Research Memorandum

Subject: Hardware and Software Co-Design using Vivado and Vitis Version 2023.2

First Author: Isaac Towne

Date of First Creation: 10/11/2024

Team 08 “Neuroprocessor”

How this is relevant to our project

The board we are performing simulations using for this project is a Xilinx ZedBoard, which requires Vivado and Vitis to program both the FPGA and ARM cores present on the board. As doing hardware and software co-design is a new topic for most of the team, it is important that everyone know the steps to make a basic block design, generate a bitstream, write a program based on that bitstream, and then create a bootloader to program the ZedBoard. This memo describes how to do that using the 2023.2 versions of Vivado and Vitis, which is what we are using for this project.

Installing Vivado and Vitis 2023.2

The following are instructions for installing Vivado and Vitis 2023.2 on LINUX. In this tutorial, the installation is done for Ubuntu 22.04. The instructions should be nearly the same for installing on Windows or MacOS, but a different installer will need to be used, so keep that in mind.

1. First, there are a few library dependencies that must be installed for Vivado and Vitis to work [1]. Run the following line in a terminal window on your Linux System or VM to install the libraries prior to continuing:

```
sudo apt-get install libncurses5-dev libncursesw5-dev libtinfo5 build-essential libncurses5
```

2. Go to https://www.xilinx.com/support/download.html and click the 2023.2 link found on the left under “Version.” In future years, you may have to instead click on “Vivado Archive” to find 2023.2 as newer versions are released. The page should look like the one in Figure 1.

3. Scroll down and click on the download link for the “AMD Unified Installer for FPGAs & Adaptive SoCs 2023.2: Linux Self Extracting Web Installer.” AMD will have you sign in and fill in some basic information before you can download the installer.

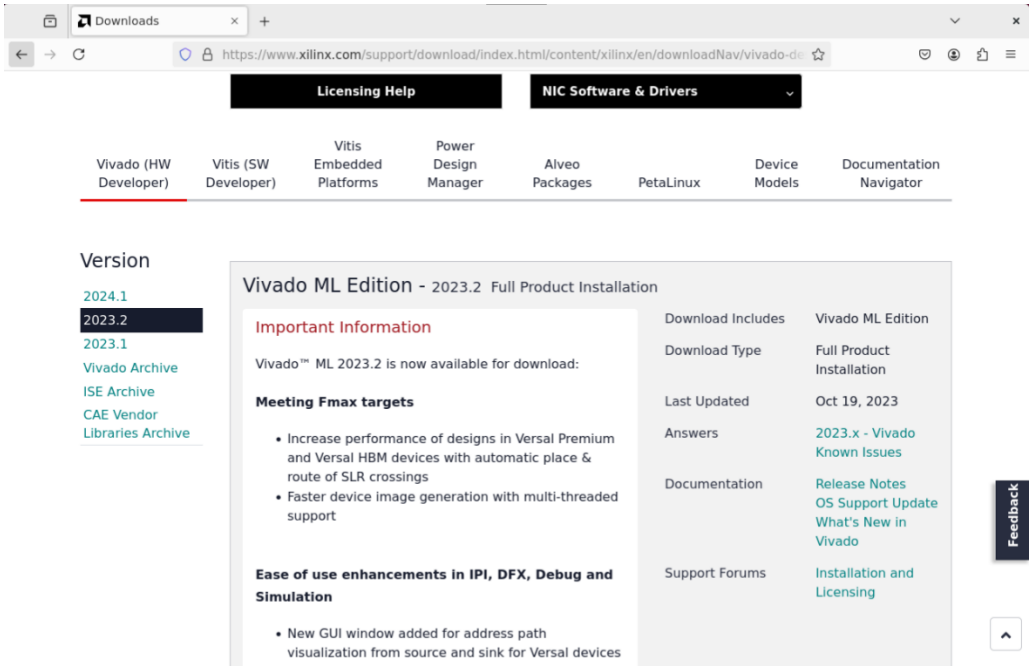


Figure 1: The AMD download page for Vivado/Vitis version 2023.2.

4. Once the .bin file has been downloaded, open a terminal window, go to your downloads, and run “chmod +x” to make the .bin file executable. Once that has been done, execute the installer as seen in Figure 2.

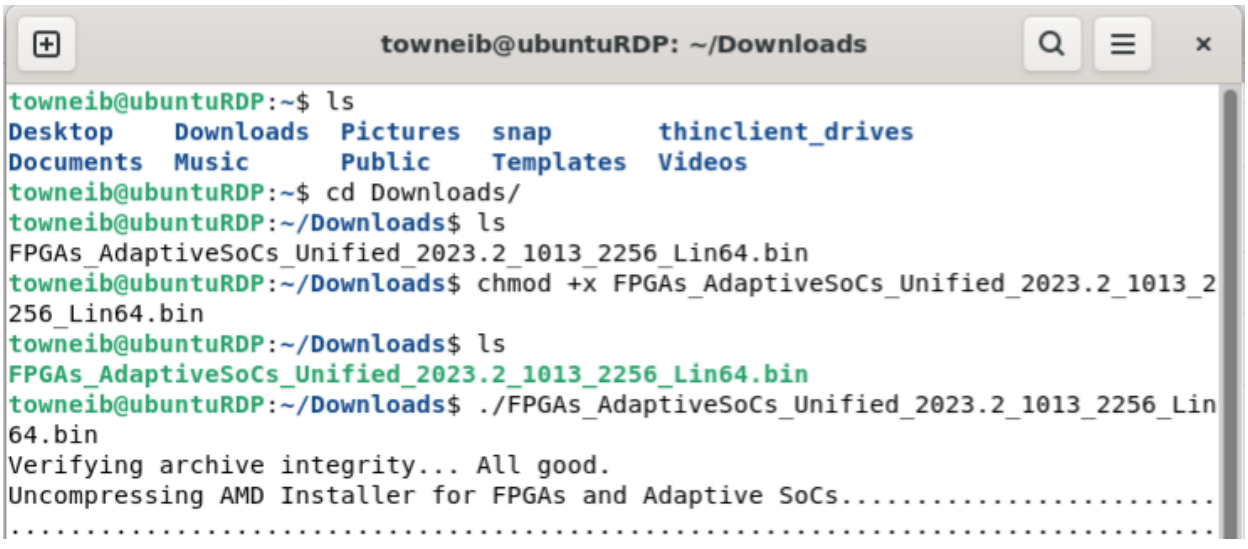


Figure 2: Example terminal commands for installing the downloaded Vivado/Vitis installer.

5. An installer GUI will open if this was done correctly. Click “Continue” on the window that pops up saying that a newer version of Vivado is available. Then click next if the OS of your Linux system or VM is one of the ones listed on the GUI welcome screen. For this demo, we are using Ubuntu 22.04. **If this is not the case, please change the OS to be one of the listed ones as earlier attempts to make Vivado and Vitis work on an unsupported OS have failed.**
6. On the next page, shown in Figure 3, enter your AMD username and password and then click next. You must log in or the web installer will not be able to download the required files.

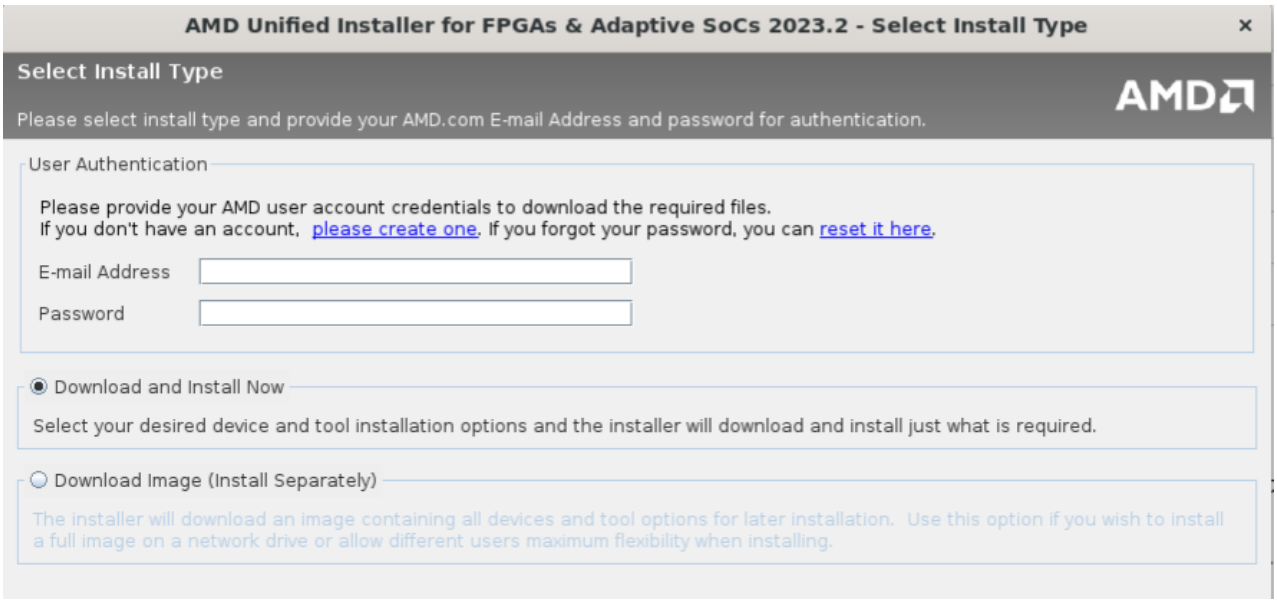


Figure 3: What the user authentication page looks like for the Linux Vivado/Vitis 2023.2 installer.

7. Select Vitis (the top option) as the product to install on the next screen. On the following screen you must select what packages you want installed. Only select those marked in Figure 4. It is unnecessary to install all the board support files when we are only using the Zedboard and the Zybo board.
8. On the next screen, agree to all the different license agreements, and continue to the next screen where you must designate an installation directory. If you are using a VM where you have set up multiple users that all must be able to access the program, I would recommend putting it in the directory below the directory with the different users (in our

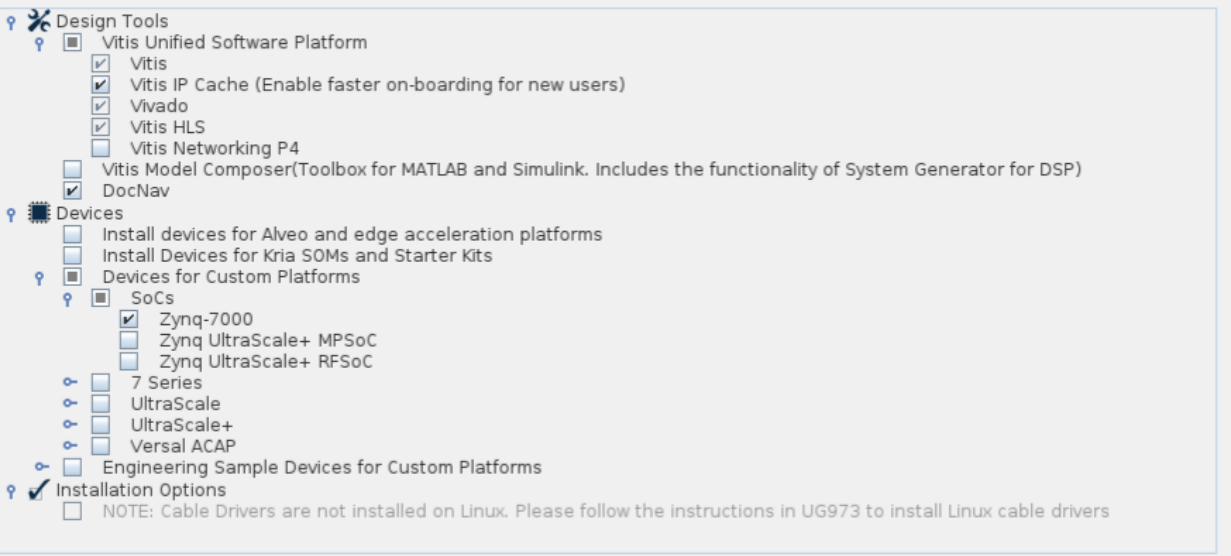


Figure 4: What packages and tools to select for the Vivado/Vitis 2023.2 install.

case, we made a folder at /opt/software/bin/Xilinx). If you are only installing it for your user, I would recommend making a folder in your home directory for it (I used /home/<YOUR USERNAME>/bin/Xilinx). After continuing and choosing the option to create the installation directory if it does not exist, your installation summary should look something like Figure 5.

9. If everything looks correct and you have enough disk space, click install and wait for the files to download. The full installation process will likely take 30 to 60 minutes depending on your internet speed. I recommend being connected via ethernet to make the process faster.

<u>Adding Board Files and Folders:</u>

Now that Vivado/Vitis has been installed, the board files for the Zybo board and ZedBoard must be installed so you can begin development. During this step we will also create a folder for Vivado to store its backup log and journal files (.log and .jou, respectively) in so they do not clutter up your main directory.

1. Open a terminal window and go to the director where you installed the programs (for me this is /home/towneib/bin/Xilinx. Use the “mkdir” command to make a folder called “Log_Journal_Files.” In this folder, use the “mkdir” command to make “Vivado_jou”

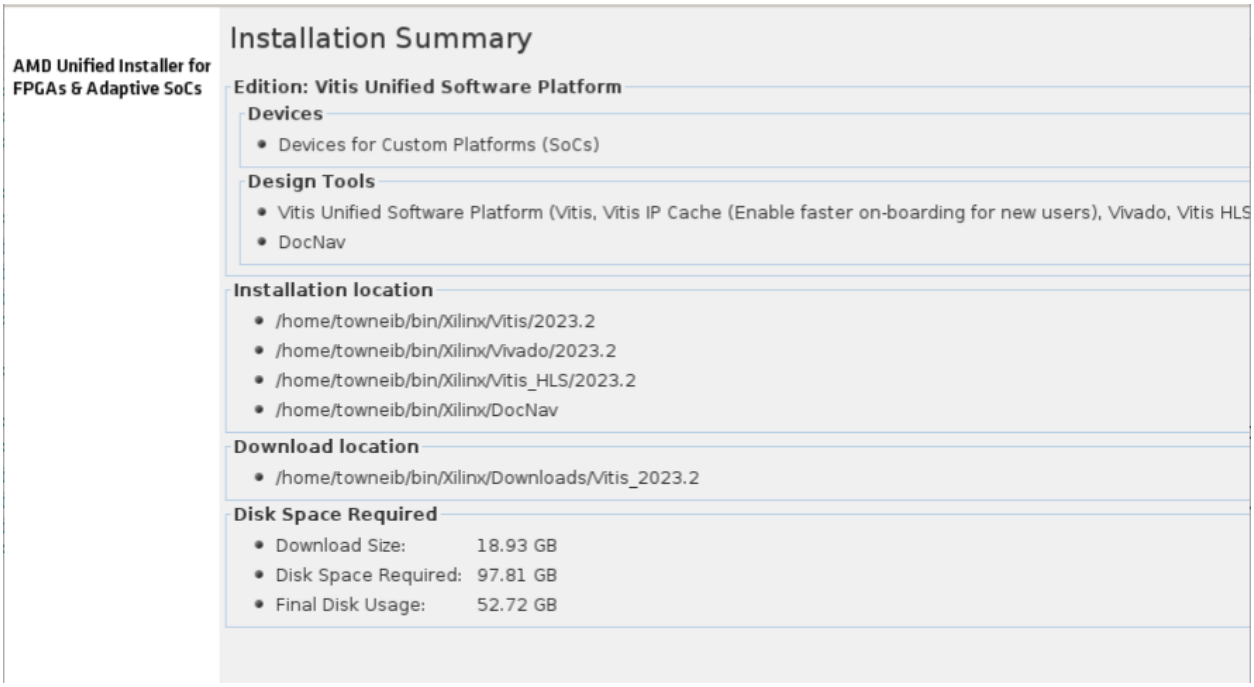


Figure 5: Example installation summary for the Vivado/Vitis 2023.2 installer.

and "Vivado_log" folders. This is where the log and journal files Vivado creates will be saved. An example of a terminal window doing this can be seen below in Figure 6.

2. Go back to your home directory and open the .bashrc file in your preferred text editor. Add the following lines to it to make it so you can start Vivado, Vitis, and Vitis HLS from the command line. Replace the "Install Directory" parts with the install path you chose when installing the programs (for me this was /home/<YOUR USERNAME>/bin/Xilinx/). See Figure 7 for an example .bashrc file.
    - source <Install Directory>/Vivado/2023.2/settings64.sh
    - source <Install Directory>/Vitis/2023.2/settings64.sh
    - source <Install Directory>/Vitis_HLS/2023.2/settings64.sh

Figure 6: Example terminal commands for making the new folders in the Xilinx install folder.

3. Also, in the .bashrc file, add the following alias to create a command that will start Vivado and save the log and journal files in the folders you made in step 1. Once again, replace the "Install Directory" parts with the install path you chose when installing the programs. See Figure 7 for an example .bashrc file.:

alias start_vivado=`vivado -log <Install Directory>/Log_Journal_Files/Vivado_log/vivado.log -journal <Install Directory>/Log_Journal_Files/Vivado_jou/vivado.jou'

```
# Source lines for the Xilinx programs
source /home/towneib/bin/Xilinx/Vivado/2023.2/settings64.sh
source /home/towneib/bin/Xilinx/Vitis/2023.2/settings64.sh
source /home/towneib/bin/Xilinx/Vitis_HLS/2023.2/settings64.sh

# Aliases for the Xilinx programs
alias start_vivado='vivado -log /home/towneib/bin/Xilinx/Log_Journal_Files/Vivado_log/v
ivado.log -journal /home/towneib/bin/Xilinx/Log Journal Files/Vivado jou/vivado.jou'
```

Figure 7: Example .bashrc file lines for sourcing the Xilinx programs and creating aliases.

4. To make the .bashrc file changes take effect, run the line "source ~/.bashrc" in your terminal window.
5. To install the correct board files, first a folder named "board files" must be created in the directories of the program installs. Using the command line, go to the following directories and use "mkdir" to make a folder named "board_files":
    - <Install Directory>/Vivado/2023.2/data/boards
    - <Install Directory>/Vitis/2023.2/data/boards
    - <Install Directory>/Vitis_HLS/2023.2/data/boards
6. Next, download the "Master Branch ZIP Archive" on https://digilent.com/reference/programmable-logic/guides/install-board-files and unzip the file to get a folder called "vivado-boards-master." Inside of that folder, click on "new," then "board_files," and then copy the "zedboard," "zybo-z7-20," and "nexys-a7-100t" folders.
7. Paste the two folders copied in the previous step into each of the newly created "board_files" folders created in step five. Keep in mind that if you do not do this, you will not be able to generate a bitstream for the ZedBoard, Zybo board, or Nexys A7-100t! A shortcut for doing this, assuming you are in the vivado-boards-master/new/board_files folder, is this:

cp -r <Download Directory>/vivado-boards-master/new/board_files/<folder to copy> <Install Directory>/<Program>/2023.2/data/boards/board_files

where <folder to copy> is the what you want to copy, <Install Directory> is where you installed Vivado, Vitis, and Vitis HLS, <Download Directory> is the location of the unzipped file downloaded in step 6, and <Program> is which of those programs (i.e. Vivado, Vitis, or Vitis_HLS) you are copying the files to.

8. Once this is done, create a directory to save your project files to for consistency. Xilinx programs do not do well with OneDrive or any folders with spaces in the file name, so I recommend just making a directory called "XUP" on your hard drive to save everything in.
9. Before moving on to the next section, one last script needs run to make sure that the correct libraries are installed for Vitis as described in [2]. Open a terminal window, navigate to <Install Directory>/Vitis/2023.2/scripts, and run "sudo ./installLibs.sh" to run the library installer. Once the installer runs, everything should be set up correctly.

Creating a Block Diagram and Exporting Hardware

This section will go through making a block diagram and exporting the hardware for a project on the ZedBoard SoC that will flash the eight LEDs found on the board. The general practices in this section can be applied to other projects as well.

1. Open Vivado with the "start_vivado" alias made earlier and click on "Create Project." This will open a new project wizard. Name the project "ZedBoard_tutorial" and set the project location to wherever you want to save your project files. **WARNING: DO NOT PUT ANY SPACES IN THE FILE NAME OR VIVADO WILL BREAK!!!**
2. On the "Project Type" screen, select "RTL Project" and click "Next". There is nothing that needs to be changed on this page. Also click "Next" on the "Add Sources" and "Add Constraints" screens. While these can be helpful if you are importing parts of an existing project, they are not necessary for this example.
3. On the "Boards" screen, click on the "Boards" tab and then click on "Zedboard" as showing in Figure 8. **If you do not see options to select the Zedboard, Zybo Z7-20, or Nexys A7-100t on this screen, please go back to the previous section on installing**

board files and repeat the steps to ensure everything is in the right place. Then click "Next" and "Finish" to create the project.

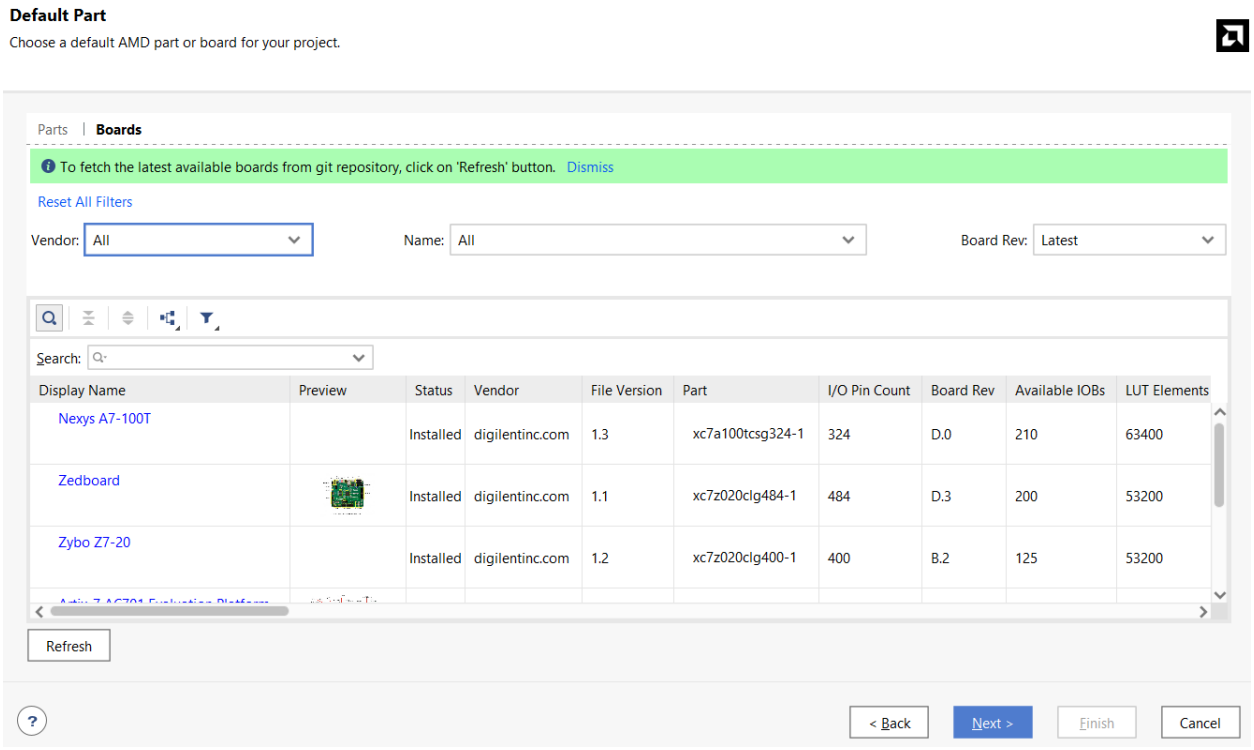


Figure 8: The "Boards" screen of Vivado's new project wizard. Note the options to select the ZedBoard, Zybo Z7-20, and Nexys A7-100t that are present because of the board files added in the previous section.

4. **One of the most important things to keep in mind when working with Vivado and Vitis is that the correct board is selected when the project is made. Otherwise, synthesis of the processor and other Intellectual Properties (IPs) will be done incorrectly.** If a situation arises where you need to change the selected board, simply go

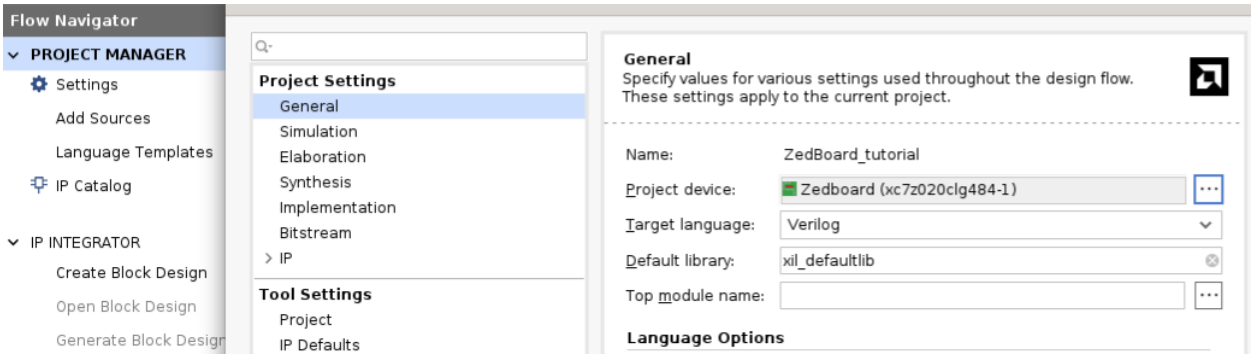


Figure 9: The menu that allows you to change the board a project is built for. Clicking the button outlined in blue will open the same window as in Figure 8.

to “Settings” under “Project Manager” and click on the three dots next to the “Project device” setting under “General” as shown in Figure 9.

5. To start your project, click “Create Block Design” under “IP Integrator.” Your block design is the blueprint for every SoC implementation, as it dictates how the processor cores on an SoC interacts with various IPs and peripherals. Name the design “zedboard_tutorial.” This should generate an empty block design.
6. Click the “+” button on the screen or use the Ctrl + I shortcut to open the IP library. This is where IPs like standard GPIO, the SoC cores, and more are found. In the search bar,

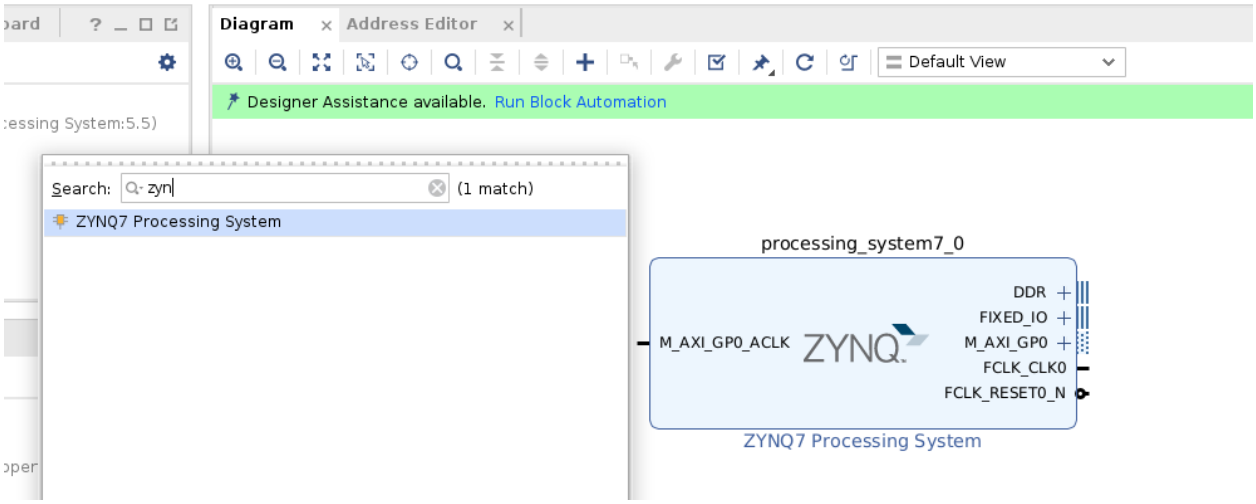


Figure 10: The IP library menu and a block diagram with only the processing system added.

type “zynq” and drag and drop the “ZYNQ7 Processing System” onto the block diagram. If done correctly, the block diagram will look like Figure 10.

7. Now, click the “Run Block Automation” button seen in Figure 10 and click “OK” on the screen that pops up. **It is absolutely imperative that you do this. Otherwise, the**

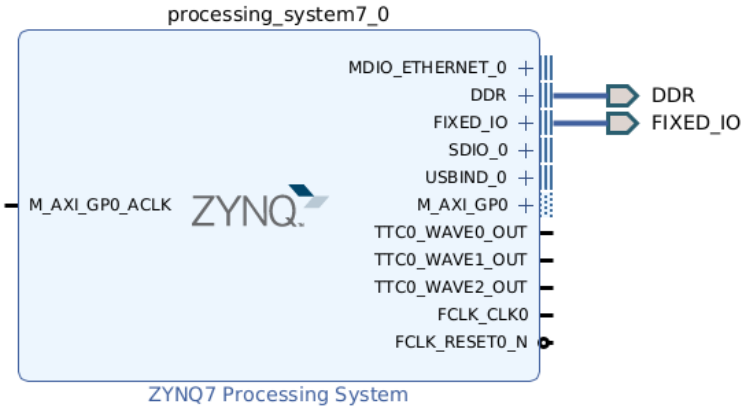


Figure 11: The processing system after block automation is run.

processing system will not be configured correctly. Doing this should result in the block diagram seen in Figure 11.

8. Next, it is time to add the GPIO for the LEDs. To add a generalized GPIO module, each of which have two 32-bit buses that can be configured as input or output, as well as interrupts, open the IP library again and type in "GPIO" and drag and drop an "AXI GPIO" module onto the block diagram. **Do not run connection automation yet! We want to change a few settings first.**
9. First, change the name of the module in the "Block Properties" window to the left of the block diagram. I named mine "LEDs". This will help us find the module later when configuring it in Vitis. Then, double-click on the newly-renamed GPIO module to open its settings. The window should look like the one in Figure 12.

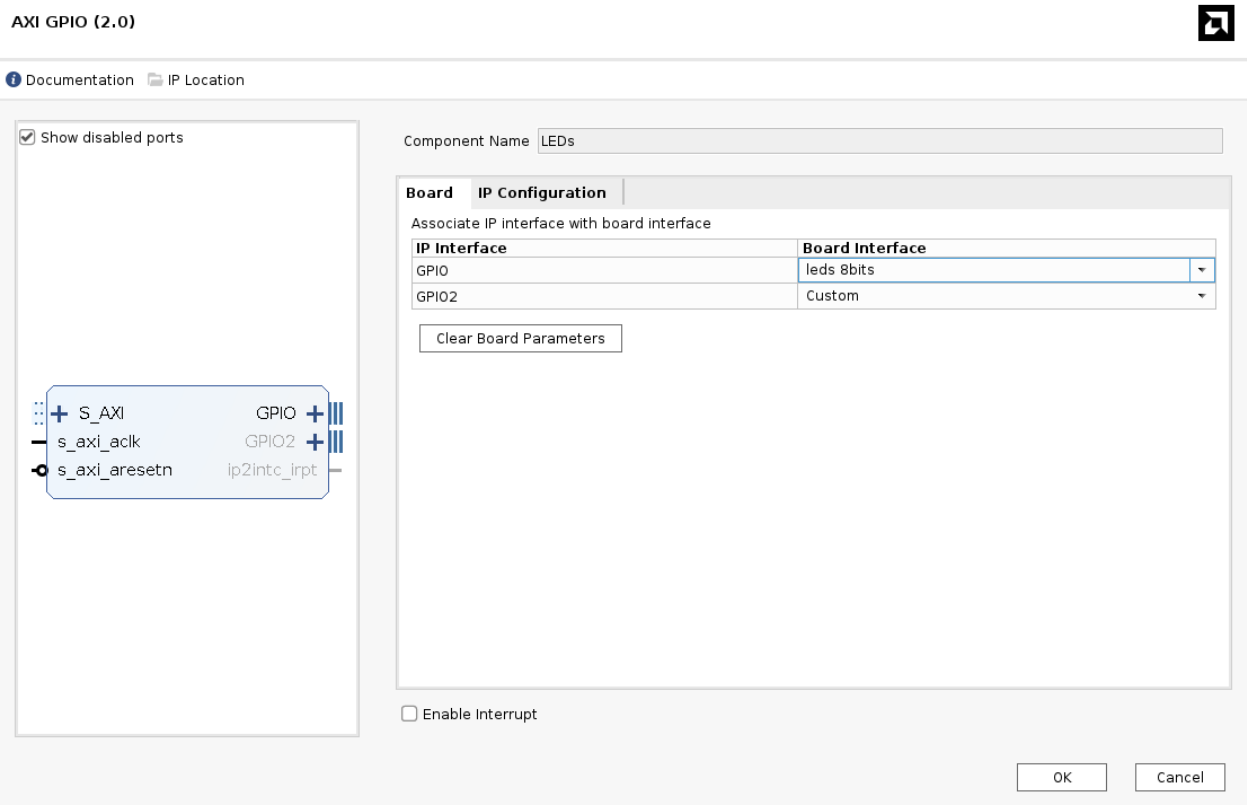


Figure 12: The settings menu for the "LEDs" GPIO IP.

10. While we do not need to mess with it for this example, the "IP Configuration" tab seen in Figure 12 is where the direction of the GPIO channels, if both channels are used, etcetera can be modified. Also note that at the bottom of Figure 12 there is a check box for enabling interrupts for the module. For our example, all we need to do is change the

"Board Interface" setting beside GPIO. In most cases, this can be left as "Custom," as then it is possible to create external pins that are mapped directly to the ports on the ZedBoard or whatever development board you are using. Some boards, however, have some predetermined pins, usually for things like switches, buttons, and LEDs that are built into the board. Click on the "Board Interface" dropdown and change it from "Custom" to "leds 8bits" as shown in Figure 12.

11. Once this is done, click "OK" to return to the block diagram. Click the "Run Connection Automation" link in the green bar at the top of the window. Click the box marked "All Automation" in the window that opens and click "OK" to run the automation. This will automatically generate the AXI interconnect and resets necessary to link the processing

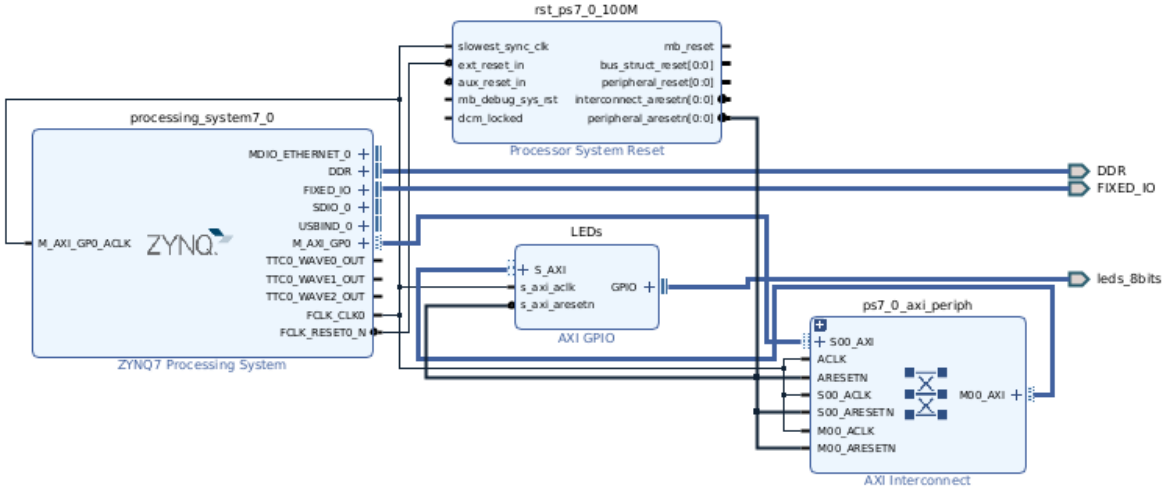


Figure 13: The full block diagram for the "ZedBoard_tutorial" project.

system to the peripheral. The block diagram should now feature a "Processor System Reset" and an "AXI Interconnect" IP, as shown in Figure 13.

12. The last thing that needs checked prior to compiling the block diagram is that memory addresses were correctly allocated for our IPs. Click on the "Address Editor" tab that is above where the block diagram is shown. If an address range is shown for the "LEDs" IP, we can continue. If not, clicking on the "Master Base Address" section for the desired IP should allocate memory for it.
13. Perform the following steps to generate hardware that can be exported to Vitis for the next steps:
    i. Go to the "Sources" tab, click on the "Design Sources" folder, and then right-click on the "zedboard_tutorial.bd" file as shown in Figure 14. Click "Create

HDL Wrapper" to generate a top-level Verilog module for your design. Simply click "OK" on the window that appears and then click "OK" on the "Critical Messages" screen that appears. The four warnings about negative skew values are something that appears in nearly every project, and I was advised to disregard it.

ii. Going back to the "Design Sources" folder, click on the "zedboard_tutorial_wrapper" dropdown and then right-click on the block diagram file again, this time clicking "Generate Output Products" as shown in Figure 14. Click "Generate" on the window that appears.

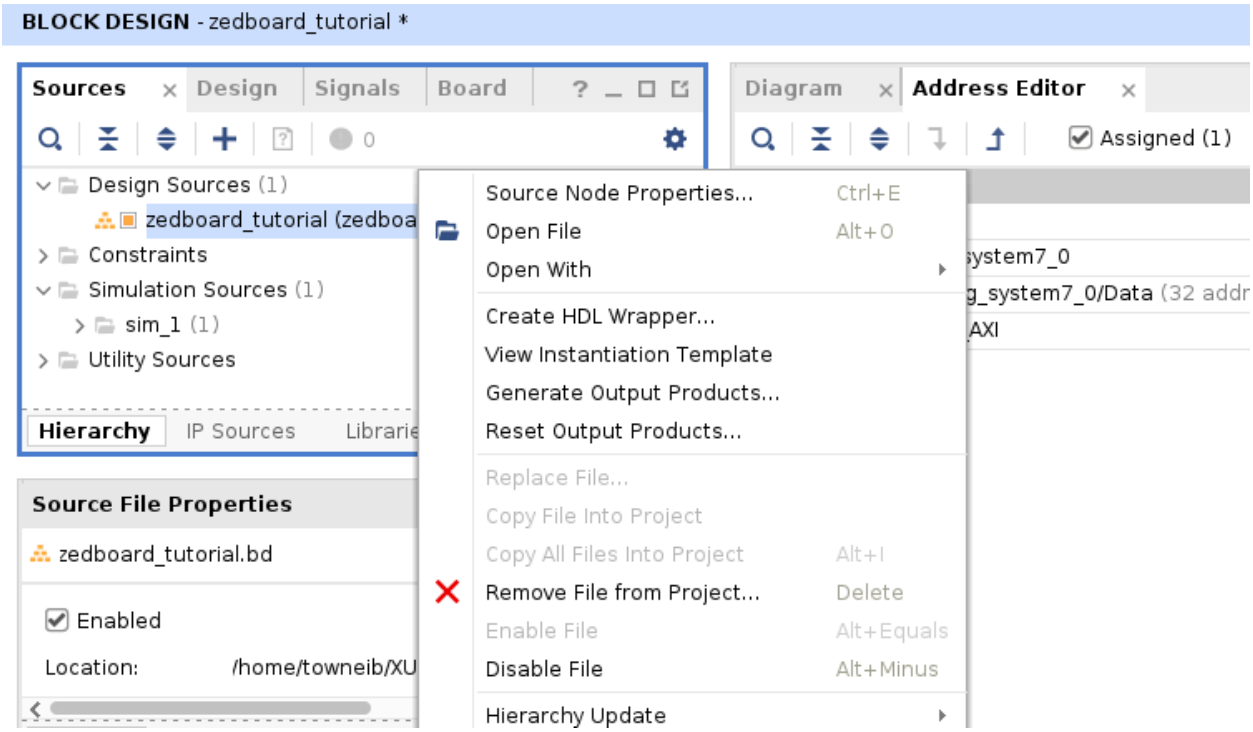
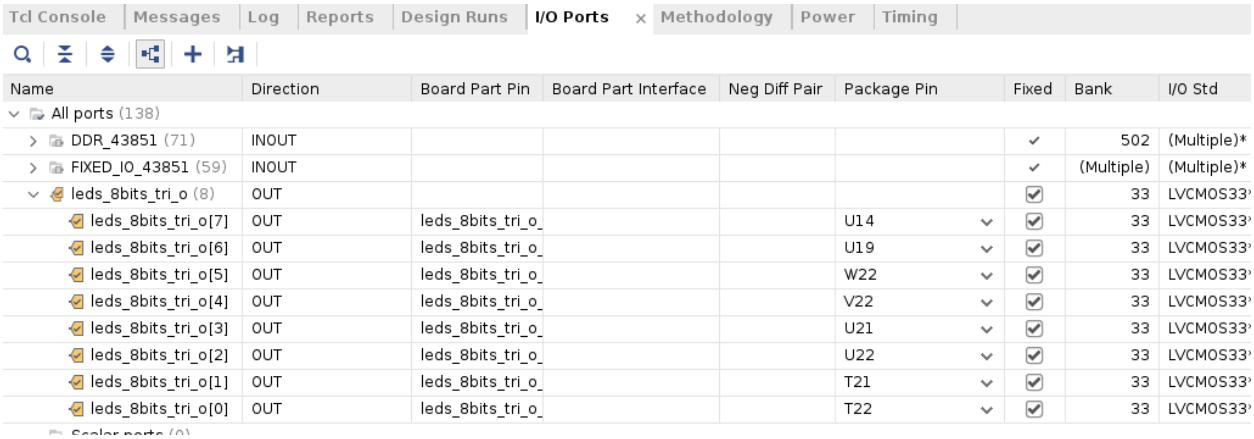


Figure 14: The "Sources" tab and the context menu opened by right-clicking on the block diagram file.

iii. Click "Run Implementation" under the "Implementation" tab on the left side of the screen. Use the default settings in the windows that open. Once it finishes running, click "Open Implemented Design."

iv. Once the implemented design is opened, we need to make sure the I/O pins are correctly assigned. To do this, go to the "Window" menu at the top of the application and click on "I/O Ports." This will open a window like the one shown in Figure 15. In future projects where custom ports are used, it will be necessary to click on the "Scalar ports" dropdown and assign package pins and voltage



Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (138)								
DDR_43851 (71)	INOUT					✓	502	(Multiple)*
FIXED_IO_43851 (59)	INOUT					✓	(Multiple)	(Multiple)*
leds_8bits_tri_o (8)	OUT					☑	33	LVCMOS33*
leds_8bits_tri_o[7]	OUT	leds_8bits_tri_o_			U14	☑	33	LVCMOS33*
leds_8bits_tri_o[6]	OUT	leds_8bits_tri_o_			U19	☑	33	LVCMOS33*
leds_8bits_tri_o[5]	OUT	leds_8bits_tri_o_			W22	☑	33	LVCMOS33*
leds_8bits_tri_o[4]	OUT	leds_8bits_tri_o_			V22	☑	33	LVCMOS33*
leds_8bits_tri_o[3]	OUT	leds_8bits_tri_o_			U21	☑	33	LVCMOS33*
leds_8bits_tri_o[2]	OUT	leds_8bits_tri_o_			U22	☑	33	LVCMOS33*
leds_8bits_tri_o[1]	OUT	leds_8bits_tri_o_			T21	☑	33	LVCMOS33*
leds_8bits_tri_o[0]	OUT	leds_8bits_tri_o_			T22	☑	33	LVCMOS33*

Figure 15: The I/O Ports tab of the implemented design. Note that the pins and I/O standards for the LEDs are already defined, as they are part of the board itself.

standards to every port. In this case, because we are using predefined I/O ports, we do not need to do this.

v. Now that the I/O ports have been confirmed, click “Generate Bitstream” under the “Program and Debug” tab on the left side of the screen. Use the default settings on the windows that open. Click “Cancel” on the window that opens once the generation has finished.

14. It is finally time to export the generated hardware and bitstream to Vitis to create the final SoC project. To export the hardware, go to File, Export, Export Hardware as shown in Figure 16. This will open an exporting wizard.

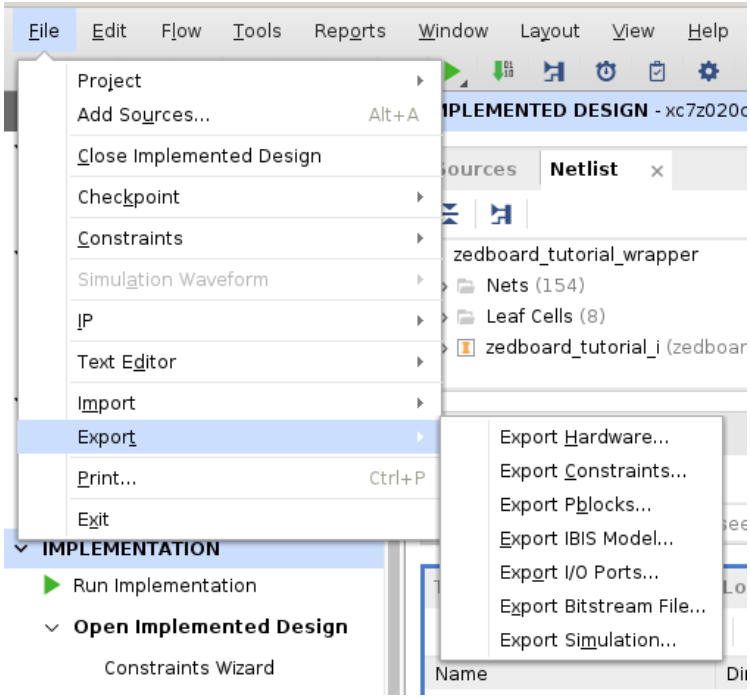


Figure 16: The file menu leading to the “Export Hardware” option.

15. In the wizard, on the “Output” page select “Include bitstream” so that we can package that in with the program bootloader later, use the default settings for the file location, and click “Finish” to generate the .xsa file that contains the platform we will build our C script from. Once this is generated, you can close out of Vivado.

Setting Up Vitis and Creating a Project:

This section will go through setting up a Vitis workspace, importing hardware, creating an application project, and exporting a bootloader for a project on the ZedBoard SoC that will flash the eight LEDs found on the board. The general practices in this section can be applied to other projects as well. See [3] for the tutorial this section was based off of.

1. Open Vitis. It should look similar to Figure 17.

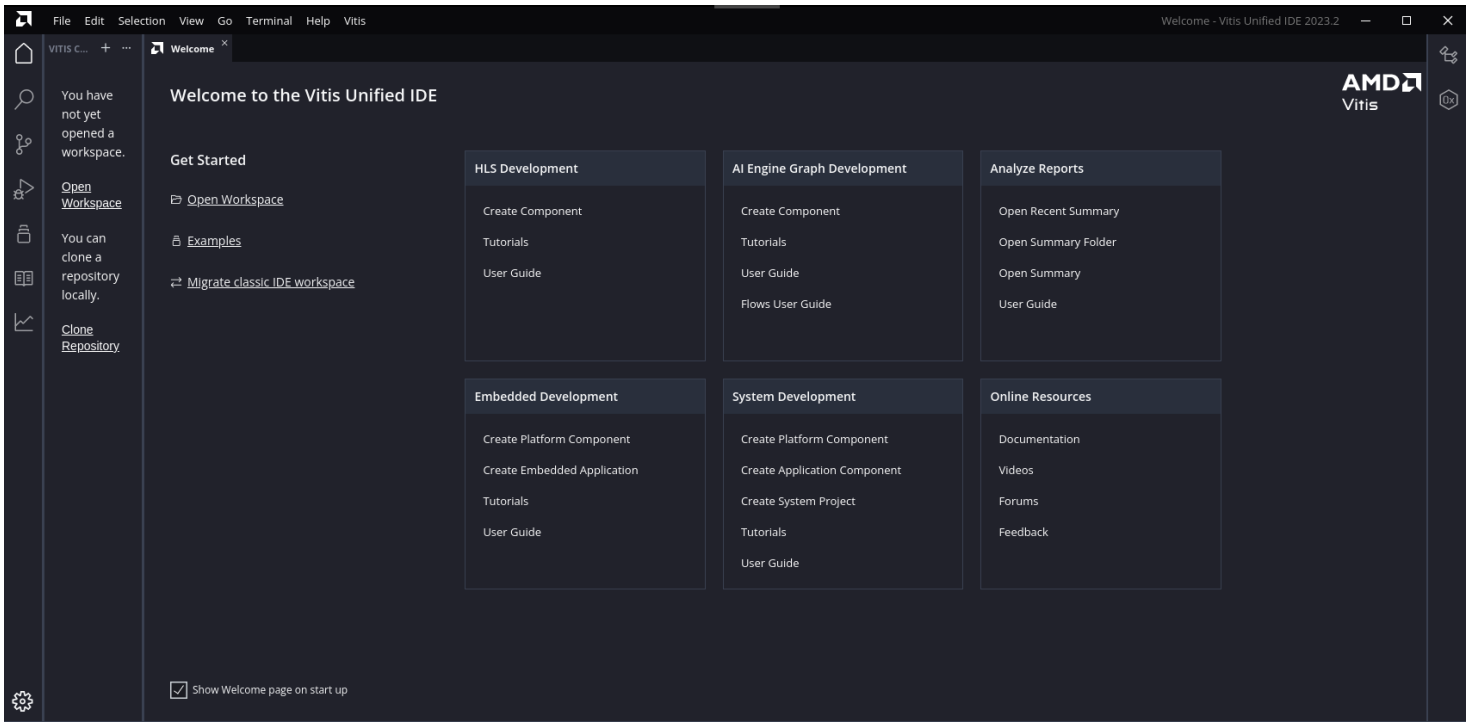


Figure 17: The Vitis welcome screen.

2. Click “Open Workspace” and select where you would like to store your files. I personally created a “Vitis_Workspaces” folder and created a “ZedBoard_tutorial” folder within that to keep my Vitis files separate from my Vivado files while grouping individual projects together. Selecting a workspace will cause Vitis to reinitialize and look like it is crashing, but it is supposed to do that.

3. Next, we have to create a platform component to serve as a base for our application. This is where the .xsa file we exported from Vivado will come in. Go to File, New Component, Platform to open the platform wizard. Call the platform "ZedBoard_tutorial_platform" and store it in your Vitis project folder. Click "Next." To get to the "Select Platform Creation Flow" page. Click "Browse" next to the "Select Hardware Design" dropdown and navigate to the .xsa file exported from Vivado in the previous section. It should be in the project folder from the Vivado project. If you followed the same naming schemes I did, it is called "zedboard_tutorial_wrapper.xsa." Click "Next" once this is done.
4. On the "Select Operating System and Processor" page, set the operating system to be "standalone" and the processor to be "ps7_cortexa9_0." The chip on the ZedBoard and Zybo Z7-20 has two ARM cores in it, so it is necessary to specify which one we are using. Make sure the box for generating boot artifacts is checked as shown in Figure 18.

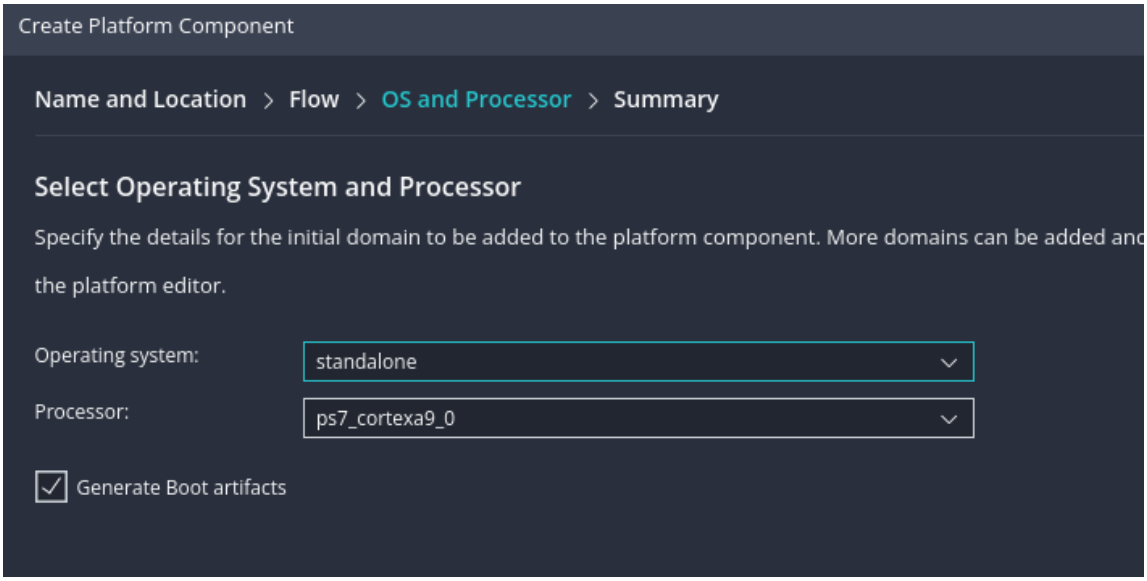


Figure 18: The "Select Operating System and Processor" screen with the correct settings for the ZedBoard_tutorial project.

5. Click "Next" and then "Finish" to create the platform. This will open a window with a summary of the platform as shown in Figure 19. If you click the "Hardware Specification" link on the window, it will read from the .xsa file and report the address map for the processors. If you scroll down on the map for the ps7_cortexa9_0 processor, you should be able to see an entry for the "LEDs" module created in the previous section. **If you cannot find the AXI GPIO IP, please redo the export procedure in the**

previous section. If the addresses are not mapped correctly, you will be unable to interface with the IP in subsequent parts. The entry will look like Figure 20.

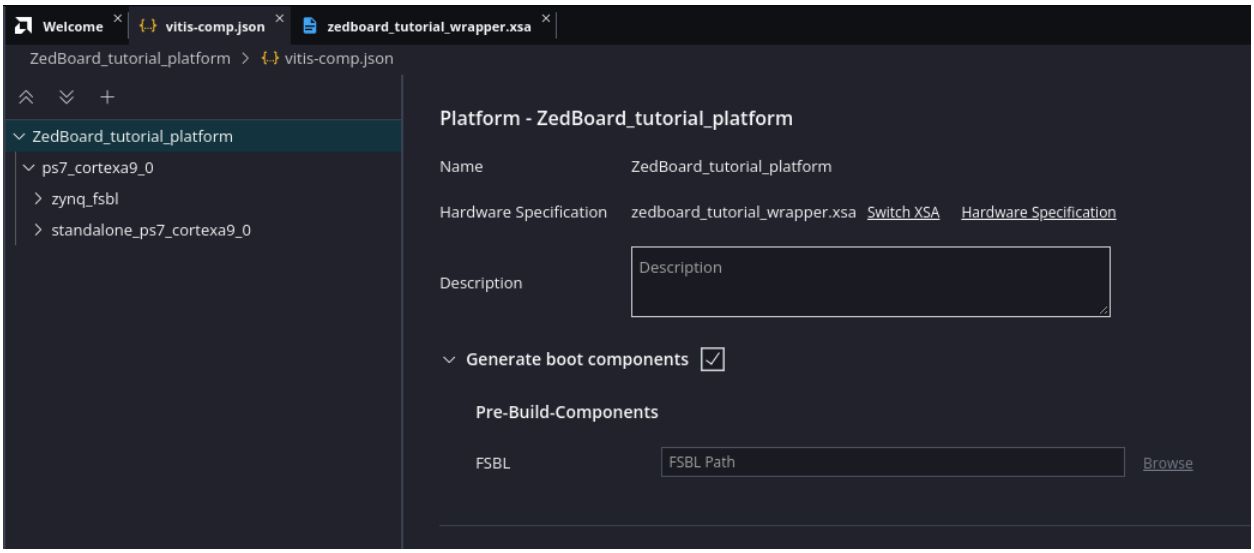


Figure 19: The platform summary screen for the platform imported from Vivado.

CELL	BASE ADDRESS	HIGH ADDRESS	SLAVE INTERFACE	ADDR RANGE TYPE
ps7_scugic_0	0xf8f00100	0xf8f001ff		REGISTER
ps7_ethernet_0	0xe000b000	0xe000bfff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
ps7_dma_ns	0xf8004000	0xf8004fff		REGISTER
ps7_sd_0	0xe0100000	0xe0100fff		REGISTER
LEDs	0x41200000	0x4120ffff	S_AXI	REGISTER
ps7_gpv_0	0xf8900000	0xf89fffff		REGISTER

Figure 20: An example address map for the ps7_cortexa9_0 processor. Note that the LEDs IP has its own row and the addresses match what was in the address editor in Vivado.

6. Once the platform has been created, we can create an empty application to start development in. To find example templates of projects, including one for an empty application, go to the bar on the left side of the screen and click the icon that looks like a three-tiered cake. This will open the “Examples” tab. Scroll down to “Embedded Software Examples” and select the “Empty Application” template. Click “Create Application Component from Template” on the window that appears as shown in Figure 21.

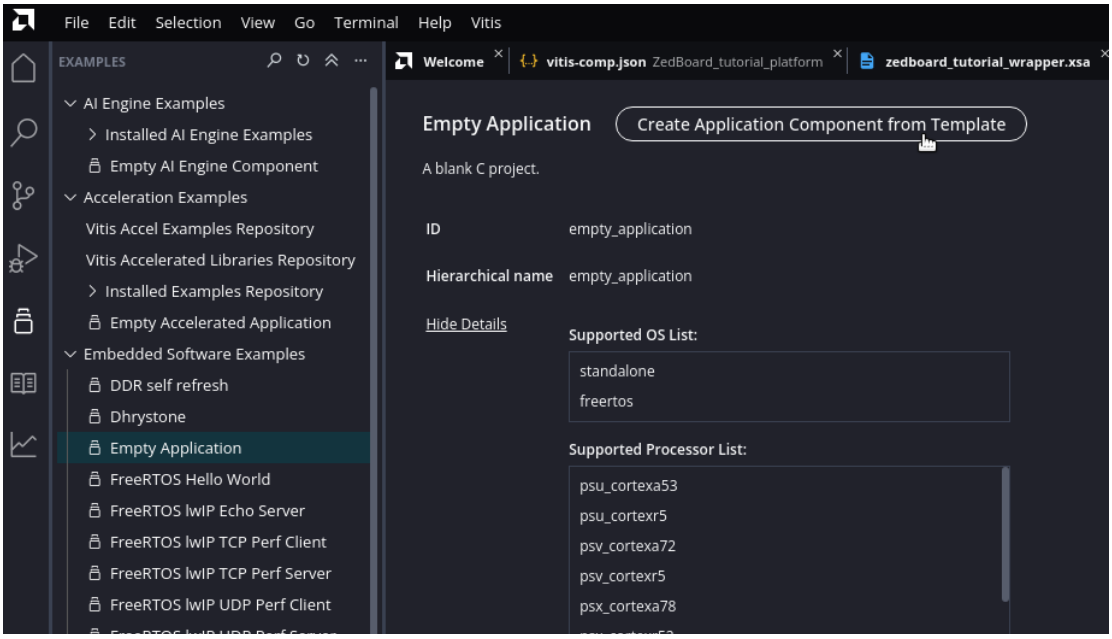


Figure 21: The "Examples" tab and the window for creating an empty application.

7. Name the application "blink_led" and click "Next." Select the "ZedBoard_tutorial_platform" that was imported earlier and click "Next." Select the "standalone_ps7_cortexa9_0" as the domain (this is the processor we will be running our code on) and click "Next" and then "Finish" to create the application.
8. On the summary page that appears, click on the "Navigate to BSP Settings" button. BSP stands for Board Support Package. This is effectively a massive library of header files and drivers that are generated based on the platform the application is running on. These header files allow us to interact with GPIO, get the addresses of different peripherals, and more. Another thing they let us do is make FSBLs, or First-Stage Bootloaders, which we will be using to package our code. However, before we can generate an FSBL, we need to include two libraries in the BSP. Clicking the "Navigate to BSP Settings" button will bring you back to the platform summary screen shown in Figure 19, but a few levels down as shown in Figure 22. Under "Supported Libraries" check the boxes for xilffs and xilrsa as shown in Figure 22. Once this is done, click the "Regenerate BSP" button also shown in Figure 22 to make sure the BSP contains the new changes. This process enables us to make FSBLs for **this application only**. The same settings will need to be applied to other projects if you want to make an FSBL.
9. To make sure that the settings are built into the platform we made, in the "Vitis Components" tab (if you are still in the "Examples" tab, clicking the house icon will open

the components tab), click on the platform object and click “Build” in the “Flow” window below it. This will rebuild the platform to incorporate the changes in the BSP.

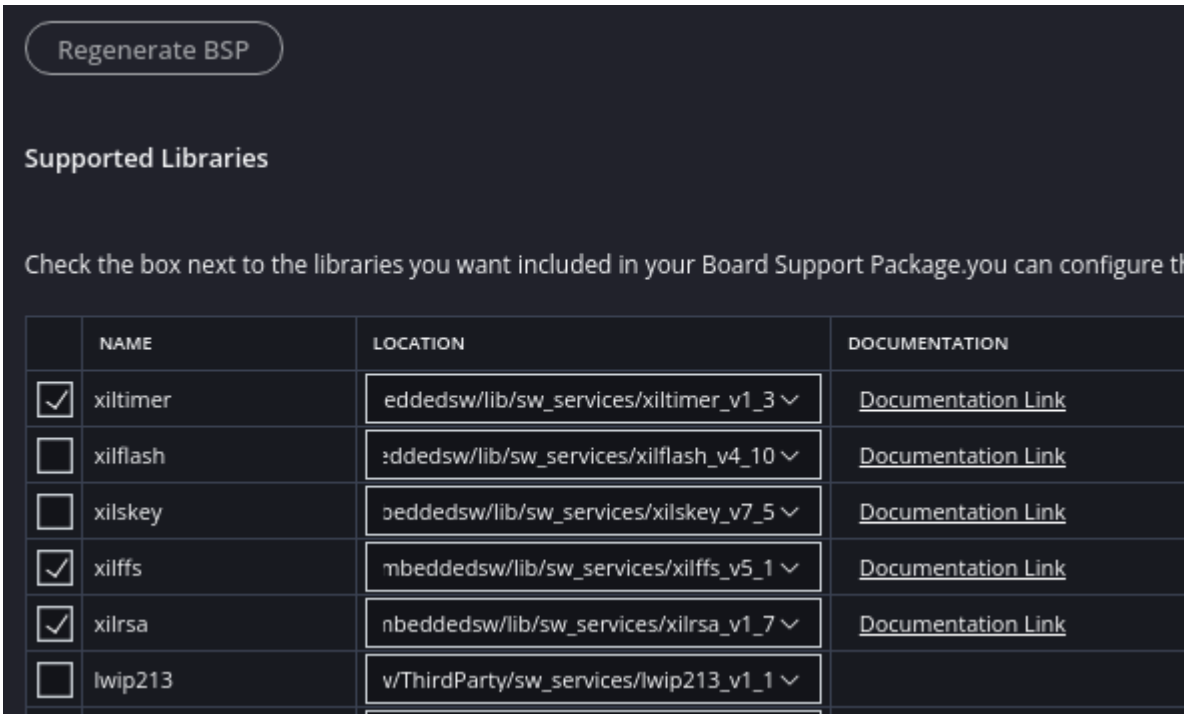


Figure 22: The BSP page where the xilffs and xilrsa libraries have been added.

10. Now that the BSP has been configured, we can write the blink code. Click the dropdown under the application in your “Vitis Components” tab click “Sources,” right-click on “src,” and create a new file named “blink.c.” **Something to note when using Vitis is that sometimes Vitis will indicate that there are errors in code that aren’t actually errors. To fix this, make sure to build your project to ensure that the BSP and other libraries are properly incorporated into it. See Step 13 below for more information.**
11. **The first step of developing in Vitis should always be to include two header files: xparameters.h and xgpio.h.** The former is the most important header file, as it contains the addresses and parameters of everything in the hardware platform. Without it, you cannot do SoC design. The latter contains drivers for configuring the AXI GPIO IPs. If you do not have any in your design is not necessary to include it.
12. The blink code we will be using is depicted in Figure 23. It is an adaptation of the basic blink code in [4]. A breakdown of the code in Figure 23 is below. **While this program only shows the configuration and initialization of XGpio objects, the same generalized procedure of looking up a configuration, configuring a peripheral, and**

reading and writing to it applies to most IPs that can be interfaced with, even those that are not under the umbrella of XGpio):

blink_led > src > C blink.c > ...

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

int main() {
    XGpio_Config *config_ptr;
    XGpio leds;

    // Place holder for if a function successfully completed or not
    int status;

    // Pull the configuration for the LEDs
    config_ptr = XGpio_LookupConfig(XPAR_LEDS_BASEADDR);

    // Initialize the leds object
    status = XGpio_CfgInitialize(&leds, config_ptr, config_ptr->BaseAddress);
    if(status != XST_SUCCESS)
        xil_printf("SOMETHING WENT WRONG");

    // Set the data direction to output
    XGpio_SetDataDirection(&leds, 1, 0x00000000);

    // Create a placeholder for the led data
    uint8_t led_mask = 0xff;
    int i;
    while(1) {

        // Write led_mask to the leds
        XGpio_DiscreteWrite(&leds, 1, led_mask);

        // Invert led_mask to create a blink effect
        led_mask = ~led_mask;

        // Delay timer before next write
        for(i = 0; i < 5000000; i++);
    }
}
```

Figure 23: The completed blink.c file.

- All the peripherals that will be interfaced with will be interfaced with through pointers. They will first need to be configured, and then they can be written to and read from. In the case of the AXI GPIO, the configuration data can be stored in the “XGpio_Config” pointer object found in line 5. Line 6 has the definition for the GPIO object that we will be configuring for our LEDs.

- Line 12 looks up the configuration data, which includes things like the base address and such for the peripheral. The “XPAR_LEDS_BASEADDR” may have a different name if you did not name your module “LEDs” like I did, but it will still be found in xparameters.h, along with things like if there is an interrupt enabled, what the bus width is, etcetera. **Once again, all these constants are stored in xparameters.h, making it the most important header file for SoC development in Vitis.**
- Line 15 takes in a pointer for the XGpio object we want to configure, its configuration data, and its base address and initializes the passed in XGpio object. This must be done before any object can be used, regardless of if it is a timer, a GPIO, or an ADC. The return value of functions like this will either be XST_SUCCESS or XST_FAILURE, which can be used to catch errors in initialization via if statements like the one starting at line 16.
- Line 17 uses xil_printf to print if something went wrong with configuring the XGpio. xil_printf specifically prints via UART 1, which is auto-configured when “Run Block Automation” is used in Vivado. The baud rate for the network is 115200, and you can hook up your computer to UART 1 serially from one of the USB A ports on the development board.
- Line 20 sets the data direction for the XGpio object. The first parameter is a pointer to the XGpio object, the second parameter is what GPIO channel is being modified (recall that each AXI GPIO has two channels, with the default channel being channel 1), and the third parameter is the mask for making the pin output or input. Each GPIO channel has a maximum width of 32 bits, so it is easiest to use a 32-bit value to set the data direction. **If a bit is set to 0, that wire is an output. If a bit is set to 1, that wire is an input.**
- Line 23 defines a mask for writing to the LEDs on the board. The convention is that 1 is on and 0 is off, so writing 0xFF will turn all eight LEDs on, writing 0x11 would turn the first and fifth LEDs on and turn the rest of, etcetera. Note that because we only have eight LEDs, we only need an eight-bit value to control all the LEDs.

- Line 28 is where data is written to the LEDs. The first parameter is a pointer to the XGpio object, the second parameter is the GPIO channel being written to, and the third parameter is the data to be written. **The function definition for this, reading, and other functions can be found in xgpio.h.**
- Lastly, line 34 is a loop to make a buffer between turning on and turning off the LEDs so it can be perceived by the human eye. For reference, the clock speed of the CPU on the ZedBoard and Zybo Z7-20 is 667 MHz, and the programmable logic clock is 100 MHz.

13. Now that the code has been written, the last step is to build the project and create a boot image. To build the project, much like with building the platform, click on the application in the "Vitis Components" tab and then click "Build" under the "Flow" tab below it, as

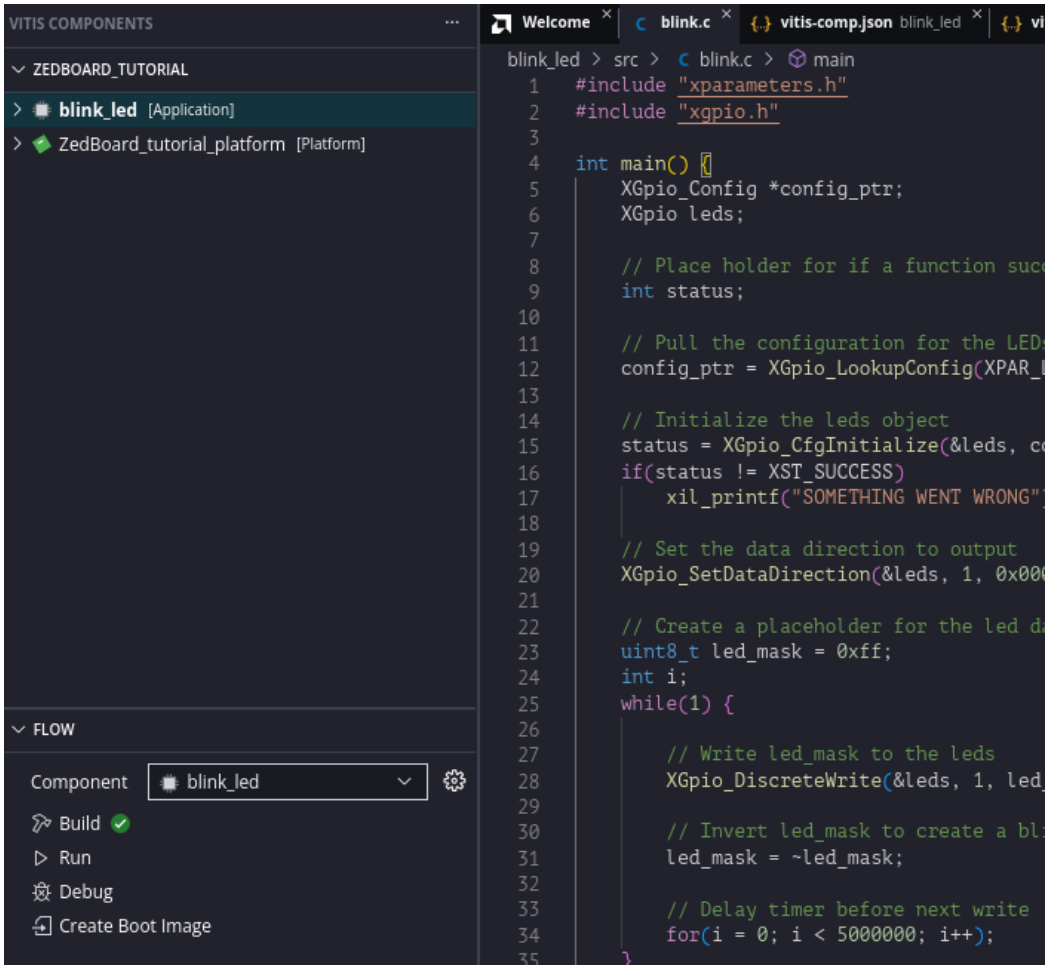


Figure 24: The "Vitis Components" and "Flow" tabs when the blink application is selected.

seen in Figure 24. If it builds correctly, click the “Create Boot Image” button in that same menu.

14. A window will open like shown in Figure 25. In the left column two files can be seen, fsbl.elf, which is the FSBL itself, and blink_led.elf, which contains the built blink script. A third component must be added to this image as we are using programmable logic in our design: the bitstream file we made in Vivado. To add this, click the “+” button and click “browse” to open a file explorer where you can locate the bitstream file. It should be located within the folder for the Vivado (not the Vitis) project created in the previous section under the “impl_1” folder in the folder ending in “.runs.” If you followed the same naming conventions as me, it will be named zedboard_tutorial_wrapper.bit. Once you select the file, keep the type as “datafile” and click “OK.” **If the bitstream from Vivado is already shown in the window, there is no need to add the file again.**

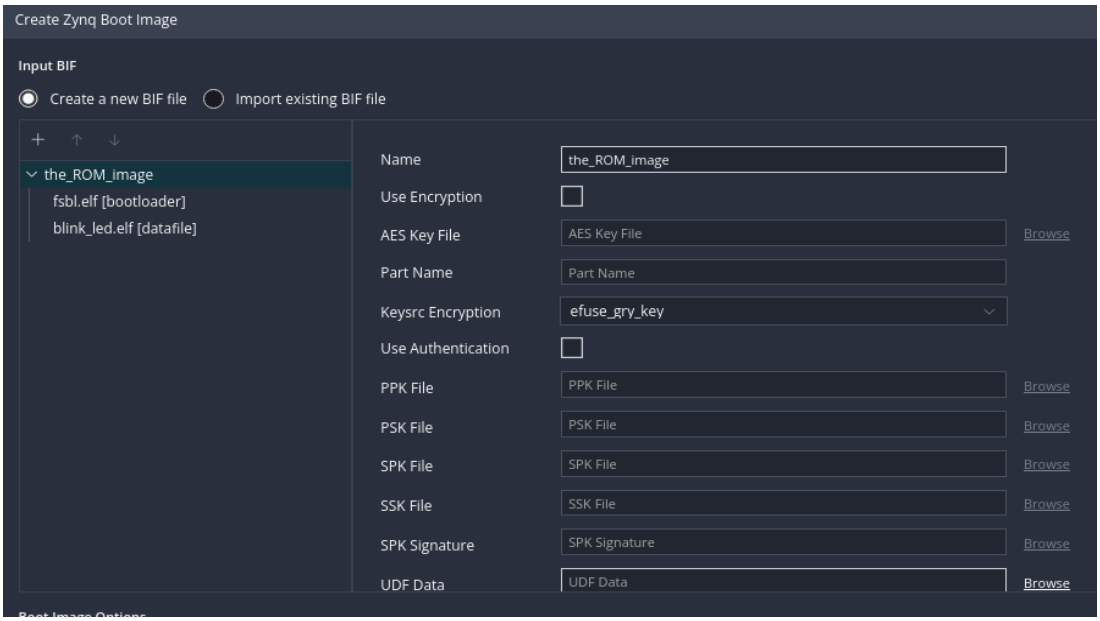


Figure 25: The default window once the “Create Boot Image” button is clicked.

15. Before designating a file path for the output image (I would suggest making that the same folder as your Vitis project), the order of the data files within the image must be changed. The order should be the FSBL, the bitstream, and then the application file. To adjust this, click on the newly added .bit file and click the up arrow to the left of the “+” button used to add files. This will change the order of the files, making it look like Figure 26 once the output file path has also been specified.

16. Click “Create Image” to create the boot image. **Something to note is that the output image can be named whatever you want when saving it, but when it goes on the SD card to program the ZedBoard or Zynq Z7-20, it must be named BOOT.bin or it will not work as intended!** Once the image is created, simply go to where it is saved and copy and paste it from the VM to your personal desktop. The next step is to program the board with the boot image.

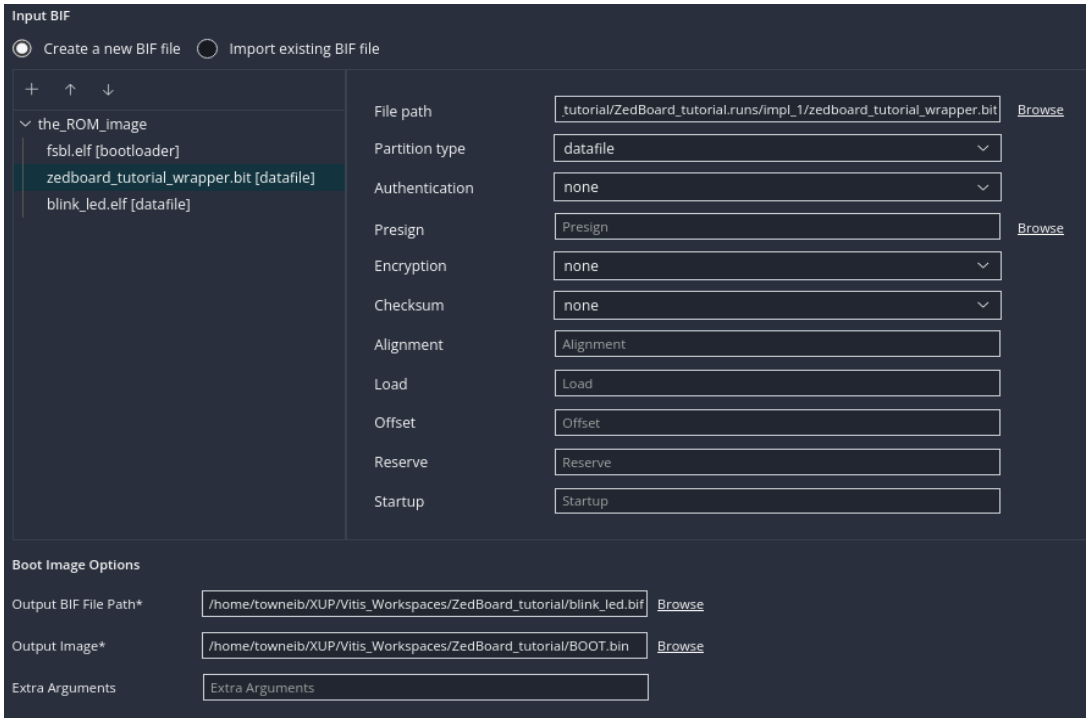


Figure 26: The “Create Boot Image” window once the bitstream file has been added, the order of the files changed, and an output file path specified.

Programming the ZedBoard:

This section covers using a boot image to program the ZedBoard via an SD card. This section will also be significantly shorter than the others, as it is relatively simple to load a boot image onto the ZedBoard.

1. **Take an SD card, remove all files from it, and copy only the BOOT.bin file onto it. Remember, it must be named BOOT.bin or this will not work!**
2. There is a row of five jumpers that is right below the Diligent logo on the ZedBoard. These must be arranged to allow loading from an SD card. There is a table in [5] that lists the jumper configurations for different protocols. On the ZedBoard, the pins should look

like Figure 27. On the Zybo Z7-20 board, the configuration is simpler, as you only have one jumper to move as shown in Figure 28.

3. Power on the board and wait while it programs. If you see flashing lights, you successfully have completed this tutorial!

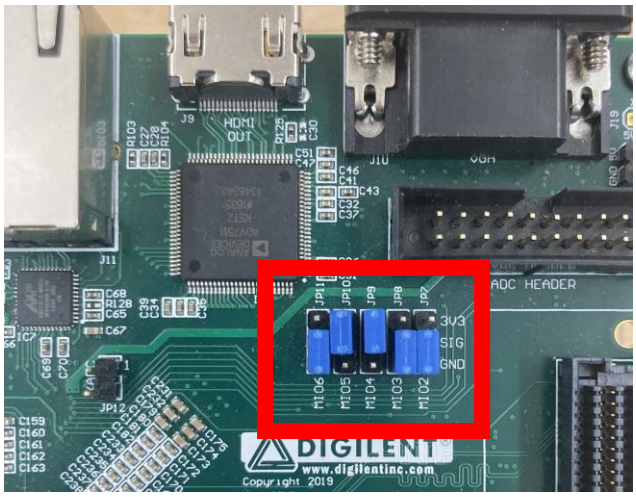


Figure 27: The ZedBoard jumper configuration to load a boot image from an SD card.

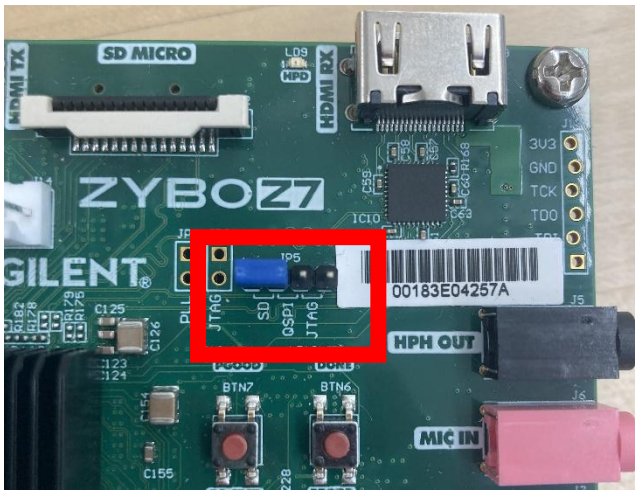


Figure 28: The Zybo Z7-20 jumper configuration to load a boot image from an SD card.

References:

[1] "How to install Vivado 2023.2 on Linux - tomas-fryza/vhdl-course GitHub Wiki," *Github-wiki-see.page*, 2023. https://github-wiki-see.page/m/tomas-fryza/vhdl-course/wiki/How-to-install-Vivado-2023.2-on-Linux.

[2] "AMD Technical Information Portal," *Amd.com*, 2024. https://docs.amd.com/r/en-US/ug1400-vitis-embedded/Installing-the-Vitis-Software-Platform.

[3] Xilinx, "Embedded-Design-Tutorials/docs/Getting_Started/Zynq7000-EDT/2-using-zynq.rst at master · Xilinx/Embedded-Design-Tutorials," *GitHub*, 2020. https://github.com/Xilinx/Embedded-Design-Tutorials/blob/master/docs/Getting_Started/Zynq7000-EDT/2-using-zynq.rst.

[4] A. Brown, "Create a Main C Source to Control AXI GPIO Peripherals - Digilent Reference," *Digilent.com*, 2020. https://digilent.com/reference/programmable-logic/guides/vitis-create-blinky-software.

[5] "ZedBoard Zynq™ Evaluation and Development Hardware User's Guide," 2012. Available: https://digilent.com/reference/_media/zedboard:zedboard_ug.pdf

[#] "Vitis Embedded Software Debugging Guide Overview — Embedded Design Tutorials 2021.2 documentation," *Github.io*, 2021. https://xilinx.github.io/Embedded-Design-Tutorials/docs/2021.2/build/html/docs/Vitis-Embedded-Software-Debugging/docs/1-xilinx-debug-solution-overview/README.html.

Updates to the Memo:

- Isaac Towne, 10/30/2024: Added sections on making projects in Vivado and Vitis
- Isaac Towne, 10/31/2024: Added information about setup script in Vitis files for library installation and pictures for jumper positions for loading from SD card
- Isaac Towne, 11/10/2024: Updated blink code to make it clearer that only a uint8_t is needed for configuring the LEDs, added section about regenerating the BSP prior to starting to program
- Isaac Towne, 11/17/2024: Added a note about building an application project to remove any false errors from the BSP and relevant libraries not being integrated into the project
- Isaac Towne, 5/20/2025: Added parts about adding the Nexys-A7-100t board to the board files as well and updated relevant figures