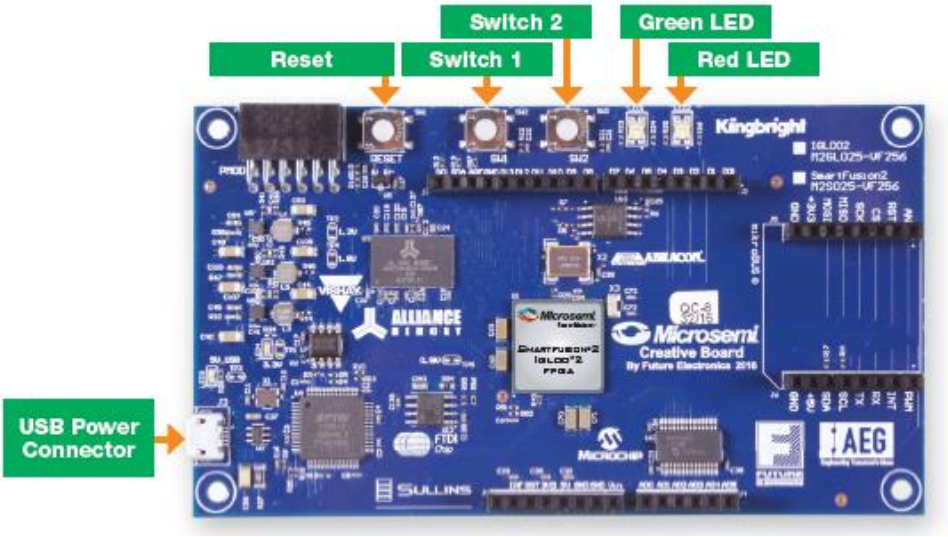
Future Electronics – Microsemi

IGLOO2 Creative Development Board

LAB 2

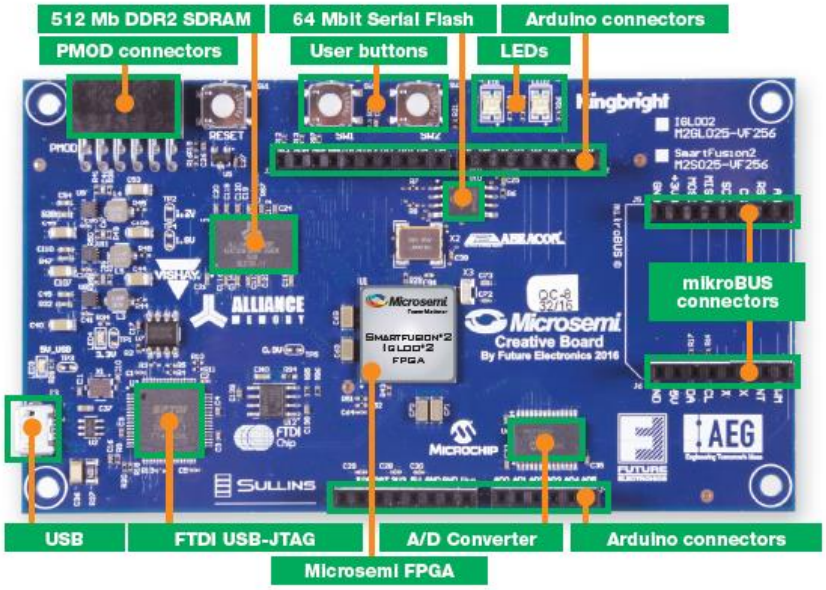
Release - Version A2

The IGLOO2 Board



SW1, SW2, Green LED and Red LED located on top board edge

USB connector
(for power and programming located on left board edge)



Hello and welcome to the Future Electronics Creative Board Lab 2 Instructions.

In order to complete this series of Labs you will first have had to download the Microsemi SoC Libero Development tools and installed them onto your laptop (or desktop).

You will also have to obtain a free license from Microsemi and install it on your machine.

This lab was originally written to work with Version 11.7 SP1 of the Libero tool suite.

Refer to the Revision table at the end of this document for updates and improvements to this Lab set.

INSTALL THE LAB

If you have not already obtained a copy of the Labs via other sources, please copy the **Future** folder from the USB stick onto the Root of your C: (or D:) drive.

When installed, you will have the following folder paths on your computer:

C:\Future\FPGA\Microsemi\CreativeBoard\IGLOO2\Source

C:\Future\FPGA\Microsemi\CreativeBoard\IGLOO2\Lab2_VHDL

C:\Future\FPGA\Microsemi\CreativeBoard\IGLOO2\Lab2_ver

C:\Future\FPGA\Microsemi\CreativeBoard\IGLOO2\Lab3

If you have just completed Lab1 and wish to simply press onward you will also have the following folder path on your computer.

C:\Future\FPGA\Microsemi\CreativeBoard\IGLOO2\Lab1 (It will have either VHDL or Verilog.)

LAB 2 GOALS

This is what you will learn in Lab2:

- ChipPlanner Features
- Post Layout Implementation tools and reports
- Beginner SmartDebug features

Here is a list of the expected results for Lab 2:

- At power up --> LED 1(green) blinks every 0.5 sec and LED 2(red) blinks every 4 sec
- Press and hold SW1 --> LED 1(green) blinks every 0.25 sec and LED2(red) blinks every 4 sec
- Press and hold SW2 --> LED 2(red) blinks every 0.5 sec and LED1(green) blinks every 0.5 sec
- Press and hold SW1 and SW2 --> LED 1(green) and LED 2(red) turn off

Start the Libero SoC tool set by double clicking the ICON on your desktop or pathing to the installed folder and double clicking on the file called **Libero SoC v11.7** or via the **Start Menu.**

Note – If you completed Lab1 – you can just continue working below, or you should go do Lab1 now thru the **New Project Create, Use Advanced Constraints Flow, and Constraints Manager** selection (page 18) or you can open the **Lab2_VHDL** or **Lab2_ver** design via the **Project -> Open Project** and path to either the Lab2_VHDL or Lab2_ver folder and double-clicking on the appropriate **.prjx** file.

Modify the Code

First we will adjust the Blink Rates in the code from Lab1 for use in Lab2.

We will modify the code by adjusting some values so that it will run at a new desired speed. The code fragments have already been written for you. All you will need to do is comment out certain lines and uncomment others.

In the left window, change from the **Design Flow** tab to the **Design Hierarchy** tab.

Double-click the **LEDBlinkingDSpeed** source file to open it for editing (VHDL code shown).

```
81
82  ------------------------------------------------------------------------------------------------------------------------
83      -- Scale Factor used to simulate for Lab1
84      -- They are shorter times to make the simulation go quicker
85      -- Comment these if you want to make real hardware
86      signal Scale_Factor0 : std_logic_vector(31 downto 0) := "00000000000000000000001111101000"; --every 20 us
87      signal Scale_Factor1 : std_logic_vector(31 downto 0) := "00000000000000000000000011111010"; --every 5 us
88      signal Scale_Factor3 : std_logic_vector(31 downto 0) := "00000000000000000000000111110100"; --every 10 us
89  ------------------------------------------------------------------------------------------------------------------------
90      -- Scale Factor used to program the board for Lab1
91      -- They provide longer delays so you can see the lEDs blink
92      -- Comment these factors if you want to simulate
93      --signal Scale_Factor0 : std_logic_vector(31 downto 0) := "00000101111101011110000100000000"; --every 2 sec
94      --signal Scale_Factor1 : std_logic_vector(31 downto 0) := "00000001011111010111100001000000"; --every 0.5 sec
95      --signal Scale_Factor3 : std_logic_vector(31 downto 0) := "00000010111110101111000010000000"; --every 1 sec
96  ------------------------------------------------------------------------------------------------------------------------
97      -- Scale Factor used to program the board for Lab 2
98      -- These change the Blink rates from Lab1 for Lab2
99      -- Comment these factors if you want to simulate
00      --signal Scale_Factor0 : std_logic_vector(31 downto 0) := "00001011111010111100001000000000"; --every 4 sec
01      --signal Scale_Factor1 : std_logic_vector(31 downto 0) := "00000000101111101011110000100000"; --every 0.25 sec
02      --signal Scale_Factor3 : std_logic_vector(31 downto 0) := "00000001011111010111100001000000"; --every 0.5 sec
03
04
05      -- component
06  component Reset_out
```

Scroll down the file until you find the above code.

comment out **signal Scale_Factor0**, **signal Scale_Factor1** and **signal Scale_Factor3** under the comment *"Scale Factor used to simulate for Lab1"* or *"Scale Factor used to program the board for Lab1"*

uncomment **signal Scale_Factor0**, **signal Scale_Factor1** and **signal Scale_Factor3** under the comment *"Scale Factor used to program the board for Lab2"*

There is a fast way to do this to multiple lines of code as follows;

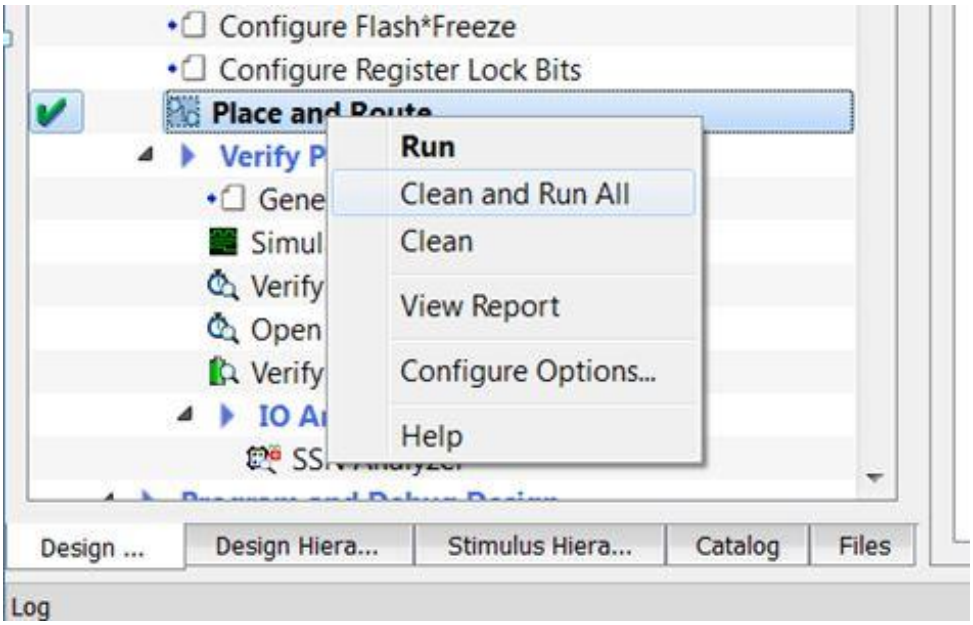
Select multiple lines in the editor, *right-click* and select **comment selection** or **uncomment selection** to make the edits quickly.

Save the file as the existing file name (overwrite it if asked).

(Re)synthesize & **Place and Route the Design** all in one step.

Back in the **Design Flow** tab, find and *Right-click* **Place and Route** and *select* **Clean and Run All**.

This will **rerun** the flow through Synthesis (due to updated source code) and rerun **Place and Route**.



Your screen should look something like this depending on if you started fresh or continued on from Lab1.

Once Synthesize and Place and Route are complete, we will go take a look at how the tools placed the design.

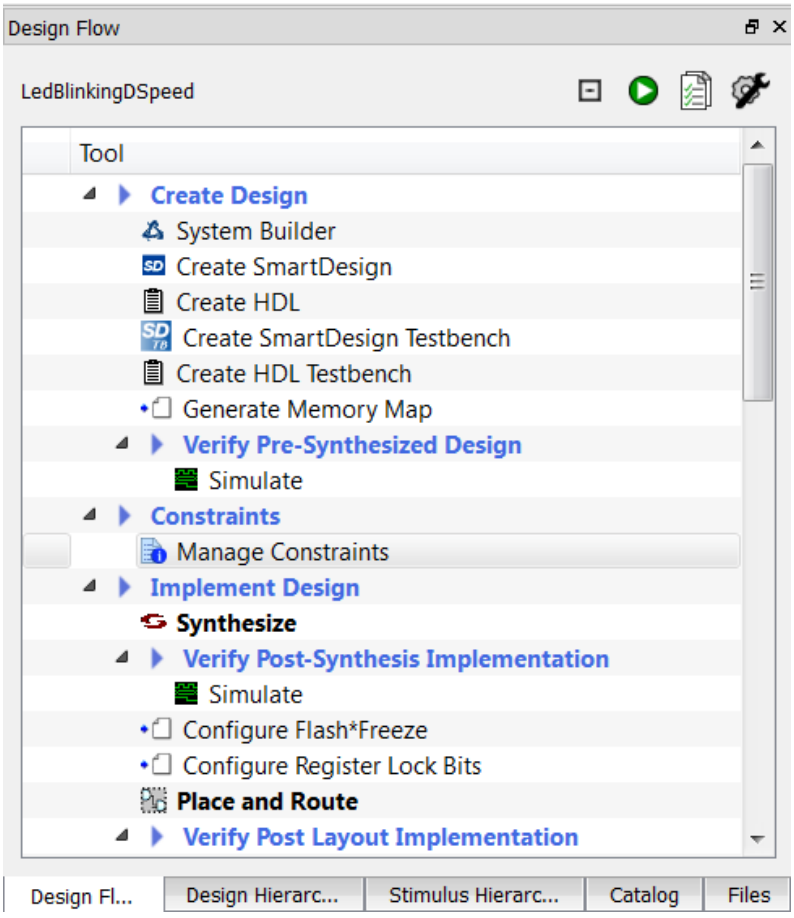
Chip Planner

For a basic design like we are using, this tool is not really needed, but if you are going to be doing a little more advanced designs, or if you want to do a little bit of a design now and then add more later, there are some nice capabilities in the Chip Planner tool.

We will launch the Chip Planner tool and take a look around inside, then we will set a restriction on the area that Place and Route can use on the die, and rerun Place and Route and inspect again to see what changed.

Because we have selected to use the Advanced Constraints flow, we will access Chip Planner via the Manage Constraints tool.

Click on the **Design Flow** tab.

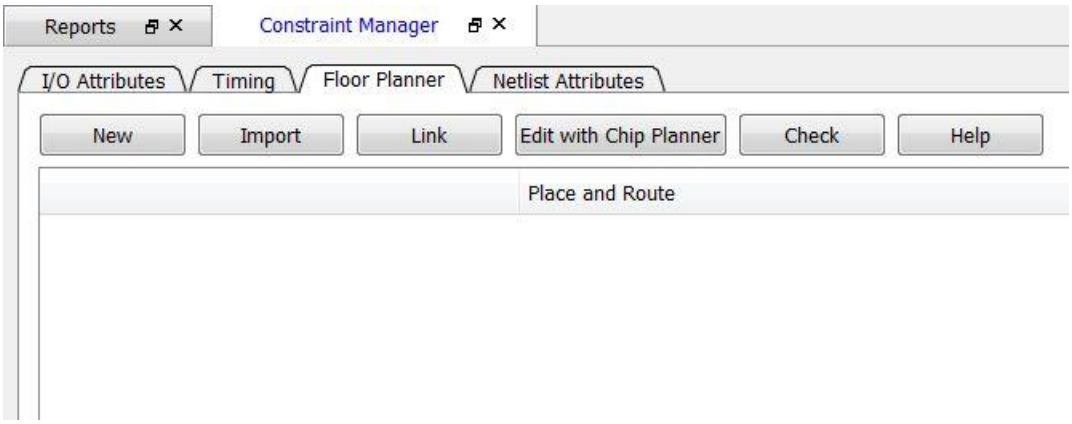


It should look something like above.

Double-click on **Manage Constraints** under **Constraints**.

The Constraints Manager window will open up in the window to the right.

Select the **Floor Planner** sub tab.

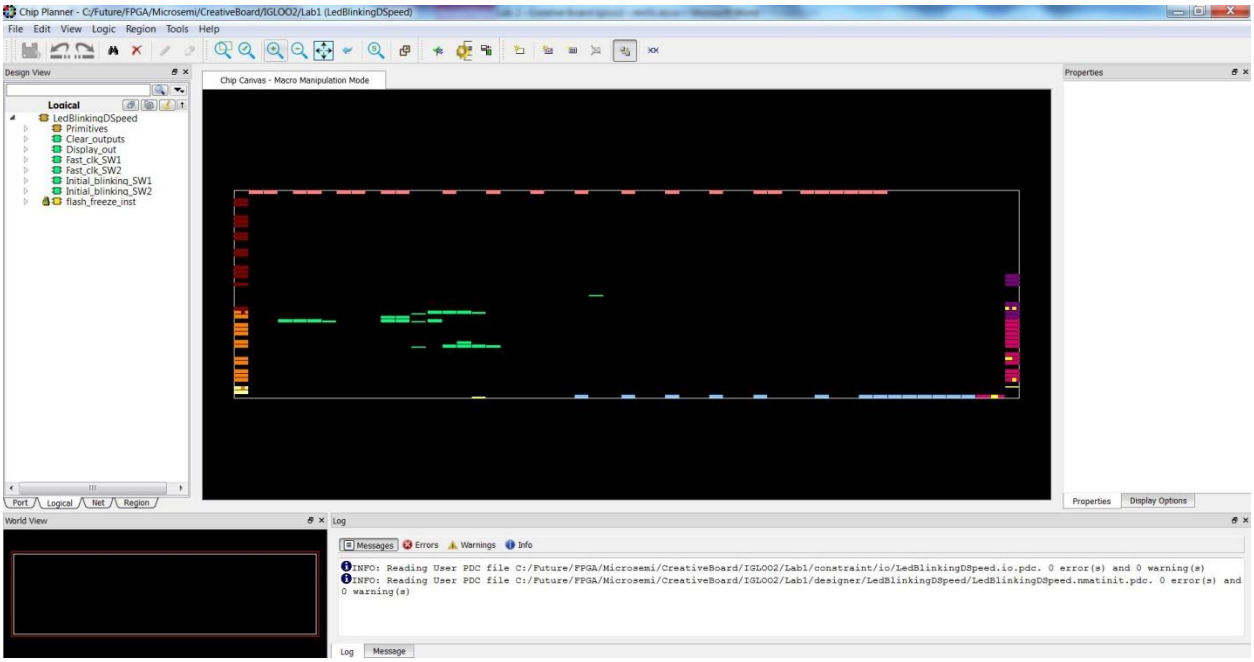


From this interface, you can author new placement constraints as well as import them from other projects from work you may have already done before.

For this lab we will dive right in and take a look at what the Place and Route phase already did for us with the design.

Click on **Edit with Chip Planner**.

The Chip Planner splash box will open momentarily, then the screen will get filled with the Chip Planner interface. And its ICON will be on the Task Bar.



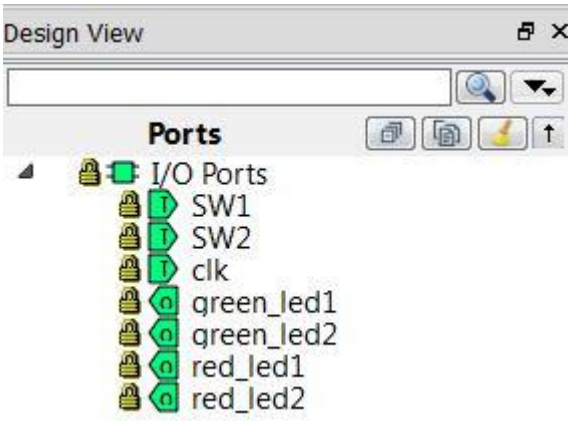
Let’s take a quick look around the interface.

On the left side we can look at different views of the design elements.

Click on the **Port** tab at the bottom of the **Design View** window.

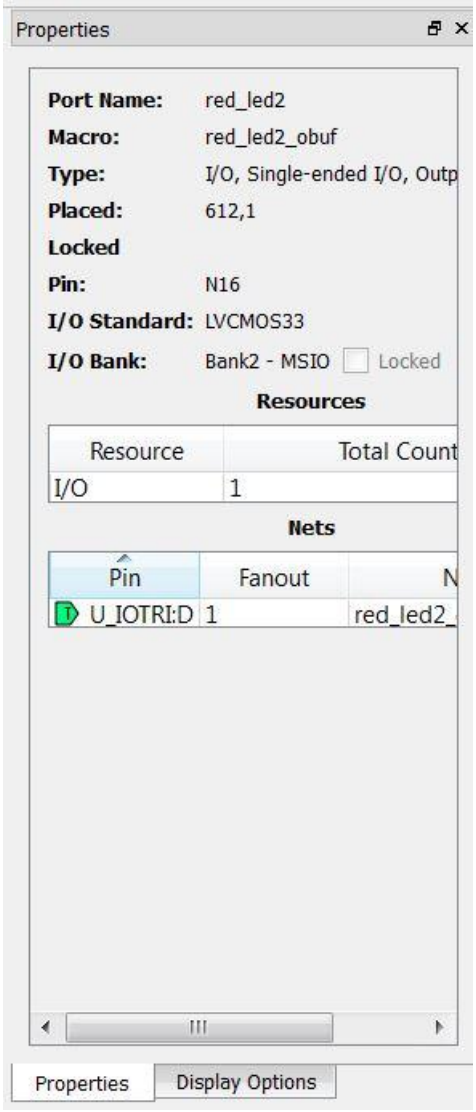
Then expand the **IO PORTS** so we can see all the I/O of the design.

It should look like this.



Click on the different I/O ports and notice a few things are happening in the other windows.

On the right hand side click on the Properties tab at the bottom of that window.



The right window will display attributes of the IO selected.

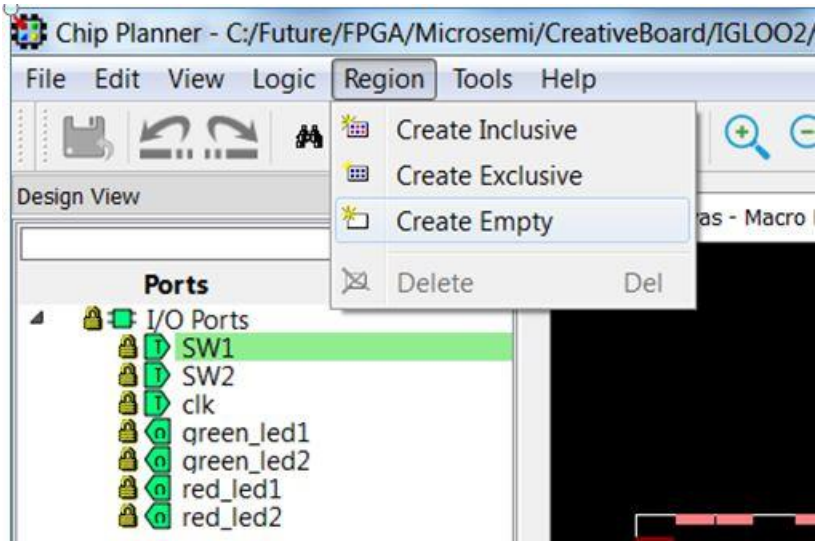
And look at the World window in the bottom left.

Can you see the little ‘dot’ moving around the window based on which IO you select?

In the central larger window, can you see the IO change color when you select different ones?

Take a brief look around the windows, right-clicking here and there, and hovering over the buttons at the top to explore what they do.

Now we are going to create a placement constraint, or more correctly, a non-placement constraint to force the Place and Route engine to not use that portion of the die. It is formally called an **Empty Region**.



In the top Pulldown Menu bar, Click on Region and the Create Empty.

The curser will change to an interesting looking 8 spoked star and you can select a region of the die to set as an Empty Region. In my design most of the logic was placed in the left half of the die. So let's select the entire left hand side (not including the IOs or the clock drivers in the middle) to be our Empty Region. This should force Place and Route to put the resources somewhere else in the part.

It should look like the screen capture below:

Right-click on the **EmptyRegion1** and *rename* it to **FutureModem**. (No spaces please)

Save the design and *close* the window.

Back in Libero, *double-click* on **Place and Route** and let's see if the tool followed our requested keep out region.

When P&R is complete, go back to the **Constraints Manager** and reopen the **Chip Planner** by *clicking* on **Edit with Chip Planner**.

Well look at that, all of our design has now moved over to the *right* half of the chip.

In the lower right hand side, *click* on the **Region** tab, and then *click* on the newly named **FutureModem**.

The right side resource window will give some statistics about the empty region.

If you are adventurous, try making another empty region and then rerun P&R and see if you can force the tool to get a 'no route' due to lack of resources. You might need to go back into the Chip Planner a few times to get it right. When done, make sure that you delete all but the FutureModem region, and rerun P&R so that you have a routed design.

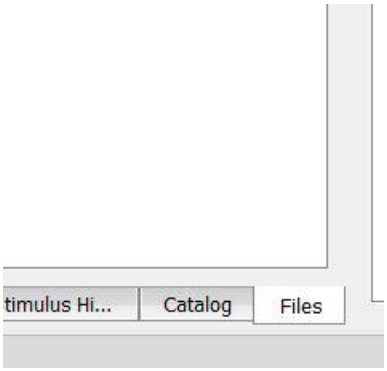
Close the **Chip Planner** window.

Close the **Constraints Manager** tab.

So where exactly did that constraint get saved by Libero?

Let's go find it.

In the left window at the bottom, *click* on the **Files** tab.



I purposely did not show you the top of the new window so that you can go explore a little bit.

Where do you think the **f**loor**p**lanner constraint file is located? _____

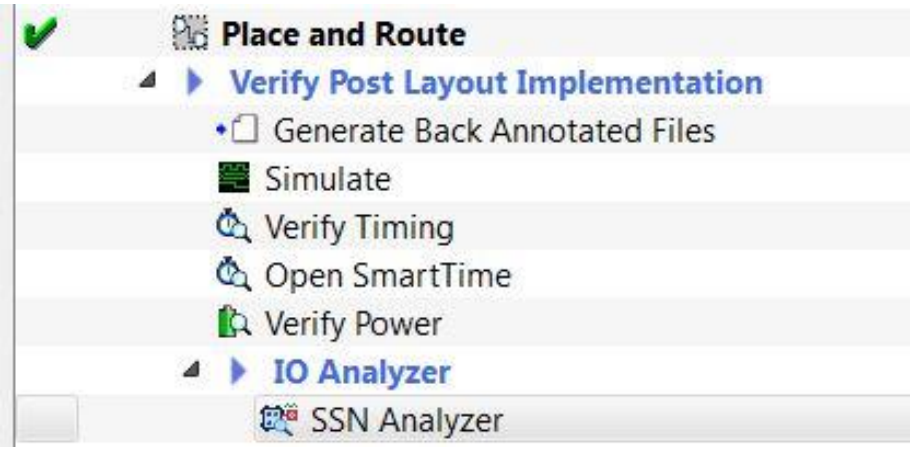
And what file(s) are created by the tool? _____

Give up? I gave you a few hints above.

Now we are going to take a look at a few of the reports and tools to help us analyze a design after it has been Placed and Routed.

Switch back to the **Design Flow** tab.

Reviewing Post Layout Project Tools and Reports



Generate Back Annotated Files

When using 3rd party tools with Libero, this process will create the needed files necessary to pass timing and other information back to those tools. You would use this if you were using a version of ModelSim standalone that you had licensed directly from Mentor Graphics. You do not need to do this step because with the integrated version of ModelSim ME, the needed files are automatically generated when you run the Simulator after Place and Route. The files (LedBlinkingDSpeed_ba.v(hd) and LedBlinkingDSpeed_ba.sdf) can be seen on the Files tab (designer > LedBlinkingDSpeed).

Post Route Simulation

You should verify your design with all the timing delays associated with placing and routing your design into the target part. We could make you do that here, but remember, we changed the length of time for all the delays to something longer so we could see the LEDs blinking, so we will save time in this lab by not making you run a simulation again with a different testbench file.

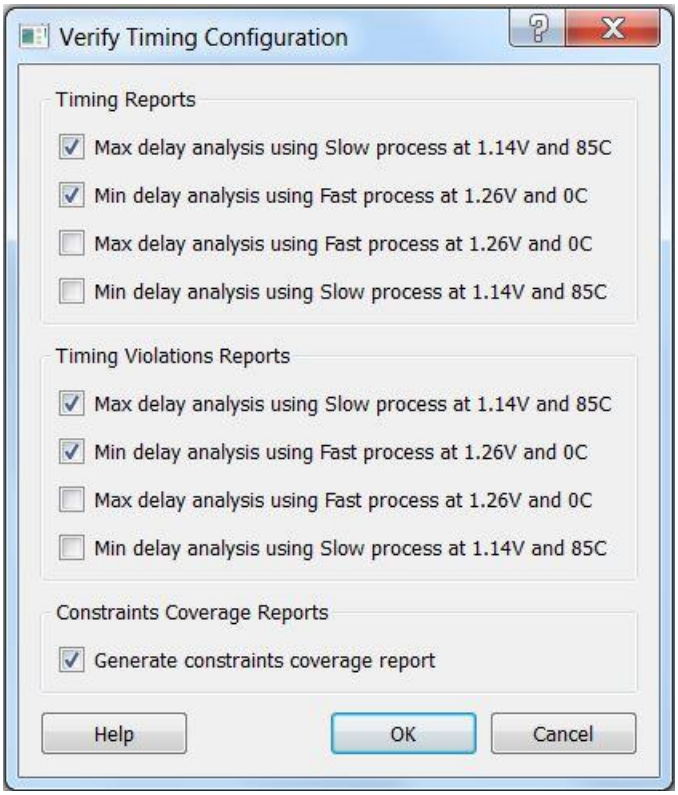
Verify Timing

The Verfiy Timing process helps us understand more about the timing of the design.

Right-click on **Verify Timing** and *click* on **Configuration Options.**

An overlay window will pop up and present you with options you can choose to run.

Take a look at the screenshot for all of the reports that can be run.



As you create more advanced designs, this is how you can tell your design will work under desired conditions. *Click* **Cancel** when you are done looking.

Double-click on **Verify Timing** and take a look at some of the reports it produces.

Notice that the reports that were checked in the Verify Timing Configuration options menu are checked Green and those that were not selected are marked with a "?"

Take a look at a few of them to better understand what each one is telling you.

Verify Power

Here is an excerpt from the information you can read if you right-click on Verify Power and select HELP.

Welcome to SmartPower, the Microsemi SoC state-of-the-art power analysis tool. SmartPower enables you to globally and in-depth visualize power consumption and potential power consumption problems within your design, so you can make adjustments – when possible – to reduce power.

SmartPower provides a detailed and accurate way to analyze designs for Microsemi SoC FPGAs, from top-level summaries to deep down specific functions within the design, such as gates, nets, IOs, memories, clock domains, blocks, and power supply rails.

Double-click on **Verify Power** and take a look at the report that it generates.

How much power does this design consume? Total _____ Static _____ Dynamic _____

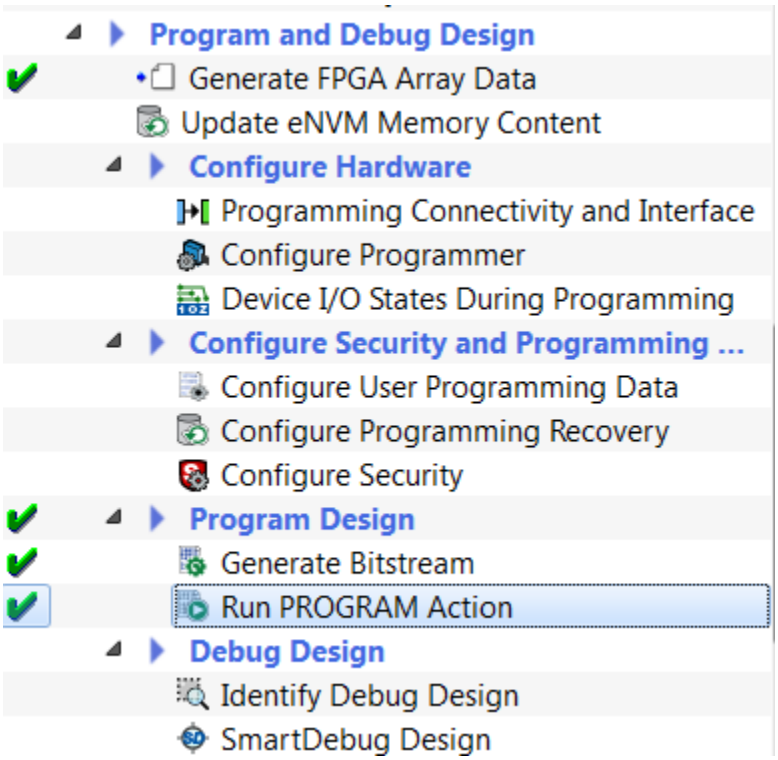
Let's go program our modified design and see what has changed.

Program the Device

Under **Program Design**, *Double-click* on **Run PROGRAM Action** to program the IGLOO2 device on the board. Notice we will actually execute two steps at once here. First the tool will run the **Generate Bitstream** process then it will run the **Program Action.**

(If your Laptop went to sleep, remove and insert the USB cable because the USB driver went to sleep.)

(If you cannot find **Run PROGRAM Action** – *Right-Click* on **Run xxxxx Action** and *Click* on **Configure**.)



The Programmer will begin to verify that a cable is connected, Blank the Device, and Program it.

After about 85 seconds (give or take), you should notice a **Green check mark** will be placed on **Program Design** and associated 2 sub-actions.

The LEDs should now be blinking different than in Lab1.

Go back and reread what you are expecting the LEDs to be doing, and verify your results.

Press some buttons – do you remember which one should do what?

What does SW1 do? _____

And what does SW2 do? _____

Can you get them both to blink together?

Introduction to SmartDebug

Design debug is a critical phase of FPGA design flow. Microsemi's SmartDebug tool complements design simulation by allowing verification and troubleshooting at the hardware level. SmartDebug provides access to non-volatile memory (eNVM), SRAM, SERDES, DDR controller, and probe capabilities. Microsemi SmartFusion2 System-on-chip (SoC) field programmable gate array (FPGA), IGLOO2 FPGA, and RTG4 FPGA devices have built-in probe logic that greatly enhance the ability to debug logic elements within the device. SmartDebug accesses the built-in probe points through the Active Probe and Live Probe features, which enables designers to check the state of inputs and outputs in real-time without re-layout of the design.

Debugging with SmartDebug

In this lab you will use SmartDebug to view internal circuits in the design.

The SmartDebug tool provides a way to debug the Microsemi FPGA array and SERDES (if your part has them) without using an internal logic analyzer (ILA). SmartDebug provides the following methods to debug the IGLOO2 FPGA array:

- **Live Probe:** Two dedicated probes can be configured to observe a Probe Point which is any input or output of a logic element. The probe data can then be sent to an oscilloscope or logic analyzer. *Note: We will not be looking at this feature in a lab because we did not include the **LIVE_PROBE_FB** macro in the design.*

- **Active Probe:** Active Probe allows dynamic asynchronous read and write to a flip-flop or probe point. This enables a user to quickly observe the output of the logic internally or to quickly experiment as to how the logic will be affected by writing to a probe point.

- **Memory Debug:** Memory debug is used to debug embedded FPGA fabric memories (uSRAM and LSRAM) by reading and writing to the memory block. Memory Debug also supports reading eNVM memory content. *We'll use this capability more in the next lab.*

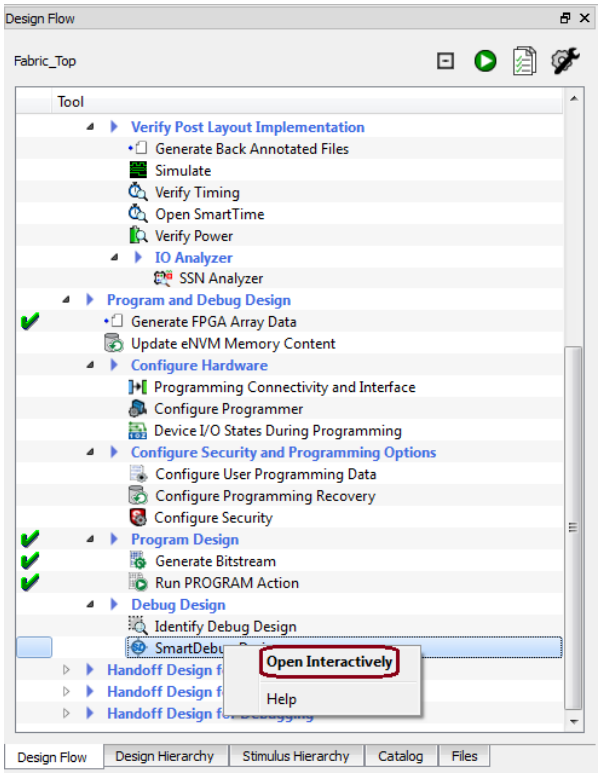
- **Probe Insertion:** Probe insertion is used to insert probes into the design and bring signals out to the FPGA package pins to evaluate and debug the design. *Note: You could use this feature to bring a collection of internal signals out to the Arduino, PMOD, or MikroBUS Connectors and connect a Logic Analyzer or O-Scope.*

- **SERDES Debug**: SmartDebug is used to debug high speed serial interfaces by utilizing on-chip PRBS features and signal integrity controls. *Note: The IGLOO2 mounted on the Creative Board does NOT have SERDES capabilities. You can tell if a part has SERDES capabilities if it has a "T" in the part number.*

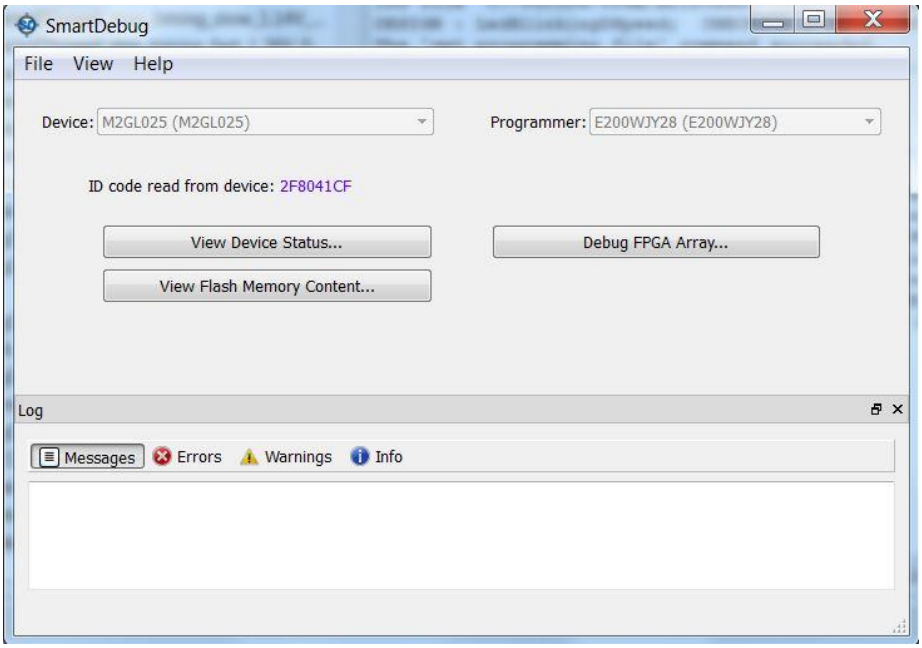
In the next section we will take a look at **Active Probe**, **Memory Debug**, and **Probe Insertion**.

Starting Smart Debug

Expand **Debug Design** in the Libero Design Flow window. Select **SmartDebug Design** then *Right-click* and select **Open Interactively**.

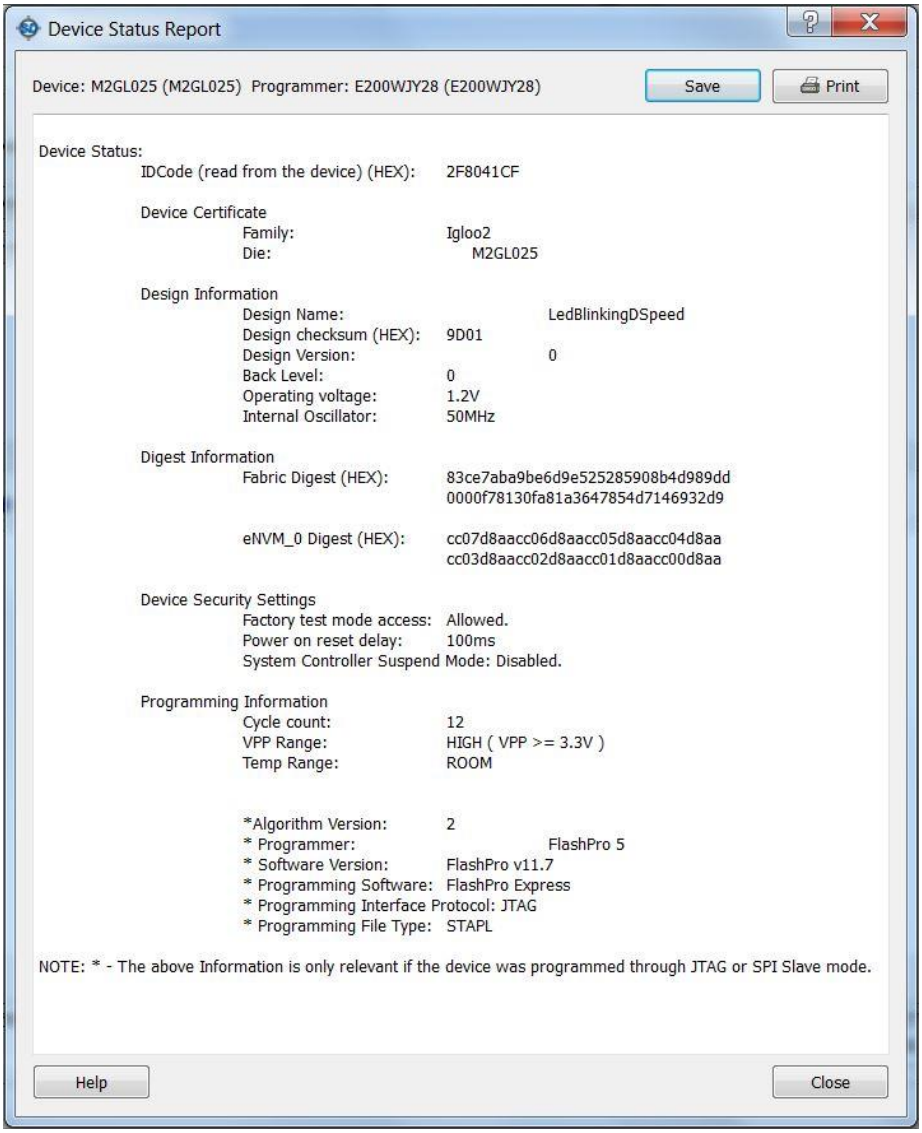


The SmartDebug GUI will open.



Let's take a look at the **Device Status** for the part mounted on your board.

Click on **View Device Status…**



Resize the window and take a look at the information presented.

What is the Design Checksum? _____
The checksum shown is for the programming file only. The Fabric digest and eNVM digest are more interesting. These are cryptographically secure digests of the entire fabric configuration and eNVM blocks that are configured as ROM. The digest can be checked at power up or on demand to confirm that the fabric and eNVM contents have not been tampered with.

Under Programming Information, what is the Cycle count? _____
Are these values different than others doing the lab around you?

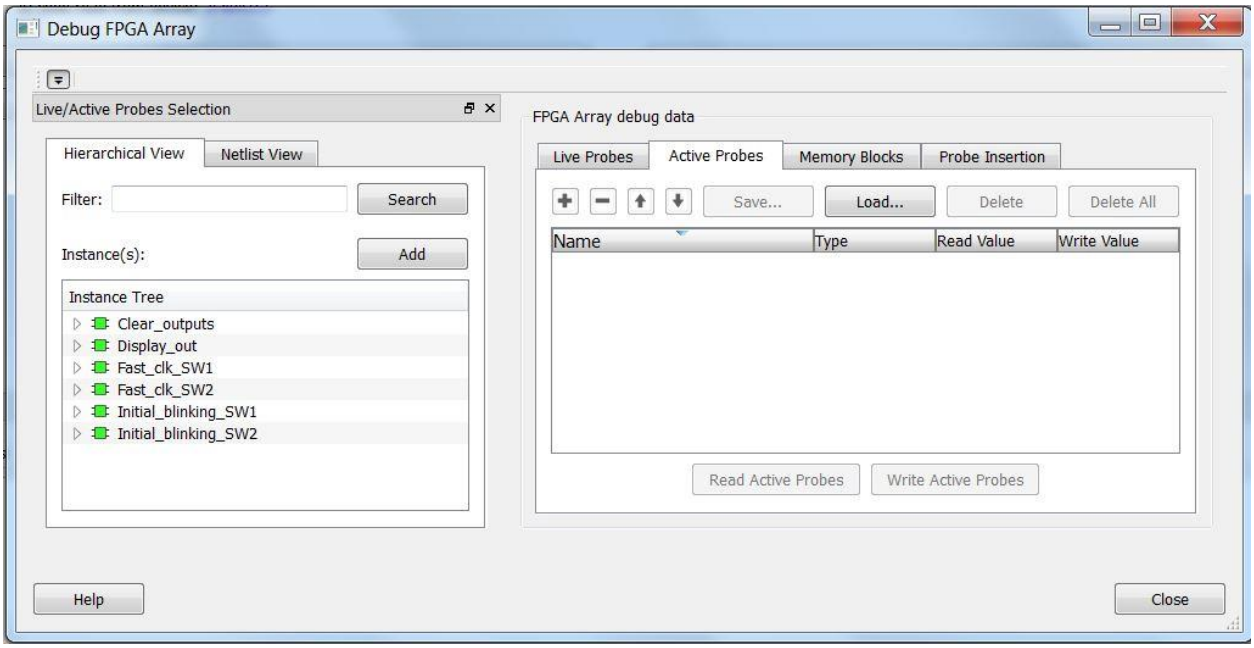
What is the Power on Reset Delay? _____
Does this match the default value you accepted back in Lab1 when you created the design?
When done, **Close** the window.

SmartDebug Active Probes

In this step we will use the Active Probes to take a look at the value of the counters used as delay counts for toggling the blink rate of the LEDs and the signals used to drive the two LEDs.

In the SmartDebug window, *Click* the **Debug FPGA Array…** button to open the window below.

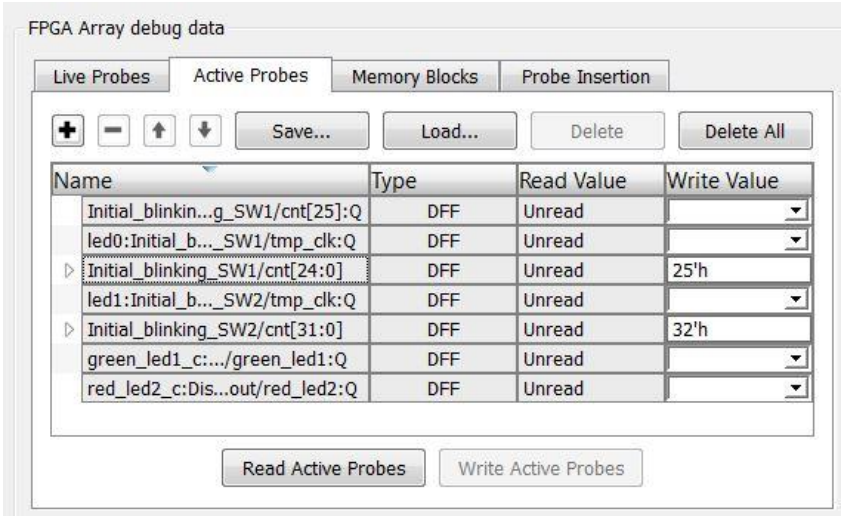
Select the **Active Probes** tab in the **Debug FPGA Array** window. The design hierarchy is displayed in the left window. Signals that you select for probing will be displayed in the right window.



In the **Hierarchical View**, under the **Instance Tree,** select **Initial_blinking_SW1** and *click* **ADD**.

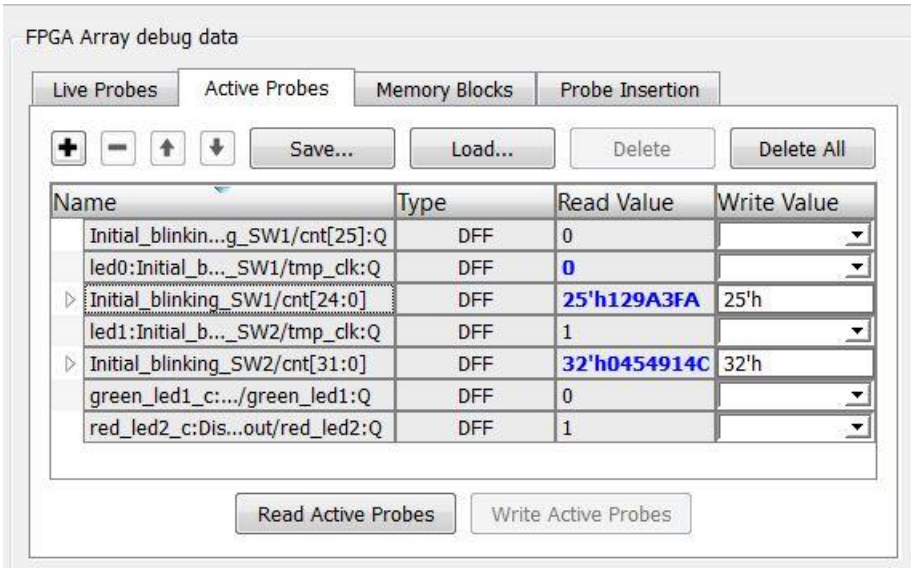
Do the same for **Initial_blinking_SW2** and **Display_out**.

The signals will appear in the Active Probes tab as shown below.



Click the **Read Active Probes** button. The state of the signals will be displayed.

Clicking **Read Active Probes** repeatedly will allow observation of the counter outputs and LED drive values.



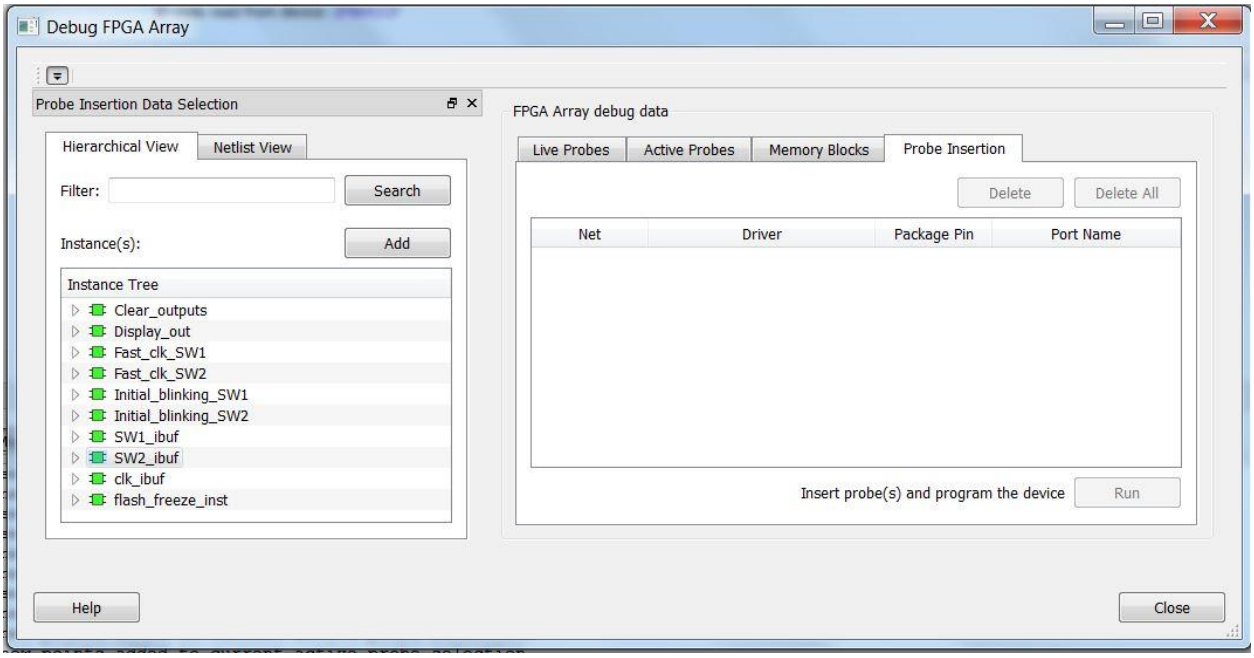
We could write values to these internal elements, but the actively running design would instantly advance the counters and the LEDs would toggle again so it would be hard to see if you had any effect.

In a real design of yours, if something is not working quite right, you could use this capability to force a signal that is not getting asserted to go active (or inactive) and see if the behavior of the design changes.

This might help you determine what is 'not quite working correctly'.

This design does NOT use any internal SRAM so we will skip over the **Memory Blocks** tab.

Select the **Probe Insertion** tab in the **Debug FPGA Array** window. The design hierarchy is displayed in the left window. Signals that you select for probing will be displayed in the right window.

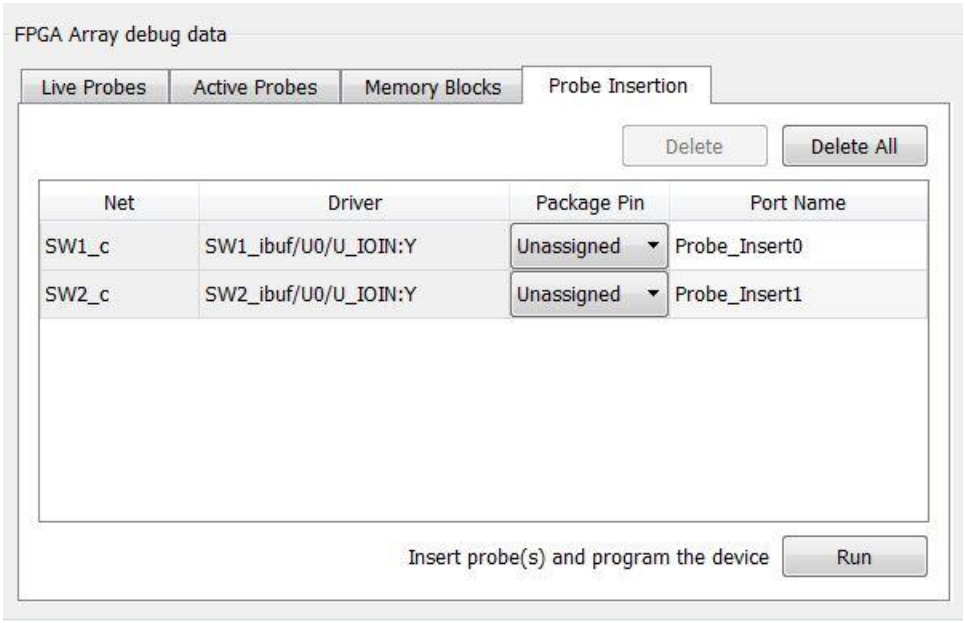


Adjust the size of the window to eliminate the scroll bars by making it a little bit taller and narrowing the width of the Driver column.

In the **Hierarchical View**, under the **Instance Tree,** select **SW1_ibuf** and *click* **ADD**.

Do the same for **SW2_ibuf**.

The signals will appear in the Active Probes tab as shown below.



The LEDs on the board are actually dual color LEDs. Each LED has a Green and Red segment.

Here is a part of the I/O Constraint file (go take a look if you think you know where it is located).

```
set_io green_led1 -pinname J16 -fixed yes -DIRECTION OUTPUT
set_io red_led1   -pinname K16 -fixed yes -DIRECTION OUTPUT
set_io green_led2 -pinname M16 -fixed yes -DIRECTION OUTPUT
set_io red_led2   -pinname N16 -fixed yes -DIRECTION OUTPUT
```

Your design is currently blinking green_led1 (J16) and red_led2 (N16).

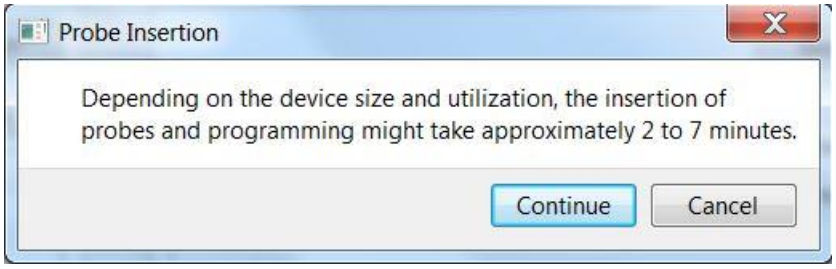
That means that we have pins K16 (red_led1) and M16 (green_led2) available.

Let's assign these two pins to the output of the push button input buffers inside your design.

On the Probe Inset tab,
assign pin K16 to SW1_c by clicking on the word unassigned and scrolling down.
assign pin M16 to SW2_c by clicking on the word unassigned and scrolling down.

Select **Insert probe(s) and program the device**. *Click* the **RUN** button.

A pop-up window will appear. *Click* **Continue**.



OK, so while this is happening, what exactly is going on?

The SmartDebug tool is aware of all the elements and nets in your design. You have selected the outputs of the I/O buffers that the switches are connected to on the board. You then connected them to the output buffers that drive the unused LEDs on the board.

Libero added the connections that you have requested and downloaded that into the part.
Remember, your design is still running.

What do you think is going to happens when you press SW1?

_____

Press SW1. What happened? _____

You should notice two things going on.

Your design makes the left Green LED blink twice as fast just like before.
What else is happening?
Did the left Red LED come on solid?

What do you think is going to happen when you press SW2?

_____

Press SW2. What happened? _____

You should notice two things going on.

Your design makes the right Red LED blink twice as fast, so it is now the same rate as the blinking Green LED just like before.

What else is happening?

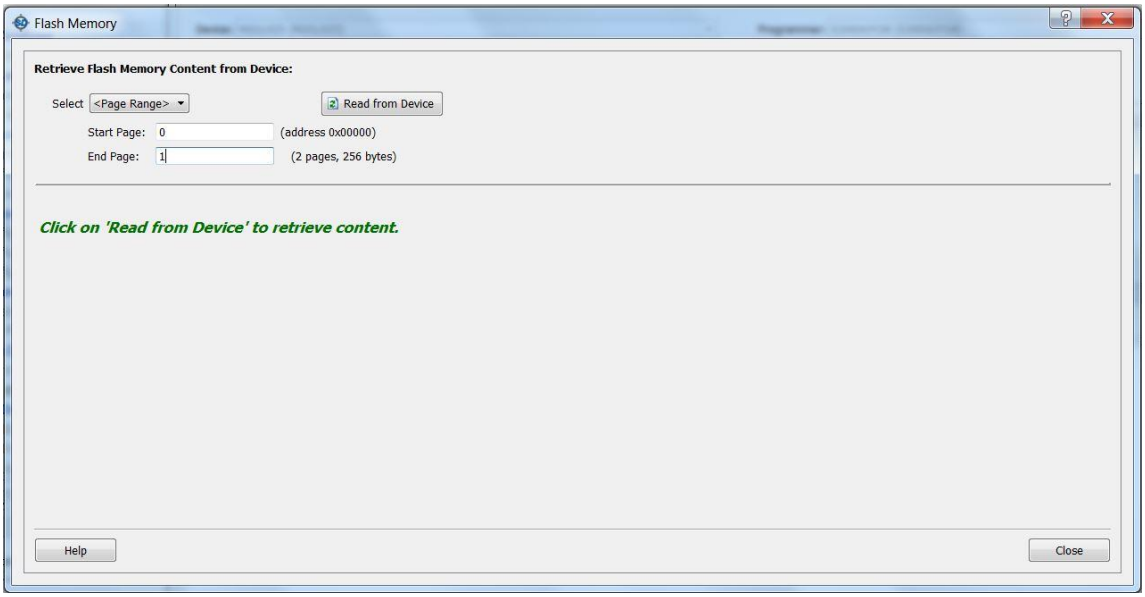
Did the right Green LED come on solid?

What is going to happen when you press both SW1 and SW2 at the same time?

Click **Close** to close the **Debug FPGA Array** window

If you are asked about saving the active probes, *Click* **Cancel**.

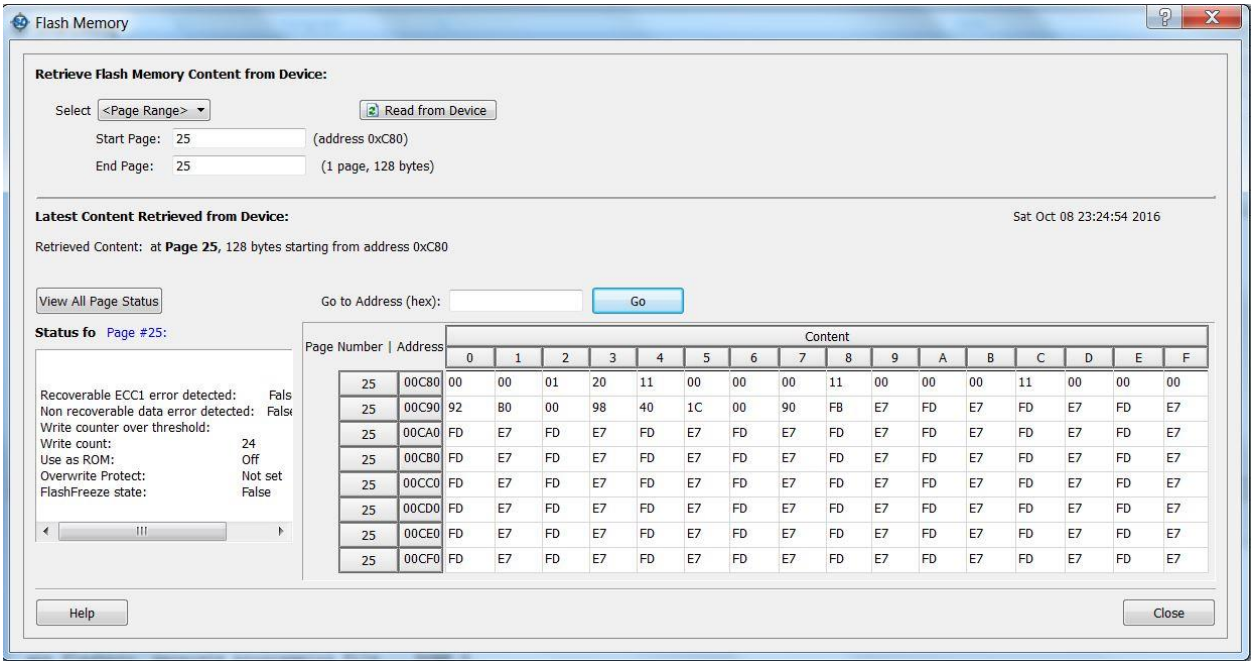
In the SmartDebug window, *Click* the **View Flash memory content…** button to open the window below.



Enter the following:

Start Page: 25 End Page: 25

And *click* **Read from Device**



Resize the window so you can see the full table of data. Knowing how to do this might be useful in Lab3.

Close the **Flash Memory** window. Close SmartDebug (**File > Exit**) and Libero SoC (**Project > Exit**).

Congratulations!!! Lab 2 is now complete.

Revision History:

Initial Release	A	11-Oct-16	Daryl Lee Specter
Correct a few typos on Pages 10 & 22	A1	20-Oct-16	Daryl Lee Specter
Correct a few typos on Page 11, 13, & 22	A2	13-Nov-16	Daryl Lee Specter

NOTES

NOTES

NOTES