THE GEORGE WASHINGTON UNIVERSITY

School of Engineering and Applied Science [SEAS]

FALL 2024 BATCH

Course: Operating System Engineering ECE 6045

Professor: Dr. Howie Huang (Ph.D)

Project Titled:

OS - Level Virtualization in FPGA's

REPORT BY:

RAVI L BELLUBBI (G32127843)

CONTENTS

A. Introduction

In recent years, Field-Programmable Gate Arrays (FPGAs) have emerged as versatile hardware platforms capable of supporting diverse high-performance applications across fields such as data centres, machine learning, and telecommunications. The unique reconfigurability of FPGAs enables them to meet the high-performance, low-power requirements of these domains, yet it also introduces challenges when multiple applications or users must share the same physical FPGA hardware. Virtualization on FPGA systems address this by abstracting hardware resources, allowing multiple applications or users to share a single physical FPGA in a manner similar to OS-level virtualization on CPUs. This OS-level virtualization enables efficient, isolated access to FPGA resources, improving utilization, flexibility, and scalability while maintaining the benefits of custom hardware acceleration.

However, virtualizing FPGAs presents unique technical challenges, including managing the stringent timing constraints inherent to FPGA circuits, balancing resource allocation, and ensuring the security and isolation of virtual instances. These challenges have driven research in virtualization techniques specific to FPGAs, each focusing on different virtualization goals. For instance, Beyer et al. (2020) discuss *full virtualization*, which creates complete virtual instances of FPGA hardware at the cost of increased resource usage, while Schaeffer et al. (2021) explore *para-virtualization*, which modifies applications to run within the constraints of available FPGA resources, enhancing performance by reducing virtualization overhead. Another approach, described by Li et al. (2022), involves *hardware-assisted virtualization*, which integrates custom FPGA logic to support virtualization with minimal interference in resource performance, optimizing both flexibility and resource utilization.

These approaches serve to address some of the fundamental issues in FPGA virtualization, yet each method entails trade-offs in terms of performance, flexibility, and resource overhead. Beyer et al. (2020) and Schaeffer et al. (2021) further emphasize the importance of efficient resource partitioning strategies to ensure that critical FPGA resources like lookup tables (LUTs), memory blocks, and I/O channels are dynamically managed, meeting the needs of each virtualized instance without introducing significant performance degradation. More recent advancements focus on security, as multi-tenant FPGAs are vulnerable to side-channel attacks that can compromise data confidentiality between virtual machines. Research by Huang et al. (2023) suggests robust security protocols for FPGA VMs, including data encryption techniques and isolation frameworks to protect data integrity and enhance multi-user FPGA environments. The compiler generates virtual addresses (VAs) while the OS and the hardware map them onto physical addresses (PAs) as shown in the figure 1.

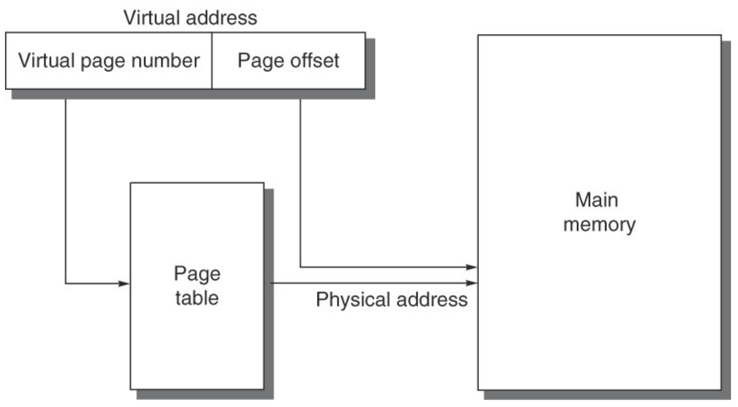


Fig 1. The mapping of a virtual address to a physical address via a page table

FPGA virtualization can be classified based on various evolving criteria, such as techniques, use cases, or execution models. To provide a stable framework adaptable to future trends, we propose a classification based on abstraction levels in standard computational systems. This classification divides FPGA virtualization into three levels: Resource level, Node level, and Multi-node level.

1. **Resource Level**: This level focuses on individual FPGA resources, categorized as reconfigurable or non-reconfigurable. It includes architecture virtualization through overlays for abstraction and I/O virtualization for transparent resource sharing in multi-tenant systems.

2. **Node Level**: Representing a single FPGA, this level addresses infrastructure and resource management for concurrent applications. Key techniques include virtual machine monitor (VMM) support, runtime systems, and FPGA OS or Hypervisor-vFPGA approaches that facilitate multiple user accelerators.

3. **Multi-node Level**: This level encompasses clusters of multiple FPGAs working together to accelerate complex workloads. It includes techniques for interconnecting FPGAs, such as Leap, MapReduce, and Catapult, which enhance the processing of large-scale tasks.

This classification framework helps organize current and emerging FPGA virtualization efforts across different computational scales, facilitating a wide range of applications.

The proposed project seeks to build upon these foundational principles by designing an OS-level virtual machine monitor (VMM) for FPGAs, aimed at implementing resource-sharing virtualization techniques while evaluating the performance, power, and area overhead introduced by the virtualization layer. Leveraging platforms like the Xilinx Zynq Z-7010 AP SoC, this study will focus on creating a scalable VMM capable of dynamically managing FPGA resources for concurrent applications. By examining existing virtualization methods and addressing limitations specific to FPGA architecture, this project aims to contribute to the body of research on FPGA-based virtualization, providing a pathway toward more flexible, efficient FPGA utilization in high-performance computing environments.

B. Problem Statement

a. As FPGAs are increasingly used in high-performance environments, there is a need for effective virtualization to support AI - ML advancement. Convolution CPU virtualization methods don't directly apply to FPGAs due to their reconfigurable nature.
b. This project aims to develop a Virtual Machine Monitor (VMM) that enables multiple VMs to share FPGA resources with minimal overhead, ensuring efficient resource management, security, and scalability. The goal is to create a virtualization layer that supports both simple and complex FPGA applications, addressing the unique challenges of FPGA systems.

C. Objectives

- To design an OS level virtualization on an FPGA-based VLSI system.
- Implement techniques to share resources among multiple virtual machines (VMs).
- Evaluate performance impacts of virtualization.

D. Applications

The main reason of FPGA virtualization are similar to the core objectives that resulted in the development of virtualization used in traditional CPU/software systems.

- Multi-tenancy (supporting multiple users on one FPGA)
- Resource management (abstraction layers and task scheduling)
- Flexibility (supporting diverse accelerators)
- Isolation (providing users exclusive FPGA resource access)
- Scalability (low-overhead support for multiple FPGAs or users)
- Performance (minimizing virtualization impact on FPGA resource use)
- Security (protecting user data and infrastructure)
- Resilience (maintaining service despite failures)
- Enhancing programmer productivity (reducing design deployment complexity and time)

E. Tools used

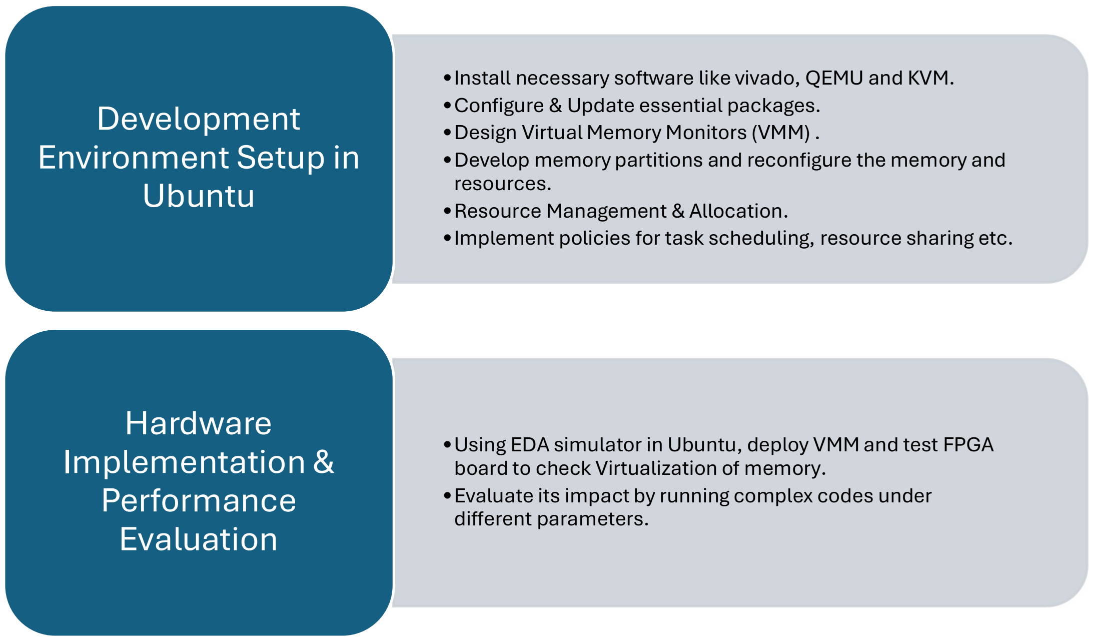
i. FPGA development board - Xilinx Zynq Z-7010 AP SoC FPGA
ii. Linux OS - Ubuntu
iii. EDA Simulators - Xilinx Vivado Simulator

F. Methodology

In Phase 1, the software implementation starts by setting up the development environment, which includes installing Vivado, QEMU, and KVM on Ubuntu and configuring ARM Cortex cross-compilation tools for the Zynq platform. Next, a Virtual Machine Monitor (VMM) will be designed to partition and allocate FPGA resources securely, creating an abstraction layer for virtualizing memory, I/O, and reconfigurable fabric. The VMM will then be deployed in a QEMU/KVM environment to simulate VM behaviour, allowing for debugging and performance evaluation of multi-tenancy and task scheduling. Resource management policies for task scheduling, resource sharing, and I/O virtualization will also be implemented, with metrics developed to measure resource usage, latency, and response time during simulations.

Phase 2 focuses on hardware implementation, where the VMM design will be transferred to the Xilinx Zynq Z-7010 FPGA using Vivado, ensuring alignment with parameters tested in the software phase. The FPGA fabric will be configured to support resource partitioning and VM isolation. A series of hardware-level tests will be conducted to measure performance, power, and area overheads introduced by virtualization, alongside evaluating the VMM's effectiveness in managing multiple VMs and its impact on FPGA performance. Based on the results, VMM parameters and resource allocation strategies will be optimized and refined to enhance overall performance.

Finally, performance evaluation and analysis will involve collecting metrics to measure the overhead, power consumption, and area impact introduced by the VMM. The results will be compared to baseline FPGA performance to assess the efficiency of virtualization, with a focus on evaluating the effectiveness of multi-tenancy, scalability, and security of isolation within the system.


Development Environment Setup in Ubuntu

- Install necessary software like vivado, QEMU and KVM.
- Configure & Update essential packages.
- Design Virtual Memory Monitors (VMM) .
- Develop memory partitions and reconfigure the memory and resources.
- Resource Management & Allocation.
- Implement policies for task scheduling, resource sharing etc.

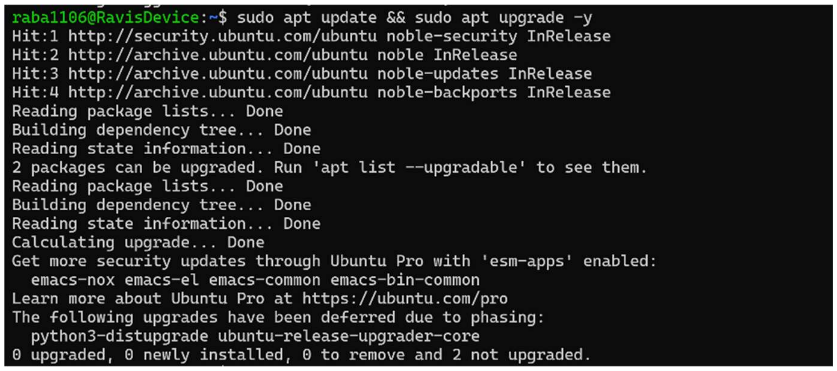
Hardware Implementation & Performance Evaluation

- Using EDA simulator in Ubuntu, deploy VMM and test FPGA board to check Virtualization of memory.
- Evaluate its impact by running complex codes under different parameters.

G. Results & Analysis

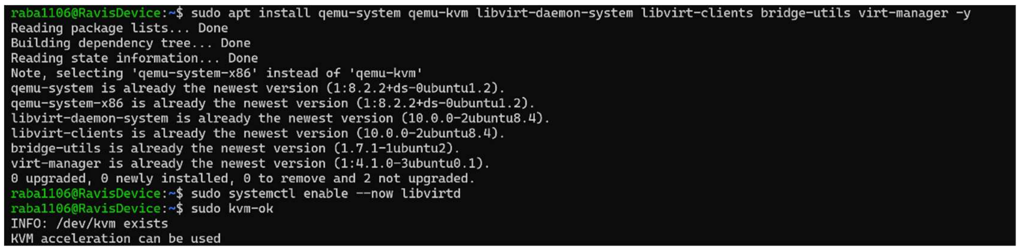
Phase 1, we setup ubuntu environment suitable for running FPGA simulator, install vmm packages like kvm, qemu and arm processor packages.

Configure & Update essential packages.



```
raba1106@RavisDevice:~$ sudo apt update && sudo apt upgrade -y
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
Get more security updates through Ubuntu Pro with 'esm-apps' enabled:
  emacs-nox emacs-el emacs-common emacs-bin-common
Learn more about Ubuntu Pro at https://ubuntu.com/pro
The following upgrades have been deferred due to phasing:
  python3-distupgrade ubuntu-release-upgrader-core
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

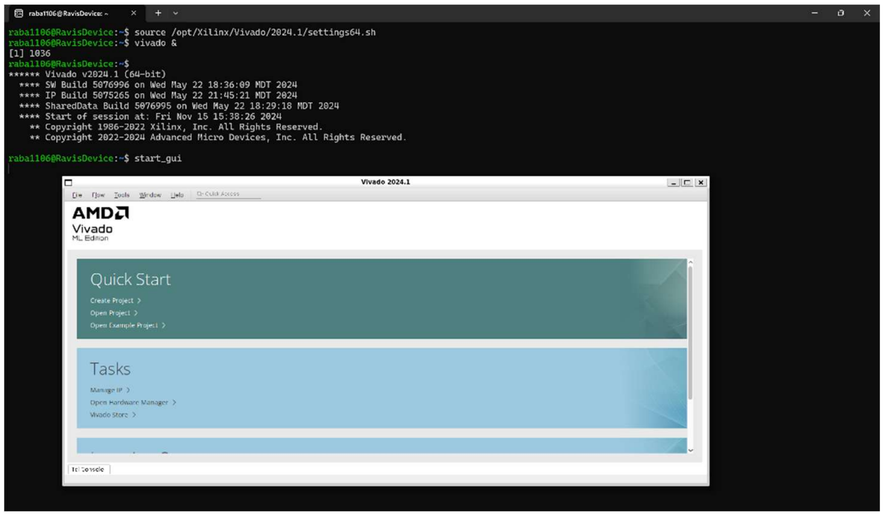
Installed necessary software like vivado, QEMU and KVM.



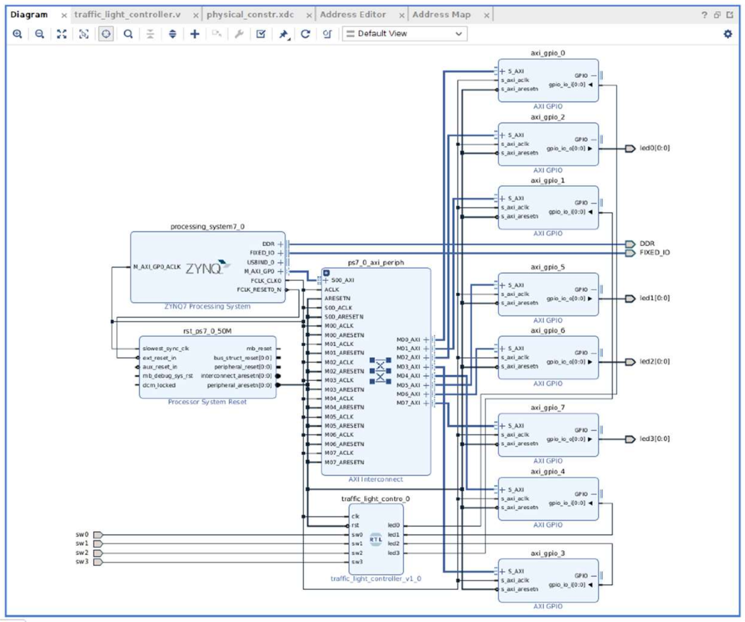
```
raba1106@RavisDevice:~$ sudo apt install qemu-system qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virt-manager -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'qemu-system-x86' instead of 'qemu-kvm'
qemu-system is already the newest version (1:8.2.2+ds-0ubuntu1.2).
qemu-system-x86 is already the newest version (1:8.2.2+ds-0ubuntu1.2).
libvirt-daemon-system is already the newest version (10.0.0-2ubuntu8.4).
libvirt-clients is already the newest version (10.0.0-2ubuntu8.4).
bridge-utils is already the newest version (1.7.1-1ubuntu2).
virt-manager is already the newest version (1:4.1.0-3ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
raba1106@RavisDevice:~$ sudo systemctl enable --now libvirtd
raba1106@RavisDevice:~$ sudo kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

x86/Linux OS

Installed Vivado EDA simulator in X86/Windows to simulate traffic light control module to evaluate the necessary parameters for comparison study.



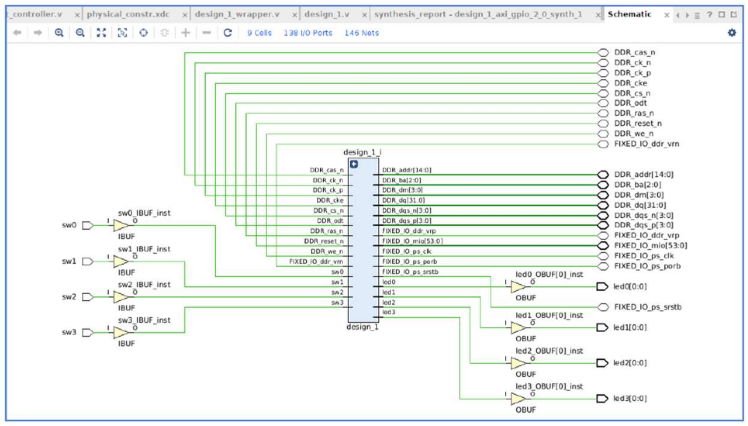
```
raba1106@RavisDevice:~$ source /opt/Xilinx/Vivado/2024.1/settings64.sh
raba1106@RavisDevice:~$ vivado &
[1] 1036
raba1106@RavisDevice:~$
****** Vivado v2024.1 (64-bit)
  **** SW Build 5076996 on Wed May 22 18:36:09 MDT 2024
  **** IP Build 5075265 on Wed May 22 21:45:21 MDT 2024
  **** SharedData Build 5076995 on Wed May 22 18:29:18 MDT 2024
  **** Start of session at: Fri Nov 15 15:38:26 2024
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
    ** Copyright 2022-2024 Advanced Micro Devices, Inc. All Rights Reserved.

raba1106@RavisDevice:~$ start_gui
```



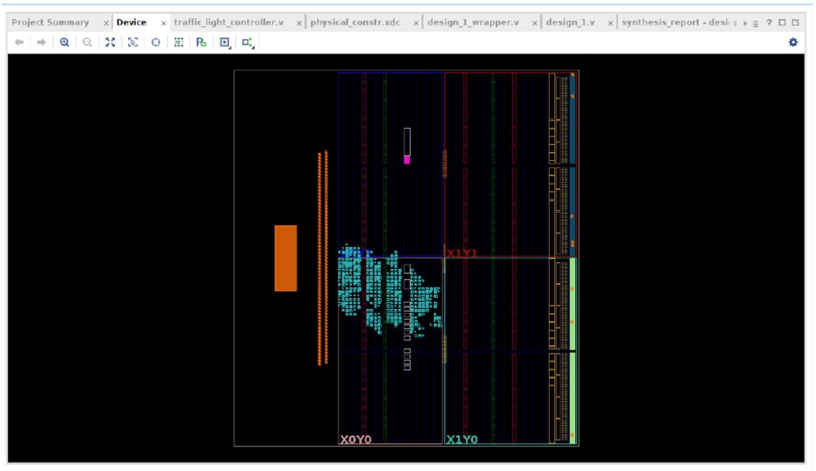
This is the Finite state machine design of traffic light controller where the design will have 4 input switches and 4 output lights working as a standard traffic signal.

Module Declaration: The code defines a Verilog module named Traffic_Light_Controller with inputs clk, rst, sw0, sw1, sw2, and sw3, and outputs led0, led1, led2, and led3.
State Machine: The module uses a finite state machine (FSM) to control the traffic lights. The FSM has 6 states (3'b000 to 3'b101) and uses a counter (count) to stay in each state for a specific number of clock cycles.
LED Outputs: The LED outputs are determined by the current state of the FSM. Each state corresponds to a specific combination of LED outputs.

Switch Overrides: The code also includes logic to override the LED outputs based on the state of the switches (sw0 to sw3). If a switch is asserted, the corresponding LED will be turned on, regardless of the current state of the FSM.

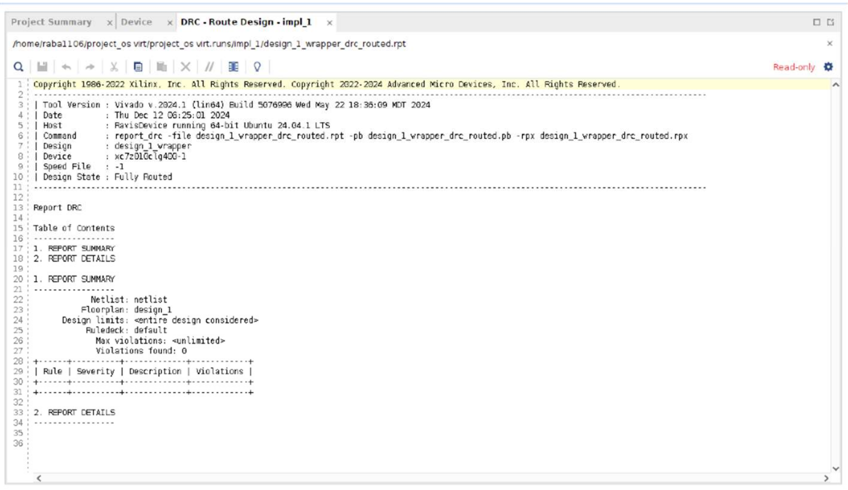


This is the RTL version of the traffic light controller design which is implemented on Zynq 7010 FPGA board.



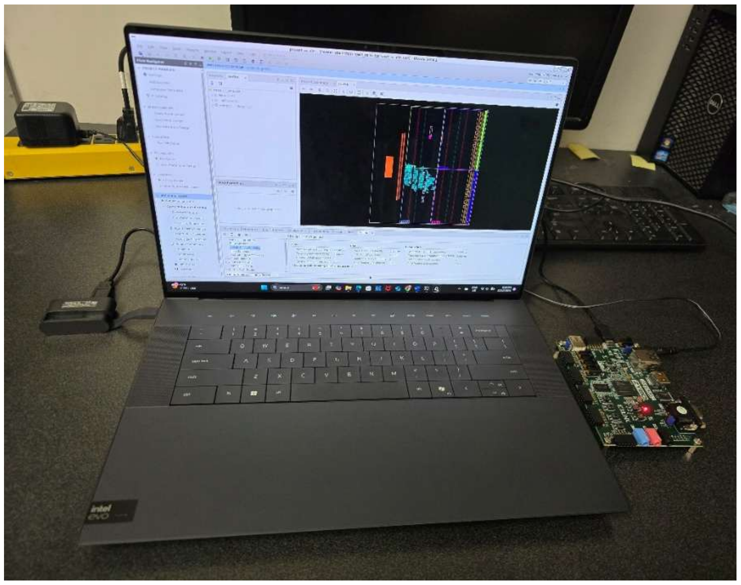
This is the Layout design after Post synthesis of the FPGA design.

This is the Design Rule Check (DRC) report to confirm that my design has no faults and errors and is perfect for evaluation.



/home/raba1106/project_os virt/project_os virt.runs/impl_1/design_1_wrapper_drc_routed.rpt

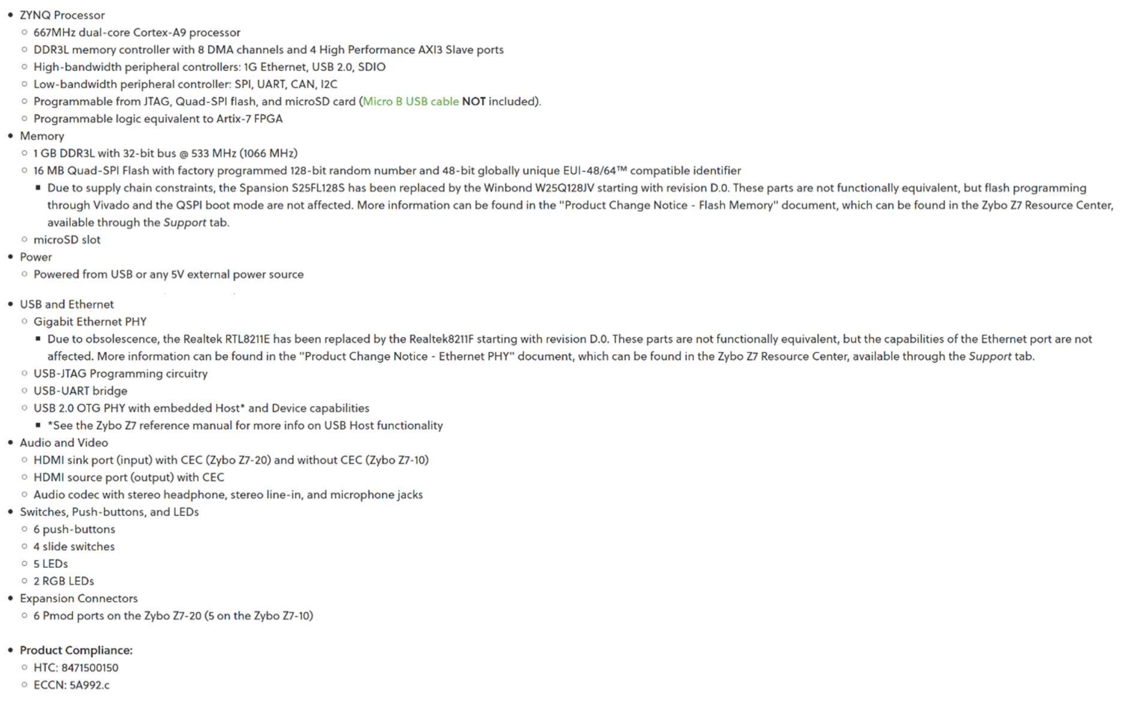
```
Copyright 1986-2022 Xilinx, Inc. All Rights Reserved. Copyright 2022-2024 Advanced Micro Devices, Inc. All Rights Reserved.
---------------------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2024.1 (lin64) Build 5076996 Wed May 22 18:36:09 MDT 2024
| Date         : Thu Dec 12 06:25:01 2024
| Host         : RavisDevice running 64-bit Ubuntu 24.04.1 LTS
| Command      : report_drc -file design_1_wrapper_drc_routed.rpt -pb design_1_wrapper_drc_routed.pb -rpx design_1_wrapper_drc_routed.rpx
| Design       : design_1_wrapper
| Device       : xc7z010clg400-1
| Speed File   : -1
| Design State : Fully Routed
---------------------------------------------------------------------------------------------------------------------------

Report DRC

Table of Contents
-----------------
1. REPORT SUMMARY
2. REPORT DETAILS

1. REPORT SUMMARY
-----------------
            Netlist: netlist
          Floorplan: design_1
      Design limits: <entire design considered>
           Ruledeck: default
             Max violations: <unlimited>
             Violations found: 0
+------+----------+-------------+------------+
| Rule | Severity | Description | Violations |
+------+----------+-------------+------------+
+------+----------+-------------+------------+

2. REPORT DETAILS
-----------------


```



This is the setup. The X86 machine and virtual ubuntu machine from the laptop is connected to hardware i.e., Zynq z7010 FPGA board.

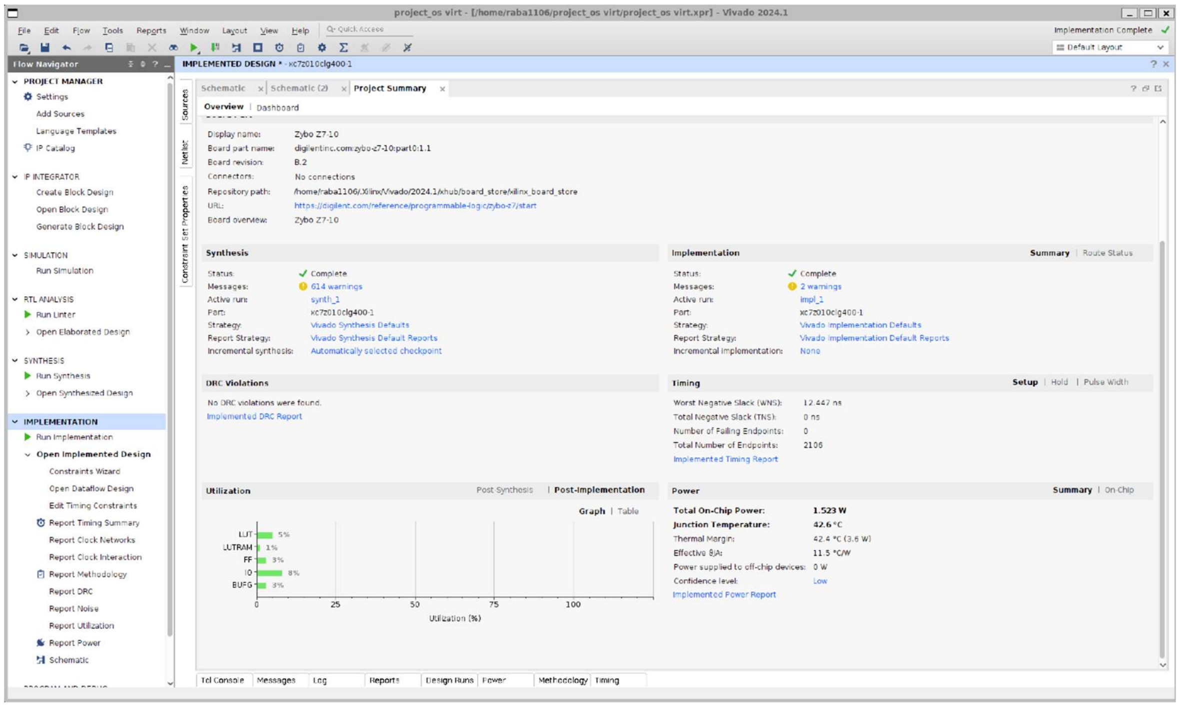
Upon designing the traffic light controller module, we feed the design wrapper into the FPGA machine.

Details about FPGA board is as below.



- ZYNQ Processor
  - 667MHz dual-core Cortex-A9 processor
  - DDR3L memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
  - High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
  - Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
  - Programmable from JTAG, Quad-SPI flash, and microSD card (Micro B USB cable **NOT** included).
  - Programmable logic equivalent to Artix-7 FPGA
- Memory
  - 1 GB DDR3L with 32-bit bus @ 533 MHz (1066 MHz)
  - 16 MB Quad-SPI Flash with factory programmed 128-bit random number and 48-bit globally unique EUI-48/64™ compatible identifier
    - Due to supply chain constraints, the Spansion S25FL128S has been replaced by the Winbond W25Q128JV starting with revision D.0. These parts are not functionally equivalent, but flash programming through Vivado and the QSPI boot mode are not affected. More information can be found in the "Product Change Notice - Flash Memory" document, which can be found in the Zybo Z7 Resource Center, available through the *Support* tab.
  - microSD slot
- Power
  - Powered from USB or any 5V external power source
- USB and Ethernet
  - Gigabit Ethernet PHY
    - Due to obsolescence, the Realtek RTL8211E has been replaced by the Realtek8211F starting with revision D.0. These parts are not functionally equivalent, but the capabilities of the Ethernet port are not affected. More information can be found in the "Product Change Notice - Ethernet PHY" document, which can be found in the Zybo Z7 Resource Center, available through the *Support* tab.
  - USB-JTAG Programming circuitry
  - USB-UART bridge
  - USB 2.0 OTG PHY with embedded Host* and Device capabilities
    - *See the Zybo Z7 reference manual for more info on USB Host functionality
- Audio and Video
  - HDMI sink port (input) with CEC (Zybo Z7-20) and without CEC (Zybo Z7-10)
  - HDMI source port (output) with CEC
  - Audio codec with stereo headphone, stereo line-in, and microphone jacks
- Switches, Push-buttons, and LEDs
  - 6 push-buttons
  - 4 slide switches
  - 5 LEDs
  - 2 RGB LEDs
- Expansion Connectors
  - 6 Pmod ports on the Zybo Z7-20 (5 on the Zybo Z7-10)
- **Product Compliance:**
  - HTC: 8471500150
  - ECCN: 5A992.c

The Timing, Area and FPGA Resource Utilization Reports of FPGA Design implemented in X86 Operating System.



project_os virt - [/home/raba1106/project_os virt/project_os virt.xpr] - Vivado 2024.1

IMPLEMENTED DESIGN * - xc7z010clg400-1

Project Summary — Overview

Display name: Zybo Z7-10
Board part name: digilentinc.com:zybo-z7-10:part0:1.1
Board revision: B.2
Connectors: No connections
Repository path: /home/raba1106/.Xilinx/Vivado/2024.1/xhub/board_store/xilinx_board_store
URL: https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview: Zybo Z7-10

Synthesis

Status: ✓ Complete
Messages: 614 warnings
Active run: synth_1
Part: xc7z010clg400-1
Strategy: Vivado Synthesis Defaults
Report Strategy: Vivado Synthesis Default Reports
Incremental synthesis: Automatically selected checkpoint

Implementation

Status: ✓ Complete
Messages: 2 warnings
Active run: impl_1
Part: xc7z010clg400-1
Strategy: Vivado Implementation Defaults
Report Strategy: Vivado Implementation Default Reports
Incremental implementation: None

DRC Violations

No DRC violations were found.
Implemented DRC Report

Timing

Worst Negative Slack (WNS): 12.447 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 2106
Implemented Timing Report

Utilization (Post-Implementation)

LUT	5%
LUTRAM	1%
FF	3%
IO	8%
BUFG	3%

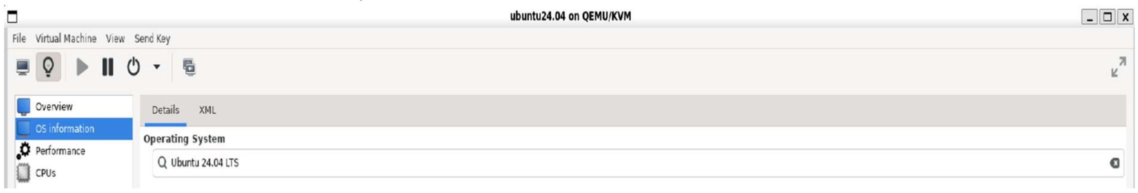
Power

Total On-Chip Power: 1.523 W
Junction Temperature: 42.6 °C
Thermal Margin: 42.4 °C (3.6 W)
Effective θJA: 11.5 °C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low
Implemented Power Report

The details and explanation of the results are discussed in performance evaluation section. The same traffic light controller design is now implemented in virtual operating system for evaluation.

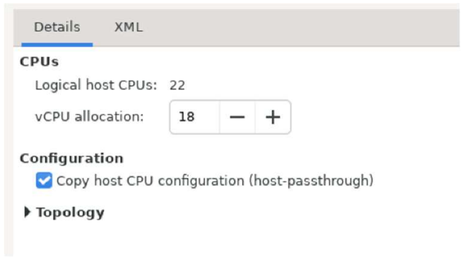
Virtual Linux OS

To design a virtual environment, I am using QEMU/KVM virtual machine.
Inside this virtual machine, I have installed Ubuntu operating system and alloted 18 cpu cores out of 22 cores, 4GB out of 16GB memory available.



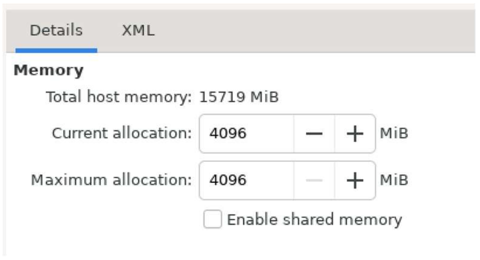
ubuntu24.04 on QEMU/KVM

Operating System: Ubuntu 24.04 LTS



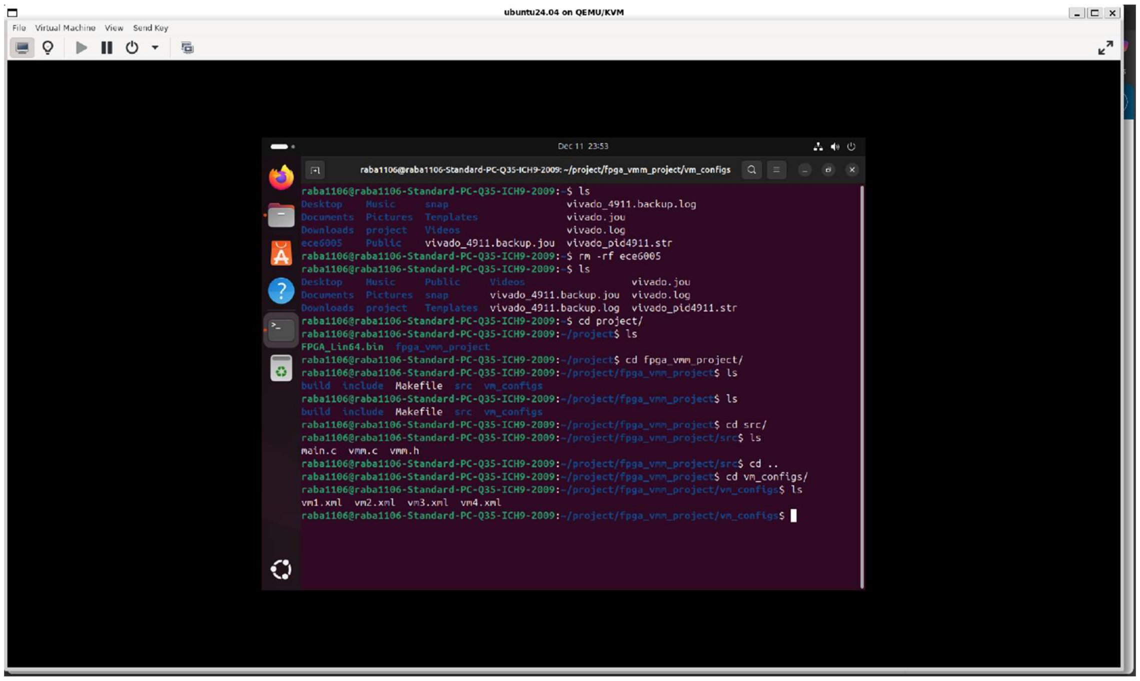
CPUs
Logical host CPUs: 22
vCPU allocation: 18

Configuration
☑ Copy host CPU configuration (host-passthrough)



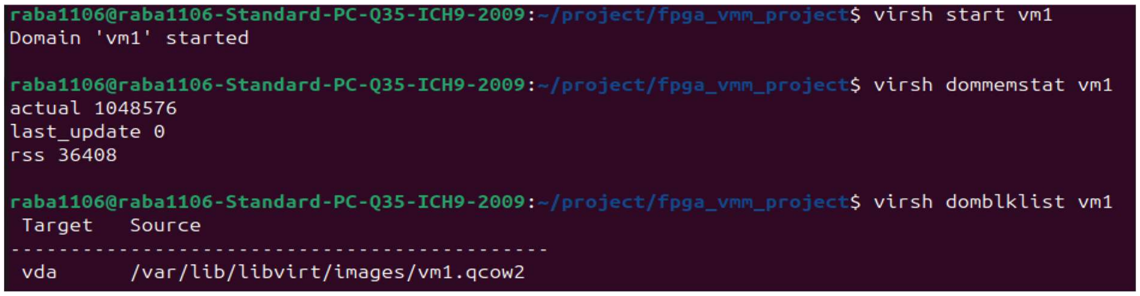
Memory
Total host memory: 15719 MiB
Current allocation: 4096 MiB
Maximum allocation: 4096 MiB
☐ Enable shared memory

To divide the FPGA resources, I have created VMMs to manage resource allocation, scheduling and other virtualization tasks in the background while vivado simulator runs. This will ensure that all the task will be performed efficiently.

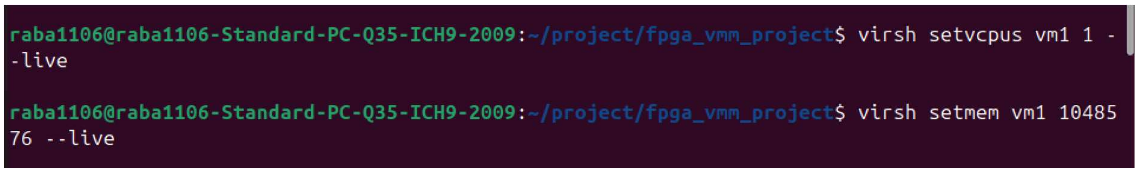

ubuntu24.04 on QEMU/KVM
File Virtual Machine View Send Key
Dec 11 23:53
raba1106@raba1106-Standard-PC-Q35-ICH9-2009: ~/project/fpga_vmm_project/vm_configs

```
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~$ ls
Desktop    Music     snap                  vivado_4911.backup.log
Documents  Pictures  Templates             vivado.jou
Downloads  project   Videos                vivado.log
ece6005    Public    vivado_4911.backup.jou  vivado_pid4911.str
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~$ rm -rf ece6005
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~$ ls
Desktop    Music     Public     Videos                  vivado.jou
Documents  Pictures  snap       vivado_4911.backup.jou  vivado.log
Downloads  project   Templates  vivado_4911.backup.log  vivado_pid4911.str
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~$ cd project/
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project$ ls
FPGA_Lin64.bin  fpga_vmm_project
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project$ cd fpga_vmm_project/
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ ls
build  include  Makefile  src  vm_configs
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ ls
build  include  Makefile  src  vm_configs
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ cd src/
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project/src$ ls
main.c  vmm.c  vmm.h
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project/src$ cd ..
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ cd vm_configs/
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project/vm_configs$ ls
vm1.xml  vm2.xml  vm3.xml  vm4.xml
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project/vm_configs$
```

- Defined the VMs
- Start the VMs operation
- Verify Resource Allocation: Memory allocation, I/O allocation
- Dynamic Resource Allocation: CPU hotplug, Memory hotplug
- Scheduling: Round-Robin



```
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ virsh start vm1
Domain 'vm1' started

raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ virsh dommemstat vm1
actual 1048576
last_update 0
rss 36408

raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ virsh domblklist vm1
 Target   Source
---------------------------------------------
 vda      /var/lib/libvirt/images/vm1.qcow2
```

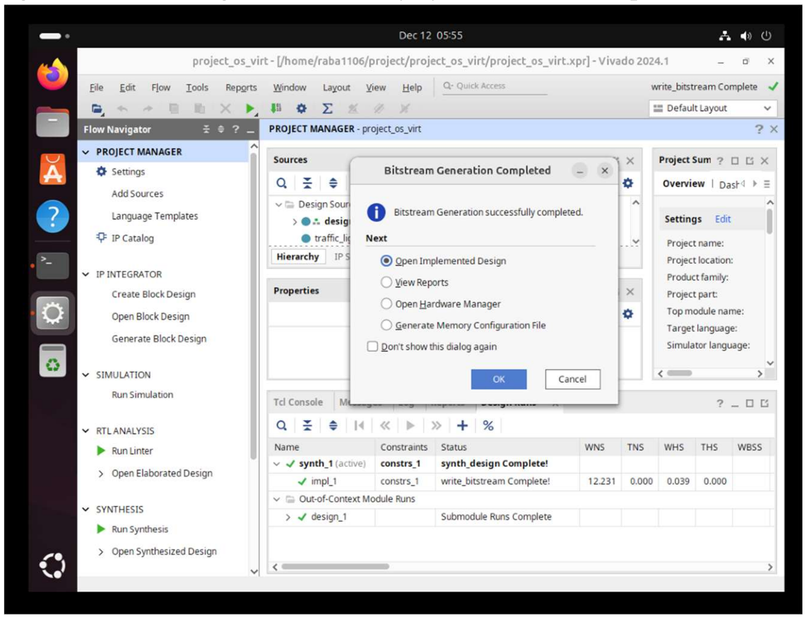


```
raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ virsh setvcpus vm1 1 -
-live

raba1106@raba1106-Standard-PC-Q35-ICH9-2009:~/project/fpga_vmm_project$ virsh setmem vm1 10485
76 --live
```

VM1 = 1GB size
VM2 = 2GB size
VM3 = 3GB size
VM4 = 4GB size

I will attach the files with this report

The traffic light controller design is successfully synthesized and implemented in FPGA board.



Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

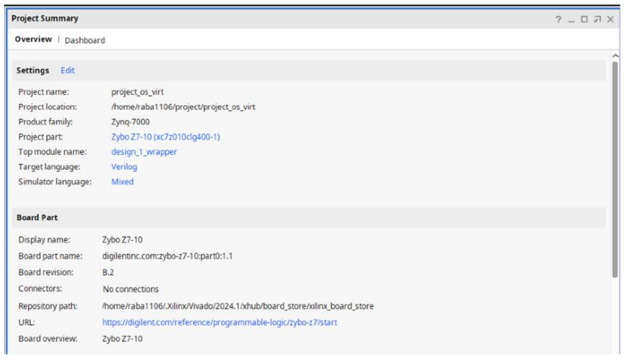
[✓ Open Implemented Design]
View Reports
Open Hardware Manager
Generate Memory Configuration File

☐ Don't show this dialog again

OK   Cancel

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS
✓ synth_1 (active)	constrs_1	synth_design Complete!					
✓ impl_1	constrs_1	write_bitstream Complete!	12.231	0.000	0.039	0.000	
Out-of-Context Module Runs							
✓ design_1		Submodule Runs Complete					

The Timing, Area and FPGA Resource Utilization Reports of FPGA Design implemented in X86 Operating System.



Project Summary

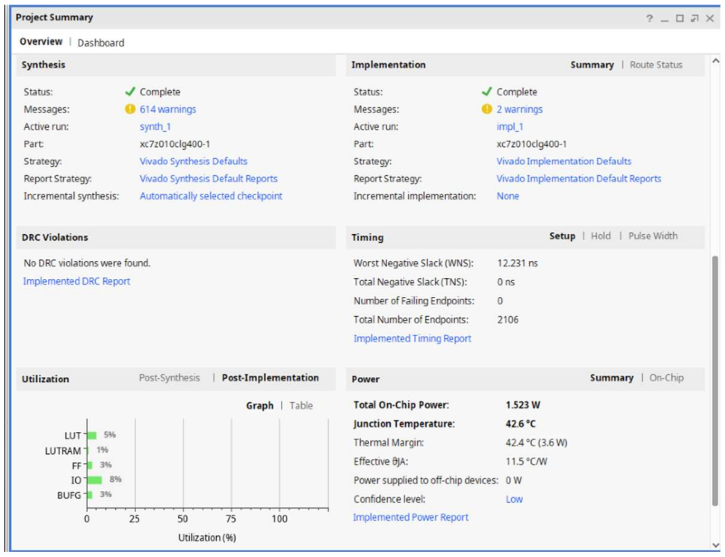
Overview | Dashboard

Settings Edit

Project name:	project_os_virt
Project location:	/home/raba1106/project/project_os_virt
Product family:	Zynq-7000
Project part:	Zybo Z7-10 (xc7z010clg400-1)
Top module name:	design_1_wrapper
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	Zybo Z7-10
Board part name:	digilentinc.com:zybo-z7-10:part0:1.1
Board revision:	B.2
Connectors:	No connections
Repository path:	/home/raba1106/.Xilinx/Vivado/2024.1/xhub/board_store/xilinx_board_store
URL:	https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview:	Zybo Z7-10



Project Summary

Overview | Dashboard

Synthesis

Status:	✓ Complete
Messages:	614 warnings
Active run:	synth_1
Part:	xc7z010clg400-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	Automatically selected checkpoint

Implementation   Summary | Route Status

Status:	✓ Complete
Messages:	2 warnings
Active run:	impl_1
Part:	xc7z010clg400-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

No DRC violations were found.

Implemented DRC Report

Timing   Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	12.231 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2106

Implemented Timing Report

Power   Summary | On-Chip

Total On-Chip Power:	**1.523 W**
Junction Temperature:	**42.6 °C**
Thermal Margin:	42.4 °C (3.6 W)
Effective θJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

H. Performance Evalution

OS	CPU	Memory	Timing Report	Power Report	Synthesis Report
X86	22 Cores	16GB	WNS = 12.447ns	Commercial = 1.52w Industrial = 1.62w	Memory peak = 2442.891mB Free physical = 1344mB Free virtual = 5301mB CPU duration = 35s
Virtual OS	18 Cores	4GB	WNS = 12.231ns	Commercial = 1.523w Industrial = 1.62w	Memory peak = 2351.082mB Free physical = 144mB Free virtual = 4085mB CPU duration = 34s

- **CPU and Memory**: The X86 system, with 22 cores and 16GB of memory, outperforms the Virtual OS, which has 18 cores and 4GB of memory.
- **Timing Report**: The Virtual OS shows a slightly better Worst Negative Slack (WNS) of 12.231ns compared to the X86 system's 12.447ns, indicating a marginal improvement in timing performance.
- **Power Consumption**: Both systems exhibit similar power consumption, with the Virtual OS consuming 1.523W in commercial settings and 1.62W in industrial settings, closely matching the X86 system.
- **Memory Usage**: The X86 system has a higher memory peak and more free physical and virtual memory compared to the Virtual OS, which indicates better memory management in the X86 system.
- **CPU Duration**: The Virtual OS completes tasks slightly faster, with a CPU duration of 34 seconds compared to the X86 system's 35 seconds.

I. Conclusion

The project on OS-level virtualization of FPGA demonstrates a significant advancement in the efficient management and utilization of FPGA resources. By enabling multiple applications to share FPGA resources without interference, this approach enhances flexibility and resource allocation. The virtualization technique simplifies the development process by providing a standardized interface for FPGA access, making it more accessible for various applications.

Future work on OS-level virtualization of FPGA will focus on several key areas to enhance its performance and usability. This includes optimizing timing performance and resource allocation to reduce Worst Negative Slack (WNS) and improve efficiency. Scalability will be addressed by extending support to larger and more complex FPGA systems, as well as multi-FPGA environments. Power efficiency will be improved through advanced power management and adaptive power scaling techniques. Security enhancements will ensure reliable operation and protection against threats, while user interface improvements and automation tools will simplify management. Comprehensive benchmarking and testing with real-world applications will validate the effectiveness of these enhancements.

J. References

1. S. Shreejith, M. D. Santambrogio, D. Sciuto, and F. Bruschi, "A Survey on FPGA Virtualization," ACM Computing Surveys (CSUR), vol. 51, no. 6, pp. 1-39, Dec. 2018. doi: [10.1145/3230634](https://doi.org/10.1145/3230634)

2. K. Vipin, F. A. de Lima Kastensmidt, and S. A. Fahmy, "A Hypervisor for Shared-Memory FPGA Platforms," IEEE Transactions on Computers, vol. 65, no. 8, pp. 2258-2269, Aug. 2016. doi: [10.1109/TC.2015.2511175](https://doi.org/10.1109/TC.2015.2511175)

3. F. Chen et al., "Enabling FPGAs in the Cloud," in Proceedings of the 11th ACM Conference on Computing Frontiers, ser. CF '14, 2014

4. W. Wang, "pvFPGA: Accessing an FPGA-based Hardware Accelerator in a Paravirtualized Environment," in CODES+ ISSS. IEEE, 2013

5. M. Asiatici et al., "Virtualized Execution Runtime for FPGA Accelera tors in the Cloud," IEEE Access, vol. 5, 2017

6. J. Zhang, et al., "The Feniks FPGA Operating System for Cloud Computing," in 8th APSys, 2017

7. S. Byma et al., "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in FCCM, May 2014

8. S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA Accelerators for Efficient Cloud Computing," in CloudCom, Nov 2015.