ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

CUSTOM DIRECT MEMORY ACCESS MODULE DESIGN AND IMPLEMENTATION

SENIOR DESIGN PROJECT

Asya TURHAL
Meyra ALPASLAN

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

JUNE 2022

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

CUSTOM DIRECT MEMORY ACCESS MODULE DESIGN AND IMPLEMENTATION

SENIOR DESIGN PROJECT

Asya TURHAL
(040180730)

Meyra ALPASLAN
(040170006)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

Project Advisor: Prof. Dr. Berna Örs Yalçın

JUNE 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

ÖZEL DOĞRUDAN BELLEK ERİŞİMİ MODÜL TASARIMI VE UYGULAMASI

LİSANS BİTİRME TASARIM PROJESİ

Asya TURHAL
(040180730)

Meyra ALPASLAN
(040170006)

Proje Danışmanı: Prof. Dr. Berna Örs Yalçın

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

HAZİRAN, 2022

We are submitting the Senior Design Project Report entitled as “CUSTOM DIRECT MEMORY ACCES MODULE DESIGN AND IMPLEMENTATION”. The Senior Design Project Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity .

Asya TURHAL .... [illegible]
(040180730)

Meyra ALPASLAN .... [illegible]
(040170006)

FOREWORD

We would like to thank our esteemed advisor, Prof. Dr. Berna Örs Yalçın, for her constant help, assistance and support throughout the project. We would also like to thank our university's VLSI Laboratory and its members, who provided us with the tool we need to use for the realization of the project and for their help and support in every way related to these tools and the project.

June 2022

Asya TURHAL
Meyra ALPASLAN

TABLE OF CONTENTS

Page

ABBREVIATIONS

DMA	**:** Direct Memory Access
CPU	**:** Central Processing Unit
ASIC	**:** Application-Specific Integrated Circuit
RISC	**:** Reduced Instruction Set Computer
RAM	**:** Random Access Memory
DPRAM	**:** Dual Port RAM
SoC	**:** System on Chip
ISA	**:** Instruction Set Architecture
FIFO	**:** First In First Out
RTL	**:** Register Transfer Level

LIST OF FIGURES

**CUSTOM DIRECT MEMORY ACCESS
MODULE DESIGN AND IMPLEMENTATION**

SUMMARY

With the development of technology, the boundaries of the studies, researches and the devices have expanded. Especially in the sectors such as defense, communication, space and health, there is a constant data flow and this data is frequently in the form of great sizes due to the type of the collected data and also the high velocity of arrival of the data to the systems. As a result of this expansion, the amount of data that needs to be processed and used for these studies and devices has also increased. This increase in the amount of data is an extra burden on the devices where these operations are performed. It is not possible to solve this burden on devices only with the adjustments on the software side, also the improvements made on the hardware part give a much more effective result in terms of data performance. Due to this reason, a unique DMA design is created. In this project, which is developed for high performance purpose, an Application-Specific Integrated Circuit (ASIC) Direct Memory Access (DMA) implementation that could be used in vast amounts of data processing studies, which provides high performance guarantee in data processing, is studied. For this reason, the DMA design has been combined with a processor core, a Dual-Port Random Access Memory (DPRAM) and a adder core design so that the operations on the DMA could be observed.

The DMA unit in this design is previously tested on the FPGA, and proved that it is meeting the desired qualifications. However, it is known that the speed on the FPGA is not the maximum speed that this design could reach. As a result, to be able to increase the performance of the DMA, the design codes are handed to our side by its designers, and during the project, we have carried out the ASIC implementation studies.

For implementing the DMA, the necessary tool is the Cadence, which is provided to us from our university, and for the other units connected to DMA in this overall system design, a knowledge of HDL description languages and another helpful tool Vivado is also compulsory.

The work that has been done, the problems that are encountered and the final results during the project time are reported in detail as a consequence of our study. This project we have done regarding to this unique DMA, which is a newly designed unit, is considered as a basis for creating a starting point for further studies.

ÖZEL DOĞRUDAN BELLEK ERİŞİMİ MODÜL TASARIMI VE UYGULAMASI

ÖZET

Teknolojinin gelişmesiyle birlikte yapılan çalışmaların, araştırmaların, üretilen cihazların hitap ettiği sınırlar genişlemiştir. Bu gelişmelerin özellikle yaşandığı alanlar olan savunma, haberleşme, uzay ve sağlık benzeri sektörlerde kullanılan sistemlerin sürekli olarak maruz kaldığı veri akışında, bu veriler çok büyük boyutlarda olduğu gibi bu akış çoğu zaman aralıksız olarak sisteme veri girişine sebep olmaktadır. Bu artış sonucunda ise bu sektörlerde devam eden çalışmalar ve kullanılan cihazlar için işlenmesi, kullanılması gerekli olan veri miktarındaki bu artış, çalışmaların gerçeklendiği cihazlara fazladan bir yük olmaktadır. Bu yükün cihazlarda yalnızca yazılım tarafında yapılacak olan düzenlemelerle çözülmesi mümkün olmadığı gibi, donanım tarafında yapılan iyileştirmelerin, veri performansı konusunda çok daha etkili bir sonuç verdiği görülmüştür. Bu amaç doğrultusunda geliştirilen bu projede, bu çalışmalarda kullanılabilecek, veri işleme konusunda yüksek performans garantisi sunan bir Uygulamaya Özel Tümleşik Devre (ASIC) Doğrudan Bellek Erişim Modülü (DMA) implementasyonu üzerine çalışılmıştır. Bu nedenle sahip olunan DMA tasarımı, DMA üzerinde gerçekleşen işlemlerin gözlenebilmesi için bir işlemci çekirdeği, bir çift portlu bellek ve bir toplayıcı çekirdek tasarımı ile birleştirilmiştir.

Proje süresince kullandığımız özgün DMA modülü, öncelikli olarak tasarımcıları tarafından FPGA üzerinde test edilmiş, tasarımın beklenilen kriterlere uygun olduğu görülmüş ancak bir FPGA üzerinde olmasındansa ASIC olacak şekilde implementasyonu gerçekleştirildiğinde modülün hızını çok daha fazla arttıracağı düşüncesiyle ASIC olarak gerçeklemenin başlayabilmesi amacıyla modulün kodları tarafımıza teslim edilmiştir. Proje süresince sürdürülen bütün çalışma bizlere tasarımcıları tarafından teslim edilen modül kod ve dökümantasyonları üzerine inşa edilmiştir.

Çalışmanın önceliği DMA’in ASIC olarak gerçeklenmesi ve bu gerçeklenmenin üzerine DMA’in içerisinde bir alt sistem olarak yer alacağı daha büyük bir tam sistemde de istenilen kriterleri sağlaması, düzgün bir şekilde çalıştığının gözlenmesi olmuştur. Bu sebeple de gerek görülen bu daha büyük sistem bizler tarafından bir toplayıcı çekirdeği, bir çift portlu bellek ve bir işlemci çekirdeği ile bütün bu sistemi ASIC olacak şekilde gerçeklemek ve DMA performansını raporlamak olmuştur.

DMA'i ve dahil olduğu bu sistemi ASIC olarak gerçeklemek için gerekli program, üniversitemiz tarafından bize sağlanmış olan Cadence'dır ve bu bütün sistemin tasarımında DMA'e bağlı diğer birimler için Donanım Tanımlama Dilleri (HDL) bilgisi ve başka bir yardımcı program olarak proje süresince kullanılan Vivado da zorunludur. Proje süresince bahsi geçen bu programlarda ilk olarak alt modüller sonrasında da bütün sistemin simülasyonları yapılmış, simülasyon sonuçlarına göre ise sentez aşamasına geçilmiştir.

Proje süresince yapılan işler, karşılaşılan problemler ve sonuçlar çalışmamız neticesinde detaylı olarak raporlanmıştır. Henüz yeni tasarlanmış bir ünite olan bu özgün DMA ile ilgili yaptığımız bu proje, devamında yapılacak çalışmalar için bir başlangıç noktası oluşturması açısından bir temel olarak düşünülmüştür.

1. INTRODUCTION

High-tech systems used in areas such as defense, intelligence, health services and finance are consistently responsible for processing extreme amounts of data. Considering data operations in a system, it is primarily the processor that is considered as the responsible unit, since the processor is expected to provide the instructions, such as address, related to the data. However, in the sectors considered, since the amount of the data is tremendous, this increase in the amount of data that needs to be processed causes a crucial performance problem. Thus, in order to increase data transfer efficiency, the speed of data transfer between the peripheral units of the hardware system is of great importance. As a solution to this issue, Direct Memory Access (DMA) unit is used in the hardware systems for the effective data transfer between the peripheral units of the hardware and the system memory. In a system with a heavily busy data flow, the inclusion of the processing unit to the flow creates a major disadvantage in terms of speed, since these data operations occupy the processor excessively. The workload of the processor is fairly reduced by involving a DMA unit to the system, thus the time and effort during the data transfer process is significantly decreased by removing the processor from the dataflow. When the DMA is used in a system data transfer path, the peripheral units in the hardware provide access to the main memory directly via DMA with no dependency to the system processor.

With this project, it is aimed to implement the ASIC design of a system with a unique DMA design by using the design tool Cadence in order to increase the data flow performance of the high-tech systems used in substantial sectors. ...

1.1 General Information

In this project, this distinctive DMA design is simulated and synthesized on the tool Cadence by forming the complete system consisting of the RISC-V Hornet core, a dual-port RAM, the DMA and a newly designed adder core, resulting with the indication that the ASIC design could be implemented.

1.1.1 RISC-V Core

When the core of the system is considered, the processor/CPU also directly occurs in the minds. However, it would not be quite correct to speak of a core in the system directly as a CPU, because a core is a small processor placed in a larger CPU. This small processor functions as a brain within the system in which it is used, performing all the essential computational jobs. In this project, for these necessary computational tasks, a RISC-V core entitled "Hornet" is implemented to the overall system as the main brain.

1.1.2 Dual Port RAM

There are units called "memory" in the systems where the transferred information or the information to be transferred is stored. In these memories, the information is kept in binary form. At certain locations with different "addresses" defined for each, this binary information could be read or written to the specific memory area located at this particular address, according to the address information given to the unit as an input in the instruction. Since the address of the information to be read does not subject to any restriction, this memory unit is called "Random-Access Memory", RAM.

A fundamental RAM unit has three inputs and one output: one input for the data, one input for the address information, one input for controlling the actualizing operation on the unit (write or read), and one output for the data to be transferred. On the other hand, a dual port RAM (DPRAM) has two distinct ports for two distinct data inputs, address information, write-read controlling input and data outputs, enabling to perform two different tasks on the different ports at the same time. A DPRAM is requisite for this project, since it is significant for the overall system to transfer information from the core to the DMA, or from DMA to the core over one memory featured channel. In this case, this proper channel is the DPRAM, one port for the core and the other port for the DMA.

1.1.3. Adder Core

Since it is built with a distinctive approach, one of the main purposes in this project is to test whether the DMA design is working properly in the way it is desired or not. This testing process requires a core, or as it is called in this project, a processing element connected to the DMA, which could make it possible to observe whether all the other elements in the system is working properly or not. In this project, this process element is chosen as an adder core, which simply sums two numbers which are delivered to the adder core via the DMA.

1.2 Literature Review

As it is stated in the previous introduction section, electronic devices which contain high amount information transaction requires read and write access constantly to the random memory. In order to reduce time consuming in data transactions between I/O ports (or some other part of the hardware) and memory, DMA is being used in many System on Chip (SoC) implementations. Through this information, to be able to implement the DMA, as a first step of this project a detailed research related to the various DMA designs is resulted. Although many studies have been conducted on the subject in recent years, not many studies have been found directly related to this thesis subject. However, articles of similar studies found as *A Low-Area Direct Memory Access Controller Architecture for a RISC-V Based Low-Power Microcontroller[1], Design and implementation of Efficient Direct Memory Access (DMA) Controller in Multiprocessor SoC[2], Direct Memory Access Remapping for Thunderbolt, Feature Deployment at Platform Level[3].* These articles are read in great detail and taken in the account throughout the continuing working process.

Furthermore, in order to be able to understand the work flow of the Hornet RISC-V Core, the initial step has been seeking out the materials and textbooks related to the RISC-V. The known main source used in the Hornet design is *Computer Organization and Design by Patterson & Hennessy[4].* According to that information, this book has been read for the necessary knowledge and instructions, since the textbook explains design in a processor with examples.

2. IMPLEMENTATION AND TESTING OF THE HORNET RISC-V CORE

2.1 The Hornet Core

Hornet core, which is one of the main units of the overall system as the processor core, is designed as senior design project by Yavuz Selim TOZLU and Yasin YILMAZ in year 2021. Since Hornet is also implemented in ASIC domain, it is chosen as the core in this project and before connecting the core to the system, the first stage is to check whether the core is working accurately or not.

2.1.1 The RISC-V Instruction Set Architecture (ISA)

The Reduced Instruction Set Computer - V (RISC-V) is an open source instruction set architecture. RISC-V succeeds in distinguishing itself from other processors due to the privileged features it provides, as the main object of the RISC-V being decreasing the intricacy of the operations which are performed by the hardware. A few of these significant features can be listed as being an open source architecture which does not cause any patent problems, having a wide range of compatible microarchitectures, and providing an adaptable use.

2.2 Environment Set Up

Prior to start simulating the Hornet in Cadence, the first environment to simulate the Hornet is chosen as Ubuntu, since it is an open source Linux distribution and uploading the necessary simulation tools to the Ubuntu is a smooth process. By installing Oracle's VirtualBox, which allows to extend the computer to be able to run more than one operating systems, Ubuntu 18.04 LTS is set to the computer as the operating system. The following step is to install the RISC-V GNU Toolchain to be able to create the required simulation files during the test. For the previous simulation of the Hornet core performed by Yavuz Selim TOZLU and Yasin YILMAZ, Verilator was used since it does not require any payments and it is an open-source program that helps simulating the hardware desings by generating a C++ code form of the given module. Due to this reason, before moving to Cadence in the project, Verilator is used for code genaration. As the next step, GTKWave, which is also a free and open-source program that allows the users to be able to see simulation waveforms, is installed. All these environment set up is done by following the article of the Hornet, which is DESIGN AND IMPLEMENTATION OF A 32-BIT RISC-V CORE[5].

2.3 Realization of RISC-V Hornet Core

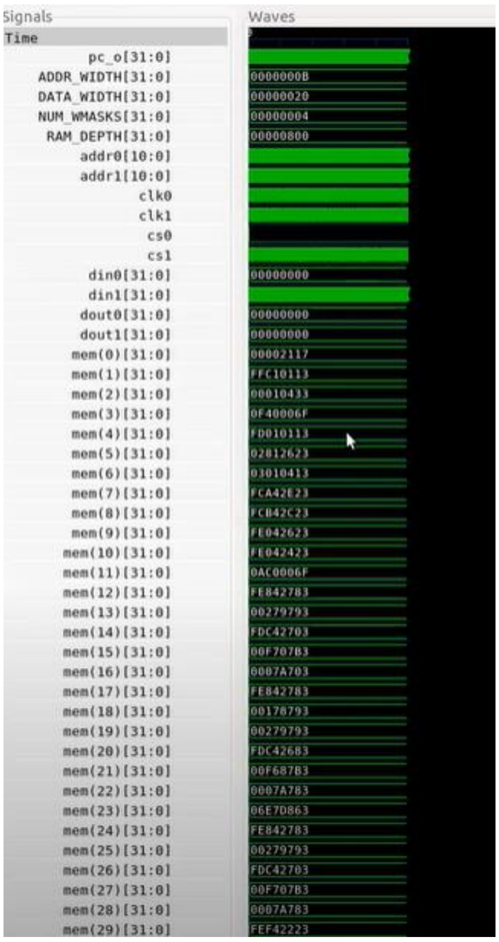
2.3.1 Preparing the Necessary Files

In order to be able to run the C codes on the Hornet core, the work explained in detail in the Hornet core article needs to be repeated. For the test of the core, the exact same steps and the bubble sort C code provided by Yavuz Selim Tozlu and Yasin Yılmaz is used also for this project. After the executions, all the files are prepared for simulating with Verilator.

2.3.2 Simulation on Verilator

For this section, Hornet's article is also used, and the commands as Yavuz Selim TOZLU and Yasin YILMAZ declared in their thesis.

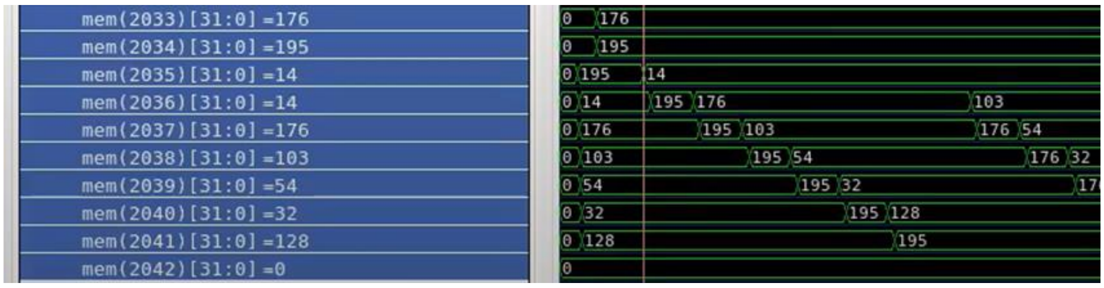
It can be seen that the program counter works properly and some of the memory addresses does not change since they store instructions in the Figure 2.1 below.



```
Signals                 Waves
Time
pc_o[31:0]
ADDR_WIDTH[31:0]        0000000B
DATA_WIDTH[31:0]        00000020
NUM_WMASKS[31:0]        00000004
RAM_DEPTH[31:0]         00000800
addr0[10:0]
addr1[10:0]
clk0
clk1
cs0
cs1
din0[31:0]              00000000
din1[31:0]
dout0[31:0]             00000000
dout1[31:0]             00000000
mem(0)[31:0]            00002117
mem(1)[31:0]            FFC10113
mem(2)[31:0]            00010433
mem(3)[31:0]            0F40006F
mem(4)[31:0]            FD010113
mem(5)[31:0]            02812623
mem(6)[31:0]            03010413
mem(7)[31:0]            FCA42E23
mem(8)[31:0]            FCB42C23
mem(9)[31:0]            FE042623
mem(10)[31:0]           FE042423
mem(11)[31:0]           0AC0006F
mem(12)[31:0]           FE842783
mem(13)[31:0]           00279793
mem(14)[31:0]           FDC42703
mem(15)[31:0]           00F707B3
mem(16)[31:0]           0007A703
mem(17)[31:0]           FE842783
mem(18)[31:0]           00178793
mem(19)[31:0]           00279793
mem(20)[31:0]           FDC42683
mem(21)[31:0]           00F687B3
mem(22)[31:0]           0007A783
mem(23)[31:0]           06E7D863
mem(24)[31:0]           FE842783
mem(25)[31:0]           00279793
mem(26)[31:0]           FDC42703
mem(27)[31:0]           00F707B3
mem(28)[31:0]           0007A783
mem(29)[31:0]           FEF42223
```

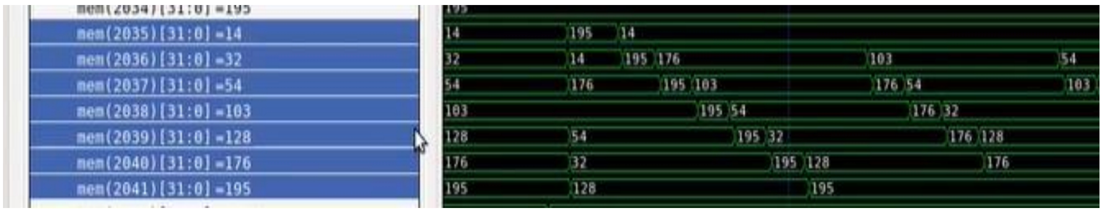
Figure 2. 1: Bubble Sort Simulation 1

In order to be able to understand whether the bubble sort algorithm works accurately on the implementation, the memory addresses having changes are examined.



```
mem(2033)[31:0] =176    0  176
mem(2034)[31:0] =195    0  195
mem(2035)[31:0] =14     0 195  14
mem(2036)[31:0] =14     0 14  195 176  103
mem(2037)[31:0] =176    0 176  195 103  176 54
mem(2038)[31:0] =103    0 103  195 54  176 32
mem(2039)[31:0] =54     0 54  195 32  17
mem(2040)[31:0] =32     0 32  195 128
mem(2041)[31:0] =128    0 128  195
mem(2042)[31:0] =0      0
```

Figure 2. 2: Bubble Sort Simulation 2



```
mem(2034)[31:0] =195    195
mem(2035)[31:0] =14     14  195 14
mem(2036)[31:0] =32     32  14 195 176  103  54
mem(2037)[31:0] =54     54  176  195 103  176 54  103
mem(2038)[31:0] =103    103  195 54  176 32
mem(2039)[31:0] =128    128  54  195 32  176 128
mem(2040)[31:0] =176    176  32  195 128  176
mem(2041)[31:0] =195    195  128  195
```

Figure 2. 3: Bubble Sort Simulation 3

It could be seen that [195, 14, 176, 103, 54, 32, 128] given as main array and the algorithm is able to sort this array as [14,32,54,103,176,195]. As a result, the way a processor core works and how to run a C program on this Hornet core can be understood by following the steps of the above work flow. Subsequently, the core is proven to be ready to run any other C programs on itself.

2.3.3 Bubble Sort Algorithm

To be able to test whether the core is operating correctly or not, a basic sorting algorithm, bubble sort, is simulated. This algorithm is one of the sorting algorithms developed to keep the data in order in memory which is based on comparing each element with the adjacent element on the given array. To explain in a bit more detail, first when an array with n elements is considered, maximum n steps will be required to complete this sorting operation. In the first step, the first element of the array (on the left) is compared with the next second element in the given array. In this comparison, if the element on the left is greater than the second element, these two elements are swapped; so that the greater element stays at the right when the smaller one stays on the left. Then in the new array, the greater element for the previous step becomes the second element of the array and it is compared to the third element. Again, the greater one passes to the right and becomes the third element. In the continuation, the third element is compared to the fourth element. This process continues until the last element of the given array is reached. At the end, the greatest term of the given sequence is placed to the far right and thus, the first step of the operation is concluded.

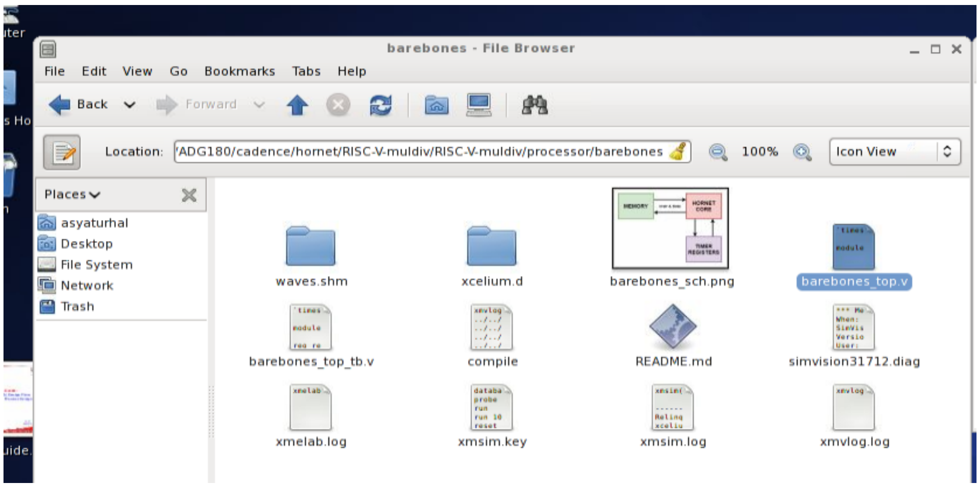
In the second step, since the greatest right-most term is removed in the first step, the same transactions done in the first step is applied to the remaining subarray. This operation, again, selects the greatest element of the subarray as in the previous step and places it at the right end. When the whole array is considered, it can be seen that the two greatest terms of the given array are placed at the far right, sorted among themselves. The order is in descending order from right to left.

In the third step, two greatest elements which are placed at the rightmost in the array are ejected and the same operation is applied to the remaining elements on the array. Thus, the greatest element of the subarray is detected and placed at the right end of the subsequence. At the end of this step, in descending order from right to left, the three greatest elements of the given array will be placed at the far right. Since the array has n elements, the transactions explained above continues step by step for each element in the remaining subarrays until the elements of the array are sorted from right to left in a descending order, which is equivalent to an array being in an ascending order from left to the right. This completes the sorting of the elements in an array with the bubble sort algorithm. However, the beginning sorting of the given array is crucially important for the time efficiency of the algorithm. Since the algorithm has two loops: one for comparison and the second for swapping, the better way would be having a reasonably sorted array to reduce the number of the loops. Due to this reason, it can be said that for a large dataset which is not sorted fairly, bubble sorting would not be efficient.

2.3.4 Simulation on Cadence

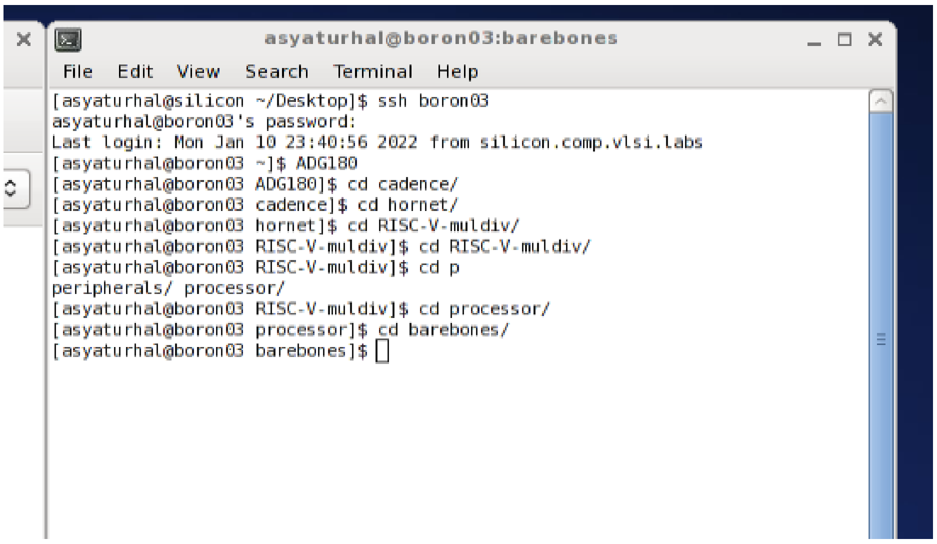
For this part of the project, it is a necessity to refer to Hornet's article and the given work flow is followed.

Hornet's files are copied to the Cadence accounts, then over the terminal, the location of the files are reached.



Location: 'ADG180/cadence/hornet/RISC-V-muldiv/RISC-V-muldiv/processor/barebones

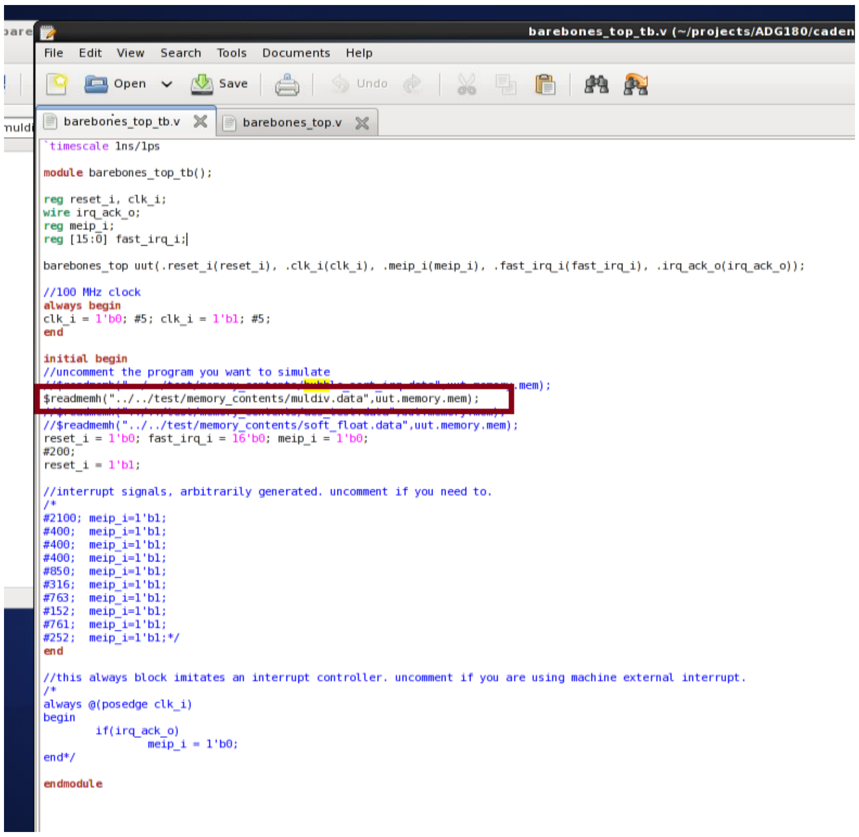
Figure 2. 4: Folder of RISC-V


asyaturhal@boron03:barebones
File Edit View Search Terminal Help

```
[asyaturhal@silicon ~/Desktop]$ ssh boron03
asyaturhal@boron03's password:
Last login: Mon Jan 10 23:40:56 2022 from silicon.comp.vlsi.labs
[asyaturhal@boron03 ~]$ ADG180
[asyaturhal@boron03 ADG180]$ cd cadence/
[asyaturhal@boron03 cadence]$ cd hornet/
[asyaturhal@boron03 hornet]$ cd RISC-V-muldiv/
[asyaturhal@boron03 RISC-V-muldiv]$ cd RISC-V-muldiv/
[asyaturhal@boron03 RISC-V-muldiv]$ cd p
peripherals/ processor/
[asyaturhal@boron03 RISC-V-muldiv]$ cd processor/
[asyaturhal@boron03 processor]$ cd barebones/
[asyaturhal@boron03 barebones]$ 
```

Figure 2. 5: Entering the location of the files on terminal

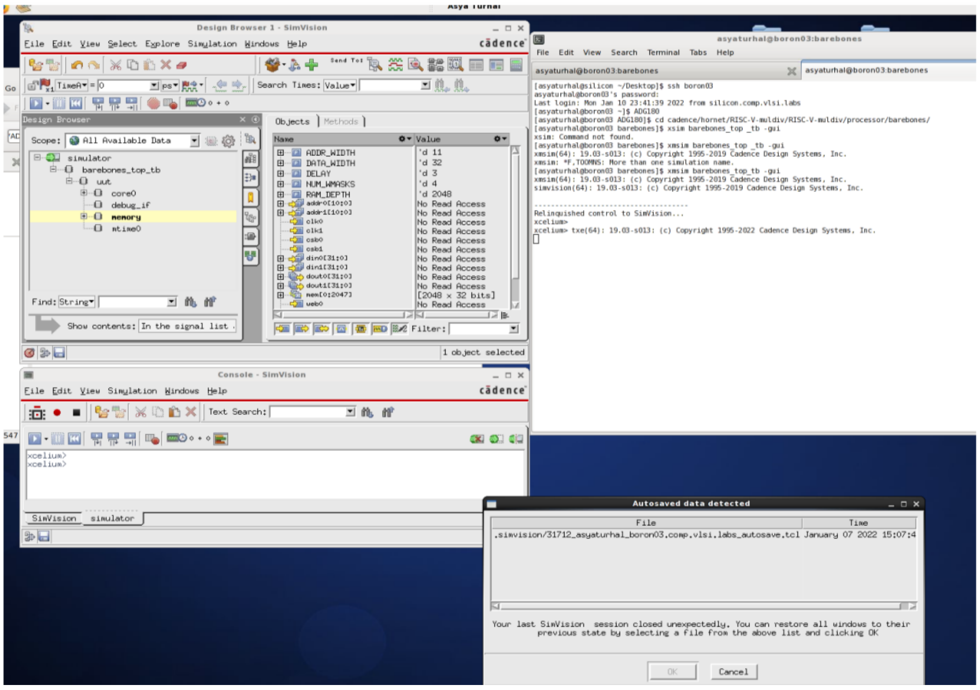
As the following step, the line which would load the bubble_sort_tb.data to the processor memory is uncommented in the given Hornet file barebones_top_tb.v, in order to simulate the bubble sort file.


barebones_top_tb.v (~/projects/ADG180/caden
File Edit View Search Tools Documents Help
Open
Save
Undo
barebones_top_tb.v
barebones_top.v

```
`timescale 1ns/1ps

module barebones_top_tb();

reg reset_i, clk_i;
wire irq_ack_o;
reg meip_i;
reg [15:0] fast_irq_i;

barebones_top uut(.reset_i(reset_i), .clk_i(clk_i), .meip_i(meip_i), .fast_irq_i(fast_irq_i), .irq_ack_o(irq_ack_o));

//100 MHz clock
always begin
clk_i = 1'b0; #5; clk_i = 1'b1; #5;
end

initial begin
//uncomment the program you want to simulate
//$readmemh("../../test/memory_contents/bubble_sort_irq.data",uut.memory.mem);
$readmemh("../../test/memory_contents/muldiv.data",uut.memory.mem);
//$readmemh("../../test/memory_contents/[illegible].data",uut.memory.mem);
//$readmemh("../../test/memory_contents/soft_float.data",uut.memory.mem);
reset_i = 1'b0; fast_irq_i = 16'b0; meip_i = 1'b0;
#200;
reset_i = 1'b1;

//interrupt signals, arbitrarily generated. uncomment if you need to.
/*
#2100; meip_i=1'b1;
#400;  meip_i=1'b1;
#400;  meip_i=1'b1;
#400;  meip_i=1'b1;
#850;  meip_i=1'b1;
#316;  meip_i=1'b1;
#763;  meip_i=1'b1;
#152;  meip_i=1'b1;
#761;  meip_i=1'b1;
#252;  meip_i=1'b1;*/
end

//this always block imitates an interrupt controller. uncomment if you are using machine external interrupt.
/*
always @(posedge clk_i)
begin
        if(irq_ack_o)
                meip_i = 1'b0;
end*/

endmodule
```

Figure 2. 6: Testbench of the Bubble Sort Algorithm

The SimVision of Cadence, where the simulations are executed is opened.



```
[asyaturhal@silicon ~/Desktop]$ ssh boron03
asyaturhal@boron03's password:
Last login: Mon Jan 10 23:41:39 2022 from silicon.comp.vlsi.labs
[asyaturhal@boron03 ~]$ ADG180
[asyaturhal@boron03 ADG180]$ cd cadence/hornet/RISC-V-muldiv/RISC-V-muldiv/processor/barebones/
[asyaturhal@boron03 barebones]$ xsim barebones_top _tb -gui
xsim: Command not found.
[asyaturhal@boron03 barebones]$ xmsim barebones_top _tb -gui
xmsim(64): 19.03-s013: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
xmsim: *F,TOOMNS: More than one simulation name.
[asyaturhal@boron03 barebones]$ xmsim barebones_top_tb -gui
xmsim(64): 19.03-s013: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
simvision(64): 19.03-s013: (c) Copyright 1995-2019 Cadence Design Systems, Inc.

-------------------------------------
Relinquished control to SimVision...
xcelium>
xcelium> txe(64): 19.03-s013: (c) Copyright 1995-2022 Cadence Design Systems, Inc.
```

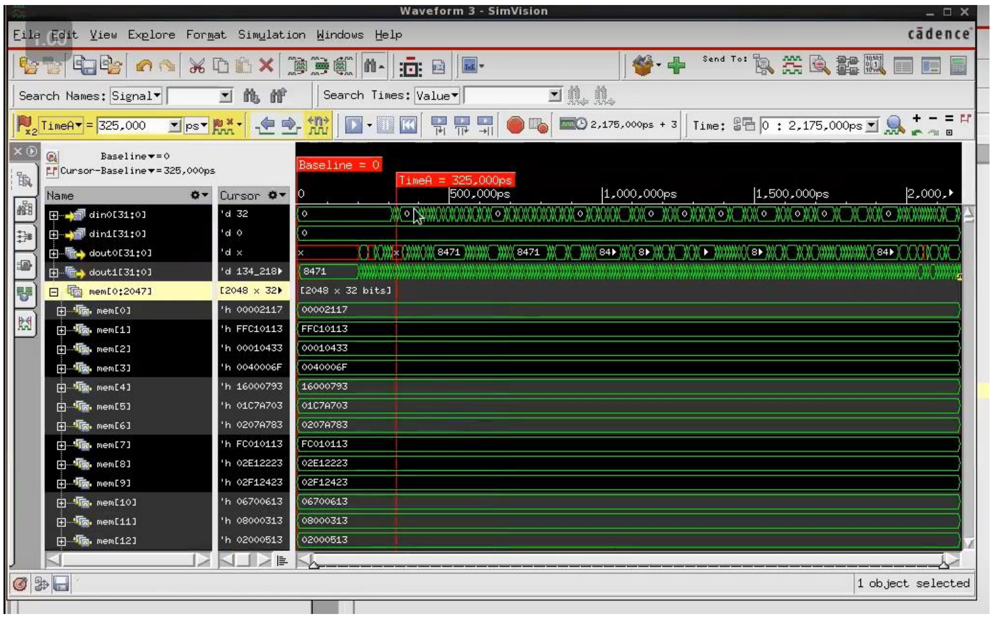
Name	Value
ADDR_WIDTH	'd 11
DATA_WIDTH	'd 32
DELAY	'd 3
NUM_WMASKS	'd 4
RAM_DEPTH	'd 2048
addr0[10:0]	No Read Access
addr1[10:0]	No Read Access
clk0	No Read Access
clk1	No Read Access
csb0	No Read Access
csb1	No Read Access
din0[31:0]	No Read Access
din1[31:0]	No Read Access
dout0[31:0]	No Read Access
dout1[31:0]	No Read Access
mem[0:2047]	[2048 x 32 bits]
web0	No Read Access

File	Time
.simvision/31712_asyaturhal_boron03.comp.vlsi.labs_autosave.tcl	January 07 2022 15:07:4

Your last SimVision session closed unexpectedly. You can restore all windows to their previous state by selecting a file from the above list and clicking OK

Figure 2. 7: SimVision

The memory addresses are sent to the waveform window in order to be able to wiev them.



Name	Cursor	
din0[31:0]	'd 32	0
din1[31:0]	'd 0	0
dout0[31:0]	'd x	x
dout1[31:0]	'd 134_218▸	8471
mem[0:2047]	[2048 x 32▸	[2048 x 32 bits]
mem[0]	'h 00002117	00002117
mem[1]	'h FFC10113	FFC10113
mem[2]	'h 00010433	00010433
mem[3]	'h 0040006F	0040006F
mem[4]	'h 16000793	16000793
mem[5]	'h 01C7A703	01C7A703
mem[6]	'h 0207A783	0207A783
mem[7]	'h FC010113	FC010113
mem[8]	'h 02E12223	02E12223
mem[9]	'h 02F12423	02F12423
mem[10]	'h 06700613	06700613
mem[11]	'h 08000313	08000313
mem[12]	'h 02000513	02000513

Figure 2. 8: Simulation waveform

As a result of this overall work, the scheduled operation so far is managed by implementing Hornet Core and run C program on the core, both using Verilator and Cadence Xcellium. Consequently, it is proved that the RISC-V Hornet core is operating accurately and it is ready to be used as the processor core unit of the overall system planned in this project.

3. OVERALL DESIGN ELEMENTS

Since this project is a TUBITAK project, there is a present DMA designed for this particular project. The codes of that DMA are adjusted to this project and connected to the Hornet core. During the beginning of the DMA implementation process, it has been realized that this overall system needs a DPRAM, and a FIFO in order to synthesize the system. However, for ASIC design, neither of these elements are free. Therefore, a research process has taken place during this period of the project. One of the best solutions considered was to implement an open source RAM and FIFO to the system, but the remaining time would not be enough for realizing both RAM and FIFO. As a result, the last decision is made as using non-synthesisable RAM and FIFO to be able to conclude the project, since the main purpose of the project is to implement the DMA; FIFO and RAM are the elements that are necessary to prove that the DMA is operating as it is desired and can be implemented as an ASIC design.

3.1 RAM and FIFO Search for Sythesizing the Design

As it is mentioned in the above section, to realize the system, a configurable RAM and FIFO with a convenient speed for the design is needed. However, in Cadence libraries that are available to use for the project, no RAM or FIFO blocks could be found. For approximately two weeks, a solution to this problem has been searched.

First of all, for the system, a dual port, configurable RAM is compulsory. To have a RAM adjustable to this project, the RAM with the given qualities can be purchased from Cadence, yet this leads a budget problem for the project. However for the FIFO, a purchase cannot be possible since there are no FIFO blocks on sale, as a result, the arising solution to this, designing a FIFO and a DPRAM in Cadence is considered. Also it is decided that this process would take approximately three months.

For the RAM problem, without any purchasing option, designing and sythesizing an Open RAM has appeared as the second solution. However the researches showed that this design process for an Open RAM would take almost the same time as a graduation project would take, and due to this reason, this option is also eliminated.

During these researches, a memory tool from Cadence: Legato is also found. Nevertheless, this tool also requires purchasing and the budget problem arised for this solution, too.

It should also be considered that this design is promising speed, the main reason that this DMA is implemented as ASIC is due to the need of high speed, which makes the case with FIFO and RAM even more difficult. The speed for a designed FIFO is estimated 1 GHz, and since there is not a complete implementation yet, the definite speed required for the design could not be decided, or the speed of the FIFO which needs to be designed from scratch would be enough or not. Also because of this speed issue, instead of a RAM, using the register blocks in the Cadence libraries are pointless in terms of speed. Nevertheless, to be able to observe the DMA work's accuracy and performance by simulating the system, it is decided to write a FIFO and DPRAM modules in Verilog.

3.2 FIFO

It is decided to use an open FIFO verilog code and modify it according to our project. This code is taken from "Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition" book and the writer of the code is Venkata Ramana Kalapatapu. This FIFO design cosists of multiple registers. This registers are controlled by a control block and this unit makes registers according to FIFO behaviours. The reason this code is chosen that it is synthesizable and customable. It is suitable to change the size of it according to needed requirements.

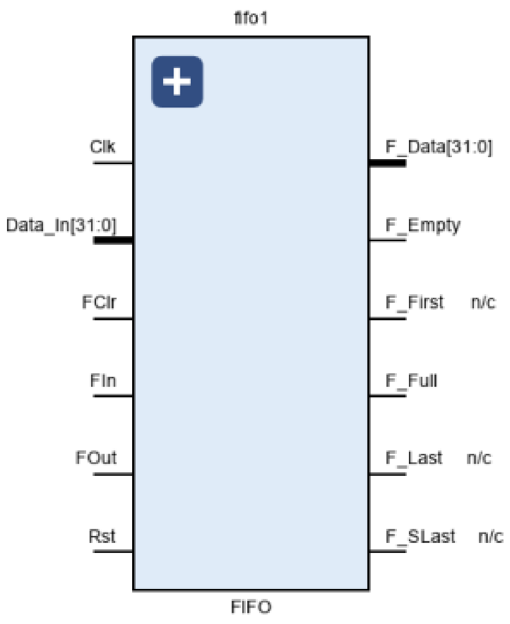


Figure 3. 1: RTL Schematic of FIFO

For this project it is needed to have a FIFO which has 32 bit data length, 4 bit data depth. In order to achieve requirements, FIFO code has been configured according to given requirements. After changing the FIFO verilog code, testbench is written in order to do behavioral simulation. It is approved that FIFO is working correctly as seen in Figure 3.2. Data is written in to FIFO as 5, 9, 25, 550 respectively and data is read from the FIFO correctly as seen in FIFODataOut port in the simulation.

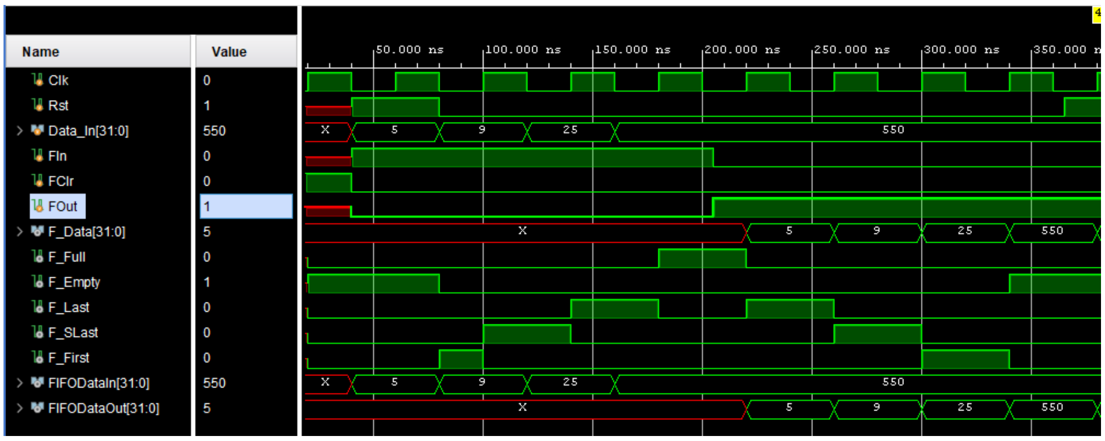


Figure 3. 2: Behavioral Simulation of FIFO

3.3 Processing Element Adder Core with FIFO

In the overall design of the DMA, there are some processing elements, cores and in order to make a simulation we designed one of them. The core we design includes an adder, an input FIFO, an output FIFO and a control unit. FIFOs have 32 bit width and 4 depth. Which means we can write 4 different numbers with 32 bits. In adder block we wrote a combinential adder but added a clock in order to maintain some delay. Control unit is an finite state machine with five different states. We designed the state diagram of the control unit as seen in Figure 3.3.

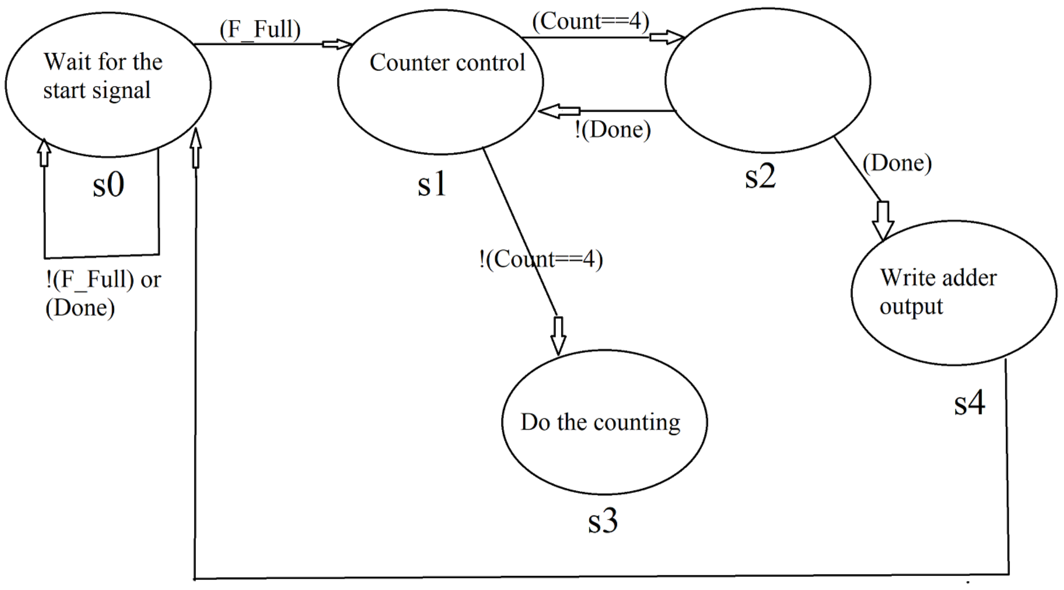


Figure 3. 3: State Diagram of the Core's Control Unit

In the s0 state, our circuit waits for the start signal in order to start reading from the input FIFO. Also "Done" signal show that all the calulations are done, so when it is high circuit stays in the s0 state. "F-full" signal shows that input FIFO is full which means we can read from it.

In s1 state we checked the counter and if it is 4, current state goes to s2 state, else current state goes to s3 state, keeps counting and gives the adder start signal.

In s2 we checked the done signal of the adder . If it is high our circuit starts to write the result to the output FIFO.

Since our FIFO has 4 different numbers, we needed to read from it four times and save the numbers to registers. So we did the counting, reading from first FIFO s3 state and writing to registers in s1 state.

When count is 4, our current state goes to s4 state and gives output of the adder to the output FIFO.

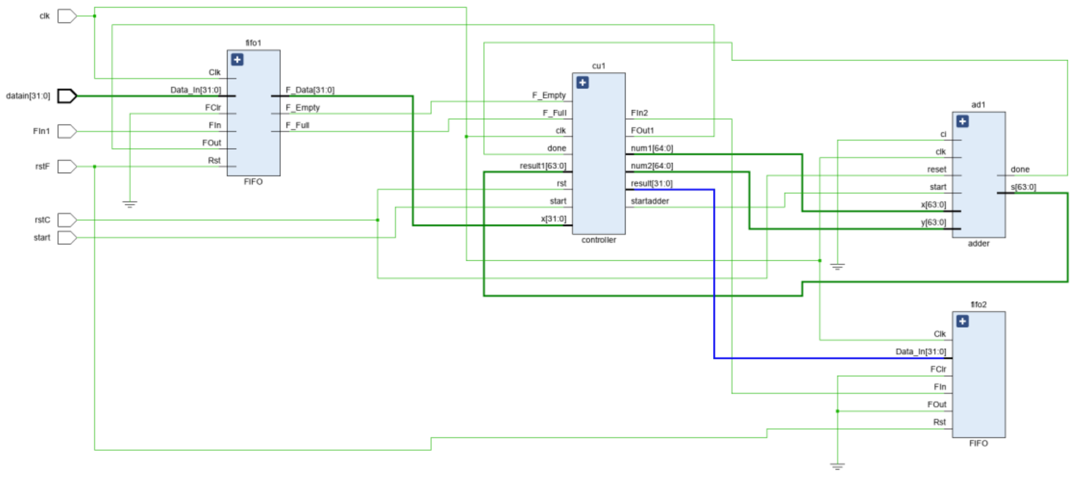


Figure 3. 4: RTL Schematic of The Adder Core

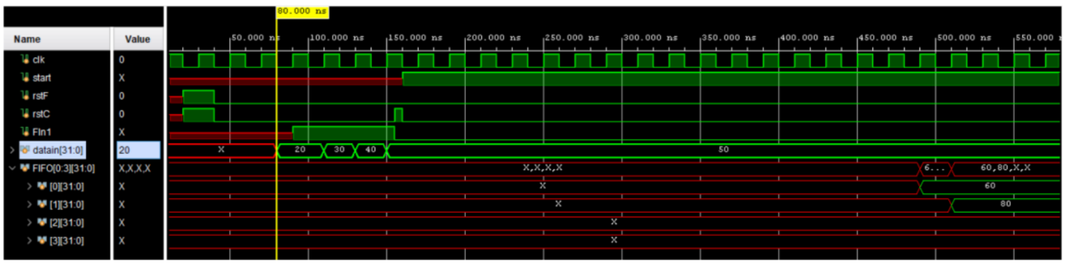


Figure 3. 5: Behavioral Simulation of The Adder Core

After finishing adder core design and simulation, we managed to replicate cores in the DMA design which we want to implement in ASIC.

3.4 Dual Port RAM (DPRAM)

3.4.1 Dual Port Ram (DPRAM) Design

As it is mentioned repeatedly, the design consists of a DMA block entitled “mem_cpy”, a processor core and a dual port RAM. This dual port RAM is crucial to the design, since the communication between the mem_cpy block and the processor is maintained by this DPRAM. This DPRAM makes it possible for the processor and the mem_cpy to execute write and read operations on the memory block at the same time.

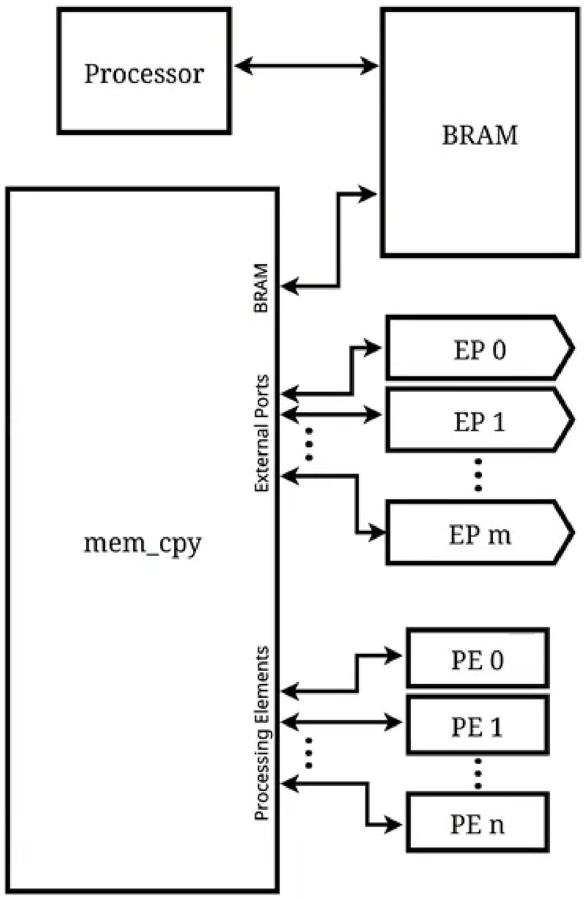


Figure 3. 6: Simple Block Schematic of the Overall Design

The research held in the previous steps revealed that an implemetation of this DPRAM cannot be realized due to lack of budget and time. Therefore, as a result of this research, it is decided to design a dual port RAM which can be used for the simulation of the system. This DPRAM is written in Verilog hardware description language and it can be synthesized on Cadence by the registers that the library available consists of.

In accordance with this purpose, a DPRAM with the width of 64 and the data length of 32 bits is designed. DPRAM has one clock and reset inputs to activate its running process and two separate data in and two separate data out ports. One of these data in and data out input-outputs are responsible for the data transfer between the processor, and the other data in and out ports are responsible for the data transfer between the mem_cpy block. DPRAM has also two separate write-read commands: one of them for the processor and the other for the mem_cpy block, and finally with two different address inputs, the design of the DPRAM is completed.

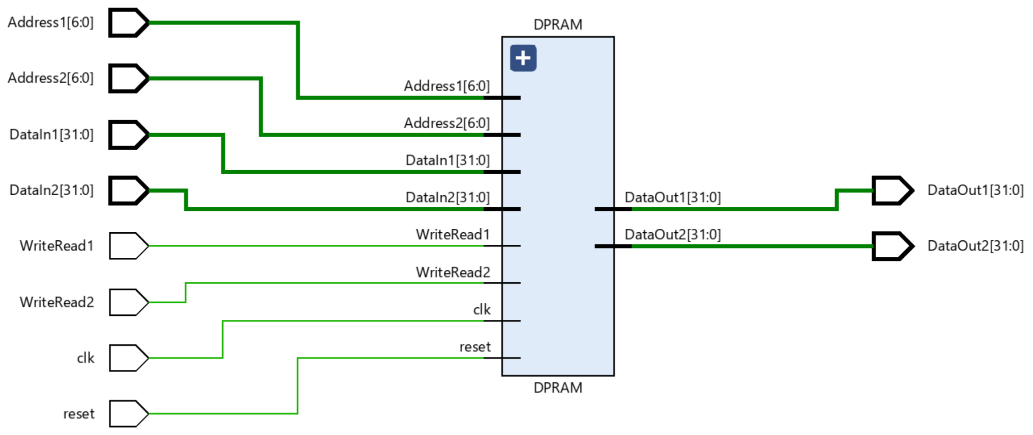


Figure 3. 7: Elaborated Design Schematic of the DPRAM

3.4.2 DPRAM Simulation

The design is simulated on both Vivado and Cadence Xcellium to test whether it is working under the desired conditions.

For the simulation in Cadence Xcellium, the necessary files are the HDL code of the design and a testbench code. When these files exists, the simulation process can be started as follows:

First of all, to be able to execute Cadence Xcellium, it is necessary to go to the location over the terminal where both HDL and testbench codes are. After getting to the right location, the command to invoke the Xcellium is,

```
xrun -access rwc -linedebug -gui tb_DPRAM.v DPRAM.v
```

When this command is given on the terminal, the Xcellium simulation tool starts on the Virtual Machine and by choosing the desired unit elements, they all can be send to waveform window as it is done.

It can be observed that the data inputs 1 and 2 are written or read according to the information on address inputs 1 and 2, depending to the WriteRead 1-2 inputs based on the reset value in the design.

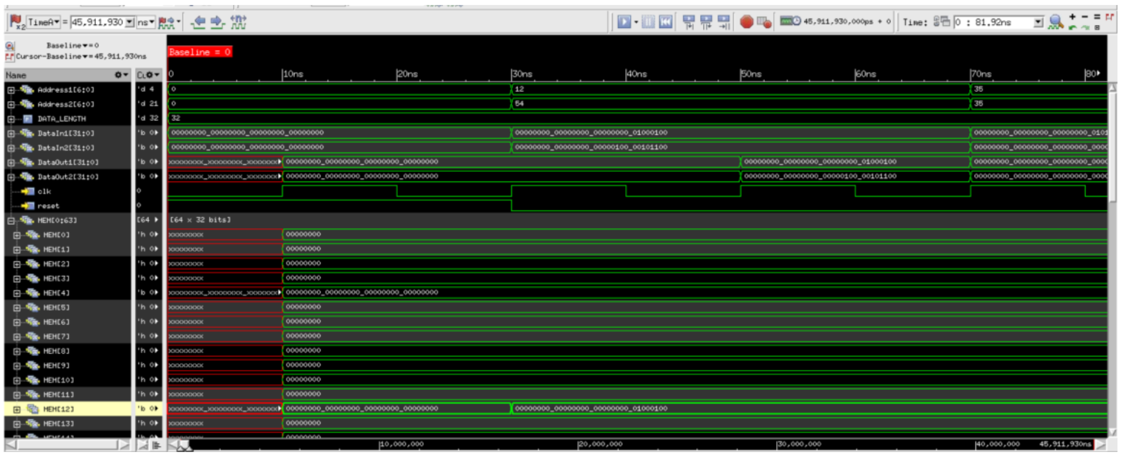


Figure 3. 8: Simulation results of DPRAM on Cadence

3.5 Direct Memory Access (DMA)

The main design of the project, the DMA, is already designed and tested on the FPGA by its own designers and the HDL codes of this DMA is delivered to this project for the ASIC implementation to have a better speed performance. However, since the unit is tested on the FPGA, the related work has been done on Vivado tool and as a result, the libraries specified for Vivado is also used. Since for the ASIC design, the tool needs to be used is Cadence, when synthesizing the mem_cpy unit, the Vivado libraries could not be used. Consequently, the RAM inside of the DMA has been altered and since as mentioned in the previous sections, there are no available RAM blocks in the Cadence libraries available, this RAM is replaced with another RAM which is just like DPRAM, generated by the registers.

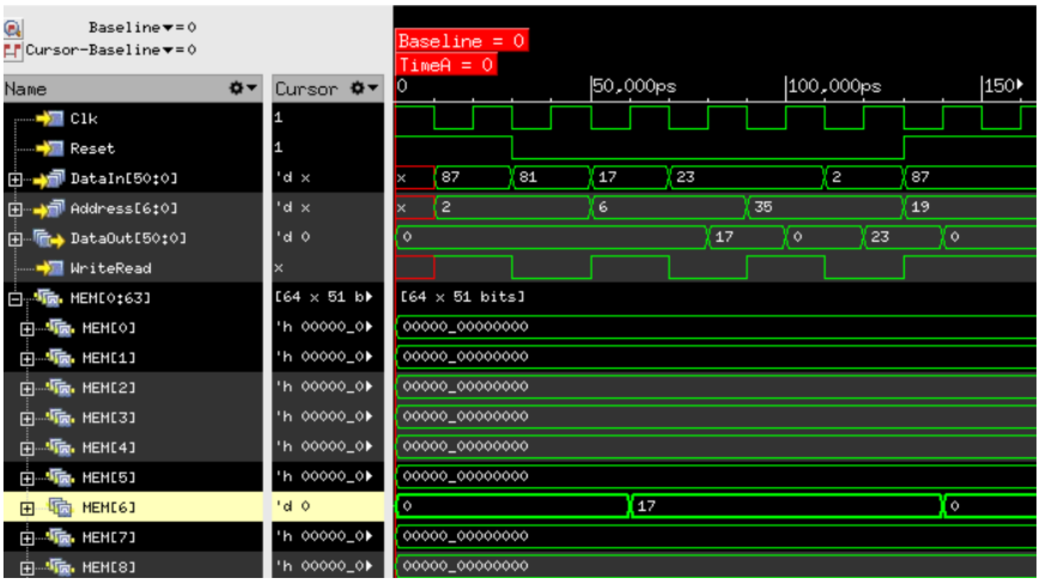
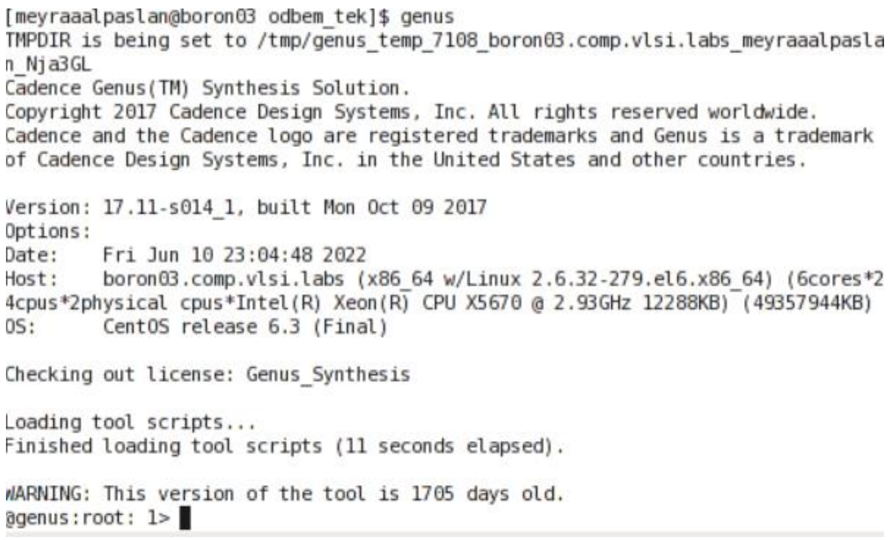


Figure 3. 9: Simulation Results of RAM of Mem_cpy on Cadence

When this RAM issue is also resolved, before moving on, the primary thing to do is synthesizing the mem_cpy block without any other connected units to check whether any other problem would occur or not.

For synthesizing the mem_cpy block, the Cadence Genus tool is used.

Similar to the simulation process, the first step of the synthesis is also going to the correct location where the HDL files exist. When it is reached to the correct location, the basic command "`genus`" on the terminal would invoke the synthesis tool.



```
[meyraaalpaslan@boron03 odbem_tek]$ genus
TMPDIR is being set to /tmp/genus_temp_7108_boron03.comp.vlsi.labs_meyraaalpasla
n_Nja3GL
Cadence Genus(TM) Synthesis Solution.
Copyright 2017 Cadence Design Systems, Inc. All rights reserved worldwide.
Cadence and the Cadence logo are registered trademarks and Genus is a trademark
of Cadence Design Systems, Inc. in the United States and other countries.

Version: 17.11-s014_1, built Mon Oct 09 2017
Options:
Date:    Fri Jun 10 23:04:48 2022
Host:    boron03.comp.vlsi.labs (x86_64 w/Linux 2.6.32-279.el6.x86_64) (6cores*2
4cpus*2physical cpus*Intel(R) Xeon(R) CPU X5670 @ 2.93GHz 12288KB) (49357944KB)
OS:      CentOS release 6.3 (Final)

Checking out license: Genus_Synthesis

Loading tool scripts...
Finished loading tool scripts (11 seconds elapsed).

WARNING: This version of the tool is 1705 days old.
@genus:root: 1> █
```

Figure 3. 10: Invoking Genus

When the terminal is switched to the Genus, the commands needs to be given are as follows:

- `set_db lib_search_path /work/kits/tsmc/lib/90lp/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tcbn90lpbwp14t_211a`
- `set_db library {tcbn90lpbwp14ttc.lib}`

These two commands set the technology library. For this project, tsmc 90nm library is used.

- `set_db hdl_vhdl_read_version 2008`

Since the DMA is written in VHDL language, it is essential to set the correct VHDL version library.

- `set_db hdl_search_path {source}`

By this command, the folder to be searched for the design codes is given. Since during this synthesis, the codes are under the folder "source" at the location where Genus is invoked, the name between the curly braces is written as source.

- `read_hdl -vhdl sync_ram.vhd`
- `read_hdl -vhdl mem_cpy.vhd`
- `read_hdl -vhdl mem_cpy_top.vhd`

With these three commands, the necessary code files for the mem_cpy block is read and checked for any kind of errors by the Genus. If there is no problem until this stage, the next step would be elaborating the design by writing the command,

- `elaborate mem_cpy_top`

If Genus elaborates the design without any error on the terminal, the elaborated design could be seen with the commmand,

- `gui_show`

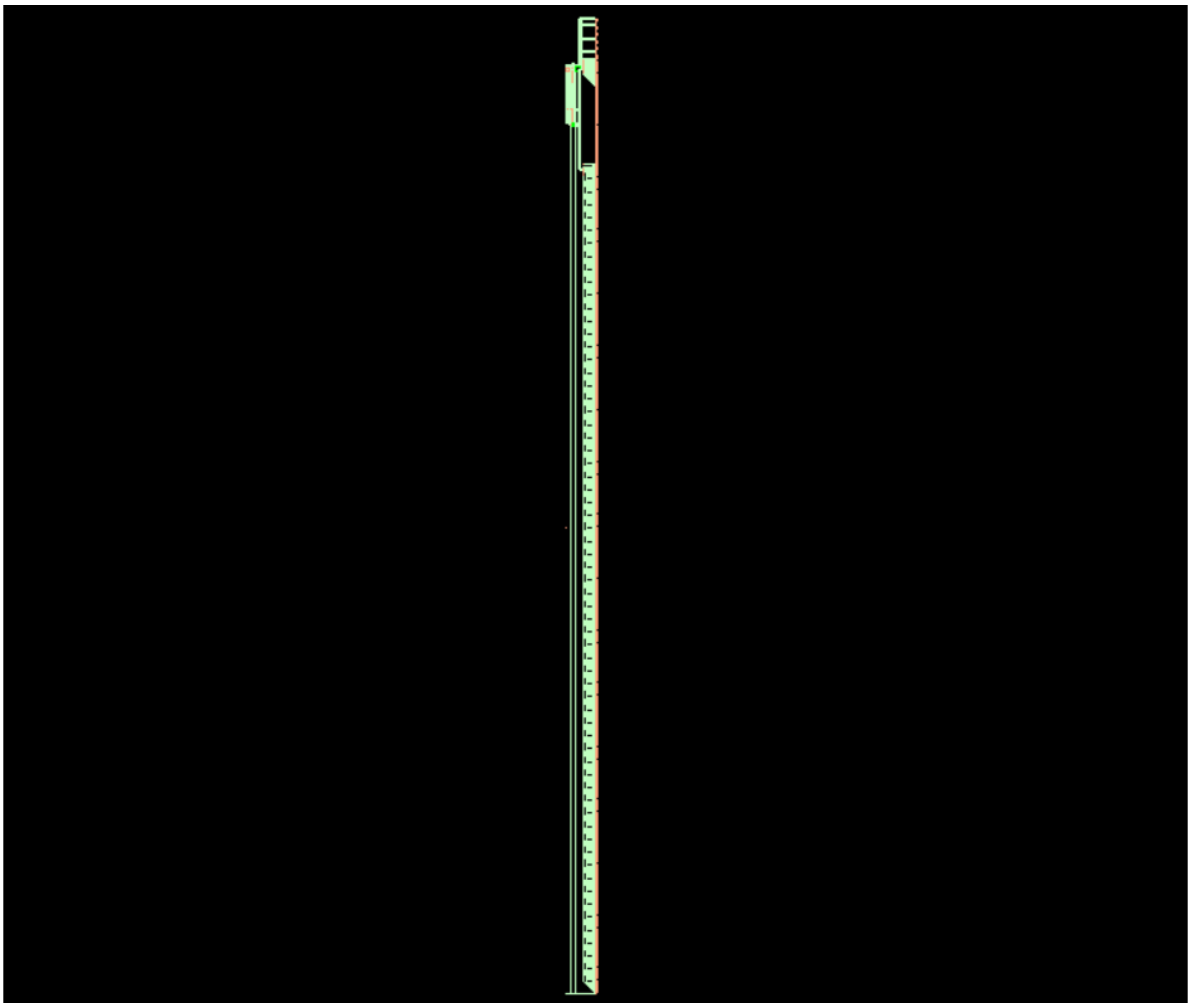


Figure 3. 11: Elaborated Mem_cpy

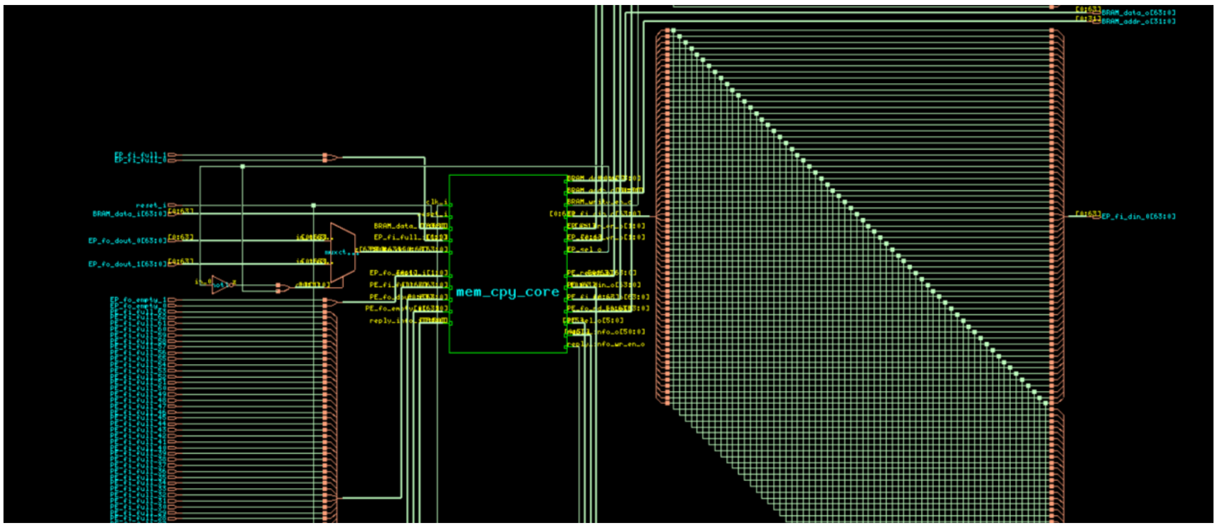


Figure 3. 12: Elaborated Mem_cpy Zoomed In for the Mem_cpy Core

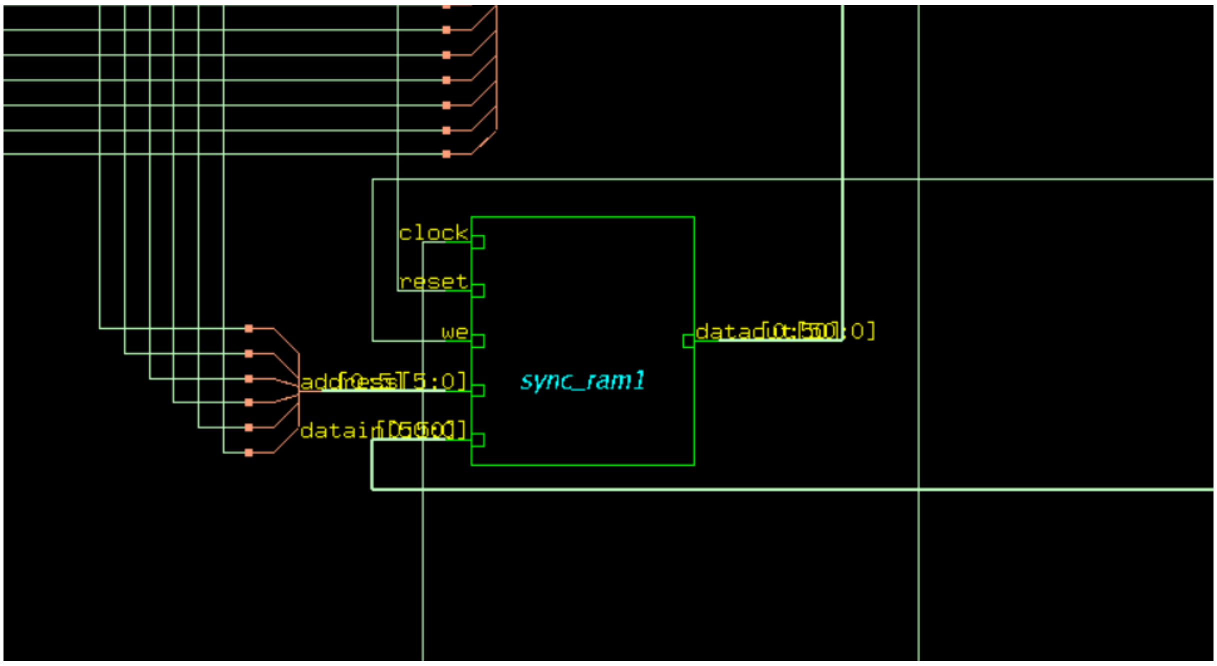


Figure 3. 13: Elaborated Mem_cpy Zoomed In for the RAM of the Block

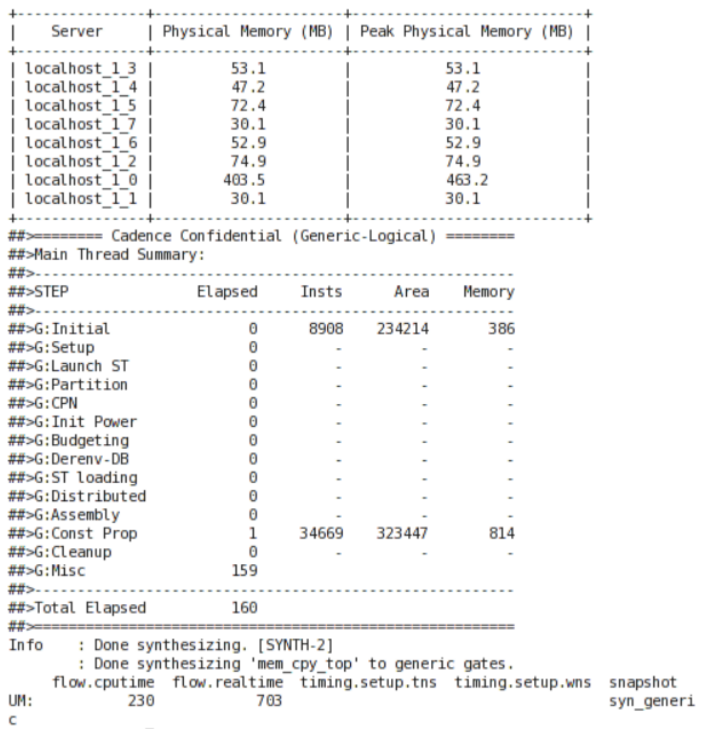
When the elaboration is also succeeded, the following step would be synthesizing the design. To be able to synthesize the design, a clock needs to be given to the system.

- `create_clock -name clk_i -period 2 -waveform {0 1} clk_i`

By this command, the clock of the system is entitled clk_i at the `-name clk_i` part of the command, a clock with a period of 2 ns is given with `-period 2` part, `-waveform {0 1}` defines the rise and fall edge times for one clock period of the clock waveform, and at the last part `clk_i` is the name of the clock port in the desing is given to the Genus.

- `syn_gen`

This command is to synthesizes the design with the generic gates included in the technology library and optimizes the RTL.

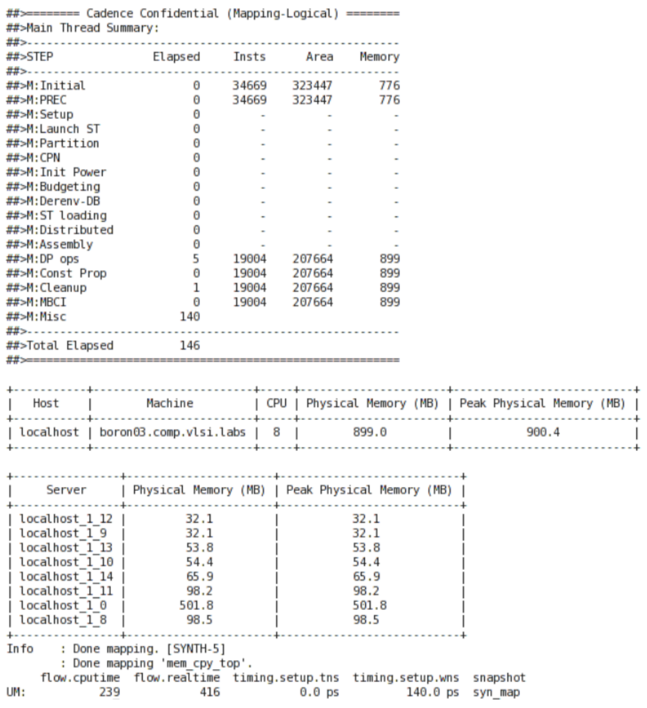


```
+---------------+----------------------+--------------------------+
|    Server     | Physical Memory (MB) | Peak Physical Memory (MB) |
+---------------+----------------------+--------------------------+
| localhost_1_3 |        53.1          |          53.1            |
| localhost_1_4 |        47.2          |          47.2            |
| localhost_1_5 |        72.4          |          72.4            |
| localhost_1_7 |        30.1          |          30.1            |
| localhost_1_6 |        52.9          |          52.9            |
| localhost_1_2 |        74.9          |          74.9            |
| localhost_1_0 |       403.5          |         463.2            |
| localhost_1_1 |        30.1          |          30.1            |
+---------------+----------------------+--------------------------+
##>======== Cadence Confidential (Generic-Logical) ========
##>Main Thread Summary:
##>-----------------------------------------------------------
##>STEP              Elapsed     Insts      Area    Memory
##>-----------------------------------------------------------
##>G:Initial               0      8908    234214       386
##>G:Setup                 0         -         -         -
##>G:Launch ST             0         -         -         -
##>G:Partition             0         -         -         -
##>G:CPN                   0         -         -         -
##>G:Init Power            0         -         -         -
##>G:Budgeting             0         -         -         -
##>G:Derenv-DB             0         -         -         -
##>G:ST loading            0         -         -         -
##>G:Distributed           0         -         -         -
##>G:Assembly              0         -         -         -
##>G:Const Prop            1     34669    323447       814
##>G:Cleanup               0         -         -         -
##>G:Misc                159
##>-----------------------------------------------------------
##>Total Elapsed         160
##>===========================================================
Info    : Done synthesizing. [SYNTH-2]
        : Done synthesizing 'mem_cpy_top' to generic gates.
     flow.cputime  flow.realtime  timing.setup.tns  timing.setup.wns  snapshot
UM:           230            703                                      syn_generi
c
```

Figure 3. 14: Generic Gate Design Report of Mem_cpy

- `syn_map`

This command is for mapping the block to the cells included in the given technology library.

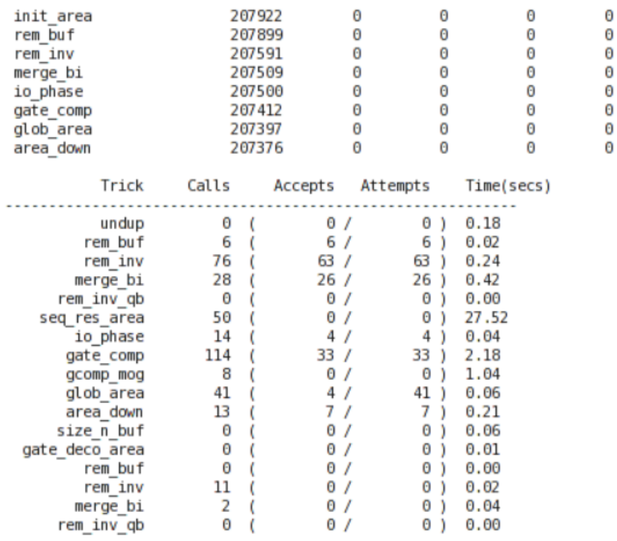


```
##>======== Cadence Confidential (Mapping-Logical) ========
##>Main Thread Summary:
##>-----------------------------------------------------------
##>STEP              Elapsed     Insts      Area    Memory
##>-----------------------------------------------------------
##>M:Initial               0     34669    323447       776
##>M:PREC                  0     34669    323447       776
##>M:Setup                 0         -         -         -
##>M:Launch ST             0         -         -         -
##>M:Partition             0         -         -         -
##>M:CPN                   0         -         -         -
##>M:Init Power            0         -         -         -
##>M:Budgeting             0         -         -         -
##>M:Derenv-DB             0         -         -         -
##>M:ST loading            0         -         -         -
##>M:Distributed           0         -         -         -
##>M:Assembly              0         -         -         -
##>M:DP ops                5     19004    207664       899
##>M:Const Prop            0     19004    207664       899
##>M:Cleanup               1     19004    207664       899
##>M:MBCI                  0     19004    207664       899
##>M:Misc                140
##>-----------------------------------------------------------
##>Total Elapsed         146
##>===========================================================

+-----------+-----------------------+-----+----------------------+---------------------------+
|   Host    |        Machine        | CPU | Physical Memory (MB) | Peak Physical Memory (MB) |
+-----------+-----------------------+-----+----------------------+---------------------------+
| localhost | boron03.comp.vlsi.labs |  8  |        899.0         |          900.4            |
+-----------+-----------------------+-----+----------------------+---------------------------+

+----------------+----------------------+---------------------------+
|     Server     | Physical Memory (MB) | Peak Physical Memory (MB) |
+----------------+----------------------+---------------------------+
| localhost_1_12 |        32.1          |          32.1             |
| localhost_1_9  |        32.1          |          32.1             |
| localhost_1_13 |        53.8          |          53.8             |
| localhost_1_10 |        54.4          |          54.4             |
| localhost_1_14 |        65.9          |          65.9             |
| localhost_1_11 |        98.2          |          98.2             |
| localhost_1_0  |       501.8          |         501.8             |
| localhost_1_8  |        98.5          |          98.5             |
+----------------+----------------------+---------------------------+
Info    : Done mapping. [SYNTH-5]
        : Done mapping 'mem_cpy_top'.
     flow.cputime  flow.realtime  timing.setup.tns  timing.setup.wns  snapshot
UM:           239            416            0.0 ps          140.0 ps  syn_map
```

Figure 3. 15: Mapping Report of Mem_cpy

- `syn_opt`

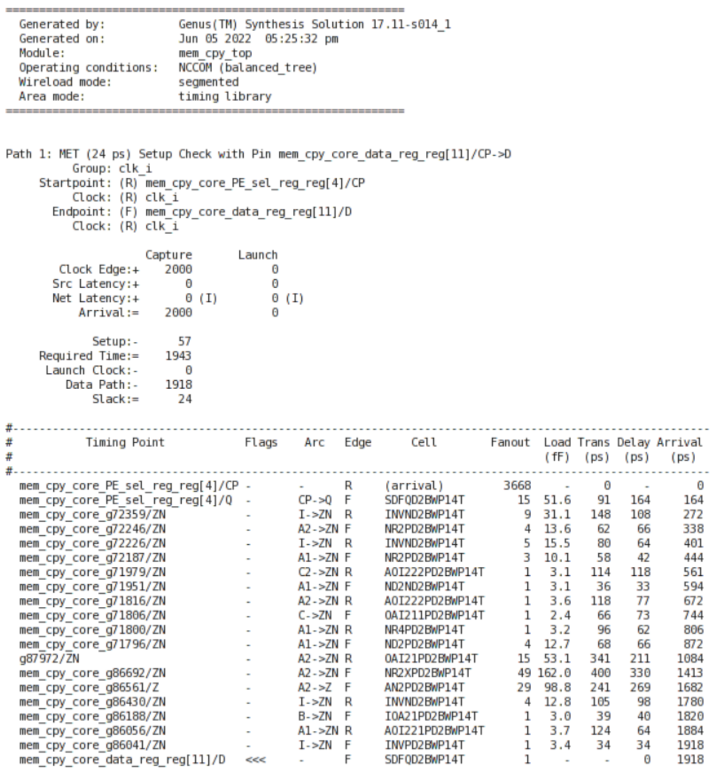
This command is for performing the optimization at gate level for enhancing the timing on crucial paths and saving area for the paths which are not crucial.



```
init_area             207922      0        0        0       0
rem_buf               207899      0        0        0       0
rem_inv               207591      0        0        0       0
merge_bi              207509      0        0        0       0
io_phase              207500      0        0        0       0
gate_comp             207412      0        0        0       0
glob_area             207397      0        0        0       0
area_down             207376      0        0        0       0

         Trick     Calls     Accepts   Attempts    Time(secs)
-----------------------------------------------------------
         undup         0  (        0 /        0 )  0.18
       rem_buf         6  (        6 /        6 )  0.02
       rem_inv        76  (       63 /       63 )  0.24
      merge_bi        28  (       26 /       26 )  0.42
    rem_inv_qb         0  (        0 /        0 )  0.00
  seq_res_area        50  (        0 /        0 )  27.52
      io_phase        14  (        4 /        4 )  0.04
     gate_comp       114  (       33 /       33 )  2.18
     gcomp_mog         8  (        0 /        0 )  1.04
     glob_area        41  (        4 /       41 )  0.06
     area_down        13  (        7 /        7 )  0.21
    size_n_buf         0  (        0 /        0 )  0.06
gate_deco_area         0  (        0 /        0 )  0.01
       rem_buf         0  (        0 /        0 )  0.00
       rem_inv        11  (        0 /        0 )  0.02
      merge_bi         2  (        0 /        0 )  0.04
    rem_inv_qb         0  (        0 /        0 )  0.00
```

Figure 3. 16: Optimization Report of Mem_cpy

- `report_timing :` Creates the timing report for the design.



```
============================================================
  Generated by:          Genus(TM) Synthesis Solution 17.11-s014_1
  Generated on:          Jun 05 2022  05:25:32 pm
  Module:                mem_cpy_top
  Operating conditions:  NCCOM (balanced_tree)
  Wireload mode:         segmented
  Area mode:             timing library
============================================================


Path 1: MET (24 ps) Setup Check with Pin mem_cpy_core_data_reg_reg[11]/CP->D
          Group: clk_i
     Startpoint: (R) mem_cpy_core_PE_sel_reg_reg[4]/CP
          Clock: (R) clk_i
       Endpoint: (F) mem_cpy_core_data_reg_reg[11]/D
          Clock: (R) clk_i

                     Capture       Launch
      Clock Edge:+    2000            0
     Src Latency:+       0            0
     Net Latency:+       0 (I)        0 (I)
         Arrival:=    2000            0

           Setup:-      57
   Required Time:=    1943
    Launch Clock:-       0
       Data Path:-    1918
           Slack:=      24

#-----------------------------------------------------------------------------------------------
#          Timing Point           Flags    Arc   Edge      Cell        Fanout  Load Trans Delay Arrival
#                                                                              (fF)  (ps)  (ps)   (ps)
#-----------------------------------------------------------------------------------------------
  mem_cpy_core_PE_sel_reg_reg[4]/CP -       -      R     (arrival)         3668    -     0     -      0
  mem_cpy_core_PE_sel_reg_reg[4]/Q  -       CP->Q  F     SDFQD2BWP14T        15  51.6    91   164    164
  mem_cpy_core_g72359/ZN            -       I->ZN  R     INVND2BWP14T         9  31.1   148   108    272
  mem_cpy_core_g72246/ZN            -       A2->ZN F     NR2PD2BWP14T         4  13.6    62    66    338
  mem_cpy_core_g72226/ZN            -       I->ZN  R     INVND2BWP14T         5  15.5    80    64    401
  mem_cpy_core_g72187/ZN            -       A1->ZN F     NR2PD2BWP14T         3  10.1    58    42    444
  mem_cpy_core_g71979/ZN            -       C2->ZN R     AOI222PD2BWP14T      1   3.1   114   118    561
  mem_cpy_core_g71951/ZN            -       A1->ZN F     ND2ND2BWP14T         1   3.1    36    33    594
  mem_cpy_core_g71816/ZN            -       A2->ZN R     AOI222PD2BWP14T      1   3.6   118    77    672
  mem_cpy_core_g71806/ZN            -       C->ZN  F     OAI211PD2BWP14T      1   2.4    66    73    744
  mem_cpy_core_g71800/ZN            -       A1->ZN R     NR4PD2BWP14T         1   3.2    96    62    806
  mem_cpy_core_g71796/ZN            -       A1->ZN F     ND2PD2BWP14T         4  12.7    68    66    872
  g87972/ZN                         -       A2->ZN R     OAI21PD2BWP14T      15  53.1   341   211   1084
  mem_cpy_core_g86692/ZN            -       A2->ZN F     NR2XPD2BWP14T       49 162.0   400   330   1413
  mem_cpy_core_g86561/Z             -       A2->Z  F     AN2PD2BWP14T        29  98.8   241   269   1682
  mem_cpy_core_g86430/ZN            -       I->ZN  R     INVND2BWP14T         4  12.8   105    98   1780
  mem_cpy_core_g86188/ZN            -       B->ZN  F     IOA21PD2BWP14T       1   3.0    39    40   1820
  mem_cpy_core_g86056/ZN            -       A1->ZN R     AOI221PD2BWP14T      1   3.7   124    64   1884
  mem_cpy_core_g86041/ZN            -       I->ZN  F     INVPD2BWP14T         1   3.4    34    34   1918
  mem_cpy_core_data_reg_reg[11]/D   <<<     -      F     SDFQD2BWP14T         1    -     -     0   1918
```

Figure 3. 17: Timing Report of Mem_cpy

These reports are included in this report to create a starting point for the future works related this DMA.

4. OVERALL DIRECT MEMORY ACCESS (DMA) SYSTEM

Overall system includes HORNET RISC-V Core, one dual port ram for instruction memory, one dual port ram for data memory and programming element. These module are connected together as seen in Figure 4.1.

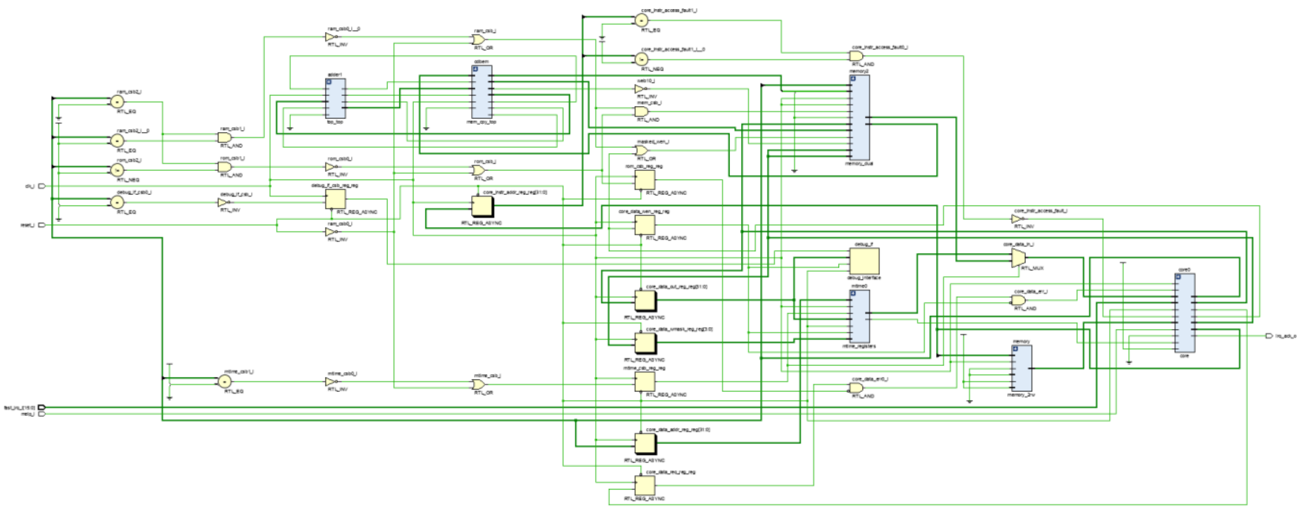
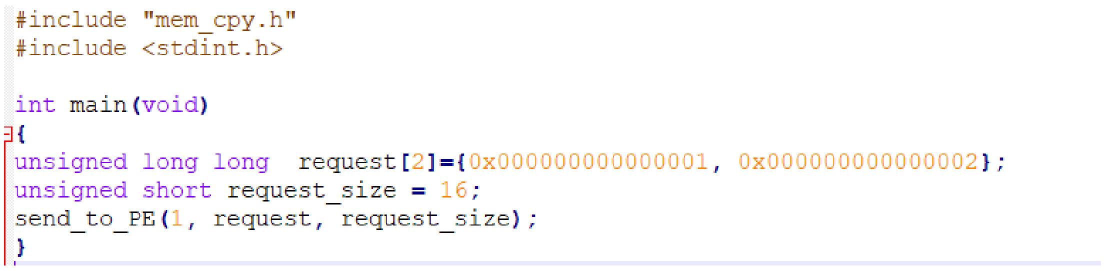


Figure 4. 1: Overall System

4.1 Overall System Simulation

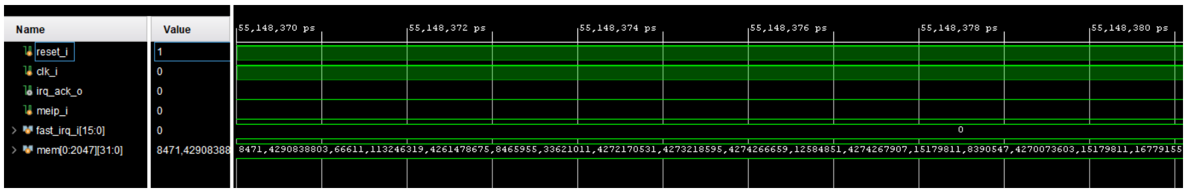
In order to test the system, a C code is written as seen below.



```
#include "mem_cpy.h"
#include <stdint.h>

int main(void)
{
unsigned long long  request[2]={0x0000000000000001, 0x0000000000000002};
unsigned short request_size = 16;
send_to_PE(1, request, request_size);
}
```

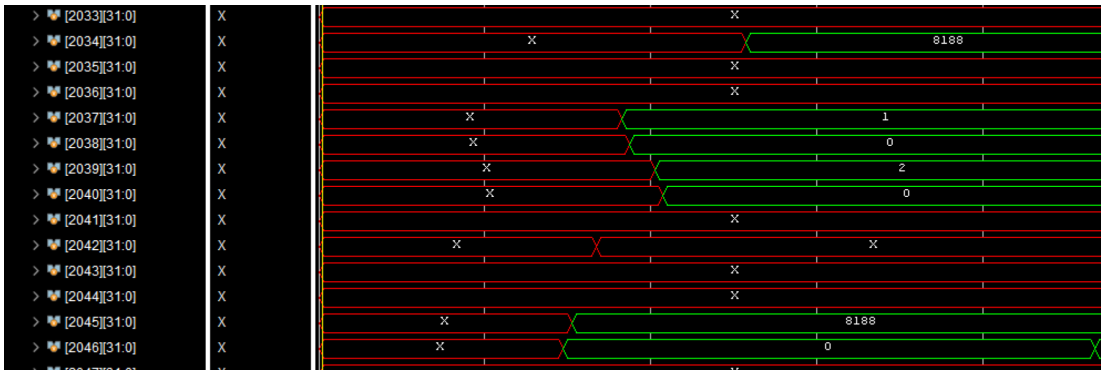
Figure 4. 2: C Code for the Simulation

In this code, send_to_PE command, reads the data in the request array from the data memory and sends it to programming element. This code has been compiled by GCC Compiler to be compliant with RISC-V processor, it is loaded to instruction memory. Then the simulation is ran. It is seen from the instruction memory that, C code loading has been succesfull. It is also seen that data memory works but since the addressing of the processor has its own algorithm, control address for DMA could not be given correctly.



```
reset_i              1
clk_i                0
irq_ack_o            0
meip_i               0
fast_irq_i[15:0]     0
mem[0:2047][31:0]    8471,42908388   8471,4290838803,66611,113246319,4261478675,8465955,33621011,4272170531,4273218595,4274266659,12584851,4274267907,15179811,8390547,4270073603,15179811,16779155
```

Figure 4. 3: RISC-V Core Instruction Memory Simulation



```
[2033][31:0]   X   X
[2034][31:0]   X   X  8188
[2035][31:0]   X   X
[2036][31:0]   X   X
[2037][31:0]   X   X  1
[2038][31:0]   X   X  0
[2039][31:0]   X   X  2
[2040][31:0]   X   X  0
[2041][31:0]   X   X
[2042][31:0]   X   X  X
[2043][31:0]   X   X
[2044][31:0]   X   X
[2045][31:0]   X   X  8188
[2046][31:0]   X   X  0
```

Figure 4. 4: Data Memory Simulation

In this simulation, it is seen that DMA works but RISC-V core has an algorithm that changes the data memory adress. Since it is crucial to have the same address for control in order to DMA to work correctly, DMA could not read the control adress. Since all the system elements have been tested seperately, there is no reason for this system to not to work after adressing core problem is solved.

4.2 Overall System Synthesis

As the final step of the project, since with the previous work it could be seen that the DMA is working and the only problem is the address problem which is related only to the processor, not to the DMA, the overall system is synthesized on Genus by Cadence. To be able to start synthesis, as with synthesizing only the DMA block mem_cpy, it is necessary to be at the accurate location on the terminal where the files exist for invoking Genus. The commands are exactly the same and the only differences are the names of the HDL files and since for a more organized folder arrangement the processor core is seperated to its own units, at the commands section there are commands which changes the folders to look for the HDL files.

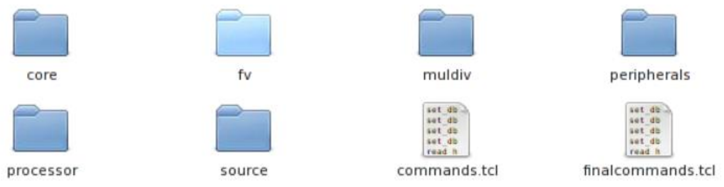


Figure 4. 5: The Files for the Overall System Synthesis

When Genus is invoked, the commands to write are as follows,

- `set_db lib_search_path /vlsi/kits/xfab/xkit/xh018/diglibs/D_CELLS_HD/v3_0/liberty_LPMOS/v3_0_0/PVT_1_80V_range`

Differing from the previous mem_cpy synthesis, for this operation the technology library is changed to another library, as the path of this library is given in this command. It should be acknowledged that this change with the library is not related to the DMA, the tsmc 90nm technology library is capable of synthesizing the DMA. This change is done only because the adder core had some instances that were not covered in the tsmc 90nm library.

- `set_db library { D_CELLS_HD_LPMOS_fast_1_98V_125C.lib}`

The name of the new technology library is given to the Genus by this command. The remaining commands are the same as mem_cpy synthesis commands, only the name of the code files and the name of the folders for the codes to be looked for are changed.

- `set_db hdl_search_path {source}`
- `set_db hdl_vhdl_read_version 2008`

- `read_hdl -vhdl sync_ram.vhd`
- `read_hdl -vhdl mem_cpy.vhd`
- `read_hdl -vhdl mem_cpy_top.vhd`
- `set_db hdl_search_path {core}`
- `read_hdl ALU.v`
- `read_hdl control_unit.v`
- `read_hdl core.v`
- `read_hdl csr_unit.v`
- `read_hdl forwarding_unit.v`
- `read_hdl hazard_detection_unit.v`
- `read_hdl imm_decoder.v`
- `read_hdl load_store_unit.v`
- `set_db hdl_search_path {muldiv}`
- `read_hdl divider_32.v`
- `read_hdl MULDIV_ctrl.v`
- `read_hdl MULDIV_in.v`
- `read_hdl MUL_DIV_out.v`
- `read_hdl multiplier_32.v`
- `read_hdl MULDIV_top.v`
- `set_db hdl_search_path {peripherals}`
- `read_hdl debug_interface.v`
- `read_hdl loader.v`
- `read_hdl memory_2rw.v`

- `read_hdl mtime_registers.v`
- `read_hdl uart.v`
- `set_db hdl_search_path {source}`
- `read_hdl FIFO_MEM_BLK.v`
- `read_hdl FIFO.v`
- `read_hdl adder.v`
- `read_hdl adder_top.v`
- `read_hdl memory_dual.v`
- `read_hdl top_top.v`
- `set_db hdl_search_path {processor}`
- `read_hdl barebones_top.v`
- `elaborate barebones_top`

If there is no error until this stage, it means that the elaboration is successfully done. The command to see the elaborated design is,

- `gui_show`

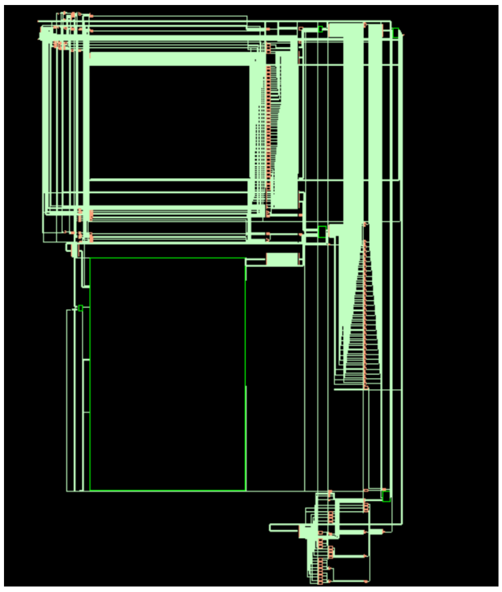


Figure 4. 6: Elaborated Overall System

Since the overall system is extremely large, by zooming in, the mem_cpy block entitled as odbem, DPRAM, processor core and the adder core could be observed.

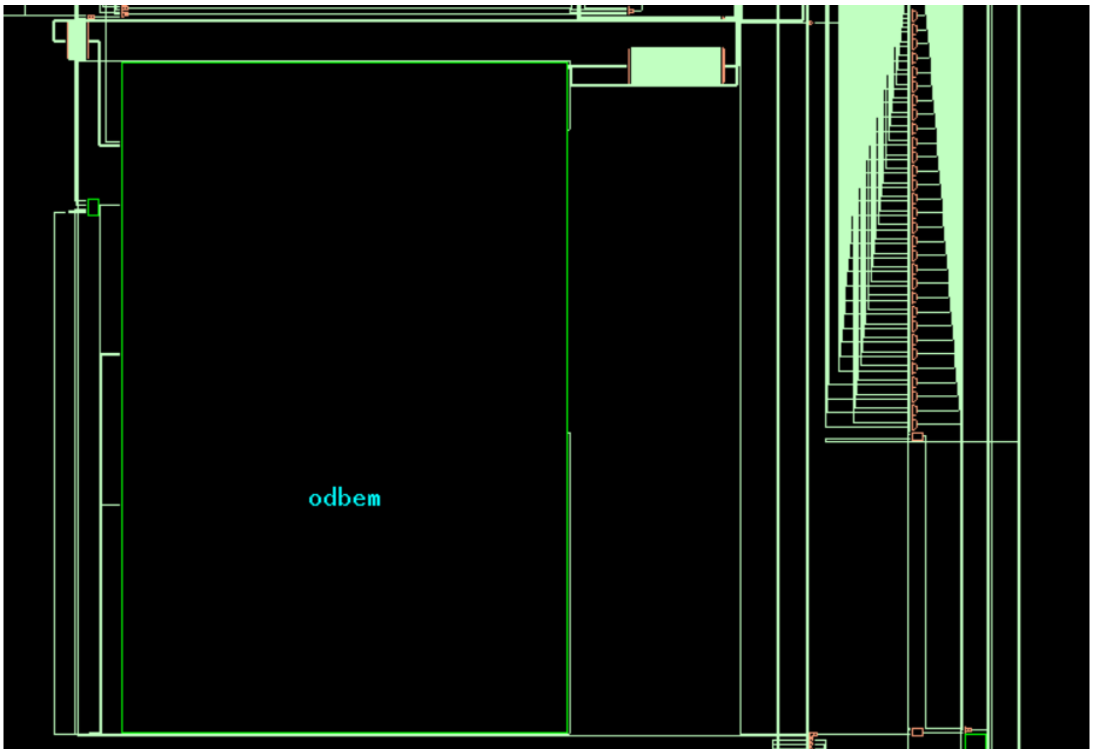


Figure 4. 7: Mem_cpy Block in the Overall System

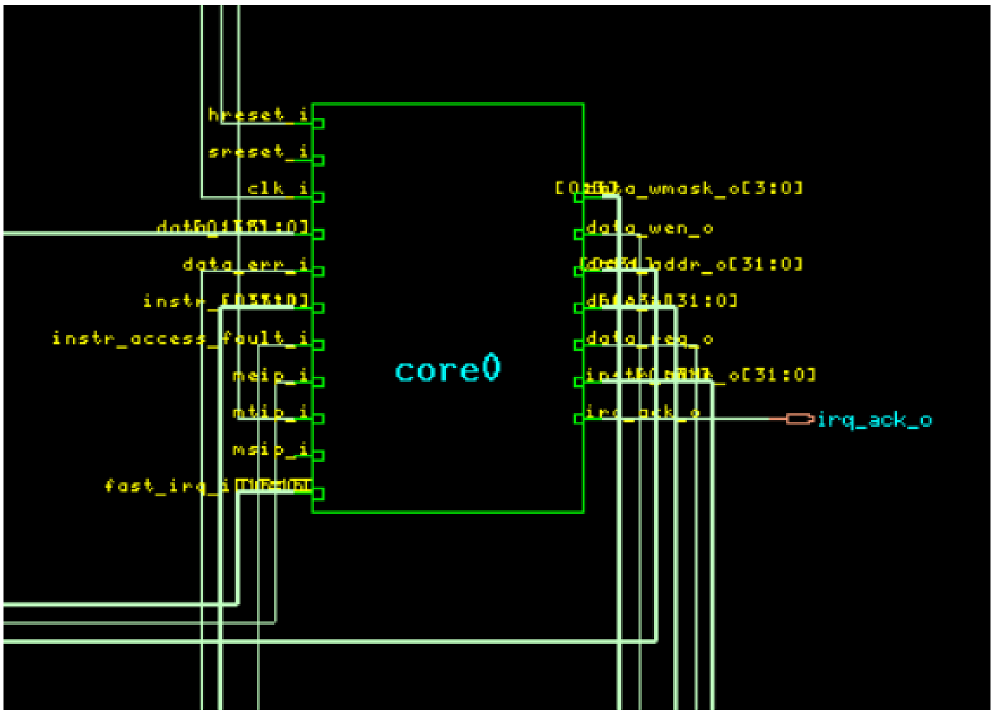


Figure 4. 8: Elaborated Processor Core in the Overall System

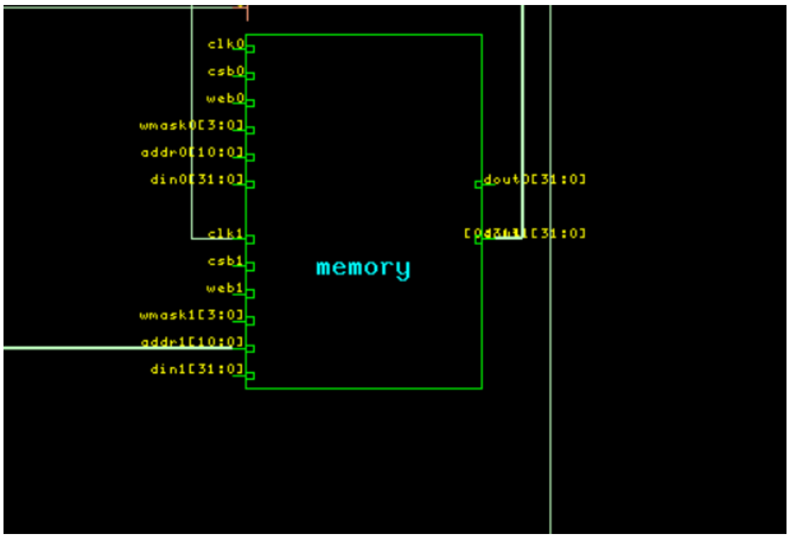


Figure 4. 9: Elaborated DPRAM in the Overall System

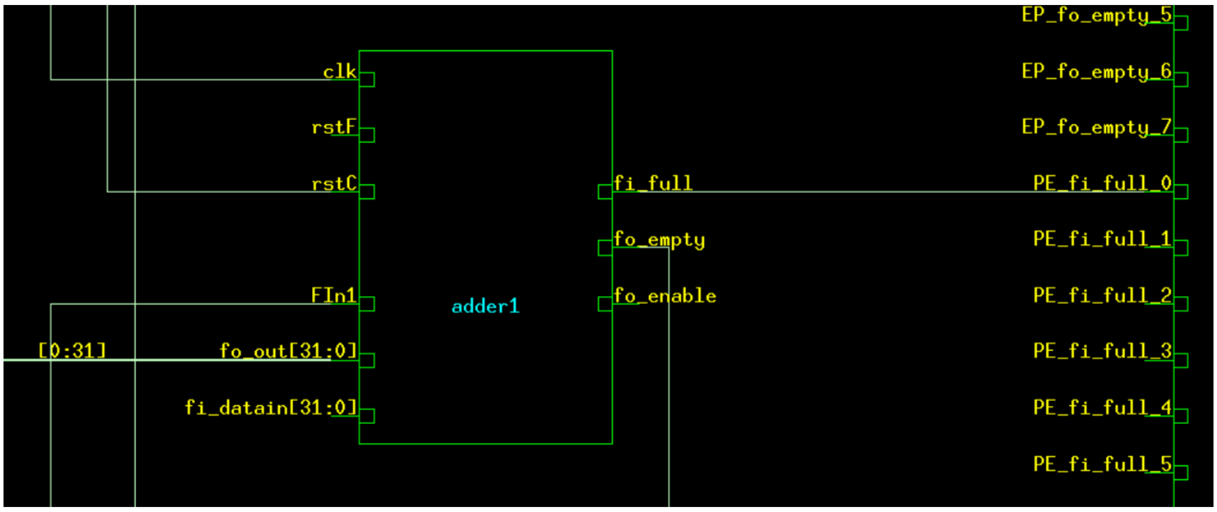
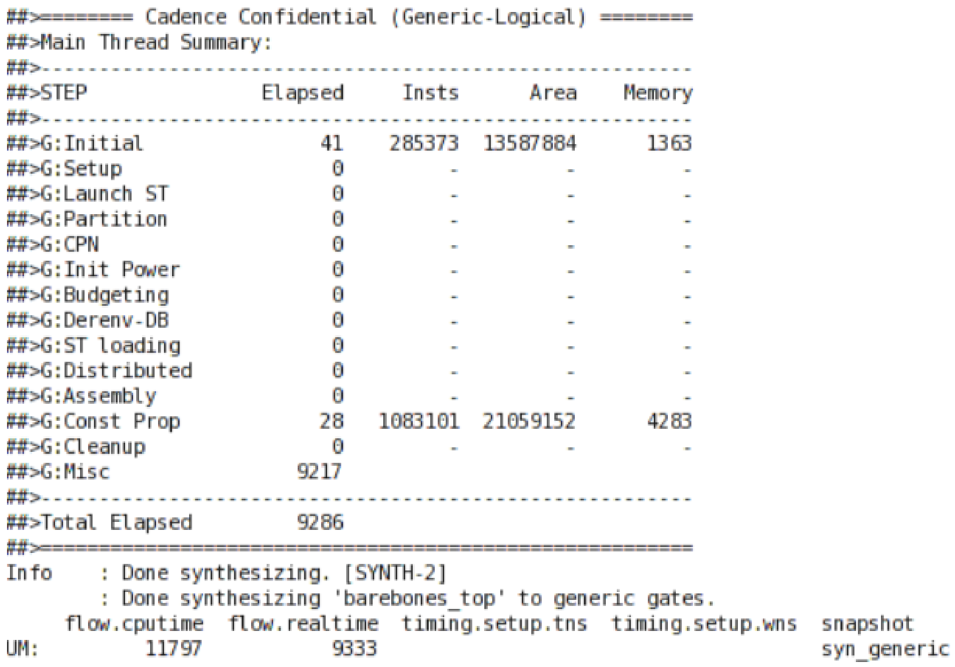


Figure 4. 10: Adder Core in the Overall System

Since the elaboration is successfully done, the system is ready for the synthesis. The commands for the syhnthesis are as follows,

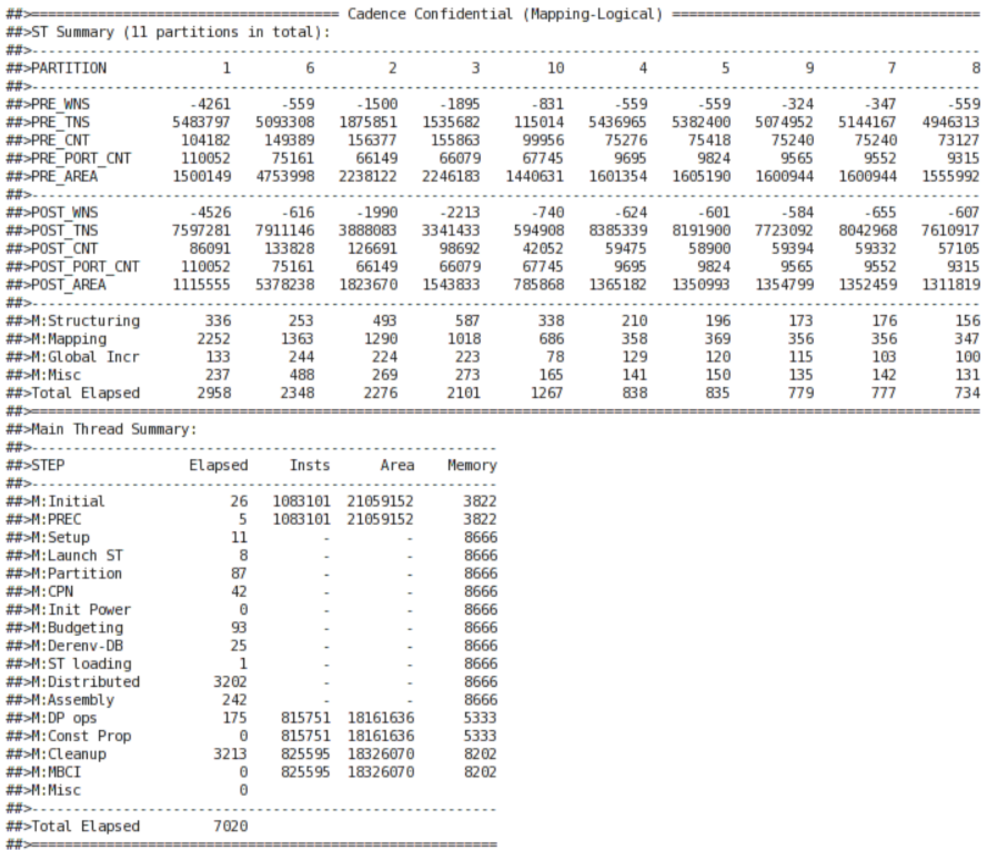
- `create_clock -name clk_i -period 2 -waveform {0 1} clk_i`
- `syn_gen`



```
##>======== Cadence Confidential (Generic-Logical) ========
##>Main Thread Summary:
##>-----------------------------------------------------
##>STEP                 Elapsed     Insts      Area    Memory
##>-----------------------------------------------------
##>G:Initial                41    285373  13587884      1363
##>G:Setup                   0         -         -         -
##>G:Launch ST               0         -         -         -
##>G:Partition               0         -         -         -
##>G:CPN                     0         -         -         -
##>G:Init Power              0         -         -         -
##>G:Budgeting               0         -         -         -
##>G:Derenv-DB               0         -         -         -
##>G:ST loading              0         -         -         -
##>G:Distributed             0         -         -         -
##>G:Assembly                0         -         -         -
##>G:Const Prop             28   1083101  21059152      4283
##>G:Cleanup                 0         -         -         -
##>G:Misc                 9217
##>-----------------------------------------------------
##>Total Elapsed          9286
##>=====================================================
Info    : Done synthesizing. [SYNTH-2]
        : Done synthesizing 'barebones_top' to generic gates.
     flow.cputime  flow.realtime  timing.setup.tns  timing.setup.wns  snapshot
UM:         11797           9333                                      syn_generic
```

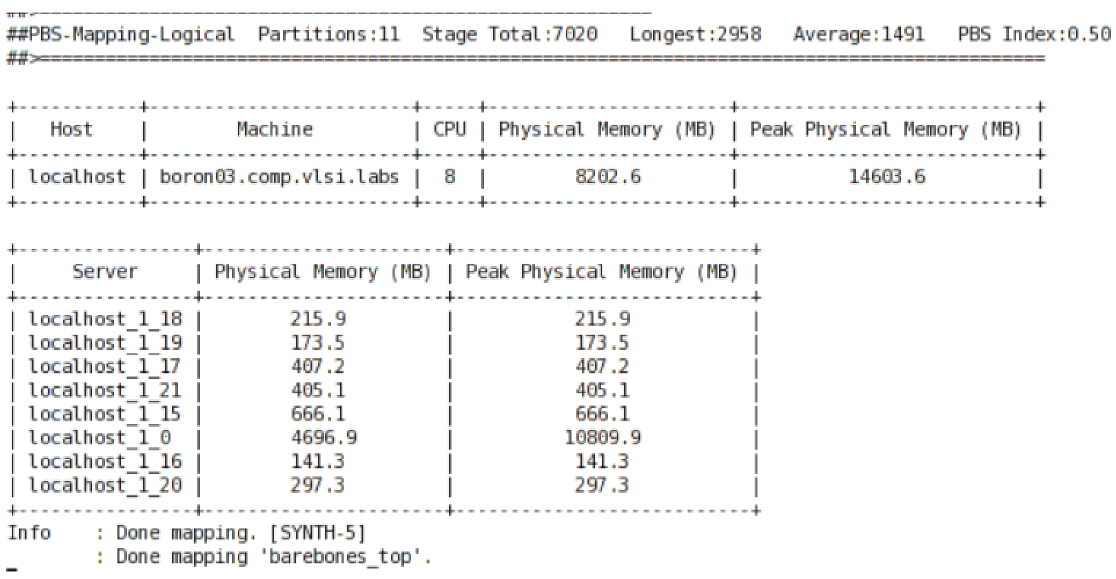
Figure 4. 11: Generic Gate Design Report of the Overall Systems

- syn_map



```
##>==================================== Cadence Confidential (Mapping-Logical) ====================================
##>ST Summary (11 partitions in total):
##>----------------------------------------------------------------------------------------------------------------
##>PARTITION              1          6          2          3         10          4          5          9          7          8
##>----------------------------------------------------------------------------------------------------------------
##>PRE_WNS            -4261       -559      -1500      -1895       -831       -559       -559       -324       -347       -559
##>PRE_TNS          5483797    5093308    1875851    1535682     115014    5436965    5382400    5074952    5144167    4946313
##>PRE_CNT           104182     149389     156377     155863      99956      75276      75418      75240      75240      73127
##>PRE_PORT_CNT      110052      75161      66149      66079      67745       9695       9824       9565       9552       9315
##>PRE_AREA         1500149    4753998    2238122    2246183    1440631    1601354    1605190    1600944    1600944    1555992
##>----------------------------------------------------------------------------------------------------------------
##>POST_WNS           -4526       -616      -1990      -2213       -740       -624       -601       -584       -655       -607
##>POST_TNS         7597281    7911146    3888083    3341433     594908    8385339    8191900    7723092    8042968    7610917
##>POST_CNT           86091     133828     126691      98692      42052      59475      58900      59394      59332      57105
##>POST_PORT_CNT     110052      75161      66149      66079      67745       9695       9824       9565       9552       9315
##>POST_AREA        1115555    5378238    1823670    1543833     785868    1365182    1350993    1354799    1352459    1311819
##>----------------------------------------------------------------------------------------------------------------
##>M:Structuring        336        253        493        587        338        210        196        173        176        156
##>M:Mapping           2252       1363       1290       1018        686        358        369        356        356        347
##>M:Global Incr        133        244        224        223         78        129        120        115        103        100
##>M:Misc               237        488        269        273        165        141        150        135        142        131
##>Total Elapsed       2958       2348       2276       2101       1267        838        835        779        777        734
##>================================================================================================================
##>Main Thread Summary:
##>-----------------------------------------------------
##>STEP                 Elapsed     Insts      Area    Memory
##>-----------------------------------------------------
##>M:Initial                 26   1083101  21059152      3822
##>M:PREC                     5   1083101  21059152      3822
##>M:Setup                   11         -         -      8666
##>M:Launch ST                8         -         -      8666
##>M:Partition               87         -         -      8666
##>M:CPN                     42         -         -      8666
##>M:Init Power               0         -         -      8666
##>M:Budgeting               93         -         -      8666
##>M:Derenv-DB               25         -         -      8666
##>M:ST loading               1         -         -      8666
##>M:Distributed           3202         -         -      8666
##>M:Assembly               242         -         -      8666
##>M:DP ops                 175    815751  18161636      5333
##>M:Const Prop               0    815751  18161636      5333
##>M:Cleanup               3213    825595  18326070      8202
##>M:MBCI                     0    825595  18326070      8202
##>M:Misc                     0
##>-----------------------------------------------------
##>Total Elapsed          7020
##>=====================================================
```

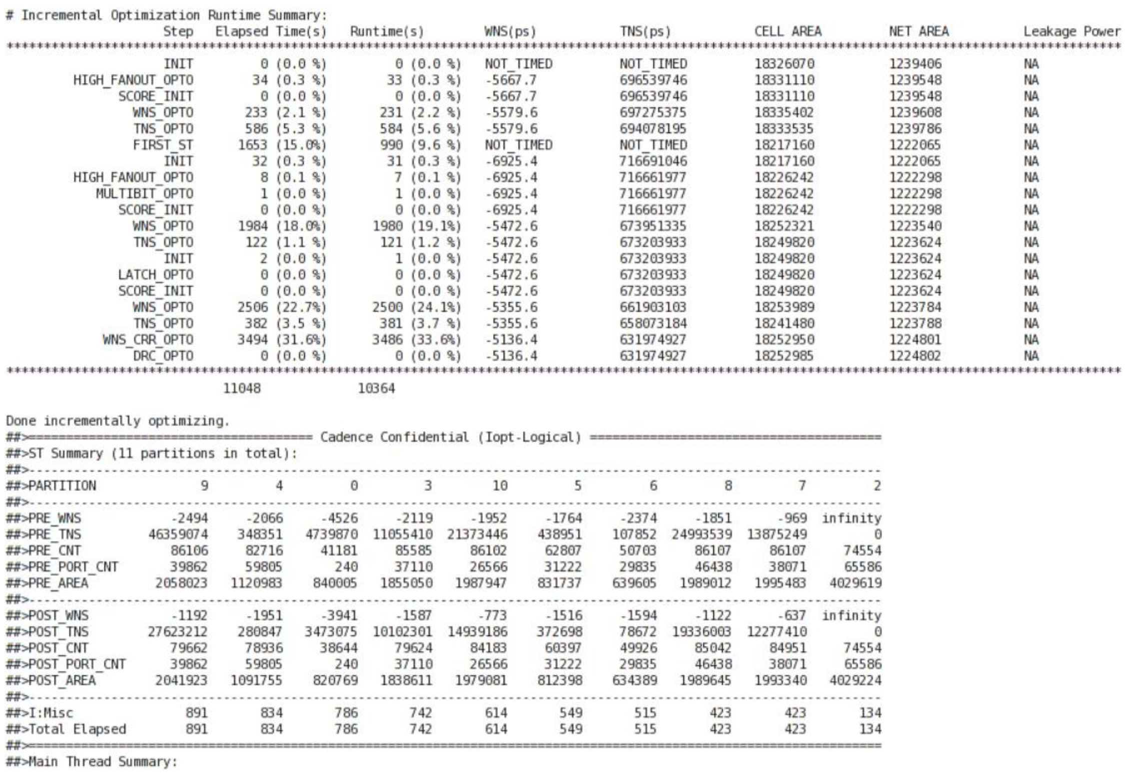
Figure 4. 12: Mapping Report Part 1 of Overall System



```
##PBS-Mapping-Logical   Partitions:11   Stage Total:7020   Longest:2958   Average:1491   PBS Index:0.50
##>==========================================================================================

+-----------+-------------------------+-----+-----------------------+---------------------------+
|   Host    |         Machine         | CPU | Physical Memory (MB)  | Peak Physical Memory (MB) |
+-----------+-------------------------+-----+-----------------------+---------------------------+
| localhost | boron03.comp.vlsi.labs  |  8  |        8202.6         |          14603.6          |
+-----------+-------------------------+-----+-----------------------+---------------------------+

+---------------+----------------------+---------------------------+
|     Server    | Physical Memory (MB) | Peak Physical Memory (MB) |
+---------------+----------------------+---------------------------+
| localhost_1_18 |       215.9          |           215.9           |
| localhost_1_19 |       173.5          |           173.5           |
| localhost_1_17 |       407.2          |           407.2           |
| localhost_1_21 |       405.1          |           405.1           |
| localhost_1_15 |       666.1          |           666.1           |
| localhost_1_0  |       4696.9         |          10809.9          |
| localhost_1_16 |       141.3          |           141.3           |
| localhost_1_20 |       297.3          |           297.3           |
+---------------+----------------------+---------------------------+
Info    : Done mapping. [SYNTH-5]
        : Done mapping 'barebones_top'.
_
```

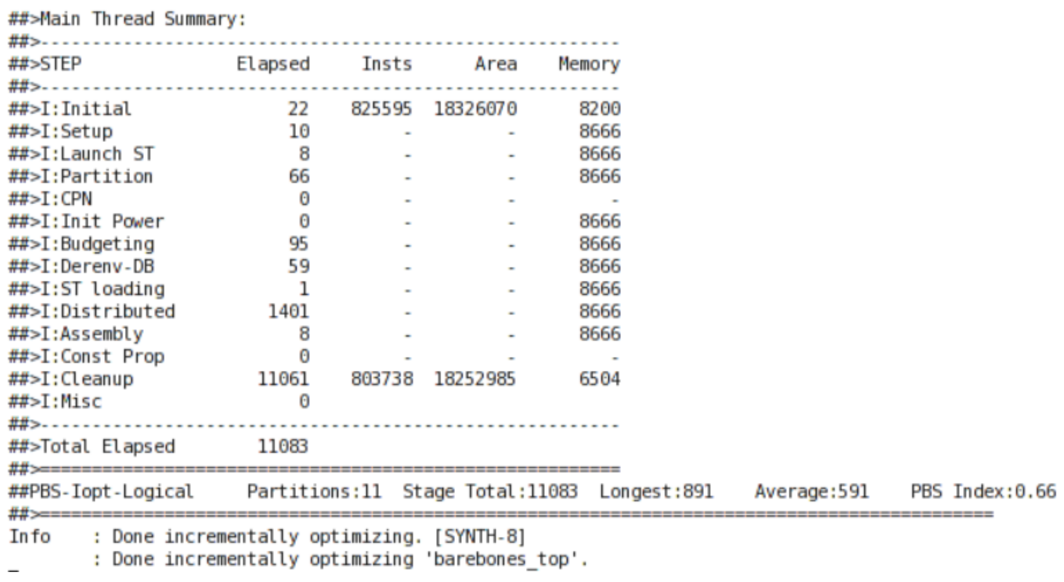
Figure 4. 13: Mapping Report Part 2 of Overall System

- `syn_opt`



```
# Incremental Optimization Runtime Summary:
                     Step   Elapsed Time(s)   Runtime(s)       WNS(ps)          TNS(ps)          CELL AREA        NET AREA         Leakage Power
*****************************************************************************************************************************************************
                     INIT        0 (0.0 %)        0 (0.0 %)   NOT_TIMED        NOT_TIMED        18326070         1239406          NA
         HIGH_FANOUT_OPTO       34 (0.3 %)       33 (0.3 %)   -5667.7          696539746        18331110         1239548          NA
               SCORE_INIT        0 (0.0 %)        0 (0.0 %)   -5667.7          696539746        18331110         1239548          NA
                 WNS_OPTO      233 (2.1 %)      231 (2.2 %)   -5579.6          697275375        18335402         1239608          NA
                 TNS_OPTO      586 (5.3 %)      584 (5.6 %)   -5579.6          694078195        18333535         1239786          NA
                 FIRST_ST     1653 (15.0%)      990 (9.6 %)   NOT_TIMED        NOT_TIMED        18217160         1222065          NA
                     INIT       32 (0.3 %)       31 (0.3 %)   -6925.4          716691046        18217160         1222065          NA
         HIGH_FANOUT_OPTO        8 (0.1 %)        7 (0.1 %)   -6925.4          716661977        18226242         1222298          NA
            MULTIBIT_OPTO        1 (0.0 %)        1 (0.0 %)   -6925.4          716661977        18226242         1222298          NA
               SCORE_INIT        0 (0.0 %)        0 (0.0 %)   -6925.4          716661977        18226242         1222298          NA
                 WNS_OPTO     1984 (18.0%)     1980 (19.1%)   -5472.6          673951335        18252321         1223540          NA
                 TNS_OPTO      122 (1.1 %)      121 (1.2 %)   -5472.6          673203933        18249820         1223624          NA
                     INIT        2 (0.0 %)        1 (0.0 %)   -5472.6          673203933        18249820         1223624          NA
               LATCH_OPTO        0 (0.0 %)        0 (0.0 %)   -5472.6          673203933        18249820         1223624          NA
               SCORE_INIT        0 (0.0 %)        0 (0.0 %)   -5472.6          673203933        18249820         1223624          NA
                 WNS_OPTO     2506 (22.7%)     2500 (24.1%)   -5355.6          661903103        18253989         1223784          NA
                 TNS_OPTO      382 (3.5 %)      381 (3.7 %)   -5355.6          658073184        18241480         1223788          NA
             WNS_CRR_OPTO     3494 (31.6%)     3486 (33.6%)   -5136.4          631974927        18252950         1224801          NA
                 DRC_OPTO        0 (0.0 %)        0 (0.0 %)   -5136.4          631974927        18252985         1224802          NA
*****************************************************************************************************************************************************
                             11048            10364

Done incrementally optimizing.
##>==================================== Cadence Confidential (Iopt-Logical) ====================================
##>ST Summary (11 partitions in total):
##>-----------------------------------------------------------------------------------------------------------
##>PARTITION            9        4        0        3       10        5        6        8        7        2
##>-----------------------------------------------------------------------------------------------------------
##>PRE_WNS          -2494    -2066    -4526    -2119    -1952    -1764    -2374    -1851     -969  infinity
##>PRE_TNS       46359074   348351  4739870 11055410 21373446   438951   107852 24993539 13875249         0
##>PRE_CNT          86106    82716    41181    85585    86102    62807    50703    86107    86107     74554
##>PRE_PORT_CNT     39862    59805      240    37110    26566    31222    29835    46438    38071     65586
##>PRE_AREA       2058023  1120983   840005  1855050  1987947   831737   639605  1989012  1995483   4029619
##>-----------------------------------------------------------------------------------------------------------
##>POST_WNS         -1192    -1951    -3941    -1587     -773    -1516    -1594    -1122     -637  infinity
##>POST_TNS      27623212   280847  3473075 10102301 14939186   372698    78672 19336003 12277410         0
##>POST_CNT         79662    78936    38644    79624    84183    60397    49926    85042    84951     74554
##>POST_PORT_CNT    39862    59805      240    37110    26566    31222    29835    46438    38071     65586
##>POST_AREA      2041923  1091755   820769  1838611  1979081   812398   634389  1989645  1993340   4029224
##>-----------------------------------------------------------------------------------------------------------
##>I:Misc             891      834      786      742      614      549      515      423      423       134
##>Total Elapsed      891      834      786      742      614      549      515      423      423       134
##>===========================================================================================================
##>Main Thread Summary:
```

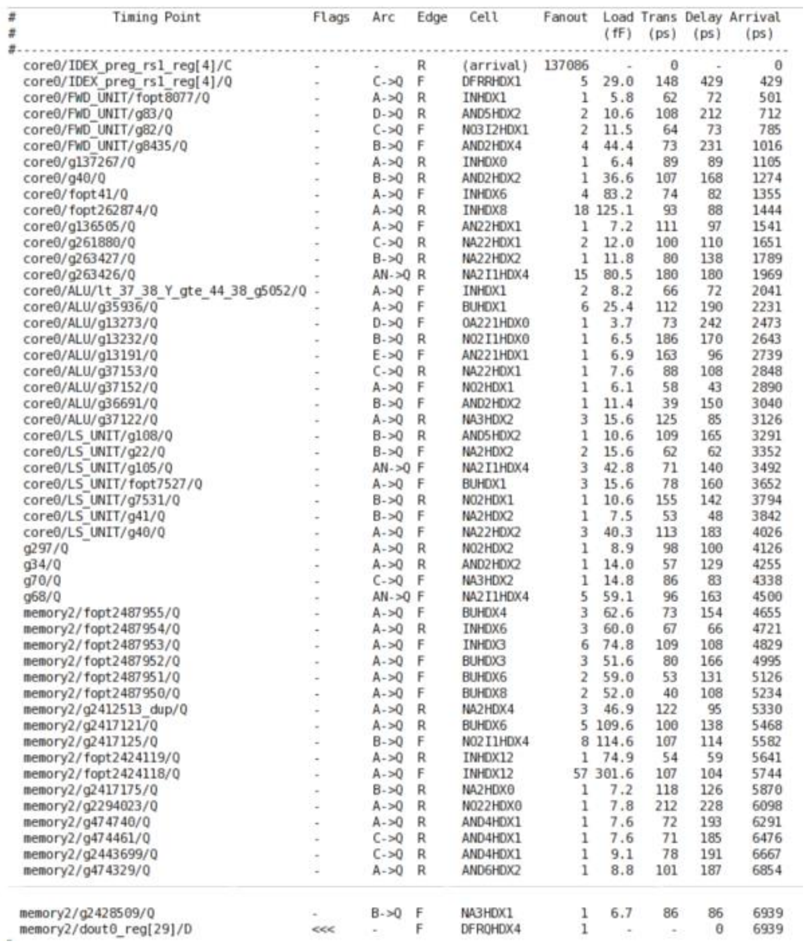
Figure 4. 14: Optimization Report Part 1 of Overall System



```
##>Main Thread Summary:
##>-------------------------------------------------------
##>STEP                 Elapsed     Insts      Area    Memory
##>-------------------------------------------------------
##>I:Initial                 22    825595  18326070      8200
##>I:Setup                   10         -         -      8666
##>I:Launch ST                8         -         -      8666
##>I:Partition               66         -         -      8666
##>I:CPN                      0         -         -         -
##>I:Init Power               0         -         -      8666
##>I:Budgeting               95         -         -      8666
##>I:Derenv-DB               59         -         -      8666
##>I:ST loading               1         -         -      8666
##>I:Distributed           1401         -         -      8666
##>I:Assembly                8         -         -      8666
##>I:Const Prop               0         -         -         -
##>I:Cleanup             11061    803738  18252985      6504
##>I:Misc                     0
##>-------------------------------------------------------
##>Total Elapsed         11083
##>=======================================================
##PBS-Iopt-Logical     Partitions:11  Stage Total:11083  Longest:891    Average:591    PBS Index:0.66
##>==================================================================================
Info    : Done incrementally optimizing. [SYNTH-8]
_       : Done incrementally optimizing 'barebones_top'.
```

Figure 4. 15: Optimization Report Part 1 of Overall System

- `report_timing`



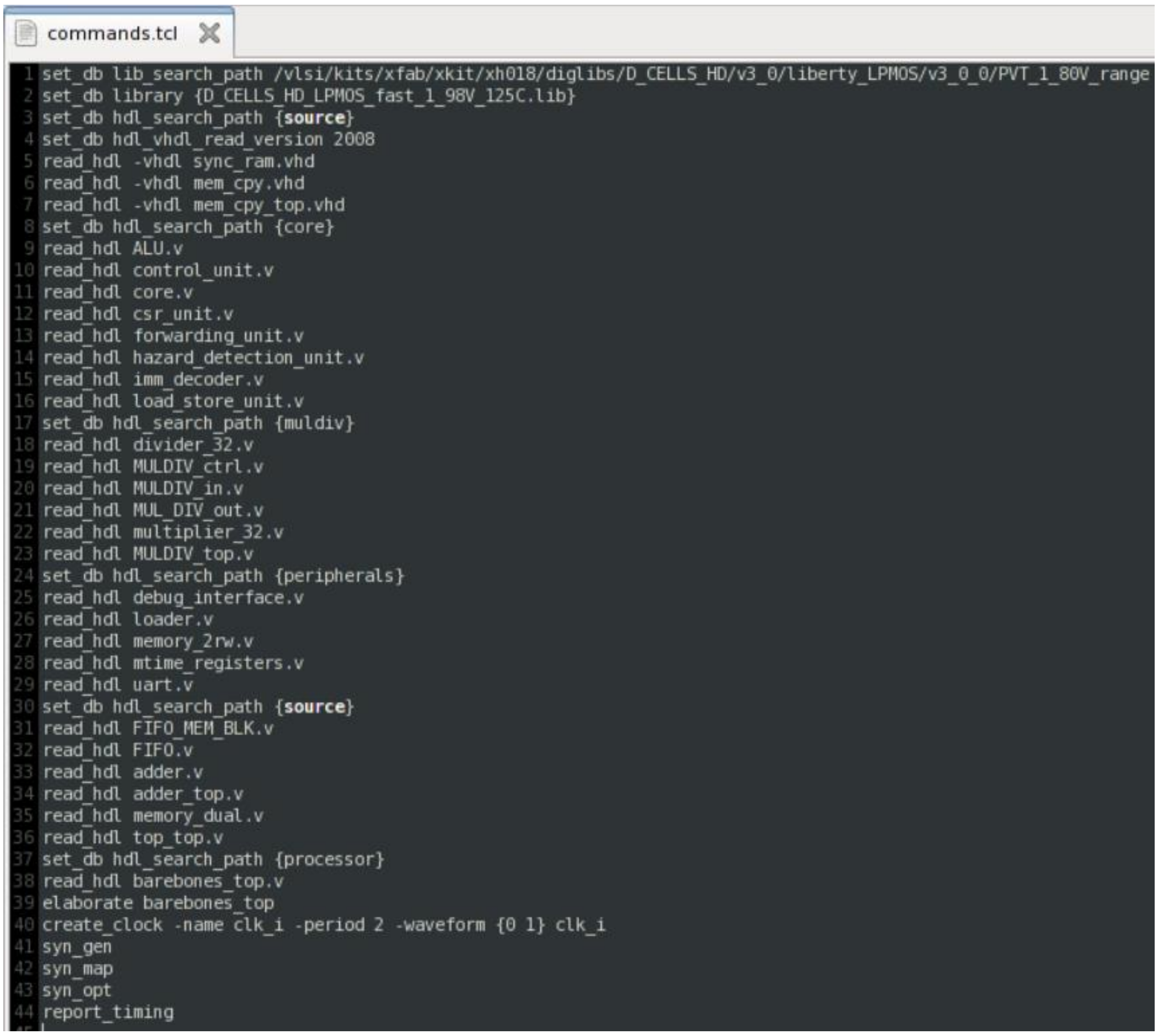
```
#              Timing Point                Flags   Arc    Edge   Cell        Fanout  Load  Trans Delay Arrival
#                                                                                    (fF)  (ps)  (ps)   (ps)
#---------------------------------------------------------------------------------------------------------------
  core0/IDEX_preg_rs1_reg[4]/C               -      -      R    (arrival)   137086    -      0     -      0
  core0/IDEX_preg_rs1_reg[4]/Q               -     C->Q    F    DFRRHDX1         5  29.0   148   429    429
  core0/FWD_UNIT/fopt8077/Q                  -     A->Q    R    INHDX1           1   5.8    62    72    501
  core0/FWD_UNIT/g83/Q                       -     D->Q    R    AND5HDX2         2  10.6   108   212    712
  core0/FWD_UNIT/g82/Q                       -     C->Q    F    NO3I2HDX1        2  11.5    64    73    785
  core0/FWD_UNIT/g8435/Q                     -     B->Q    F    AND2HDX4         4  44.4    73   231   1016
  core0/g137267/Q                            -     A->Q    R    INHDX0           1   6.4    89    89   1105
  core0/g40/Q                                -     B->Q    R    AND2HDX2         1  36.6   107   168   1274
  core0/fopt41/Q                             -     A->Q    F    INHDX6           4  83.2    74    82   1355
  core0/fopt262874/Q                         -     A->Q    R    INHDX8          18 125.1    93    88   1444
  core0/g136505/Q                            -     A->Q    F    AN22HDX1         1   7.2   111    97   1541
  core0/g261880/Q                            -     C->Q    R    NA22HDX1         2  12.0   100   110   1651
  core0/g263427/Q                            -     B->Q    R    NA22HDX2         1  11.8    80   138   1789
  core0/g263426/Q                            -     AN->Q   R    NA2I1HDX4       15  80.5   180   180   1969
  core0/ALU/lt_37_38_Y_gte_44_38_g5052/Q     -     A->Q    F    INHDX1           2   8.2    66    72   2041
  core0/ALU/g35936/Q                         -     A->Q    F    BUHDX1           6  25.4   112   190   2231
  core0/ALU/g13273/Q                         -     D->Q    F    OA221HDX0        1   3.7    73   242   2473
  core0/ALU/g13232/Q                         -     B->Q    R    NO2I1HDX0        1   6.5   186   170   2643
  core0/ALU/g13191/Q                         -     E->Q    F    AN221HDX1        1   6.9   163    96   2739
  core0/ALU/g37153/Q                         -     C->Q    R    NA22HDX1         1   7.6    88   108   2848
  core0/ALU/g37152/Q                         -     A->Q    F    NO2HDX1          1   6.1    58    43   2890
  core0/ALU/g36691/Q                         -     B->Q    F    AND2HDX2         1  11.4    39   150   3040
  core0/ALU/g37122/Q                         -     A->Q    R    NA3HDX2          3  15.6   125    85   3126
  core0/LS_UNIT/g108/Q                       -     B->Q    R    AND5HDX2         1  10.6   109   165   3291
  core0/LS_UNIT/g22/Q                        -     B->Q    F    NA2HDX2          2  15.6    62    62   3352
  core0/LS_UNIT/g105/Q                       -     AN->Q   F    NA2I1HDX4        3  42.8    71   140   3492
  core0/LS_UNIT/fopt7527/Q                   -     A->Q    F    BUHDX1           3  15.6    78   160   3652
  core0/LS_UNIT/g7531/Q                      -     B->Q    R    NO2HDX1          1  10.6   155   142   3794
  core0/LS_UNIT/g41/Q                        -     B->Q    F    NA2HDX2          1   7.5    53    48   3842
  core0/LS_UNIT/g40/Q                        -     A->Q    F    NA22HDX2         3  40.3   113   183   4026
  g297/Q                                     -     A->Q    R    NO2HDX2          1   8.9    98   100   4126
  g34/Q                                      -     A->Q    R    AND2HDX2         1  14.0    57   129   4255
  g70/Q                                      -     C->Q    F    NA3HDX2          1  14.8    86    83   4338
  g68/Q                                      -     AN->Q   F    NA2I1HDX4        5  59.1    96   163   4500
  memory2/fopt2487955/Q                      -     A->Q    F    BUHDX4           3  62.6    73   154   4655
  memory2/fopt2487954/Q                      -     A->Q    R    INHDX6           3  60.0    67    66   4721
  memory2/fopt2487953/Q                      -     A->Q    F    INHDX3           6  74.8   109   108   4829
  memory2/fopt2487952/Q                      -     A->Q    F    BUHDX3           3  51.6    80   166   4995
  memory2/fopt2487951/Q                      -     A->Q    F    BUHDX6           2  59.0    53   131   5126
  memory2/fopt2487950/Q                      -     A->Q    F    BUHDX8           2  52.0    40   108   5234
  memory2/g2412513_dup/Q                     -     A->Q    R    NA2HDX4          3  46.9   122    95   5330
  memory2/g2417121/Q                         -     A->Q    R    BUHDX6           5 109.6   100   138   5468
  memory2/g2417125/Q                         -     B->Q    F    NO2I1HDX4        8 114.6   107   114   5582
  memory2/fopt2424119/Q                      -     A->Q    R    INHDX12          1  74.9    54    59   5641
  memory2/fopt2424118/Q                      -     A->Q    F    INHDX12         57 301.6   107   104   5744
  memory2/g2417175/Q                         -     B->Q    R    NA2HDX0          1   7.2   118   126   5870
  memory2/g2294023/Q                         -     A->Q    R    NO22HDX0         1   7.8   212   228   6098
  memory2/g474740/Q                          -     A->Q    R    AND4HDX1         1   7.6    72   193   6291
  memory2/g474461/Q                          -     C->Q    R    AND4HDX1         1   7.6    71   185   6476
  memory2/g2443699/Q                         -     C->Q    R    AND4HDX1         1   9.1    78   191   6667
  memory2/g474329/Q                          -     A->Q    R    AND6HDX2         1   8.8   101   187   6854

 memory2/g2428509/Q                          -     B->Q    F    NA3HDX1          1   6.7    86    86   6939
 memory2/dout0_reg[29]/D                    <<<     -      F    DFRQHDX4         1    -      -     0   6939
_
```

Figure 4. 16: Timing Report of Overall System

Instead of writing all these commands one by one, at the accurate location a file named “commands.tcl” could be created with all these commands written inside of it, and when the Genus is invoked the command,

- `source commands.tcl`

could be run. This command would give all the commands to the Genus.



commands.tcl

```
set_db lib_search_path /vlsi/kits/xfab/xkit/xh018/diglibs/D_CELLS_HD/v3_0/liberty_LPMOS/v3_0_0/PVT_1_80V_range
set_db library {D_CELLS_HD_LPMOS_fast_1_98V_125C.lib}
set_db hdl_search_path {source}
set_db hdl_vhdl_read_version 2008
read_hdl -vhdl sync_ram.vhd
read_hdl -vhdl mem_cpy.vhd
read_hdl -vhdl mem_cpy_top.vhd
set_db hdl_search_path {core}
read_hdl ALU.v
read_hdl control_unit.v
read_hdl core.v
read_hdl csr_unit.v
read_hdl forwarding_unit.v
read_hdl hazard_detection_unit.v
read_hdl imm_decoder.v
read_hdl load_store_unit.v
set_db hdl_search_path {muldiv}
read_hdl divider_32.v
read_hdl MULDIV_ctrl.v
read_hdl MULDIV_in.v
read_hdl MUL_DIV_out.v
read_hdl multiplier_32.v
read_hdl MULDIV_top.v
set_db hdl_search_path {peripherals}
read_hdl debug_interface.v
read_hdl loader.v
read_hdl memory_2rw.v
read_hdl mtime_registers.v
read_hdl uart.v
set_db hdl_search_path {source}
read_hdl FIFO_MEM_BLK.v
read_hdl FIFO.v
read_hdl adder.v
read_hdl adder_top.v
read_hdl memory_dual.v
read_hdl top_top.v
set_db hdl_search_path {processor}
read_hdl barebones_top.v
elaborate barebones_top
create_clock -name clk_i -period 2 -waveform {0 1} clk_i
syn_gen
syn_map
syn_opt
report_timing
```

Figure 4. 17: Commands.tcl file

The system is successfully synthesized on Cadence Genus tool, and all the reports related to the synthesis are presented. It is proved that this newly designed DMA is meeting its design criterias also for the ASIC design implementation.

5. RESULTS AND RECOMMENDATIONS

As it is mentioned in the overall system simulation section, we have managed to connect all the system together and ran the simulation by running the C code in order to test the system. Due to data adress algorithm of the procosser, DMA could not read the commands from the data memory. Nevertheless, system would work after adressing issue is fixed since all the elements are tested and verified seperately.

In this project, all the memory elements such as FIFO, RAM aree written by us in order to be able to implement them in ASIC since ASIC memory elements are expensive. When implementing this system on chip, it would be muc more effcient in terms of performance and area usage to buy memory elements rather than writing them by using register elements.

6. REALISTIC CONSTRAINTS AND CONCLUSIONS

The amount of the data that needs to be processed in the technological systems is increasing day by day due to the advanced technological developments around the world. As a result, processing this vast amount of data with a high speed is essential and crucial for this high-tech systems. This DMA project is presented as a solution to this performance problem, and with the work done in this project, it is shown that this DMA is able to be implemented as ASIC.

6.1 Practical Application of This Project

The vast majority of technological developments aimed at increasing our quality of life and the way that we understand the universe, including in vital sectors, do this by perceiving the data around it or by processing the data directly uploaded to the system. This DMA block is designed to increase the performance of these products by using them in these technological products.

6.2 Realistic Constraints

During this project, as the work progressed, many problems arised mostly due to the lack of budget and the time.

6.2.1 Social, environmental and economic impact

This DMA design differs significantly from the other DMA designs realized until now. To be able to utilize this DMA for its considerable advantages, this DMA needs to be purchased by the users.

6.2.2 Cost analysis

Since there is a limited budget allocated for this project, limitations were encountered at several stages during the course of the project, such as the need to purchase FIFO, RAM, DPRAM for the overall system. In addition, Cadence, the tool on which the project was implemented, and Vivado, which is used as an assistant throughout the project, are also paid, and both platforms were made available to us free of charge by our university.

6.2.3 Standards

Throughout the project, the hardware description languages Verilog and VHDL were studied within the scope of the standards set by IEEE.

6.2.4 Health and safety concerns

In the work progress of this project, there has been no health and safety concerns.

6.2.5 Future Work and Recommendations

Our project is part of a two-year spanned TÜBİTAK project and we are the first people to work on the ASIC implementation of this unique design. Related to this issue, since our work proved the ASIC design also meets the required criterias and the DMA module is working in a larger system which includes connections to a processor core, a DPRAM and a processing element, our recommendation for the future work related to this DMA is improving the timing performance of the design by using our work as a base. The technology we used while synthesizing the design should be considered, and it should be on mind that changing the technology is also an option for improving the performance.

7. REFERENCES

[1] H. Morales, C. Duran and E. Roa, "A Low-Area Direct Memory Access Controller Architecture for a RISC-V Based Low-Power Microcontroller," 2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS), 2019, pp. 97-100, doi: 10.1109/LASCAS.2019.8667579.

[2] Y. J. M. Shirur, K. M. Sharma and A. A, "Design and implementation of Efficient Direct Memory Access (DMA) Controller in Multiprocessor SoC," 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), 2018, pp. 1-6, doi: 10.1109/ICNEWS.2018.8903991.

[3] A. Rani, A. Pai, B. Naware, Z. H. Yang and T. -Y. Huang, "Direct Memory Access Remapping for Thunderbolt, Feature Deployment at Platform Level," 2020 IEEE International Conference for Innovation in Technology (INOCON), 2020, pp. 1- 5, doi: 10.1109/INOCON50539.2020.9298289.

[4] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface. Waltham, MA: Elsevier, 2012.

[5] Y. Yılmaz, Y. S. Tozlu,(2021), Design and Implementation of a 32-bit RISC-V Core [Bachelor’s Thesis, Istanbul Technical University]. https://web.itu.edu.tr/~orssi/thesis/2021/YavuzTozlu_bit.pdf

[6] ‘ Hornet RISC-V Core’ https://github.com/yavuz650/RISC-V

[7] Palnitkar, S. (2003), Verilog HDL: A Guide to Digital Design and Synthesis, (2nd ed.) , Prentice Hall PTR