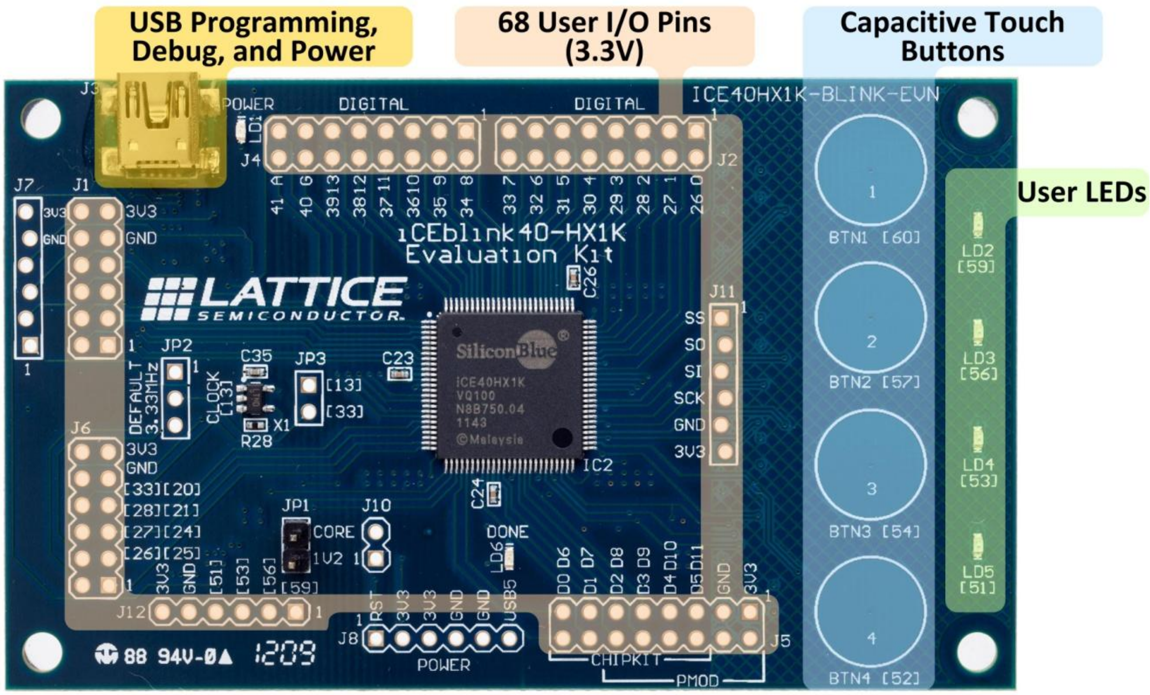
iCEblink40™ iCE40HX1K Evaluation Kit User Guide

- **High-performance, low-power iCE40HX1K mobileFPGA**
- **USB programming, debugging, virtual I/O functions, and power supply**
- **Four user LEDs**
- **Four capacitive-touch buttons**
- **3.3 MHz clock source**
- **1Mbit SPI serial configuration PROM**
- **Supported by Lattice iCEcube2 design software**
- **68 LVCMOS/LVTTL (3.3V) digital I/O connections on 0.1" through-hole connections**
- **Supports third-party I/O expansion boards and modules, including 3.3V Arduino Shield boards (requires additional sockets, not supplied)**

Figure 1: **iCEblink40 HX1K Evaluation Board and Major Hardware Features**



Introduction

Thank you for choosing the Lattice Semiconductor iCEblink40™ HX1K Evaluation Kit.

This guide describes how to begin using the iCEblink40 Evaluation Kit, an easy-to-use platform for rapidly prototyping designs using iCE40 mobileFPGAs.

Software Requirements

Before using the iCEblink40 board, please be sure to download and install iCEcube2 Release 2011.12 or later. This and later versions include the programming software for the iCEblink40 board. Currently, the programming software is only available for the Windows operating system.

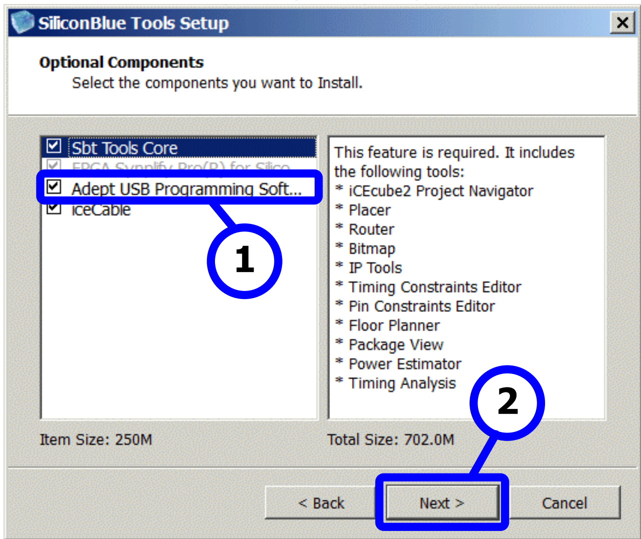
http://www.latticesemi.com/products/designsoftware/icecube2/downloads.cfm

During the installation process, be sure to install the Adept USB Programming Software, as shown in Figure 2.

1. Make sure that **`Adept USB Programming Software`** is checked. This is the default setting.

2. Click **Next**.

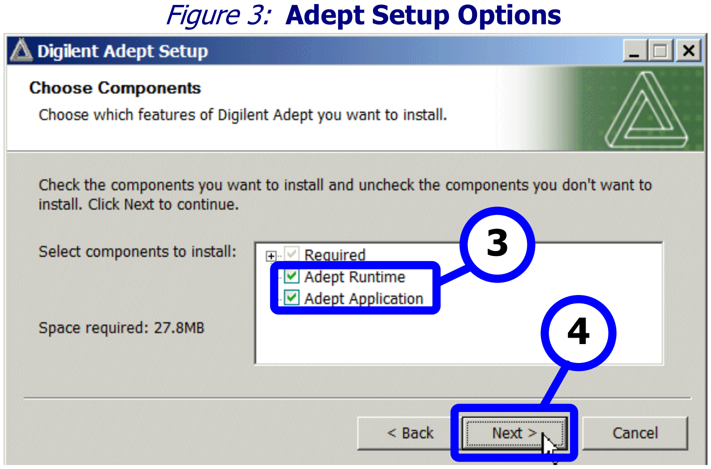
Figure 2: **Select the Adept Programming Software for Installation**



A few steps later, select the installation for the Adept programming software, as shown in Figure 3.

3. Make sure that both the **Adept Runtime** and **Adept Application** options are checked, which are the default settings.
4. Click **Next**.

Figure 3: **Adept Setup Options**



Connecting to the iCEblink40 Evaluation Board

Before connecting the iCEblink40 board, be sure to download and install a supported version of the iCEcube2 software.

Connect the iCEblink40 evaluation board to your PC using the USB cable provided. The USB connector on the board is labeled with reference designator J3 and is located in the upper left corner. Once connected, the red power-good LED (LD1) adjacent to the USB connector illuminates. See Figure 4 to locate the power-good LED.

Power and Configuration Status LEDs

The iCEblink40 evaluation board has two status LEDs, as shown in Figure 4. These two status LEDs indicate the current status of the iCEblink40 board, as listed in Table 1. The red LED, LD1, located near the USB connector indicates if the USB power supply, the 3.3V supply, and the 1.2V supply are within the specified ranges.

The yellow LED, LD6, located below the mobileFPGA indicates whether the mobileFPGA is configured properly. This LED lights up when the FPGA is correctly loaded with a valid bitstream.

Figure 4: **iCEblink40 Status LEDs**

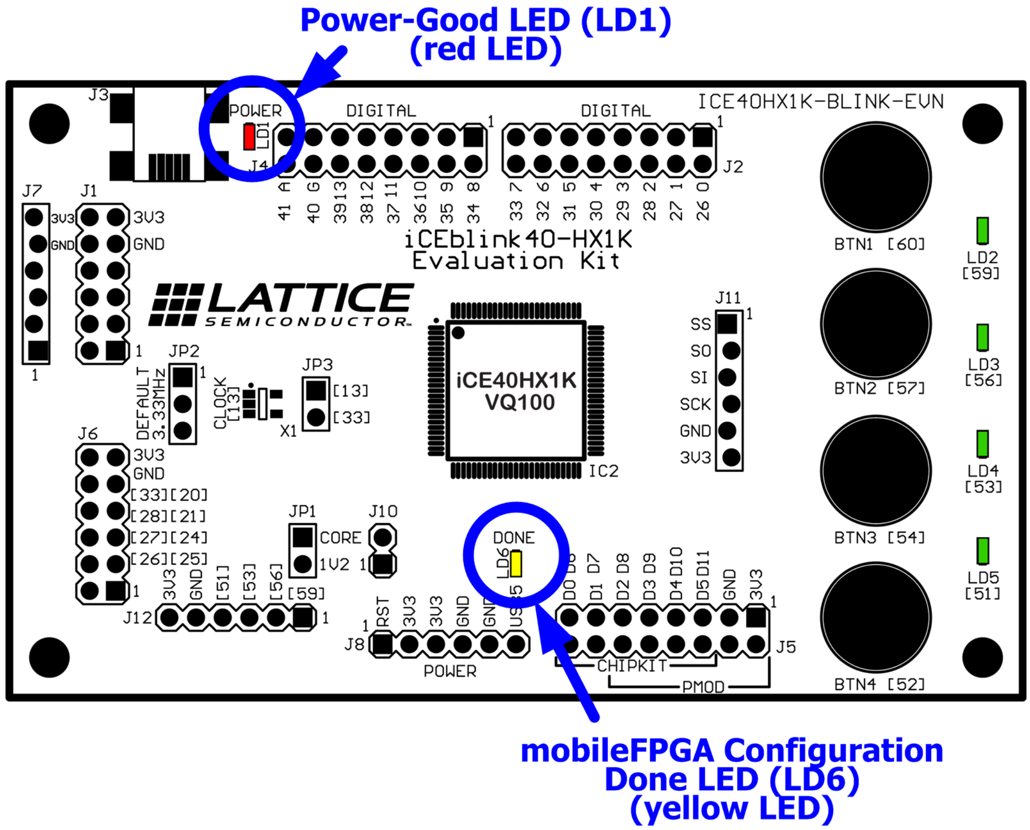


Table 1: **iCEblink40 Status LEDs and Their Meaning**

Power-Good LED (LD1)	Configuration DONE LED (LD6)	Meaning
On	**On**	The board is powered, the mobileFPGA successfully configured and the mobileFPGA application is operating.
Off	**Off**	Board is unpowered. Connect the board to a computer USB port, a powered hub, or a USB-based wall plug. If board is plugged in and previously operating, indicates that an SPI Flash programming operation is in progress
On	**Off**	The board is powered but the mobileFPGA is not yet configured. ACTION: Program the onboard SPI Flash PROM with a valid mobileFPGA configuration bitstream.
Off	**On**	ERROR: The board is powered but there is a problem with the USB power supply or with the on-board regulator.

Pre-programmed Demonstration Design

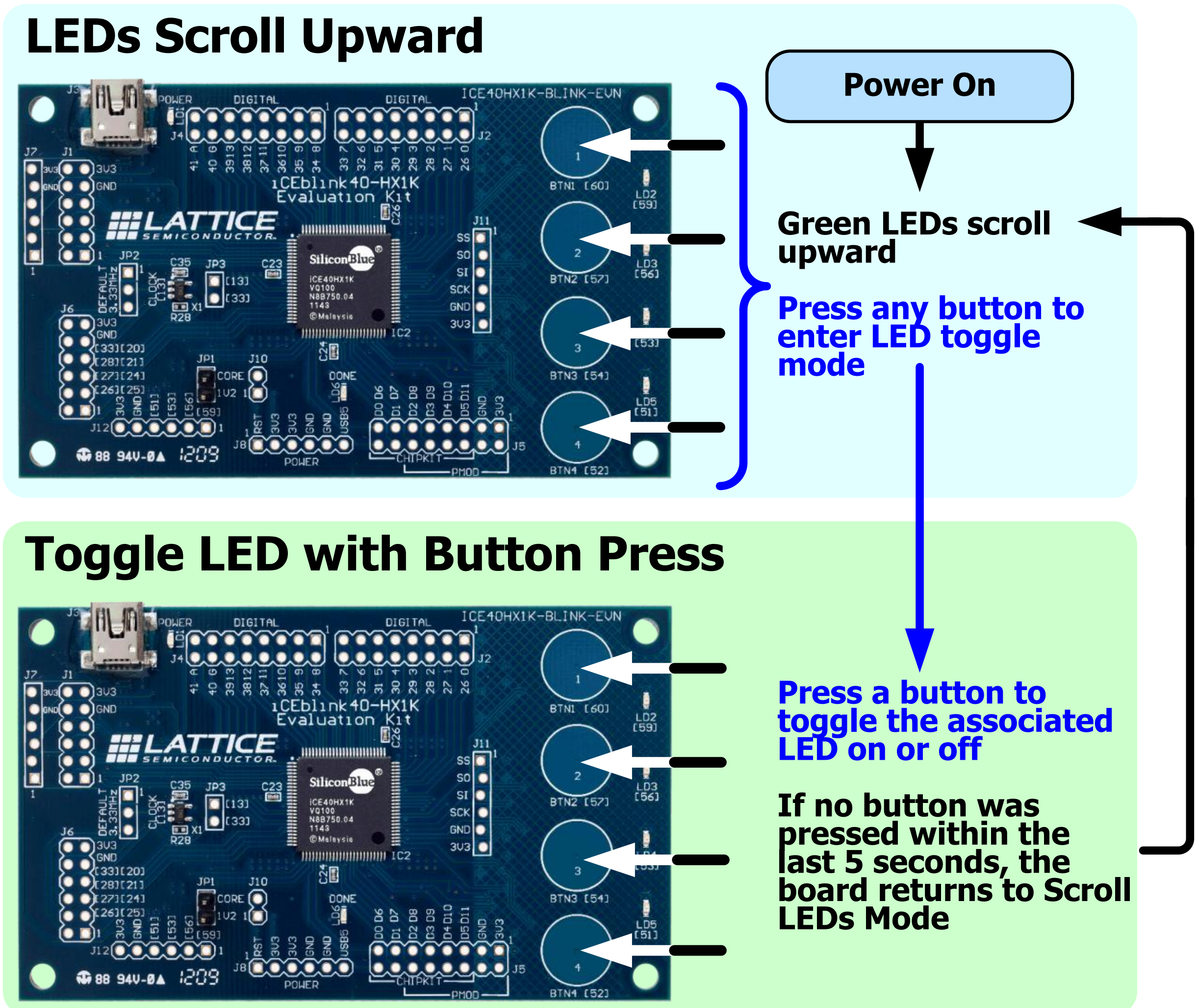
The iCEblink40 board comes preprogrammed with a demonstration application. The application supports two interfaces.

1. Control the LEDs from the four capacitive touch buttons on the board itself.
2. Control the LEDs and other internal logic using the USB-based I/O expansion interface.

Operating the Capacitive-Touch Buttons

Upon power up, the green LEDs on the board scroll in an upward pattern, as described in Figure 5. Pressing any of the capacitive touch buttons stops the LEDs from scrolling and places the board in a different operating mode.

Figure 5: **Preprogrammed Demonstration Design**



In the second operating mode, toggle individual LEDs on and off by pressing the associated capacitive touch button.

If no button was pressed during the last five seconds, the board returns to scrolling the LEDs.

> The demonstration application is available for download from the Lattice Semiconductor web site at …
> `www.latticesemi.com/iceblink40-hx1k`

Virtual I/O Expansion Debugging Interface

The iCEblink40 board is powered and programmed via the USB interface. Additionally, the USB interface also provides a convenient means to monitor and control logic inside the mobileFPGA, as shown in Figure 6. The USB controller drives a byte-wide parallel port expander implemented within the mobileFPGA, controlled by software running on the PC. The Digilent ADEPT2 I/O Expansion screen, shown in Figure 7, provides a mix of virtual switches, pushbuttons, LEDs, light bars, and 32-bit input and outputs.

Figure 6: **iCEblink40 Board Supports Virtual I/O Connections over USB**

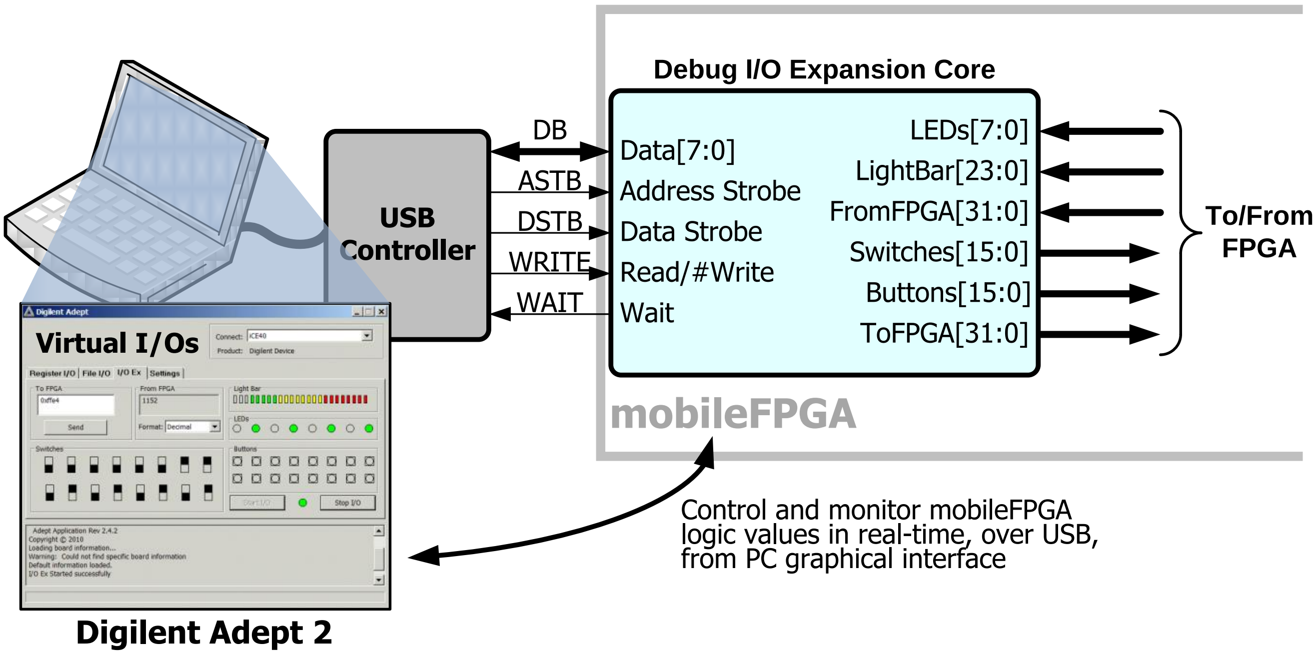
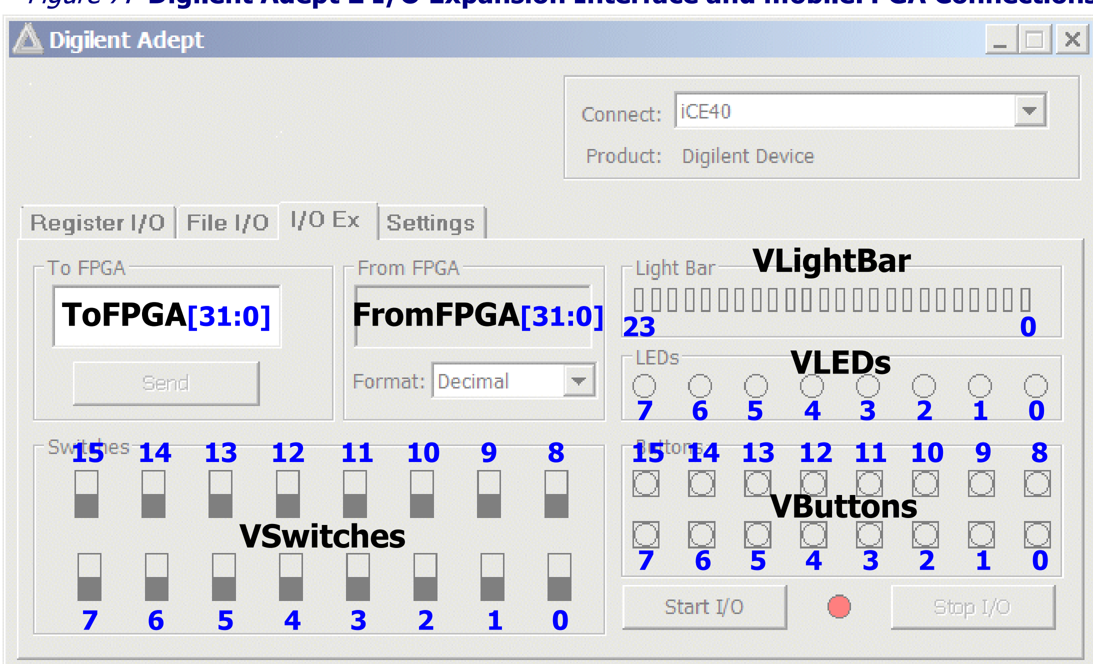


Figure 7: **Digilent Adept 2 I/O Expansion Interface and mobileFPGA Connections**



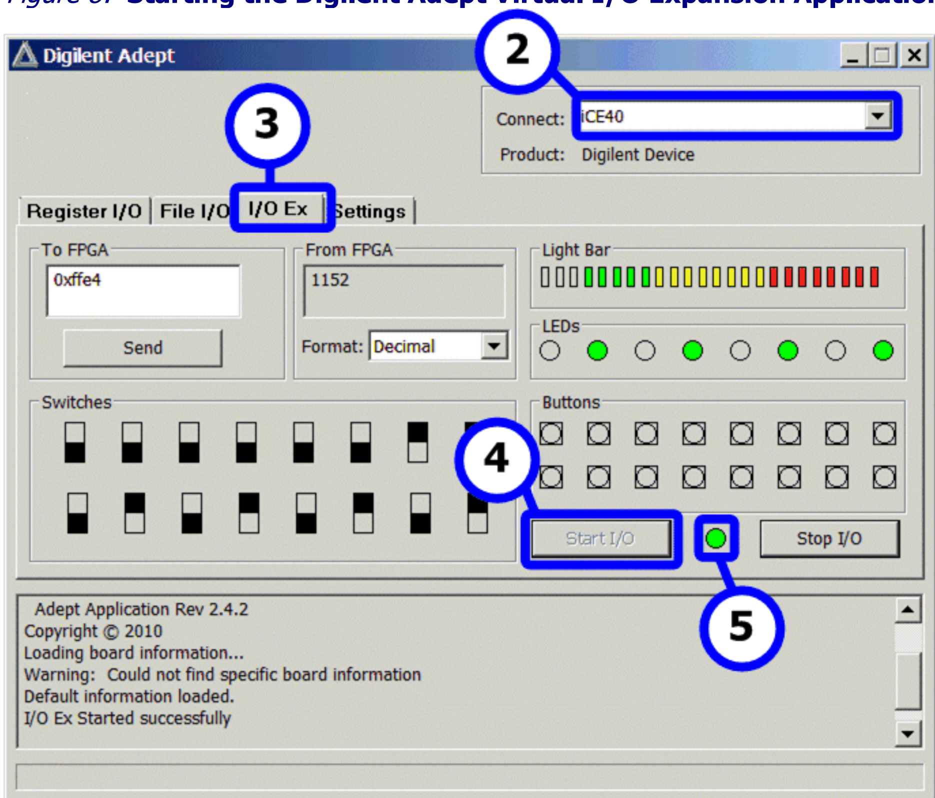
Using the Virtual I/O in the Demo Application

By default, the Virtual I/Os are disconnected and the USB controller's I/O connections to the FPGA are high-impedance (Hi-Z).

To connect the Virtual I/O, perform the following steps outlined in Figure 8.

1. From the Windows Start menu, select **Start → All Program → Digilent → Adept → Adept**
2. Ensure that the Adept interface connects to the **iCE40**.
3. Click the **I/O Ex** tab.
4. Click **Start I/O**. Remember, the associated I/O Expander design must be part of the compiled FPGA design before the Virtual I/Os work.
5. If the virtual I/O expansion design is functioning correctly, the green virtual status LED will turn from red to green.

Figure 8: **Starting the Digilent Adept Virtual I/O Expansion Application**



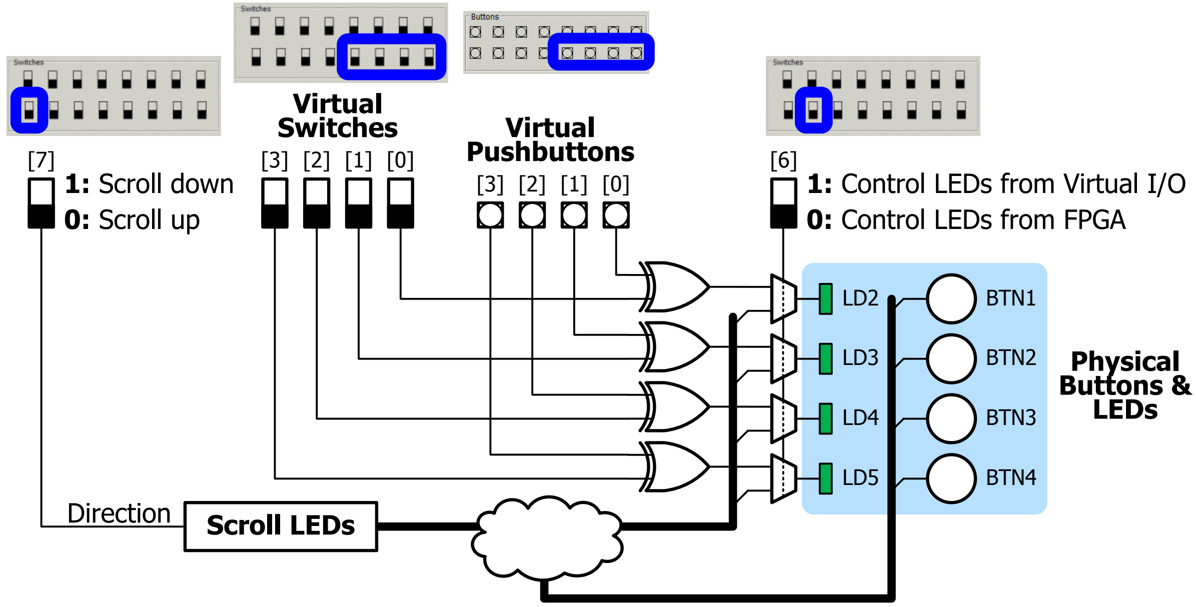
To disconnect the virtual I/O interface, simply click the **Stop I/O** button in the graphical interface.

Controlling the Physical LEDs from Virtual I/Os in the Demonstration Design

When active, the virtual I/Os optionally control the physical LEDs on the board, as shown in Figure 9. For example, with the virtual I/Os active, change the position of virtual switch [7] (the bottom left switch in the graphical interface). Note how the physical LEDs on the board change direction.

Change virtual switch [6] to the up position. Now, the physical LEDs are controlled by the virtual switches [3:0] and virtual pushbuttons [3:0]. The values of the virtual switches are XORed together inside the FPGA. The virtual slide switches set a specific value for the physical LEDs. The pushbutton momentarily inverts the value while the virtual pushbutton is pressed in the graphical interface.

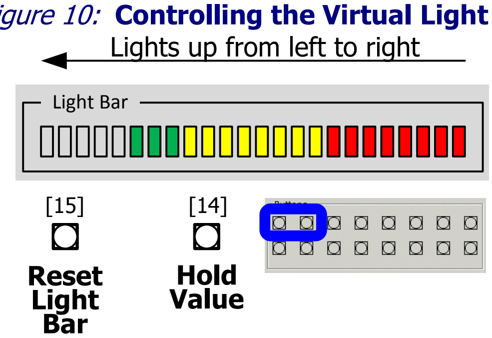
Figure 9: **Controlling the Physical LEDs from Virtual I/O**



Controlling the Virtual Light Bar in the Demonstration Design

When the virtual I/Os are active, the virtual light bar lights up from left to right, controlled by logic inside the FPGA. The virtual pushbuttons [15] and [14] control the light bar. Pushbutton [15] resets the light bar, clearing all the lights. Pushbutton [14] forces the light bar to hold its current value.

Figure 10: **Controlling the Virtual Light Bar**



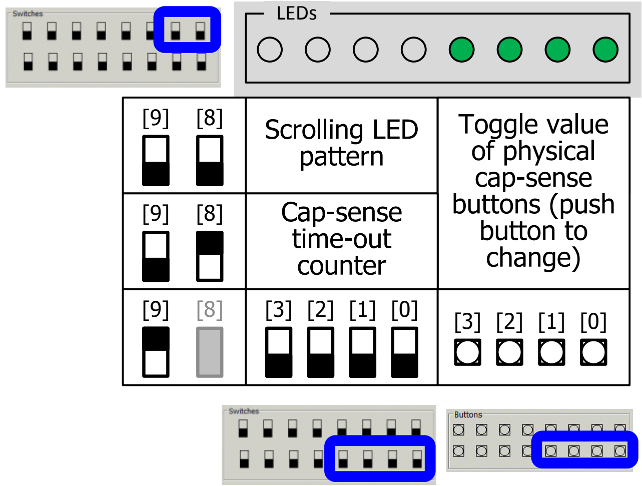
Controlling the Virtual LEDs in the Demonstration Design

The virtual I/O interface includes eight, round, green LEDs, as shown in Figure 11. The values displayed on these virtual LEDs depends on the settings of virtual switches [9] and [8].

The eight LEDs are separated into left and right halves. When both virtual switches [9] and [8] are Low—the down position—the left LEDs echo the scrolling pattern of the LEDs, regardless if a physical cap-sense button was pressed. Use virtual switch [7] to reverse the direction of these LEDs. The right-most LEDs show the current toggle status of the four physical cap-sense buttons.

Changing virtual switch [8] to High—the up position—the four left-most LEDs then show the current value of the time-out counter than marks the five seconds after pressing a cap-sense button. Press a cap-sense button to reset the timer and note that the toggle status of the physical button changes on the right-most LEDs. The timer resets each time a physical button is pressed. Wait five seconds and the physical LEDs change back to the scrolling pattern.

Figure 11: **Controlling the Virtual LEDs**

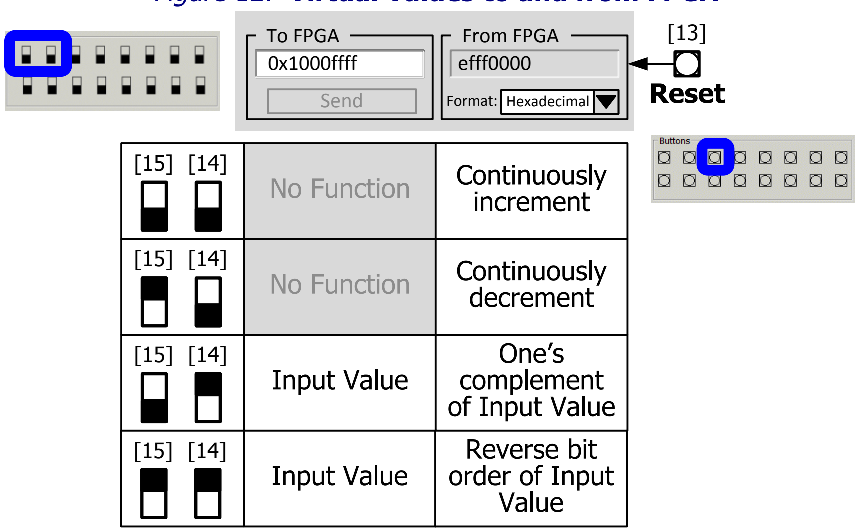


When virtual switch [9] is High—in the up position—then the left LEDs are controlled by virtual switches [3:0] and the right LEDs are controlled by virtual pushbuttons [3:0].

Virtual Values to and from FPGA in the Demonstration Design

The virtual I/O interface also includes a 32-bit value from the FPGA logic and a 32-bit value to the FPGA logic, as shown in Figure 12. Two virtual switches, [14] and [15], control the behavior in of the virtual 32-bit values in the demonstration design.

Figure 12: **Virtual Values to and from FPGA**



Clocking Resources

The iCEblink40 board includes a Linear Technology [LT1799](#) oscillator (X1 on the board and in the schematic) to generate a 3.33 MHz clock.

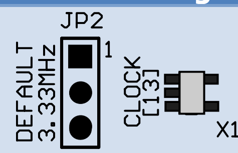
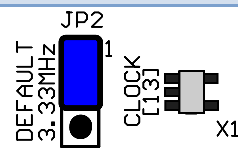
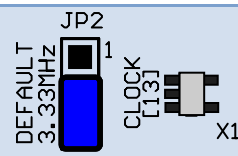
FPGA Input

The output from the LT1799 oscillator feeds pin 13 of the iCE40HX1K mobileFPGA. FPGA pin 13 is also the global buffer input GBIN7.

Supporting Other Frequencies

On the iCEblink40 board, the LT1799 produces a 3.3 MHz clock output by default. Other frequencies are possible via simple modifications of the board using the 1x3 connections on JP2, as listed in Table 2.

Table 2: **Selecting Other Oscillator Frequencies Using Jumper JP2**

Clock Frequency	JP2 Setting	Jumper Position
3.33 MHz (default)		None
333 kHz		Upper Position
33.3 MHz		Lower Position

User LEDs

The iCEblink40 iCE40HX1K evaluation kit board includes four green user LEDs, located along the left side of the board, as shown in Figure 1.

Operation

To light a user LED, drive the associated mobileFPGA pin High, as shown in Table 3. To darken the LED, drive the associated mobileFPGA pin Low.

Table 3: **User LED Operation**

Operation	FPGA Action
Light LED	Drive High (1)
Darken LED	Drive Low (0)

The LEDs may appear to glow slightly before the FPGA is configured or if the mobileFPGA pin is unused. This is because the mobileFPGA I/Os have a soft pull-up resistor which may provide just enough current for the LED to glow dimly. To completely turn off an LED, drive it Low.

FPGA Connections

The FPGA drives the user LEDs using the mobileFPGA pins listed in Table 4. These same signals also connect to the J12 header located in the lower left corner.

Table 4: **User LED Connections**

Designator	Location	mobileFPGA Pin	Header Connections
LD2	 LD2 [59]	59	J12.1
LD3	 LD3 [56]	56	J12.2
LD4	 LD4 [53]	53	J12.3
LD5	 LD5 [51]	51	J12.4

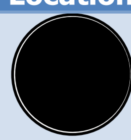
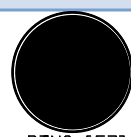
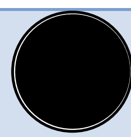
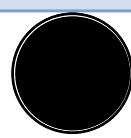
Capacitive Touch Buttons

The iCEblink40 iCE40HX1K evaluation kit board has four capacitive-touch buttons, located toward the left side of the board, as shown in Figure 1. These buttons have dedicated connections only to the FPGA. These signals go nowhere else on the board and are not available on any of the breakout headers.

FPGA Connections

Table 5 lists the four capacitive touch buttons on the iCEblink40 board and the associated mobileFPGA pins.

Table 5: **Capacitive Touch Buttons**

Designator	Location	mobileFPGA Pin
BTN1	 BTN1 [60]	60
BTN2	 BTN2 [57]	57
BTN3	 BTN3 [54]	54
BTN4	 BTN4 [52]	52

Operation

Figure 13 shows the circuit used for each capacitive-touch button. Each button is attached to one I/O pin on the mobileFPGA. Each signal line includes a 100 kΩ pull-up resistor to 3.3V and a 100 pF capacitor down to ground.

Figure 13: **Example Capacitive Touch Button Circuit**

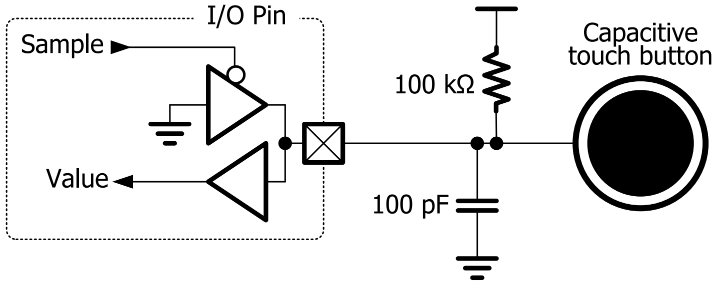
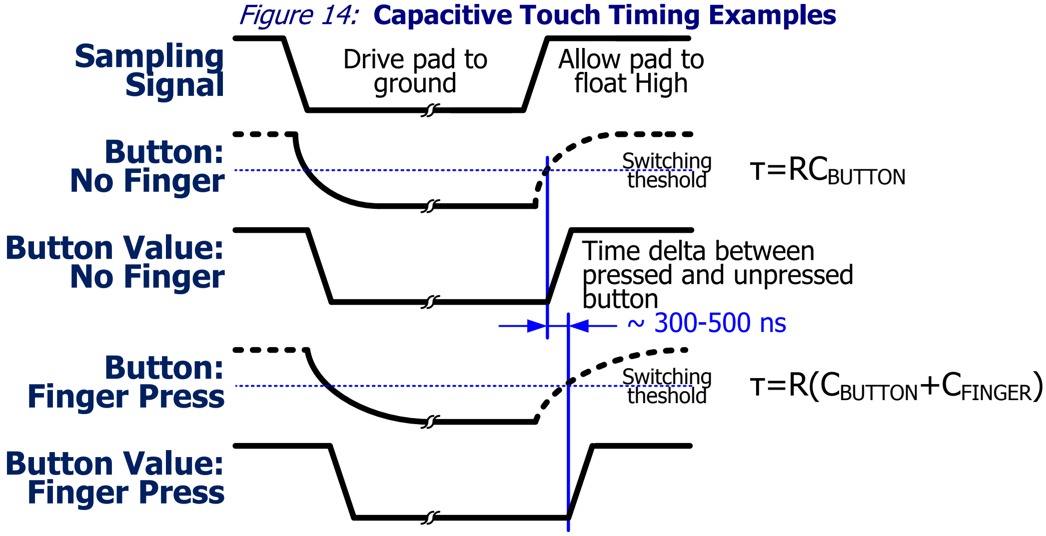


Figure 15 shows the general overall flowchart for the demonstration design to read the value on a capacitive touch button.

The sampling signal drives the voltage on the capacitive-touch button to ground in order to bleed of any residual charge as shown in Figure 14. After a period of time, depending on the button sample frequency, the button is allowed to float High.

Once the mobileFPGA output goes to Hi-Z (high-impedance, floating, three-state), the 100kΩ pull-up resistor to 3.3V charges the 100 pF capacitor. After about an RC time constant (τ or tau), the voltage on the pad exceeds the input switching threshold of the mobileFPGA. A finger pressed against the capacitive-touch button adds about another 5 pF of capacitance, increasing the RC constant and delaying the Low-to-High transition for a pressed button.

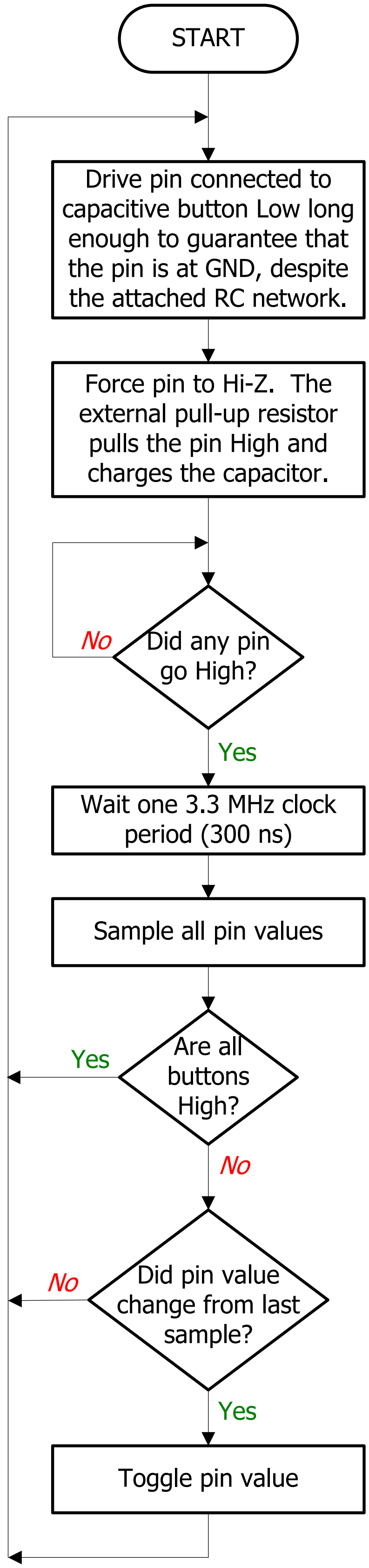
Figure 14: **Capacitive Touch Timing Examples**



The switching time difference between a unpressed and one or more pressed buttons is roughly 300 to 500 ns. Using the 3.33 MHz input, this amounts to a one clock delay difference between an unpressed and pressed buttons.

The simple circuit used on the iCEblink40 board detects simultaneous button presses on up to three of the capacitive-touch buttons. Pressing all four buttons is the same as pressing no buttons.

Figure 15: **iCEblink40 Demo Application Capacitive Touch Button Flowchart**



User I/O Connections

Figure 16 shows the location of the 3.3V-compatible digital I/O connections on the iCEblink40 board. Each connection shows the pin number of the mobileFPGA I/O pin that attaches to the connection. Likewise, Table 6 lists the various I/O headers and their designed usage.

Figure 16: **Location of the 3.3V Digital I/O Connections and the mobileFPGA Pin Number**

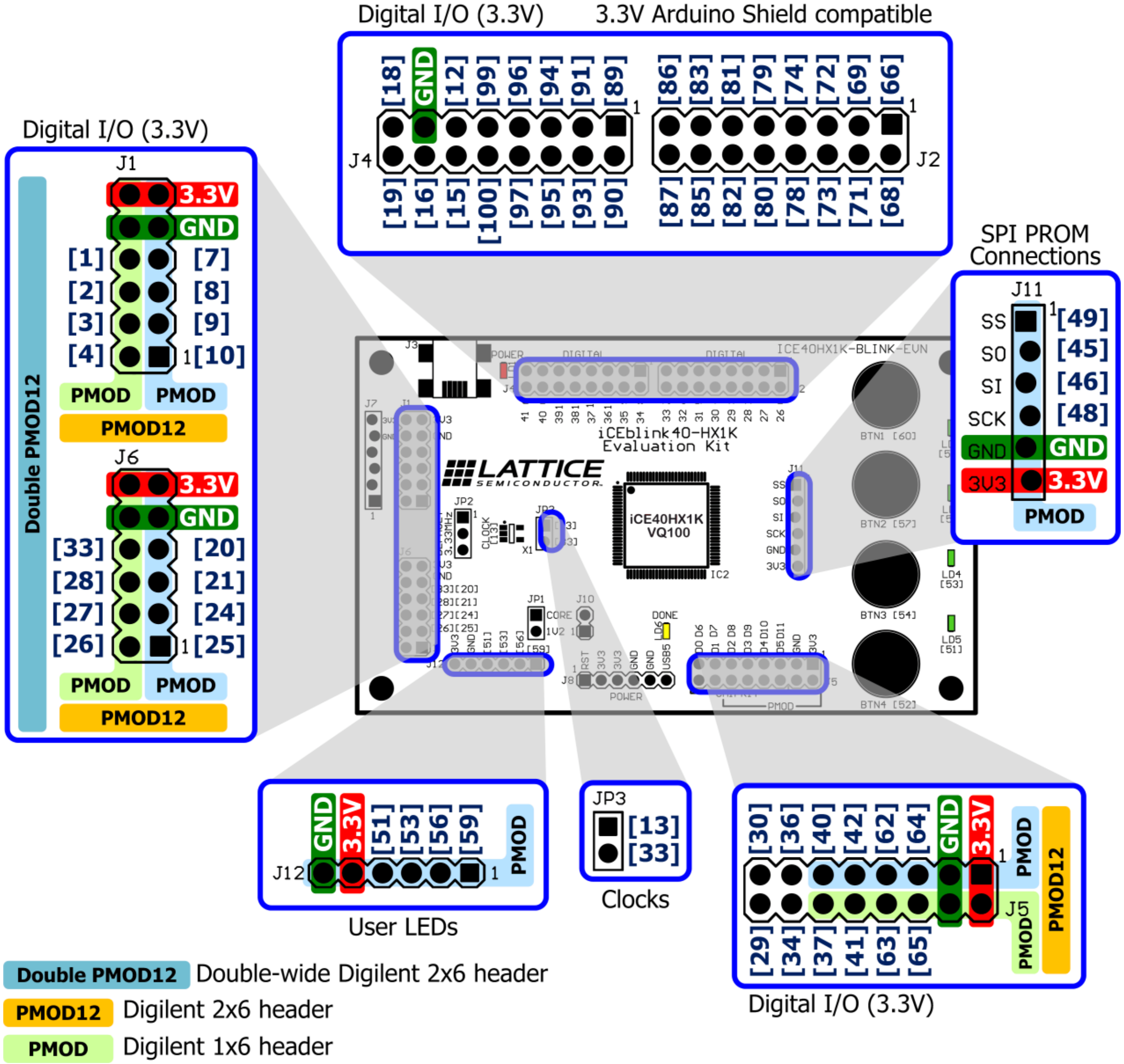


Table 6: **Digital I/O Headers and Their Functions**

I/O Header Group	Header Type	Location	Function
J2	2x8 0.1" centers	Top edge, middle	3.3V digital I/O. Compatible with 3.3V Arduino Shield boards.
J4	2x8 0.1" centers	Top edge, left side	3.3V digital I/O. Compatible with 3.3V Arduino Shield boards.
J1	2x6 0.1 centers	Left edge, top	3.3V digital I/O. 3.3V digital I/O. Compatible with Digilent 1x6 and 2x6 PMod modules. Also supports double PMod12 modules when used with header J6.
J6	2x6 0.1 centers	Left edge bottom	3.3V digital I/O. Compatible with Digilent 1x6 and 2x6 PMod modules. Also supports double PMod12 modules when used with header J1.
J7	1x6 0.1 centers offset	Left edge, top	Production programming of USB controller
J11	1x6 0.1 centers offset	Middle, toward left	3.3V digital I/O. Connections between the mobile FPGA and the SPI PROM. Compatible with Digilent 1x6 PMod modules.
J5	2x8 0.1" centers	Bottom edge, right side	3.3V digital I/O. Portions compatible with Digilent 1x6 and 2x6 PMod modules. Portions also compatible with 3.3V Arduino Shield boards.
JP3	1x2 0.1" centers	Middle, to left of mobileFPGA	3.3V digital I/O. Clock connections from the LTC1799 oscillator (GBIN7) and possible into GBIN2.
J12	1x6 0.1" centers	Bottom edge, left side	3.3V digital I/O. Connections to the user LED I/O. Compatible with Digilent 1x6 Pmod modules.

Supported Pmod Peripheral Modules

As shown in Figure 16, the iCEblink40 board supports a variety of Pmod peripheral modules for easy I/O expansion. Table 7 lists the 0.1" through-hole headers on the iCEblink40 board that support Pmod modules. Pmod modules come in a few different form factors and each Pmod header includes power and ground supplies. Figure 17 shows the how the different Pmod form factors interrelate. The easiest way to support a Pmod module is to add the appropriate female socket listed, or an equivalent. Straight-through or right-angle through-hole sockets are listed. Male headers are also possible solutions when using the interface cable provided with most Pmod modules.

Table 7: **Pmod Module Headers**

Header	Type	Female Socket (Manufacturer/Part Number)	
		Straight-through	Right-angle
J1, J12	2x6 header on 0.1" centers. Each is a Pmod12 header that supports two six-pin Pmod modules. Both together form a double-wide Pmod connection.	Sullins Connector Solutions PPPC062LFBN-RC	Sullins Connector Solutions PPPC062LJBN-RC
J5	2x8 header on 0.1" centers. The left side of header J5 forms a Pmod12 header, as shown in Figure 16. A 2x6 header similar to J1, J12 can also be used but must be mounted toward the right end of the holes as marked.	Sullins Connector Solutions PPPC082LFBN-RC	Sullins Connector Solutions PPPC082LJBN-RC

As shown in Figure 17, a Pmod module has six connections—four I/O plus power and ground. A Pmod12 module has 12 connections and the module is effectively two six-pin Pmod modules stacked together. Finally, a double-wide Pmod12 consists of two Pmod12 headers spaced apart. Most of the Pmod modules also include interface cables to allow easy connection to other header types.

Figure 17: **Pmod Module Types**

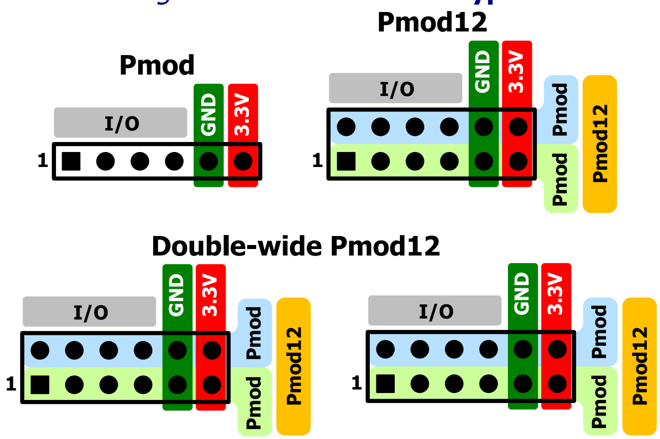
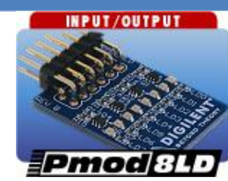


Table 8 provides a sampling of the currently available Pmod modules. Click the image to see more details on each module. For a complete list of Pmod peripheral modules, visit the Digilent web site.

http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,401&Cat=9

Table 8: **Sampling of Available Pmod Modules**

Displays				
 128x32 pixel OLED display	 16x2 character LCD, parallel interface	 16x2 character LCD, serial interface	 Two 7-segment LEDs	 Eight LEDs
Wireless Communication				
 802.11b/g/n WiFi interface	 Bluetooth interface	 IEEE 802.15 radio transceiver	 Wireless radio transceiver	
Networking/Communication				
 10/100 Ethernet NIC	 USB to UART interface	 RS-232 interface	 PS/2 keyboard/mouse connector	
Sensors				
 3-axis digital gyroscope	 3-axis accelerometer	 Infrared light sensor	 Thermometer/thermo stat	

Digital-to-Analog (D/A) and Analog-to-Digital (A/D) Conversion

Resistor Ladder D/A Converter

Four 8-bit D/A outputs

Two 12-bit D/A outputs

Two 12-bit A/D inputs

Four-channel, 12-bit A/D inputs

Audio

Microphone with digital interface

I^2S stereo audio output

Speaker/headphone amplifier

RCA audio jacks

Keypads, buttons, switches, and joysticks

16-button keypad

Four pushbuttons

Four slide switches

Rotary encoder/switch

Two-axis joystick

Connectors

RJ45 connector pair

BNC connectors

Wire terminal connectors

R/C servo control connectors

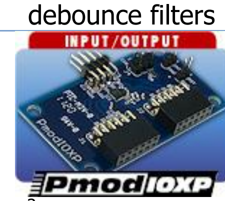
Four digital inputs with diode protection, debounce filters

H-bridge with feedback

H-bridge with feedback

Open-drain output

Open-collector output



I^2C I/O expansion module

Bread board and testpoint headers

Wire wrap/bread board

Pmod test header

Pmod12 test header

Memory

SD card slot

SPI Flash PROM

x1, x2, x4 SPI Flash PROM

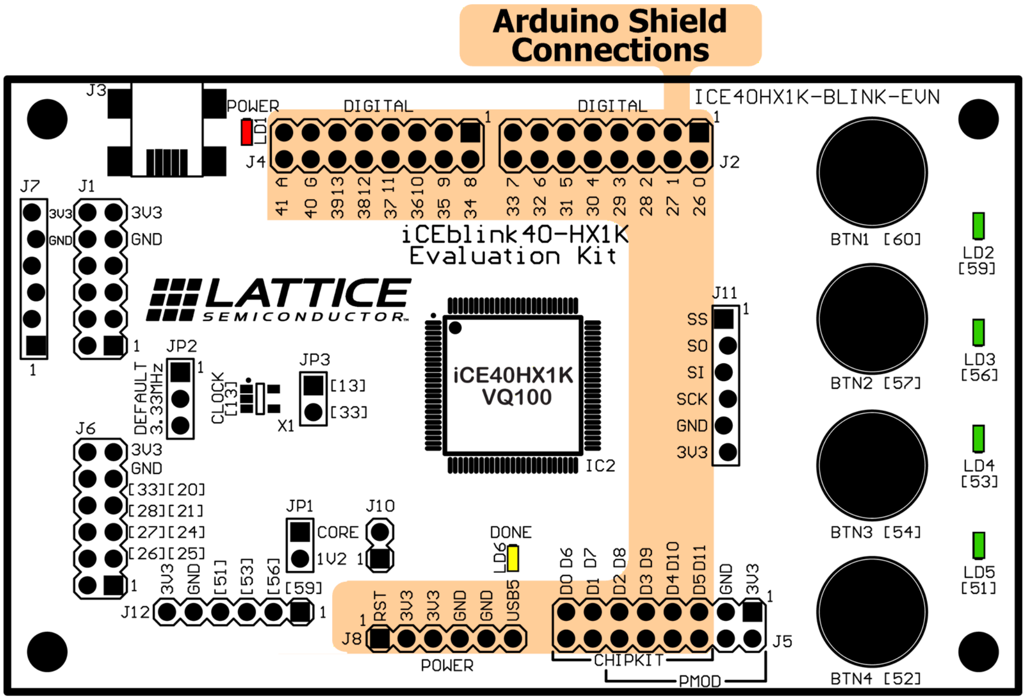
Miscellaneous

Real-time clock/calendar

Arduino Shield Board Support

The iCEblink40 board also mechanically and electrically supports select 3.3V Arduino Shield boards popular in the microcontroller development community. The Shield connections are located on headers J4, J2, J8 and a portion of J5 as shown in Figure 18. Headers jumper J4 and J2 are 3.3V digital I/O connections. Header J8 provides power connections to the Shield board. The left side of header J5 also provides 3.3V digital I/O but the 3.3V and GND connections do not connect to the Shield board.

Figure 18: **Arduino Shield Board Connections**



Required Header Sockets

To support Arduino Shield boards, the indicated headers must be loaded with female socket headers on 0.1" headers, as listed in Table 9.

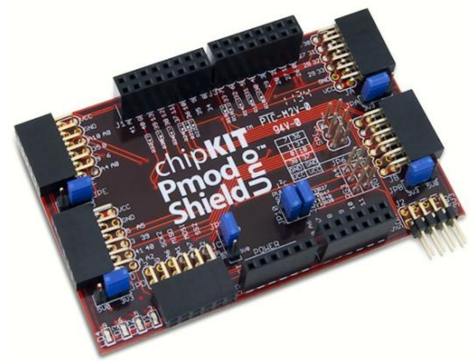
Table 9: **Sockets to Support Arduino Shield Boards**

Header(s)	Description	Quantity	Manufacturer/ Part Number
J2, J4, J5	2x8 female header socket on 0.1″ centers	3	Sullins Connector Solutions PPPC082LFBN-RC
J8	1x6 female header socket on 0.1″ centers	1	Sullins Connector Solutions PPPC061LFBN-RC

Tested Arduino Boards

Table 10 lists the Arduino Shield boards have been tested for basic compatibility. Other Arduino Shield boards may also be compatible.

Table 10: **Compatible Arduino Shield Boards**

Arduino Shield Boards	
 chipKITBasic I/O Shield	 chipKIT Pmod Shield-Uno

USB Interface

The iCEblink40 board is power by connecting the board to a computer USB port, a power USB hub, or a USB-based AC adapter, commonly used in consumer electronics. A typical USB port provides up to 500 mA at 5V, providing up to 2.5W of total power. The iCE40HX1K consume SIGNIFICANTLY LESS than power, even when operating at full performance. However, be careful when using the board to power off-board peripheral funds.

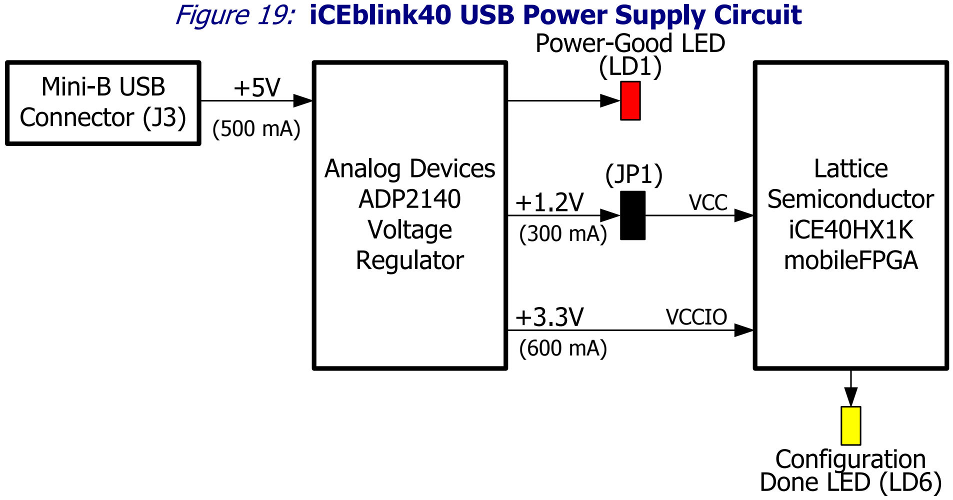
Connector

The USB connector to the board is located in the upper left corner, labeled J3. The board connects using a standard USB cable with a male mini-B connector.

Power Supply

Figure 19 shows the iCEblink40 power supply circuit that derives power from the USB mini-B connector (J3). The USB connector provides up to 500 mA at +5V DC. An Analog Devices ADP2140 regulator generates +1.2V for the mobileFPGA core VCC and +3.3V for all I/O connections. The regulator also indicates when power is good and lights up the red power-good LED (LD1).

Figure 19: **iCEblink40 USB Power Supply Circuit**



Jumper JP1 provides a convenient location from which to measure core power to the mobileFPGA.

SPI Flash Programming

The USB interface also provides Flash programming for the on-board SPI PROM, as described in "Programming the iCEblink40 Board" on page 19.

Digilent Parallel Port (DPP)

The Digilent Parallel Port (DPP) interface is used for virtual I/O and debugging using a USB connection to the board from a Windows PC. See "Virtual I/O Expansion Debugging Interface" on page 5 for additional information.

1Mbit SPI Configuration PROM

The configuration bitstream for the iCE40 mobileFPGA is stored in an M25P10A 1Mbit SPI serial Flash PROM. The PROM is large enough to hold two configuration images and supports the iCE40 WarmBoot feature, if so enabled within the FPGA application. The PROM is physically located on the back-side of the board.

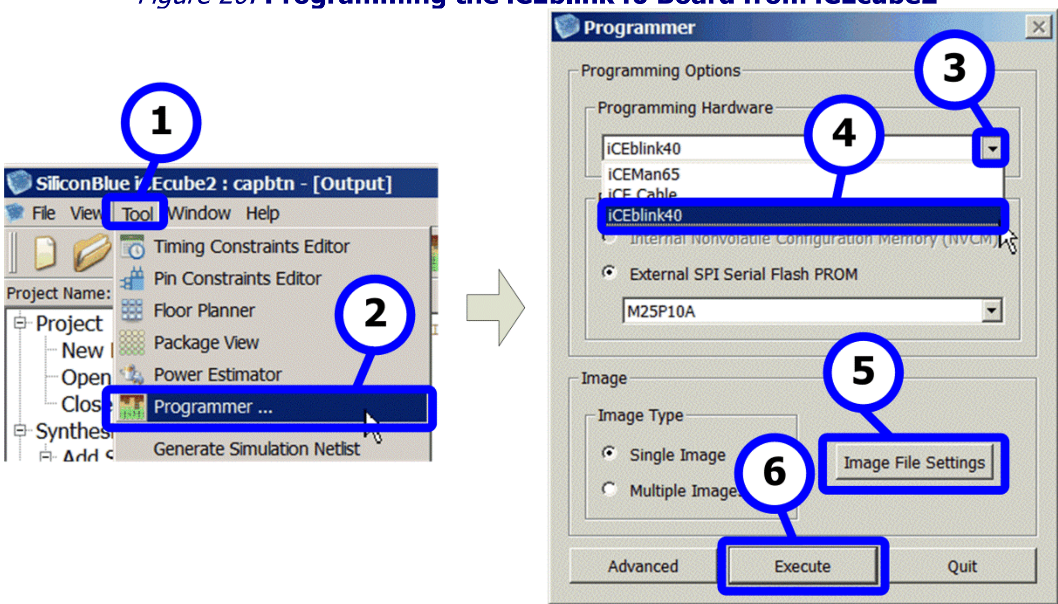
Programming the iCEblink40 Board

The iCEblink40 board includes on-board USB-based programming support either from the Lattice Semiconductor iCEcube2 software or using a command from a console window or DOS box.

From iCEcube2

Figure 20 shows the commend sequence for programming the SPI Flash PROM on the iCEblink40 board using the iCEcube2 development software.

Figure 20: **Programming the iCEblink40 Board from iCEcube2**



1. Select **Tool** from the iCEcube2 menu bar …
2. … followed by **Programmer**.
3. Click the dropdown button ( ) under **Programming Hardware**.
4. Select **iCEblink40**.
5. The bitstream file should already be set appropriately based on the iCEcube2 project settings. If not, click **Image Files Settings** to select the configuration bitstream file.
6. Click `Execute` to program the iCEblink40 board.

If all is working correctly, the power-on LED and the configuration done LED will both go out momentarily as iCEcube2 programs the on-board SPI Flash PROM. After programming is complete, both LEDs should light up again and the mobileFPGA will execute the new configuration image.

From Command Line

The iCEblink40 programming software can also be executed from a console window or DOS box. To open a console window or DOS box, click the Start button and type **cmd** in the textbox immediate above the Start button.

Executable Location

After installation, the programming software executable is called **iceutil.exe** and is located in the **\SbtTools\sbt_backend\bin\win32\opt** directory. The **iecutil.exe** executable can be copied into the same directory as the mobileFPGA bitstream image or can be pointed to on the command line.

mobileFPGA Bitstream Configuration File

The required bitstream image is part of the iCEcube2 project. Multiple versions of the bitstream are stored in the **<projname>_Implmnt\sbt\outputs\bitmap** directory. The raw hexadecimal version of the bitstream is called **<projname>_bitmap.hex**. The alternate format of the same information is an Intel hexadecimal file called **<projname>_bitmap_int.hex**.

Raw Hexadecimal Command Example

```
<path>/iceutil -d iCE40 -res -cr -m M25P10A -fh -w <path/projname>_bitmap.hex
```

Intel Hexadecimal Command Example

```
<path>/iceutil -d iCE40 -res -cr -m M25P10A -fi -w <path/projname>_bitmap_int.hex
```

Help

```
<path>/iceutil -help
```

Testing Core Power

Jumper JP1 provides the ability to measure core power consumption by the mobileFPGA. Two power measurement methods are supported.

> The iCEblink40 HX1K evaluation board uses an early version of the iCE40HX1K silicon that has higher than expected static current consumption. Although the demonstration application consumes less than 500 µA, the production silicon will consume even less current. Similarly, the lower power iCE40LP1K devices use even less power than the iCE40HX1K mobileFPGA.

Easy Method using a Multimeter

Connect the iCEblink40 board through your high-accuracy multimeter. Use a meter with a minimum of 10,000 counts; 50,000 counts or more is recommended for better accuracy.

To take a quick measurement, follow these steps.

1. Disconnect power to the iCEblink40 board by removing the USB cable connection, either at the board or at the computer.
2. Remove the jumper JP1, which isolates the mobileFPGA's core supply from the 1.2V supply on the board.
3. Connect your multimeter's alligator or test clips to the stake pins on header JP1.
4. Configure the multimeter to measure current using its highest mA or Amp range. This setting typically has the lowest voltage drop internally within the meter.
5. Re-connect the USB cable that supplies power to the iCEblink40 board and configure the mobileFPGA device if necessary.
6. Observe the power reading on the multimeter. At low clock rates, which resulst in lower power consumption, switch the meter to a lower amperage setting for better accuracy. However, this also may increase the resistance across the meter leads. Using too low of a meter setting causes a large voltage drop within the meter, potentially violating the minimum input voltage specification to the mobileFPGA device.
7. The value measured by the multimeter is a current. Convert the measurement to power using Equation 1. The voltage is the operating voltage, the voltage across the jumper. This value can be accurately measured with a second multimeter to show the voltage drop across the first. However, just measuring the initial voltage, before taking any current readings, usually provides acceptable accuracy and the voltage drop across the meter is generally small.

Equation 1

$$\text{Power} = \text{Current} \times \text{Voltage}$$

Although this method is easy, here are a few caveats and pointers.

- Always start at the highest current setting for your meter. Using too small a setting may damage your meter! After determining the maximum current range for your measurement, then you can safely use the appropriate lower current setting.
- The voltage drop across the meter leads may violate the minimum supply voltage specification for the mobileFPGA device. To determine the voltage drop, use a second multimeter to measure either the voltage across the first meter's leads during a test or the resistance between the first meter's leads.
- Using the highest current measurement setting typically results in the lowest voltage drop.

Using High-Precision, Small-Value Resistors

For more-accurate, time-sensitive measurements, place a low-value resistor across the jumper test point. According to Ohm's Law, the current passing through the resistor produces a voltage drop. Measure the voltage differential across the resistor during expected operation. Convert the measurement to power using Equation 2. The voltage is the measured voltage across the resistor; the resistance is the value of the resistor.

Equation 2

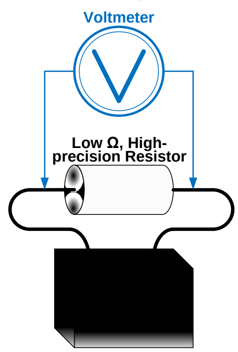
$$\text{Power} = \frac{(\text{Voltage})^2}{\text{Resistance}}$$

The following are a few guidelines on selecting a resistor.

- Use a high-precision resistor.
- The resistor must handle the power dissipated under the anticipated test conditions.
- Too small a resistor value may result in too small a voltage difference across the resistor to measure with your test equipment.
- Too large a resistor value may result in too large of a voltage difference across the resistor. Too large a voltage drop might violate the minimum voltage specifications for the mobileFPGA device.

Figure 21 shows an example header block designed to fit over one of the jump locations. Measure the voltage drop across the low-value resistor, either with a voltmeter or with data acquisition equipment.

Figure 21: **Resistor Header Block**

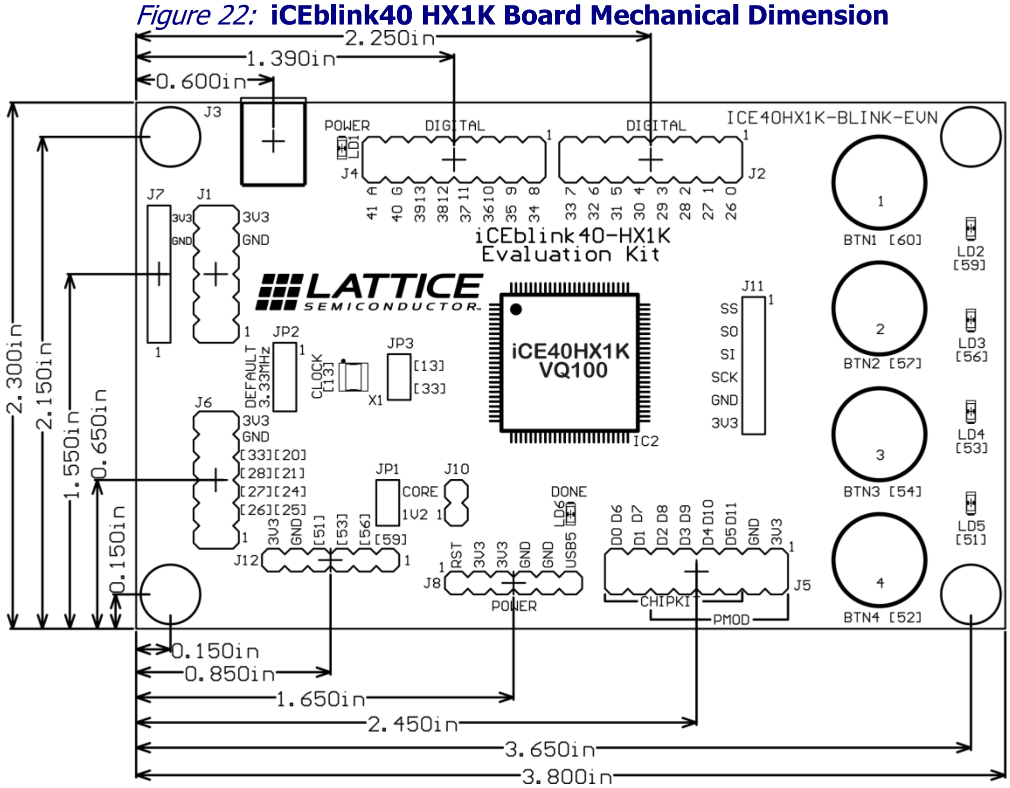


This method is recommended for taking power measurements over time.

Mechanical Specifications

Figure 22 shows the mechanical dimensions for the iCEblink40 board, including the location of the four mounting holes. With a jumper installed on JP1, the board height is approximately 0.700 inches high, including the four rubber feet mounted on the bottom side of the board.

Figure 22: **iCEblink40 HX1K Board Mechanical Dimension**



Ordering Information

Table 11 lists the available or planned iCEblink40 boards, the mobileFPGA mounted on the board, and the ordering information for the boards.

Table 11: **Ordering Information for iCEblink40 Evaluation Kits**

Product	mobileFPGA	Part Number
iCEblink40-HX1K Evaluation Kit	iCE40HX1KVQ100	ICE40HX1K-BLINK-EVN
iCEblink40-LP1K Evaluation Kit (in development)	iCE40LP1KQN84	ICE40LP1K-BLINK-EVN

Demonstration Application (Verilog)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// CAPACTIVE TOUCH BUTTON demo for iCEblink40 board, VQ100 version
//
// Version 1.1
// Date:  1-MAR-2012
// ===============================================================================
// Description:
//
// When board powers on, the green LEDs along the right edge scroll upward.
// If any button is pressed, the LEDs instead display the current toggle state of
// the buttons.  If no button is pressed in five seconds, then the LEDs return to
// displaying the upward-scrolling pattern.
//////////////////////////////////////////////////////////////////////////////////

module iceblink40_demo(
   input  CLK_3P3_MHZ,    // 3.3 MHz clock from LTC1799 oscillator (pin 13)
   inout  BTN1 ,          // Connection to cap-sense button BTN1    (pin 60)
   inout  BTN2 ,          // Connection to cap-sense button BTN2    (pin 57)
   inout  BTN3 ,          // Connection to cap-sense button BTN2    (pin 54)
   inout  BTN4 ,          // Connection to cap-sense button BTN3    (pin 52)
   output LED1 ,          // Drives LED LD2          (pin 59)
   output LED2 ,          // Drives LED LD3          (pin 56)
   output LED3 ,          // Drives LED LD4          (pin 53)
   output LED4 ,          // Drives LED LD5          (pin 51)
   // -- Digilent ADEPT 2 I/O Expander Debug Interface
   input  ASTB ,          // Address strobe          (pin 26)
   input  DSTB ,          // Data strobe             (pin 27)
   input  WRITE ,         // Read/Write control      (pin 28)
   inout [7:0] DB ,       // Data bus, byte-wide     (pins 29, 30, 34, 36, 37, 40, 41, 42)
   output WAIT ,          // Wait signal             (pin 33)
   output SS_B            // SPI slave-select output (pin 49)
);

   wire LED_CLOCK ;               // Controls the flashing rate of LEDs in scroll mode (about 0.8 seconds)
   wire BTN_SAMPLE ;              // Controls how often the capacitive-sense buttons are sampled
   wire [3:0] ROTATER ;           // Generates the scrolling pattern for the LEDs
   wire [3:0] TIMEOUT_COUNT ;     // Counter that tracks last time any button was pressed
   wire TIMEOUT ;                 // Generates time-out signal after 5 seconds if no button pressed
   // -- Digilent ADEPT 2 I/O Expander Debug Graphical Interface (GUI) connections
   wire [15:0] VSwitches ;        // 16 virtual slide switches controlled from graphical interface
   wire [15:0] VButtons ;         // 16 virtual pushbuttons controlled from graphical interface
   reg  [ 7:0] VLEDs ;            // 8 virtual LEDs controlled by FPGA; displayed on graphical interface
   wire [31:0] ToFPGA ;           // 32-bit value written to the FPGA, specified via textbox on GUI
   wire [31:0] FromFPGA ;         // 32-bit value provided by the FPGA; displayed in textbox on GUI
   wire [23:0] VLightBar ;        // 24 virtual lights controlled by the FPGA (8 green, 8 yellow, 8 red)
                                  // ... Green = [23:16], Yellow = [15:8], Red = [7:0]

   wire DLED1 ;                   // Display control for LED1, multiplexed cap-sense buttons and virtual I/O
   wire DLED2 ;                   // Display control for LED2, multiplexed cap-sense buttons and virtual I/O
   wire DLED3 ;                   // Display control for LED3, multiplexed cap-sense buttons and virtual I/O
   wire DLED4 ;                   // Display control for LED4, multiplexed cap-sense buttons and virtual I/O

   wire BTN1_TOGGLE_STATUS ;      // Holds current toggle status of BTN1
   wire BTN2_TOGGLE_STATUS ;      // Holds current toggle status of BTN2
   wire BTN3_TOGGLE_STATUS ;      // Holds current toggle status of BTN3
   wire BTN4_TOGGLE_STATUS ;      // Holds current toggle status of BTN4

   assign SS_B = 1'b1 ;           // Disable SPI Flash after configuration


   // Generates the LED_CLOCK and the BTN_SAMPLE signals
   // Also provides the prescaler for the 5 second time-out counter
   CLK_DIVIDER_3P3MHz CLK_DIV (
      .CLK_3P3MHz(CLK_3P3_MHZ),
      .LED_CLOCK(LED_CLOCK),
      .BTN_SAMPLE(BTN_SAMPLE),
      .TC(DIVIDER_TC)
   );

   // Simply scrolls the LEDs in one direction
   ROTATE_LED BLINKY (
      .CLK(LED_CLOCK),
      .LEFT(VSwitches[7]),        // controls LED scrolling direction; upward by default
      .LED(ROTATER)
   );

   // Chooses whether to scroll or to toggle the buttons
   DISPLAY_MODE SELECT_OUTPUT(
      .CLK(CLK_3P3_MHZ),
      .BTN_CHANGED(ANY_BTN_CHANGED),
      .TIMEOUT(TIMEOUT),
      .MODE(MODE)
   );
```

```
    // Generates a time-out reset signal if no button is pressed within last 5 seconds
    TIMEOUT_COUNTER DELAY_5_SECONDS (
       .CLK(CLK_3P3_MHZ) ,
       .ENABLE(DIVIDER_TC) ,
       .RESET(ANY_BTN_CHANGED) ,
       .TIMEOUT_COUNT(TIMEOUT_COUNT) ,
       .TIMEOUT(TIMEOUT)
    );

    // Capacitive-sense button controller
    CAPSENSEBUTTONS BUTTONS (
       .CLK(CLK_3P3_MHZ) ,
       .BTN1(BTN1) ,
       .BTN2(BTN2) ,
       .BTN3(BTN3) ,
       .BTN4(BTN4) ,
       .BTN_SAMPLE(BTN_SAMPLE),
       .ANY_BTN_CHANGED(ANY_BTN_CHANGED),
       .BTN1_TOGGLE_STATUS(BTN1_TOGGLE_STATUS) ,
       .BTN2_TOGGLE_STATUS(BTN2_TOGGLE_STATUS) ,
       .BTN3_TOGGLE_STATUS(BTN3_TOGGLE_STATUS) ,
       .BTN4_TOGGLE_STATUS(BTN4_TOGGLE_STATUS)
    );

    // Depending on the operating mode, the LEDs either scroll or can be individually toggled
    assign DLED1 = ( (MODE) ? BTN1_TOGGLE_STATUS : ROTATER[0] ) ;
    assign DLED2 = ( (MODE) ? BTN2_TOGGLE_STATUS : ROTATER[1] ) ;
    assign DLED3 = ( (MODE) ? BTN3_TOGGLE_STATUS : ROTATER[2] ) ;
    assign DLED4 = ( (MODE) ? BTN4_TOGGLE_STATUS : ROTATER[3] ) ;

    // Allow USB debug to control the interface
    assign LED1 = ( (VSwitches[6]) ? (VButtons[0] ^ VSwitches[0]) : DLED1 ) ;
    assign LED2 = ( (VSwitches[6]) ? (VButtons[1] ^ VSwitches[1]) : DLED2 ) ;
    assign LED3 = ( (VSwitches[6]) ? (VButtons[2] ^ VSwitches[2]) : DLED3 ) ;
    assign LED4 = ( (VSwitches[6]) ? (VButtons[3] ^ VSwitches[3]) : DLED4 ) ;

    // Controls the three-color display on the Digilent ADEPT 2 virtual I/O display
    DIOX_SHIFTER_DISPLAY LIGHTBAR_DISPLAY (
       .CLK(LED_CLOCK) ,
       .RESET(VButtons[15]) ,
       .HOLD(VButtons[14]) ,
       .SHIFTER(VLightBar)
    );

    // Controls which values are displayed in the FromFPGA textbox on Digilent ADEPT 2 virtual I/O display
    DIOX_RETURN_DISPLAY FromFPGA_DISPLAY (
       .CLK(LED_CLOCK) ,
       .SEL(VSwitches[15:14]) ,
       .RESET(VButtons[13]) ,
       .DATA_TO_FPGA(ToFPGA) ,
       .DATA_FROM_FPGA(FromFPGA)
    );

    always @(*)
       if (VSwitches[9:8] == 2'b00) // Display rotater and current toggle button status
          VLEDs = ({ROTATER[3:0], BTN4_TOGGLE_STATUS, BTN3_TOGGLE_STATUS,
                    BTN2_TOGGLE_STATUS, BTN1_TOGGLE_STATUS});
        else if (VSwitches[9:8] == 2'b01) // Button time-out counter and current toggle button status
           VLEDs = ({TIMEOUT_COUNT, BTN4_TOGGLE_STATUS, BTN3_TOGGLE_STATUS,
                     BTN2_TOGGLE_STATUS, BTN1_TOGGLE_STATUS});
        else // if (VSwitches[9:8] == 2'b11) // Display values based on VSwitches[3:0], VButtons[3:0]
           VLEDs = ( {VSwitches[3:0], VButtons[3:0]} );

    Digilent_IOX USB_DEBUG (
       .ASTB(ASTB) ,              // Address strobe
       .DSTB(DSTB) ,              // Data strobe
       .WRITE(WRITE) ,            // Read/Write control
       .DB(DB) ,                  // Data bus, byte-wide
       .WAIT(WAIT) ,              // Wait control
        // --- Virtual I/O signals
       .VLEDs(VLEDs) ,            // 8 virtual LEDs on GUI, open circles
       .VLightBar(VLightBar) ,    // 24 virtual lights on GUI (8 green, 8 yellow, 8 read)
       .VSwitches(VSwitches) ,    // 16 virtual slide switches on GUI
       .VButtons(VButtons) ,      // 16 virtual momentary pushbuttons on GUI
       .FromFPGA(FromFPGA) ,      // 32-bit value (From FPGA on GUI)
       .ToFPGA(ToFPGA)            // 32-bit value (To FPGA on GUI)
 );

 endmodule
```

```
module CAPSENSEBUTTONS (
   inout  BTN1 ,
   inout  BTN2 ,
   inout  BTN3 ,
   inout  BTN4 ,
   input  BTN_SAMPLE ,
   input  CLK ,
   output ANY_BTN_CHANGED ,
   output reg BTN1_TOGGLE_STATUS ,
   output reg BTN2_TOGGLE_STATUS ,
   output reg BTN3_TOGGLE_STATUS ,
   output reg BTN4_TOGGLE_STATUS
);

   reg STATUS_ALL_BUTTONS = 1'b0 ;        // Indicates the status of all four buttons
   reg STATUS_ALL_BUTTONS_LAST = 1'b0 ; // Indicates the status during the last clock cycle
   reg SAMPLE_BTN1 = 1'b0 ;               // Captures the value on BTN1
   reg SAMPLE_BTN2 = 1'b0 ;               // Captures the value on BTN2
   reg SAMPLE_BTN3 = 1'b0 ;               // Captures the value on BTN3
   reg SAMPLE_BTN4 = 1'b0 ;               // Captures the value on BTN4
   reg SAMPLE_BTN1_LAST = 1'b0 ;          // Hold the previous value of BNT1
   reg SAMPLE_BTN2_LAST = 1'b0 ;          // Hold the previous value of BNT2
   reg SAMPLE_BTN3_LAST = 1'b0 ;          // Hold the previous value of BNT3
   reg SAMPLE_BTN4_LAST = 1'b0 ;          // Hold the previous value of BNT4
   reg BTN1_TOGGLE_STATUS = 1'b0 ;        // Holds current toggle status of BTN1
   reg BTN2_TOGGLE_STATUS = 1'b0 ;        // Holds current toggle status of BTN2
   reg BTN3_TOGGLE_STATUS = 1'b0 ;        // Holds current toggle status of BTN3
   reg BTN4_TOGGLE_STATUS = 1'b0 ;        // Holds current toggle status of BTN4

   wire BTN1_CHANGED ;            // Indicates that the value on BTN1 changed from the previous sample
   wire BTN2_CHANGED ;            // Indicates that the value on BTN2 changed from the previous sample
   wire BTN3_CHANGED ;            // Indicates that the value on BTN3 changed from the previous sample
   wire BTN4_CHANGED ;            // Indicates that the value on BTN4 changed from the previous sample

   // Capacitive buttons are driven to a steady Low value to bleed off any charge,
   // then allowed to float High.  An external resistor pulls each button pad High.
   assign BTN1 = ( (BTN_SAMPLE) ? 1'bz : 1'b0 ) ;
   assign BTN2 = ( (BTN_SAMPLE) ? 1'bz : 1'b0 ) ;
   assign BTN3 = ( (BTN_SAMPLE) ? 1'bz : 1'b0 ) ;
   assign BTN4 = ( (BTN_SAMPLE) ? 1'bz : 1'b0 ) ;

   // Indicates when ANY of the four buttons goes High
   always @(posedge CLK)
      if (~BTN_SAMPLE)  // Clear status when buttons driven low
         STATUS_ALL_BUTTONS <= 1'b0 ;
      else
      // Trigger whenever any button goes High, but only during first incident
         STATUS_ALL_BUTTONS <= (BTN1 | BTN2 | BTN3 | BTN4) & ~STATUS_ALL_BUTTONS_LAST ;

   // Indicates the last status of all four buttons
   always @(posedge CLK)
      if (~BTN_SAMPLE)  // Clear status when buttons driven low
         STATUS_ALL_BUTTONS_LAST <= 1'b0 ;
      else if (STATUS_ALL_BUTTONS)
         STATUS_ALL_BUTTONS_LAST <= STATUS_ALL_BUTTONS ;

   always @(posedge CLK)
      if (STATUS_ALL_BUTTONS)            // If any button went High after driving it low ...
      begin                              //     ... wait one clock cycle before re-sampling the pin value
         SAMPLE_BTN1  <= ~BTN1 ;  // Invert polarity to make buttons active-High
         SAMPLE_BTN2  <= ~BTN2 ;
         SAMPLE_BTN3  <= ~BTN3 ;
         SAMPLE_BTN4  <= ~BTN4 ;
         SAMPLE_BTN1_LAST  <= SAMPLE_BTN1 ; // Save last sample to see if the value changed
         SAMPLE_BTN2_LAST  <= SAMPLE_BTN2 ;
         SAMPLE_BTN3_LAST  <= SAMPLE_BTN3 ;
         SAMPLE_BTN4_LAST  <= SAMPLE_BTN4 ;
      end


   // Toggle switch effect
   assign BTN1_CHANGED = ( SAMPLE_BTN1 & !SAMPLE_BTN1_LAST ) ; // Sampled pin value changed
   assign BTN2_CHANGED = ( SAMPLE_BTN2 & !SAMPLE_BTN2_LAST ) ;
   assign BTN3_CHANGED = ( SAMPLE_BTN3 & !SAMPLE_BTN3_LAST ) ;
   assign BTN4_CHANGED = ( SAMPLE_BTN4 & !SAMPLE_BTN4_LAST ) ;

   // Indicates that one of the buttons was pressed
   assign ANY_BTN_CHANGED = ( BTN1_CHANGED | BTN2_CHANGED | BTN3_CHANGED | BTN4_CHANGED ) ;
```

```verilog
    // If any button is pressed, toggle the button's current value
    always @(posedge CLK)
    begin
    if (BTN1_CHANGED)
       BTN1_TOGGLE_STATUS <= ~(BTN1_TOGGLE_STATUS) ;
    if (BTN2_CHANGED)
       BTN2_TOGGLE_STATUS <= ~(BTN2_TOGGLE_STATUS) ;
    if (BTN3_CHANGED)
       BTN3_TOGGLE_STATUS <= ~(BTN3_TOGGLE_STATUS) ;
    if (BTN4_CHANGED)
       BTN4_TOGGLE_STATUS <= ~(BTN4_TOGGLE_STATUS) ;
    end

endmodule

// The clock divider to generate the LED_CLOCK and BTN_SAMPLE signals
module CLK_DIVIDER_3P3MHz (
    input  CLK_3P3MHz,
    output LED_CLOCK ,
    output BTN_SAMPLE,
    output TC
);

    reg [19:0] COUNTER = 20'b0 ;

    always @(posedge CLK_3P3MHz)
       COUNTER <= COUNTER + 1;

    assign LED_CLOCK = COUNTER[17] ;

    assign BTN_SAMPLE = COUNTER[19] ;

    assign TC = (COUNTER == 20'b11111111111111111111) ;

endmodule

// Generates a time-out reset signal if no button pressed within 5 seconds
module TIMEOUT_COUNTER (
    input  CLK ,
    input  ENABLE ,
    input  RESET ,
    output reg [3:0] TIMEOUT_COUNT = 4'b0 ;
    output TIMEOUT
);

    always @(posedge CLK)
       if (RESET)
          TIMEOUT_COUNT <= 4'b0 ;
       else if (ENABLE)
          TIMEOUT_COUNT <= TIMEOUT_COUNT + 4'b0001 ;

    assign TIMEOUT = (TIMEOUT_COUNT == 4'b1111) ;

endmodule

// Scrolls the LEDs
module ROTATE_LED (
    input        CLK,
    input        LEFT,
    output [3:0] LED
);

    reg [5:0] ROTATE = 6'b0 ;

    always @(posedge CLK)
       if (LEFT)
          ROTATE = ({ROTATE[4:0], ~ROTATE[5]});
       else
          ROTATE = ({~ROTATE[0], ROTATE[5:1]});

    assign LED = ROTATE[3:0] ;

endmodule
```

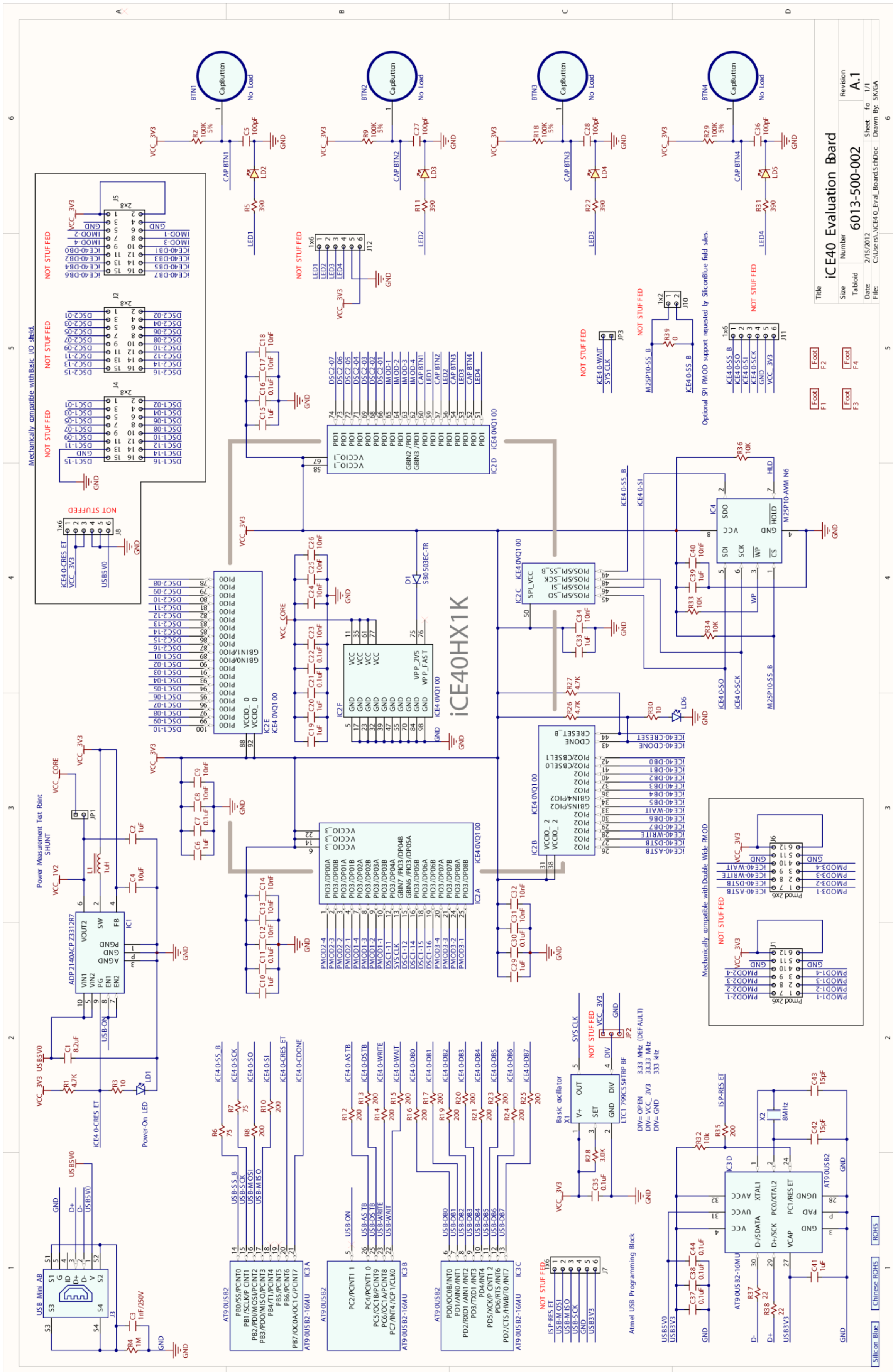
```
// Controls whether the LEDs scroll or show the current toggle state of the buttons
module DISPLAY_MODE (
   input  CLK,
   input  BTN_CHANGED,
   input  TIMEOUT,
   output reg MODE
);

   always @(posedge CLK)
      // Enter toggle mode if BTN2 or BTN3 is pressed
      if ( BTN_CHANGED )
         MODE <= 1'b1 ;
      // If all the buttons are turned off, then re-enter scroll mode
      else if ( TIMEOUT )
         MODE <= 1'b0 ;

endmodule

module DIOX_SHIFTER_DISPLAY (
   input CLK ,
   input RESET ,
   input HOLD ,
   output reg [23:0] SHIFTER = 24'b0
);

   always @(posedge CLK)
      if (RESET | SHIFTER[23])
         SHIFTER <= 24'b0 ;
      else if (~HOLD)
         SHIFTER <= ( { SHIFTER[22:0], 1'b1 } ) ;

endmodule

module DIOX_RETURN_DISPLAY (
   input CLK ,
   input [1:0] SEL ,
   input RESET ,
   input [31:0] DATA_TO_FPGA ,
   output reg [31:0] DATA_FROM_FPGA = 32'b0
);

   always @(posedge CLK)
      if (RESET)
         DATA_FROM_FPGA <= 32'b0 ;
      else if (SEL == 2'b00)
         DATA_FROM_FPGA <= DATA_FROM_FPGA + 32'b1 ; // Increment
      else if (SEL == 2'b10)
         DATA_FROM_FPGA <= DATA_FROM_FPGA - 32'b1 ; // Decrement
      else if (SEL == 2'b01)
         DATA_FROM_FPGA <= ~(DATA_TO_FPGA) ;  // Complement
      else // if (SEL == 3'b10)
      begin : swapper // Reverse bits
         integer n;
         for (n = 0; n < 32 ; n=n+1)
            DATA_FROM_FPGA[31-n] <= DATA_TO_FPGA[n] ;
      end

endmodule


module Digilent_IOX (
   // -- Connections to Digilent Parallel Port (DPP) interface
   // -- Controlled by Digilent ADEPT 2 software and USB controller
   input ASTB ,      // Address strobe
   input DSTB ,      // Data strobe
   input WRITE ,     // Read/Write control
   inout [7:0] DB,   // Data bus, byte-wide
   output WAIT ,     // Wait control
   // --- Virtual I/O signals
   input [7:0] VLEDs ,           // 8 virtual LEDs on GUI, open circles
   input [23:0] VLightBar ,      // 24 virtual lights on GUI (8 green, 8 yellow, 8 red)
   output reg [15:0] VSwitches , // 16 virtual slide switches on GUI
   output reg [15:0] VButtons ,  // 16 virtual momentary pushbuttons on GUI
   input [31:0] FromFPGA ,       // 32-bit value (From FPGA on GUI)
   output reg [31:0] ToFPGA      // 32-bit value (To FPGA on GUI)
);

   reg [7:0] AddressRegister ;   // Address register
   reg [7:0] CommValidRegister ; // Confirm communication link reading complement of value written to FPGA
   reg [7:0] busIOXinternal ;    // Internal data bus

   // Assert WAIT signal whenever ASTB or DSTB are Low.  Maximum port speed.
   assign WAIT = ( !ASTB | !DSTB ) ;

   // Control data direction to/from IOX interface
   // If WRITE = 1, then read value on busIOXinternal.  If WRITE = 0, set outputs to three-state (Hi-Z)
   assign DB = ( (WRITE) ? busIOXinternal : 8'bZZZZZZZZ ) ;
```

```
    // Read values from inside FPGA application and display on GUI
    always @(*)
    begin
       if (!ASTB)                                 // When ASTB is Low
          busIOXinternal <= AddressRegister ;     // ... Read address register
       else if (AddressRegister == 8'h00)         // When address is 0x00
          busIOXinternal <= CommValidRegister ; // ... return complement of CommValidRegister value
       else if (AddressRegister == 8'h01)         // When address is 0x01
          busIOXinternal <= VLEDS ;               // ... value presented on virtual LEDs
       else if (AddressRegister == 8'h02)         // When address is 0x02
          busIOXinternal <= VLightBar[7:0] ;      // ... value presented on right-most Light Bar (red lights)
       else if (AddressRegister == 8'h03)         // When address is 0x03
          busIOXinternal <= VLightBar[15:8] ;     // ... value presented on middle Light Bar (yellow lights)
       else if (AddressRegister == 8'h04)         // When address is 0x04
          busIOXinternal <= VLightBar[23:16] ;    // ... value presented on left-most Light Bar (green lights)
       else if (AddressRegister == 8'h0d)         // When address is 0x0D
          busIOXinternal <= FromFPGA[7:0] ;       // ... value presented in "From FPGA" text box, bits 7:0
       else if (AddressRegister == 8'h0e)         // When address is 0x0E
          busIOXinternal <= FromFPGA[15:8] ;      // ... value presented in "From FPGA" text box, bits 15:8
       else if (AddressRegister == 8'h0f)         // When address is 0x0F
          busIOXinternal <= FromFPGA[23:16] ;     // ... value presented in "From FPGA" text box, bits 23:16
       else if (AddressRegister == 8'h10)         // When address is 0x10
          busIOXinternal <= FromFPGA[31:24] ;     // ... value presented in "From FPGA" text box, bits 31:24
       else
          busIOXinternal <= 8'b11111111 ;         // Otherwise, read all ones (any non-data value)
    end

    // EPP Address Register
    // If WRITE = 0, load Address Register at rising-edge of ASTB
    always @(posedge ASTB)
       if (!WRITE)
          AddressRegister <= DB;

    // Write Various Registers based on settings from GUI controls
    // If WRITE = 0, load register selected by Address Register at rising-edge of DSTB
    always @(posedge DSTB)
       if (!WRITE)
       begin
          if (AddressRegister == 8'h00)           // When address is 0x00
             CommValidRegister <= ~DB ;           // ... load Verification register with complement of value
                                                  // ... written.  The GUI writes to this register to verify
                                                  // ... proper communication with target
          else if (AddressRegister == 8'h05)      // When address is 0x05
             VSwitches[7:0]  <= DB ;              // ... load value from bottom set of slide switches in GUI
          else if (AddressRegister == 8'h06)      // When address is 0x06
             VSwitches[15:8] <= DB ;              // ... load value from top set of slide switches in GUI
          else if (AddressRegister == 8'h07)      // When address is 0x07
             VButtons[7:0]   <= DB ;              // ... load value from bottom set of pushbuttons in GUI
          else if (AddressRegister == 8'h08)      // When address is 0x08
             VButtons[15:8]  <= DB ;              // ... load value from top set of pushbutton switches in GUI
          else if (AddressRegister == 8'h09)      // When address is 0x09
             ToFPGA[7:0]   <= DB ;                // ... load value from "To FPGA"  in GUI, bits 7:0
          else if (AddressRegister == 8'h0a)      // When address is 0x0A
             ToFPGA[15:8]  <= DB ;                // ... load value from "To FPGA"  in GUI, bits 15:8
          else if (AddressRegister == 8'h0b)      // When address is 0x0B
             ToFPGA[23:16]  <= DB ;               // ... load value from "To FPGA"  in GUI, bits 23:16
          else if (AddressRegister == 8'h0c)      // When address is 0x0C
             ToFPGA[31:24]  <= DB ;               // ... load value from "To FPGA"  in GUI, bits 31:24
       end

 endmodule
```

I/O Constraints File

```
#################################
# iCEblink40 HX1K Demonstration Design
# Family & Device:    iCE40HX1K
# Package:            VQ100
#################################

###IOSet List 22

# Capacitive-sense buttons
set_io BTN1 60
set_io BTN2 57
set_io BTN3 54
set_io BTN4 52

# LED outputs
set_io LED1 59
set_io LED2 56
set_io LED3 53
set_io LED4 51

# 3.3 MHz clock input
set_io CLK_3P3_MHZ 13

# Digilent ADEPT2 USB interface
set_io ASTB  26
set_io DB[0] 42
set_io DB[1] 41
set_io DB[2] 40
set_io DB[3] 37
set_io DB[4] 36
set_io DB[5] 34
set_io DB[6] 30
set_io DB[7] 29
set_io DSTB  27
set_io WAIT  33
set_io WRITE 28

# SPI Flash enable control
set_io SS_B 49
```

Schematic

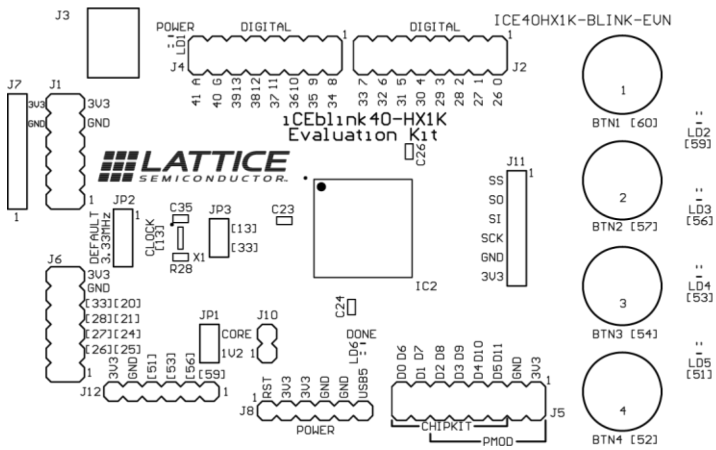


Bill of Materials (Major Components)

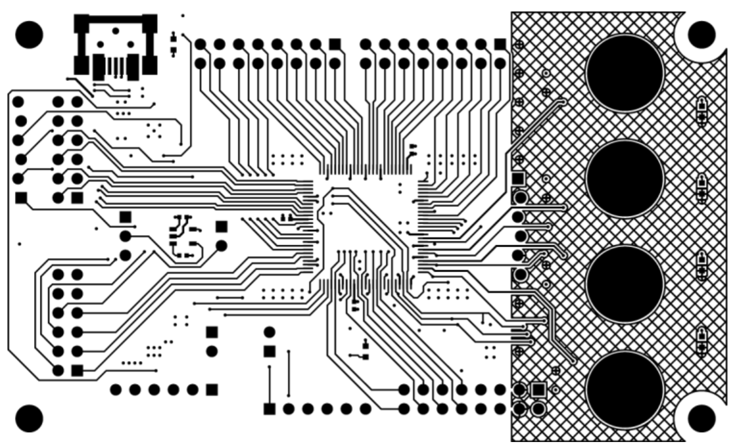
Reference	Vendor	Part Number	Description
IC2	Lattice Semiconductor	iCE40HX1K-VQ100	iCE40 HX-series mobileFPGA
IC1	Analog Devices, Inc.	ADP2140ACPZ3312R7	Low-quiescent buck/LDO regulator (1.2V, 3.3V)
X1	Linear Technology	LTC1799CS5#TRPBF	Oscillator
IC4	Micron Technology, Inc.	M25P10-AVMN6	1Mbit SPI serial configuration Flash PROM
IC3	Atmel Corporation	AT90USB162-16MU	USB programming and debugging interface

Printed Circuit Board Layout

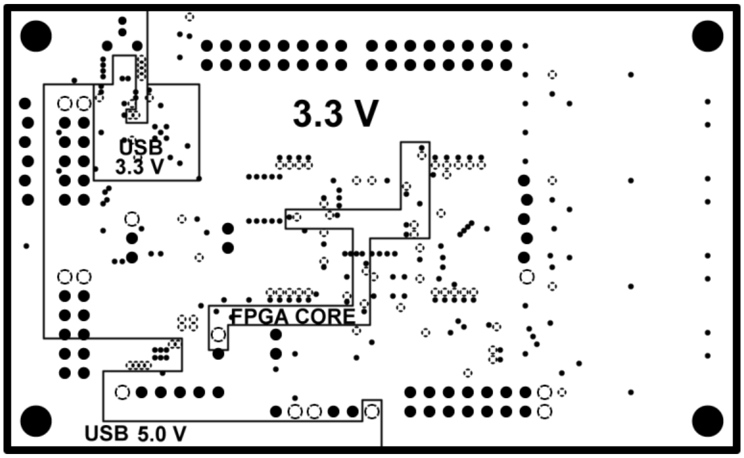
Top Silkscreen



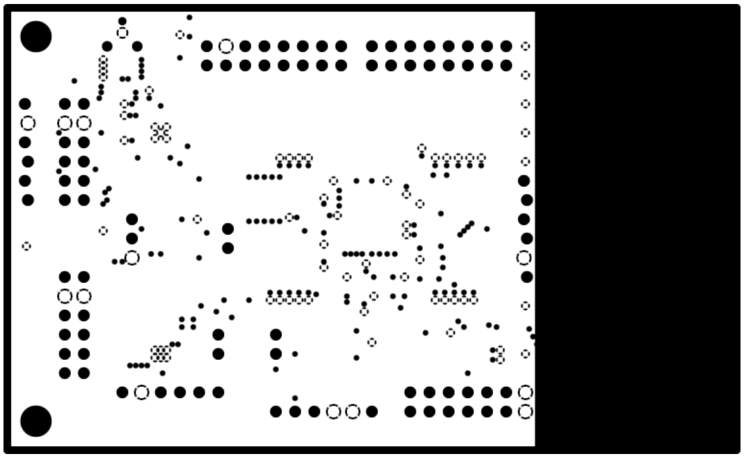
Top Signal Trace Layer



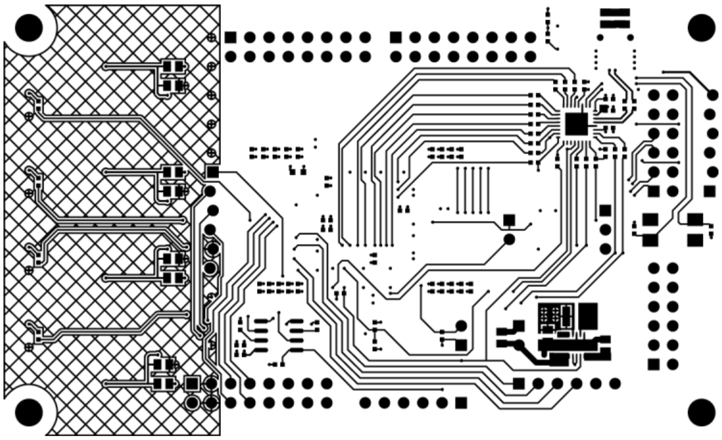
Power Plane



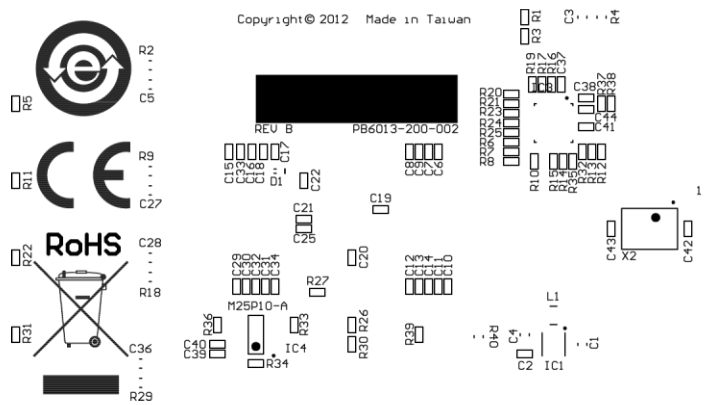
Ground Plan



Bottom Signal Trace Layer (flipped)



Bottom Silkscreen (flipped)



Revision History

Version	Date	Description
0.9.4	2-APR-2012	Added information on the capacitive touch button algorithm. Added a bill of materials for the major components. Added example layout for the board layers and silkscreen.
0.9.3	22-MAR-2012	Added information about the pre-programmed demonstration application, including the physical interface, the virtual I/O interface, the Verilog source application, and the I/O constraints file.
0.9.2	16-FEB-2012	Added details about Pmod modules and which modules are available.
0.9.1	15-FEB-2012	Added schematic diagram.
0.9	13-FEB-2012	Draft release for pre-production prototype the iCEblink40 board.

Lattice Semiconductor Corporation
5555 N.E. Moore Court
Hillsboro, Oregon 97124-6421
United States of America

Tel: +1 503 268 8000
Fax: +1 503 268 8347