FPFA Emulation and Live Debugging Framework with UART-Based Traffic Generation

BITS ZG628T: Dissertation

By

Sachin Singh
2022HT01625

Dissertation work carried out at

Synopsys Inc.
DLF Tech Park Sector 143, Noida - 201307

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI (RAJASTHAN)

November 2024

FPFA Emulation and Live Debugging Framework with UART-Based Traffic Generation

BITS ZG628T: Dissertation

by

Sachin Singh
2022HT01625

Dissertation work carried out at

Synopsys Inc.
DLF Tech Park Sector 143, Noida - 201307

Submitted in partial fulfilment of **WILP** degree programme

Under the Supervision of

Nishtha Bhatia
Synopsys Inc.
DLF Tech Park Sector 143, Noida - 201307

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)

November 2024

CERTIFICATE

This is to certify that the Dissertation entitled **FPFA Emulation and Live Debugging Framework with UART-Based Traffic Generation** and submitted by Sachin Singh having ID-No. 2022HT01625 for the partial fulfillment of the requirements of M.Tech. WILP Embedded Systems degree of BITS, embodies the bonafide work done by him/her under my supervision.

____Nishtha Bhatia______

Signature of the Supervisor

Place : ___Noida_____

Date : ___18th Nov 2024____

Sachin Singh
R&D Engineer, Synopsys Inc.
____________Noida_________
Name, Designation & Organization &Location

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)
WILP Division

Organization: Synopsys Inc. **Location:** Noida
Duration: 4 months **Date of Start:** 5th August 2024
Date of Submission: 17th November 2024

Title of the Project: FPFA Emulation and Live Debugging Framework with UART-Based Traffic Generation

Name of the Student: Sachin Singh
ID No.: 2022HT01625

Supervisor's Name: Nishtha Bhatia **Additional Examiner Name:** Shishir Pandey
Designation: Staff Engineer **Designation:** Sr Manager

Name of the Faculty mentor: Sanjay Vidhyadharan

Key Words: FPGA, UART, SPI TFT, Python, C++ Testbench, COE File, Image Processing

Project Areas: Digital Design, FPGA Development, Serial Communication Protocols, Embedded Systems, Display Technologies

Abstract: This project involves transmitting image data from a software testbench to an FPGA using the UART protocol and displaying the image on an SPI TFT display. A Python script converts the image into a COE (Coefficient) file, which is then sent to the FPGA through a C++ testbench. The UART receiver on the FPGA side processes the incoming data, stores it in Block RAM, and once the RAM is filled with 128x160 pixel data, it is transferred to the SPI TFT for display.
Challenges encountered include clock synchronization between the UART transmitter and receiver, limited debugging capabilities on the Spartan 7 FPGA, and managing baud rate discrepancies. Through iterative debugging and testing, the project achieves successful image transmission and display on the SPI TFT. This work highlights the use of FPGAs in embedded system design for real-time image processing and display applications

Sachin Singh
Signature of Student

Date:- 18th November 2024

Nishtha Bhatia
Signature of your Supervisor

Date:- 18th November 2024

ACKNOWLEDGEMENTS:

I would like to express my gratitude to my supervisor, **Nishtha Bhatia**, for their guidance and support throughout this project. I am also thankful to **Synopsys Inc.** for providing the resources and environment necessary to complete this project. Lastly, I would like to thank my colleagues, friends, and family for their encouragement and help during the course of this project.

Contents

List of Figures

List of Abbreviations

Acronym	Description
FPGA	Field Programmable Gate Array
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
COE	Coefficient File (used for initializing memory)
ILA	Integrated Logic Analyzer

1. Introduction

Field Programmable Gate Arrays (FPGAs) are widely used in custom hardware design due to their flexibility and ability to perform parallel processing. They are particularly useful in applications such as signal processing, communications, and control systems, where high-speed data processing is required. However, real-time traffic generation, debugging, and testing of FPGA designs can be challenging, particularly when integrating complex communication protocols like UART (Universal Asynchronous Receiver-Transmitter) and SPI (Serial Peripheral Interface). The need for an efficient testing environment to verify hardware performance in real-world scenarios is crucial.

In this project, a framework is developed to address these challenges by providing an end-to-end solution that integrates image-to-COE conversion, UART communication, FPGA data handling, and real-time signal debugging using Xilinx's Integrated Logic Analyzer (ILA). This enables designers to simulate real-time traffic, process data, and validate hardware designs with minimal manual intervention.

2. Objectives

The primary objectives of the project are:

- Develop a Python script that converts images to COE format, which can be used to initialize FPGA memory.
- Develop a software testbench in C/C++ that transmits COE data to an FPGA over UART.
- Establish a UART-based communication interface between the software and the Spartan-7 FPGA for bidirectional traffic transfer.
- Implement a custom design on the FPGA to receive and process traffic, forwarding it to the design under test (DUT).
- Demonstrate a proof of concept (POC) using an SPI TFT display for converting UART traffic to SPI protocol.
- Utilize the Integrated Logic Analyzer (ILA) in Vivado for live debugging during emulation to improve the design validation process.

3. Literature Review

1.1. FPGA Emulation:

FPGA emulation has become a key aspect of the hardware verification process, particularly in large-scale systems where traditional software simulation becomes too slow. Emulation frameworks like Synopsys ZeBu enable designers to validate complex systems, but they come with high costs. In this project, custom traffic generation and real-time emulation using UART are implemented to provide a cost-effective and flexible solution.

1.2. UART Communication:

UART (Universal Asynchronous Receiver-Transmitter) is a widely used protocol for serial communication. It provides a simple and reliable method of data exchange between the software running on a PC and the FPGA.

1.3. Live Debugging:

Xilinx's Integrated Logic Analyzer (ILA) is a powerful tool for monitoring internal signals in real-time, providing detailed insights into FPGA design behavior. Live debugging helps detect and fix issues during emulation, making the verification process more efficient.

1.4. SPI Protocol:

The SPI (Serial Peripheral Interface) protocol is widely used for communication with peripherals like displays and sensors. Its integration into FPGA designs ensures a seamless interface between external devices and the processing logic within the FPGA.

4. Project Methodology

4.1. System Architecture Overview

The system architecture consists of the following components:

1. **Python Image Conversion Tool**:

   This tool takes any input image and converts it to COE format, resizing it to a 128x160 pixel resolution and encoding it in RGB565 format. The COE file is used to initialize FPGA memory.

2. **C++ Testbench**:

   The testbench reads the COE file generated by the Python script and transmits the data byte by byte over UART to the FPGA.

3. **FPGA Design**:

   The FPGA design includes a UART receiver, RAM for storing the COE data, and an SPI controller for driving the TFT display. Once the image data is stored in RAM, it is displayed on the SPI TFT screen.

4. **Real-Time Debugging**:

   The ILA is integrated into the FPGA design to capture internal signals, monitor data flow, and identify any timing issues during data transmission or display.

Below block diagrams show the system architecture:

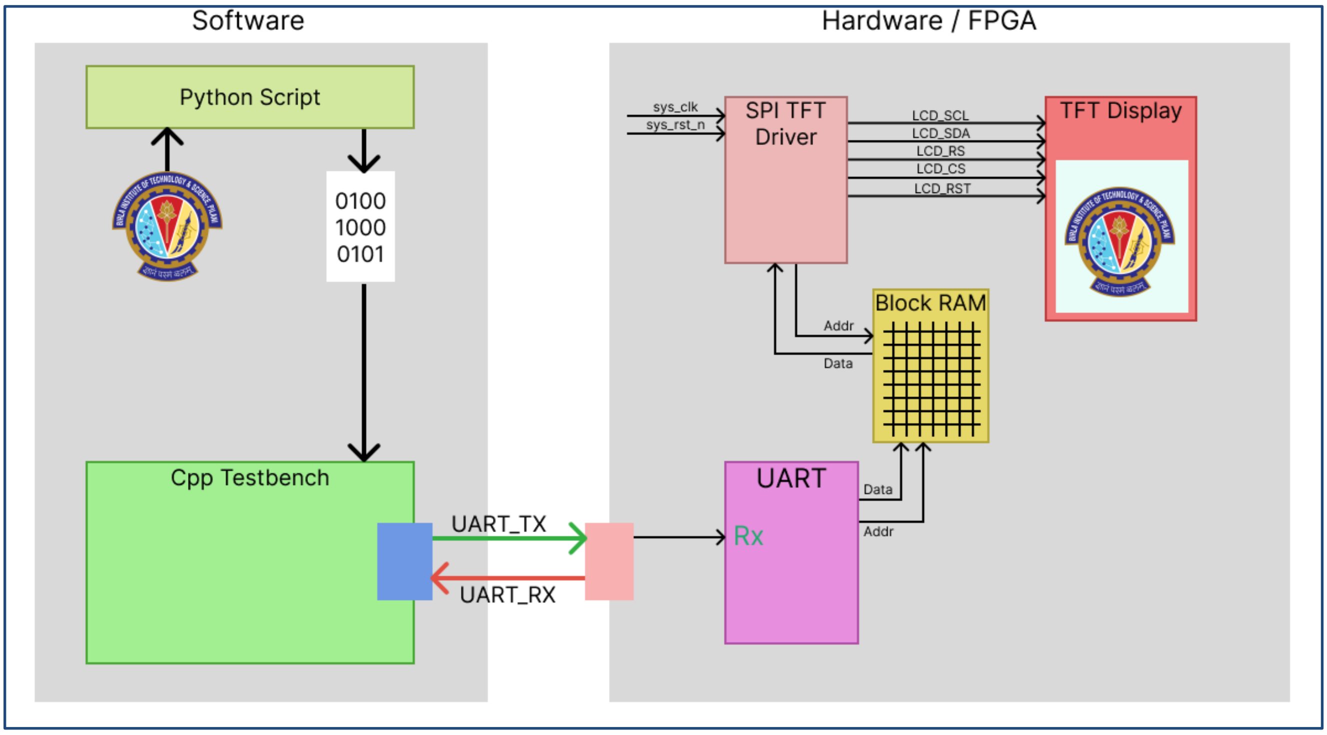

Figure 1: System Architecture Block Diagram

4.2. COE Image Generation

A Python script was developed to automate the conversion of standard images into COE format. The script resizes the image to 128x160 pixels, converts the pixel values to RGB565 format, and generates a COE file that can be loaded into FPGA memory. This file is used to initialize the contents of the FPGA's RAM, which is later processed and displayed on the TFT screen

4.3. UART Communication to FPGA

The C++ testbench establishes a UART connection between the PC and the FPGA. The testbench reads the COE file and sends the pixel data to the FPGA over a serial connection. The UART receiver on the FPGA captures the incoming data and stores it in a RAM instance for further processing.

4.4. FPGA Design for Traffic Handling and Display

The FPGA design includes several key components:

- **UART Receiver**: This module captures incoming data from the UART interface and stores it in RAM.

- **RAM**: A block of memory is allocated to store the COE image data. Once the entire image is received, the data is sent to the SPI controller for display.

- **SPI Controller**: The SPI controller drives the TFT display, sending the image data stored in RAM to the display in the appropriate format.

4.5. Proof of Concept (POC)

For the POC, the FPGA is connected to an SPI TFT display. Traffic received via UART is converted into the SPI format and displayed on the screen. This demonstrates the ability of the system to handle protocol conversion and validate the design in real-world scenarios.

4.6. Live Debugging with ILA

The **Integrated Logic Analyzer (ILA)** is a powerful tool provided by FPGA design suites (such as Xilinx's Vivado) for real-time, internal signal monitoring and debugging within the FPGA. Unlike traditional external logic analyzers, the ILA is embedded within the FPGA design, allowing it to capture and monitor signals directly from the FPGA fabric. This section describes the detailed working process of the ILA, including how to integrate it into an FPGA design, how to select signals for probing, and how the ILA functions in capturing data.

4.6.1. Overview of ILA Functionality

The ILA is part of the suite of embedded debug cores in the FPGA design tool. Once inserted into the FPGA design, it enables users to observe internal signal activity, store data, and trigger events. The ILA allows designers to debug their design while it operates at full speed, making it ideal for capturing real-time events that may not be observable from outside the FPGA.

4.6.2. Steps to Add ILA to the FPGA Design

To use ILA, the following steps are typically followed:

Step 1: Insert ILA Core

- Open the FPGA design project in a tool like Vivado.
- Go to the IP Catalog and search for the ILA Core.
- Instantiate the ILA core into the design. The ILA core acts as a virtual probe that connects to the internal signals (nets and registers) you wish to monitor.

Step 2: Connect ILA to Signals

- The next step is to connect the ILA core to the signals you want to monitor. These signals can be nets (connections between logic blocks), registers, or any other internal logic in the FPGA.
- For example, if you want to debug the UART receiver module, you can attach the ILA to the signals such as uart_rx_data, rx_ready, or baud_rate_clk.

Step 3: Define Probe Widths and Depth

- Depending on the signals you're monitoring, you may need to configure the width of the probes. For example, a 16-bit signal would require a probe width of 16.
- Additionally, the ILA core allows you to set capture depth, which is the number of samples you want to store during each capture session. The depth can be

configured to store a few hundred or thousands of clock cycles, depending on the available resources in the FPGA.

Step 3: Define Probe Widths and Depth

- Depending on the signals you're monitoring, you may need to configure the width of the probes. For example, a 16-bit signal would require a probe width of 16.
- Additionally, the ILA core allows you to set **capture depth**, which is the number of samples you want to store during each capture session. The depth can be configured to store a few hundred or thousands of clock cycles, depending on the available resources in the FPGA.

4.6.3. Selecting Nets and Registers for Probing

When adding an ILA core to a design, selecting the right signals (nets or registers) for probing is crucial. This involves identifying the signals that are critical for debugging the particular functionality of interest.

5. Challenges Faced

5.1. Traffic Generation Complexity:

Designing the software testbench to handle various traffic protocols has been challenging, particularly in ensuring accurate simulation of real-world protocols.

5.2. UART Bandwidth Limitations:

The limited data rate of UART presents challenges in handling high-volume traffic, requiring optimizations in both the testbench and FPGA design.

5.3. Timing and Synchronization:

Managing the timing between UART reception and SPI display updates has been tricky, with timing mismatches causing display errors.

5.4. Debugging Integration:

Implementing and utilizing ILA effectively requires careful planning, especially in setting trigger conditions and capturing the necessary signals.

6. Conclusion

The project aimed to create a system that could transmit an image from a software testbench to an FPGA, convert the image into a format suitable for display on an SPI TFT screen, and perform real-time debugging. The project was successfully implemented in stages, with each component being tested and validated individually.

Key challenges encountered during development included the synchronization of UART communication between the software testbench and FPGA, especially due to independent clock domains for transmission and reception. The initial incorrect configuration of baud rate counters led to missed or incorrectly sampled transactions, which was resolved by adjusting the baud rate to 4800, providing more stable and error-free communication. Debugging this issue using onboard 7-segment displays for UART status monitoring proved to be a critical learning experience, as the limited resources of the Spartan 7 FPGA restricted the use of advanced debugging tools like the Integrated Logic Analyzer (ILA). While the theoretical aspects of live debugging with ILA were explored, they could not be demonstrated practically due to resource constraints.

The Python script to convert images into COE format was a crucial part of the project, enabling the C++ testbench to send the COE data via UART to the FPGA. Once the data was received and stored in the Block RAM, the FPGA was able to control the SPI TFT display, showing the transmitted image. The successful integration of software (Python and C++) with hardware (FPGA, UART, and SPI TFT) highlights the versatility of FPGAs in handling real-time data processing.

In conclusion, the project demonstrated a practical approach to data transmission between software and FPGA, converting and displaying image data in real-time on an SPI TFT screen. Despite the challenges faced, the system proved functional and met the core objectives. Future enhancements could include the implementation of more advanced debugging features, higher-resolution displays, and faster data transmission methods. The project lays a strong foundation for future work in real-time FPGA-based display systems.

7. Appendices

Appendix A: Python Script for Image to COE Conversion

The Python script used to convert images into a COE file format for storage in the Block RAM of the FPGA is provided below. The script ensures the image is resized to the required 128x160 pixel resolution before generating the COE file.

```
from PIL import Image

def image_to_coe(image_path, output_file, width=128, height=160):
    # Load and resize image
    img = Image.open(image_path)
    img = img.resize((width, height))
    img = img.convert('RGB')

    # Open output COE file
    with open(output_file, 'w') as coe_file:
        coe_file.write("memory_initialization_radix=16;\n")
        coe_file.write("memory_initialization_vector=\n")

        for y in range(height):
            for x in range(width):
                r, g, b = img.getpixel((x, y))
                pixel_value = (r >> 3) << 11 | (g >> 2) << 5 | (b >> 3)
                coe_file.write(f'{pixel_value:04X},\n')

        coe_file.write(";")
```

Appendix B: C++ Testbench Code for UART Transmission

The following is the C++ code that implements the testbench for sending the COE image data to the FPGA via UART.

```
#include <iostream>
#include <fstream>
#include <string>
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>

void send_data_via_uart(const std::string &file_path, const std::string &uart_device) {
    std::ifstream file(file_path);
    if (!file.is_open()) {
        std::cerr << "Error opening COE file!" << std::endl;
        return;
    }

    int uart_fd = open(uart_device.c_str(), O_RDWR | O_NOCTTY);
    if (uart_fd < 0) {
        std::cerr << "Error opening UART device!" << std::endl;
        return;
    }

    struct termios options;
    tcgetattr(uart_fd, &options);
    cfsetispeed(&options, B4800);
    cfsetospeed(&options, B4800);
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    tcsetattr(uart_fd, TCSANOW, &options);

    std::string line;
    while (std::getline(file, line)) {
        if (line != ";" && line != "memory_initialization_radix=16;" && line !=
"memory_initialization_vector=") {
            line.erase(line.find(",")); // Remove trailing comma
            unsigned short data = std::stoul(line, nullptr, 16);
            write(uart_fd, &data, sizeof(data));
            usleep(1000); // Delay for stable transmission
        }
    }

    close(uart_fd);
    file.close();
}

int main() {
    std::string image_path = "image.coe";
    std::string uart_device = "/dev/ttyUSB0";  // Update this path as per your system
    send_data_via_uart(image_path, uart_device);
    return 0;
}
```

Appendix C: UART Receiver Design

The UART receiver module on the FPGA reads the incoming serial data from the C++ testbench, converts it to parallel, and stores it in Block RAM for display on the TFT screen. Below is the Verilog code snippet for the UART receiver:

```
module uart_rx (
    input wire clk,
    input wire rst,
    input wire rx,
    output reg [15:0] data_out,
    output reg valid
);
    // Baud rate generator and receiver logic here
    // Data gets sampled and stored in data_out
endmodule
```

Appendix D: Block RAM Configuration

The configuration for Block RAM to store 128x160 pixel data is provided in this appendix.

```
module block_ram (
    input wire clk,
    input wire we,
    input wire [15:0] din,
    input wire [15:0] addr,
    output reg [15:0] dout
);
    reg [15:0] mem [0:20479];  // 128x160 pixels

    always @(posedge clk) begin
        if (we)
            mem[addr] <= din;
        else
            dout <= mem[addr];
    end
endmodule
```

Appendix E: ILA Integration for Debugging

The ILA core is used for live debugging signals in the FPGA. Although it is not implemented in this project due to hardware constraints, the steps for adding ILA cores are described below:

1. In Vivado, select "IP Integrator" and add the **ILA** core from the IP catalog.
2. Connect the **probes** (nets or registers) you want to monitor to the ILA input ports.
3. Configure the ILA core parameters such as depth and trigger conditions.
4. Generate the bitstream and program the FPGA.
5. Open the **Hardware Manager** in Vivado and use it to view real-time data from the ILA probes during FPGA execution.

This section provides the theoretical foundation for how the ILA would have been used to debug UART transmissions and Block RAM updates in real time had the hardware supported it.

8. References

- Xilinx. (n.d.). *Vivado Design Suite User Guide: Embedded Processor Hardware Design*. Available at: https://www.xilinx.com
- Quartus, A. (2023). *FPGA Design and Implementation with Verilog*. Available at: https://www.intel.com
- Harris, S. & Harris, D. (2012). *Digital Design and Computer Architecture*. Elsevier, 2nd Edition.
- Xilinx. (n.d.). *Spartan-7 FPGA Family Data Sheet*. Available at: https://www.xilinx.com/products/silicon-devices/fpga/spartan-7.html
- "UART Communication Protocol", Texas Instruments. Available at: https://www.ti.com
- Pallavi, P. (2023). *Designing TFT Displays for Embedded Systems*. Electronics Weekly, Issue 12.
- Python Documentation. (2023). *Using Python for Image Conversion*. Available at: https://docs.python.org
- McGregor, J. (2019). *Real-time Data Transfer Using UART in FPGA Systems*. IEEE Transactions on Circuits and Systems, Vol. 65, No. 4, pp. 1123-1135.

Checklist of Items for the Final Dissertation / Project / Project Work Report

This checklist is to be attached as the last page of the final report.

This checklist is to be duly completed, verified and signed by the student.

	Is the final report neatly formatted with all the elements required for a technical Report?	**Yes**
2.	Is the Cover page in proper format as given in Annexure A?	**Yes**
3.	Is the Title page (Inner cover page) in proper format?	**Yes**
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	**Yes** **Yes**
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	**Yes** **Yes**
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	**Yes**
7.	Have you included the List of abbreviations / Acronyms?	**Yes**
8.	Does the Report contain a summary of the literature survey?	Yes
9.	Does the Table of Contents include page numbers? i.Are the Pages numbered properly? (Ch. 1 should start on Page # 1) i.Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) i.Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) /.Are the Captions for the Figures and Tables proper? /.Are the Appendices numbered properly? Are their titles appropriate	**Yes** **Yes** **Yes** **Yes** **Yes** **Yes**
10	Is the conclusion of the Report based on discussion of the work?	**Yes**
11	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	**Yes** **Yes** **Yes**
12	Is the report format and content according to the guidelines? The report should not be a mere printout of a PowerPoint Presentation, or a user manual. Source code of software need not be included in the report.	**Yes**

Declaration by Student:
I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Noida **Signature of the Student:** Sachin Singh

Date: 18th November 2024 **Name:** Sachin Singh

ID No.: 2022HT01625