Main Project Report
on
Graphical Processing Unit

submitted by

Sahil Athrij (12170060)
Rahul Sethu (12170057)

In partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering.

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

APRIL 2020

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CERTIFICATE

Certified that this is a bonafide record of the Main Project titled

Graphical Processing Unit

done by
Sahil Athrij (12170063)
Rahul Sethu (12170057)

of VIII Semester, Computer Science and Engineering in the year 2019 in partial fulfillment requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering of Cochin University of Science and Technology.

Dr. Latha R Nair	Mrs. Ancy Zachariah	Mr. Vinod Kumar P P
Head of Division	Project Coordinator	Project Guide

Acknowledgement

We take this opportunity to thank the supreme being, the source of all knowledge whose blessings are our guiding light in any venture we take up. We're in short of words to express our gratitude to Mr. Vinod Kumar P P , our project guide and staff advisor who guided us and helped us constantly with her inputs and suggestions, without which we couldn't have implemented this project the way it is working today. We are highly indebted to Mrs. Ancy Zachariah, our staff advisor for her constant supervision and support in completing this project. We also express our heartfelt thanks to Mrs. lino Murali who was in charge of the project lab during the semester for helping us by providing all the necessary amenities for completing the project. A bouquet of gratitude to Dr. Latha R Nair, Head of Division of Computer Science and Engineering for all kinds of encouragement extended to us.

Rahul Sethu(12170057)
Sahil Athrij(12170063)

Declaration

We, Mr. Rahul Sethu, Mr. Sahil Athrij hereby declare that this main project is the record of authentic work carried out by us during the academic year 2019 - 2020 and has not been submitted to any other University or Institute towards the award of any degree.

Abstract

The Graphics Card is responsible for rendering an image to your monitor, it does this by converting data into a signal your monitor can understand.The better your graphics card the better, and smoother an image can be produced.

It is basically an electronic chip which is mounted on a video card (Graphics card). Occasionally called visual processing unit (VPU) is a specialized processor that offloads 3D graphics rendering from the microprocessor. The modern GPU is not only a powerful graphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth that substantially outpaces its CPU counterpart.

As the GPU creates images, it needs somewhere to hold information and completed pictures. It uses the card's RAM for this purpose, storing data about each pixel, its color and its location on the screen. Part of the RAM can also act as a frame buffer, meaning that it holds completed images until it is time to display them. Typically, video RAM operates at very high speeds and is dual ported, meaning that the system can read from it and write to it at the same time.

Contents

List of Figures

Chapter 1

Introduction

A graphics card is a type of display adapter or video card installed within most computing devices to display graphical data with high clarity, color, definition and overall appearance. A graphics card provides high-quality visual display by processing and executing graphical data using advanced graphical techniques, features and functions.A graphics card is also known as a graphics adapter, graphics controller, graphics accelerator card or graphics board.

GPU is similar to a computer's CPU. A GPU, however, is designed specifically for performing the complex mathematical and geometric calculations that are necessary for graphics rendering. Some of the fastest GPUs have more transistors than the average CPU. A GPU produces a lot of heat, so it is usually located under a heat sink or a fan.
A graphics card is primarily designed to remove the graphical processing tasks from the processor or RAM. It includes a dedicated graphical processing unit (GPU) and a dedicated RAM that help it to process graphical data quickly. Like most processors, a graphics card also has a dedicated heat sink to keep the heat out of the GPU. A graphics card enables the display of 3-D images, image rasterization, higher pixel ration, a broader range of colors and more. Moreover, a graphics card includes various expansion ports such as AGP, HDMI, TV and multiple monitor connectivity. A graphics card can be integrated within the motherboard or be added on as an extension card.

Chapter 2

Literature Review

2.1 Survey on Graphical Processing Unit

A graphics processing unit (GPU) is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images. In the early days of computing, the central processing unit (CPU) performed these calculations. As more graphics-intensive applications such as AutoCAD were developed, however, their demands put strain on the CPU and degraded performance. GPUs came about as a way to offload those tasks from CPUs and free up processing power. [2]. Today, graphics chips are being adapted to share the work of CPUs and train deep neural networks for AI applications. A GPU may be found integrated with a CPU on the same circuit, on a graphics card or in the motherboard of a personal computer or server. NVIDIA, AMD, Intel and ARM are some of the major players in the GPU market. In addition to its processing power, a GPU uses special programming to help it analyze and use data. ATI and nVidia produce the vast majority of GPUs on the market, and both companies have developed their own enhancements for GPU performance. To improve image quality.The programmable units of the GPU follow a single program multiple-data (SPMD) programming model. For efficiency, the GPU processes many elements (”vertices or fragments) in parallel using the same program. Each element is independent from the other elements, and in the base programming model, elements cannot communicate with each other. All GPU programs must be structured in this way: many parallel elements, each processed in parallel by a single program.

Chapter 3

System Analysis

3.1 Existing System

The Most Powerful Graphical processing units that currently exist in the market today are the GPU that are made on the Turing micro architecture. refer figure 3.1 that shows the internal TU102 Die and fig 3.2 shows the architecture of the TU102

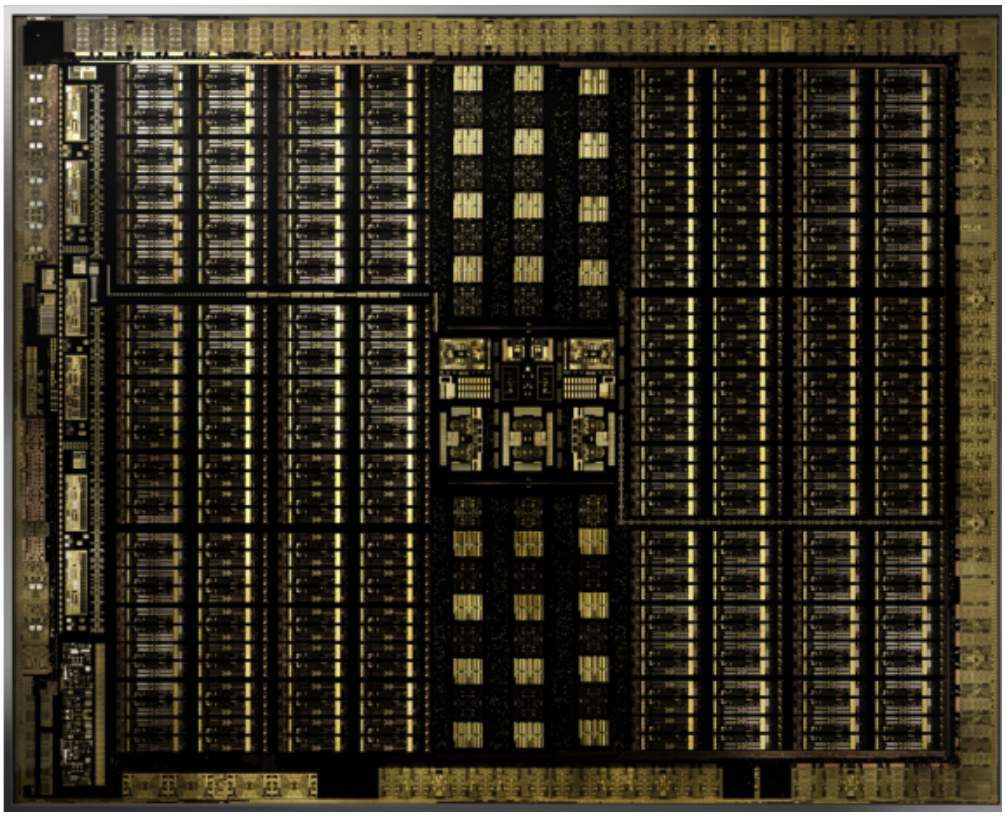

Figure 3.1: the die Structure of the TU102 that powers the RTX Titan

Figure 3.2: the die Structure of the TU102 that powers the RTX Titan

These GPUs like the Quadro RTX 8000 and the RTX Titan have Tera-Flops of compute performance and special units that speed up vector calculations. Features includes in these cards are:

1. **5000 Stream processors**

   - **stream processor**: stream processors or better known as CUDA cores for the nvidia cards are like the processors of a CPU , but they are much more simpler and numerous on a single die. more CUDA cores a GPU has ,the more tasks it can handle at once.
   - **RTX cores**: RTX cores are like CUDA cores , they are more powerful, being able to perform complex matrix and vector operations that are necessary for machine learning and complex ray tracing. when not in use for this purpose RTX cores can also double as CUDA cores

2. **1GHz+ clock speed**

   - **clock speed**: Clock Speed determines the rate at the bus is synchronised and machine cycle is executed by a machine, a instruction

is executed in 'n' number of clock cycles , so faster clock speed will have faster instruction execution.Clock Speed also affects the power consumed and the heat generated

3. **48 GB of on board video memory**
    - **video memory (CPU rendering)**: VRAM is a buffer between the computer processor and the display, and is often called the frame buffer. When images are to be sent to the display, they are first read by the processor as data from some form of main (non-video) RAM and then written to VRAM. the data is then directly sent to the display device without further modifications. The VRAM was not separate from main memory
    - **video memory (GPU rendering)**: The CPU places objects to be rendered in the memory of the GPU known as VRAM . This VRAM is separate from the main memory of the system. the GPU the calculates the value of each pixel that is to be rendered on screen using the object data and send it to the display
4. **14 Gbps memory speed on PC-IE-3.0 and HBM-2**
    - **PC-IE-3.0**: Peripheral Component Interconnect Express: PCI Express 3.0 can carry a bit rate of 8 gigatransfers per second (GT/s), and that it would be backward compatible with existing PCI Express implementations
    - **HBM-2** High Bandwidth Memory (HBM) is a high-performance RAM interface for 3D-stacked SD-RAM. HBM achieves higher bandwidth while using less power in a substantially smaller form factor than DDR4 or GDDR5 This is achieved by stacking up to eight DRAM dies (thus being a Three-dimensional integrated circuit)

The products from Nvidia although unmatched in performance , it is still closed source , the architecture is kept highly secret and is not available to study. The open source front of GPU is much less advanced . unlike the CPU front where there have been tons of open source architectures , there has been only one project in the GPU front , named MIAOW , it was developed by Vertical Research Group at the University of Wisconsin-Madison led by

Professor Karu Sankaralingam. but the project was last updated 3 years. The project was based on AMD's open source instruction set GCN or Graphic core next. Fig 3.3 shows the MIAOW Architecture

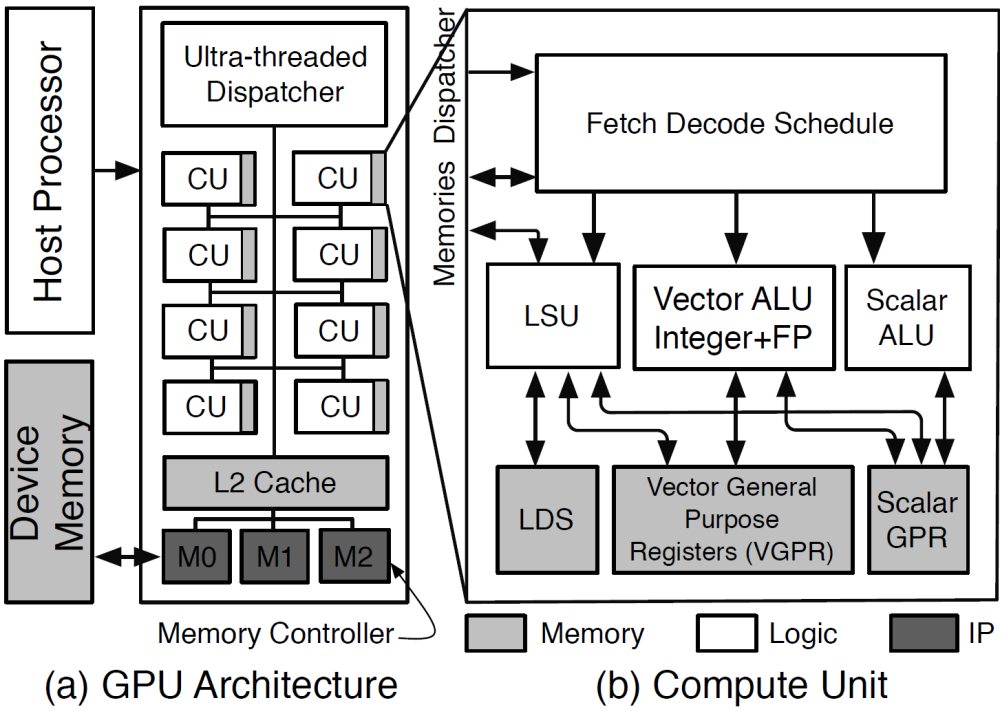

Figure 3.3: the die Structure of the TU102 that powers the RTX Titan

3.2 Proposed System

Our proposed Graphical Processing Unit is a basic GPU model build to simulate the working of graphical processing unit.the goal is not to compete with commercial GPU.But to get more open source projects in the GPU space, that if based on a RISC-V , there could be wider support for. This project is not done to sell a product, neither is it done even to make a big impact on the open source scene , this project is done mostly as a learning exercise . the marketability has no impact on the product

the proposed system will have:

- 5 modular stream processors
- 4 - 16 stream processors
- 16 bit arithmetic and logic operations

- 16 bit memory and data bus
- modular registers
- 4- 16 registers per core
- 128 Kb of memory as 2 * 64Kb frame buffers
- a VGA port for outputting analog video

Chapter 4

System Study

In this section, we are going to present the SRS, system objectives and hardware and software tool requirements.

4.1 Software Requirements Specification

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform.

Purpose

- To Create A general purpose graphical processing unit.
- To Create a Basic Custom assembly language or implement.
- To Create a very basic C compiler for the custom assembly language.

Project Overview

Product Perspective

The outline steps involved in making a GPU are:

1. Create a basic CPU
2. Create an instruction Set
3. Create 4 - 16 more CPU's
4. Modify CPU design to add thread dispatcher
5. Combine the 4 - 16 CPU along with the master control CPU
6. Create specialized instruction set for the GPU
7. Create circuit to output alternative frame buffer
8. Create a assembler for the instruction set
9. Create a C compiler for the assembly language

4.2 Interface Requirements

Hardware Interface

The hardware requirement is a a lot of semiconductor electronics. TTL is the chosen for its lower cost as compared to CMOS. over 10000 individual THT or SMD components will be required to complete this project , including more than 1600 PCBs

Software Interface

To develop the product , a electronic simulator like modelSim or verilog is used to generate the PCB layout. custom software would be required to create

a assembly language that takes text input and converts it into machine code and sends it to the GPU. more custom software would be required to create a C compiler for that assembly language

4.3 System Features

The features implemented in this system are hardware division and multiplication, 16 bit instruction set,parallel instruction execution and On Screen Rendering

1. **hardware division and multiplication**

   - ***Description and Priority***
     High priority.
     Have the Arithmetic and logic unit compute division and multiplication happen in O(1) time.
   - ***Response Sequences***
     The MUL or DIV instruction is executed within 10 clock cycles on.

   - ***Functional Requirements***
     REQ-1: Two numbers stored to the ALU register.
     REQ-2: The Multiplier or divider output selected.

2. **16 bit instruction set**

   - ***Description and Priority***
     High priority.
     create a 16 bit instruction set that the machine can understand.

   - ***Response Sequences***
     The instruction set is kept in a EEPROM in control units of each of the cores, the instructions can then be decoded to the appropriate control signals per clock tick.

- ***Functional Requirements***
  REQ-1:Translation table for each BYTECODE from the machine code to the control signals.

  – Input: byte code to the table.
  – Output: sequence of control signals to perform the instruction.

  REQ-2: EEPROM to store the control signals.
  – Input: BYTECODE as address to the EEPROM.
  – Output: The control signals to the data lines of the EEPROM

  REQ-3: Instruction Decoder to decode the instruction on a per clock basis
  – Input: A clock to the instruction decoder.
  – Output: The control signals per clock tick.

3. **parallel instruction execution**

- ***Description and Priority***
  High priority
  Use the multiple cores available to perform multiple different instruction at the same time.

- ***Response Sequences***
  The master Thread control CPU assigns programs to each of the cores and each core can access memory and read and write data to it.The cores then individually finish the assigned process and notifies the Thread Dispatcher that its job is complete.

- ***Functional Requirements***
  REQ-1: Thread Dispatcher to assign jobs.
  – Input: instruction to the Thread dispatcher to assign specific code to specific core
  – Output: Thread dispatcher puts the location into the program counter

  REQ-2: Stream Processing cores that can be started by the thread dispatcher.

- Input:Thread Dispatcher sends signal to start execution
- Output: the core that was told to start execution starts running the program

REQ-3: Buffer in the thread dispatcher where individual cores can notify its job is complete or not

- Input:Core sends signal into a register of the thread dispatcher that the programlet has finished executing
- Output:the core goes to the idle state and the thread dispatcher can asssign more jobs if required

REQ-4: individual memory access for the individual cores.

- Input:Cores write to frame buffer at the same time
- Output:the values are written to the memory at the same time, conflicts are ignored and later values wins

4. **On-Screen Rendering**

- ***Description and Priority***
  High priority.
  Render the image on display.

- ***Response Sequences***
  When Display circuit is activated.
- ***Functional Requirements***
  REQ-1:select Alternating frame buffer to be selected.
  REQ-2: 16 bits in frame buffer 4 for Red, 4, for Green 4 for Blue converted to analog.
  - Input: 16 bit RGB digital number.
  - Output: voltage values between 0-.7 that determine the brightness of each colour per pixel.

  REQ-3: sending voltage value to the display.
  - Input: the volatage value of each pixel.
  - Output: the voltage values are sent to the R G and B pins on the VGA output of the display

4.4 Other Nonfunctional Requirements

Performance Requirements

The product must run at 1 megahertz, must be able to output 1 frame per 30 micro-second, must be able to perform all basic operations within 12 clock cycles and must be able to continuously perform the above base tasks indefinitely.

Security Requirements

there will be no speculative execution or multiple different programs executing at the same time , so all security problems that will arise must be only due to malicious code and badly written code and not due to hardware design .

Hardware Quality Attributes

The Electronics must be of good quality, having low gate propagation delays, the boards must be 4 layer PCBs stacked on top of each other to make good contact with the connector pins

Chapter 5

System Design

5.1 Introduction

Designing requires a careful planning and thinking on the part of the system designer. Designing a system means to plan how the various parts of it are going to achieve the desired goal. After the software requirements have been analyzed and specified, design is the first of the three technical activities. Designing, coding and testing are required to build and verify the mobile application.

5.2 Data Flow Diagram

The following figures from Figure 5.1 to 5.4 shows the different levels of data flow from the most abstract level to the most detailed level.

Level-0 Data Flow

Figure 5.1 shows the data flow between the Main Memory , CPU and GPU. Upon receiving the input from the CPU, the program to be executed is fetched from the main memory into the GPU memory or VRAM, which is then processed by the GPU cores and either rendered on screen or sent back to the main memory.

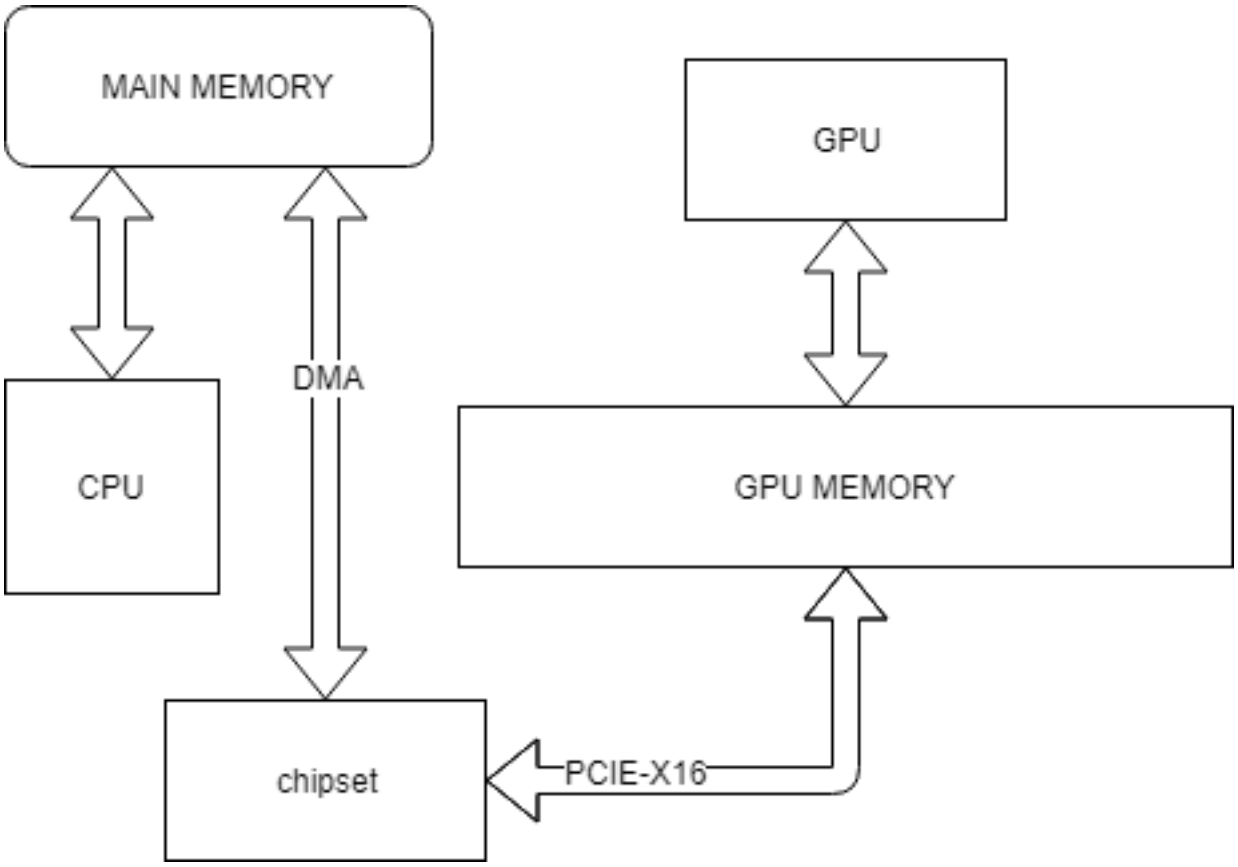

Figure 5.1: Schematic diagram Entire computer

Level-1 Data Flow CPU

Figure 5.2 shows the data flow on the CPU side. The Instruction is sent to the instruction register, It is decoded and control signals generated. the control signals then tell each individual component what it must do on each clock cycle until the next instruction is loaded and the cycle continues

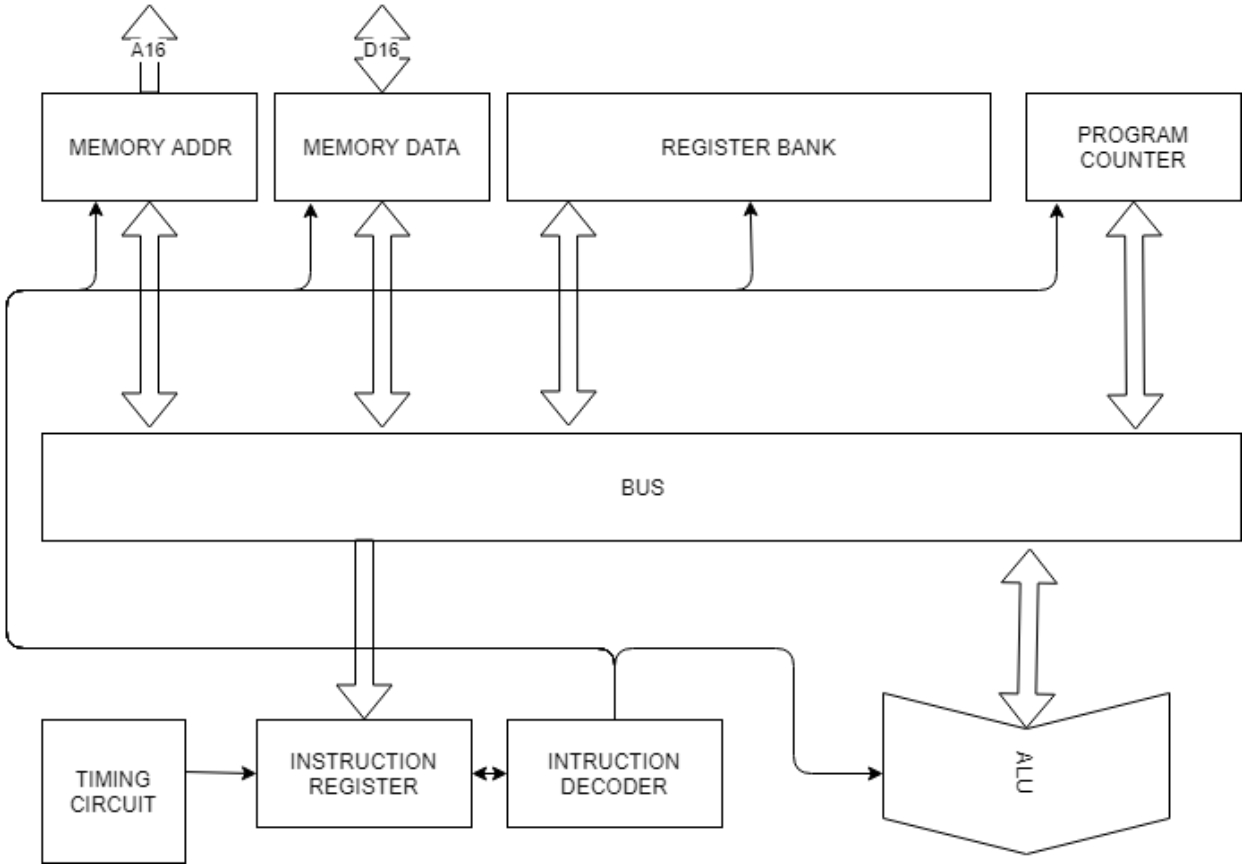

Figure 5.2: Schematic diagram of CPU

Level-1 Data Flow Of GPU

Figure 5.3 shows the data flow on the GPU. When a new JOB is assigned by the CPU , the thread dispatcher puts the required local values for all the individual cores in the local memory of the cores, then the cores start executing the program assigned by the thread dispatcher , present on the shared video ram , the value results are then stored back onto the video ram

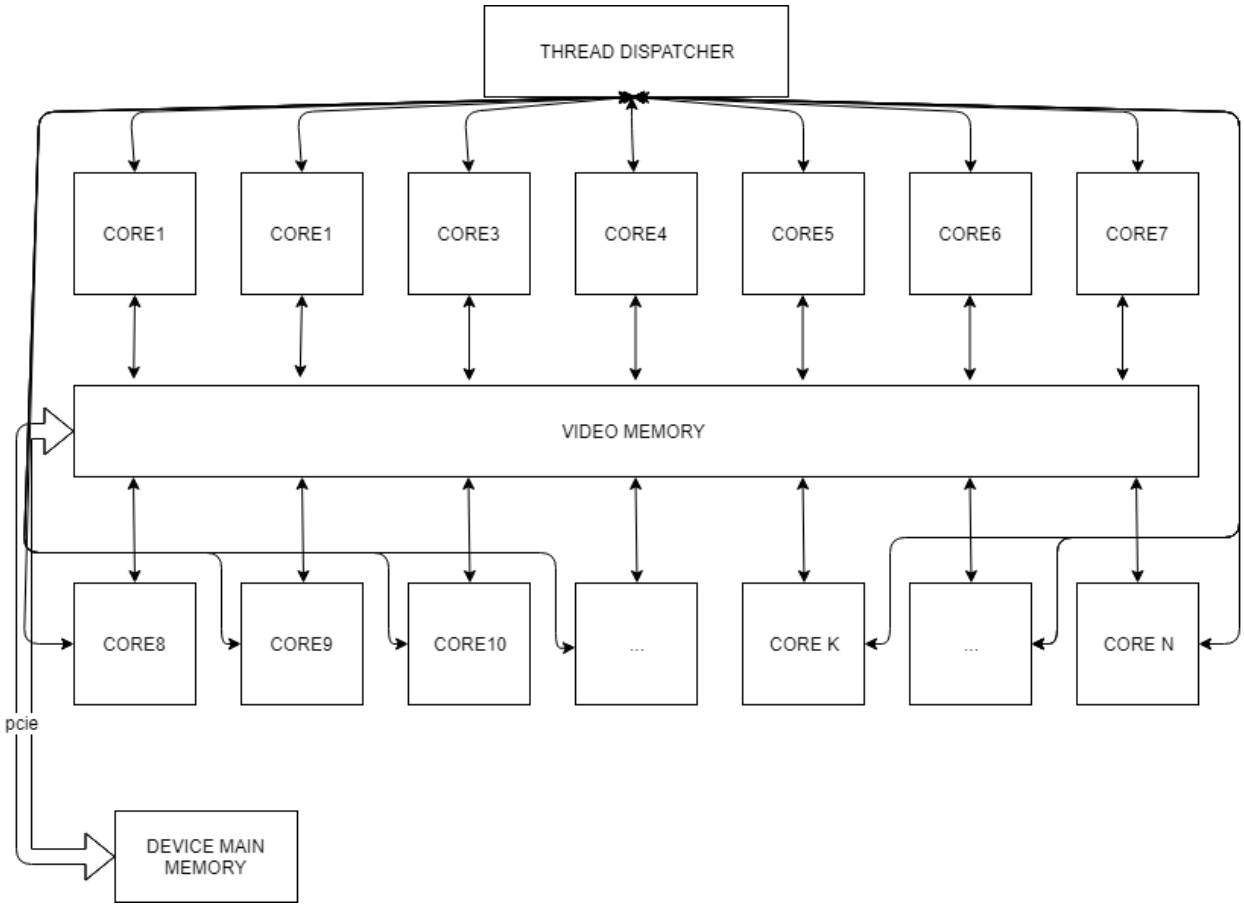

Figure 5.3: Schematic diagram of GPU

Level-2 Data Flow

Figure 5.4-5.7 shows the internal structure of the important functional units of the hardware . fig 5.5-5.7 shows the structure of a register. fig 5.8-5.9 shows the structure of a adder 5.10-5.11 shows the structure of a fixed cycle multiplier and fig 5.12 shows the structure of a fixed cycle divider

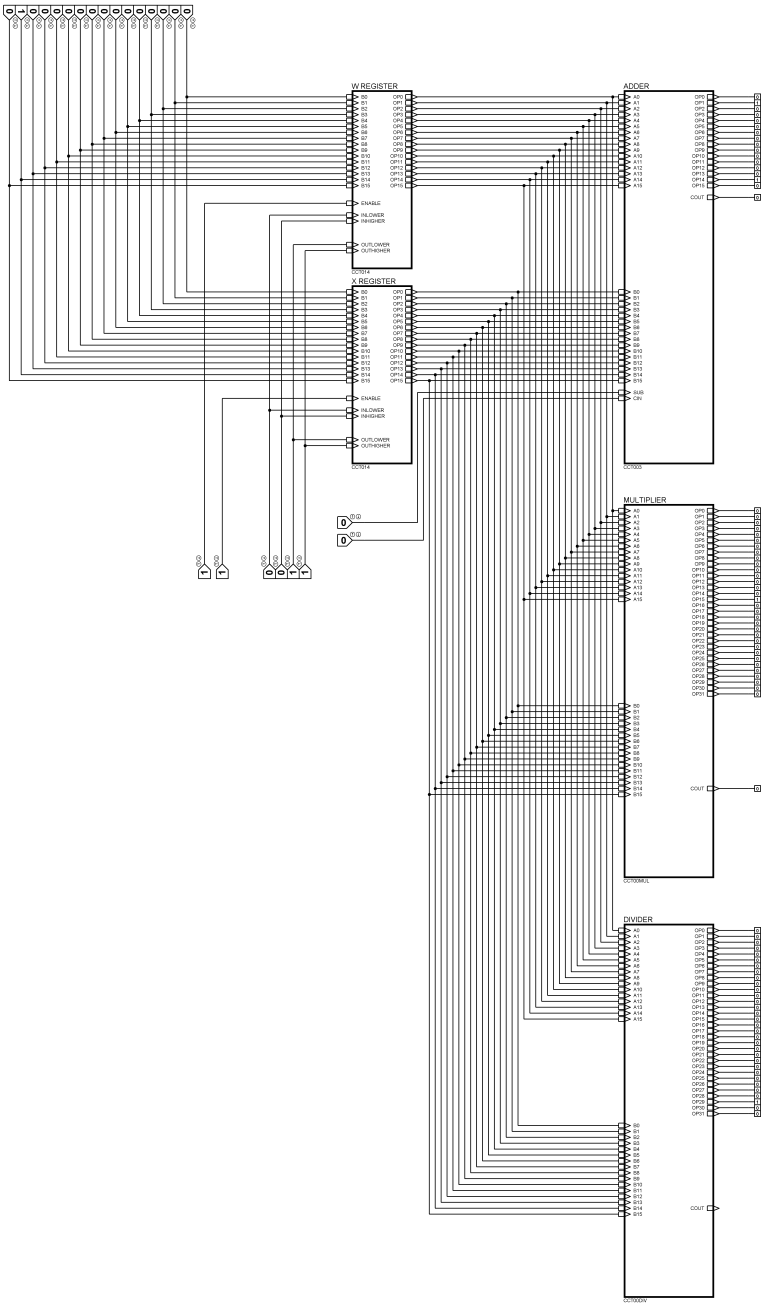

Figure 5.4: Schematic diagram of ALU

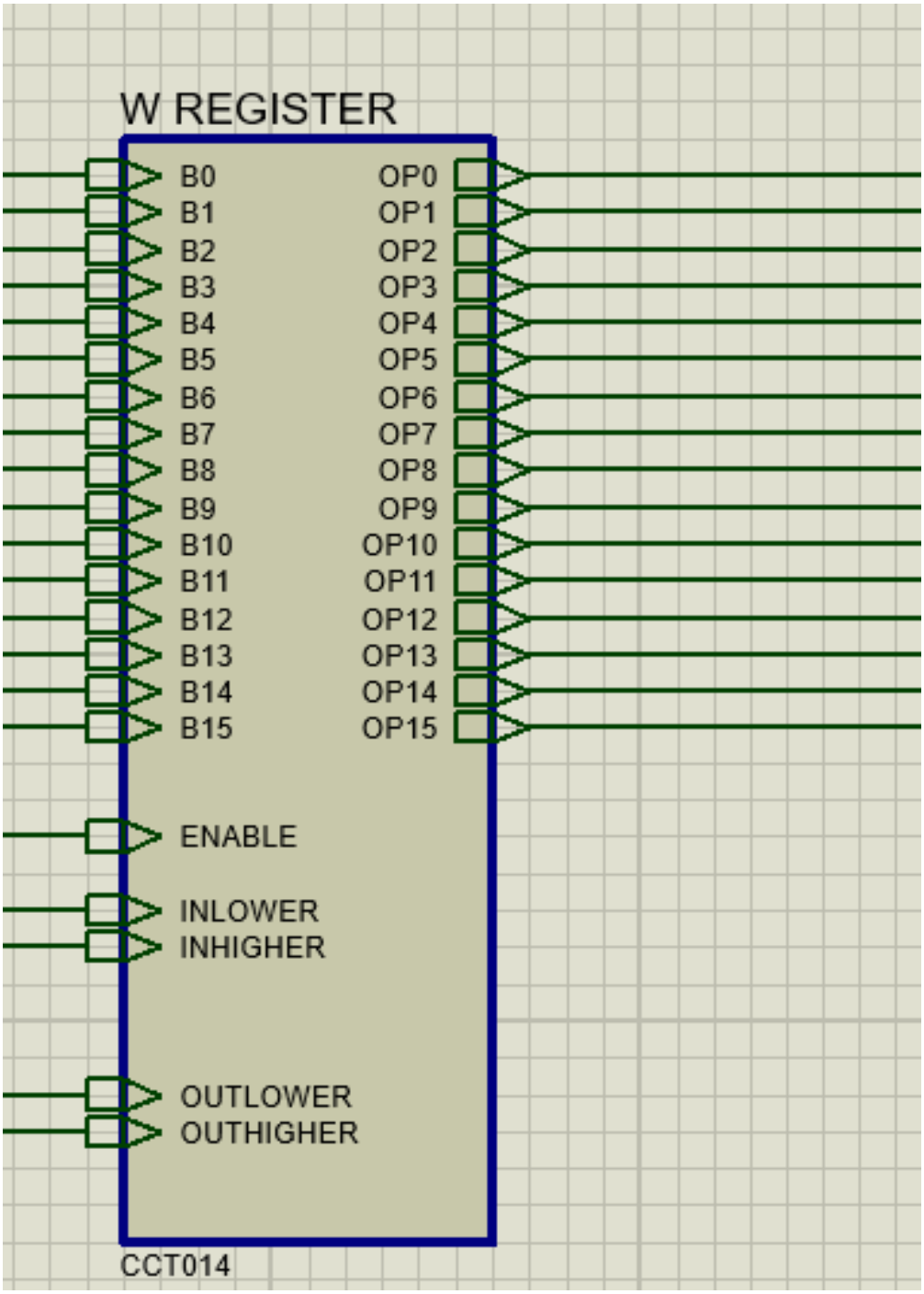

Figure 5.5: Schematic diagram of register

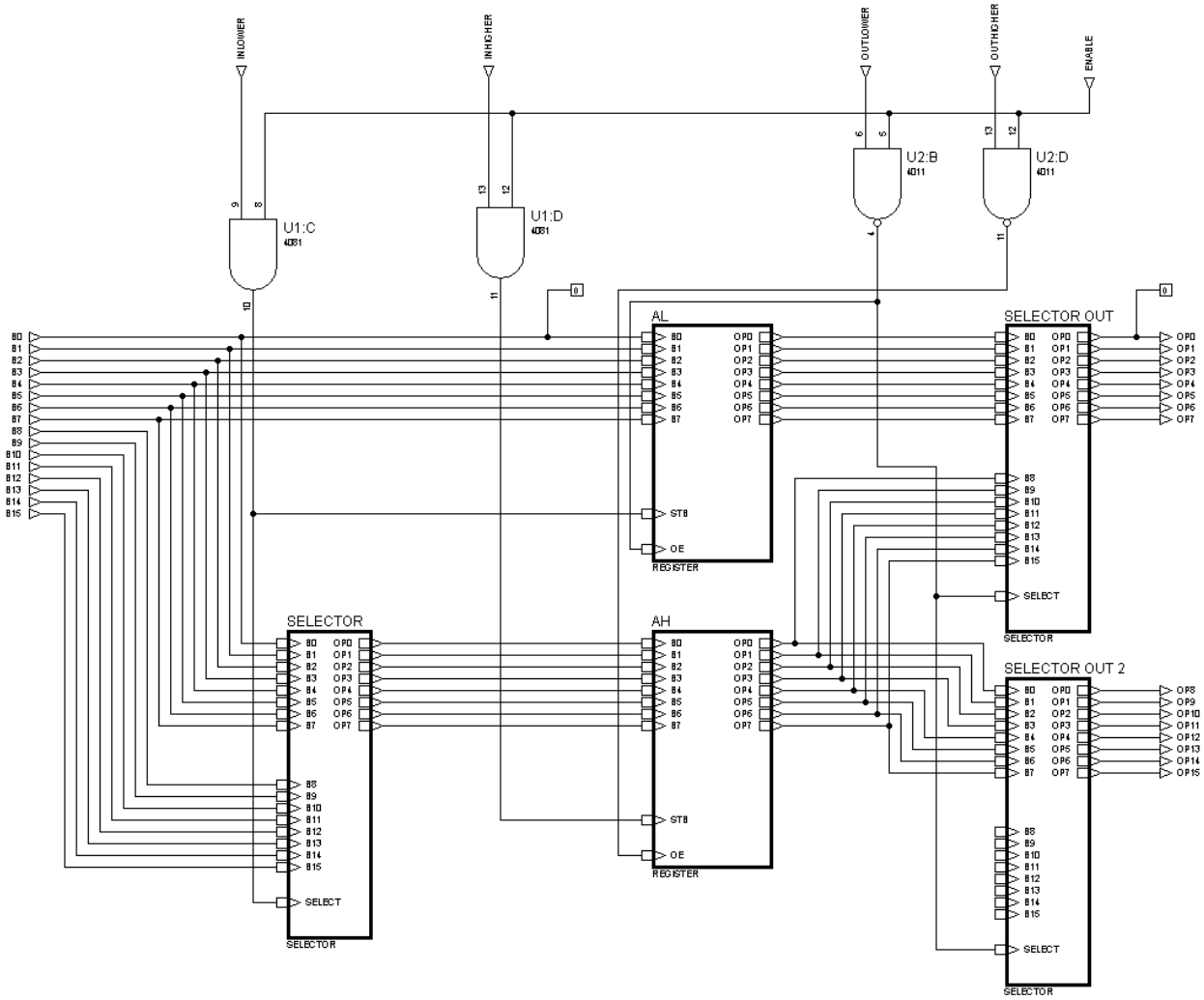

Figure 5.6: Schematic inside of a full register

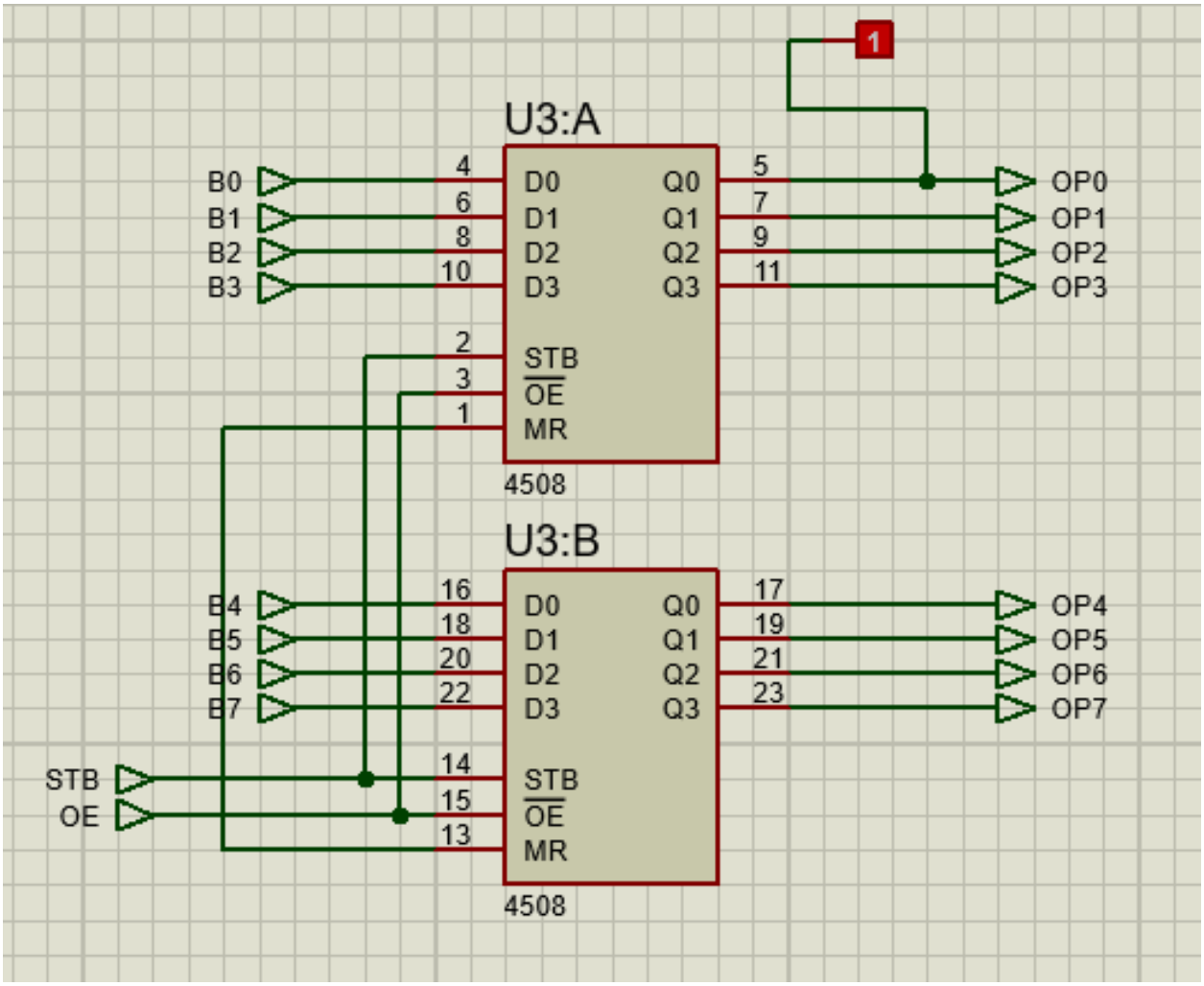

Figure 5.7: Schematic inside of a half register

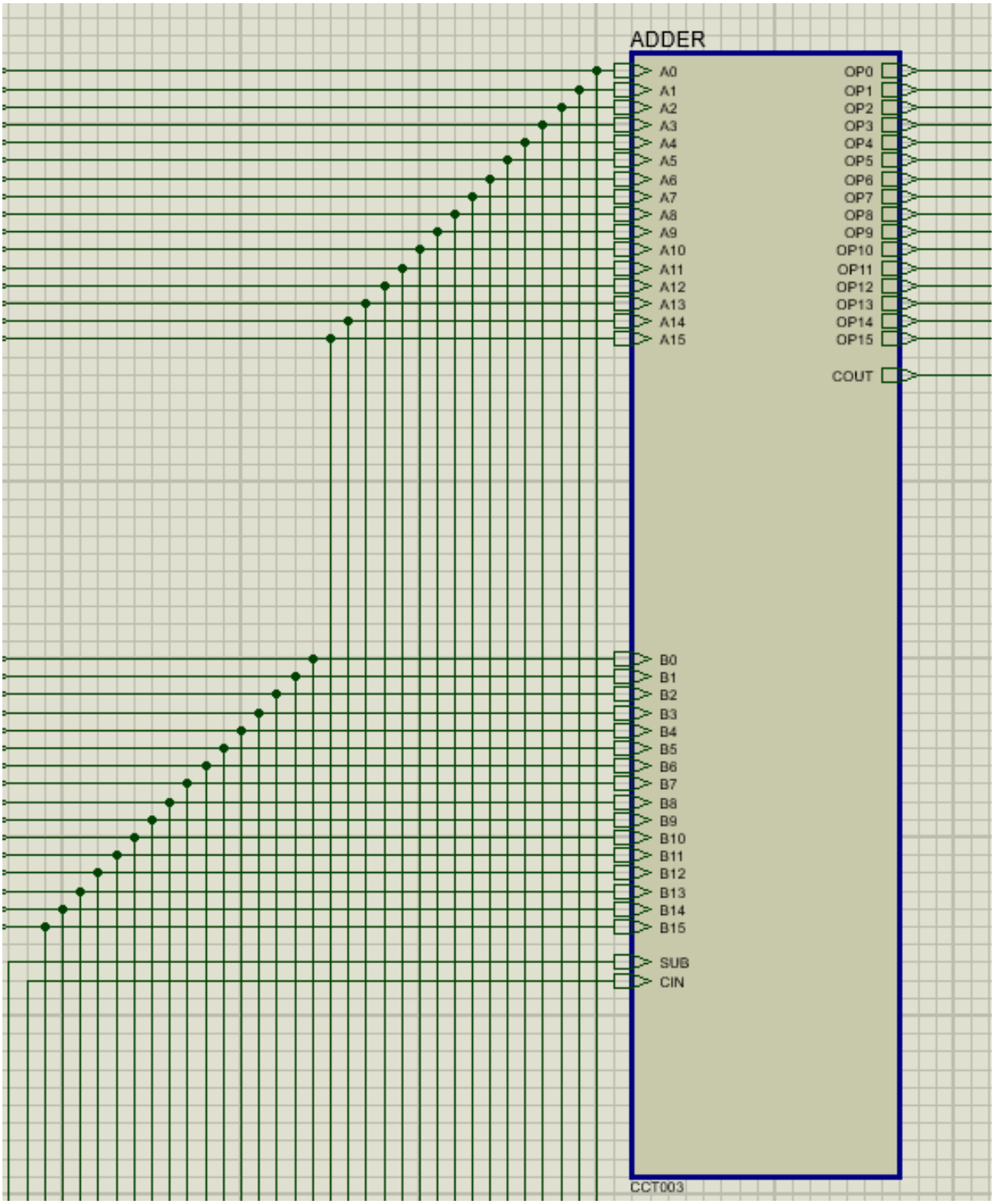

Figure 5.8: Schematic diagram of Adder

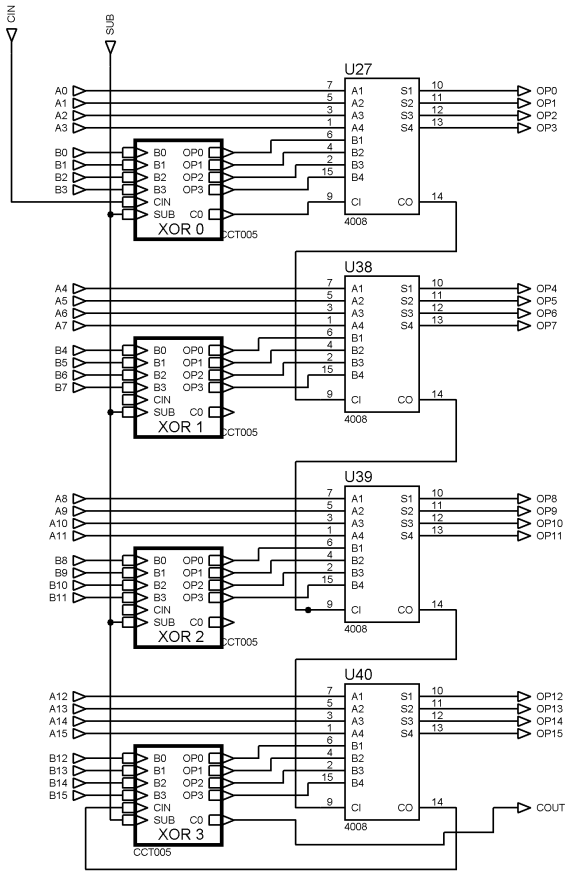

Figure 5.9: Schematic diagram of inside Adder

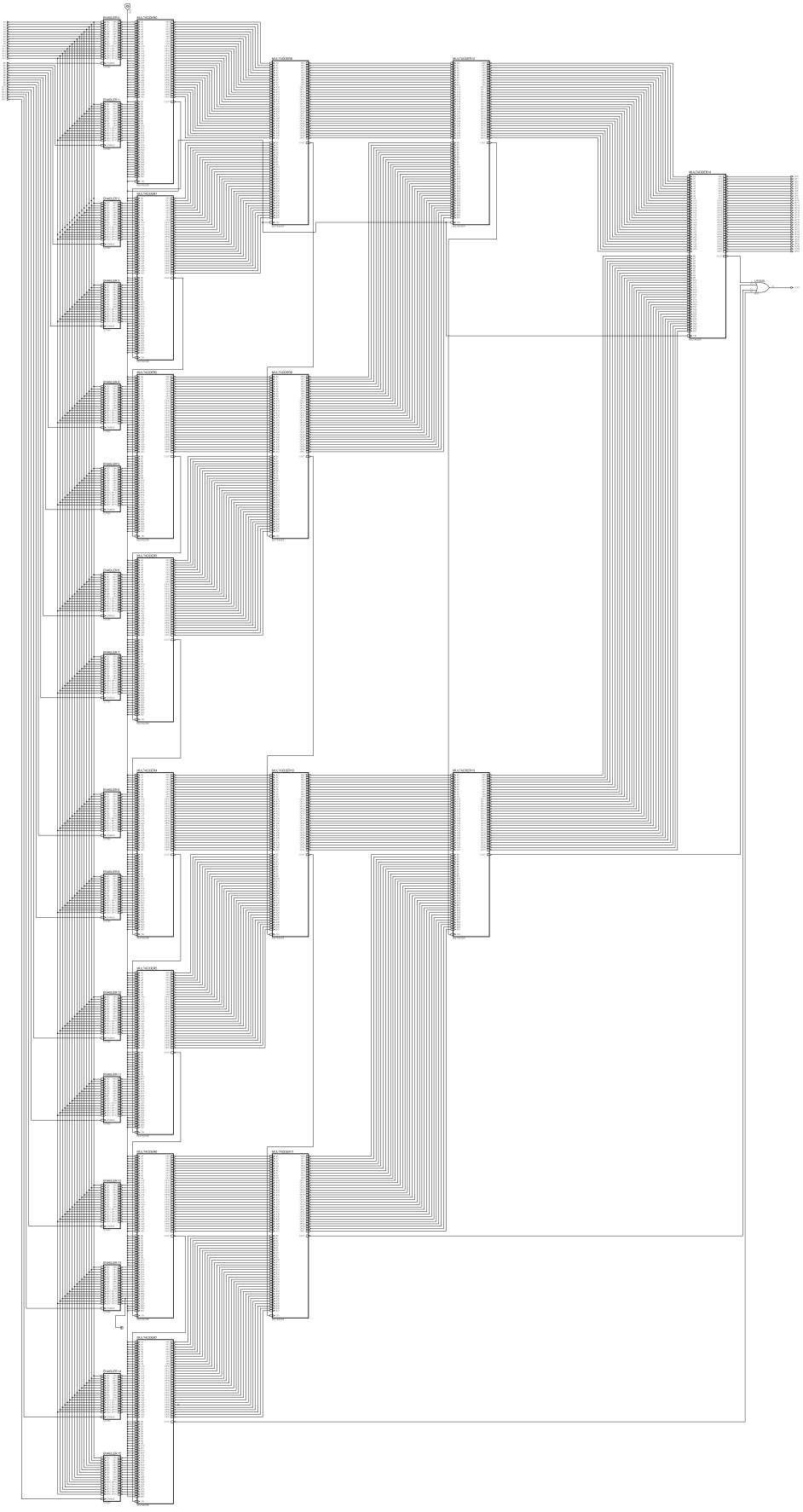

Figure 5.10: Schematic diagram of inside multiplier

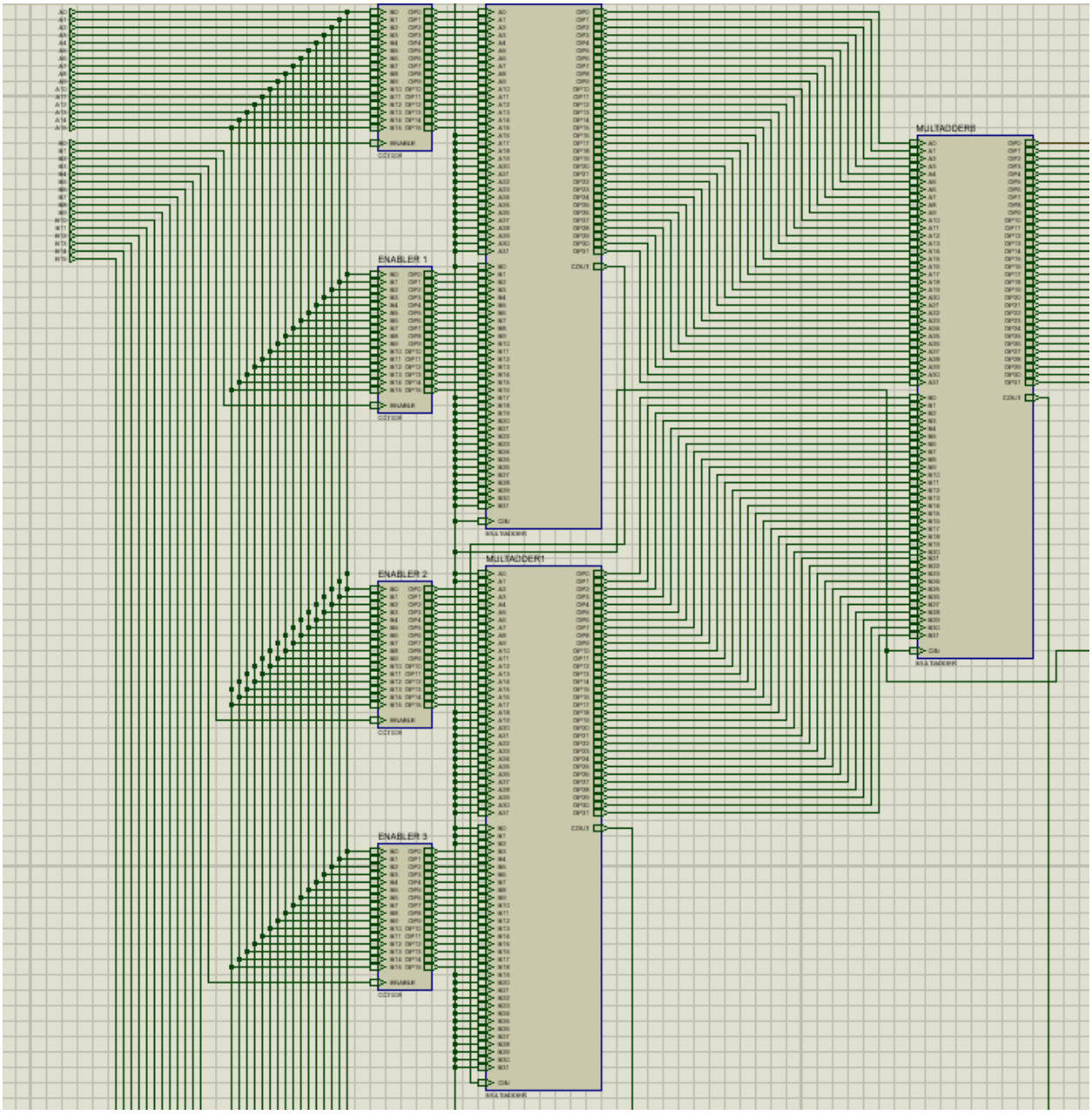

Figure 5.11: Schematic diagram of 4 bit multiplier multiplier

Figure 5.12: Schematic diagram of divider

Figure 5.13: Schematic diagram of divider

Chapter 6

Instructions And Uses

Data transfer Instruction

- ***mov r1,r2***
  Copy content of register r1 and then store it in register r2.
- ***mov r1,addr***
  Copy content of register r1 and store it at address 'adddr'.
- ***mov addr,r1***
  Copy content in memory location addr and store them in register r1.
- ***mvi r1,data***
  Copy content of register r1 and then store it in register r2.

Arithmetic Instruction

- ***Add r1,r2***
  Add content of register r1 and r2 then store the result in r2.
- ***Adi r1,data***
  Add content of register r1 and constant 'data' then store the result in r1.
- ***Sub r1,r2***
  Subtract content of register r2 from r1 and then store the result in register r2.
- ***Sbi r1,data***
  Subtract constant 'data' from register r1 and then store the result in register r1.

- ***Mul r1,r2***
  Multiply the content of register r1 with r2 and then store the result in r2.
- ***Div r1,r2***
  Divide the content in register rq with content in register r2 and then store the result in r2.

Logical Instruction

- ***XOR r1,r2***
  Apply exclusive OR logical operation on content of r1 with r2 and store the result in register r2.
- ***OR r1,r2***
  Apply OR logical operation on content of r1 with r2 and store the result in register r2.
- ***AND r1,r2***
  Apply AND logical operation on content of r1 with r2 and store the result in register r2.
- ***NOT r1***
  Negate the content in register r1 and store the result in itself.

Branching Instructions

- ***JMP addr***
  Change the program control flow to address location addr.
- ***JNZ addr***
  Change the program control flow to address location addr if the zero flag is 1.
- ***JZ addr***
  Change the program control flow to address location addr if the zero flag is 0.
- ***JC addr***
  Change the program control flow to address location addr if the carry flag is 1.

- ***JNC addr***
  Change the program control flow to address location addr if the carry flag is 0.
- ***JO addr***
  Change the program control flow to address location addr if the overfow flag is 1.
- ***JNO addr***
  Change the program control flow to address location addr if the overfow flag is0.

Calling Instructions

- ***CALL addr***
  Execute the subroutine starting at memory location addr and return program flow to called location after the execution of the subroutine.
- ***CLZ addr***
  Execute the subroutine starting at memory location addr if zero flag is 1 and return program flow to called location after the execution of the subroutine.
- ***CLNZ addr***
  Execute the subroutine starting at memory location addr if zero flag is 0 and return program flow to called location after the execution of the subroutine.
- ***CLC addr***
  Execute the subroutine starting at memory location addr if carry flag is 1 and return program flow to called location after the execution of the subroutine.
- ***CLNC addr***
  Execute the subroutine starting at memory location addr if carry flag is 0 and return program flow to called location after the execution of the subroutine.
- ***CLO addr***
  Execute the subroutine starting at memory location addr if overfow flag

is 1 and return program flow to called location after the execution of the subroutine.

- ***CLNO addr***
  Execute the subroutine starting at memory location addr if overflow flag is 0 and return program flow to called location after the execution of the subroutine.

Chapter 7

CPU Core Management

Core Structure

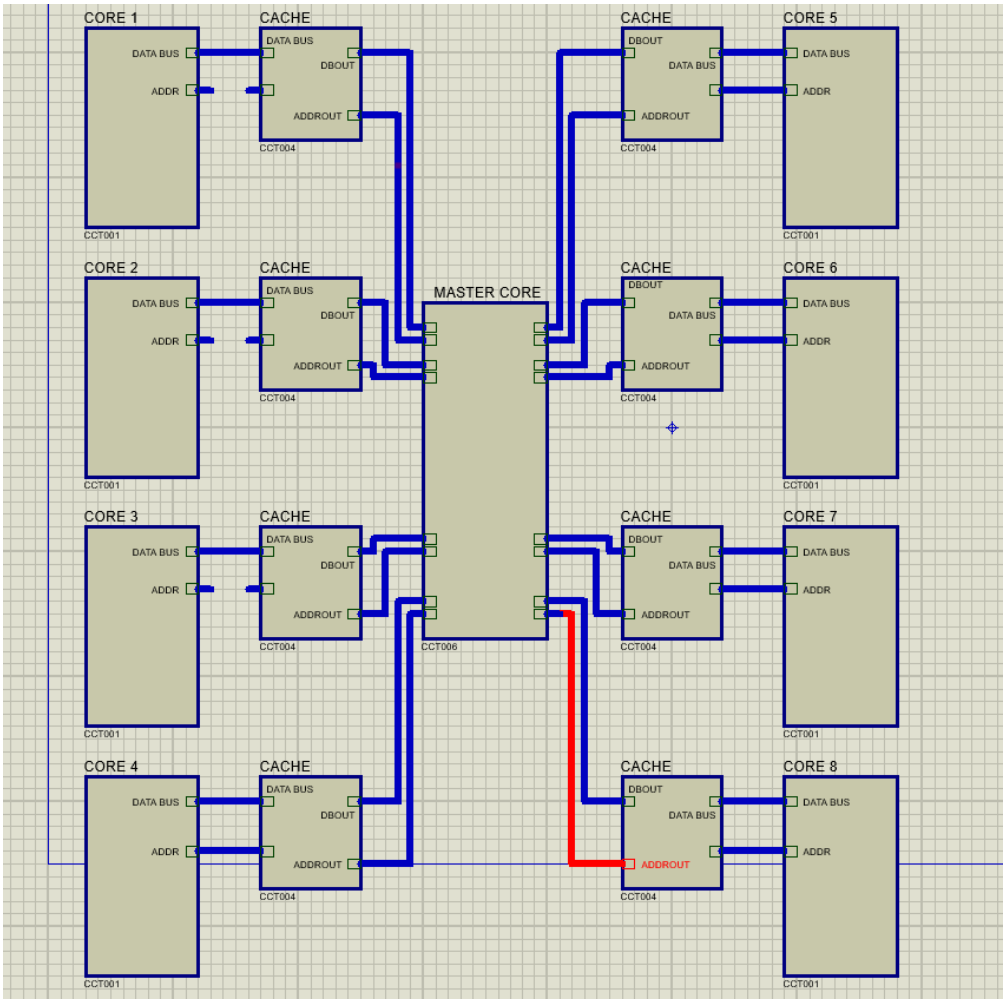

Figure 7.1: Core Structure Of GPU

Vector Instructions

- ***MVIV register,start***
  Loads a incremented vector the registers of all the cores to the register of all 8 cores
- ***LOADV addr , size***
  Loads a vector from memory adddress addr of the specified size into cache and divide it among the 8 cores max size of(8)
- ***STOREV addr,size***
  Stores a vector to memory address addr of the specified size from cache max size of(8)
- ***ADDV addr,size***
  Add the current vector stored in cache with the vector at address Addr max size of(8)
- ***SUBV addr,size***
  Subtract the current vector stored in cache with the vector at address Addr max size of(8)
- ***MULV addr,size***
  Multiply the current vector stored in cache with the vector at address Addr max size of(8)
- ***DIVV addr,size***
  divide the current vector stored in cache with the vector at address Addr max size of(8)
- ***CALLV addr***
  make the individual cores run the program at address Addr max size of(8)

Chapter 8

CPU Memory Selector

Selector Circuit of Memory

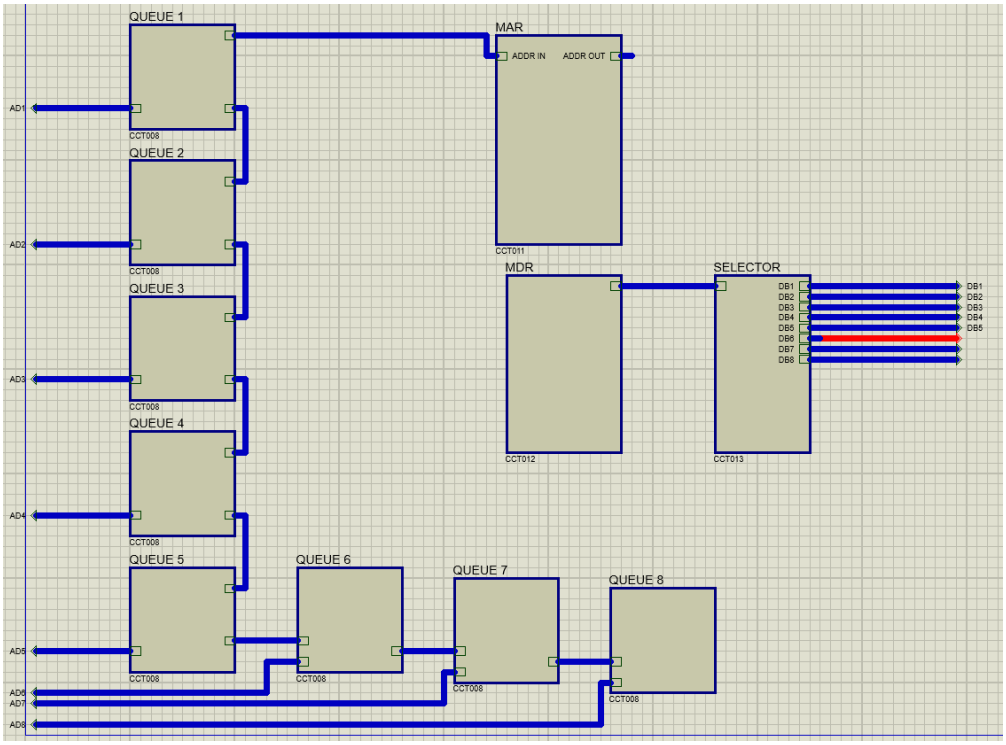

Figure 8.1: Selector memory Circuit

Frame Instructions

- ***MOVVF register,addr***
  Loads a vector from frame address addr to the register of all 8 cores
- ***MOVVF addr,register***
  stores a vector to frame address addr from the registers of all 8 cores
- ***LOADVF addr,size***
  Loads a vector from frame address addr of the specified size into cache and divide it among the 8 cores max size of(8)
- ***STOREVF addr,size***
  Stores a vector to frame address addr of the specified size from cache max size of(8)

Chapter 9

Sample Programs

Simple Addition

```
;program to add two numbers in memory and store to memory
;memory location 3040 and 3041 to store in 3042
_start:
    mov r1,3040
    mov r2,3041
    add r1,r2
    mov r3,z ;take from temp z register
    mov 3042,r3
```

Figure 9.1: Simple addition

Simple Pixel Check

```
;program to check if pixel is part of a square #F000 red or line #0F00 green or #0000 black
;square 5,5 to 10,10  (A,A) line  0,5 to 5,15 (5,F)
;frame buffer size is 256 (FF) , 16x16 (10,10) square
;assume pixel index in r1
_start:
    mvi r2,#10
    div r1,r2
    mov r3 ,w ; remainer (x)
    mov r4, z ; quotient (y)

    mvi r5,#5
    sub r3,r5
    jnz sqtest1

    mvi  r5,#0F
    mvi  r6,#05

    sub r5,r6
    mov r5,z  ;delta y

    mvi  r6,#05
    mvi  r7,#00

    sub r6,r7
    mov r6,z; delta x

    div r5,r6 ; m = y/x
    mov r7,z  ; m

    mul r7,r3
    mov r3,z
    mvi r8,#00
    add r3,r8
    mov r3,z
```

Figure 9.2: Pixel Check

```
    mvi r1,#0F00
    sub r3,r4
    jz final

    mvi r1,#0000
    jz final


sqtest1:
    mvi r5,#0A
    sub r3,r5
    jnz sqtest2

sqtest2:
    mvi r5,#05
    sub r4,r5
    jnz sqtest3

sqtest3:
    mvi r5,#0A
    sub r4,r5
    jnz finalsquare


finalsquare:
    mvi r1,#F000
    jmp final
final:
    nop
```


Figure 9.3: Pixel Check

Program to draw entire Frame

```
;program to draw a square #F000 red or line #0F00 green or #0000 black
;square 5,5 to 10,10  (A,A) line  0,5 to 5,15 (5,F)
;frame buffer size is 256 (FF) , 16x16 (10,10) square
;assume pixel index in r1

_global:
    mviv r1,#0
    mov r2,#08
    mov r3,#FF
    mov r1,#0

_global_loop:
    callv _start
    movvf [r1],r2
    add r1,r2
    sub r3,r2
    jnc _global_loop:
    hlt
```

Figure 9.4: Entire frame drawn

Output Frame Buffer

```
0000 0000 0000 0000 0000 0F00 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0F00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0F00 0000 0000 F000 F000 F000 F000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 F000 F000 F000 F000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 F000 F000 F000 F000 0000 0000 0000 0000 0000
0000 0000 0F00 0000 0000 0000 F000 F000 F000 F000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0F00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0F00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

Figure 9.5: Entire frame as hex

Output Frame Buffer

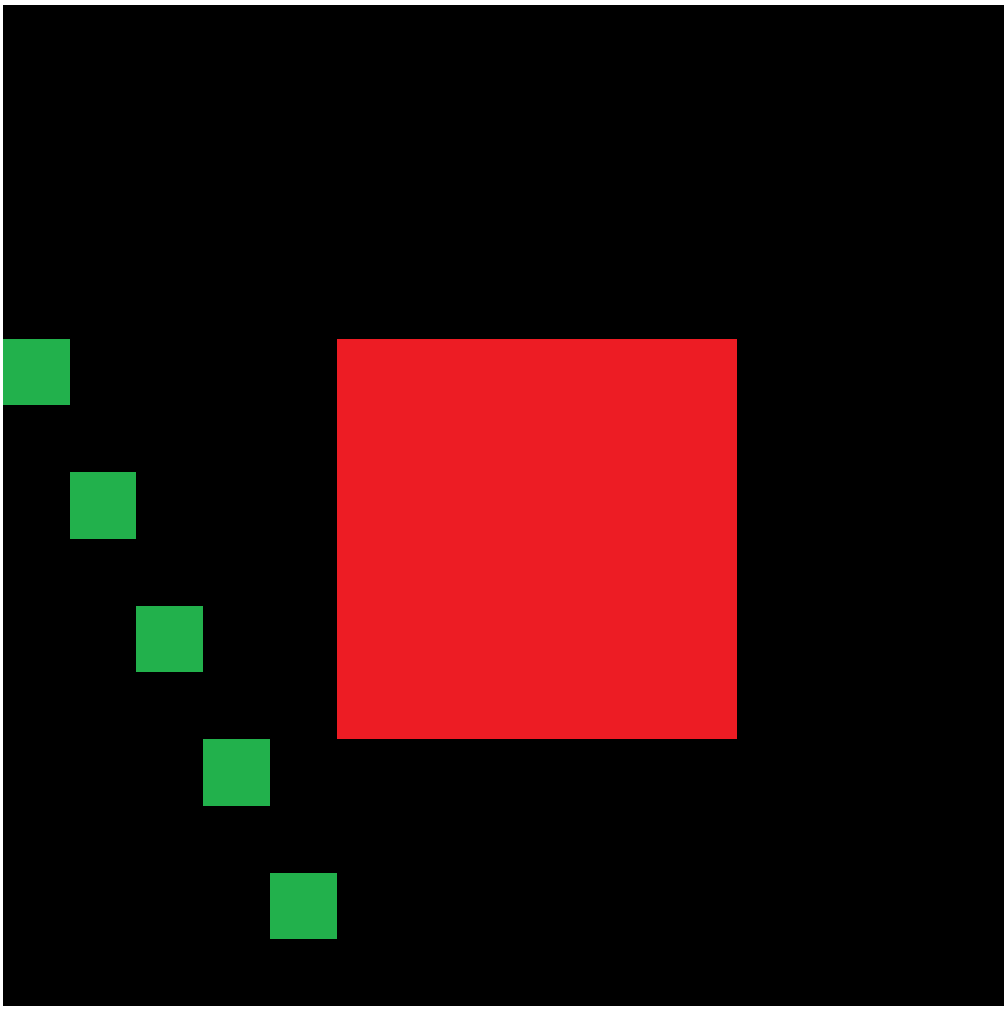

Figure 9.6: Entire frame as output

Chapter 10

Conclusion

GPU's are digital circuiteries which basically enabled video rendering possible,here a basic GPU is designed and implemented using logical designing.The basic version can simulate the processing speed of programs.The functionalities can be increased by the use of advanced algorithms.Thus a basic version of GPU which support multiprogramming has been successfully designed.

Chapter 11

Reference

[1] Denny Atkin,
"ComputerShopper:TheRightGPUforYou".
Archivedfromtheoriginalon2007-05-06.Retrieved2007-05-15.

[2] Sanders, Jason; Kandrot, Edward,
https://books.google.com/books?id=49OmnOmTEtQC

[3] John Nickolls ,
https://www.youtube.com/watch?v=nlGnKPpOpbE

[4] F.Robert A. Hopgoodv ,
https://books.google.com/books?id=2j4hTAqxJ_sC&pg=PA169

[5] Mittal et al., & Xiangyu Wang ,
https://www.academia.edu/40135801/A_Survey_of_Techniques_for_
Optimizing_Deep_Learning_on_GPUshttps:
//www.academia.edu/40135801/A_Survey_of_Techniques_for_
Optimizing_Deep_Learning_on_GPUs

[6] Bradley Sanford,
https://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/Integrated_Graphics_Solutions_white_paper_rev61.pdf

[7] Raina, Rajat ,
http://dl.acm.org/citation.cfm?id=1553486

[8] S Harding and W Banzhaf ,
http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/eurogp07_harding.html

[9] Smith, Ryan , https://www.anandtech.com/show/11367/nvidia-volta-unveiled-gv100-gpu-and-tesla-v100-accelerator-announced

[10] Hill, Brandon ,
https://hothardware.com/news/amd-7nm-navi-gpu-ai-deep-learning