# Baremetal_VivadoVitis

Parsa Vares, Hakan Hasan

January 2025

## Introduction

The demand for efficient and optimized computing systems has led to the exploration of low-level programming paradigms that leverage hardware resources directly. In this context, the project titled *Baremetal Development with Vivado and Vitis* investigates the implementation and management of hardware designs and software applications at the baremetal level, targeting FPGA platforms.

Field-Programmable Gate Arrays (FPGAs) have become a cornerstone in modern embedded systems due to their flexibility, parallel processing capabilities, and real-time responsiveness. These characteristics make FPGAs a preferred choice for applications in domains such as signal processing, artificial intelligence, and communication systems. However, exploiting the full potential of FPGA-based systems requires a deep understanding of hardware-software co-design, alongside proficiency in development environments like Vivado for hardware design and Vitis for software application development.

This project focuses on designing, implementing, and testing a baremetal application on an FPGA platform using the Xilinx Vivado and Vitis toolchains. The objective is to establish a workflow that integrates hardware design and software development seamlessly, ensuring efficient use of FPGA resources. The project also emphasizes understanding the interaction between the hardware abstraction layer (HAL) and low-level drivers to achieve optimal system performance.

The scope of this project includes the following:

- Development of an FPGA hardware design using Vivado.
- Implementation of a baremetal software application using Vitis.
- Integration of hardware and software components.
- Testing and validation of the implemented system.

By pursuing this project, significant insights are gained into the challenges and best practices of baremetal programming on FPGA platforms. This knowledge contributes to a broader understanding of hardware-software integration in resource-constrained environments, providing a foundation for future advancements in embedded system design and development.

## 1 Vivado

We created a Vivado project for Basys 3 Boards as shown in the tutorial. We created a block design named "design_1" and left the rest of the fields as default.

To add a processor to the block design we followed the "Add a Microblaze Processor to a Block Design" section, because the Basys 3 Board has an Artix-7 device. We followed the steps for boards without DDR memory. The first step there is to right-click on the system clock in the Board tab and select Connect Board Component. However, this is not possible, so we add the Clocking Wizard IP using the Add IP button. Then click the Run Connection Automation button and connect the system clock and the system reset.

We added the MicroBlaze IP to the design and clicked Run Block Automation. In the Run Block Automation dialog, we set the following settings (see figure 1).

Then we connect the USB UART interface and Run Connection Automation as shown in the tutorial.

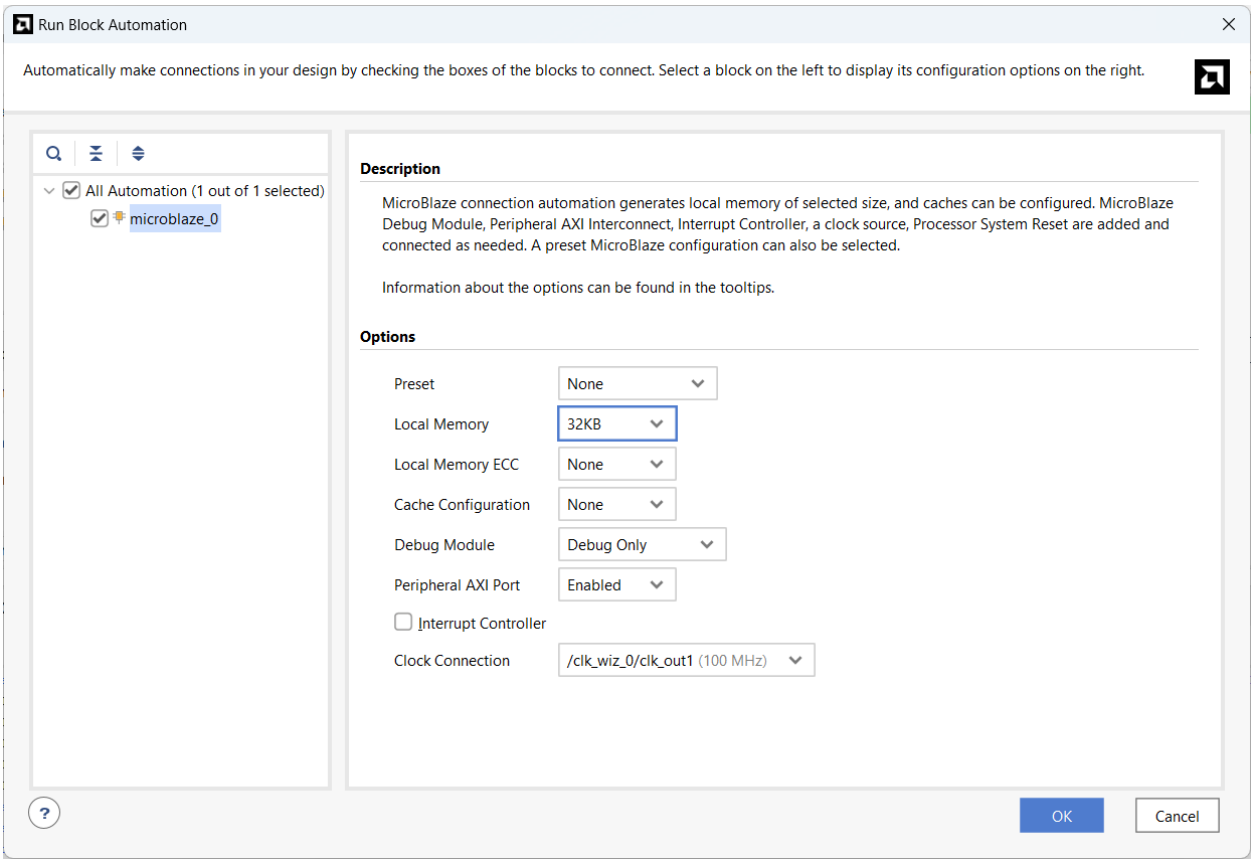


Figure 1: MicroBlaze Block Automation settings.

For adding GPIO Peripherals to the Block Design in the tutorial, two methods were shown. For connecting both the LEDs and the buttons, we used the first method, which takes advantage of the board files to automatically generate constraints. After that, we clicked the Run Connection Automation button to connect the AXI GPIO IP blocks to the Microblaze processor.

While configuring the buttons, we noticed a deviation from the tutorial and common examples. The buttons were set up differently to accommodate five inputs instead of four, as seen in the following configuration:

```
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {btn_tri_io[0]}]#[get_ports rst]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports {btn_tri_io[1]}]#[get_ports b1]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports {btn_tri_io[2]}]#[get_ports b2]
set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports {btn_tri_io[3]}]#[get_ports b3]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {btn_tri_io[4]}]#[get_ports b4]
```

This approach ensured that the buttons were configured correctly for the Basys 3 board, even though it differed from the tutorial. The rest of the GPIO connections followed the standard procedure.

The figure 2 shows the final layout of the block design diagram.

Finally, we validated the block design, created an HDL Wrapper, and built the Vivado project by running it through Synthesis and Implementation, and finally generating a bitstream. These are identical to the instructions and screenshots in the tutorial.

In the end, we exported a Post-Synthesis Hardware Platform with the bitstream included. This setup ensured a seamless integration with the Basys 3 board, leveraging its specific hardware capabilities for our project.

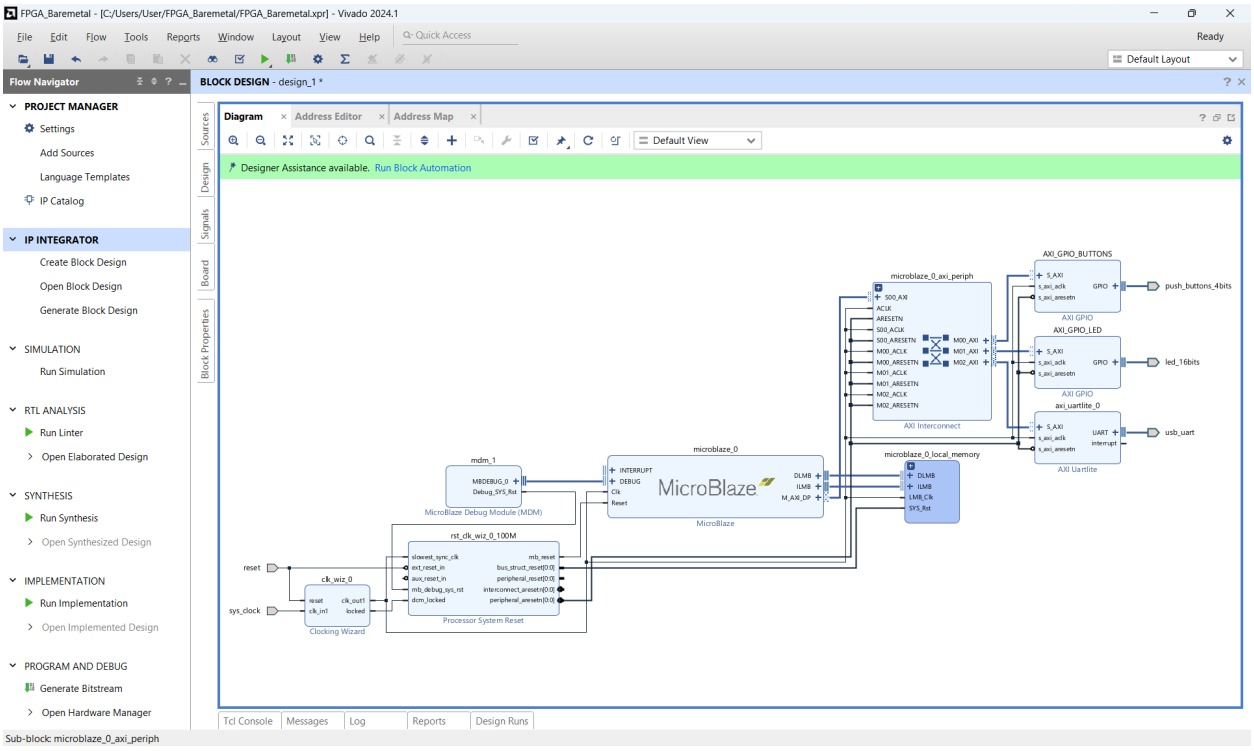


Figure 2: Block Design Diagram Layout.

# 2 Vitis

We are using Vivado and Vitis 2024.1. Since the tutorial was written for an older version of these tools, there are significant differences with the new Vitis Unified IDE. Luckily for us, the installation of version 2024.1 comes with the Vitis Classic IDE, which has an alternative user interface that is more similar to the previous versions. That is why we used it.

We created a New Application Project as shown in the tutorial. We created a main C source to control AXI GPIO Peripherals, and this file contains the code described in the tutorial, only changing the LED_MASK to 0b1111111111111111, since the number of LEDs connected to the GPIO peripheral in the hardware design is 16. (see fig. 3)

We built a Vitis Application ([System] project). We plugged the Basys 3 board into the computer. But instead of using Tera Term or PuTTY as recommended in the tutorial, we used the built-in serial terminal in Vitis, included in its Debug view. We connect to the appropriate serail port and use a baud rate of 9600, because of the Microblaze designs. Then we started the software application in an identical way as shown in the tutorial. And everything works as expected. At the bottom of figure 4, you can see that the text "Entered function main" is printed on the console.

At this link you can see a very short video showing the functioning of the program: `https://drive.google.com/file/d/1kotmfbHRYbTjCSpVj-5XSn_XeJaiLBVE/view?usp=sharing`

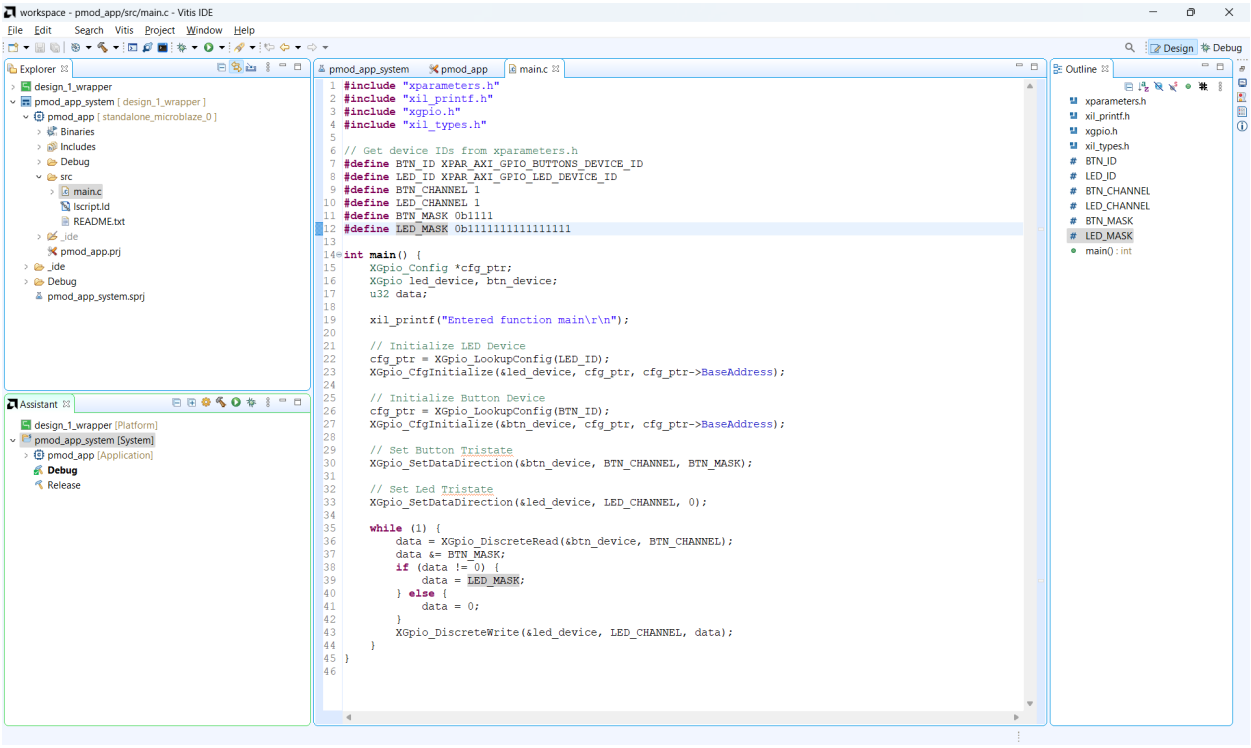


Figure 3: Code of the main.c file.

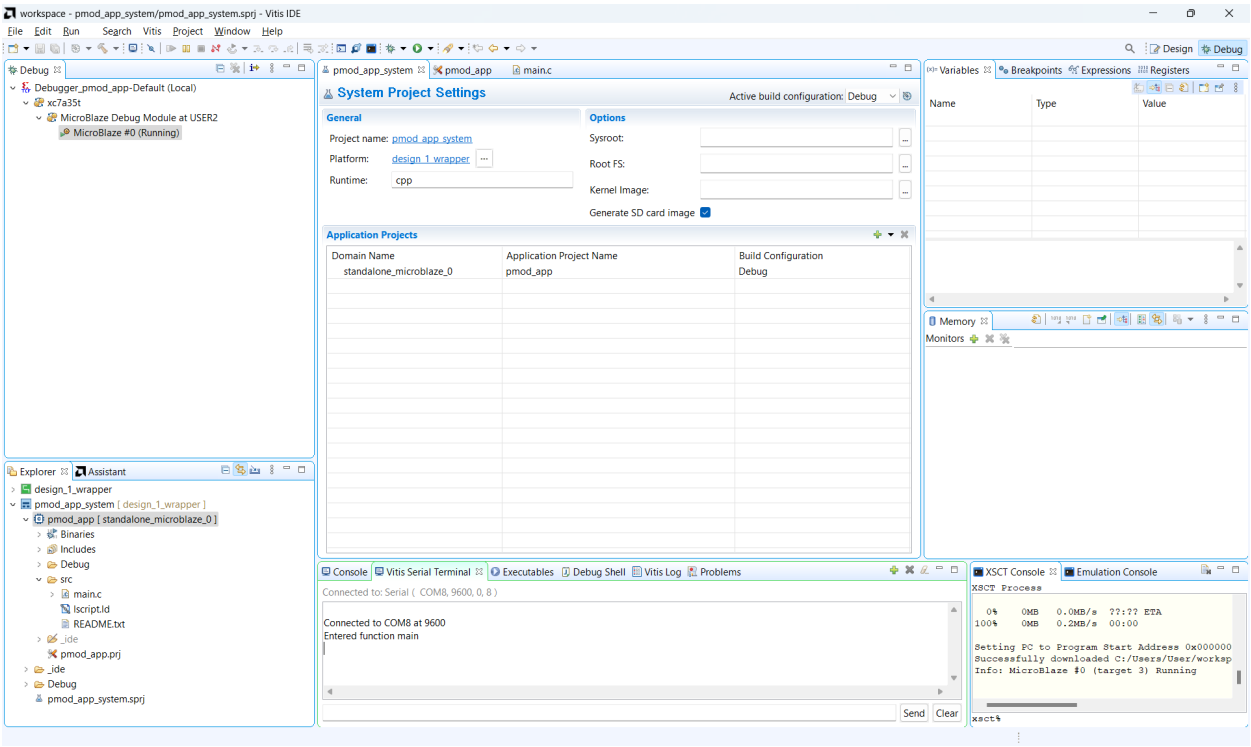


Figure 4: Vitis IDE window with Vitis Serial Terminal Connected to the Basys 3 board.