Getting Started with Vivado and Vitis for Baremetal Software Projects

Overview

This guide will work you through the process of setting up a project in Vivado and Vitis. A simple hardware design including a processor with several AXI GPIO peripherals connected to buttons and LEDs will be created. This design will then be exported to the Vitis IDE, and a baremetal software project will be created and run which polls the buttons and writes to the LEDs.

Note: Screenshots presented in this guide may not have been taken with your version of the tools. The workflow presented here has been verified in Vivado and Vitis 2020.1.

Inventory

- A Digilent FPGA Development Board
- USB Programming cables, USB UART cables, and Power Supply, as required by the board.
- Vivado and Vitis installations
  - You also need the board files for your board. This guide is intended for use with the board files available from Digilent's vivado-library repo on Github. You can get these files using the process described in the Installing Vivado, Vitis, and Digilent Board Files (https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-vitis) guide. These files will also be available through the Vivado application itself in versions 2022.1 and newer.

Note: If you are using a version of Vivado that includes Xilinx SDK (2019.1 or older), check out Getting Started with Vivado IP Integrator and Xilinx SDK (https://digilent.com/reference/vivado/getting-started-with-ipi/2018.2).

Guide

Launch Vivado

Select the dropdown corresponding to your operating system, below.

Windows

Open Vivado through the start menu or desktop shortcut created during the installation process.

Linux

Open a terminal, and change directory (cd) to a folder where log files for your Vivado session can be placed, then run the following commands:

```
source <install_path>/Vivado/<version>/settings64.sh
vivado
```
Create a Vivado Project

In Vivado's welcome screen, several options are presented:

- **Create Project**: Opens a wizard used to begin creating a Vivado project from scratch, which will be used here.
- **Open Project**: Can be used to open a Vivado project (defined by an XPR file) that has been previously created or downloaded from the internet.
- **Open Hardware Manager**: Can be used to program an FPGA development board with a bitstream, without opening the associated project.

**Note**: Various other options are available, but are not described here.

For the purposes of this guide, click the **Create Project** button.

---

The first page of the New Project wizard summarizes the steps involved in creating a project. Click **Next**.

---

The first step is to set the **name** for the project. Vivado will use this name when generating its folder structure.

**Important**: Do NOT use spaces in the project name or location path. This will cause problems with Vivado. Instead use an underscore, a dash, or **CamelCase**.

Pick a memorable **location** in your filesystem to place the project.

Checking the **Create project subdirectory** box will create a new folder in the chosen location to store the project's files. This is recommended.

Click **Next** to continue.
At the Select Project Type screen, choose **RTL Project** and check the *Do not specify sources at this time* box. Advanced users may want to use the other options on this screen, but they will not be covered here.

Click **Next** to continue.

Next, a part or a board must be chosen for the project to target. The project will only be usable with the chosen device (though the selection can later be changed through the project's Settings).

Selecting a board over a part is recommended, as the board files provide additional configuration information for a variety of peripherals and components in a design. Click the **Board** button to open the board tab.

**Important**: If your board does not appear in the list, Digilent's board files have not been installed. Review Installing Vivado, Vitis, and Digilent Board Files for instructions on installation of these files.

Click **Next** to continue.
The last screen of the New Project wizard summarizes what was chosen in the previous screens. Click **Finish** to open your project.

Create a Block Design

Click the **Create Block Design** button in the **IP Integrator** dropdown of Vivado's **Flow Navigator** pane. A block design provides a visual representation of your hardware design, and can be used to easily connect and configure IP cores.
In the dialog that pops up, give your block design a name (or use the default “design_1”).

**Important:** Do NOT use spaces in the block design name. Instead use an underscore, a dash, or CamelCase. (https://en.wikipedia.org/wiki/CamelCase).

The other two fields should be left as defaults. *Directory* can be used to place the block design's source files outside of the project directory, and *Specify source set* can be used to create block designs that are not part of the normal Design Sources, which are used to build the project.

Click **OK** to continue.

---

**Add a Processor to a Block Design**

The process of adding a processor to your design is very different depending on the processor used. The Microblaze dropdown should be selected only if using a board that does not have a Zynq or Zynq UltraScale+ device.

**Add a Microblaze Processor to a Block Design**

The Microblaze soft-core processor IP can be used to instantiate a processor within your FPGA design. This processor can be very useful for controlling and configuring hardware components. This section discusses how you can add a Microblaze processor and several useful components, including UART for standard output and DDR memory support, to your block design.

**Note:** This section is intended for use with boards without a Zynq chip. For Zynq boards, the Zynq7 Processing System should be used instead.

**Note:** If you aren't sure whether your board has DDR memory, check the Memory column of the specification table on this site’s Programmable Logic page.

**Follow the steps in this dropdown for boards without DDR memory**

First, an external clock should be added to the block design, so that it can be used to generate any other clocks required by the design. Open the **Board** tab, and find the system clock. Right-click on it and select **Connect Board Component**.

---
In the **Connect Board Component** dialog, select the CLK_IN1 of a new Clocking Wizard IP. This will add a clocking wizard to the design, which can be used to easily configure the board's MMCMs and PLLs to generate any required clocks. Click **OK** to continue.

Next, an external reset port should be added to the design that can be used to reset the entire system. Click the **Run Connection Automation** button in the green **Designer Assistance** toolbar.
In the Run Connection Automation dialog, in the list on the left side of the dialog, make sure that the Clocking Wizard's reset box is checked. The Select Board Part Interface dropdown lists any options for resets that are specified in your board's board files. For most boards, only one option will be available. Click OK to continue.

**Note:** Depending on the polarity of the reset button (active high or active low), a Utility Vector Logic IP may be inserted between the reset port and the Clocking Wizard. This is used to ensure that the active high reset pin of the IP is provided with the correct polarity of reset signal, and that the design will not be held in reset while the reset button is not pressed.

Your design will require at least one clock from the clocking wizard. If your design requires more clocks, then they must be added through the Clocking Wizard.

Double click on the Clocking Wizard IP core to edit its settings. Navigate to the Output Clocks tab. Additional clocks are added to the ClockingWizard by checking a box in the Output Clock column, and specifying a Requested Output Frequency. Additionally, if desired, the ports can be named according to their intended purpose.

If your design requires additional clocks (such as for an ext_spi_clock pin), they should be added now.

Click OK to confirm your changes to the Clocking Wizard's settings.
If you aren’t sure that you have all of the clocks you need, don’t worry, you can always come back and add them by reconfiguring this IP. This task can be performed whenever in the design process it becomes necessary.

Next, use the Add IP button (\(\text{+)\}) to add the MicroBlaze IP to the design. The block that is added represents the core of the Microblaze processor.

Next, block automation will be run so that additional supporting infrastructure can be added to the design. Click Run Block Automation in the green Designer Assistance bar.

**Note:** The screenshot to the right is not representative for a design not using DDR, as these designs will not contain a MIG IP core.
In the *Run Block Automation* dialog, several settings can be specified for how the Microblaze IP will be connected to the rest of the design:

- **Local Memory** specifies how much block RAM memory will be dedicated to the processor. DDR-less designs require more memory, and the amount of memory necessary depends heavily on the size of the software application being run. 32KB is enough for many small applications.
- **Cache Configuration** can help the speed of designs using DDR memory. It should be enabled when using DDR and disabled otherwise.
- **Debug Module** allows you to specify the capabilities of the debugger. The default *Debug Only* option is recommended.
- **Peripheral AXI Port** enables or disables the AXI master interface of the processor. It must be enabled to allow the processor to be connected to hardware peripherals.
- **Interrupt Controller** specifies whether the processor can be interrupted by its peripherals. Whether or not it needs to be enabled depends on your design requirements. If any IP that you intend to connect to the processor must have interrupts to function correctly, the box must be checked.
- **Clock Connection** specifies the processor's clock. Designs using DDR should use the MIG's iui_clk pin, while designs without DDR should use the Clocking Wizards clk_out1 pin.

**Note:** Settings not present in this list are out of the scope of this guide, and can safely be left as their default.

Confirm that the settings meet your design requirements. It should be noted that while it is possible to change these settings manually later (for example, by manually adding an AXI INTC IP and connecting it to the processor), the easiest way to do so will be to clear the Microblaze processor out of your block design and restart the process of adding the processor. This is to say, the settings chosen here are important. Getting them right the first time will save you time in the long run.

Click **OK** to continue.

Next, in order for the software design to be able to print to a serial console on a host computer, a UART peripheral must be connected. Find the **USB UART** interface in the *Board* tab, right click on it, and select **Connect Board Component**.
In the dialog that pops up, select a new AXI Uartlite IP's UART interface, and click OK to add the block to the diagram.

To connect all remaining AXI cores that have not yet been connected to the processor, click the Run Connection Automation button in the green Designer Assistance bar. Check the All Automation box in the list on the left side of the window to select all of the remaining AXI cores. The dropdown settings available for each core can safely be left as their default values. Click OK to automatically connect them to the processor.

Follow the steps in this dropdown for boards with DDR memory

When creating a design with DDR, it's best to add the DDR interface first, as it is typically also used to generate the clock or clocks that will be used by the rest of your design.
In the Board tab, right click on the DDR interface and select “Auto Connect”. This process will add a MIG and the external DDR interface to the design. Two clock pins are also created, which will need to be modified.

Delete the “clk_ref_i” pin. This can be accomplished either by right-clicking on the pin and selecting delete or by selecting and pressing the delete key.

Verify that the “ui_addn_clk_0” pin has a frequency near 200 MHz by selecting it and looking at the “Frequency” value in the “Block Pin Properties” pane.
Manually connect the “ui_addn_clk_0” pin to the “clk_ref_i” pin by clicking and dragging from one to the other.

Returning to Vivado, click the **Add Sources** button in the **Project Manager** section of the **Flow Navigator** pane. This will launch a dialog that you can use to add a variety of types of source files to the project (or create new ones).

It’s important to note that the “sys_clk_i” pin will not be constrained by the board files, and you will need to add a constraint file to map it to the corresponding pin location on the FPGA.

If your project doesn’t contain the master Xilinx Design Constraint (XDC) file for your board, the dropdown below details how to add it. This file contains the constraints that your board places on designs using it - specific interfaces wired up to specific pins, clock frequencies, and FPGA bank voltages, for some examples. Click the dropdown below for a walkthrough of how to add this file to your project.

**Add a Master XDC File to a Vivado Project**

Download and extract digilent-xdc-master.zip. This file includes all of the latest template XDC files released for Digilent’s boards, which are available on Github in the digilent-xdc repository.


On the first screen, select Add or create constraints. Click Next to continue.

In the next screen, make sure that the constraint set specified (the one that the master XDC will be added to) is set to constrs_1, and that it is the active set. Click the Add Files button.
In the dialog that pops up, navigate to the folder that the `digilent-xdc-master.zip` file was extracted into. Highlight the XDC file for your board, then click OK to continue.

Back in the Add Sources dialog, make sure that your chosen constraint file appears in the table. Also, make sure that the Copy constraint files into project box is checked. If this box is unchecked, the file will be linked by your project, and any modifications made within the project will affect the version you downloaded. Since you may need to use this file again in other projects, copying the constraint file is recommended, so that you can always work from a fresh copy.

Click Finish to add the constraint file to your project.

Once added, the XDC file will appear in the Sources tab (in the same pane as the Board tab). Double click it to open the file.
Find the set_property and create_clock constraints for your board's 100 MHz input clock, uncomment them by removing the # at the start of each line, and change the name of the port that they are constraining to sys_clk_i, to match the name of the port in the block design.

If your board has multiple clocks, the 100 MHz one can be determined by observing that the create_clock constraint specifies a 10.000 ns period, as seen in the screenshot to the right.

Make sure to save your changes.

Next, the MIG's reset pin will be connected to the board's reset button. Click “Run Connection Automation” in the green bar at the top of the window.

In the list to the left side of the dialog that pops up, make sure that the “sys rst” box is checked. Click OK to connect the reset to the corresponding board part interface.

The MIG block may have other ports which will need to be connected to ensure that it works correctly. This section discusses each of those ports.

Open this dropdown if your MIG has a "device_temp_i" port
If your MIG block has a “device_temp_i” port, it means that the MIG is not using the chip's XADC analog-to-digital converter feature. We’ll ground this port to prevent any warnings that it may throw. Add a “Constant” IP to the design.

Double click on the block to open its configuration wizard and modify it to have a Value of “0” and a Width of “12”.

Connect its output port to the “device_temp_i” port on the MIG.

If your design requires more clocks than just the ui_clk provided by the MIG, you will need to add a clocking wizard IP that is driven by the MIG’s clk.
Use the Add IP button to search for and add a Clocking Wizard to the design.

Manually connect the Clocking Wizard's clk_in1 and reset ports to the MIG's ui_clk and ui_clk_sync_rst ports, respectively.

Finally, double-click on the Clocking Wizard to open and configure it. The third tab, Output Clocks, contains all of the settings required to specify how many clocks you need, and of what frequencies. The screenshot to the right shows the wizard configured to create a 100 MHz \( \text{clk}_\text{out1} \) and a 50 MHz \( \text{clk}_\text{out2} \).

If your design requires additional clocks (such as for an ext_spi_clock pin), they should be added here.
At the end of this process, the block design will look something like this:

Next, use the **Add IP** button (↓) to add the **MicroBlaze** IP to the design. The block that is added represents the core of the Microblaze processor.

Next, block automation will be run so that additional supporting infrastructure can be added to the design. Click **Run Block Automation** in the green **Designer Assistance** bar.

**Note:** The screenshot to the right is not representative for a design not using DDR, as these designs will not contain a MIG IP core.
In the Run Block Automation dialog, several settings can be specified for how the Microblaze IP will be connected to the rest of the design:

- **Local Memory** specifies how much block RAM memory will be dedicated to the processor. DDR-less designs require more memory, and the amount of memory necessary depends heavily on the size of the software application being run. 32KB is enough for many small applications.

- **Cache Configuration** can help the speed of designs using DDR memory. It should be enabled when using DDR, and disabled otherwise.

- **Debug Module** allows you to specify the capabilities of the debugger. The default *Debug Only* option is recommended.

- **Peripheral AXI Port** enables or disables the AXI master interface of the processor. It must be enabled to allow the processor to be connected to hardware peripherals.

- **Interrupt Controller** specifies whether the processor can be interrupted by its peripherals. Whether or not it needs to be enabled depends on your design requirements. If any IP that you intend to connect to the processor must have interrupts to function correctly, the box must be checked.

- **Clock Connection** specifies the processor's clock. Designs using DDR should use the MIG's `ui_clk` pin, while designs without DDR should use the Clocking Wizards `clk_out1` pin.

**Note:** Settings not present in this list are out of the scope of this guide, and can safely be left as their default.

Confirm that the settings meet your design requirements. It should be noted that while it is possible to change these settings manually later (for example, by manually adding an AXI INTC IP and connecting it to the processor), the easiest way to do so will be to clear the Microblaze processor out of your block design and restart the process of adding the processor. This is to say, the settings chosen here are important. Getting them right the first time will save you time in the long run.

**Important!** When working with multiple clocks in a design (as happens to always be the case when working with the MIG) it is important to verify that you are picking the correct *Clock Connection* from the
dropdown. In the screenshot to the right, the 100 MHz clk_out1 from a clocking wizard is chosen. You may instead wish to run your design off of ui_clk itself. Do not select the system clock input to the MIG.

Click OK to continue.

Next, the MIG's AXI interface must be connected to Microblaze's cache ports, in order to allow data to move back and forth between processor and DDR memory.

To connect the MIG's AXI interface to the processor, click the Run Connection Automation button in the green banner at the top of the block design.

In the dialog that pops up, you will be presented with several options for interfaces that can be connected to other interfaces. Both the Microblaze's IC and DC ports, as well as the MIG's AXI port will appear. You should only run connection automation for one side of the connection - the S_AXI port, as shown in the screenshot to the right.

Make sure that its box is checked. Check that the Master interface is set to “/microblaze_0 (Cached)”, indicating that the microblaze local memory will act as a cache for the DDR memory, then click OK to make the connections.

Next, in order for the software design to be able to print to a serial console on a host computer, a UART peripheral must be connected. Find the USB UART interface in the Board tab, right click on it, and select Connect Board Component.
In the dialog that pops up, select a new AXI Uartlite IP’s UART interface, and click **OK** to add the block to the diagram.

To connect all remaining AXI cores that have not yet been connected to the processor, click the **Run Connection Automation** button in the green *Designer Assistance* bar. Check the *All Automation* box in the list on the left side of the window to select all of the remaining AXI cores. The dropdown settings available for each core can safely be left as their default values. Click **OK** to automatically connect them to the processor.
Add a Zynq Processor to a Block Design

The Zynq7 Processing System IP represents the non-FPGA components of a Zynq chip, referred to as the Processing System, or PS. It must be used in a block design that wants to connect anything to the processor, and to configure PS-side peripherals, clocks, and other settings.

**Note:** This section only applies to boards with a 7-series Zynq chip.

In the block diagram pane’s toolbar, click the **Add IP** button (➕).

In the pop up, search for and double click on **ZYNQ7 Processing System**.

Click **Run Block Automation** in the Design Assistance banner (the green bar).
In the dialog that pops up, leave all settings as defaults. *Apply Board Preset* should be checked.


The Zynq PS can generate multiple clocks that are then provided to the FPGA fabric. These clocks are referred to as FCLKs, and can be found in the **Clock Configuration** tab of the Zynq PS configuration wizard. They are located under the **PL Fabric Clocks** dropdown. They can be enabled (or disabled) with a checkbox, the hardware used to drive the clock can be changed, and the frequency can be modified.

All board files for Digilent Zynq boards enable a single Zynq PL clock by default, which is intended to be used with peripherals connected to the Zynq’s M_AXI_GP0 port.

Some designs may require additional clocks of specific frequencies be added to your design. In these cases, enable a second clock and specify the needed frequency, as seen in the image to the right.

**Note:** This section can always be returned to later, as the addition of an additional clock can be performed any time before the hardware is built.


Zynq devices can also use interrupts generated in FPGA fabric to trigger interrupts within the Processing System. Interrupt-related settings can be changed within the configuration wizard's interrupts tab. These interrupts typically use the IRQ_F2P port, which can be found under the **Fabric Interrupts** → **IRQ_F2P** dropdown. To enable this port, both the Fabric Interrupts and IRQ_F2P ports must be enabled.

While interrupts can be directly connected to the IRQ_F2P port (by clicking and dragging from one port to another), some designs may require multiple interrupt sources. In these cases, add a Concat IP to your block design, and manually connect it to the IRQ_F2P port. Additional input ports can be added to a Concat block through its configuration wizard (opened by double clicking on the IP).

Add a Zynq UltraScale+ Processor to a Block Design

The Zynq UltraScale+ MPSoC IP represents the non-FPGA components of a Zynq UltraScale chip, referred to as the Processing System, or PS. It must be used in a block design that wants to connect anything to the processor, and to configure PS-side peripherals, clocks, and other settings.

Note: This section only applies to boards with a Zynq UltraScale+ chip.

In the block diagram pane's toolbar, click the Add IP button ( ).

In the pop up, search for and double click on Zynq UltraScale+ MPSoC.
Click **Run Block Automation** in the Design Assistance banner (the green bar).

In the dialog that pops up make sure **Apply Board Preset** is checked. This will apply the preset configuration from the board files to the IP, which saves a lot of time and prevents potential issues with doing the configuration entirely manually. Click **OK** to continue.
The PS can generate multiple clocks that are then provided to the FPGA fabric. These clocks are referred to as PL clocks, and can be found in the Clock Configuration tab of the MPSoC configuration wizard. They are located under the Low Power Domain Clocks → PL Fabric Clocks dropdowns. They can be enabled (or disabled) with a checkbox, the hardware source used to drive the clock can be changed, and the frequency can be modified.

Board files for Digilent Zynq UltraScale boards enable at least one low power domain PL clock by default, which is intended to be used with peripherals connected to the MPSoC’s M_AXI_HPM0_LPD port.

Some designs may require additional clocks of specific frequencies be added to your design. In these cases, enable a second clock and specify the needed frequency, as seen in the image to the right.

**Note:** This section can always be returned to later, as the addition of an additional clock can be performed any time before the hardware is built.

UltraScale devices can also use interrupts generated in FPGA fabric to trigger interrupts within the Processing System. Interrupt-related settings can be changed within the configuration wizard’s PS-PL Configuration tab. These interrupts can use the IRQ0 port, which can be found under the General → Interrupts → PL to PS dropdowns. To enable this port, the IRQ0 dropdown should be set to “1”.

While interrupts can be directly connected to the pl_ps_irq0 (IRQ0) port by clicking and dragging from one port to another, some designs may require multiple interrupt sources. In these cases, add a Concat IP to your block design, and manually connect it to the pl_ps_irq0 port.

Additional input ports can be added to a Concat block through its configuration wizard (opened by double clicking on the IP).
Add GPIO Peripherals to a Block Design

This section covers the steps involved in adding a GPIO peripheral to a block design. While an AXI GPIO IP is used, other IPs and interfaces can potentially be added to your design in the same ways. Two methods are presented here, one for each of the two AXI GPIO peripherals that will be connected. The first takes advantage of board files to automatically generate constraints, the second presents how an IP interface can be connected to chosen pins manually.

Interfaces for Digilent boards supported by the board files can be found in the Board tab. For the purposes of this guide, find the GPIO section of the list, right click on an LED interface, and select the Connect Board Interface option.

Note: If your board does not have single-color LEDs, you can use its RGB LEDs instead. Note that these interfaces have three pins for each LED, for the R, G, and B components.

In the dialog that pops up, choose the “GPIO” interface (not GPIO2) of a new AXI GPIO IP.

Some boards use one of their user buttons as reset sources. In these cases, make sure to choose the Component mode that does not include the reset button.

Click OK to continue. This will add the IP to your design, and connect it to an external port, which will not require any further work to constrain.
Next, select the axi_gpio_0 block. The Block Properties pane to the left of the Diagram and below the Board tab will allow you to view some information about the block, and modify it in some ways, without running through its customization wizard. For now, just change its name to “AXI_GPIO_0_LED” by typing in the Name field. Pressing enter or clicking out of the text box confirms the change. Using memorable names in your block design makes it easier to remember which IP does what when you are later writing software in Vitis.

Next, a second AXI_GPIO_ IP will be manually added to the block diagram, and manually constrained with an XDC file. Click the Add IP button (+) and search for “AXI_GPIO_”. Double click on the only result to add the second AXI_GPIO_ block to the design. Once added, rename this IP “AXI_GPIO_BUTTONS”.

Select the AXI_GPIO BUTTONS IP’s GPIO interface by clicking on the text “GPIO”, right click on the highlighted text, and select Make External. This option creates a new external interface port that
does not rely on the board files. Because the board files are not used here, a Xilinx Design Constraint (XDC) file must be added to the project to tell Vivado which FPGA pins to connect the interface to.

If your project doesn’t contain the master Xilinx Design Constraint (XDC) file for your board, the dropdown below details how to add it. This file contains the constraints that your board places on designs using it - specific interfaces wired up to specific pins, clock frequencies, and FPGA bank voltages, for some examples. Click the dropdown below for a walkthrough of how to add this file to your project.

Add a Master XDC File to a Vivado Project

Download and extract digilent-xdc-master.zip

This file includes all of the latest template XDC files released for Digilent’s boards, which are available on Github in the digilent-xdc repository.

Returning to Vivado, click the Add Sources button in the Project Manager section of the Flow Navigator pane. This will launch a dialog that you can use to add a variety of types of source files to the project (or create new ones).
On the first screen, select *Add or create constraints*. Click **Next** to continue.

In the next screen, make sure that the constraint set specified (the one that the master XDC will be added to) is set to *constr_1*, and that it is the *active* set. Click the **Add Files** button.

In the dialog that pops up, navigate to the folder that the *digilent-xdc-master.zip* file was extracted into. Highlight the XDC file for your board, then click **OK** to continue.
Back in the Add Sources dialog, make sure that your chosen constraint file appears in the table. Also, make sure that the Copy constraint files into project box is checked. If this box is unchecked, the file will be linked by your project, and any modifications made within the project will affect the version you downloaded. Since you may need to use this file again in other projects, copying the constraint file is recommended, so that you can always work from a fresh copy.

Click Finish to add the constraint file to your project.

Once added, the XDC file will appear in the Sources tab (in the same pane as the Board tab). Double click it to open the file.
Master XDC files for Digilent boards contain pin constraints for I/O interfaces the board offers. These constraints are sorted by interface. Scroll down until you see constraints for the user buttons. These constraints typically are for a bus port named “btn”. Un-comment the button constraints by removing the single leading '#' character in each line corresponding to the buttons, as seen in the screenshot to the right.

Next, the names of the block design’s GPIO port for the buttons must be determined, so that the buttons can be properly constrained. Reopen the Diagram tab, and select the GPIO external port that is connected to the AXI_GPIO BUTTONS block. Change the name of the external interface to “btn” in the Properties pane.

The AXI_GPIO IP automatically uses tri-state buffers for the pins its interfaces are connected to. The individual I, O, and T buses can be seen when expanding the interface through the plus button (+) next to the interface name on the IP block. As can be seen, the individual ports that make up the interface are named <interface>_tri_i, <interface>_tri_o, and <interface>_tri_t. When constrained to tristate buffers, the bus that is connected to FPGA ports is named <interface>_tri_io.

With this knowledge, return to the XDC file, and change the name of the button bus that is constrained. Specifically, change the text after the “get_ports” call on each line of the button interface to “btn_tri_io[#]”, where # is a decimal number, counting up from zero. When finished, save the file.

With the constraints for the port finished, the AXI_GPIO BUTTONS must be manually configured. In particular, the width of the GPIO interface must match the number of buttons available on the board. Take note of how many buttons are constrained in the XDC.

Return to the Diagram tab, and double click on the AXI_GPIO BUTTONS block. This will open a dialog that will allow you to configure the IP's settings. Switch to the configuration wizard's IP Configuration tab.

Only one setting need be changed for the purposes of this guide. Enter the number of buttoons you constrained into the GPIO Width field. When finished, click OK to save your changes.
Finally, the two AXI GPIO IP blocks need to be connected to the processor in your design. Click the Run Connection Automation button in the green Designer Assistance bar.

In the dialog that pops up, make sure that the boxes for the S_AXI interfaces for both of the AXI GPIO IPs are checked. Click OK to run connection automation and connect the AXI GPIO IP blocks to your processor.
In the unlikely event that Vivado fails to correctly assign addresses to each AXI IP connected to your processor, you may need to manually set their addresses. If this occurs, errors will pop up during validation of the block design, and the bitstream will not be able to be generated. The Address Editor can be accessed through its tab in the Diagram pane. Addresses can be assigned to unmapped peripherals by typing the desired address into the peripheral's Master Base Address column.

It should be noted that addresses must be aligned in the memory space - for instance, an address with a range of 4K (bytes) takes up a range of 0x1000 addresses, and must have three trailing zeros. Address ranges for different segments cannot overlap.

Assigning an segment to address 0 may result in assertions in some software drivers and should be avoided.

After manually assigning addresses, the block design should be re-validated.

Validate a Block Design

Before the Vivado project can be built, the block design must be validated. This step runs an automatic check of the block design to see if there are any potential issues with it. Click the Validate Design button (in the Diagram pane’s toolbar (or press the F6 key).

If the design has issues, a dialog will pop up that lists them. It should be noted that most Warnings can be ignored, as can some Critical Warnings. These issues can also be viewed in the Messages tab of the pane at the bottom of the window.

If there are no issues, a dialog will pop up that will tell you so. Click OK to continue.
**Create an HDL Wrapper**

Additionally, an HDL wrapper must be created for the block design. This process translates the block design into a source file that can be read by the Vivado tools, and is used to build the actual design.

Open the Sources pane and locate the block design file (.bd) under the Design Sources dropdown. Right click on it and select **Create HDL Wrapper**.

In the dialog that pops up, you can decide whether to let Vivado edit the wrapper file itself. **Let Vivado manage wrapper and auto-update** is recommended, as a user rarely needs to manually edit the wrapper file. Click **OK** to continue.

---

**Build a Vivado Project**

At this point, the Vivado Project is ready to be built, by running it through Synthesis and Implementation, and finally generating a bitstream. Click the **Generate Bitstream** button in the Program and Debug section of the Flow Navigator pane at the left side of the window.
A dialog will pop up with several options for how Synthesis and Implementation should be run. Most should be left as defaults. Of particular importance is the Number of jobs dropdown, which is used to specify how much of the resources of your computer should be dedicated to the build. A larger number of jobs will dedicate more resources, which will allow the build to be completed faster. It is recommended to choose the highest available number.

**Note:** Critical warnings about how IPs included within another IP were packaged with a different board value can be safely ignored. The same is true for warnings related to negative CK-to-DQS delays seen on some Zynq boards.

Depending on the complexity of the design, the board used, and the strength of your computer, the process of building the project can take between 5 and 60 minutes.

When complete, a dialog will pop up that presents several options for what to do next:

- **Open Implemented Design** can be used to view the actual hardware design that has been implemented and will be placed onto the chip.
- **View Reports** can be used to view additional information about the design, including how much of the resources of the FPGA will be used by the design.
- **Open Hardware Manager** can be used to go directly to Vivado’s Hardware Manager, which can be used to program a hardware design onto a board. This is typically used for designs that do not involve a software component.
- **Generate Memory Configuration File** can be used to create a file for programming an FPGA-only design into flash memory.

If none of these options are desired, click **Cancel** to continue.

**Export a Fixed Post-Synthesis Hardware Platform**

Once the project has been built, the design must be exported from Vivado so that Vitis has access to information about the hardware that a software application is being developed for. This includes the set of IP connected to the processor, their drivers, their addresses, and more. Exporting hardware after the bitstream has been generated allows you to program your board directly from within Vitis.

To export the hardware design, click **Export → Export Hardware** in the **File** menu.
The wizard that pops up guides you through the options available for hardware export. The first screen allows you to select a Fixed or Expandable platform. In this case, choose a Fixed platform and click Next to continue.

The Output screen allows you to select whether only the hardware specification (Pre-synthesis) should be exported, or whether the bitstream should be included. Since the bitstream has already been generated, it should be included in the platform so that Vitis can automatically figure out where it is when programming a board. Select Include bitstream and click Next to continue.
The *Files* screen gives you the option to choose a name for the Xilinx Shell Architecture (XSA) file, and provide a path to a folder that the file will be placed within. Give your XSA file a name, and choose a memorable location to place it in. This file will later be imported into Vitis, so take a note of where it is placed and what it is called.

**Important:** Do not use spaces in the file name or export path. Underscores or camelCase ([https://en.wikipedia.org/wiki/Camel_case](https://en.wikipedia.org/wiki/Camel_case)) are recommended instead.

Click **Next** to continue.

The final screen of the wizard summarizes the options you selected. Click **Finish**.
Open Vitis through the start menu or desktop shortcut created during the installation process.

Upon launching Vitis, a dialog will appear where a workspace must be chosen. The workspace is the directory where all of the projects and files for the application being developed will live. If a folder that does not currently exist is chosen, it will be created. Choose a workspace and click Launch to finish launching Vitis.

**Launch Vitis**

* Select the dropdown corresponding to your operating system, below:

**Windows**
Open Vitis through the start menu or desktop shortcut created during the installation process.

**Linux**
Open a terminal and run the following commands. The install path is /opt/Xilinx by default.

```
source <install_path>/Vitis/2020.1/settings64.sh
vitis
```

**Note:** Regardless of OS, if Vivado is open, Vitis can also be launched through the Tools → Launch Vitis toolbar option.

**Create a New Application Project**
With Vitis open, an application project must be created to hold your source files. In creating an application project, a hardware platform will also be created from an XSA file previously exported from Vivado.

On Vitis' welcome screen, click **Create Application Project**. The wizard that launches will be used to create and configure a new application.

The first screen of the wizard is a welcome page, which summarizes what each of the components of a software design are. Click **Next** to continue.

Next, the platform that the application targets must be created. Open the **Create a new platform...** tab.
Browse your file system to find the Xilinx Shell Architecture previously exported from Vivado. With the XSA file highlighted, click Open to select it and return to the Platform screen of the wizard.

Once you have found the XSA file and opened it, make sure that it is selected in the Hardware Specification list. Give your platform a name (the default uses whatever the name of the XSA file is and will work fine). The Generate boot components box can be used to automatically build all of the additional components necessary to boot the application from flash memory or an SD card. Leaving this box checked is recommended. Click Next to continue.

The next screen is used to set some options for the application project and the system project. The names of both projects can be set, as well as which processor core will be used to run the application. All settings can be left as defaults. Click Next to continue.

Note: A system project can contain multiple application projects, which can all be run at once.
Next, the domain that the application project operates in will be defined. In this case all default settings will be used. Click **Next** to continue.

Lastly, a template project will be chosen. Each template pre-configures the application project for a different purpose. Depending on whether your application will be written in C or C++, choose **Empty Application** or **Empty Application (C++)**. You will be adding an example main source file later, as opposed to working from and editing an example.

Click **Finish** to finish creating the project.

---

Create a Main C Source to Control AXI GPIO Peripherals

*An application needs source files to define its behavior. This step will show how to create a new source file for the application, and provide some example code.*

In Vitis’ **Explorer** pane, find the application projects “src” directory. Right click on it and select **New → File**.
In the dialog that pops up, name the file “main.c”. The parent folder can be specified as well, but through the use of the right click in the previous step, the correct folder has already been chosen.

Copy and paste the code to the right into the empty main.c file that has now been opened. Change the BTN_MASK and LED()_MASK macros so that they contain a number of ‘1’s equal to the number of buttons and leds connected to the GPIO() peripherals in the hardware design.

Header Files - Additional Information
This code pulls in several headers that are automatically pulled into the Vitis workspace:

`xparameters.h` is a file generated during the process of exporting a platform from Vivado. It includes information on the hardware design, including addresses and some configuration parameters for AXI IPs. This is used by the example code to find the device IDs that must be passed to the GPIO drivers, so that they can look up the driver configuration required to correctly initialize the GPIO devices.

`xil_printf.h` gives access to the xil_printf function, which can be used to print to standard output, and requires less memory space than the stdio library.

`xgpio.h` gives access to the XGpio drivers, which are used to provide a standard API for controlling AXI GPIO peripherals. Several functions from this API are used in the example, including the GPIO reads, writes, and direction-setting calls.

`xil_types.h` contains a variety of different C types. In this case, it is only used to get access to the “u32” (unsigned 32-bit int) type, which is used in arguments to XGpio function calls.

**What the Example Code Does**

When the example is started, the message “Entered function main” is printed to a connected serial console. After that, the AXI GPIO IPs and drivers are initialized, and the application constantly loops, checking whether any button is pressed, and if they are, setting the LEDs high. When no buttons are pressed, the LEDs are held low.

```c
#include "xparameters.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "xil_types.h"

// Get device IDs from xparameters.h
#define BTN_ID XPAR_AXI_GPIO_BUTTONS_DEVICE_ID
#define LED_ID XPAR_AXI_GPIO_LED_DEVICE_ID
#define BTN_CHANNEL 1
#define LED_CHANNEL 1
#define BTN_MASK 0b1111
#define LED_MASK 0b1111

int main() {
    XGpio_Config *cfg_ptr;
    XGpio led_device, btn_device;
    u32 data;

    xil_printf("Entered function main\r\n");

    // Initialize LED Device
    cfg_ptr = XGpio_LookupConfig(LED_ID);
    XGpio_CfgInitialize(&led_device, cfg_ptr, cfg_ptr);

    // Initialize Button Device
    cfg_ptr = XGpio_LookupConfig(BTN_ID);
    XGpio_CfgInitialize(&btn_device, cfg_ptr, cfg_ptr);

    // Set Button Tristate
    XGpio_SetDataDirection(&btn_device, BTN_CHANNEL,
                           BTN_MASK);

    // Set Led Tristate
    XGpio_SetDataDirection(&led_device, LED_CHANNEL,
                           LED_MASK);

    while (1) {
        data = XGpio_DiscreteRead(&btn_device, BTN_CHANNEL);
        if (data != 0) {
            data = LED_MASK;
        } else {
            data = 0;
        }
        XGpio_DiscreteWrite(&led_device, LED_CHANNEL, data);
    }
}
```

**Build a Vitis Application**

Once an application project has been set up and includes all necessary sources, it should be built. To build the project and all of its dependencies, select the system project in the Assistant pane, and either click the **Build** button ( ), or press Ctrl-B on your keyboard.

**Note:** There are three types of build targets in the Assistant pane, Platforms, Systems, and Applications. Building the application will not trigger any other applications in the system to be built, but will build the wrapper as a dependency. Building the platform will only build the platform, as it has no dependencies. Building the system causes each application in the system, as well as the platform, to be built.

This process may take several minutes to complete. When done, the **Console** tab at the bottom of the window will display a “Build Finished” message.
First, many applications require that a serial console is connected to the board, so that standard output (from print statements) can be viewed. For this purpose, a serial terminal should be used. Use a serial terminal application to connect to the board’s serial port. Unless otherwise stated, Zynq designs use a baud rate of 115200 and Microblaze designs with an AXI UART Lite IP use a baud rate of 9600.

Note: While Vitis has a built in serial terminal included in its Debug view, it sends characters to a board on a line-by-line basis. Some software examples require the use of character-by-character reception of data. Tera Term (https://ttssh2.osdn.jp/index.html.en) or PuTTY (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) are recommended if you are not sure what will work.

It’s time to program the application project onto your board! Plug your board into your computer through its USB programming and USB UART port/s, connect an external power supply (if necessary), and turn on the board.

Launch a Vitis Baremetal Software Application

First, many applications require that a serial console is connected to the board, so that standard output (from print statements) can be viewed. For this purpose, a serial terminal should be used. Use a serial terminal application to connect to the board’s serial port. Unless otherwise stated, Zynq designs use a baud rate of 115200 and Microblaze designs with an AXI UART Lite IP use a baud rate of 9600.

Note: While Vitis has a built in serial terminal included in its Debug view, it sends characters to a board on a line-by-line basis. Some software examples require the use of character-by-character reception of data. Tera Term (https://ttssh2.osdn.jp/index.html.en) or PuTTY (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) are recommended if you are not sure what will work.

In the Explorer pane at the left side of the screen, right click on the application or system project that is to be run, and select Run as → 1 Launch on Hardware (Single Application Debug). The FPGA will be programmed with the bitstream, the ELF file created by the software build is loaded into system memory, and the application project will begin to run. You will need to click back over to the Vitis Serial Terminal from the Console tab.

Note: Once the project has been run at least once, you can use the green run button (1) in the toolbar at the top of the screen to program the board instead.
At this point, your application is running, and printed messages can be seen. Congratulations, you have finished this guide!!!

The hardware project and application created here can be used as a basis for future work. See instructions found in Update an Existing Vitis Platform's Hardware Specification (https://digilent.com/reference/programmable-logic/guides/vitis-update-hardware-specification) for additional information on how the hardware design can be switched out later.

For more guides and demos for your board, return to the device's resource center, linked from the Programmable Logic (https://digilent.com/reference/programmable-logic/start) page of this wiki.

For technical support, please visit the FPGA (https://forum.digilentinc.com/forum/4-fpga/) section of the Digilent Forums.