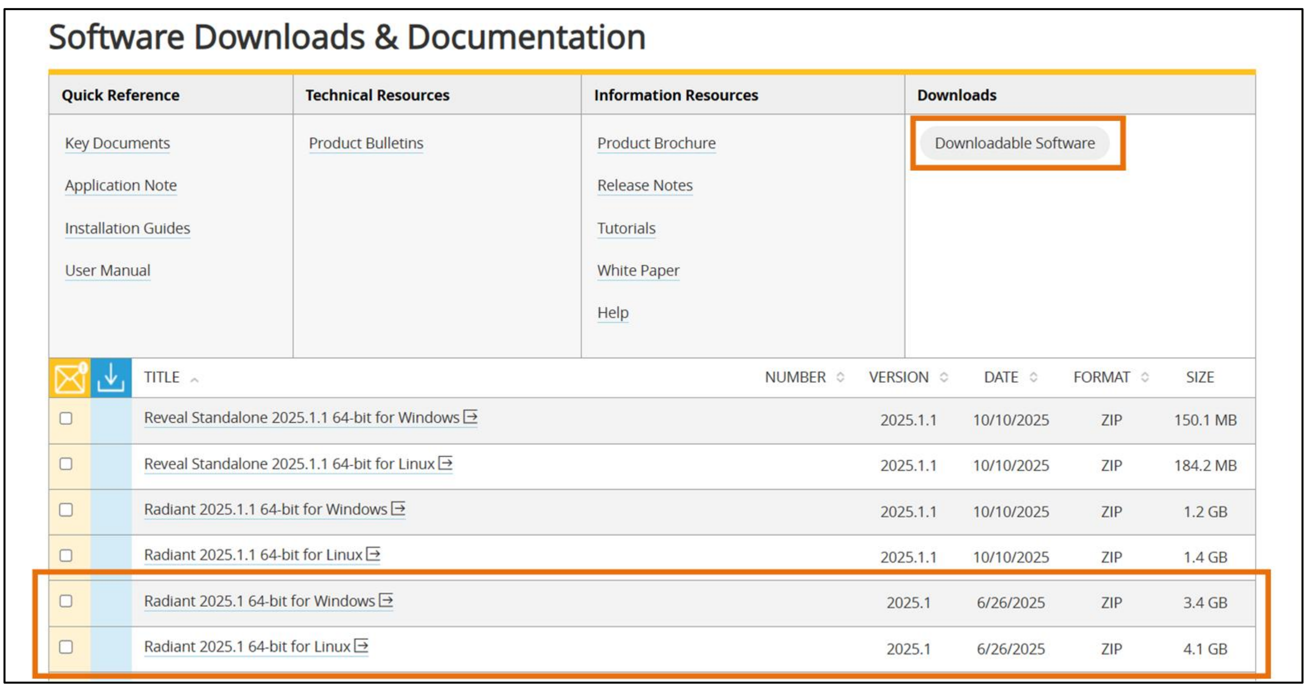


Figure 5.1. Lattice Radiant Software Download

For installation requirements refer to the Lattice Radiant Software Installation Guides under Installation Guides.

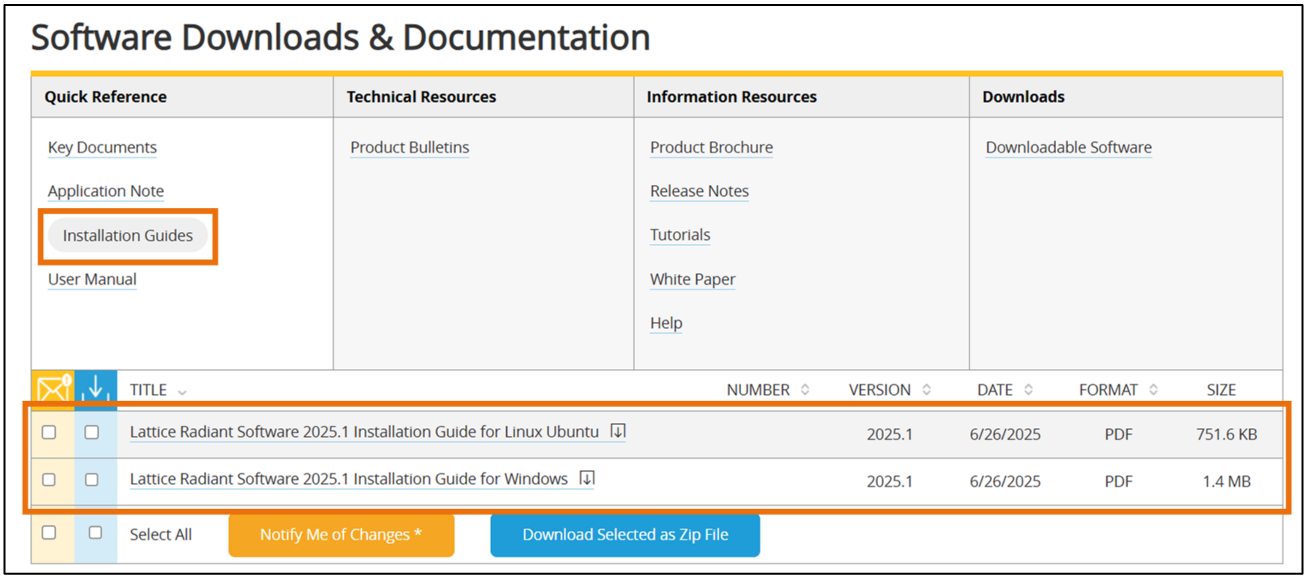


Figure 5.2. Lattice Radiant Software Installation Guide

5.2. License

5.2.1. iCEcube2 License

You can request an iCEcube2 license from iCEcube2 license web page.

5.2.2. Lattice Radiant License

Lattice Design Tools require a license to utilize the software. User may request the Lattice Radiant Software license from the links below.

- Node-locked License
- Floating License
- Buy or Renew License:
  - Lattice Online Store
  - Local Sales Representative or Distributor

5.3. iCEcube2 Linux Installation

Before installing iCEcube2 on Linux based operating system, you must install specific package on your operating system. Install or update the following packages before following the instruction in sections 5.3.1, 5.3.2, or 5.3.3.

For Ubuntu or Debian:

```
sudo apt-get install gcc-multilib
sudo apt-get install g++-multilib
sudo apt-get update
```

For CentOS or RHEL:

```
sudo yum update
```

5.3.1. iCE40 Installation (RHEL or CentOS)

1. Run the command below to install all necessary dependencies. These packages provide 32-bit compatibility required for iCEcube2 to run on RHEL or CentOS systems.
   ```
   sudo yum install zlib-1.2.7-20.el7_9.i686 libpng12-1.2.50-10.el7.i686 libSM-1.2.2-2.el7.i686 libXi-1.7.9-1.el7.i686 libXrender-0.9.10-1.el7.i686 libXrandr-1.5.1-2.el7.i686 libXfixes-5.0.3-1.el7.i686 libXcursor-1.1.15-1.el7.i686 libXinerama-1.1.3-2.1.el7.i686 freetype-2.8-14.el7_9.1.i686 fontconfig-2.13.0-4.3.el7.i686 glib2-2.56.1-9.el7_9.i686
   ```
2. iCEcube2 requires the primary network interface to be named *eth0* for license validation.
   a. Run the command below to identify your current network interface name. This command displays all active network interfaces. Note the name of the interface you plan to rename (for example, in the figure below, *enp0s3*)
      ```
      ifconfig
      ```
   b. Disable the interface before renaming. Replace <host_id_name> with the name identified in step a.
      ```
      sudo ip link set <host_id_name> down
      ```
   c. Rename the interface to eth0 so iCEcube2 can validate the license using the correct host ID.
      ```
      sudo ip link set <host_id_name> name eth0
      ```
   d. Restore network connectivity after renaming the interface.
      ```
      sudo ip link set eth0 up
      ```
3. Navigate to the directory that contains the installer file and run the installer using this command.
   ```
   sudo ./iCEcube2setup_Dec_10_2020_2012
   ```

5.3.2. iCE40 Installation (Ubuntu)

Note: The following installation steps are for Ubuntu. While the installation process in not officially supported, you can run the installation by following these steps.

1. Run the command below to install the necessary 32-bit packages for iCEcube2. These also enable iCEcube2 to run on 64-bit Ubuntu system by providing required 32-bit compatibility.

```
sudo apt install gcc-10-base:i386 ibc6:i386 libcrypt1:i386 libgcc-s1:i386 libidn2-0:i386
libunistring2:i386 zlib1g:i386 libxext6:i386 libsm6:i386 libxi6:i386 libxrender1:i386
libxrandr2:i386 libxfixes3:i386 libxcursor1:i386 libxinerama1:i386 libfreetype6:i386
libfontconfig1:i386 libglib2.0-0 lib32stdc++6 libglib2.0-0:i386
```

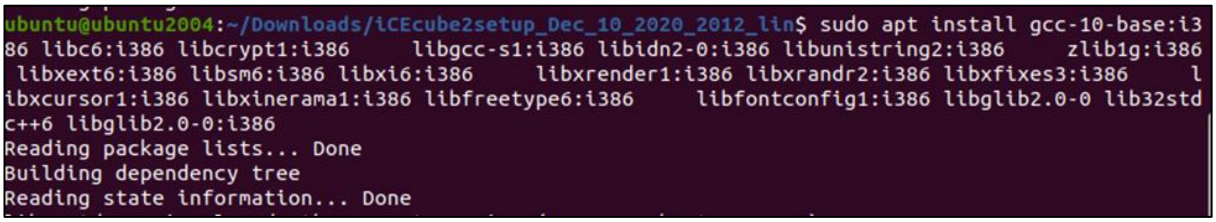


Figure 5.3. Installing iCEcube2 Packages in Ubuntu

2. Download and install the libpng12 package using the following commands:

```
wget https://download.dominiosistemas.com.br/instalacao/diversos/linux_dw/libpng12-
0_1.2.54-1ubuntu1b_i386.deb
sudo dpkg -i libpng12-0_1.2.54-1ubuntu1b_i386.deb
```

3. iCEcube2 requires the primary network interface to be named *eth0* for license validation.
   a. Run the command below to identify your current network interface name. This command displays all active network interfaces. Note the name of the interface you plan to rename. For example, *enp0s3* as shown in the figure below.

   ```
   ifconfig
   ```

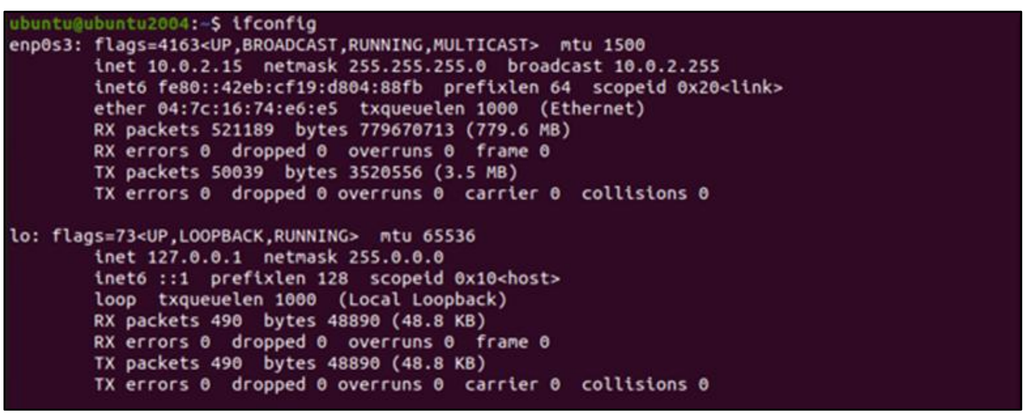


Figure 5.4. Active Network Interfaces Displayed by ifconfig Command in Ubuntu

   b. You must disable the interface before renaming it. Run the command below to bring the interface down. Replace <host_id_name> with the name you identified in step a (in the case above enp0s3).

   ```
   sudo ip link set <host_id_name> down
   ifconfig
   ```

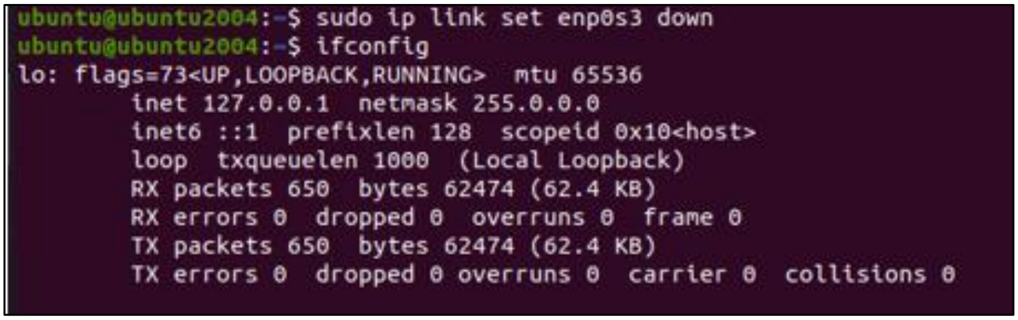


Figure 5.5. Disabled enp0s3 Network Interface Before Renaming in Ubuntu

c. Renaming the interface to *eth0* ensures iCEcube2 can validate the license using the correct host ID. Run the command below to rename the interface to *eth0*:

```
sudo ip link set <host_id_name> name eth0
```

d. Run the command below to bring the renamed interface back online. This restores network connectivity after renaming the interface.

```
sudo ip link set eth0 up
```

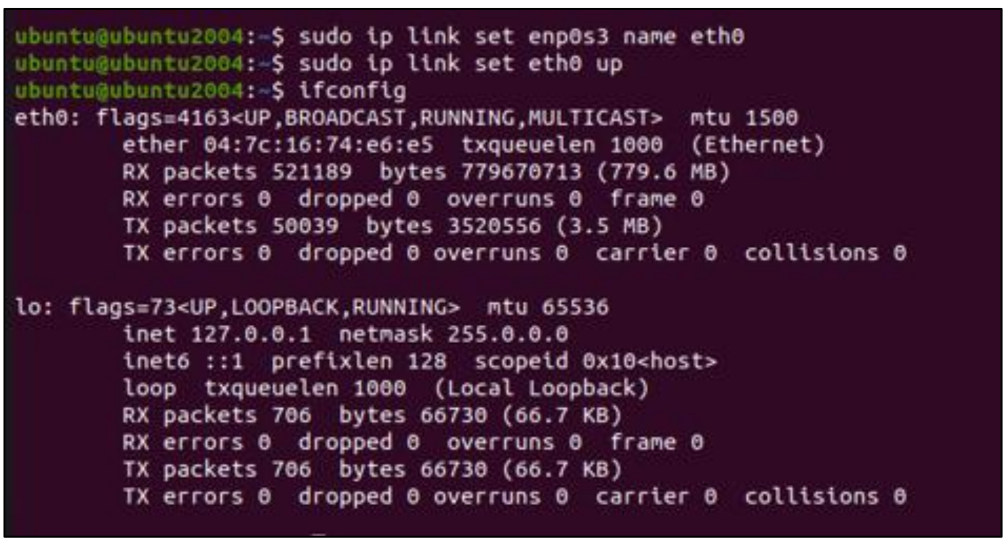


Figure 5.6. Renamed Network Interface and Restored Network in Ubuntu

4. Navigate to the directory that contains the installer file and run the installer using this command.

```
sudo ./iCEcube2setup_Dec_10_2020_2012
```

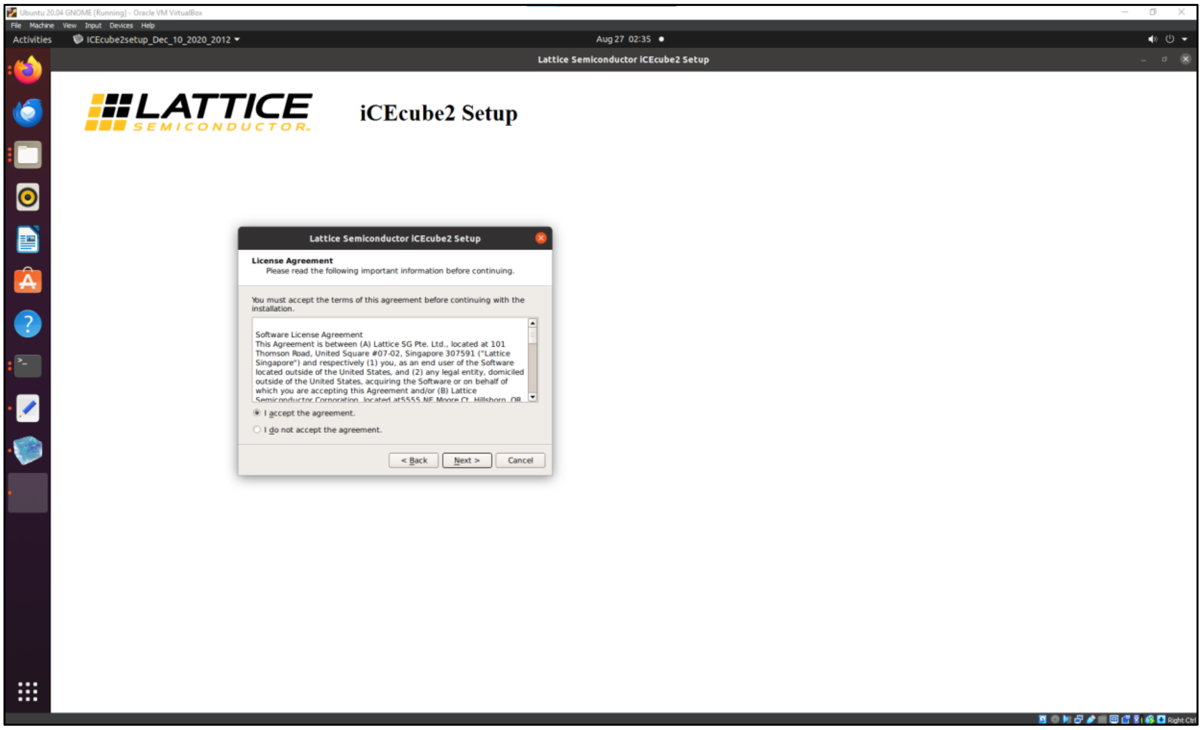


Figure 5.7. iCEcube2 Installer Run in Ubuntu

5.3.3. iCE40 Installation for Debian

Note: The following installation steps are for Debian OS. While the installation process in not officially supported, you can run the installation by following these steps.

Since Ubuntu is a different variation of Debian OS there is a different set of packages that are needed to allow the installer to ran. In this case you may be encounter the following errors when running the installer or when running the package installations:

```
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ ./iCEcube2setup_Dec_10_2020_2012
bash: ./iCEcube2setup_Dec_10_2020_2012: cannot execute: required file not found
```

Figure 5.8. Example Error Encountered During Installation in Debian

1. Change the current working directory to the folder where the installer file is located and run the following command to identify missing packages to run the installer.

```
ldd iCEcube2setup_Dec_10_2020_2012
```

```
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ ldd iCEcube2setup_Dec_10_2020_2012
        linux-gate.so.1 (0xf7f89000)
        libz.so.1 => not found
        libXext.so.6 => not found
        libX11.so.6 => not found
        libpng12.so.0 => not found
        libSM.so.6 => not found
        libICE.so.6 => not found
        libXi.so.6 => not found
        libXrender.so.1 => not found
        libXrandr.so.2 => not found
        libXfixes.so.3 => not found
        libXcursor.so.1 => not found
        libXinerama.so.1 => not found
        libfreetype.so.6 => not found
        libfontconfig.so.1 => not found
        libgthread-2.0.so.0 => not found
        libglib-2.0.so.0 => not found
        librt.so.1 => /lib32/librt.so.1 (0xf7f69000)
        libdl.so.2 => /lib32/libdl.so.2 (0xf7f62000)
        libpthread.so.0 => /lib32/libpthread.so.0 (0xf7f5d000)
        libstdc++.so.6 => /lib32/libstdc++.so.6 (0xf7c00000)
        libm.so.6 => /lib32/libm.so.6 (0xf7e58000)
        libgcc_s.so.1 => /lib32/libgcc_s.so.1 (0xf7e31000)
        libc.so.6 => /lib32/libc.so.6 (0xf7800000)
        /lib/ld-linux.so.2 (0xf7f8b000)
```

Figure 5.9. Identify Missing Packages for Installer Execution in Debian Using ldd Command

2. Debian uses 64-bit architecture by default. These commands enable support for 32-bit packages required by iCEcube2. Run the command below to allow installation of 32-bit packages.

```
sudo dpkg --add-architecture i386
sudo apt-get update
```

```
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ sudo dpkg --add-architecture i386
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ sudo apt-get update
Hit:1 https://deb.debian.org/debian bookworm InRelease
Hit:2 https://deb.debian.org/debian bookworm-updates InRelease
Get:3 https://deb.debian.org/debian bookworm/main i386 Packages [8,680 kB]
Hit:4 https://security.debian.org/debian-security bookworm-security InRelease
Get:5 https://deb.debian.org/debian bookworm-updates/main i386 Packages [13.8 kB]
Get:6 https://security.debian.org/debian-security bookworm-security/main i386 Packages [178 kB]
Fetched 8,872 kB in 1s (7,878 kB/s)
Reading package lists... Done
```

Figure 5.10. Adds 32-Bit Architecture and Update Package Lists on Debian

3. For some packages, they can be directly installed through sudo apt install command, however there are some instances that you may need to grab the package installer through a different resource. Below are the steps done to install these packages.
   a. Install the following packages:

```
sudo apt install lib32z1
sudo apt install libx11-6:i386
sudo apt-get install aptitude
sudo apt install libx11-6:i386
```

   b. Validate if the packages are installed correct by using the ldd command once again.

```
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ ldd iCEcube2setup_Dec_10_2020_2012
        linux-gate.so.1 (0xf7f63000)
        libz.so.1 => /lib32/libz.so.1 (0xf7f2d000)
        libXext.so.6 => not found
        libX11.so.6 => /lib/i386-linux-gnu/libX11.so.6 (0xf7ddb000)
        libpng12.so.0 => not found
        libSM.so.6 => not found
        libICE.so.6 => not found
        libXi.so.6 => not found
        libXrender.so.1 => not found
        libXrandr.so.2 => not found
        libXfixes.so.3 => not found
        libXcursor.so.1 => not found
        libXinerama.so.1 => not found
        libfreetype.so.6 => not found
        libfontconfig.so.1 => not found
        libgthread-2.0.so.0 => not found
        libglib-2.0.so.0 => not found
        librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xf7dd4000)
        libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xf7dcd000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xf7dc8000)
        libstdc++.so.6 => /lib32/libstdc++.so.6 (0xf7a00000)
        libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xf7cc3000)
        libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xf7c9c000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf7600000)
        libxcb.so.1 => /lib/i386-linux-gnu/libxcb.so.1 (0xf7c6e000)
```

Figure 5.11. Validate Installed Packages Using ldd Command in Debian

   c. For the missing packages, download them using the command below:

```
wget http://ftp.cn.debian.org/debian/pool/main/libx/libxrender/libxrender1_0.9.10-1.1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/g/glib2.0/libglib2.0-0_2.74.6-2+deb12u3_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libx/libxfixes/libxfixes3_6.0.0-2_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libx/libxcursor/libxcursor1_1.2.1-1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libx/libxinerama/libxinerama1_1.1.4-3_i386.deb
http://ftp.cn.debian.org/debian/pool/main/f/freetype/libfreetype6_2.12.1+dfsg-5+deb12u3_i386.deb
http://ftp.cn.debian.org/debian/pool/main/f/fontconfig/libfontconfig1_2.14.1-4_i386.deb http://ftp.cn.debian.org/debian/pool/main/g/glib2.0/libglib2.0-0_2.74.6-2+deb12u3_i386.deb http://ftp.cn.debian.org/debian/pool/main/u/util-linux/libuuid1_2.38.1-5+deb12u1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libp/libpng1.6/libpng16-16_1.6.39-2_i386.deb http://ftp.cn.debian.org/debian/pool/main/b/brotli/libbrotli1_1.0.9-2+b6_i386.deb http://ftp.cn.debian.org/debian/pool/main/e/expat/libexpat1_2.5.0-1_i386.deb http://ftp.cn.debian.org/debian/pool/main/p/pcre2/libpcre2-8-0_10.42-1_i386.deb http://ftp.cn.debian.org/debian/pool/main/libx/libxext/libxext6_1.3.4-1+b1_i386.deb https://archive.debian.org/debian-security/pool/updates/main/libp/libpng/libpng12-0_1.2.44-1+squeeze4_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libs/libsm/libsm6_1.2.3-1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libi/libice/libice-dev_1.0.10-1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libi/libice/libice6_1.0.10-1_i386.deb
http://ftp.cn.debian.org/debian/pool/main/libx/libxi/libxi6_1.8-1+b1_i386.deb
```

```
http://ftp.cn.debian.org/debian/pool/main/libx/libxrandr/libxrandr2_1.5.2-
2+b1_i386.deb
```

d. Install the downloaded packages using the command below:

```
sudo dpkg -i libxi6_1.8-1+b1_i386.deb libice6_1.0.10-1_i386.deb libice-dev_1.0.10-
1_i386.deb libsm6_1.2.3-1_i386.deb libxrender1_0.9.10-1.1_i386.deb libpng12-
0_1.2.44-1+squeeze4_i386.deb libxext6_1.3.4-1+b1_i386.deb libglib2.0-0_2.74.6-
2+deb12u3_i386.deb libxfixes3_6.0.0-2_i386.deb libxcursor1_1.2.1-1_i386.deb
libxinerama1_1.1.4-3_i386.deb libfreetype6_2.12.1+dfsg-5+deb12u3_i386.deb
libfontconfig1_2.14.1-4_i386.deb libglib2.0-0_2.74.6-2+deb12u3_i386.deb
libuuid1_2.38.1-5+deb12u1_i386.deb libpcre2-8-0_10.42-1_i386.deb libexpat1_2.5.0-
1_i386.deb libbrotli1_1.0.9-2+b6_i386.deb libpng16-16_1.6.39-2_i386.deb
libxrandr2_1.5.2-2+b1_i386.deb
```

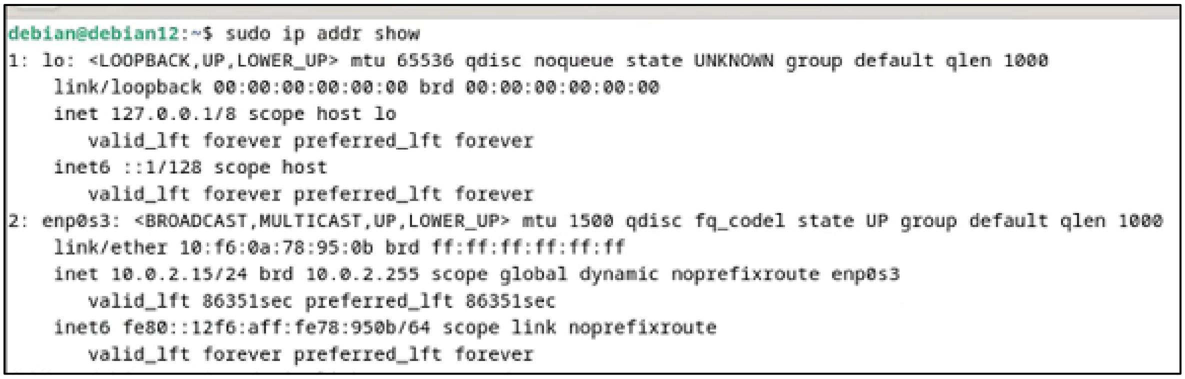
e. Validate if the packages are installed correct by using the ldd command once again.

```
debian@debian12:~/Downloads/iCEcube2setup_Dec_10_2020_2012_lin$ ldd iCEcube2setup_Dec_10_2020_2012
        linux-gate.so.1 (0xf7f00000)
        libz.so.1 => /lib32/libz.so.1 (0xf7ec9000)
        libXext.so.6 => /lib/i386-linux-gnu/libXext.so.6 (0xf7eb3000)
        libX11.so.6 => /lib/i386-linux-gnu/libX11.so.6 (0xf7d61000)
        libpng12.so.0 => /lib/libpng12.so.0 (0xf7d3d000)
        libSM.so.6 => /lib/i386-linux-gnu/libSM.so.6 (0xf7d32000)
        libICE.so.6 => /lib/i386-linux-gnu/libICE.so.6 (0xf7d15000)
        libXi.so.6 => /lib/i386-linux-gnu/libXi.so.6 (0xf7d00000)
        libXrender.so.1 => /lib/i386-linux-gnu/libXrender.so.1 (0xf7cf2000)
        libXrandr.so.2 => /lib/i386-linux-gnu/libXrandr.so.2 (0xf7ce3000)
        libXfixes.so.3 => /lib/i386-linux-gnu/libXfixes.so.3 (0xf7cd9000)
        libXcursor.so.1 => /lib/i386-linux-gnu/libXcursor.so.1 (0xf7ccc000)
        libXinerama.so.1 => /lib/i386-linux-gnu/libXinerama.so.1 (0xf7cc7000)
        libfreetype.so.6 => /lib/i386-linux-gnu/libfreetype.so.6 (0xf7bf8000)
        libfontconfig.so.1 => /lib/i386-linux-gnu/libfontconfig.so.1 (0xf7ba5000)
        libgthread-2.0.so.0 => /lib/i386-linux-gnu/libgthread-2.0.so.0 (0xf7ba0000)
        libglib-2.0.so.0 => /lib/i386-linux-gnu/libglib-2.0.so.0 (0xf7a4a000)
        librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xf7a45000)
        libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xf7a40000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xf7a3b000)
        libstdc++.so.6 => /lib32/libstdc++.so.6 (0xf7800000)
        libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xf76fb000)
        libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xf76d4000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf7400000)
        libxcb.so.1 => /lib/i386-linux-gnu/libxcb.so.1 (0xf76a6000)
        libuuid.so.1 => /lib/i386-linux-gnu/libuuid.so.1 (0xf7a2f000)
        libbsd.so.0 => /lib/i386-linux-gnu/libbsd.so.0 (0xf768f000)
        libpng16.so.16 => /lib/i386-linux-gnu/libpng16.so.16 (0xf7652000)
        libbrotlidec.so.1 => /lib/i386-linux-gnu/libbrotlidec.so.1 (0xf7644000)
        libexpat.so.1 => /lib/i386-linux-gnu/libexpat.so.1 (0xf73d4000)
        libpcre2-8.so.0 => /lib/i386-linux-gnu/libpcre2-8.so.0 (0xf7335000)
        /lib/ld-linux.so.2 (0xf7f02000)
        libXau.so.6 => /lib/i386-linux-gnu/libXau.so.6 (0xf7a28000)
        libXdmcp.so.6 => /lib/i386-linux-gnu/libXdmcp.so.6 (0xf763d000)
        libmd.so.0 => /lib/i386-linux-gnu/libmd.so.0 (0xf762e000)
        libbrotlicommon.so.1 => /lib/i386-linux-gnu/libbrotlicommon.so.1 (0xf7312000)
```

Figure 5.12. Revalidate Installed Packages Using ldd Command in Debian

4. iCEcube2 requires the primary network interface to be named *eth0* for license validation.
5. Run the command below to identify your current network interface name. Note the name of the interface you plan to rename (for example, in the figure below, *enp0s3*).

```
sudo ip addr show
```



```
debian@debian12:~$ sudo ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 10:f6:0a:78:95:0b brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 86351sec preferred_lft 86351sec
    inet6 fe80::12f6:aff:fe78:950b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 5.13. Active Network Interfaces Displayed by ip addr show Command in Debian

6. You must disable the interface before renaming it. Run the command below to bring the interface down. Replace <host_id_name> with the name you identified in step above (in the case above en0s3).

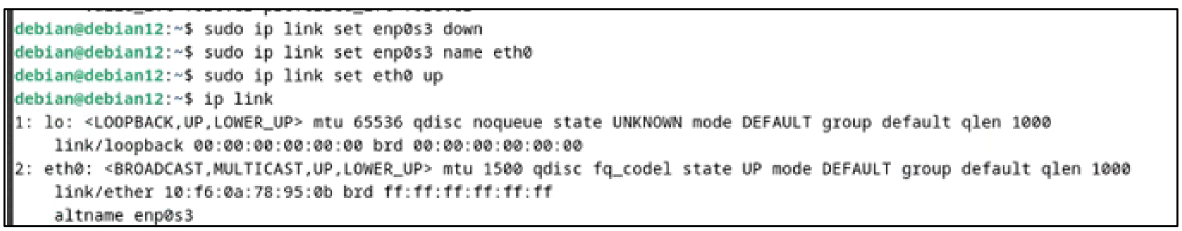
```
sudo ip link set <host_id_name> down
ip link
```

7. Renames the interface to *eth0* so that iCEcube2 can validate the license using the correct host ID.

```
sudo ip link set <host_id_name> name eth0
```

8. Restores network connectivity after renaming the interface.

```
sudo ip link set eth0 up
```



```
debian@debian12:~$ sudo ip link set enp0s3 down
debian@debian12:~$ sudo ip link set enp0s3 name eth0
debian@debian12:~$ sudo ip link set eth0 up
debian@debian12:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 10:f6:0a:78:95:0b brd ff:ff:ff:ff:ff:ff
    altname enp0s3
```

Figure 5.14. Disabled, Renamed, and Restored eth0 Network in Debian

9. Navigate to the directory that contains the installer file and run the installer using this command.

```
sudo ./iCEcube2setup_Dec_10_2020_2012
```

6. Reference Designs for iCE40 Devices

Below is the list of Reference Design available for iCE40 devices.

Table 6.1. iCE40 Reference Design

Type	Reference Design	iCE40 UltraPlus	iCE40 Ultra/UltraLite	iCE40 LP/HX
RD	Sensor Interfacing and Preprocessing	Available	Available	Available
RD	SPI-to-UART Expander	–	–	Available
RD	Long Range (LoRa) Wireless	Available	–	–
RD	PDM Microphone Aggregation	Available	–	–
RD	RGB LED Reference Design	Available	Available	Available
Demo	Generic Soft SPI Master Controller Demonstration	Available	–	–
Demo	Key Phrase Detection	Available	–	–
RD	Image Sensor Bridge	Available	–	–
RD	Infrared Remote Tx/Rx Reference Designs	Available	Available	Available
RD	Graphics Acceleration	Available	–	–
Demo	Human Face Detection AI Demo	Available	–	–
Demo	Human Presence Detection AI Demo	Available	–	–
Demo	Single Wire Signal Aggregation Demonstration	Available	–	–
RD	Machine Learning / On-device AI	Available	–	–
RD	LCD Controller - WISHBONE Compatible	–	–	Available
Demo	Hand Gesture Detection	Available	–	–
RD	IrDA Fast Receiver			Available
RD	Sensor Data Buffer Reference Design	Available	–	–
RD	Touch Screen Controller	–	–	Available
RD	Capacitive Touch Sense Controller	–	–	Available
RD	Barcode Emulation	–	Available	Available
RD	IrDA Fast Transmitter	–		Available
RD	Pedometer Reference Design	Available	Available	
RD	BlackIce-II by myStorm	–	–	Available
RD	Doppler by Dadamachines	Available	–	–
RD	iCE Bling by Electronut Labs	Available	–	–

7. Frequent Asked Questions (FAQs)

For FAQs related to iCE40 devices and iCEcube2 topics, visit Lattice Semiconductor Knowledge Base for access to Lattice FAQs database.

Navigate to the search bar and type ***iCE40 + iCEcube2*** to list down all the FAQ topics related to iCE40 devices and iCEcube2.

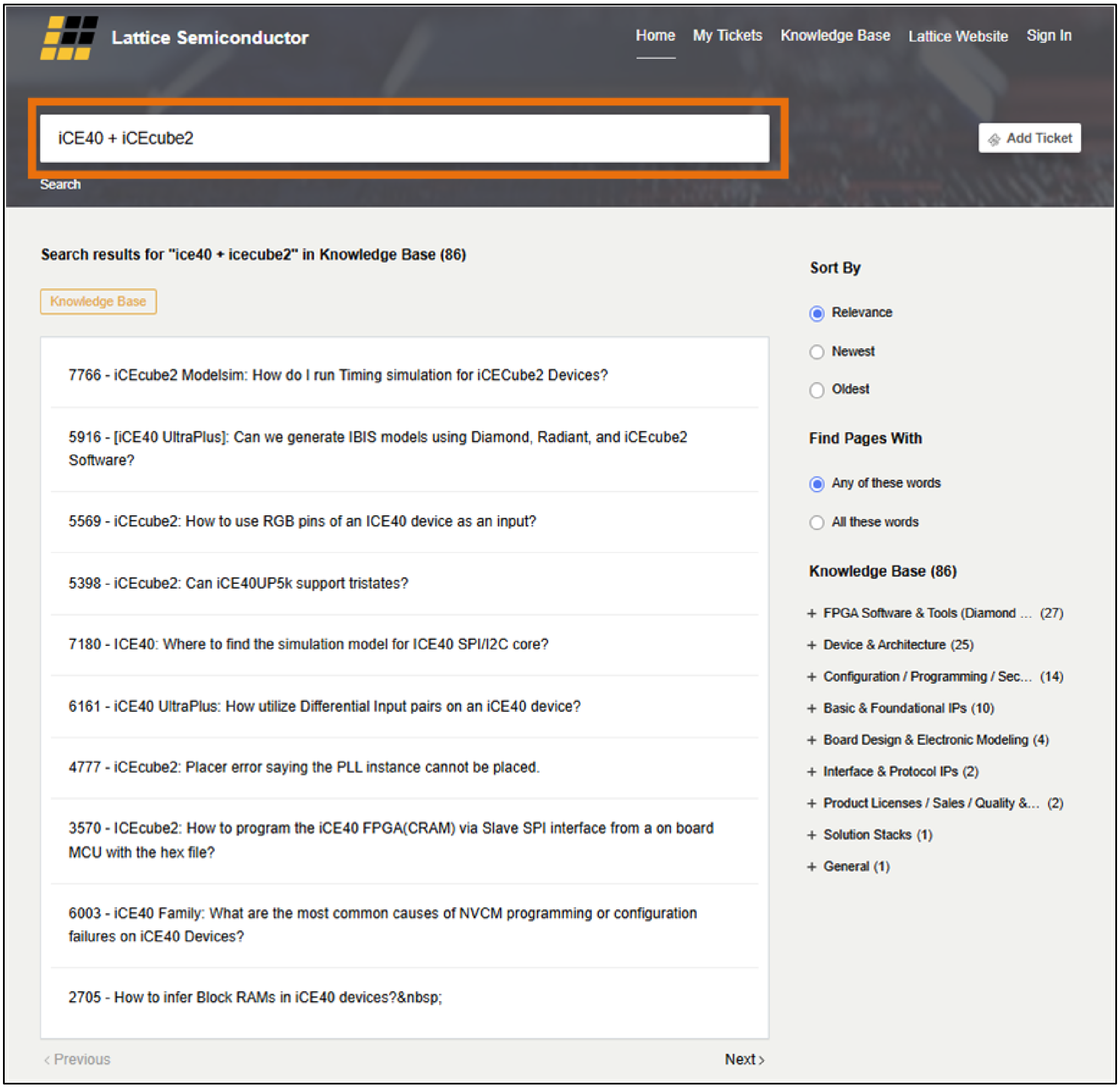
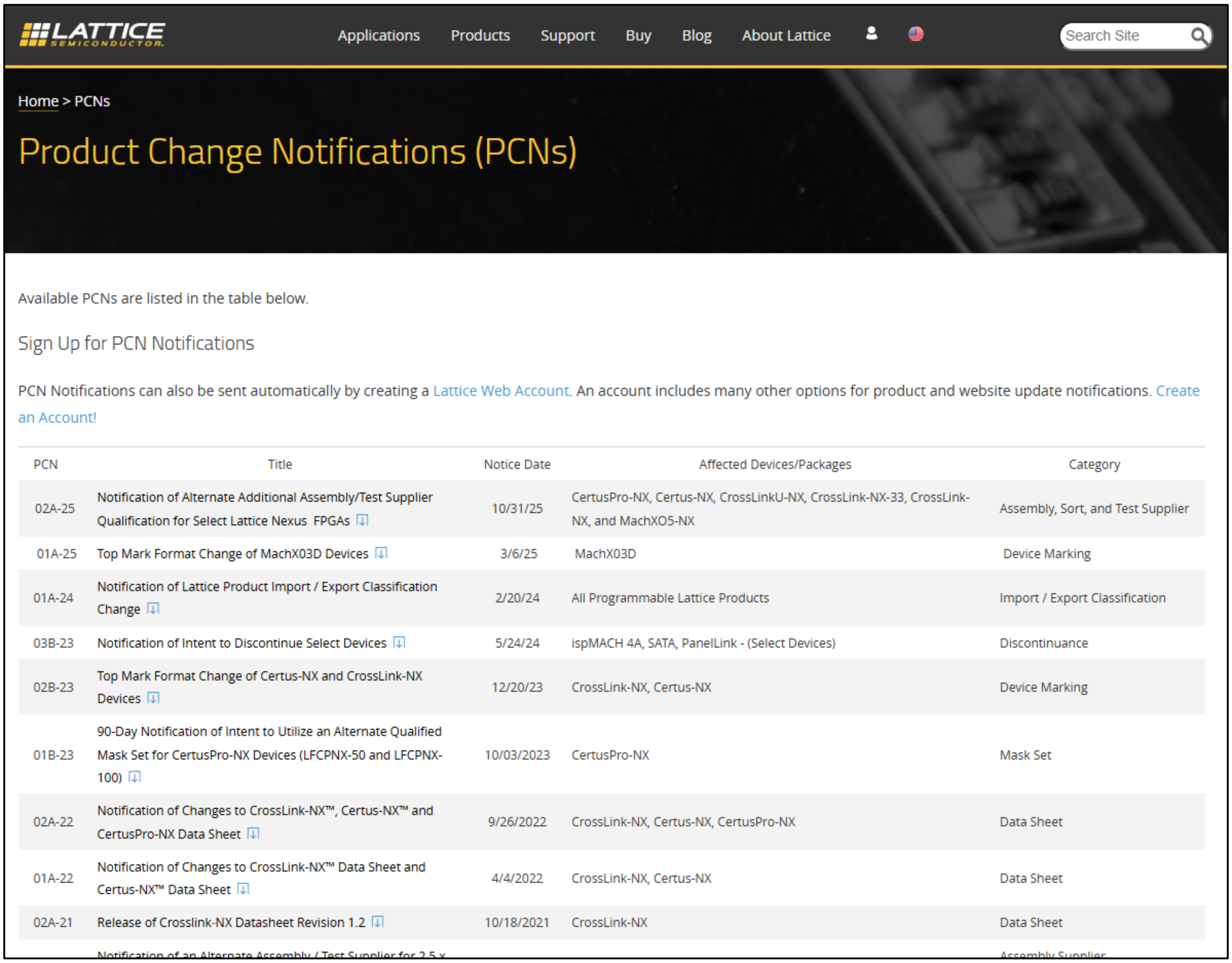


Figure 7.1. Lattice Semiconductor Knowledge Base

8. Product Change Notifications (PCNs)

You can sign up for PCN at Product Change Notifications web page. After signing up, you will receive formal notifications of any new product changes.



Applications Products Support Buy Blog About Lattice

Search Site

Home > PCNs

Product Change Notifications (PCNs)

Available PCNs are listed in the table below.

Sign Up for PCN Notifications

PCN Notifications can also be sent automatically by creating a Lattice Web Account. An account includes many other options for product and website update notifications. Create an Account!

PCN	Title	Notice Date	Affected Devices/Packages	Category
02A-25	Notification of Alternate Additional Assembly/Test Supplier Qualification for Select Lattice Nexus FPGAs	10/31/25	CertusPro-NX, Certus-NX, CrossLinkU-NX, CrossLink-NX-33, CrossLink-NX, and MachXO5-NX	Assembly, Sort, and Test Supplier
01A-25	Top Mark Format Change of MachX03D Devices	3/6/25	MachX03D	Device Marking
01A-24	Notification of Lattice Product Import / Export Classification Change	2/20/24	All Programmable Lattice Products	Import / Export Classification
03B-23	Notification of Intent to Discontinue Select Devices	5/24/24	ispMACH 4A, SATA, PanelLink - (Select Devices)	Discontinuance
02B-23	Top Mark Format Change of Certus-NX and CrossLink-NX Devices	12/20/23	CrossLink-NX, Certus-NX	Device Marking
01B-23	90-Day Notification of Intent to Utilize an Alternate Qualified Mask Set for CertusPro-NX Devices (LFCPNX-50 and LFCPNX-100)	10/03/2023	CertusPro-NX	Mask Set
02A-22	Notification of Changes to CrossLink-NX™, Certus-NX™ and CertusPro-NX Data Sheet	9/26/2022	CrossLink-NX, Certus-NX, CertusPro-NX	Data Sheet
01A-22	Notification of Changes to CrossLink-NX™ Data Sheet and Certus-NX™ Data Sheet	4/4/2022	CrossLink-NX, Certus-NX	Data Sheet
02A-21	Release of Crosslink-NX Datasheet Revision 1.2	10/18/2021	CrossLink-NX	Data Sheet

Figure 8.1. Product Change Notifications (PCNs) Web Page

References

- iCE40 Oscillator User Guide (FPGA-TN-02008)
- iCE40 sysCLOCK PLL Technical Note (FPGA-TN-02052)
- iCE40 UltraPlus Family Data Sheet (FPGA-DS-02008)
- iCE40 Ultra Family Data Sheet (FPGA-DS-02028)
- iCE40 UltraLite Family Data Sheet(FPGA-DS-02027)
- iCE40 LP/HX Family Data Sheet (FPGA-DS-02029)
- Migrating iCEcube2 iCE40 UltraPlus Designs to Lattice Radiant Software
- Lattice Radiant Software User Guide
- Lattice Radiant FPGA design software
- Lattice Solutions IP Cores web page
- Lattice Propel Design Environment web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, January 2026

Section	Change Summary
All	Initial release.