SiliconBlue™

# SiliconBlue Technologies
## VHDL / Verilog Introduction

*Programmable Solutions for Consumer Handheld*

*7-MAY-2008 (v1.0)*

## Agenda

- VHDL / Verilog
- A Few Introductory Pointers
- A Quick Run through the System

# What is VHDL and Verilog

- Popular entry method for FPGA / ASIC designs
  - Verilog popular with ASIC engineers
  - VHDL popular with FPGA designers
- Originally designed for simulation / verification
- Now used to create (synthesize) a design from the description
- Just another programming language
  - Verilog looks and feels like ‘C’
  - VHDL looks and feels like Ada

**BE NOT AFRAID!**

- **REPEAT**: Just another programming language

# So What's the Difference?

**Traditional Programming Language**

- Runs on a processor
- Processor performs sequential operations

```
Do this.
Then, do this.
Then do this.
Afterwards, do this.
Followed by this.
```

**VHDL / Verilog**

- Runs on a processor for simulation purposes
- Designed to run in actual hardware
- Hardware is inherently parallel

```
Do this.
While doing this.
And doing this.
Simultaneously doing this.
And this.
```

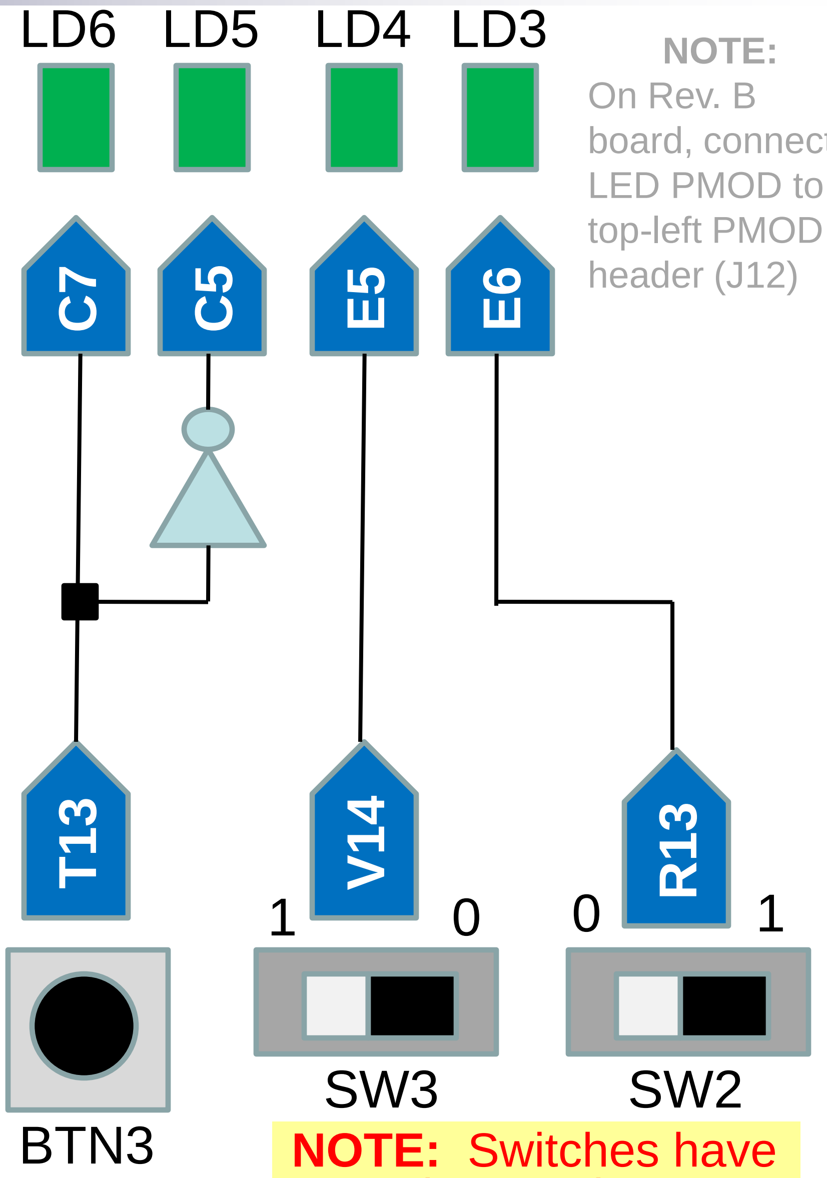
**Traditional programming languages not designed to describe parallel operations**

## SiliconBlue's Expectations

- You are not expected to be an expert VHDL / Verilog coder
- But, be able to demonstrate the iCECUBE design flow and iCEman65 board
  - Requires a basic understanding of VHDL/Verilog
  - Be able to recognize VHDL or Verilog when you encounter it

# LED Wires Example

- Your job? Simple! Build low-power programmable wires using the iCEman65 board
- Pick your language
  - VHDL
  - Verilog
- Implement it and download it to the board



**NOTE:**
On Rev. B board, connect LED PMOD to top-left PMOD header (J12)

**NOTE:** Switches have opposite polarity.

## First Things First

- When writing a program, what things do you do first?
  - Declare any required libraries
  - Declare variable names
  - Declare the data type for each variable
- Then
  - Define how variable values are assigned
- Knowing that VHDL and Verilog are just another programming language, what should we do?

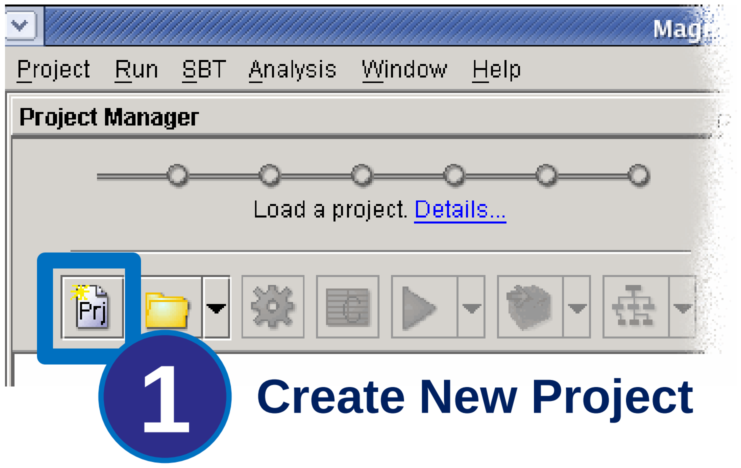
# Verilog Example

```
module led_wires (BTN3, SW, LD);
    // Declare inputs and outputs and their widths
    input        BTN3;
    input  [3:2] SW;
    output [6:3] LD;

    // Connect LEDs LD4 and LD3 directly to switches SW3 and SW2
    assign LD[4:3] = SW[3:2];

    // Connect pushbutton BTN3 directly to LD6
    assign LD[6] = BTN3;

    // Invert BTN3 and then connect it directly to LD5
    assign LD[5] = ~BTN3;

endmodule
```

# VHDL Example

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Declare inputs and outputs and their widths
entity led_wires is
    port ( BTN3 : in  STD_LOGIC;
           SW   : in  STD_LOGIC_VECTOR (3 downto 2);
           LD   : out STD_LOGIC_VECTOR (6 downto 3)
    );
end led_wires;

architecture Behavioral of led_wires is

begin

    -- Connect LEDs LD4 and LD3 directly to switches SW3 and SW2
    LD(4 downto 3) <= SW(3 downto 2) ;
    -- Connect pushbutton BTN3 directly to LD6
    LD(6) <= BTN3 ;
    -- Invert BTN3 and then connect it directly to LD5
    LD(5) <= not(BTN3) ;

end Behavioral ;
```

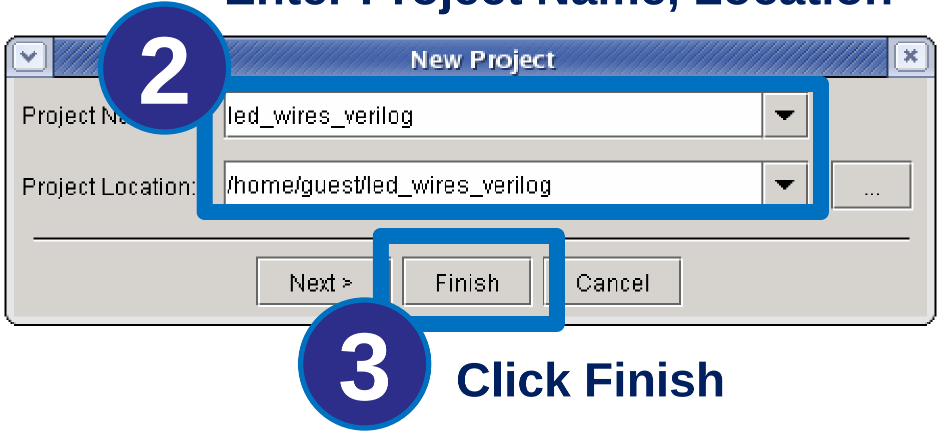
# Start a New Project



Project Run SBT Analysis Window Help
Project Manager
Load a project. Details...
Prj
1

Create New Project

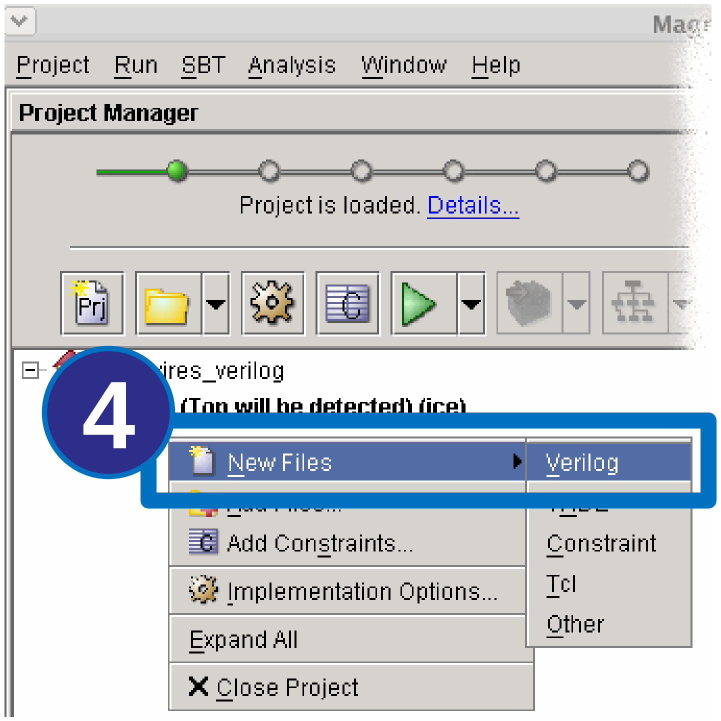
Enter Project Name, Location



Project Name: led_wires_verilog

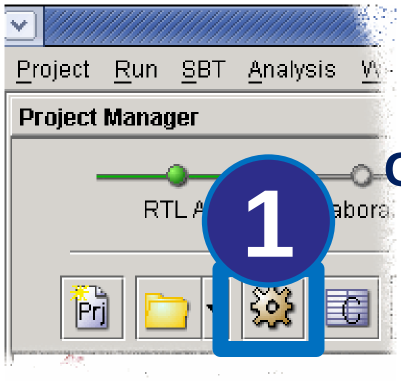
Project Location: /home/guest/led_wires_verilog

Click Finish



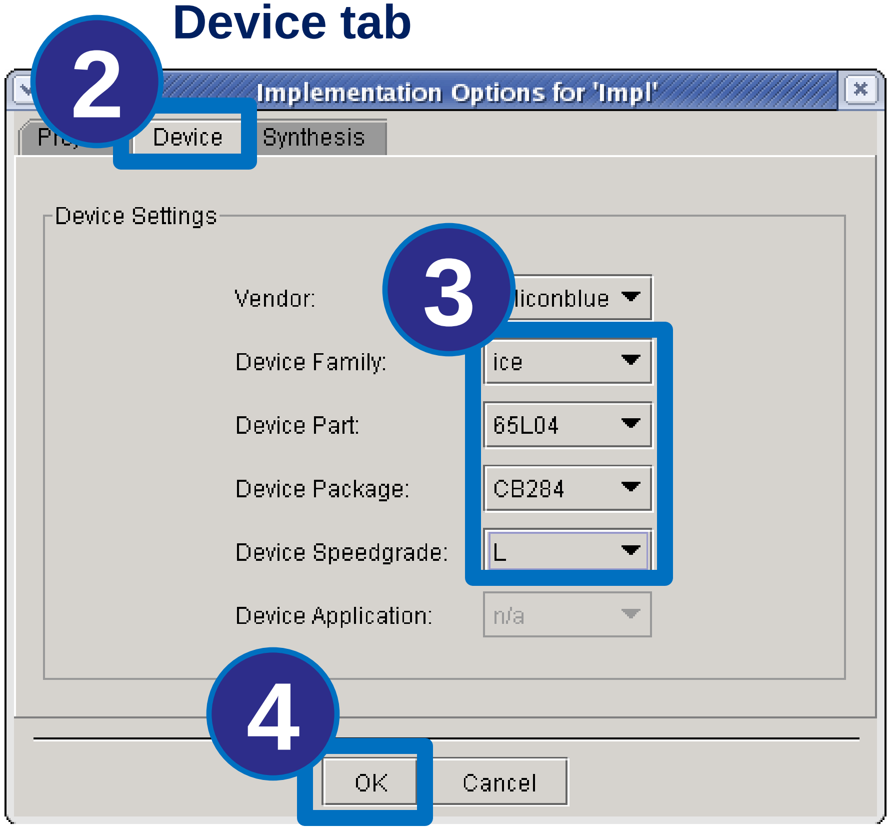
Right-click, New Files → Verilog

# Set Implementation Options



Click Implementation Options button

Device tab



Vendor: siliconblue
Device Family: ice
Device Part: 65L04
Device Package: CB284
Device Speedgrade: L
Device Application: n/a

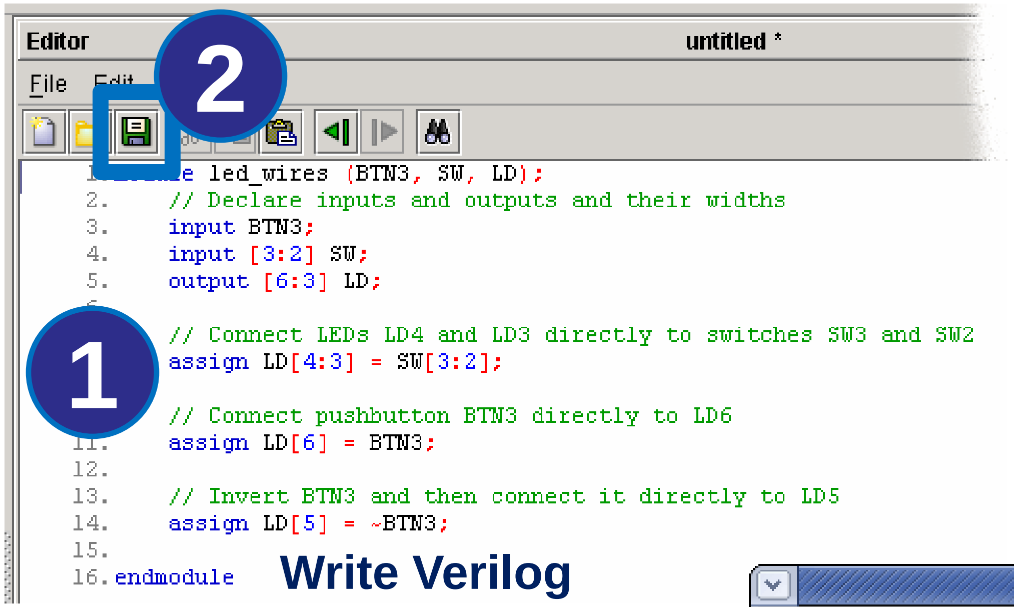
Set to iCE65 device on board

- 65L04
- CB284
- L

Click OK

# Create, Save Verilog File

Click Save

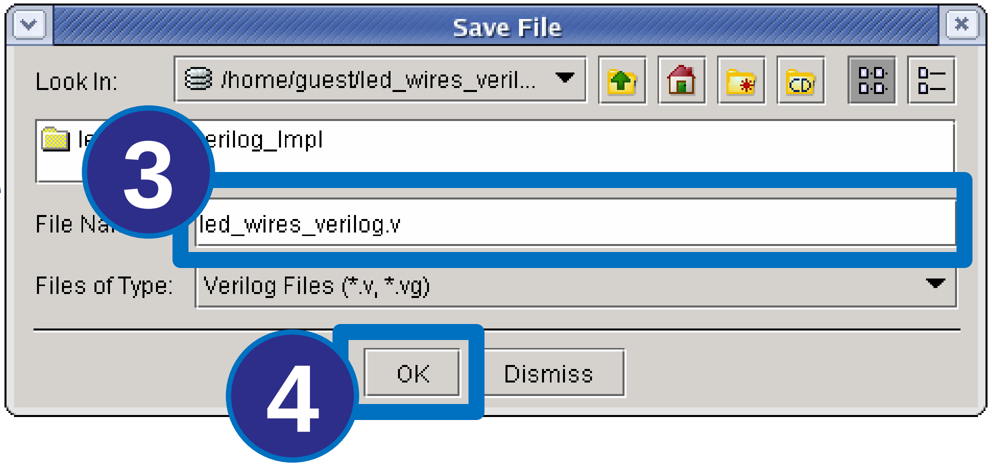


```
    module led_wires (BTN3, SW, LD);
    // Declare inputs and outputs and their widths
    input BTN3;
    input [3:2] SW;
    output [6:3] LD;

    // Connect LEDs LD4 and LD3 directly to switches SW3 and SW2
    assign LD[4:3] = SW[3:2];

    // Connect pushbutton BTN3 directly to LD6
    assign LD[6] = BTN3;

    // Invert BTN3 and then connect it directly to LD5
    assign LD[5] = ~BTN3;

endmodule
```

Write Verilog

Specify File Name

Click OK



Look In: /home/guest/led_wires_veril...

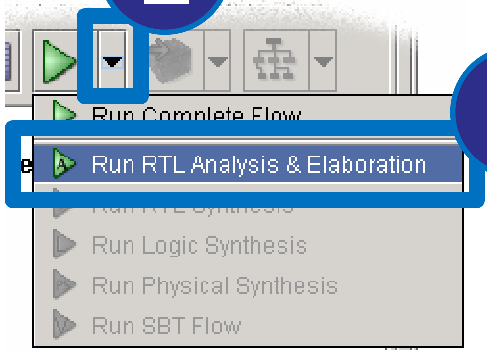
led_wires_Verilog_Impl

File Name: led_wires_verilog.v

Files of Type: Verilog Files (*.v, *.vg)

# Check Verilog for Errors

1 Click Drop-list



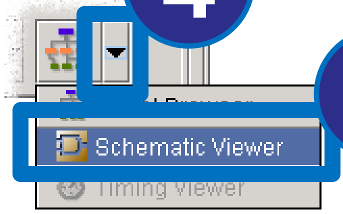
2 Click Run RTL

3 **Check for successful design message in Output window.**

If errors encountered, fix them then restart from Step 1.

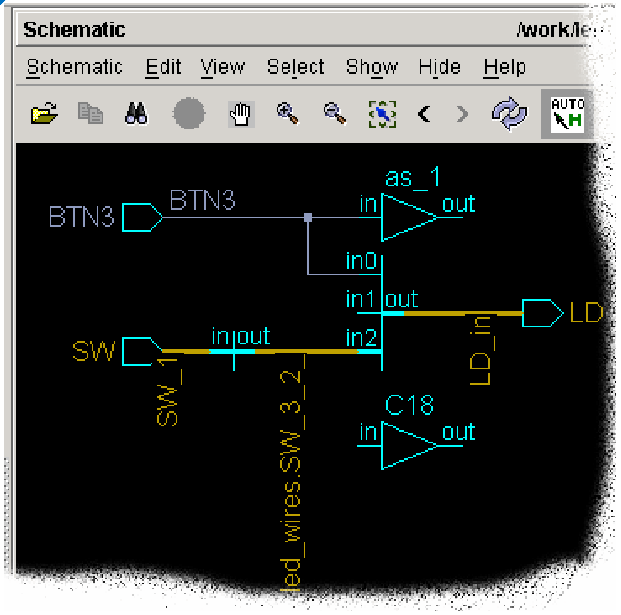
```
FPGA-7   RTL Analysis successful
```

4 Click Drop-list

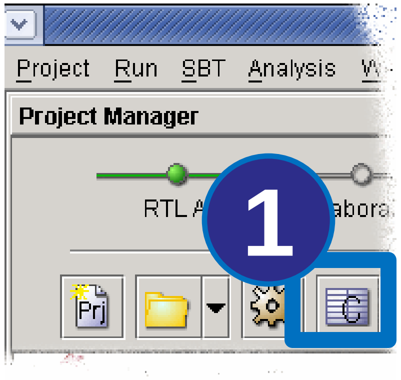


5 Click Schematic Viewer

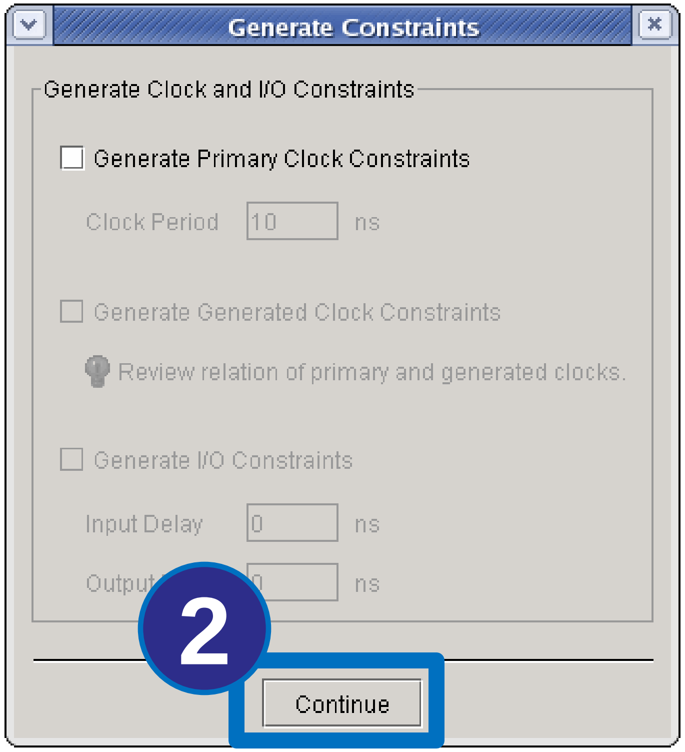
6 Check that schematic matches expected logic



# Create Pin Assignment



Click Constraints button



Generate Clock and I/O Constraints

- [ ] Generate Primary Clock Constraints

Clock Period 10 ns

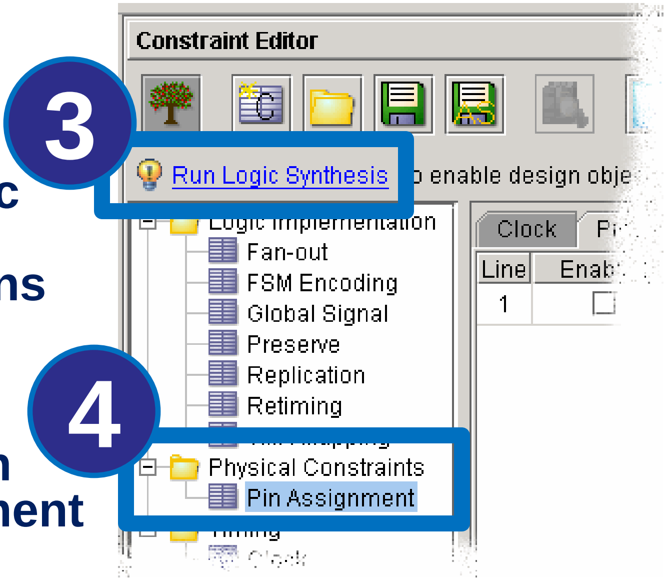
- [ ] Generate Generated Clock Constraints

Review relation of primary and generated clocks.

- [ ] Generate I/O Constraints

Input Delay 0 ns

Output 0 ns

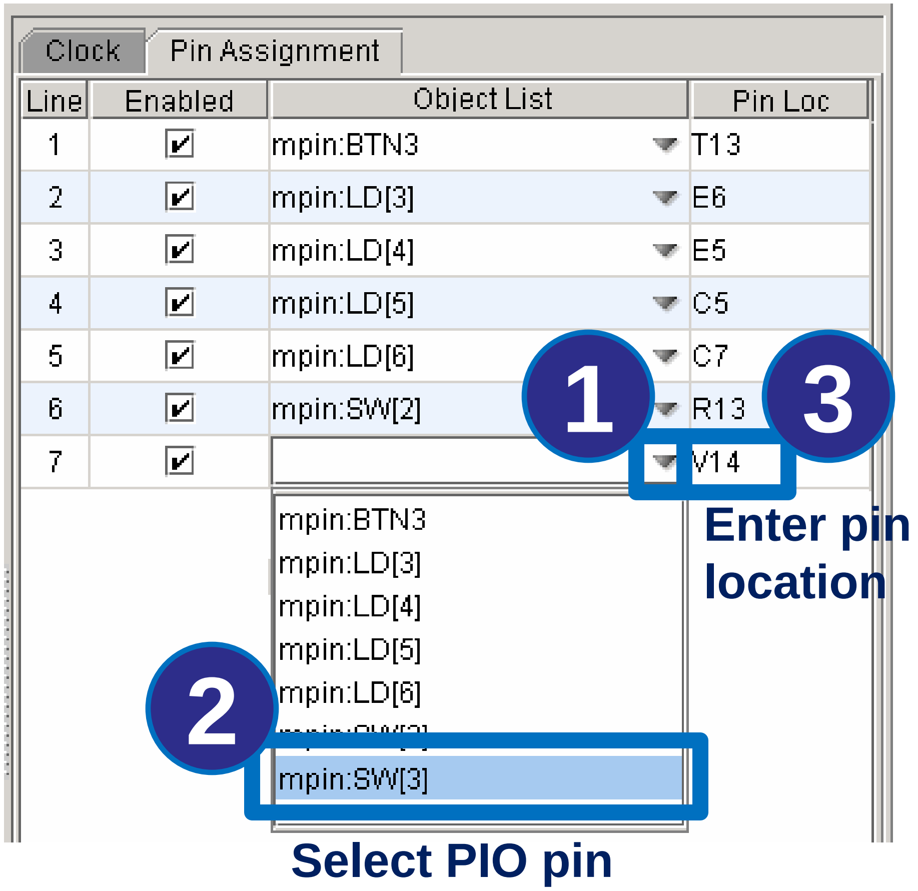


Click Run Logic Synthesis to generate I/O pins

Click Pin Assignment

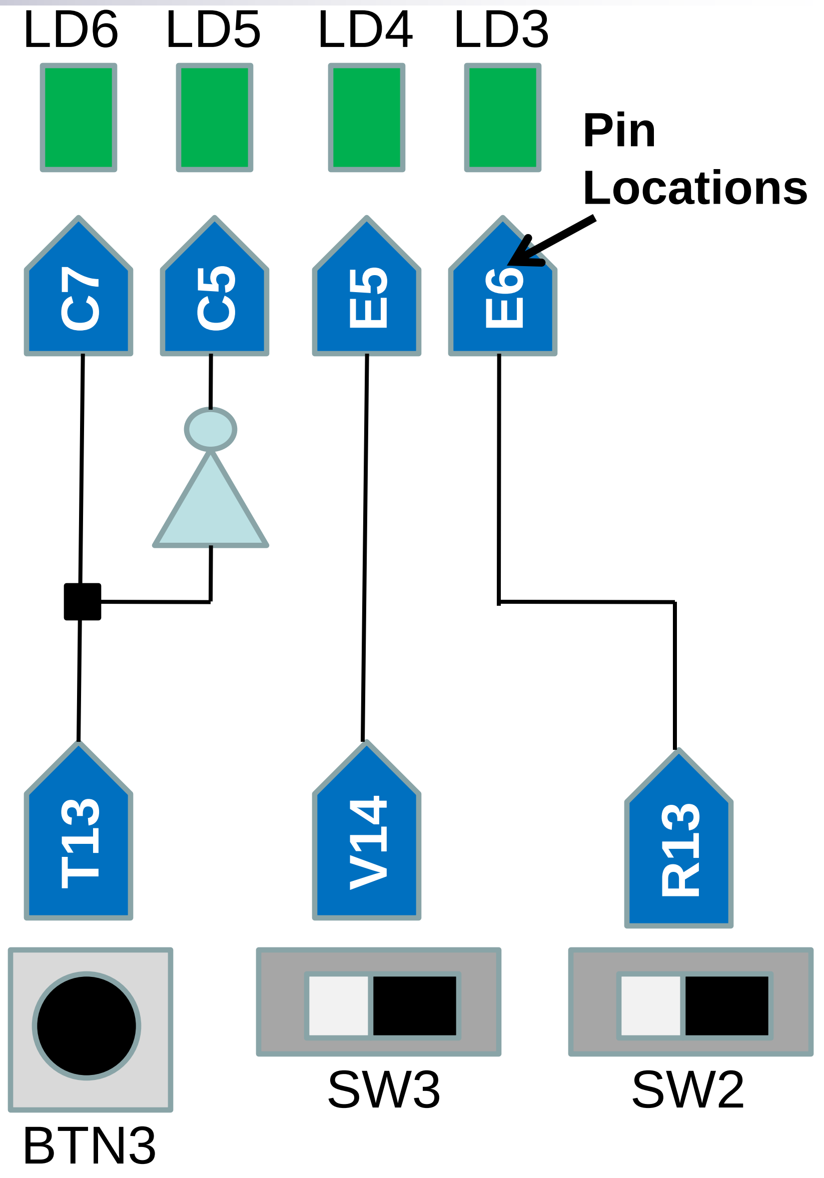
No clocks to define. Click Continue

# Assign PIOs



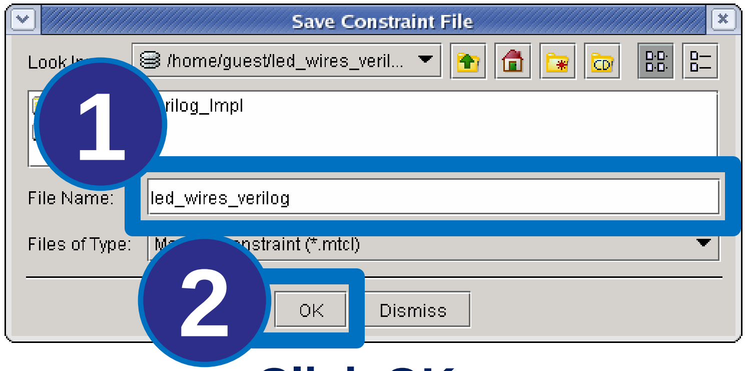
Clock | Pin Assignment

| Line | Enabled | Object List | Pin Loc |
|---|---|---|---|
| 1 | ✓ | mpin:BTN3 | T13 |
| 2 | ✓ | mpin:LD[3] | E6 |
| 3 | ✓ | mpin:LD[4] | E5 |
| 4 | ✓ | mpin:LD[5] | C5 |
| 5 | ✓ | mpin:LD[6] | C7 |
| 6 | ✓ | mpin:SW[2] | R13 |
| 7 | ✓ |  | V14 |



# Complete Design

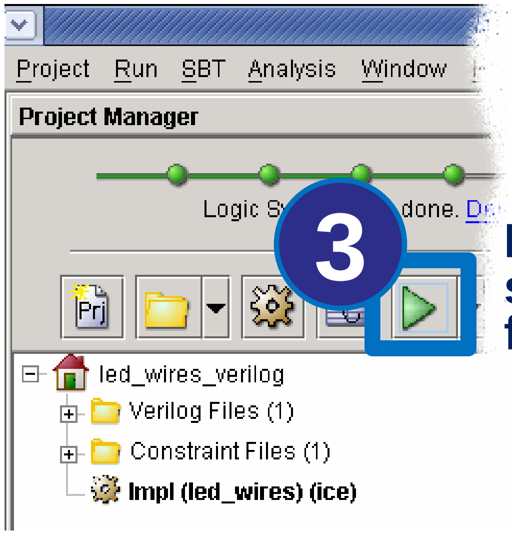
## Enter constraints file name (*.mtcl)



Click OK

## 4 Check output file that expected amount of logic was created

```
Info: Design statistics for front-end :
        Number of FFs:                          0
        Number of LUTs:                         1
        Number of RAMs:                         0
        Number of combinational nodes:          1
        Number of literals (SOP):               1
```

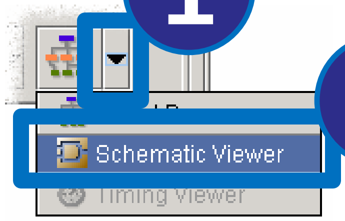


Run full synthesis flow

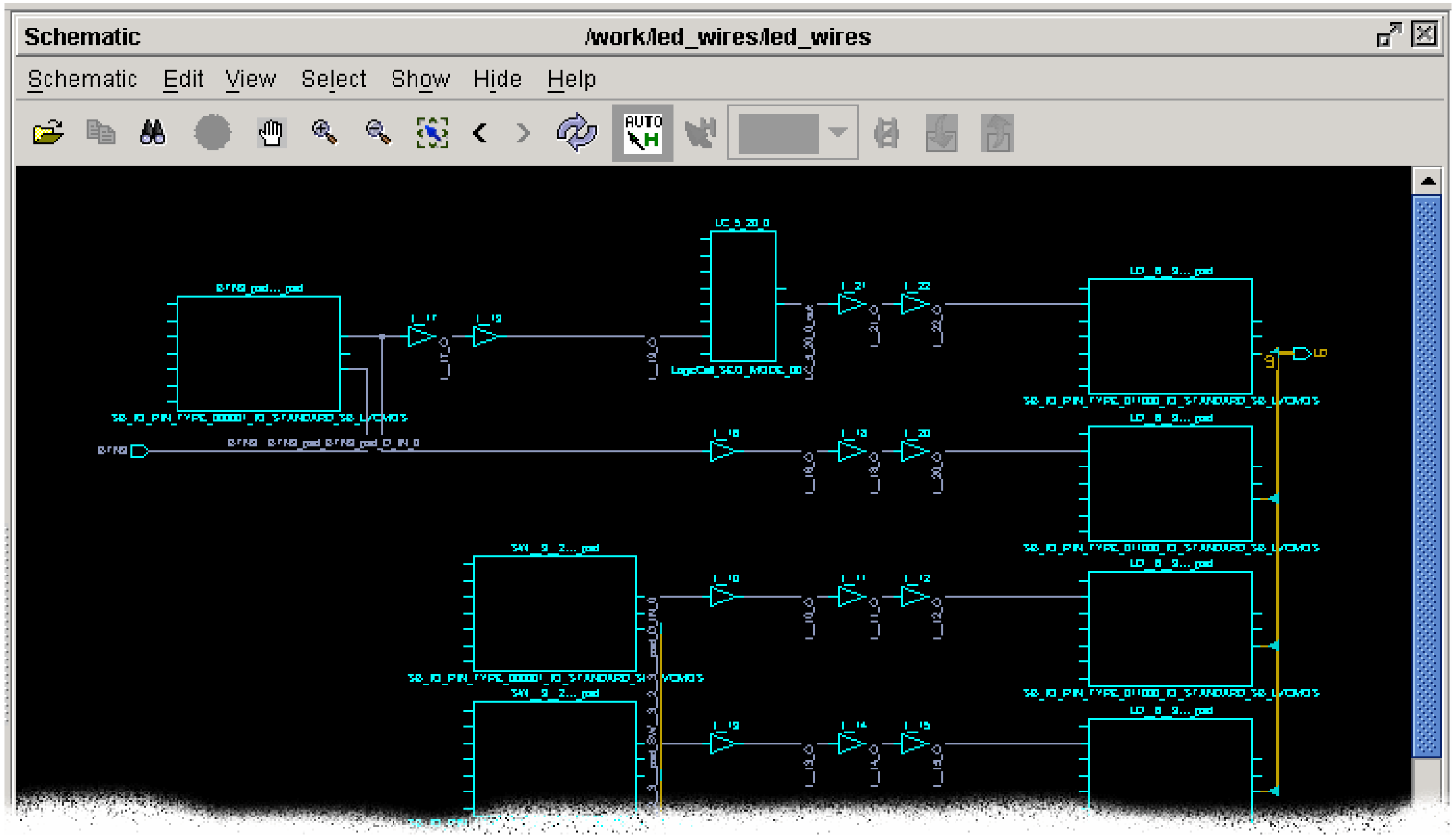
## 5 Check for successful design message in Output window.

```
FPGA-24 FPGA Flow successful
```

# View Final Results



1. Click Drop-list
2. Click Schematic Viewer
3. Check that schematic matches expected logic


Schematic
/work/led_wires/led_wires
Schematic Edit View Select Show Hide Help
AUTO H

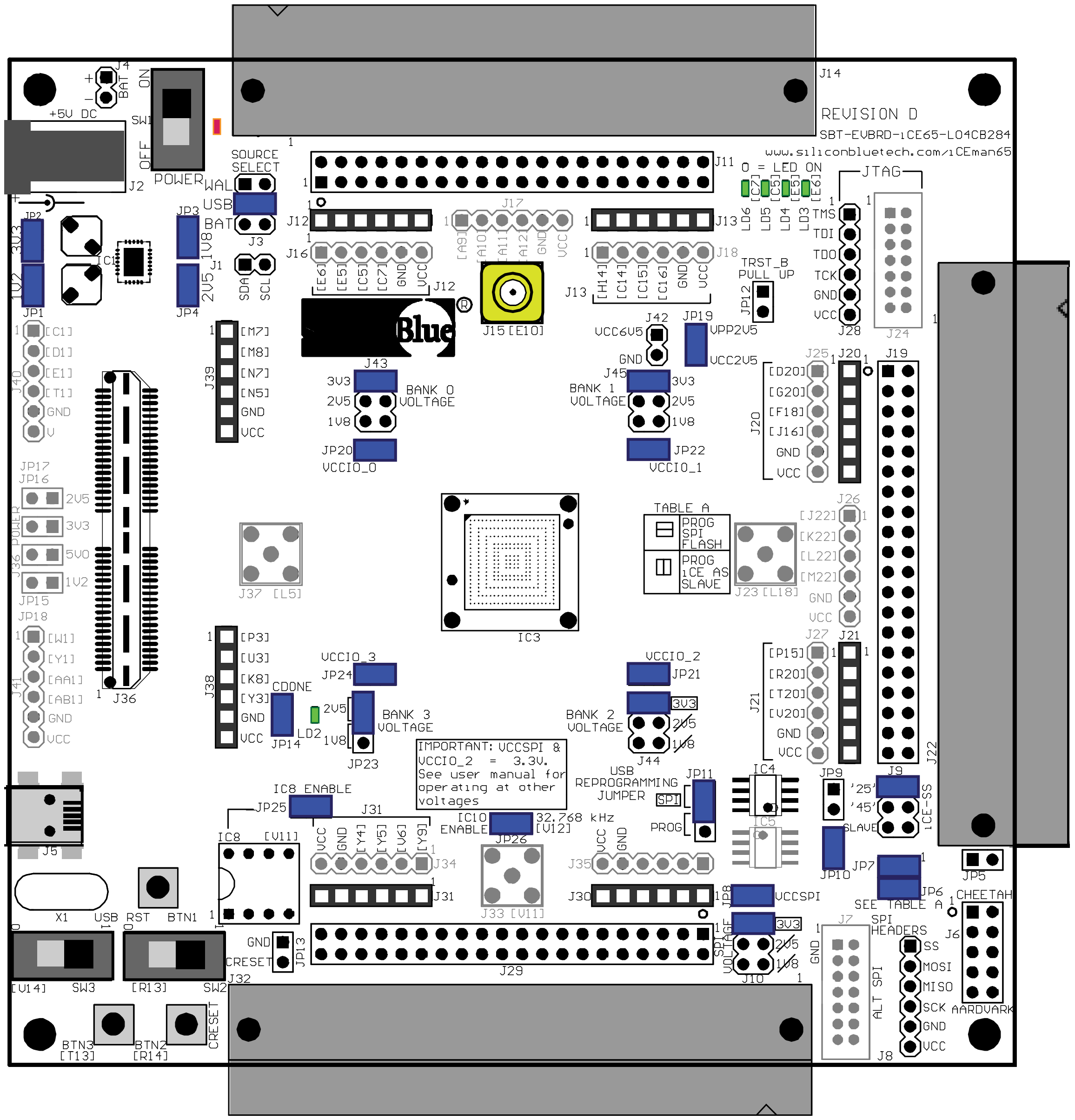
# Important Files Created

| File | Location | Description |
|---|---|---|
| `<proj_name>.prj` | . | Project file |
| `*.v` | . | Verilog source file |
| `*.vhd, *.vhdl` | . | VHDL source file |
| `*.mtcl` | . | Constraints file |
| `blastfpga.log` | ./<proj_name>_Impl | Blast FPGA log file, all outputs |
| `<proj_name>_bitmap.hex` | ./<proj_name>_Impl/sbt/outputs/bitmap | Raw hexadecimal configuration image |
| `<proj_name>_bitmap_int.hex` | ./<proj_name>_Impl/sbt/outputs/bitmap | Intel-format hexadecimal configuration image |

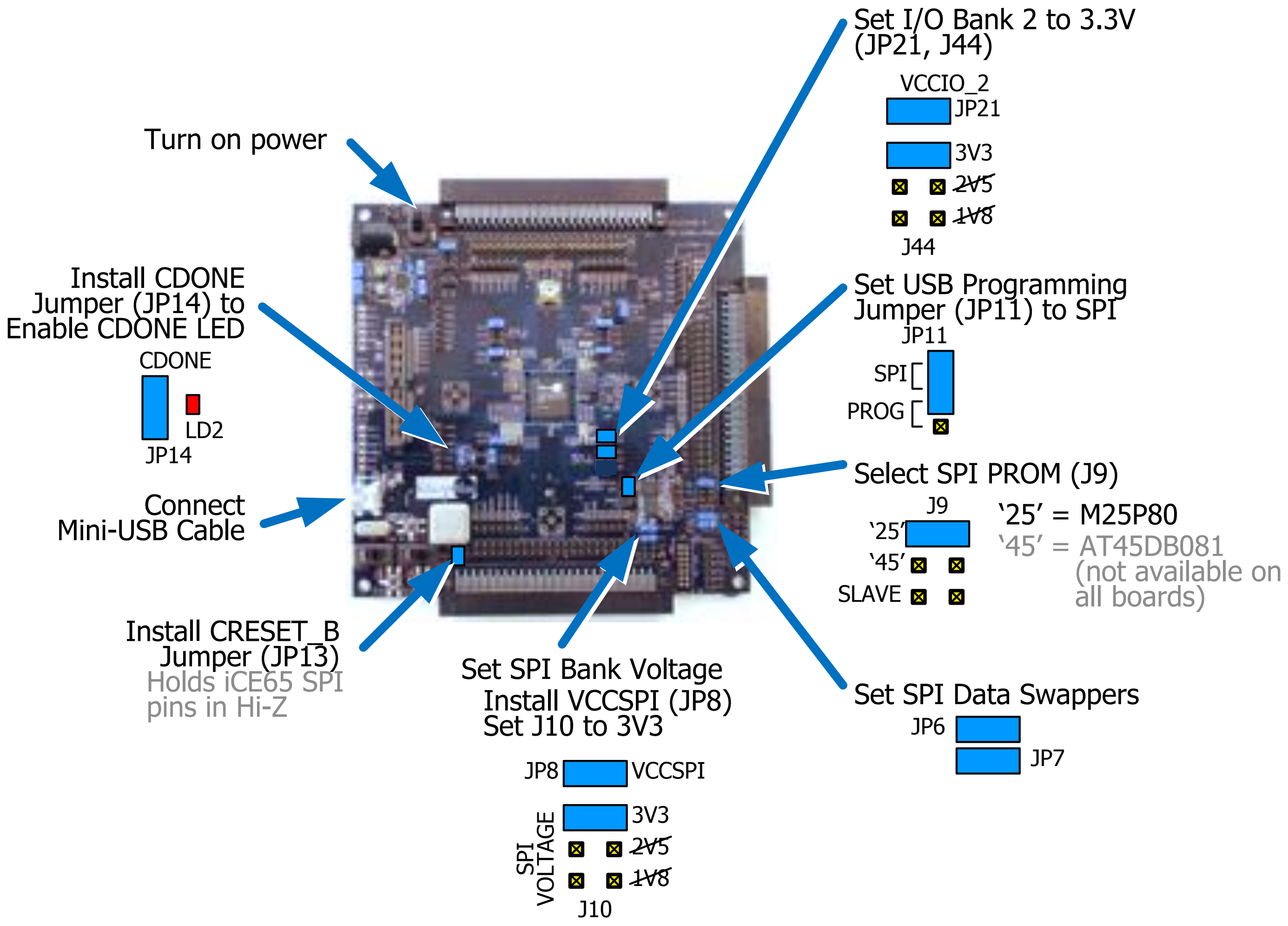
## Try It Out in Silicon!

- Check jumper settings on board (see next slide)
- Connect USB cable to board and PC
- Insert jumper on JP13
- Turn on power
- Open a DOS box or command window
- Use ICEUTIL to program SPI PROM with configuration image
- Remove jumper JP13
- CDONE LED should light
- LEDs should respond to switches

# Default Jumper Settings



# Programming Setup



- Be sure that jumper JP13 is installed to hold CRESET_B Low!

# ICEUTIL Example

- Project creates two configuration images
  - `led_wires_bitmap.hex` : raw hex file format
  - `led_wires_bitmap_int.hex:` Intel hex file format
- Program M25P80 PROM with Raw Hex

  -
    ```
    iceutil -d iCEman65 -m m25p80 -fh
    -w led_wires_bitmap.hex -v
    ```

**OR**

- Program M25P80 PROM with Intel Hex

  -
    ```
    iceutil -d iCEman65 -m m25p80 -fi
    -w led_wires_bitmap_int.hex -v
    ```
- Remember to remove JP13 when finished
  - CDONE LED should light

## Wrap Up

- So how did you do?
- What problems did you encounter?