POLITECNICO DI TORINO

Efficient FPGA implementation of High-Performance Computing applications via High-Level Synthesis

by

Guo Qinglang

A thesis for the
degree of Master of Science

in the
Department of Electronics and Telecommunications

March 2018

Declaration of Authorship

I, Guo Qinglang, declare that I have done this thesis and the work by my own. I make a confirm that:

- I have done this work in the last year of master study in purpose of obtaining a degree at this University.
- I have stated if some parts of the thesis has been submitted by other institutions.
- I have made attribution whenever I consulted the published paper.
- I appreciate all kinds of assistant from other professors or colleagues.
- I have made statement about the thesis work I have done.

Signed: _____

Date: _____

"A journey of a thousand miles begins with single step."

Lao Zi

Abstract

The aim of this work is to develop some applications via high-level synthesis based on an FPGA board, and the performance of the application could be improved as compares with these applications without high level synthesis. The idea is divided into two parts, the first part is that transforms an algorithm description into a hardware implementation in Vivado HLS and Catapult; The second part is that synthesis the hardware implementation in Vivado that interface with the hardware that generated by vivado _ hls and catapult with the processing system and the memory system in the same FPGA board, and compare the timing and utilization for different cases.

In order to achieve these objectives, three software is desired which is (1)vivado_hls and catapult, a tool can use a series of steps to generate the hardware.(2)vivado, a tool can generate the system and compare the result between these different cases.

In this work, c language is used to write the code,vivado _ hls and catapult is used to transfer the code into RTL implementation, vhdl is used to package the ip.

Acknowledgements

I want to start with the feeling of appreciation to My supervisor, Prof. Luciano Lavagno and co-supervisor Dr. Mihai Lasarescu for their support and motivation throughout the course of this work. Without their help, my work and paper might not have been so smooth.. They provided me with the opportunity to improve my technical skills and provided me the resources to fulfill this task.

I would appreciate Dr.Ma Liang, Dr.Arslan Arif, Dr.Javed Iqbal for their suggestions and useful instructions. Also the other staffs and classmates in the department, like Dr.Sarmad Uiiah, Dr.Shan Junnan, Marco De Clemente, Pooya Poolad, Pablo Henao have given me a hand when I faced problems.

I am thankful to my parents and family and friends, who have always been a source of motivation for me in the endeavor of knowledge.

Moreover, I am especially thankful to my girlfriend Zhao Rui for her endless loves and timely assistance both in the study and in my life.

Dedicated to my parents "Guo Bangqiang" and "Chen Yueying".

Contents

List of Figures

Abbreviations

HLS	**H**igh **L**evel **S**ynthesis
FPGA	**F**ield- **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**description **L**anguage
CPU	**C**entral **P**rocessing **U**nit
KNN	**K** **N**earest **N**eighbor
RTL	**R**egister **T**ransfer **L**evel
ESL	**E**lectronic **S**ystem **L**evel

Chapter 1

Introduction

1.1 Electronic System Design

In electronic system design, there are some common features in electronics system level design synthesis methodologies. In this chapter, the common principles would be specified in the first and second parts. In order to get a general picture for the ESL synthesis, the system design process would be specified in general to better understand the electronic system level design.

1.1.1 Design Flow

For the design flow of ESL level, top-down mode is the typical ESL synthesis approach. [1], I will explain the design process in this chapter in the ways of top down methodologies. Furthermore, the approach will show software synthesis steps in the task level and instruction level while it also shows the hardware synthesis steps in component level and logic level in the design flow. we can seen the double roof model [2] shown in Fig1.1.

In the real electronics system design process,the double roof model includes hardware and software systems. As it can be easily seen from the graph, in the left side, the levels of the software design process has been illuminated as well as the hardware design process in the right side. Two abstraction levels constitute both side, If we look down from the top system level, in the hardware side, includes task and instruction levels; while in the software side, includes component and logic levels. As it shows in the graph, in the top

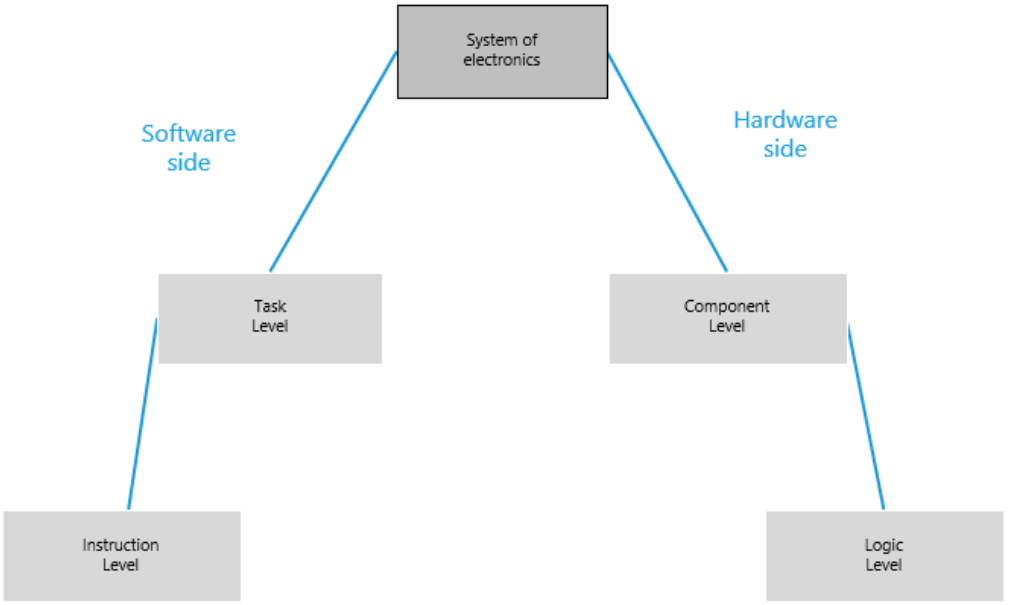

FIGURE 1.1: Electronic system design flow

there is a level named as system level, which is used in both hardware and software. After the completion of each synthesis step, an implementation would generate from the source of the specification. And inside each adjacent level,the elements or result in the current level's implementation would be the input of next down level specifications.

The double roof model shows how the electronic system level specification turns into implementation, within these top down levels, the channels is needed for the process communication. The ESL synthesis is the process for choosing an proper platform, and then can map its behavioral model into architecture, continuing to generate the implementation.

The ESL high level synthesis can contains two parts,for the first synthesis step, a functional description would polish into a structural implementation; for the second synthesis step, the FPGA based processor use as a hardware accelerator to synthesis the process into RTL implementation, ffs, memory units, functional units and connector should the combination of the RTL implementation. In this paper, we use the vivado_hls and Catapult and Vivado as the synthesis tools to perform high-level synthesis behavior. At the logic level, the ESL design would be presented in form of logic gates and flip-flops. If the logic level synthesis resource comes from the Catapult,and in Vivado, we can apply the vhdl source in the implementation within Catapult file; Else if the the logic level

synthesis resource comes from the Vivado HLS and we need to use it in Vivado, then we can generate directly predesigned intellectual property (IP),and then use the ip in Vivado.

1.1.2 Synthesis Process

As mentioned in the previous section, a specification in the higher lower would need multiple step in order to synthesis into an implementation in the lower level, the flow path could be shown in Fig1.2.

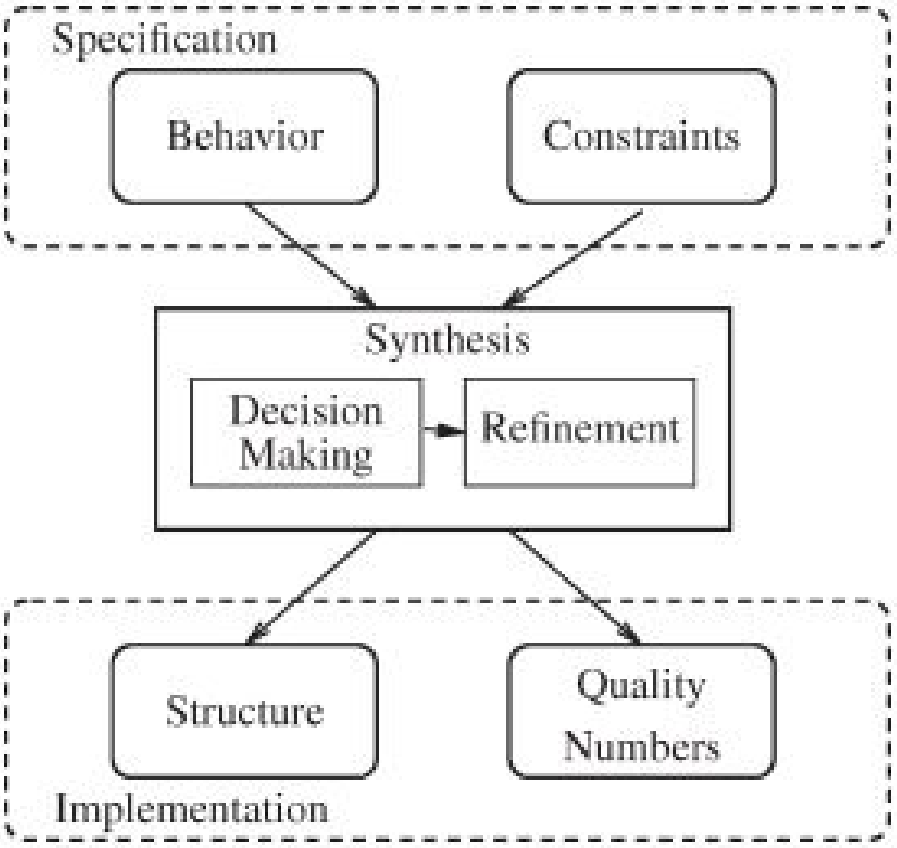

FIGURE 1.2: synthesis process from specification to implementation

Normally, a behavioral model and constraints form a specification, and the ESL system functions were expressed inside the behavioral model. This kind of model declares its expressibility and analyzability. For the software side, By using the programming language like C/C++, Java to represent the behavioral model ; For the hardware side,

the behavioral model can be written by hardware description language, like Verilog or VHDL.

The synthesis process can transform a specification into an implementation. For the specification, it consists of behavior and constraints part, the behavioral model can be refined into structure model. An implementation has two main components, structural model and quality numbers. The behavioral model under the constraints model can refine to structure model, as the structure model keeps the result architecture of synthesis performs, such as the behavior model describes and function, the structure model resulting in architecture as the combination of components.

As it shows in Figure 2, in order to generate an implementation from a specification, the synthesis need the process of decision making and refinement. Decision making means the decision of the mapping process in terms of utilization and timing, such as how many and what kinds of resources should to be use, how much time it needs to complete the synthesis process. At the mean time, in the decision making step the available resource and clock constrains also need to take into account. Moreover, the decision making step will find a solution for resource contention for the refinement operation.

After decision making, the role of refinement is arranging the improper decisions,after removing theses wrong decisions, the efficiency and accuracy could be improved in the final implementation, generating the eligible structure. At the end, a sets of optimization methods would be preformed instead of single optimization model.In this paper, I define more than ten different optimization methods for the synthesis task. In conclusion, decision making define the resource allocation, calculating the timing; wile the refinement can define the quality members as a result of synthesis.

1.2 Electronic System-level Design and High-Level Synthesis

1.2.1 The relevance between ESL and HLS

Basically, ESL design focus at the process where the low levels of abstraction can map into register-transfer level(RTL) components through a series of steps. The system inputs are typically described in the algorithms functions, and the system outputs are

typically described in VHDL or IP. HLS(high level synthesis) is a kind of enabling technology, which provides a bridge between a programming language description design and it's transfer level structural implementation. With the development of automation design, the demand for HSL has risen sharply with the requirement of very large transistor count design.

1.2.2 ESL design methodology

Based on Moore's law, the complexity of chips doubles every 18 months, the complexity of chips make RTL design unable scale with the emerging designs and cost of RTL design is not economy.This trend make the higher levels of abstraction design more economical and practical,thus the ESL designs are becoming a preferable choice .

There are three elements of ESL design in RTL abstraction.The first element is computation which specifies each component's function, and using the hardware description language like verilog and VHDL in RTL abstraction can express each component. The component's computation is obtained by how much registers transformed at each clock cycle, which specifies as per-cycle behavior. The second element is composition, which specifies how components assembled and how to organzie their computation into a larger system. In RTL abstraction, the composition is done by using HDL language like VHDL to connect the components' ports with wires. The third element is the communication, which specifies how the components exchange information through the wires. In conclusion, an RTL method can be specified as: In RTL abstraction synthesis, based on component's computation per-cycle behavior, the synthesis tool converts each entity in VHDL into gate level design; and then combine all the entities together with wires in gate level design for this level optimization. Lastly,with the default clock cycle period, the entities can simulate into processes in the simulation steps. In the contents below, two electronics system level design methods will be specified in the design abstraction level, especially for the field of component constitution and communication. And exam the mapping procedures for the component synthesis and system synthesis, and the verification procedures of component mapping to RTL and full system. There are two main methods, one is function-based ESL methods, which specify how different components into a full system.One method is function-based ESL method, one is architecture-based ESL method.

1.2.3 Function-based ESL methodology

This kind of ESL methodology focus on parallel execution of different components in the system in order to embedded into a complete system, and the communication protocols between each components. In this function based method, there are four component synthesis methods, the first one is the direct translation to RTL; the second one direct map to predesigned intellectual property component the third method is through the high-level synthesis to RTL; The fourth method is compiled to software programs. In the job I have done, these components synthesis were done by the HLS method.

1.2.4 Architecture-based ESL methodology

As some components are reused designs, some these reusable components from previous projects, other from the third parties. In the field of industry, these components named as intellectual property(IP) components. These components can be communicated based on communication protocols like AMBA bus, and this protocol refers to a set of transactions. In this method, a set of virtual components(IP) is embedded into the system. And component synthesis can be done by three main different ways. The first way is instantiating a predesigned IP component; The second is the synthesis process from an architecture description language(ADL) specification; The third is high-level design to RTL, this way is used when an existing IP cannot meet power, energy, cost constraints.

1.2.5 High-level synthesis within an ESL design methodology

High-level synthesis is a method of transforming the algorithmic descriptions into RTL designs, HLS is a role inside the ESL design method.

Inside the ESL design, there are three types of HLS:
1. Functional component synthesis, which is based on function-based ESL method, in this method, even if the communication constraints between different components are diverse, but the synthesis process for each component can be run synchronously. 2. Co-processor synthesis, which is based on architecture based ESL method. this type of HLS has the distinct feature as part of the application will execute on a programmable processor,and it acts as a software, this part also called the hardware accelerator. 3.

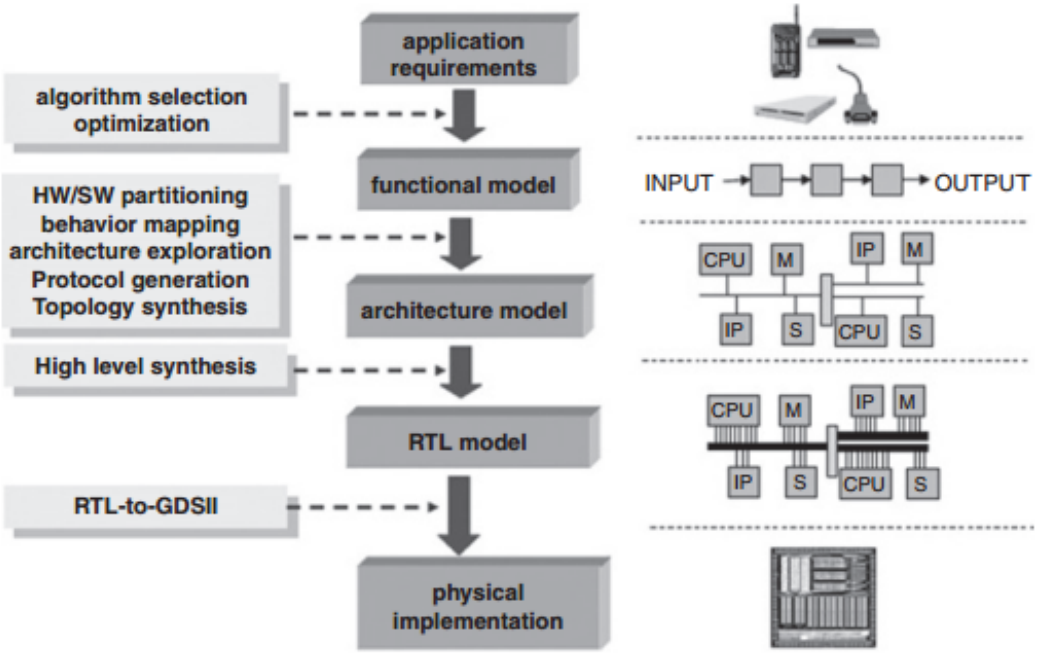

FIGURE 1.3: the role of HLS in ESL design

Application processor synthesis In ESL design, large RTL design can be done by constructed by small RTL designs, as the hardware logic is always the most costly part in all chip design, so HSL method has to be considerable more and more in the emerging design flow.

1.3 High level synthesis

In one word, the HLS takes an program that written in high level language like C as input, then generate an implementation that written in hardware description language like VHDL as output. In the algorithmic description of the program, we use algorithmic written by high level languages such as systemC, C/C++ capture the behavioral-level (high level) description of the design; In the output of integrated circuits the VHDL and Verilog language express the RTL description of the design.

1.3.1 High-level synthesis Input/Output

Giving a program sample with double loops to show the high level synthesis input and output. As it shows below, the program was written in c language

```
int A[100], B[100];
int sum , i;
sum = i = 0;
while (i < 100 )
{
sum = sum + A[ i] * B[ i];
i = i + 1;
}
```

In this behavioral description, there is no any hardware implementation detail,but the program includes these statements, variables and loop descriptions.

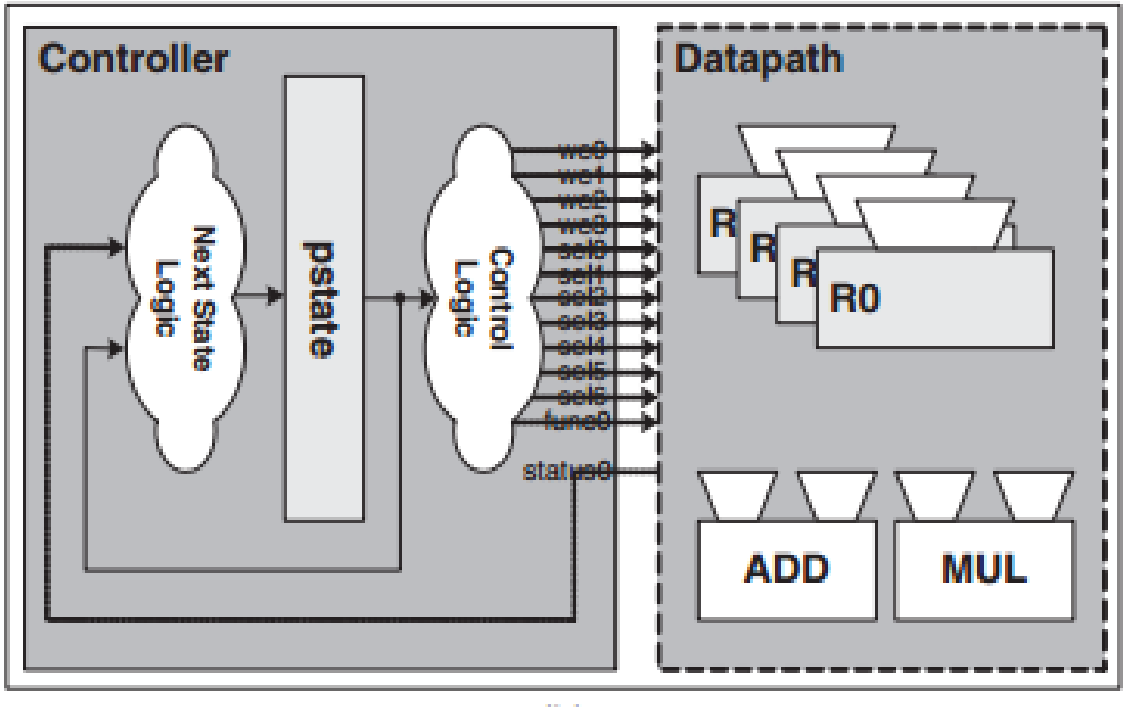

FIGURE 1.4: typical high level synthesis output

In this figure 1.4, its a typical RTL description output of HLS, It consists a controller and a datapath, while the controller represent by the finite state machine model. The controller follows algorithmic behavior to sequence the design, and the datapath consider the data types to perform the computations. The controller output a set of signals to the data path and receive a set of the status signal from data path as input; the data path combines by a set of registers, function units, and multiplexers. In one word, the controller decides when registers in the data path should be executed by sending control

signals, and the data path performs register transfers, or computation, data processing, data storage between registers.

1.3.2 Typical high-level synthesis design flow

Here a complete HLS design flow will be specified, includes behavioral description, RTL representation, scheduling, code generator, etc.The full procedures is shown in the figure.

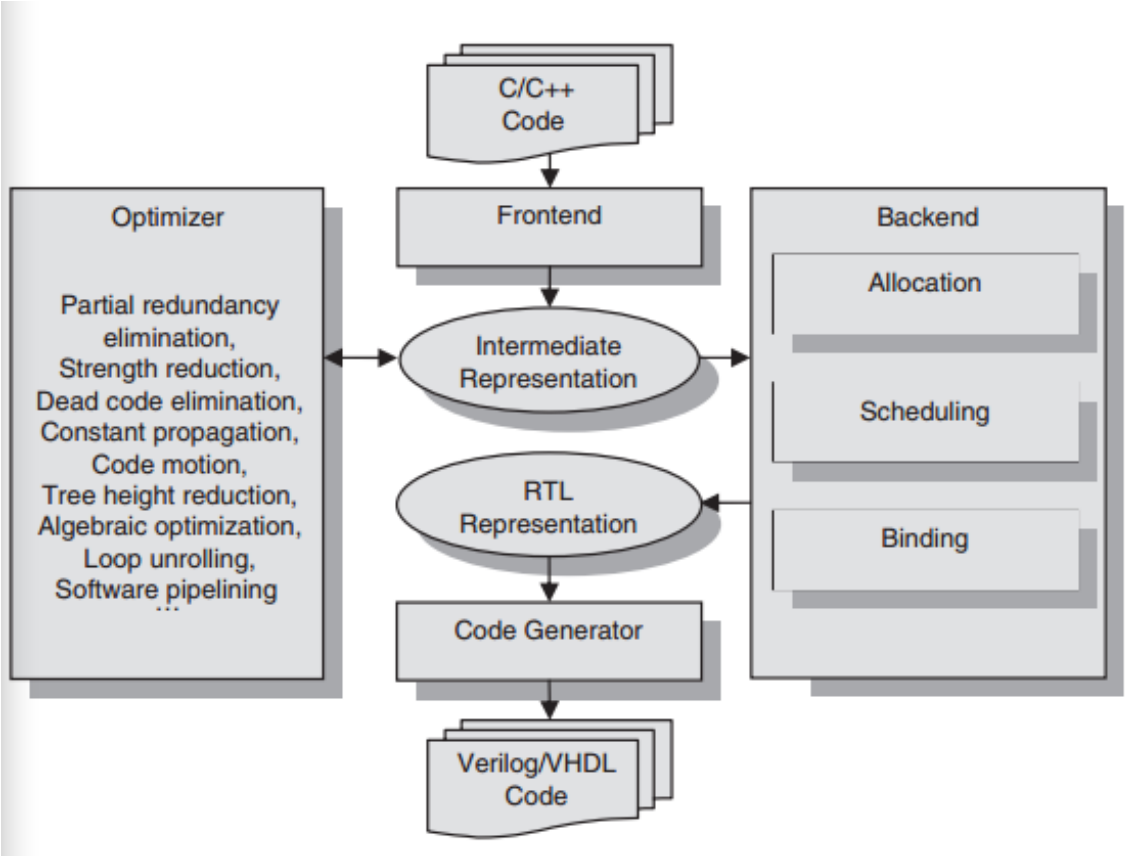

FIGURE 1.5: Typical high-level synthesis flow

The first part is the program algorithm that written in C/C++ language, the second part is the front-end which analysis the behavioral program code to build the third part as an arithmetic and logic computation based operations as intermediate representation(IR). The fourth part is an optimizer, which analysis the IR to eliminate the redundant information and extract the needed information and improve IR. Different types of optimization methods like unrolling, pipelining are used to simplify the code and improve the algorithm. The most important part of high-level synthesis is backend, which performs the allocation, scheduling, and binding operations to transform the IR into an RTL representation. Inside the three steps, the first step allocation allocates the

hardware resource to implement the operation within the IR; The second step scheduling these operations into limit clock cycles, and in order to ruduce the timing, some operations can map into a single clock cycle as we can use pipeline optimization; the third step binding, which maps each operation into a function unit. The six-part is also the final part, called code generator, which generates the Verilog/VHDL code that is used for RTL and logic synthesis. The output code is infinite state machine mode. In conclusion, HLS can make the RTL design within one tool, and generate the RTL output automatically, which allows the designers concentrate on the behavioral level, thus results in a large gain in design productivity.

1.4 Thesis Organization

In chapter 2 some fundamental concepts about tools used for high-level synthesis in the thesis, such as Vivado_hls V.S Catapult.

Chapter 3 specify an example of HLS application, the example based on the algorithm of KNN and digit recognition application

In chapter 4, The Comparison of Experimental Results of Acceleration generated by vivado_hls and catapult

In chapter 5, Conclusion and Future Work

Chapter 2

Vivado_HLS V.S Catapult

2.1 Background

Since using Hardware Description Language(HDL) as a method of hardware design is a very complex procedure, and its a time consuming for the designers perform handwritten register transfer level(RTL). In order to make the hardware devices development more economy in terms of time, HLS is becoming more and more relevant. HLS process can operate in the abstract level that transforms an c/c++ description programs into the ideal hardware implementation(VHDL). Compile the input program's code by adding hardware constraints and then synthesize the input into RTL format. So hardware designers can design at a high description level, not at the hardware level, and can avoid hardware details at design time. HLS will involve a series of steps, such as allocation, scheduling, binding, and RTL generation. these steps decide the resource utilization, FSM behavior in the controller, variables mapping related to the hardware components, vhdl codes for the RTL implementation. During these steps, time constraints and resource constraints will apply and finally, the throughput will be compared through different types of synthesis.

With the development of HSL in recent decades, different companies have developed different HSL tools for design, testing, and debugging. They have their own advantages and disadvantages. , several HLS tools are introduced such as

1.Catapult

2.Vivado HLS

3.Cynthesizer

4.Synphony C

but in this project, I will specifically use Catapult and Vivado HLS.

2.2 Vivado_HLS

2.2.1 Overview of Vivado_HLS

The Xilinx Vivado_HLS is an HLS tool that it can be implemented directly using high-level language C/C++ code to program the programmable device like FPGA, which can accelerate the IP creation, during the process, RTL can be created automatically. Here its design flow:

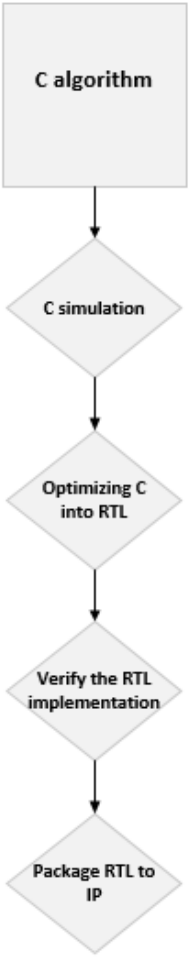

FIGURE 2.1: Vivado_HLS design flow

1.In C simulation step, run C simulations to check the functions written in C, C++, System C is functionally correct. 2.In optimizing c into RTL implement step, using different types of optimization directives, and generate reports. 3.In Vivado HLS, there is a pushbutton to verify the RTL automatically, and it shows the warning and errors. 4.In the package RTL to IP step, package the RTL implementation into IP block, using logic synthesis, IP can be synthesized into an FPGA bitstream.

2.2.2 Base knowledge of Vivado_HLS

Different from other C programs, in Vivado HLS design, the top-level function is the sub-function below main(). And a tech bench is highly recommended to be added to verify the top-level C function and Verify the RTL output. In RTL verification step, the program returns zero to main()if the RTL is functionally identical.In the RTL export step, RTL output files will embedded into IP, late in Vivado, the vivado IP catalog can be used in Vivado Design Suite.

2.2.3 Vivado_HLS performance improvement methods

Vivado HLS has several ways of improving performance, like adding directives, optimizing latency, optimizing throughput, and remove performance bottlenecks. Here is a small function as an example:

```
void func_top(a,b,c,d,*x,*y)
{
func_A();
func_B();
func_C();
func_D();
return res;
}
```

If the function A,C,D take 2 cycles,and B take 4 cycles,as the latency is defined as the clock cycle period between the initial input and output, the throughput is defined as clock cycles count until a new input, so if there is no concurrency, the latency should be as same as throughput,equal to 10 respectively. In Vivado HLS, if the pipeline is applied, then the throughput will be improved to 4 cycles.As it shows:

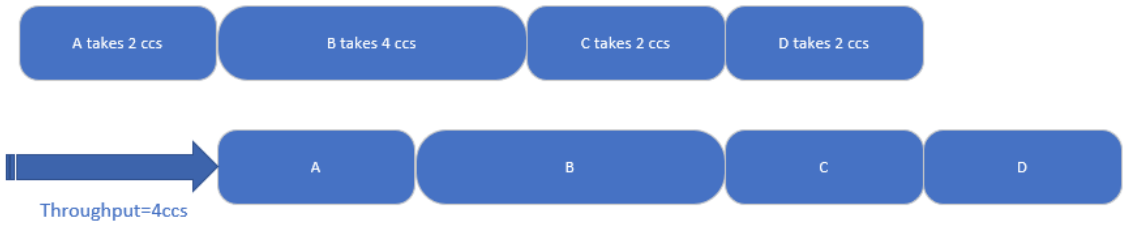

FIGURE 2.2: throughput optimizing by pipeline

2.2.3.1 Adding directives

In the first step, open the source code in the information pane, and then for the interface element, by using add directives, we can apply ARRAY_MAP,ARRAY_PARTITION,ARRAY_RESHAPE and STREAM; For each loop inside the top-level function, by using add directives, we can apply ALLOCATION, DATAFLOW, DEPENDENCE, LOOP_FLATTEN, LOOP_MERGE, LOOP_TRIPCOUNT, PIPELINE, UNROLL,etc..after applying directives respectively, the performance may improve.

2.2.3.2 Reduce latency by loop_unroll

In Vivado HLS, the loops remains rolled by default, and it means there is only one entity for these loops, the synthesis steps performed in the same hardware resource.Here is an example:

for(i=5; i>0; i–)

{

a[i] = b[i] * [i];

}

when the loop is rolled, for the hardware requirement, one multiplier and a single port block RAM can satisfies the execution of each iteration,but each iteration has the demanding for one clock cycle, so at least four clock cycles are needed for the implementation. But if I make the unroll the loop and make the partition rate set as factor 2, then for each clock cycle, two reads and two writes operation can execute parallel, which only require two multipliers and dual port RAM hardware, but the advantage is that

the implementation need takes 2 clock cycles to complete. When fully unroll the loop, if there are abundant hardware resource, each clock cycle can execute all the operations, but the implementation requires four multipliers, as well as 4 reads and writes operations executed in one clock cycle, but the single block RAM has 2 port only, so the arrays need to be partitioned.

2.2.3.3 Reduce latency by loop_flatten

Vivado HLS can automatically flatten the innermost loop, thus reducing the clock cycles between inner and outer loop. Here is the example:
Outer loop: while(j<10){
Inner loop: while(i<3){ 1 cycle to enter inner loop
...
LOOP_BODY
...
} // 1 cycle to exit inner
}
in this example, entering the inner while loop takes 1 cycle as well as exit from the inner while loop, is needed to enter inner, and 1 cycle to exit inner, so for this function, if it didnt flatten, the execution of the outer loop requires 20 extra clock cycles. But if loop_flatten is being in the Vivado HLS, the extra clock cycles will be saved.

2.2.3.4 Reduce latency by loop_merging

The loop_merge optimization directive can automatically merge loops.As it shows from the example below, this rolled loop needs 11 ccs by default, but after applying merge directive, the ccs will be reduced to 6 to execute the loop.

but there are restrictions such as FIFO access cannot be merge.

2.2.3.5 Dataflow optimization for optimizing throughput

In the top-level function, if there are a series of sequential tasks such as sub-functions, loops, Dataflow optimization can be applied.

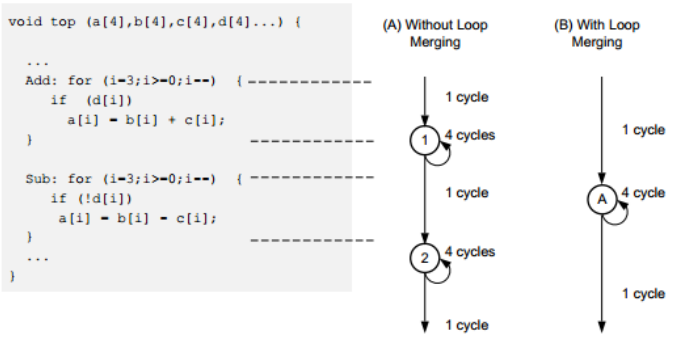

FIGURE 2.3: loop merger

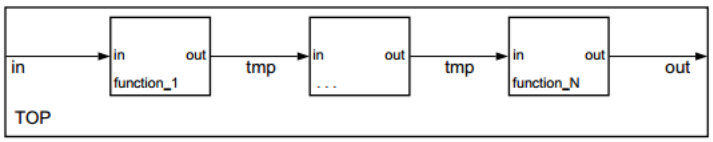

FIGURE 2.4: sequential tasks inside top function

After applying dataflow optimization, a parallel processor architecture will be created.Between these processes, it exists some channels, the channel can ensure the task can execute directly instead of spending time to wait the previous task has completed all its operations. Thus increasing the throughput of the design and reduce the latency.Here its figure

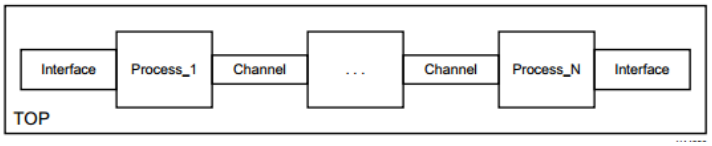

FIGURE 2.5: parallel process inside top function

Here is a latency and throughput comparison with and without dataflow optimization.

As the figure shows, there are three functions inside this top-level function, if without dataflow pipelining, the latency would be 8 cycles as well as throughput; but if the data flow pipeline is applied, the latency will reduce to 5 cycles, and the throughput will reduce to 3 cycles. But for the Dataflow optimization, it is strictly requested that the data flow follows the rule as shift from one task to the next one. Some limitations such as bypassing tasks, feedback between tasks, loops with multiple exit conditions may prevent Vivado HLS perform this optimization.

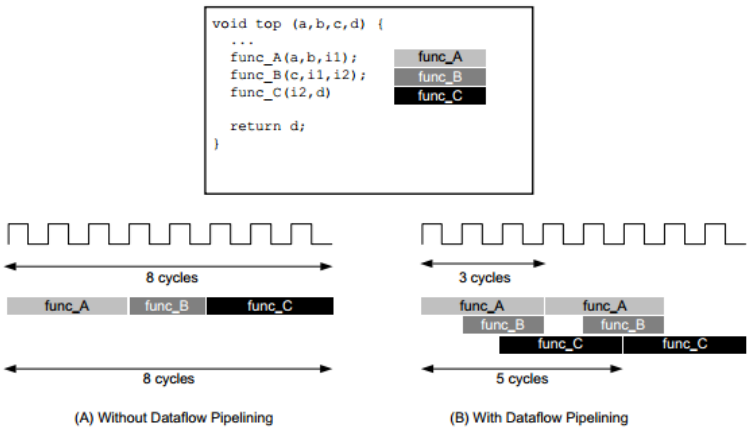

FIGURE 2.6: with and without data flow inside top function

2.2.3.6 Pipeline optimization

As it is mentioned above, dataflow pipelining has the limitation of working only on functions or loops at the top level of the hierarchy, and cannot be used in sub-functions.When there are sub-loops inside the function or loop, we may implement function or loop pipelining.In Vivado HLS, when pipeline the loop, all the sub-loops will be unrolled, but there is one thing need to notice, the inner loop with variable bounds cannot be unrolled.

2.2.3.7 Performance Bottleneck

Even there are several ways to remove bottlenecks in order to improve the performance of high-level synthesis. 1.Array Partitioning In array partitioning technique, the array partitioning can break an array into smaller elements, but all the partitions have the same resource target. 2.Array Dimensions In array dimension, an array can be divided into small parts, such as an example:
array[10][6][4] can be divided into:
array_0[10][6]
array_1[10][6]
array_2[10][6]
array_3[10][6]
3.Array Reshaping This directive combines array_partitioning with array_map to reduce

the number of block RAM. So this directive allows more data to be accessed in a single clock cycle.

2.3 Catapult

2.3.1 Overview of Catapult

The Catapult is a platform designed by the Mentor, which can help the designer to describe functions and move to an abstraction level.It Has three main parts in sense of high-level synthesis: 1. C/C++/SystemC HLS The Catapult is the only natively high-level synthesis platform that supports the high-level description by ANSI C++ and System C, and then based on thesis codes written by an abstract level description language such as C, Catapult can generate optimized Verilog or VHDL, ready for production of RTL synthesis and verification flows. 2.HLS verification In Catapult, there are three types of verifications, the first is checking user's C code before synthesis in order to find errors; the second is verification during simulation, comparing the functionality of users C source with generated RTL; the third is verify the code with the RTL from Catapult design checks. 3.Low power HLS Catapult low power(LP) is a tool that targets power as an optimization goal.The designer can use Catapult LP to explore different hardware architecture and measures the power, performance, and area of each solution.

2.4 comparision

For these two tools, both inputs are C/C++ SystemC codes, and both outputs in the form of Verilog, VHDL, SystemC, but Catapult don't support for float point arithmetic.And the performance such as timing, hardware will be shown in chapter 4.

Chapter 3

KNN and Digit Recognition Application

In this chapter, the theory of KNN and digit recognition application (DCA)will be specified respectively.The algorithm of KNN and DCA will be applied synthesis performance in Vivado HLS and Catapult respectively, so explanation the principle of the KNN and DCA is necessary.

3.1 KNN

There are lots of algorithms that used in machine learning field, KNN is one of the simple algorithms. KNN is a classification by measuring the distance between different eigenvalues. Its train of thought is: Giving a test point, and calculating the distance between the test point with neighbor points, then sort the k most near points in terms of distances,if the k points have the same category, then the test point can be predicted belong to this kind of category,as for the k value which is usually no more than 20 k integers. In KNN algorithm, the k nearest neighbors' type has already known in advance. Based on this method, by using the k nearest points category, the sample's category can be defined. In one world, a new instance can be classified by its 'K' neighbors' majority class. Based on this character, KNN can be used for classification, estimation, and prediction.In this algorithm, the large amount of training data is set, a undefined category test point can be determinated easily by inputing test data, the

characteristics of the test data and training focused on to compare the characteristics of the corresponding, and find the most similar of training focus and former K data, then the test data, the corresponding category k nearest neighbors category. Here is the algorithm

1) computing the Euclidean distance between the test data and sets of training data; 2) sorting according to the increasing in terms of distance; 3) choose K points with the smallest distance; 4) determine the occurrence frequency of the category of the previous K points; 5) using high occurrence rate category of the nearest k points to predict the type of the test point. The KNN algorithm is expressed as[3]:

```
for all the unsorted points(i)\\
   for the every known training or reference points(j)\\
         calucalte the Euclidean distance between unsorted point(i)and reference points(j)\\
         and save the in dist[j]\\
     end for\\
     sort and find the k points with smallest distances\\
     locate the types of the samples k points,like point(j1),..,point(jk)\\
     predict unsorted point(i) to the class has the high occuren in the k sample points\\
end for\\
```

3.1.1 Example of KNN

Suppose there are a group of data in the figure,and these data are divided into two different types.The first type of data is shown in the form of blue cross,as it shows below:

and the second type of data:

Based on the coordinate of each point on the data 1 and data 2, there are several steps for KNN algorithm.

From the figure above, if we want to predicate the type of one point in the figure, the better way is to find it k-nearest neighbor, if the majority of the k-nearest neighbor points belong to one cross type, then we can predicate this point belongs to the cross type. But some issues come with the example above, these issues include: 1.How is the K value should be chosen? That is, how many neighbors should be considered? 2.how to measure the distance?

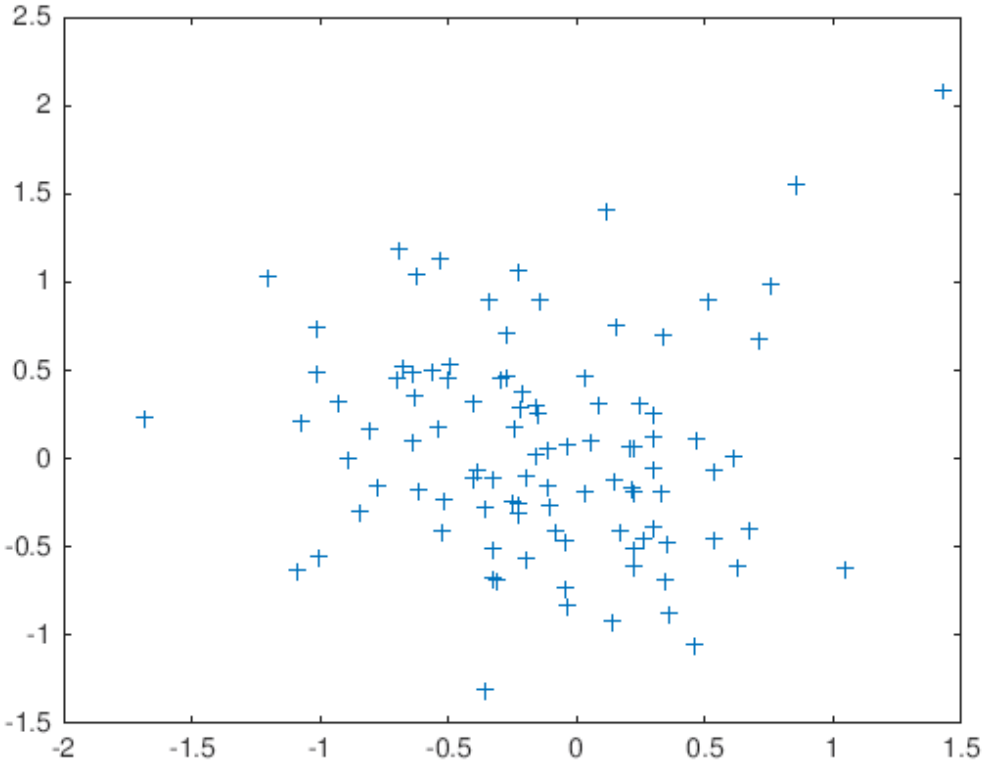

FIGURE 3.1: First type of data

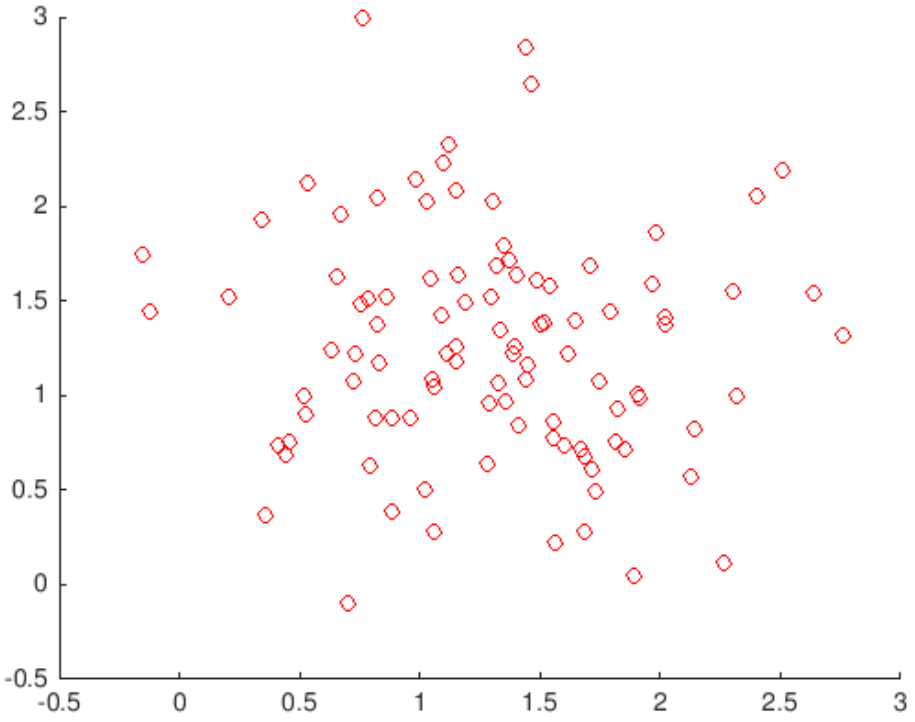

FIGURE 3.2: Second type of data

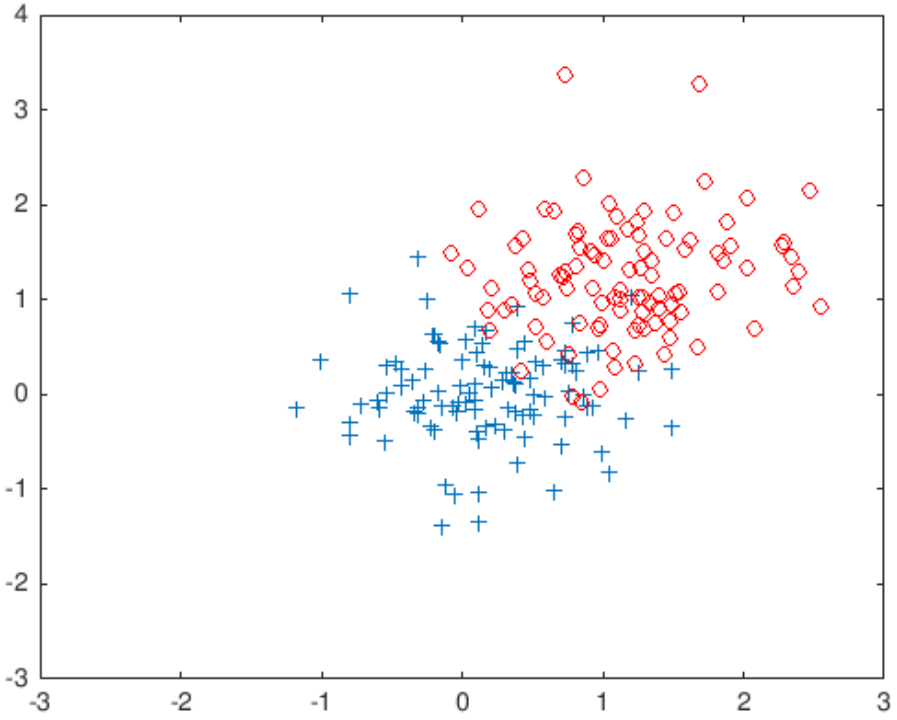

FIGURE 3.3: The effect of two data combination

3.1.2 Distance Function

In math, the distance in the space is defined by

If M is a space, and function d is a distance function in M, and it satisfies

1.d(x,y) $\geq$0, and d(x,y) = 0 if and only if x = y;

2.d(x,y) = d(y,x);

3.d(x,z) $\leq$ d(x,y) + d(y,z);

Propery1 indicate that the distance is always positive, property 2 indicates commutativity, property 3 states that the distance between the two should be the shortest distance, a third point can never shorten the distance between these two points. The most useful distance function is Euclidean distance, it represents as:

$d_{Euclidean}$(x,y) = [$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$]

3.1.3 Sorting methods

In order to find the k-nearest values, sorting should be applied to the distance array dist[j], after applying sorting, the K smallest value will be store to the sort_distance[k],

and based on majority class of sort_distance[k],the class or the type of the unknown point can be predicted.Here is list of different type of sorting methods:

Bubble Sort: Bubble sortBubble sort, the basic idea is: starting from the chaotic sequence in the head, two comparison, according to the size of the swap places, until finally the maximum (small) exchange of data elements to the disorder of the queue, and become part of an orderly sequence; The next time you continue this process, until all the data elements are sorted. The core of the algorithm is to select the largest (small) data elements of the remaining unordered sequences at the end of the queue each time through two or two comparisons.

(2) The operation of bubble sort algorithm is as follows:

1. Compare adjacent elements. If the first one is bigger than the second (small), swap them both.

2. Do the same work for each pair of adjacent elements, from the first pair to the last pair at the end. When this is done, the final element will be the largest (small) number.

3. Repeat the above steps for all elements, except for the ones that have been selected at the end.

4. Repeat the above steps for fewer and fewer elements (unordered elements) each time, until no pair of Numbers need to be compared, and the sequence is finally ordered..This algorithm is suitable for small data sets as complexity is O(n^2) if n stands for the number of elements.

Insertion Sort: Each time an ordered record is inserted, the appropriate place in the sequence of subsequences is inserted by its keyword size until the full record is inserted. Set the array as a[0... n-1]. 1. At the beginning, For I = 1, a[0] becomes an ordered area, and the disordered area is a[1..n-1].

2. A [I] is incorporated into an ordered range of a[0... I].

3. I ++ and repeat step 2 until I ==n-1. Sort done..Similar to bubble sort, the complexity is O(n^2) if n stands for the number of elements.

Merge Sort: Merge sort Refers to the operation of combining two sorted sequences into one sequence. Merge sort algorithm depends on merge operation. Merge sort has multiple merge sort, two merge sort, can be used for inner sorting, and can also be used for external ordering., it has the complexity of (n log n), so this sort method is the most respected algorithms.

Quick Sort: Fast sorting based on divide-and-conquer, The divide-and-conquer process for A typical subarray A[p...r] is three steps: 1. Break down:

A[p..r] is divided into two (possibly empty) subarrays A[p.. q-1] and A[q+1..r].
A[p. Q -1] ¡= A[q] ¡= A[q+1..r]
2. Solution: quicksort by recursive call, and sort the subarray A[p..q-1] and A[q+1..r].
3. The merger: In this sort algorithm, the complexity is O(n^2), same as bubble sort and insertion sort.

3.1.4 Choice of K

Different K values have a significant impact on the results of KNN classifier[4], the choice of K should be carefully as it plays an important role .The choice of K value will affect the prediction in both positive and negative way. If the K value is very small, the reduction of K means that the model becomes more complex and the correctness rate tends to be unstable ; but if the K is a large value, the large value of K makes the performance decreases as the consideration of other classes.

3.2 Digital Recognition Application

Among the number of algorithms in machine learning, KNN is always used in recognition pattern.A K-NN classifier can predict a class based on its k-nearest neighbors.The term nearest is considered by the distance while the k is considered as the number of K nearest values.So the KNN algorithm is quite useful in the application of digit recognition. For the job of thesis, two groups of dataset will be introduced in this application.
trainDigits: a collection of 1924 instances with known class labels which will be used as training set;
testDigits: a collection of 845 instances with known class labels but the labels will not be used by the classifier, instead, the labels will be used as a test set to check the accuracy of the predictions. Each class for each data is represented by two-dimensional arrays of 10 integers, the X-Y coordinates are represented by 32*32 space.Each point in the two-dimensional arrays is represented by 0 or 1.Here is an example of a test digit.
In the left parts, I will explain the use of KNN algorithm in digit recognition in several steps.

```
00000011111110000000011100000000
00000011111100000000011110000000
00000011111100000000001110000000
00000011111100000000001110000000
00000001111110000000000111000000
00000001111110000000000111000000
00000001111110000000000111000000
00000001111110000000000111000000
00000001111110000000001111000000
00000001111011000000001111000000
00000001111000000000011110000000
00000001111000000000011111000000
00000001111000000000011111000000
00000001111000000000111110000000
00000001111000000001111100000000
00000000111000000111111000000000
00000000111100011111110000000000
00000000111111111111110000000000
00000000011111111111100000000000
00000000011111111111100000000000
00000000001111111110000000000000
00000000000111110000000000000000
00000000000011000000000000000000
```

FIGURE 3.4: An example of testDigital(label 0)

3.2.1 Loading the dataset

In the step1: The training file should be read first by using the function.In this function, the first two loops read all the files as the first loop focus on reading the label while the second loop focuses on reading the order number of each label.The third and the fourth loop upload the binary number in each file.As it can be seen from the example of the test digit, each image size is equal to image_edge*image_edge, and each image_edge size is 32, so the image size for each file is 32*32 as 1024.Here is function of read train code(train_code),the code is shown as:

In the step2: Then back to the testbench, the outer 2 loops are used to read the

```
void readtraincode(bool b[N_TRAIN][IMAGE_SIZE])
{

    int counter = 0;
    for(int x = 0; x<NUM_CLASS;x++)
    {
        for(int y = 0; y<M_TRAIN;y++)
        {
            sprintf(traindigit,"/home/guo/Desktop/TESI/DigitRecognition/DIGIT_RECOGNITION/digitRecognition_kNN/distancesorted/trainingDigits/%d_%d.txt",x,y);
            FILE *fp = fopen(traindigit,"r");
            if(fp==NULL)
            {
                printf("OPEN FAILD\n");
            }
            for(int i=0;i<IMAGE_EDGE;i++)     //i means the number of rows in the text file
            {
                for(int j=0;j<IMAGE_EDGE;j++)  //j means the number of colums in the text file
                {
                    char temp;
                    int charvalue = fscanf(fp,"%c",&temp);
                    b[counter][i*IMAGE_EDGE+j]=temp-'0'; //put the character(eg 0,1)to an array, from 0 to 32,33 to 64,...IMAGE_SIZE,so the total length of a
                }
                char temp;
                int skip = fscanf(fp,"%[^\n]%*c",&temp);
            }

            //   printf("%d",b[0][x*32+0]);
            fclose(fp);
            counter++;
        }

    }
```

FIGURE 3.5: Loading the training file (label 0)

test data file one by one, after access each file of the testdigit, using the function of readtestcode(test_code, testdigit) to read each binary number of each files.

3.2.2 Distance Calculation

In the step3, after reading the data from both training and testing file, then the distance should be calculated.In our case in this paper, the Euclidean distance would be considered.

3.2.3 Distance Sorting

After calculating the distance, the top k minimum values should be found by sorting.In this stage, the KNN algorithm would be applied.

3.2.4 Count the occurrence

After sorting the top k minimum values based on the distance array, the occurrence of the class of these k values should be counted.If the class occurrence of one of these classes is greater than other classes occurrence, then we can predicate the type of the testing file. [**?**].

3.2.5 Verify the recognition rate

After the count the occurrence of the predicted class, use the division value between the occurrence of predicted and the number of k, then we can get the ratio of the division, we can call this ratio as the recognition rate. Higher of the recognition rate, the higher of success rate. [**?**].

3.3 Summary

The KNN based digit recognition application is a simple example of machine learning.In the consequent chapter, the high-level synthesis based on this application will be analyzed.

Chapter 4

The Comparison of Experimental Results of high level synthesis Acceleration

In this chapter, I will perform the experiment in the platform such as Vivado_hls, Catapult and Vivado, and at the same time, each experiment will be done in series different ways, and the result of the comparison will be shown in the following content.

4.1 Topkref_Distances

The Topkref Distances will be performed in both Catapult and Vivado HLS. There are two main steps in this top function, the first step is calculating the distance between the points of training file and testing file; The second step is finding the max value of the in the distance array. In the Vivado HLS, I applied the 10 different optimization methods for optimizing the function, these methods will be explained in the below section.

4.1.1 Topkref implemented in Vivado hls

The optimization will be done in three main aspects, named as interface aspect, loop aspect, architecture aspect.Here are the details that related to these aspects.

4.1.1.1 Interface Aspect

I_ARRAY: In order to improve the performance of high-level synthesis, the array argument would be implemented as some different types of RTL ports.In my design, I specify the dual ports RAM interface for input reading and specify the RESOURCE as the type of the RAM that connected to an interface.So I select RESOURCE in the directive editor and click the core option and select RAM_2P_BRAM.

I_FIFO: FIFO is a special array accesses,FIFO means the order of access the array is based on the sequential rule that entry start from zero, especially for these arrays that should be read from multiple locations, should be following the FIFO order.

I_FIFO_FULLPARTITION: For large arrays implemented in block RAM which has a limited number of ports, it may reduce the synthesis performance.In order to improve the performances, the array can be partitioned into small arrays, while at the same time, more registers would be used as the result of large array partition.In the fully-partition, we modify the partitioning type to complete.

I_FIFO_PARTITION: As compared to the above case, I modify the options type partition to block.In this case, the array was partitioned into some small arrays, but not completely partitioning.

I_AXI: In high-level synthesis, there is another type of interfaces named as AXI4, generally speaking, almost any kind of inputs an array or pointer output argument can use AXI4 interface. AXI4 interface has the ability transfer data in sequential streaming way, and three kinds of registers mode are used in the AXI-Stream interfaces, respectively as

Forward: the TDATA and TVALID signal.

Reverse: the READY signal is registered.

Both: All signals TDATA,TVALID,READY are registered.

In high-level synthesis design, the AXI4-Stream can be used in two different model, respectively as side channels and without side-channels.In this thesis, we use the AXI4-Stream interfaces without side-channels.Here is an example:

#pragma HLS INTERFACE axis port=A

4.1.1.2 Loop Aspect

p_pipeline: There are two loops in the code, I respectively apply the pipeline directive to the inner and outer loop in order to reduce initiation interval.After applying pipeline optimization, multiple operations within a function can be executed concurrently. p_unroll: For the inner loop, I unroll the for loop to generate multi independent operations instead of a single collection of operation.

4.1.1.3 Architecture Aspect

A_Dataflow: in the architecture level, allows the functions and loops execute concurrently by allowing task level pipeline. Inline: When inline a function, the function hierarchy can be removed.By reducing function call and function boundaries, the latency and interval can be reduced.

4.1.2 The result of top ref implemented in Vivado hls

Performance Estimates

- **Timing (ns)**

Clock		A_dataflow	I_AXI	I_FIFO	I_FIFO_FULLPARTITION	I_FIFO_PARTITION	Inline	P_inner_unroll_out	P_outer	p_inner
ap_clk	Target	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	Estimated	6.79	8.75	7.18	7.18	7.18	7.18	7.18	7.18	7.18

- **Latency (clock cycles)**

		A_dataflow	I_AXI	I_FIFO	I_FIFO_FULLPARTITION	I_FIFO_PARTITION	Inline	P_inner_unroll_out	P_outer	p_inner
Latency	min	51351	102477	51261	51261	51261	51261	51261	51261	51499
	max	51351	102477	51261	51261	51261	51261	51261	51261	51499
Interval	min	51352	102478	51262	51262	51262	51262	51262	51262	51500
	max	51352	102478	51262	51262	51262	51262	51262	51262	51500

Utilization Estimates

	A_dataflow	I_AXI	I_FIFO	I_FIFO_FULLPARTITION	I_FIFO_PARTITION	Inline	P_inner_unroll_out	P_outer	p_inner
BRAM_18K	4	5	1	1	1	1	1	1	1
DSP48E	0	0	0	0	0	0	0	0	0
FF	118	2251	345	412	352	352	351	351	408
LUT	606	2880	937	994	982	982	957	957	1031

FIGURE 4.1: Timing and Latency and Utilization comparison

Studies have shown that using different directives, the performance will be the difference.In terms of timing, if apply the A_dataflow, it uses the minimum time;In terms of

Resource Usage Implementation

	A_dataflow	I_AXI	I_FIFO	I_FIFO_FULLPARTITION	I_FIFO_PARTITION	Inline	P_inner_unroll_out	P_outer	p_inner
RTL	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	-	0	0	0	0	0	-	0	0
LUT	-	1424	336	317	350	350	-	341	334
FF	-	1807	293	334	300	300	-	299	327
DSP	-	0	0	0	0	0	-	0	0
SRL	-	87	0	0	0	0	-	0	0
BRAM	-	3	1	1	1	1	-	1	1

Need to run vivado synthesis/implementation to populate the real data for "-"

Final Timing Implementation

	A_dataflow	I_AXI	I_FIFO	I_FIFO_FULLPARTITION	I_FIFO_PARTITION	Inline	P_inner_unroll_out	P_outer	p_inner
RTL	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl
CP required	-	10.000	10.000	10.000	10.000	10.000	-	10.000	10.000
CP achieved post-synthesis	-	4.557	4.553	4.553	4.549	4.549	-	4.553	3.320
CP achieved post-implemetation	-	-	-	-	-	-	-	-	-

FIGURE 4.2: The resource and final timing implementation

latency, if apply I_AXI, the value of latency and interval will increase;In term of utilization, the I_AXI uses the max resource; For the final timing implementation, when pipeline the inner loop, the critical path achieved post-synthesis get the minimum value.

4.2 Topkref implemented in Catapult

In the previous section, the top function of top ref was discussed in the synthesis tool of vivado_hls, and in this section, the synthesis process will be implemented in Catapult, and then I will compare the synthesis results in the two different tools.

4.2.1 Synthesis by setting the top function

In this subsection, for the hierarchy setting of the function of "void topkref_distances(bool a[IMAGE_SIZE],bool b[N_TRAIN][IMAGE_SIZE],int r_dist[K],int r_index[K])", I choose "Top" out of Inline, Block, Top.

Here is the result of synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	63	153804	153806	1	0	
Design Total:	63	153804	153806	1	0	

FIGURE 4.3: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	315.4	1397.1	622.8
Total Reg:	0.0	0.0	0.0

FIGURE 4.4: The total area post synthesis

4.2.2 Synthesis by setting the outer loop pipeline

In this subsection, I still set the void topkref_distance as the top function, and in the synthesis task of Architecture, I set the topkref_distances_label0 and label loop_pipe as pipeline, the other parts of settings are default from the first subsection. Here is the result of synthesis by Catapult. Here is the result of synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	140	51306	51308	1	0	
Design Total:	140	51306	51308	1	0	

FIGURE 4.5: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	549.3	1553.4	900.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.6: The total area post synthesis

4.2.3 Synthesis by setting the outer loop pipeline and unroll

In this subsection, the first loop topkref_distances_label0 will be set as pipeline, and the second loop_pipe will be set as unroll, and the other parts of setting are default from the second subsection,here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	1215	51353	51355	1	0	
Design Total:	1215	51353	51355	1	0	

FIGURE 4.7: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	3528.8	19972.5	10113.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.8: The total area post synthesis

4.2.4 Synthesis by setting the outer loop pipeline and inner loop pipeline

In this subsection, the second loop topkref_distances_label0:for, loop_init, loop_unroll and loop_assign will be set as pipeline, and the other parts of setting are default from the second subsection,here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	140	51305	51307	1	0	
Design Total:	140	51305	51307	1	0	

FIGURE 4.9: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	549.3	1578.3	930.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.10: The total area post synthesis

4.2.5 Synthesis by setting the data enable interface

From this subsection, the interface synthesis in Catapult will be explained. In these three sections, SCVerify flow will be used, as the SCVerify function is proving the C++ algorithm and RTL function equivalent by giving the same input vectors. In the blow sections, I will use different methods to control the flow of data, enable signal and wait interfaces. In the first synthesis method, I click on the Flow Manager tab and make sure that SCVery is enabled.For the output interface r_dist and r_index, I choose mgc_out_stdreg_en. Here is the result after synthesis:

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	140	51306	51308	1	0	
Design Total:	140	51306	51308	1	0	

FIGURE 4.11: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	549.3	1553.4	900.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.12: The total area post synthesis

4.2.6 Synthesis by setting two way handshake interface

In this subsection, the interface will shift to two-way handshake, even the data is not available, but the catapult block can be installed. I set the interface as mgc_ioport.mgc_inout_buf_wait, here is the result after synthesis.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	140	51306	51308	1	0	
Design Total:	140	51306	51308	1	0	

FIGURE 4.13: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	549.3	1580.4	929.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.14: The total area post synthesis

4.2.7 Synthesis by adding block size in the interface

In this subsection, the setting is same as the second case, but the block size will increase from 0 to 1, here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	136	51306	51308	1	0	
Design Total:	136	51306	51308	1	0	

FIGURE 4.15: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	497.3	1335.5	852.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.16: The total area post synthesis

4.2.8 Synthesis by creating c-cores in the mapping process

In this subsection, the c-cores method will be used.C-CORE is a user-defined operation that consists of a collection of one or more operators.It can improve the area by minimizing the MUX sharing logic and reduce the run time by reducing the number of variables. For applying the c-core methodology,I first modify the code then in the mapping step, select the design type as CCORE, and then select combinational block. Here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topkref_distances/core	63	153804	153806	1	0	
Design Total:	63	153804	153806	1	0	

FIGURE 4.17: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	315.4	1397.1	622.8
Total Reg:	0.0	0.0	0.0

FIGURE 4.18: The total area post synthesis

4.2.9 Result comparision for Topkref distance in Catapult

As applying different types of synthesis methods, the result is compared in the following table:

Solution	Latency C...	Latency T...	Throughp...	Throughp...	Total Area	Slack
topkref_distan... (extract)	153804	461412.00	153806	461418.00	622.83	-2.72
topkref_distan... (extract)	51306	153918.00	51308	153924.00	899.96	-3.06
topkref_distan... (extract)	51305	153915.00	51307	153921.00	929.96	-3.06
topkref_distan... (extract)	51305	153915.00	51307	153921.00	929.96	-3.06
topkref_distan... (extract)	51305	153915.00	51307	153921.00	929.96	-3.06
topkref_distan... (extract)	51305	153915.00	51307	153921.00	1005.96	-3.06
topkref_distan... (extract)	51305	153915.00	51307	153921.00	1005.96	-3.06

FIGURE 4.19: The total area post synthesis

4.3 Topksorted_Distances

Similar to topkref distances application, the topksorted distances will also be implement in Catapult and Vivado HLS in the same method.For the function topksorted_distances,the first step is calculating the distance between the points of training file and testing file; The second step is finding the sorted K minimum value in the training file. In the Vivado HLS, I applied the 10 different optimization methods for optimizing the function, these methods will be explained in the below section.

4.3.1 Topksorted implemented in Vivado hls

The optimization will be done in three main aspects, named as interface aspect, loop aspect, architecture aspect.Here are the details that related to these aspects.

4.3.1.1 Interface Aspect

I_ARRAY: In high-level synthesis, the array argument can be implemented as a number of different types of RTL ports.In my design, I specify the dual ports RAM interface for input reading and specify the RESOURCE as the type of the RAM that connected to an interface.So I select RESOURCE in the directive editor and click the core option and select RAM_2P_BRAM.

I_INTERFACE: When the top function is synthesized, the parameters to the functions are synthesized into RTL ports.There are many types of interface synthesis methods,including m_axi,ap_fifo,ap_memory,etc.

I_FULLPARTITION: For large arrays implemented in block RAM which has a limited number of ports, it may reduce the synthesis performance.In order to improve the performances, the array can be partitioned into small arrays, while at the same time, more registers would be used as the result of large array partition.In the fully-partition, we modify the partitioning type to complete.

I_PARTITION: As compared to the above case, I modify the options type partition to block.In this case, the array was partitioned into some small arrays, but not completely partitioning.

I_AXI: There is another type of interfaces that used in high-level synthesis as AXI4, which can be applied to any kind of inputs an array or pointer output argument.AXI4

interface is a kind of interfaces that transfer data in sequential streaming way, and three kinds of registers mode are used in the AXI-Stream interfaces, respectively as
Forward: the TDATA and TVALID signal.
Reverse: the READY signal is registered.
Both: All signals TDATA,TVALID,READY are registered.
There are two ways to use AXI4-Stream in high-level synthesis design, respectively as side channels and without side-channels.In this thesis, we use the AXI4-Stream interfaces without side-channels.Here is an example:
#pragma HLS INTERFACE axis port=A:

4.3.1.2 Loop Aspect

p_pipeline: There are two loops in the code, I respectively apply the pipeline directive to the inner and outer loop in order to reduce initiation interval.After applying pipeline optimization, multiple operations within a function can be executed concurrently. p_unroll: For the inner loop, I unroll the for loop to generate multi independent operations instead of a single collection of operation.

4.3.1.3 Architecture Aspect

A_Dataflow: in the architecture level, allows the functions and loops execute concurrently by allowing task level pipeline. Inline: When inline a function, the function hierarchy can be removed.By reducing function call and function boundaries, the latency and interval can be reduced.

4.3.2 The result of top sorted implemented in Vivado hls

Studies have shown that using different directives, the performance will be the difference.In terms of timing, if apply the A_dataflow, it uses the minimum time;In terms of latency, if apply I_AXI, the value of latency and interval will increase;In term of utilization, the I_AXI uses the max resource; For the final timing implementation, when pipeline the inner loop, the critical path achieved post-synthesis get the minimum value.

Timing (ns)

Clock		INLINE	A_DATAFLOW	I_AXI	I_FULLPARTITION	I_INTERFACE	I_PARTITION	P_INNER_UNROLL_OUT	p_inner	p_outer
ap_clk	Target	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
	Estimated	7.05	8.75	8.75	7.97	7.05	7.05	9.13	8.75	8.75

Latency (clock cycles)

		INLINE	A_DATAFLOW	I_AXI	I_FULLPARTITION	I_INTERFACE	I_PARTITION	P_INNER_UNROLL_OUT	p_inner	p_outer
Latency	min	102808	563685	563635	102808	102808	102808	563759	102744	563426
	max	102808	563685	563635	102808	102808	102808	563759	102744	563426
Interval	min	102809	563686	563636	102809	102809	102809	563760	102745	563427
	max	102809	563686	563636	102809	102809	102809	563760	102745	563427

FIGURE 4.20: Timing and Latency and Utilization comparison

Utilization Estimates

	INLINE	A_DATAFLOW	I_AXI	I_FULLPARTITION	I_INTERFACE	I_PARTITION	P_INNER_UNROLL_OUT	p_inner	p_outer
BRAM_18K	1	5	5	1	1	1	5	5	6
DSP48E	0	0	0	0	0	0	0	0	0
FF	455	2425	2360	10499	455	477	2700	2721	4851
LUT	381	2273	2177	7386	381	417	2684	2413	6086

FIGURE 4.21: The resource and final timing implementation

4.4 Topksorted_distances implemented in Catapult

In the previous section, the top function of top sorted was discussed in the synthesis tool of vivado_hls, and in this section, the synthesis process will be implemented in Catapult, and then I will compare the synthesis results in the two different tools.

4.4.1 Synthesis by setting the top function

In this subsection, for the hierarchy setting of the function of "void topkref_distances(bool a[IMAGE_SIZE],bool b[N_TRAIN][IMAGE_SIZE],int r_dist[K],int r_index[K])", I choose "Top" out of Inline, Block, Top.

Here is the result of synthesis by catapult.

```
Processes/Blocks in Design
  Process                  Real Operation(s) count Latency Throughput Reset Length II Comments
  ------------------------ ----------------------- ------- ---------- ------------ -- --------
  /topksorted_distances/core                    58  154106     154108            1  0
  Design Total:                                 58  154106     154108            1  0
```

FIGURE 4.22: The latency and Throughput

4.4.2 Synthesis by setting the outer loop pipeline

In this subsection, I still set the void topkref_distance as the top function, and in the synthesis task of Architecture, I set the topksorted_distances_label0 as pipeline, the other

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	246.4	1197.8	629.4
Total Reg:	0.0	0.0	0.0

FIGURE 4.23: The total area post synthesis

parts of settings are default from the first subsection. Here is the result of synthesis by Catapult. Here is the result of synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	66	51659	51661	1	0	
Design Total:	66	51659	51661	1	0	

FIGURE 4.24: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	300.4	1235.9	660.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.25: The total area post synthesis

4.4.3 Synthesis by setting the outer loop pipeline and unroll

In this subsection, the first loop topksorted_distances_label0 will be set as pipeline, and the second loop topksorted_distance_label2:for will be set as unroll, and the other parts of setting are default from the second subsection,here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	506	51512	51514	1	0	
Design Total:	506	51512	51514	1	0	

FIGURE 4.26: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	5249.0	9633.3	6232.3
Total Reg:	0.0	0.0	0.0

FIGURE 4.27: The total area post synthesis

4.4.4 Synthesis by setting the outer loop pipeline and inner loop pipeline

In this subsection, the first loop topksorted_distances_label0 and topksorted_distances_label2:for will be set as pipeline, and the other parts of setting are default from the second subsection,here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	66	51362	51364	1	0	
Design Total:	66	51362	51364	1	0	

FIGURE 4.28: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	300.4	1305.8	666.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.29: The total area post synthesis

4.4.5 Synthesis by setting the data enable interface

From this subsection, the interface synthesis in Catapult will be explained. In these three sections, SCVerify flow will be used, as the SCVerify function is proving the C++ algorithm and RTL function equivalent by giving the same input vectors. In the blow sections, I will use different methods to control the flow of data, enable signal and wait interfaces. In the first synthesis method, I click on the Flow Manager tab and make sure that SCVery is enabled.For the output interface r_dist and r I choose mgc_import_out_stdreg_en. Here is the result after synthesis:

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	66	51362	51364	1	0	
Design Total:	66	51362	51364	1	0	

FIGURE 4.30: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	300.4	1305.8	666.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.31: The total area post synthesis

4.4.6 Synthesis by setting two way handshake interface

In this subsection, the interface will shift to two-way handshake, even the data is not available, but the catapult block can be installed. I set the interface r_dist:rsc and r_index:rsc as mgc_ioport.mgc_inout_buf_wait, here is the result after synthesis.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	66	51362	51364	1	0	
Design Total:	66	51362	51364	1	0	

FIGURE 4.32: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	300.4	1424.8	862.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.33: The total area post synthesis

4.4.7 Synthesis by adding initiation interval size in the interface

In this subsection, the setting is same as the second case, but the initiation interval will increase from 1 to 2, here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	66	102561	102563	1	0	
Design Total:	66	102561	102563	1	0	

FIGURE 4.34: The latency and Throughput

4.4.8 Synthesis by creating c-cores in the mapping process

In this subsection, the c-cores method will be used.C-CORE is a user-defined operation that consists of a collection of one or more operators.It can improve the area by

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	300.4	1499.1	892.0
Total Reg:	0.0	0.0	0.0

FIGURE 4.35: The total area post synthesis

minimizing the MUX sharing logic and reduce the run time by reducing the number of variables. For applying the c-core methodology,I first modify the code then in the mapping step, select the design type as CCORE, and then select combinational block. Here is the result of the synthesis by catapult.

Processes/Blocks in Design

Process	Real Operation(s) count	Latency	Throughput	Reset Length	II	Comments
/topksorted_distances/core	58	51806	51808	1	0	
Design Total:	58	51806	51808	1	0	

FIGURE 4.36: The latency and Throughput

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	246.4	1219.3	635.4
Total Reg:	0.0	0.0	0.0

FIGURE 4.37: The total area post synthesis

4.4.9 The result comparison of top sorted implemented in Catapult

In this section, I make a table for the comparison of different synthesis methods.As it can be see, the synthesis method of setting outer loop pipeline and inner loop pipeline have the better performance in terms of latency and area.

Solution	Latenc...	Latency T...	Throughp...	Throughp...	Total Area	Slack
topksorted_dis... (extract)	154106	462318.00	154108	462324.00	629.40	-2.16
topksorted_dis... (extract)	102711	308133.00	102713	308139.00	847.04	-2.32
topksorted_dis... (extract)	51659	154977.00	51661	154983.00	660.04	-1.20
topksorted_dis... (extract)	51512	154536.00	51514	154542.00	6232.32	-1.95
topksorted_dis... (extract)	51362	154086.00	51364	154092.00	666.04	-1.20
topksorted_dis... (extract)	51362	154086.00	51364	154092.00	666.04	-1.20
topksorted_... (extract)	**51362**	**154086.00**	**51364**	**154092.00**	**786.04**	**-2.32**

FIGURE 4.38: The comparision of different synthesis method in catapult

4.5 The Comparison of Experimental Results in Vivado Synthesis

As it shows from the above chapter, The Topkref distances and Topksorted distance was synthesized in Vivado HLS and Catapult respectively,and the result of synthesis have compared in above chapter. Moreover, in order to compare the synthesis efficient,a deeper synthesis implementation should be done in Vivado by using the same method in same Zedboard.Here is the comparison in Vivado.

4.5.1 Topkref Synthesis in Vivado

		LUT	LUTRAM	FF	BRAM	IO	Latency	Period	Throughput
vivado hls	A_Dataflow	345		197	2.5	200	563685	7.303	0,000243
	Inline	336		300	1	177	102808	5.454	0,001783
	I_AXI	1214		1807	3	482	563635	5.447	0,000326
	I_FULLPARTITION	115		301	1	171	102808	5.447	0,001786
	I_FIFO	320		293	1	105	102808	5.447	0,001786
	I_PARTITION	336		300	1	177	102808	5.451	0,001784
	P_in_unroll_out_pipe	312		299	0.5	120	563759	5.317	0,000334
	P_inner	326		327	1	120	102744	3.320	0,002932
	P_outer	327		299	1	120	563426	4.549	0,00039
catapult	catapult1	565		612		178	461412	3.289	0,000659
	catapult2	517		614		178	153918	3.671	0,00177
	catapult3	549		637		184	153915	3.289	0,001975
	catapult4	468		520	0.5	242	153915	3.109	0,00209
	catapult5	539		612		178	153915	3.984	0,001631

FIGURE 4.39: The comparision of Topkref Synthesis in Vivado

The function of Topkref was synthesis in nine different ways in Vivado HLS, and five ways in catapult, then for each IP generated by Vivado HLS and Catapult, it would be synthesis again in the tool of Vivado, and the result was list in the table. In this table, throughput was calculated by the format of 1/(Latency*Period)/1000000 as the unit of period was expressed by ns.From this table, we can found out that the LUTRAM has not been used; For the synthesis method of AXI,the maximum source will be used. In terms of BRAM, the catapult has the better performance as compare with Vivado hls.

In the Pareto-optimal point graph, the Pareto-optimal point would be showed as the gray color point. the Pareto optimization is used for in a design to decide what is good or desirable,as a involves multiple criteria such as capital cost,operating cost, quality and efficiency etc. In this thesis, The ideal design mainly involves the hardware

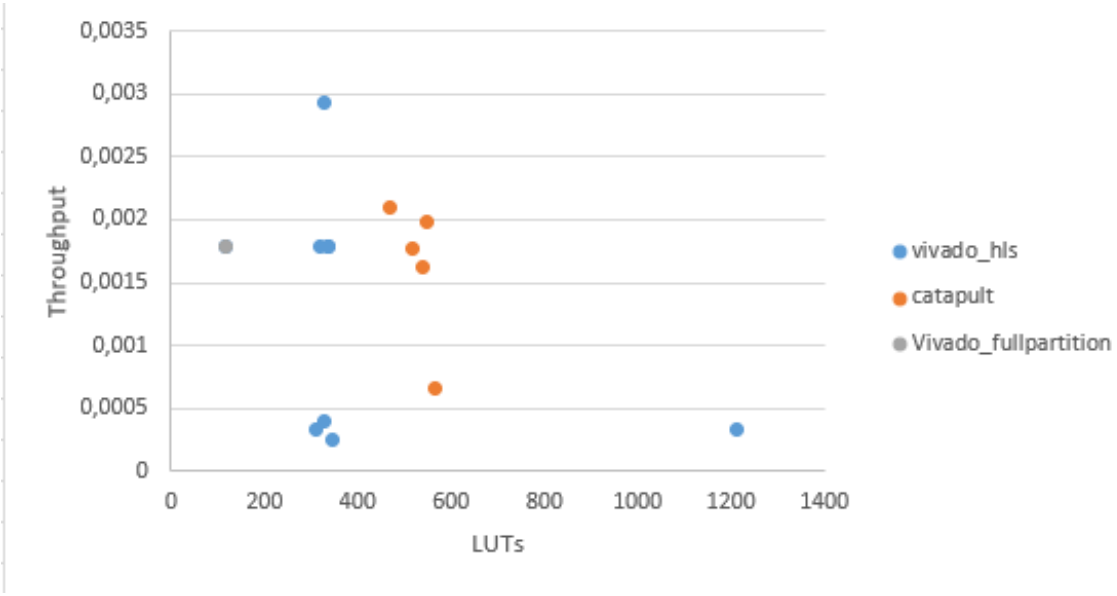

FIGURE 4.40: Pareto-optimal point in the graph

resources such as LUTs,FF(flipflop) and the other part that needed to be considered is the throughput.

4.5.1.1 Topksorted Synthesis in Vivado

		LUT	LUTRAM	FF	BRAM	IO	Latency	Period	Throughput
vivado hls	A_Dataflow	1043	111	1714	1.5	484	563685	5.889	0,00030125
	P_outer	170		280	0.5	122	563426	6.417	0,00027659
	P_inner	98		171	0.5	122	102744	6.780	0,00143554
	P_Ineer_p_unroll	4589	87	3526	2	482	563759	6.790	0,00026124
	Inline	1219	79	1725	1.5	482	102808	4.953	0,00196383
	I_PARTITION	112	79	178	0.5	162	102808	5.813	0,0016733
	I_Interface	98	79	171	0.5	122	102808	5.814	0,00167301
	I_Fullpartition	2345		230	0.5	8681	102808	5.480	0,00177498
	I_AXI	1048	79	1637	0.5	484	102808	6.813	0,00142769
catapult	catapult1	5649		3940	0.5	242	154106	6.652	0,0009755
	catapult2	447		520	0.5	242	102711	6.891	0,00141287
	catapult3	5658		3940	0.5	242	153915	6.697	0,00097015
	catapult4	4680		520	0.5	242	153915	6.891	0,00094284
	catapult5	768		520	0.5	242	153915	6.297	0,00103178

FIGURE 4.41: The comparision of Topksorted Synthesis in Vivado

Similar to the function of Topkref, the function of Topksorted will also synthesis in Vivado HLS and catapult respectively. And then in Vivado, setting the clock period and syntheis the IP that generated by the Vivado HLS and Catapult.The result of the synthesis would list in this table above.In the graph, we can see that the catapult even donnot use the resources of LUTRAM, but in terms of BRAM, it seems that there is

no big difference between Vivado HLS and Catapult. In this graph, we can see that the

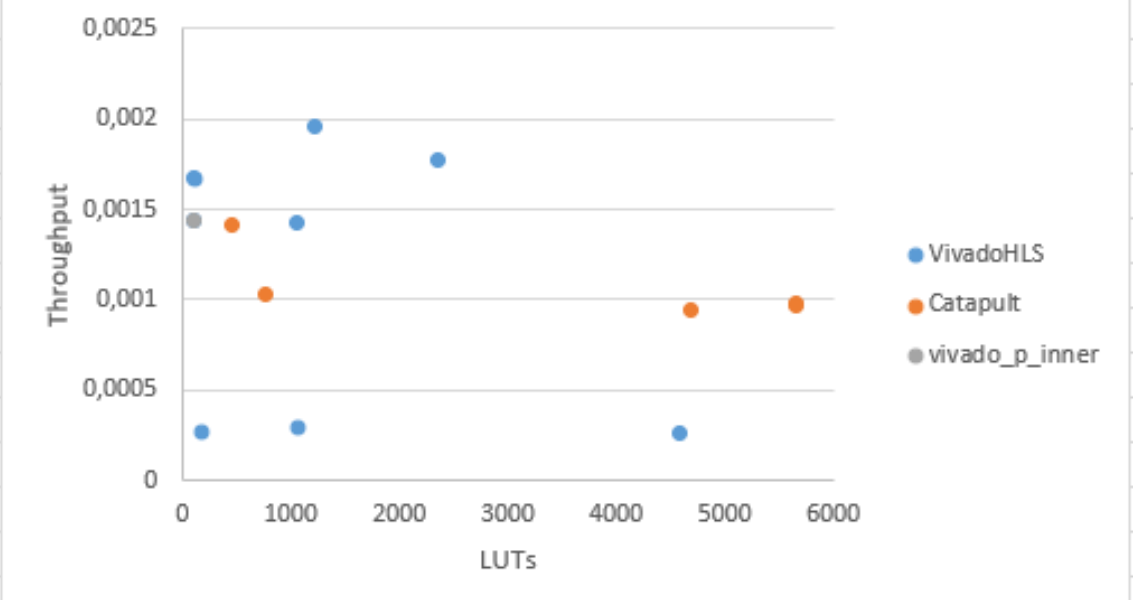

FIGURE 4.42: Pareto-optimal point in the graph

synthesis method of P_inner should be the Pareto optimal solution as compares to other solutions. In this graph, the throughput of Vivado HLS range asymmetrically, and the throughput from catapult keeps stable in terms of using different number of resources.

Chapter 5

Conclusion and Prospect

5.1 Conclusion

This thesis focus on the design of digit recognition system, and the implementation of High level synthesis to some top functions in the design. Meanwhile, topkref and topksorted function in the digit recognition system are implemented as the main technique in optimization and simulation in Vivado HLS ,and Catapult consequently in Vivado. Fortunately, some achievements have been obtained finally.

Generally speaking, the Vivado HLS is very suitable and powerful tool in the optimization of complicated architectures. In addition, Vivado is a convenient device in terms of performance computation and sensitivity analysis for the design implemented in FPGA.

5.1.1 Some Achievements

- The digit recognition system was programmed in Ubuntu are further developed for simulation and optimization in terms of timing and resource utilization in Vivado HLS and Catapult.
- The reason why choose the digit recognition system has been discussed.
- The functions of topkref and topksorted in the digit recognition system has been synthesis by using multiple methods in Vivado HLS and Catapult respectively.

- The Pareto optimal point in terms of utilization and throughput for both functions have been obtained by using above method.
- The recognition rate of the digit recognition system has been achieved to 97 percentage out of 100.

5.2 Prospect

5.2.1 Digit recognition system Developing

Automatic handwriting recognition has a variety of applications at the interface between man and machine[5]. Since the Digit recognition system is already widely used as discussed, there is still a large room to develop it both in theory and practice.

Unlike the KNN I used in the thesis, there should be more researches in digit recognition.Putra Sumari [6] implemented the handwritten Digital Recognition using Neural Network, Morocco [7] introduced the approaches of Discrete Cosine Transform (DCT) for digit recognition. in the near future, more methods to deal with the digit recognition will be proposed in order to improve the recognition rate and precision. And in terms of coding, the program and the algorithms would be developed by multiple languages instead of c/c++.

5.2.2 High Level Synthesis

The high level synthesis is still remaining on third generation,and the fourth generation will emerge from the current generation[8]. in the future work, the HLS may focus on complex algorithm and system design into FPGA forms as the FPGA provides parallel architecture as compare to traditonal processors. Any way, this thesis is not the end but the start of the research.

Appendix A

Appendix Title Here

Bibliography

[1] Andreas Gerstlauer, Christian Haubelt, Andy D Pimentel, Todor P Stefanov, Daniel D Gajski, and Jürgen Teich. Electronic system-level synthesis methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1517–1530, 2009.

[2] Jürgen Teich. Embedded system synthesis and optimization. 2000.

[3] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International Journal of Engineering Research and Applications*, 3(5):605–610, 2013.

[4] Jianping Gou, Lan Du, Yuhong Zhang, Taisong Xiong, et al. A new distance-weighted k-nearest neighbor classifier. *J. Inf. Comput. Sci*, 9(6):1429–1436, 2012.

[5] Sven Behnke, Marcus Pfister, and Raúl Rojas. Recognition of handwritten digits using structural information. In *Neural Networks, 1997., International Conference on*, volume 3, pages 1391–1396. IEEE, 1997.

[6] Saleh AK Al-Omari, Putra Sumari, Sadik A Al-Taweel, and Anas JA Husain. Digital recognition using neural network. *Journal of Computer Science*, 5(6):427, 2009.

[7] Bouchra El Qacimy, Mounir Ait Kerroum, and Ahmed Hammouch. Feature extraction based on dct for handwritten digit recognition. *International Journal of Computer Science Issues (IJCSI)*, 11(6):27, 2014.

[8] Grant Martin and Gary Smith. High-level synthesis: Past, present, and future. *IEEE Design & Test of Computers*, 26(4):18–25, 2009.