Virtual Partial Reconfiguration Framework for the
Digilent Nexys 3 Board

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Kawin Lertlaokul
Student ID: 402170
Date: 01.02.2019

Supervising tutor: Prof. Dr W. Hardt
: Dipl.-Inf M. Nagler
Acknowledgment

I gain many valuable things during conducting this research at Technische Universität Chemnitz. I would like to express my sincere gratitude to my research advisor, Michael Nagler, for this work opportunity, the valuable advice, great support and useful critiques of this research work. I would also like to extend my sincere gratitude to Dr Wolfram Hardt and Dr Ariane Heller for the patient guidance and support.

To my family, I really appreciate your support and encouragement over the years. You have always believed in me.
Abstract

The modern embedded system is getting more complicated due to the functional requirements of the system are rapidly increasing. The modern system must have more reliable, as it deals with a lot of data. The distributed systems are used in variety technologies field due to it has more reliable than single control unit. It can transfer task to other processing unit when the one part of system failed while the single control unit failed cause the system to stop operate. The FPGA are being used increasingly in the distributed system due to the benefit of FPGA over microcontroller and ASIC. FPGA is flexible than ASIC due to the ability to reconfiguration its function. FPGA processes the data in parallel, therefore, it computes the data faster than the microcontroller that computes the data in concurrence. The flexibility of FPGA supports the development of reliable distributed system. When one of FPGA failed, the other FPGA can reconfiguration itself to operate on the task of the failed FPGA. The method to reconfigure the FPGA structure is a process of loading new bitstream file into FPGA. For generating variety configurations of distributed system. The developer must develop number of bitstream file according to number of reconfiguration designs. Although the FPGA is flexible and can reconfiguration anytime, the development process of configuration file is a redundancy workload. One FPGA design structure equals one configuration file. This project focus on reduce the redundancy workload, therefore, it can reduce the development time and make the development project launching faster. This virtual partial reconfiguration framework is developed to assist the developer in generating many configuration files without coding. The framework will determine all possible combination of modules and generates all combination design files. One set of the design contain the VHDL file and UCF file. The developer can use these files to synthesise in FPGA vendor development tool and generate bitstream. This virtual partial reconfiguration framework also provides the partial reconfiguration benefits except runtime reconfiguration.

Keywords: Partial Reconfiguration, VHDL file generator, Spartan-6, Digilent Nexys 3, FPGA
Content

Acknowledgment ........................................................................................................... 1
Abstract ......................................................................................................................... 2
Content ............................................................................................................................ 3
List of Figures ................................................................................................................ 5
List of Tables .................................................................................................................. 7
List of Abbreviations ...................................................................................................... 8
1 Introduction ................................................................................................................. 9
  1.1 Distributed System ................................................................................................. 10
  1.2 Field Programmable Gate Array (FPGA) .............................................................. 12
  1.3 FPGA Development Procedure ........................................................................... 13
  1.4 Structure Overview ............................................................................................... 15
  1.5 Summary ................................................................................................................ 16
2 State of The Art ......................................................................................................... 17
  2.1 Partial Reconfiguration Technique ........................................................................ 17
  2.2 Software Tool for Partial Reconfiguration ........................................................... 18
  2.3 Partial Reconfiguration Technique for spartan-6 ................................................. 20
  2.4 Internal Configuration Access Port (ICAP) ........................................................ 23
  2.5 Related Works ...................................................................................................... 25
  2.6 Summary ................................................................................................................ 26
3 Virtual Partial Reconfiguration Framework ............................................................. 28
  3.1 Fundamental Concept of Virtual Partial Reconfiguration Framework ............... 28
  3.2 Fundamental Knowledge for the Framework Development .................................. 31
    3.2.1 VDHL language description .............................................................................. 31
    3.2.2 The Design fundamental of Virtual Partial Reconfiguration ......................... 35
  3.3 Requirements of the Framework .......................................................................... 35
    3.3.1 User-Defined Port type .................................................................................. 38
3.3.2 User-Defined Generic Type..........................................................42
3.3.3 The Number of FPGA Designs ....................................................43
3.3.4 The Module Configuration...........................................................44
3.4 Summary.........................................................................................46
4 Implementation ..................................................................................47
  4.1 Structure of the Framework ............................................................47
  4.2 The VHDL Input File Extraction Module .........................................50
    4.2.1 Regular Expression ..................................................................51
    4.2.2 VHDL File Extraction Development ..........................................53
  4.3 The Data Storing Module ...............................................................57
    4.3.1 Object-Oriented Programming (OOP) .......................................57
    4.3.2 Implement Object-Oriented Programming to the Storage Module...60
  4.4 The Design Computation Module ....................................................66
    4.4.1 Depth-first Search (DFS) ..........................................................69
    4.4.2 Implement the Module Combinations of the Designs ..................70
    4.4.3 The Output File Generating ......................................................72
  4.5 The Graphical User Interface Module ..............................................78
  4.6 Summary.........................................................................................81
5 Result and Evaluation .........................................................................83
  5.1 Evaluation .....................................................................................83
  5.2 Result ............................................................................................84
6 Conclusion ..........................................................................................86
Bibliography ...........................................................................................87
Appendix A: Contain of CD .................................................................89
List of Figures

Figure 1.1: Distribution of functional requirements over FPGA cluster [4] ............11
Figure 1.2: Overview of FPGA Architecture [5]. ........................................................13
Figure 1.3: FPGA development flow [7].................................................................14
Figure 2.1: A comparison between full configuration and partial reconfiguration [9].................................................................17
Figure 2.2: An overview of Partial Reconfiguration software flow [10]..............19
Figure 2.3: The additional procedures for developing Partial Reconfiguration [18]. ..................................................................................................................20
Figure 2.4: An assisting tool for providing partial reconfiguration [20]............21
Figure 2.5: The structure of ICAP component [13]. ..............................................24
Figure 2.6: ICAP-I design structure.........................................................................25
Figure 2.7: Fast Start-up FPGA algorithm [18].....................................................26
Figure 3.1: The system that has two reconfiguration areas for three modules....28
Figure 3.2: The three designs and the modules contained inside the designs.....29
Figure 3.3: The development views of the repetition processes for the system...30
Figure 3.4: The structure of the top-component entity...........................................32
Figure 3.5: The structure of Architecture of top-component..................................33
Figure 3.6: The port and generic defining statement.............................................34
Figure 3.7: The “begin” part of the architecture body...........................................35
Figure 3.8: The design files generated from the framework.................................36
Figure 3.9: The interconnection diagram of Global port.....................................38
Figure 3.10: The interconnection diagram of the Reset port...............................39
Figure 3.11: The interconnection diagram of Global port.....................................39
Figure 3.12: The interconnection diagram of an internal port.............................40
Figure 3.13: The interconnection diagram of Unused port...................................40
Figure 3.14: The interconnection diagram of the Split port.................................41
Figure 3.15: The interconnection diagram of Global Split port.........................41
Figure 3.16: A one-to-many connection of a static module to dynamic modules..44
Figure 3.17: The example shows all designs of 4 dynamic modules....................45
Figure 4.1: The simple diagram of the model-view-controller...........................50
Figure 4.2: The code extraction area of sub-modules..........................................51
Figure 4.3: The regular expression keyword of the regular port.........................54
Figure 4.4: The regular expression keyword of port MSB and port LSB............55
Figure 4.5: The regular expression keyword of regular generic...........................55
Figure 4.6: A comparison of generic value and generic value array .................. 56
Figure 4.7: The regular expression keyword of vector generic ...................... 56
Figure 4.8: The Object-Oriented Paradigm .............................................. 58
Figure 4.9: The class diagram of the storage module .................................. 63
Figure 4.10: The rule of split port development ......................................... 68
Figure 4.11: The designs computation algorithm diagram ............................ 70
Figure 4.12: The algorithm for storing the component and generating a new design. ................................................................. 72
Figure 4.13: The example of component instantiation statement .................. 73
Figure 4.14: The clock port mapping statement ........................................ 74
Figure 4.15: An example of a port mapping statement ................................. 77
Figure 4.16: The user interface of the virtual partial reconfiguration framework. .................................................................................... 80
Figure 5.1: The design consists of ‘dynamic_a’ and ‘dynamic_b’ .................... 84
List of Tables

Table 1.1: A data usage per second of sensors in an autonomous vehicle [1]. ........9

Table 3.1: The example values of each generic type.........................................................43

Table 4.1: The string matching result of the keyword by using character ‘^’. ........52

Table 4.2: The extracted results of the keyword ‘\w+’..........................................................53

Table 4.3: The extracted results of the keyword ‘(\w+)\s+\w+(\w+)’..............................53

Table 4.4: The method for calculating the port width.........................................................76
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT</td>
<td>Internet of thing</td>
</tr>
<tr>
<td>RCT</td>
<td>Real-Time Computing</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable logic block</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-up Table</td>
</tr>
<tr>
<td>IOB</td>
<td>Input/Output block</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very High-Speed Integrated Circuit Hardware Description Language</td>
</tr>
<tr>
<td>ICAP</td>
<td>Internal Configuration Access Port</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>UCF</td>
<td>User Constraint File</td>
</tr>
<tr>
<td>XDL</td>
<td>Xilinx Design Language</td>
</tr>
<tr>
<td>BIT</td>
<td>Bitstream file</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller Pattern</td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming</td>
</tr>
<tr>
<td>REGEX</td>
<td>Regular Expression</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>LSB</td>
<td>Less Significant Bit</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth-first search</td>
</tr>
<tr>
<td>SLOC</td>
<td>Source Lines of Code</td>
</tr>
</tbody>
</table>
1 Introduction

Nowadays, technologies are more complicated every day. Since the world is moving faster into industry 4.0 era, which is the 4th industrial revolution. The German government establishes this commitment in Hannover Fair 2011. This strategy focuses on implementing automation and data-exchange into the manufacturing process. It also includes internet of thing (IoT), cloud computing and an autonomous vehicle. As it is mentioned before, the automation and data-exchange make the system more complex. Functional requirement of the system is continuously increasing. A big chunk of data will flow between each system and also inside the system.

For example, the area of autonomous vehicles, an autonomous vehicle must aware of the surrounding environment, by interacting and examining environment data. Autonomous system of the vehicle consists of many sensors for recognising data and many sub-systems for evaluating data. The most common sensors, are used in a vehicle, are LIDAR sensor, a radar sensor and an image sensor. The sensors use large bandwidth as the table below:

<table>
<thead>
<tr>
<th>On-Road Sensor</th>
<th>Bandwidth Usage (Approximate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lidar</td>
<td>10 - 70 MBps</td>
</tr>
<tr>
<td>Radar</td>
<td>10 - 100 KBps</td>
</tr>
<tr>
<td>Camera (image sensor)</td>
<td>20 – 40 MBps</td>
</tr>
<tr>
<td>Sonar</td>
<td>10 - 100 KBps</td>
</tr>
</tbody>
</table>

Table 1.1: A data usage per second of sensors in an autonomous vehicle [1].

In 2020, it is forecasted that one vehicle will have around 200 sensors [1]. It will generate 4000 GB of data per day [1]. Systems in the vehicle are also considered as hard real-time computing (RTC), which is not entitled to missing deadline. It will cause a critical issue such as an accident. Therefore, the system must be fast enough to process extensive data before the critical deadline. An embedded system development can classify into two development types.

1. **Single control unit by using a high-performance chip.** This type of chip can conduct all the data alone. It has a lot of logical resource and memory for developer using [2]. This kind of chip is quite expensive that...
makes the chip does not suit in common development due to cost restriction [2]. If the chip has some failure occur, the system will be unable to operate. It can use redundancy logic to overcome this problem [2]. This method is irrational because it is not practical to have equally logic that doesn't use if the system normally operates. It is a waste of memory.

2. Develop a system in distributed system structure. It is a system that distributes the workload to sub-system for a specific purpose. It also has the advantage and disadvantage of it.

A distributed system is contributed to its topic because this report focuses on the development of a distributed system.

1.1 Distributed System

So far, there is no agreement for the meaning and the characteristic of a distributed system [3]. From various literature, it can assume that a distributed system is a system that consists of sub-systems processing tasks in parallelisation or sequential to fulfil system task. Each sub-system is responsible for a part of a system task.

From the concept of a distributed system, it gets the benefit performance benefit by performing the task in parallel. The distributed system also gives a system more reliable. Although one sub-system fails to operate, another parallel system can process it instead by trading-off some performance. Due to the price gap and exponential growth rate of chip cost, a distributed system of small chips can achieve cost restriction. From the next paragraph, it will focus on clarifying the research project below:

“Verteiltes Applikations- und Ressourcenmanagement für rekonfigurierbare Rechencluster” from Michael Nagler, Technische Universität Chemnitz [4]

The project above also uses a distributed system technique to develop a more reliable system further. In this topic, it develops clustering network by using Field Programmable Gate Array (FPGA). This research focuses on distributing functional requirement of application over a cluster of hardware infrastructure. This idea provides an ability to move functional requirement to the other hardware (FPGAs) as it shows in the figure below:
From the figure above, it is divided into two parts, which are software part (Application) and hardware part (FPGA Cluster). In the software part, M stands for a module. Each module is used for functional requirements computation.

For instance, this system is object detection camera for classifying products. M0 is a control module, M1 is a video camera (system input), M3 is video processing and classifying, M4 is storing videos and keeping video logs, and M2 is an output interface that exhibits product detail and moving it to store location. After developer determines all modules, the developer needs to specify each module into FPGAs. To place modules into FPGAs, it needs to consider resource constraint of each module [4]. All modules are placed as it shows in figure 1.1.

At first, FPGA 1 has M0, FPGA 2 has M1, FPGA 3 has M2, and FPGA 4 has M3 and M4. It is placed by considering the limitation of FPGAs. This setup works according to the requirement. Presuming that system has been working for a while, the FPGA 4 suddenly stops operating. If the system fails to operate, the profit loss increases according to the time spent to solve the problem. The advantage of this system can remap module to another FPGA without loading new bitstream. From figure 1, it shows that M3 is remapped to the same FPGA with M2. M4 is in M1 FPGA. This structure can make the system back to work immediately. It is questionable, why the developer does not map module to only 3 FPGAs since launched if it can fit into 3 FPGAs. As mentioned earlier, the decision to map FPGAs with the modules. It needs to consider resource of FPGA and resource that module consumed. In order to remap the module to another FPGA, it needs to
trade-off the performance of the system such as M0 gets the video in 1080p resolution. This resolution needs a heap of resources to store and process video. To remap M3, M4 on the other FPGAs, it needs to change the properties of the module to consume fewer resources by reducing resolution or colour-depth of the video. This method makes M3, M4 able to fit on the others.

This thesis intends to assist developer, developing a distributed system by reducing redundancy workload in generating all possible design. In order to develop a distributed system, it is beneficial to know FPGA development and FPGA primary characteristic. This project interested in using Digilent Nexys 3 board that is an inexpensive model and it has been used widely. This board is a part of the Spartan-6 family.

1.2 Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) was first developed two and a half decades ago [5]. It is used a lot in a research study, low and medium production. It provides more flexibility than Application Specific Integrated Circuits (ASICs). ASIC is a circuit board that specially designed for a specific task. It is a final version of the board, that cannot be changed the circuit routing inside. It contrasts to FPGA, which able to re-route the circuit part inside. ASIC is unable to use in this project because it does not allow to remap modules.

FPGA has three main components that make re-route component inside possible. It consists of [5]:

1. **Configurable logic blocks (CLBs):** It uses for computing logic function. Each logic block contains 4-input Look-up Table logic (LUT-4). This LUT-4 is used for specifying the outcome of 4-input.

2. **Programmable routing:** It is used to bridge between Programmable logic blocks to compute more complicated task. It can also bridge between Programmable Logic Block and Input/Output blocks to exchange data with another device

3. **Input/Output blocks (IOBs):** It is the outer block of FPGA. It is used for interconnection to another device.
From figure 1.2, it shows that all CLBs and IOBs are connected by programmable routing in network shape. Each CLB can processes data individually. It makes FPGA (parallel computing) is faster than Microcontroller (sequential computing). FPGA uses less power consumption than Microcontroller.

1.3 FPGA Development Procedure

In this chapter, it will focus on developer point of view. The procedure and language used for configuration FPGAs. The language uses, in FPGA, is called Hardware Description Language (HDL). It is a language used for configuration of hardware (FPGA) by describing the structure and behaviour of it. From the project, that mentioned earlier, this report will focus on Very High-Speed Integrated Circuit Hardware Description Language (VHDL). It is one of the main HDL languages.

FPGA development is the method to translate VHDL code to bitstream for loading into FPGA board. This method separates five main steps as figure below [7]:

Figure 1.2: Overview of FPGA Architecture [5].
Purple lines are programmable routing.
The process starts with developer writing and validating VHDL code. VHDL code validation should resolve functional requirement. This file is the beginning point for FPGA bitstream loading. It flows through the sequence:

1. **Synthesis**: This stage checks code syntax and examines the design hierarchy of code [7]. It checks design hierarchy to confirm that it meets selected FPGA architecture specification. The stage forges netlist file.

2. **Translate**: This stage integrates netlist files with design constraint files [7]. The constraint file contains information and limitation of the specific FPGA such as connections between port (Hardware) and pins (Software). It gathers all information and transforms information into the logical design file. The design file stores information in logic gate format.

3. **Map**: This stage maps the logic design file to the FPGA components [6]. It translates logics of the algorithm, which is a logic gate (and-gate, or-gate, not-gate), to component (LUTs in CLBs, IOBs) of FPGA.

4. **Place and Route**: This stage places designed CLBs in design file to actual CLBs on the FPGA and routes connections of CLBs and IOBs corresponding to logical design [6]. The output file of this stage contains data which CLBs, IOBs are used and routing paths between them.

5. **Bitstream generation**: This stage generates a bitstream file from the design file from the process before [7]. The bitstream file is a binary sequence file that will be transferred to FPGA, and it uses for FPGA configuration.

VDHL description language has 3 description types to realize FPGA logic [8].

1. **Behavioural description**: Developer describes logic as programming language such as c, c++, python. The code is written by using behavioural
statements such as if-else function, for-loop function. VHDL runtime realises this part in sequential.

2. **Dataflow description**: FPGA is realised in logic gate structure by using and-gate, or-gate, not-gate. VHDL executes it in parallel.

3. **Structural description**: FPGA is realised by wiring components like wiring chip in integrated circuit. The is derived by using behavioural or dataflow description.

Although FPGA is flexible, the developers need to rewrite code whenever changing the FPGA operation. If the code is realised as a single component, it is hard to modify. Codes need to be rewritten. If FPGA is realised in the structural description, it is more natural. It provides a concept of chip welding in an integrated circuit. The developers need to change the components and rewires all components and all ports.

To develop the reliable distributed system, that capable of transferring module freely when FPGA in the cluster breakdown. It is hard tasks and increases workload to the developer in exponential growth. Development one FPGA that has different designs is rewritten code one-time per one-design. The more modules in the system are the more designs to develop. For example, the system that has two modules (M0, M1). The FPGA consist of 3 design bitstreams. The designs are first design (M0, M1), second design (M0) and third design (M1). It causes redundancy work to developers. Developers need to reconstruct all hierarchy FPGA designs.

### 1.4 Structure Overview

This thesis comprises of six chapters. The first chapter is an introduction part that it shows the field of hardware that receives the benefit from partial reconfiguration development. The second chapter explains the partial reconfiguration technique. It also states of the hardware and software requirement for using this technique. The chapter describes the research that gives the idea to implement partial reconfiguration technique on the Spartan-6 board, which is the same series of the Digilent Nexys 3 Board. The third chapter begins with the explanation of why another approach is not useful for our project, and then it will be basic knowledge that are necessary to implement virtual partial reconfiguration framework. On the last part of this chapter, it states the functional requirement that framework needs to achieve. The fourth chapter shows the structure of the framework and the development of framework modules. The techniques used for developing the
module. The fifth chapter shows the evaluation and the result of the framework. The last chapter is the conclusion and future work.

1.5 Summary

This chapter clarifies the cause of this topic selected and explains fundamental element related to clarify this problem. It states the benefit of the distributed system. This system has more reliable than a single control unit system. When one part failed to operate, the system can transfer tasks to another part and continue to operate.

In contrast to single unit control when one part failed, it causes system failure. By using the backup module, it can solve this issue for the single unit control system. The backup module that is identical to the primary module will operate only when the primary module failed. This design causes a waste of system resources.

Although the distributed system is highly reliable, the system development is complicated and consumes much time in development. The developer must prepare all necessary design for changing. This thesis intends to develop a framework to assist the developer in generating design. This framework will reduce time in development.

On the last two parts, it briefly explains FPGA characteristic and components and the procedure for developing FPGA. It states the disadvantage of the development procedure, why it consumes a lot of development time.

Lastly, this project focuses on assisting the developer in developing a distributed system base on the Spartan-6 FPGA device family.
2 State of The Art

To assist the developer, reduce development time. A tool should generate different design from the developed modules, that developer developed. The tool should support the developer by mapping ports and generics of different designs, that contains different module inside. There is a technique call Partial Reconfiguration

2.1 Partial Reconfiguration Technique

Partial reconfiguration technique is developed to make more flexibility in FPGA design. FPGA is a reconfiguration device which is more flexible than ASIC. It can be reprogrammable to change functionality. Typical FPGAs must fully load the bitstream to reconfigure themselves. FPGAs requires completely format when reprograming themselves. The partial reconfiguration technique offers partial reconfiguration to FPGAs by keeping one part of FPGA as a static part, and another part is a dynamic module. Static part is not reloaded when reprogrammed. Dynamic part is only reprogrammed part.

![Image of Full Configuration and Partial Reconfiguration](image)

Figure 2.1: A comparison between full configuration and partial reconfiguration [9].

From figure 2.1, when the design needs to change, full configuration is loaded full bitstream, and it reroutes and maps the whole FPGA then system need to reboot for the new configuration. In contrast to partial reconfiguration, it is loaded partial bitstream and reconfigures that area. Rebooting the whole system is not necessary. The idea of technique extends the design possibility of FPGA. It also provides many advantageous over full configuration.
1. **Reducing the size of the FPGA [9]**: Typical FPGA system comprises many modules that resolve functional requirement. Some module works in sequential. The module needs to wait for input from module before. It is not necessary to load all modules into FPGA when it is not in use. Partial reconfiguration provides an option to choose a module according to the sequence of operation. A smaller FPGA can be chosen. Choosing decision changes from finding FPGA, that can place all modules to smaller FPGA for one module. This benefit gives cost and power consumption advantage. The smaller FPGA consumes less power and lower price.

2. **Provision of flexibility to choose logic and protocol of application [9]**: Assuming that it is a universal device such as USB device. This device will connect to the varieties USB device. This device should adapt to the connected device. The technique provides the capability to change protocol and logic follow connected device without resetting system.

3. **Accelerating configurable computing time [9]**: Partial bitstream file contains information only reconfiguration area. The file size is smaller than full bitstream, therefore loading this file to FPGA will be faster.

4. **Enabling new design techniques [9]**: As mentioned earlier, it provides new capabilities that conventional FPGA cannot. The capability of changing configuration without reboot system, it enables opportunities to develop the always-on system.

The techniques provide many potentials. There also has requirements for using this technique. First, FPGA board has components meet requirement hardware. Second, the tool uses for generating a partial bitstream.

### 2.2 Software Tool for Partial Reconfiguration

The tool flow, is mentioned in this chapter, is tool form FPGA vendor (Xilinx). The partial reconfiguration process is inadequate different to non-partial reconfiguration process. The whole process detail is informed in chapter 1, and this chapter enlarges the distinction of two processes. The partial reconfiguration process can be acknowledged as implementing multiple non-partial reconfiguration processes as figure 2.2 [10].
Firstly, the process initiates with three types of VHDL files. The first type of file is a static module which permanently establishes on FPGA. The second type of file is a reconfiguration module, and these modules are swapped corresponding to functional need at that time. The last type is a top-static file. This type must be written in the structural description. It is top-level of hierarchy structure which is the container for connections of static modules and dynamic modules.

The second phase is a synthesising process. This process synthesises the modules separately. It generates an individual netlist for each module. In the transform stage, the constraints file is isolated to constrain file of the static modules and each constraint files for the dynamic module. The static constraint file has exclusive principles that must define to use the technique. The additional constraints are area group constraint defines the border of a static part and a dynamic part, and pins partition is connection points between static modules and dynamic modules. The processes after this point are identical to the non-partial reconfiguration procedure.

From “Partial Reconfiguration User Guide” which is the Xilinx manual (UG702 v13.1) [10], it states that all Spartan device family are not supported by partial reconfiguration software. The PlanAhead tool that is a part Xilinx FPGA development tool does not permit the Spartan family to do area group partition [10]. The spartan-6 device cannot establish the static area and dynamic areas. Therefore, there are a few research papers implement partial reconfiguration to Spartan-6 model.
2.3 Partial Reconfiguration Technique for spartan-6

From limited support from vendor tool for Spartan-6 model, the partial reconfiguration technique is essential. Many techniques can further research such as increasing performance, increasing security. Therefore, many researchers invent the idea to implement this technique on the Spartan-6 FPGA family. All implement methods are divergent; however, they focus on the similar target which solves the missing stage of partial reconfiguration development.

For example, the procedure in Fast Start-up for Spartan-6 FPGAs using Dynamic Reconfiguration paper does not use a special tool to develop [18]. It adds additional procedures to the development process. It starts from establishing full bitstream of the initial design, then using binary level modification remove excessive configuration data to get initial partial bitstream as figure 2.3.

![Diagram of Partial Reconfiguration](image)

Figure 2.3: The additional procedures for developing Partial Reconfiguration [18].

Then, it uses Difference-based partial reconfiguration to compute second design by subtracting initial design from the full bitstream. In order to get subtracting bitstream, it needs to use “bitgen -r” command when generating. This command produces a second design (dark-grey colour box). This developer requires an in-depth knowledge of configuration memory structure and bitstream composition to use this technique.

The second research setups new tool called GoAhead [20]. It uses as replacement of the PlanAhead tool (vendor tool) in the Translate process as figure 2.4. It needs to use GoAhead tool combine with Xilinx ISE to develop Spartan-6 partial reconfiguration. It separates into two distinctive development parts. Firstly, the process for static part development. The procedure of the static part is:
Figure 2.4: An assisting tool for providing partial reconfiguration [20].

1. The process starts at GoAhead tool [20]. This point uses the tool to define partition areas. After defining, this tool generates the slice information in User Constraint File format (.UCF) [19]. This slice information will keep the area for reconfiguration module and prohibit from static part to use it. It also builds the structure of the bus macro template for a developer to modify it. The developer can modify the size of the data bus. A Bus Macro is generated in VHDL format by GoAhead tool. It also generated a file which contains resource usage information of static part, which resources are allowed for the static part. The information is stored in the Xilinx Design Language format (.XDL).

2. After all necessary data is prepared, the process will continue in the Xilinx ISE tool. The developer must write a static part by using VHDL code. The static part needs to implement the bus macro information that is generated in the first stage as one module in the static part. The synthesis process will produce a UCF file. After that combining information of UCF file from GoAhead with UCF file in Xilinx ISE. When VHDL file and UCF file are prepared, the process continues to on the translating process and map process in sequence.

3. It is using “Merge Blocker” command to merge Native Circuit Description file (.NCD) that generated from map process with XDL file from the first step in GoAhead tool. The NCD file is modified to prepare for the routing process. The modified NCD contains information similar to a dummy Netlist. The dummy netlist is used for dynamic resource prohibited. It
conserves resource for the dynamic module and prevents the resources from static module routing by temporally routes connection of static part to dummy dynamic part. This NCD file prevents routing of the dynamic part and allows only static part routing.

4. The routing process is a concatenation process that routes the modules follow the NCD file in block merging process (third step). After an entire FPGA is routed, it needs to remove dummy netlist that prevents resource for dynamic part. It has only static module remain on FPGA after removed.

5. It uses bitstream generation to generate a bitstream file (.BIT). This bitstream is static bitstream that ready for loading into FPGA.

For the dynamic part development, it has the procedure as below.

1. It is using GoAhead tool to define the area of dynamic part and bus macro. It needs to specify the area by specifying the slice according to the area of the static part. The tool will generate an XDL file that contains information about usable and forbidden resources. It contrasts to static part development instead of dynamic part prohibition. It prevents static part instead.

2. This process is similar to the second process in static part development. The developer needs to implement bus macro information (VHDL) as one component in the dynamic module (VHDL file). After the synthesis process generates UCF file, develop must merge information of UCF file that generated from GoAhead tool with UCF file from the synthesis process. Using all information to process further (translate process and map process). The outcome of this step is the NCD file.

3. This process is also similar to the process in static part development. It uses NCD file from the second step and XDL file from the first step to generate dummy netlist for the static part that resources are routed. It prohibits the static part area from over-routing by place and routing process in Xilinx ISE. The static part is routed, and the dynamic part is un-routed in this process.

4. It generates empty NCD file, which has no information inside, then using Bitgen to generate an empty bitstream for further using.

5. This step routes dynamic part according to information in NCD file in the third step. It is the same process to static process, except it changes from static routing to dynamic routing. The dynamic part is separated from the static part by removed routed static part from NCD file.
6. This process uses “CutoffFromDesign” commands in GoAhead tool. This command is used to remove slice resource and information outside the dynamic area. Although the NCD file passed from the removed process in the fifth step, it still has some resources that shared between the dynamic part and the static part such as BUFG Clock, Bus macro. The shared resourced are separated from the static part in this process. The process gives NCD file that has an isolated dynamic part.

7. It generates a bitstream by using Bitgen with -r argument [21]. This method uses Differential Bitgen to produce partial bitstream for dynamic part. The principle of partial bitstream method is comparison two files, selecting the different area between both file and generating new bitstream file from that area. The two input files are empty NCD file and the dynamic part NCD file form process earlier. The result is partial bitstream of the dynamic part that is produced by Bitgen -r command.

From the partial reconfiguration development above, it causes much development time for developing one static part and one dynamic part. The developer must repeat dynamic part one time per one module; therefore; this method does not suit for an enormous project which has many modules.

2.4 Internal Configuration Access Port (ICAP)

The method of FPGA reconfiguration is to load bitstream into FPGA. The bitstream will reconfigure structure of FPGA. There are three essential types of configuration ports that use for load bitstream [11]. The first port is SelectMap, which can be set master mode or slave mode. FPGA can self-load bitstream from an external source in master mode [12]. The slave mode uses an external device such processor, microcontroller and unit tester to control bitstream loading [12]. JTAG is the second type of port, that is only slave mode available. SelectMap and JTAG configuration ports support solely external memory bitstream loading.

Internal Configuration Access Port (ICAP) is the only port that provides FPGA structures and functionalities configuration at run-time. FPGA can directly access ICAP during operating.
ICAP is the component in FPGA as shown in figure 2.4. This port fundamentally uses for FPGA Multiboot operation controlling. ICAP interface is necessary developed to use ICAP. The developer needs to develop its ICAP interface. There is no ICAP prefabricated package. It has many types of research that develop an interface to access ICAP such as Sedcole [14], Claus [15], Cuoccio [16], Bok [17] and ICAP-I [11].

This part of the paper will state one example of ICAP implementation, which is ICAP-I [11]. It is an optimal ICAP design when compared to the others. The ICAP-I interface comprises two primary modules. A Storage Device Module is a device which is developed to control the storage devices. This module needs to control all storage operations. Storage commands are storage reading, storage writing and specific storage device operations such as flash memory erasing. In the storage device module, it has Storage Device Scheduler unit as figure 2.5. This unit control data transfer in and out of the storage device by using a Bulk of Transfer Unit (BTU). It organises data transferring to between the ICAP and the storage device and between the storage device and external devices.
The second module is the ICAP Controller Module (ICAP IF Module [11]. This module waits for the demands from the Storage Device Module [11]. It will manage ICAP operation reflect the request from the Storage Device Module. It usually develops ICAP interface in a finite state machine to control state (waiting state, receive state, operation state). Inside the ICAP controller module, it has ICAP Control Arbiter unit to manage the access demands from Storage Device module to ICAP. The ICAP Controller Module and the Storage Device Module can be written in VHDL package for reusable.

2.5 Related Works

As mentioned in the benefit of partial reconfiguration that it extends the opportunity in FPGA development. This part will give an example that native FPGA cannot implement this technique without supported by partial reconfiguration technique.

One of research paper executes the partial reconfiguration technique for reducing boosting time of FPGA board [18]. The idea of this concept divides configuration file loading into two groups of modules. The first module group are called high priority bitstream. The modules are classified into this group are timing-critical modules. The second group are low priority module which is a non-
timing critical module. The configuration files (bitstreams) load are shown in the figure below.

![Diagram](image)

**Figure 2.7: Fast Start-up FPGA algorithm [18].**

From figure 2.7, the initial design part is loaded first. The initial design part is high priority modules, that should operate as fast as possible. The process of partial reconfiguration bitstream loading is faster than full bitstream loading due to the small size of bitstream file and less area to the configuration. This method is called a two-steps configuration technique. The second design part will load during runtime [17]. The modules are in this group are non-critical modules that don’t have a significant impact on the system when not operating.

It is only one example that partial reconfiguration increases possibility in development.

### 2.6 Summary

In this chapter, it explains about Partial Reconfiguration Technique. This technique assists in developing different designs faster. This technique only supports some FPGAs family. Most of them are high-end FPGA. Therefore, there are a few pieces of research that focus on implement the technique to spartan-6 FPGA family. All of them focus on developing the technique to swap dynamic modules of FPGA.

Although it offers the capability to change dynamic modules, it does not assist the developer to develop partial reconfiguration easier and faster. The developer needs to repeat the dynamic module development process when it has the new dynamic module. Therefore this technique does not suit our project requirement. Another chapter in this topic explains about ICAP interface, which
supports the FPGA to reconfiguration from internal. The last part gives an example that the technique earns the benefit by using partial reconfiguration.
3 Virtual Partial Reconfiguration Framework

Virtual Partial Reconfiguration Framework is our tool, that it is developed to assist developer generates different FPGA configuration faster. It can reduce development time to do repetition works. It is titled as Virtual Partial Reconfiguration since it does not offer the capability of partial reconfiguration. It needs full reconfiguration to change FPGA characteristics. It seems curious now, why it must be developed since another research paper demonstrates the procedure to implement partial reconfiguration on Spartan-6 family in chapter two. The next chapter will explain what virtual partial reconfiguration offer and another partial reconfiguration cannot.

3.1 Fundamental Concept of Virtual Partial Reconfiguration Framework

The concept of the Virtual Partial Reconfiguration framework focuses on developer-centric. To explain it more precise, it needs to explain how developer must do when developing actual partial reconfiguration.

The ordinary development procedure is arranged into two parts. First part is the process for static part developing. The other part is dynamic module development. This dynamic module part causes the exponential increment of developer workload. The developer must repeat a dynamic module process for each dynamic module. In figure 3.1, it shows the example of a system that has three developed module and two dynamic areas (parts).

Figure 3.1: The system that has two reconfiguration areas for three modules.
The system has a capability of module swapping to change operation according to the situation. The first dynamic area and the second dynamic area has similar resources. The ordering of module is neglect due to both areas are coincident. The module 1 can be in either dynamic area one or area two. The module 2 and module 3 get the same condition. All possible designs for this system are three designs as figure 3.2.

![Figure 3.2: The three designs and the modules contained inside the designs.](image)

To develop all designs, the developer must do static development process and repeat dynamic development process four times. It repeats one time to develop module 1. The developer must repeat the dynamic development process two-times for module 2. Module 2 has two locations on FPGA resource as shown in figure 3.2. The first one is module 2 on dynamic area 1, and another is module 2 on dynamic area 2, they need to generate separately. Module 3 always places in dynamic area 2, and It needs one-time development. The procedure is shown in figure 3.3.

In dynamic module development, it must define resources area for the dynamic module by setting up slices of the area. It needs to different bus macro for different dynamic areas. For these three designs, the developer needs to repeat four times for dynamic module development process. The total process of this system is five-time repetitions of dynamic module development. This method is hard to extend such as adding module 4. Module 4 must comprise of module 4: dynamic area 1 and module 4: dynamic area 2. The additional modules that are added in this system need to develop for two dynamic areas, which can be considered as redundancy work.
The virtual partial reconfiguration framework is developed to assist the developer in generating all designs according to developed dynamic modules. The framework is virtual; therefore, it does not provide partial reconfiguration. It trades off a benefit of partial configuration which is run-time reconfiguration with development time reduction. The framework still gets other benefits of partial reconfiguration such as the FPGA board size reduction. Although it does not generate a partial configuration file, it provides full configuration file of the design which has the same dynamic modules to the partial configuration file. The disadvantage is a configuration file that is the full configuration file. To change FPGA configuration, it needs to turn off and reload configuration file. Therefore, this tool does not suit the system that needs to stay all-time online.

The summation of the framework concept, it is developed to generate all available design from dynamic modules and resource constraint. It generates full configuration bitstream of all designs. This framework serves for all FPGA development that not need to operate continuously. It assists the developer to generate a reconfiguration system in less time. The framework will generate top component VDHL file that contains module connections inside and user constraint file (UCF) that contains information of pins (FPGA hardware) and ports (FPGA internal circuit) connections. The framework detail is explained later.
3.2 Fundamental Knowledge for the Framework Development

This part explains the essential knowledge of VHDL language. It explains structure of the code to generate the top component. It also describe the partial reconfiguration design principle, as it explains the connection rule of the static module and dynamic module that the developer needs to follow when implementing partial reconfiguration.

3.2.1 VDHL language description

The framework produces top-component VHDL file that connects ports between the components. It also contains the properties of the module in values of generics. The generics values are properties that developer can modify, and it will change properties of the modules that contain it such as video resolution. The video resolution changing will change all resolution of the module inside. In order to generate the framework that can connect ports by itself. The framework that can control the properties of the modules, it needs to know the structure of the VHDL language. The top-component VHDL file is generated in the structural description format.

The structural description is the method to describe an FPGA circuit by describing the connection of sub-components and specifying sub-components are used in this component. The structural description composes of two body parts.

1. Entity body of top-component: This part of the code is used for defining the ports and generics of top-component. The ports that are defined in the top entity are interconnected ports between top-component and sub-components. The top component ports are the port that interconnected to an external device by connected through pins. The primary ports in top-component are a global clock and reset signal. The top-components generics are used for defining the properties of the sub-component by overwriting the properties of sub-component. The fundamental structure is shown in figure 3.4. All ports and generics of top component are defined inside there brackets separately. The structure of ports and generics defined are explained later.
2. **Architecture body of top-component**: This part of the code consists of two parts. The first part from the architecture declaration to the last line before the word “begin”. This part contains the components that it will be used in this top-component (system). The components that are planned to use are defined here. This defining method is called a component instantiation statement. An inside of the component instantiation statement is ports and generics of a component defining statements. It will explain the defining statement of ports and generics later. The signals defining statement are also placed in this part. The signals are an internal medium that is used inside the architecture of the component. It is generated for interconnected between the sub-components. It contrasts to ports that are connected out of the component to another component. The signal defining statement is explained later. The second part of architecture starts from the line after the word “begin”. The second part determines the interconnection of the components. It interconnects between two components by ports mapping between them. It also has generics mapping in the second part. The generics mapping is a statement for interconnected generics from top-component to sub-component. This statement provides a capability to control the properties of sub-components by controlling top-component generics. Figure 3.5 shows the structure of the architecture body that contains FPGA components and wiring details.
The component instantiation statements that are in the body of architecture are used for establishing pre-selectable components. The statement structure begins with the component name after the “component” word. The “is” syntax clarifies the ports and generics composition of the component as shown in figure 3.5. This part must define all ports and generics according to the ports and the generics of sub-components. After defined all ports and generics, the structure is closed by “end component” syntax. The sub-components that will be used later must define in architecture before using.

The ports and generics defining statements are used in two areas as mentioned earlier. They are defined in the top-component entity and each sub-component. The port defining statement has the structure as shown in figure 3.6. It begins with the port name follows with “:”. The port direction comes after “:”. The port direction shows that the port is the input port (signal receiving) or output port (signal sending). The last part of defining statement is the data type of port. The primary data types of port are STD_LOGIC and STD_LOGIC_VECTOR. The STD_LOGIC is standard logic that represents digitally encode of hardware [22]. The STD_LOGIC vector is array version of STD_LOGIC that comprises of more than one digit. The port of the component uses as connection node between component.

The generic defining statement is shown in figure 3.6 as well. The statement begins with the generic name and follows by data type. The distinction from the
port defining statement is the last part. The port has a direction to express that input or output port, but the generic contains a value. The values of the generics can represent the properties of the component. The signal defining statement is quite identical to port defining statement except signal does not have direction as shown in figure 3.5.

```vhdl
generic
  (generic_name : data_type := value; -- generic defining statement
   DATA_WIDTH : integer := 8; -- generic example
  );
port
  (port_name : direction data_type; -- port defining statement
   clk : in STD_LOGIC; -- port example
  );

Figure 3.6: The port and generic defining statement.
```

The direct instantiation is a structure for generating an instant copy of pre-defined component. The main statement of direct instantiation is in the first line, that starts with the name of the object that duplicated from a pre-defined object following by “:” symbols. The right side of the symbol is a pre-defined object name as shown in figure 3.5. The inside of direct instantiation comprises of generics mapping statement and port mapping statement.

The mapping statements of generics are identical to the port mapping statement. The mapping statement consists of two objects. The objects are divided by “=>” symbols. The object on the left side of the symbol is the name object that it needs to connect. The object on the other side is the object that is connected. The port mapping can be a one-to-one connection such as STD_LOGIC to STD_LOGIC and STD_LOGIC_VECTOR to STD_LOGIC_VECTOR with having the same width. The one-to-many can be STD_LOGIC_VECTOR to many STD_LOGIC (1 digit of STD_LOGIC_VECTOR) and STD_LOGIC_VECTOR to many STD_LOGIC_VECTOR that has a smaller width. The object on the right side can be signal or same type object for port mapping statement. The signal is used when the interconnection establishes between the sub-components (1.2 in figure 3.7). If the sub-component is connected to object in the entity of top-component entity, the object on the right is object name of top-component (1.1 in figure 3.7). For generics, it can only map generic to generic.
The structure of top-component that is explained earlier is significant to the framework. The framework must generate top-component VHDL file by itself according to information provided by the developer.

### 3.2.2 The Design fundamental of Virtual Partial Reconfiguration

This topic explained the basic design rules of partial reconfiguration. It is necessary to know the rules to make the framework work correctly. The framework follows the rule in generating output files.

The first rule is the dynamic modules forbid to directly interconnected between them. All dynamic module connections must connect to the static module. The connection type can be a one-to-one connection and a one-to-many connection between the static module and dynamic modules. The second rule and the third rule are about the one-to-many connection. The second rule is the one-side must be a static module, and port must be vector type that has port width. The port width of the static module is needed for determining the number of dynamic modules. The many-side can be dynamic module port that is standard or vector. The standard port of dynamic module counts the port width as 1 bit. The third rule is the static part can have more than one-to-many connections. The requirement is all one-to-many connections must have the same ratio between the static module and dynamic module.

### 3.3 Requirements of the Framework

The framework supports the developer to develop efficiency. Therefore, the requirements are derived from the developer viewpoint. From the concept of the framework chapter, the Framework will generate top-component VHDL files and
User constraint file (.UCF) for each top-component. The number of the top-component files depends on the number of possible designs. One design needs one full bitstream file, and the bitstream file is generated from the top-component VHDL file and the User Constraint file.

![Diagram showing the generation of design files from the framework.](image)

**Figure 3.8**: The design files generated from the framework.

In figure 3.8, it shows how FPGA changes the design and the files that use for changing. Each design represents the static module and different dynamic modules that are placed inside FPGA. It needs two information files that are top-component VHDL file and top-component UCF file. The top-component VHDL file contains the information of sub-modules (static modules, dynamic module) that are placed inside. The UCF file contains information of hardware pins map to top-component ports. When synthesis these two files, it gets full bitstream file of the design that can load into FPGA to get the design. The outputs of the framework are the VHDL file and UCF file that can use further in Xilinx ISE tool for generating bitstream file effortlessly.

The requirements of the framework are determined from the configurations that developer can modify to help framework generate output correctly. The functional requirements of the framework are listed below:

1. **Reading and Defining properties of sub-components**. The sub-components are VHDL files that are developed. The framework needs to
collect all ports and generics data of sub-components to generate sub-component in the top-component. This information is mandatory for establishing port connections and generating components that are used in the design.

2. **Generating top-component files:** Top-component file must be generated correctly. The output files can use to synthesis further without any issue. The top component VHDL file comprises of two parts that are entity body and architecture body. The entity body comprises of the information of top-component. This information is the ports and generics of top-component. These ports of top-component are the ports that are defined to connect to the external device. For more information about port types, they are explained later. The generics in top-component are generics from sub-component. These generics provide value for the generics of sub-component by overwriting the values.

3. **Generating User-defined port types:** The user-defined port types are a unique type that is defined in this virtual partial reconfiguration framework. It uses for classifying port corresponding to port connection characteristic. The user-defined port type will clarify later.

4. **Generating User-defined generic types:** The user-defined generic types are coincident to user-defined port types which is a particular parameter for this framework. It uses to determine the generic type by considering characteristic of generics. This characteristic depends on the developer decision for the generics. The type of this generics will explain later.

5. **Computing module selection:** This requirement is the sub-component properties such as which sub-components are used in this design computation and which sub-components are defined to be a static module or dynamic module by the developer.

6. **Calculating the number of possible designs:** This is the fundamental requirement of the framework. The framework should generate all possible design that is regularly written by the developer. The parameter that is used to determine the quantity of the designs is the width of the port.

7. **The Graphical User interface of the framework:** This requirement is established for the developer. It will assist the developer to reduce development time in configuration framework. The functions which are provided in this interface are sub-component VHDL file loading, the VHDL file removing, project saving, project loading, user-defined port type selecting, user-defined type selecting, module type selecting, module usage
selecting, a value of generics inputting, and user constraint file pin interconnected.

The requirements above are fundamental requirements that the framework required for generating a full bitstream of all designs. All requirements must be achieved.

3.3.1 User-Defined Port type

These User-defined port types are a unique type of port that is defined explicitly in this project. The type that is generally mentioned in VHDL is scalar datatypes of the port such as STD_LOGIC, Boolean and Integer. These User-defined types are the type classified by port connection and port characteristics. The developer can use this type to control the ports interconnections that generated by the framework. The user-defined port types are classified as below:

1. **Clock Port**: This type of port used for specifying that port is clock port. When clock type is selected, the framework will connect this port to clock port in top-component. The framework will generate a port mapping statement to a global clock (clock in top-component). It also gives the developer to define the FPGA pin that will be connected to the external clock. The example of a connection diagram is shown in figure 3.9.

![Figure 3.9: The interconnection diagram of Global port](image)

2. **Reset Port**: This reset port type is defined for interconnecting ports to an external reset signal. When this type is selected, the framework will generate top-component reset port and wiring the selected port to top-
component reset port. The framework will assist the developer to specify reset pin for the external reset signal.

![Image of Reset port interconnection diagram]

**Figure 3.10**: The interconnection diagram of the Reset port

3. **Global Port**: The developer will use global port type for the port that needs to connect to the external device. The global port type has similar idea to clock and reset port. The different is global port will define the global port name in the entity of top-component and interconnected the port from sub-component to top-component. It connects in one to one connection, and it can be a regular port or vector port.

![Image of Global port interconnection diagram]

**Figure 3.11**: The interconnection diagram of Global port

4. **Internal Port**: The internal port type is used for interconnecting between static modules and dynamic modules. According to partial reconfiguration development principal, it prohibits to establish the direct connection between two dynamic modules. The connections of internal will be the connection between two static modules and static module to a dynamic
module. The developer can select both regular port and vector be an internal port. The framework will generate a signal that used for interconnecting the two ports.

![Diagram of an internal port](image)

Figure 3.12: The interconnection diagram of an internal port

5. **Unused Port**: The idea of an unused port is used for the port that not used in the system such as debugged ports. This type of port uses for a port that necessary for individual module development, and it does not use in the system. The framework will provide '0' input for the input port and will connect the output port to 'open' as shown in figure 3.13.

![Diagram of Unused port](image)

Figure 3.13: The interconnection diagram of Unused port.

6. **Split Port**: The split port type is defined for vector port of static module to multiple dynamic module ports. It is an essential part of this framework.
The port width of static port constrains the number of dynamic modules that can place inside the design. For example, the static port has 8 bits width (0-7). The first and the second modules are 4 bits width. It can parallel connect to static port by the first module take bit 0 to bit 3 and second module use bit 4 to bit 7. The framework will calculate width left and place module to get minimum width left. It will generate the signal to connect the ports as shown in figure 3.14. The port width calculation will explain later.

![Figure 3.14: The interconnection diagram of the Split port.](image)

7. **Global Split Port:** This type of port is used for a split port that is connected to an external device. This port type does not have port width limitation. It needs to compute the width of the port in top-component. The top-component port width must wide enough to fit all port width of dynamic modules. The framework will generate port in the entity of top-component that port width is capable of placing all dynamic module.

![Figure 3.15: The interconnection diagram of Global Split port.](image)
The user-defined port types that shown above apply to all ports in sub-components. The port type will be selected by developing to guide the framework on how to configure that port.

### 3.3.2 User-Defined Generic Type

The generic defined-port types are the unique port type that is defined in this framework. It categorises generic corresponding to usage characteristic. This type of generic assists the developer by generating generics value for all sub-component. The developer does not need to specify the value for all generics one by one. The framework will provide a generic value corresponding to the configurations that are set by the developer. The User defined-generic types categorise as below:

1. **Global generic:** This type of generics is developed for specifying a constant value to all modules. This type of generic is suitable for the generics in all sub-module needs the exact value. When developer selected this type of generics and specified the value in the framework, the framework will generate the exact value in top-component then it maps this value to all sub-module that have this generic as shown in table 3.1.

2. **Incrementing generic:** The incrementing generic is used for incrementing of sub-module generics. The generics in sub-module will get the unique value that counting-up in sequence. The developer needs to determine the starting value then the framework will increment the value in each sub-module by one as shown in table 3.1.

3. **Decrementing generic:** The decrementing generic is opposite to incrementing generics. This type of generic is counting-down instead of counting-up, and each sub-module has a particular value. To use this framework, the developer needs to provide the starting value then the framework will generate the starting value in top-component, maps the value to sub-component and decrementing each sub-component by one. The example of the decrementing value of the generics is shown in table 3.1.

4. **Arbitrary generic:** The arbitrary generic is different from the other types above. The developer needs to specify value individually for each sub-module. Although, it seems that it is similar to regular coding the framework assists in finding all the generic and grouping it for a developer to specify value faster. The framework finds all generics and displays them in a group with the name of sub-module. The sub-module name is used for
determining the location of the value that defines explicitly. The values are unique for each module as shown in table 3.1.

<table>
<thead>
<tr>
<th>Generic types</th>
<th>input value</th>
<th>Top component</th>
<th>Module 1</th>
<th>Module 2</th>
<th>Module 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Increment</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5+1</td>
<td>5+2</td>
</tr>
<tr>
<td>Decrement</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5-1</td>
<td>5-2</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.1: The example values of each generic type.

The generic types mentioned above are used to control the characteristics of the sub-module to fit appropriately to FPGA. Each type is used for different functionals such as video application that it needs to change video resolution and it should change the resolution of all sub-modules. The global generic will assist the developer to provide the resolution value, and it will change all resolution in sub-module.

3.3.3 The Number of FPGA Designs

The number of designs that will be generated depending on one of the one-to-many connection types. The one-to-many connection type limits the number of dynamic modules that can place into the design. The number of modules in one-to-many connections are limited by port width of the static module as shown figure 3.8.
The one-to-many connection is used for controlling the number of dynamic modules that can place into the design. For example, it is shown that the static module has 16 bits width port in figure 3.16. The three dynamic modules are placed in the design due to the port width limit of the static module. The port width of the static module is full that the 4th dynamic module cannot be placed. The 4th dynamic module is connected to the static module in another design (not shown in figure 3.16 example). The designs are computed by summation of the module that can fit into the width of the static module port. The framework will determine all possible designs and generates it. The designs will be the complete files that can be further used in bitstream generating in Xilinx ISE tool.

### 3.3.4 The Module Configuration

The module in this virtual partial reconfiguration framework is the VHDL component file. The framework has two properties of the module that can change by the developer. After the module is loaded into the framework, each module has the property to control module usage (use or not use) and the property to define a static module or dynamic module.

The module usage property in the framework defines that module will be used in generating process or not. If the usage property is selected “yes” the framework will include that module in design computation process. If the property is selected “no”. It will be included in calculating at that time. The developer can change this property anytime and the framework can re-generate the design combination by including the module. The second property is the type of the
module static or dynamic, but these static and dynamic are different from ordinary static and dynamic in partial reconfiguration. This static and dynamic are uniquely defined in this virtual partial reconfiguration framework. In this framework, the static module is always assigned as static. The dynamic modules can be selected as static or dynamic that will affect the combination of the design. When the module is selected as static that module always includes in design combination. The framework will store permanently port width for the module that is selected as dynamic. Therefore, all design that is generated will have the module that is selected as static. If the module is selected as dynamic, the module will include in the design if it has enough port width of the static module.

![Diagram of static and dynamic modules](image)

Figure 3.17: The example shows all designs of 4 dynamic modules.

For example, if a system has four dynamic modules as shown in figure 3.17 and the dynamic module 1 is selected as static. The dynamic module 2, module 3 and module 4 are selected as dynamic. The framework will generate three design that the three designs are (1), (2) and (4) as shown in figure 3.17. The (3) design does not have dynamic module 1; therefore, the (3) design was not generated. The dynamic module 1 is static, it must be included in all designs. If dynamic module 1 change to the dynamic that causes all dynamic modules are dynamic. The result is four designs which are design (1), (2), (3) and (4).
3.4 Summary

This chapter begins with the concept of virtual partial reconfiguration framework. The concept of the framework is to develop the tool that helps the developer produce the partial reconfiguration application faster. The approach of the other research papers provides the ways that make Spartan-6 board use actual partial reconfiguration techniques. These techniques give Spartan-6 board reconfiguration itself during runtime. Although it provides the capability to reconfiguration by the partial bitstream, the procedure in bitstream developing is redundancy. It needs to repeat the procedure for each dynamic module. When the dynamic module is relocated, it needs to repeat the procedure. For the application that continuously operates and unable to stop for reprogramming. These group of research papers are suitable for it.

In our case, the application needs all design to adapt itself corresponding to the situation. It is not mandatory to reconfigure while running. The framework is developed by the commitment to reduce development time. The framework reduces redundancy work by using full bitstream instead of partial bitstream. The framework still provides other benefits of partial bitstream such as FPGA size reduction and power consumption reduction. It explains the structure of VHDL language that is required for the framework development and the partial development rule in the second part. Last part of the chapter, it talked about the requirement of the framework that is compulsory. The framework needs to be implemented to generate outputs correctly.
4 Implementation

To develop the virtual partial reconfiguration framework systematically. The systematically build of the framework will comfort the framework when it is remodified and maintenance. The framework is divided into functional modules, and each module is responsible for each task. The structure of the framework will explain each module in detail.

This project uses a python language as the primary language. The language can develop framework systematically. The benefits of python language are python is dynamic language and object-orientation programming centric since it was developed. The Object-Oriented Programming (OOP) is the idea of seeing everything as objects. The object has two fundamental properties which are attribute and behaviour. If the object is an engine of automotive, the attributes of an engine are horsepower of the engine, pistol size, the compression ratio of the engine, engine material and the piston position. The behaviours are the action of the engine such as the four states of an engine: intake, compression, combustion and exhaust. Each behaviour is the action of the object, and the action can change the attribute parameter. The OOP programming also has a characteristic that assists in development which is called Inheritance. The inheritance is the object development technique that inherits the attributes and the behaviour of the object from the parent object. This technique assists in object organisation and objects classification. It reduces code complication and redundancy of code from the similar object. The implementation of object-oriented programming to the framework is explained later.

4.1 Structure of the Framework

The requirements and the functions of the framework are summed up in the previous chapter. The structure of the framework is classified into four modules according to their purpose.

1. **VHDL file information extraction module**: This module is used to determine the ports and the generics of sub-module. The framework needs to extract all properties of each port and each generic. It uses all information to generate the ports and generics in top-component accurately.

2. **The objects storage module**: This module purpose is the collections of storage for the module, the ports and the generics. The framework stores
each port and each generic as an individual object. The port object contains all properties of the port. The generic object is comparable in design concept to port object. The module object represents one sub-component, and it is designed to be a container of the ports and the generics in that sub-component.

3. **The output generating module:** This module contains an algorithm to generate output files which are top component VHDL files and top-component UCF files. One design has one top-component VHDL file and one UCF file. Therefore, this module includes the function to manipulation the number of possible designs then placing the port and generic code by considering the user-type of port and generic.

4. **The graphical user interface module:** This module consists of all interfaces that developer interacts with them. It comprises many initial points to execute functions. The first function is an open file for loading VHDL file into the framework. It also includes adding file and removing unused sub-component. The second function is loading and saving project to reload the edited project. The last function is a group of functions that modify port properties, port values, generic properties and generics values.

The development of four modules is developed by using a design pattern. The design patterns are the design orientation of the object-oriented programming. When developers develop the applications, it always has a similar problem occurred while developing. The similar problem that occurred usually can be solved with the same solution. Therefore, the design pattern is developed from the experience that tackled the same problems over and over. The design pattern can use as a guideline to solve the problem. The design patterns are collections of problem solver method. The famous design pattern book is “Design Pattern Element of Reusable Object-Orientation Software [23]” that most of the developer state it as the bible of the design pattern. It collects many common design problems that occurred. The pattern describes the problem by four elements. The first element is the pattern name is used as symbolic to state the short description that makes more understandable between developer in the team. The second element is the problem. The problem element clarifies the problem, and it also includes the list of conditions that should meet to use the pattern [23]. The Third element is the solution; the solution is a pattern, not an exact solution. It describes loosely structure pattern that can apply to similar problems [23]. The consequences that are the last element show the result and the trade-off when applying the pattern.
The trade-off explains the implementation concern of object-orientation programming when the pattern is reused. It also includes flexibility, extensibility and portability impact on a system [23].

The model-view-controller pattern (MVC) applies to this framework. The framework is designed by modified the idea of MVC pattern. The MVC pattern provides the flexibility to the framework. The framework changes the functional modules efficiently by applying this pattern. The basic pattern of model-view-controller has three essential elements as stated in the name of the pattern. The interaction of the three elements is shown in figure 4.1. The three essential elements are:

1. **The View module**: This module contains the interface that interacts with the user. When the user interacts with the interface such as clicking button, changing a variable value, the view module sends the changing event to the controller. The controller uses this input to control the application. The view module also updates the changing of application and displays it back to the user.

2. **The Controller module**: This module is the main module that processes the data and updates the data. It determines the event from the view module and generates the modify command by considering the user input. The modify commands are sent to the model module.

3. **The Model module**: This primary purpose of this module is the data storage for the application. It controls all interaction command to the data such as modifying data, updating the data changing to the user, storing new data and removing unused data. It waits for the action command from the controller module then modifies the data corresponding to action command. When the data are changed, it will update the necessary change to the view module for displaying to the user.
For this framework, the structure of the framework is modified from the MVC pattern. The modules of the framework are divided into the four essential elements as mentioned earlier. The first VHDL file extraction module is coincident to the controller module. Another controller module is the output generating module. The model module is the object storage module that using for ports and generics storage. The user interface module is the view module of the framework. The implementation will be explained each module separately.

4.2 The VHDL Input File Extraction Module

This module was developed to search for all ports and all generics from VHDL files of sub-components. The VHDL code has two fundamental parts as mentioned in VHDL language description chapter. The framework extracts the ports and the generics from the area inside of sub-component entity. The extraction area is the code in the blue area of the sub-module code that shown in figure 29. The blue area is the area of code that uses for defining the ports and the generics of sub-component. The framework neglects architecture area (the red area in figure 29). The architecture area is used for defining the operation inside the sub-component that is developed by the developer. The framework intends to connect sub-components in partial reconfiguration style and not modify the operation of sub-components. Therefore, the red area code is neglected as shown in figure 29.
The framework needs to realize the port defining statement and the generic defining statement to detect the ports and the generics. It must know the exact position of the port properties and position of the generic properties. This framework uses the Regular Expression (regex) algorithm to determine the ports and generics.

4.2.1 Regular Expression

The regular expression is one of the strings searching algorithms. It uses special characters and connects them in sequence to form the search pattern. The search pattern of regular expression will search for a matching pattern to the input string. The regular pattern idea is closely to the wildcard notation idea. The wildcards can be considered as a basic version of the regular pattern. It is commonly found in the search box of file explorer in the window.

Python provides Regular Expression in re module. To use regular expression command, the re module need to be imported. The regular expression uses the backslash (“\”) to indicate the special character in python. The regular expression has many special characters for finding different groups of the character set. This part of the chapter will explain only the significant characters for using in the framework. These special characters information is provided in chapter 7.2. “re — Regular expression operations” form python documentation version 2.7.15 [24].
• Exact character string: this type of regular expression keyword finds out the exact string from the input string such as the keyword is ‘entity’, it will extract the first ‘entity’ word form input strings.
• ‘.’: It is substitute symbol of every character except the newline character.
• ‘^’: It uses as a prefix of the keyword to make the keyword must match from the first character of the string in line as shown in table 4.1 below:

<table>
<thead>
<tr>
<th>Input string</th>
<th>Keyword: ‘^entity’</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘entity is’</td>
<td>not found</td>
<td>the 1st string is spacebar, not ‘entity’</td>
</tr>
<tr>
<td>‘entity is’</td>
<td>found</td>
<td>the 1st string is word ‘entity’</td>
</tr>
<tr>
<td>‘number of entity’</td>
<td>not found</td>
<td>the 1st string is word ‘number’, not ‘entity’</td>
</tr>
</tbody>
</table>

Table 4.1: The string matching result of the keyword by using character ‘^’.

• ‘\’: it uses as a prefix to define alphabet character to be a special character.
• ‘\w’: it uses for matching any letter, number and underscore. The set of this character is [a-zA-Z0-9_].
• ‘\W’: The uppercase of ‘w’ is opposite to the lowercase letter of ‘\w’. It matches the character that not letter, number or underscore. The set of this character is not [a-zA-Z0-9_].
• ‘\s’: it uses for matching any space, tab and newline character. The character set is equivalent to [ \t\n\r\f\v].
• ‘\S’: The uppercase of ‘s’ is opposite to the lowercase version. It matches the character that not space, tab or newline. The character set is the set of character that not [ \t\n\r\f\v].
• ‘[ ]’: it is used to define special sets of characters to determine more specific conditions. The first set condition can be defined from the characters such as keyword [amp] will match to the single lowercase letter ‘a’ or ‘m’ or ‘p’. It can use for defining the range of number such as keyword [4-7] [0-2] the number that will match this keyword is two-digit range from 40-72.
• Using ‘^’ inside ‘[ ]’: it means that matching any character that is not in the square bracket. For example, the keyword is [^an]. The matching character can be any character that not lowercase letter ‘a’ and lowercase letter ‘n’.
• ‘+’: it uses as a suffix after the set of condition for matching the letter until the condition change such as keyword ‘\w+’, it will get all character until a space character is found then it will stop as is shown in table 4.2.
Table 4.2: The extracted results of the keyword ‘\w+’.

- ( ): It uses to specify the extraction group of strings. It needs to use along with other characters that the other character specifies condition and round-bracket use to specify the extraction area as shown in table 4.3. The regular expression extracts the first input string, and it got two string results: ‘tomorrow’ and ‘Friday’. ‘tomorrow’ is the word that is extracted from the first round-bracket and ‘Friday’ is the word from second round-bracket. It neglects the ‘is’ word due to \w+ of ‘is’ does not cover by round-bracket. The results that are not inside round bracket are discarded.

Python uses “re.compile()” to generate keyword for matching patterns. The keyword is stored as an object in python variable. It uses the keyword variable name with the search function to find the keyword from the input variable. The argument in the search function is an input string. The function syntax is “keyword_object.function(input string)”.

Table 4.3: The extracted results of the keyword ‘(\w+)\s+\w+(\w+)’.

4.2.2 VHDL File Extraction Development

From figure 4.2, it is shown that the framework concerns the code in the blue area only. The ports and the generics are defined in the blue area. Therefore, the framework will find ‘entity’ keyword to determine the starting line of the searching area. The ‘end entity’ keyword is the ending line of the searching area. From these two keywords, the framework reduces the scope of the searching area. The smaller searching area makes the framework computes faster.
After reducing numbers of the lines, the framework searches for ports by using regular expression algorithm. To set up the regular expression keyword, it needs to know the components of the port in the port declaration statement. The port is classified into two distinct types: regular port and vector port. The components of the vector port are the variation of the regular port components. The keywords need to be determined separately.

The declaration statement of regular port consists of three components which are a port name, port direction and port data type as shown in 4.3. The declaration statement can be derived as declaration pattern. The declaration pattern uses for defining the keyword that is combined with a group of regular pattern characters. The regular port pattern combines of string, space, colon, space, string and space in sequence from left to right. The regular expression extracts three parameters that are in the round bracket and stores the parameters in three individual variables.

<table>
<thead>
<tr>
<th>Declaration Statement:</th>
<th>clk : in STD_LOGIC;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration Pattern:</td>
<td>String + Space + ':' + Space + String + Space</td>
</tr>
<tr>
<td>Regular Expression Keyword:</td>
<td>(\w+)\s+::\s+(\w+)\s+(\w+)</td>
</tr>
</tbody>
</table>

Figure 4.3: The regular expression keyword of the regular port.

The vector port declaration statement is complicated than a regular port. It has five information that it needs to be extracted. The information is a port name, port direction, port data type, Most Significant Bit (MSB) and Less Significant Bit (LSB). The fundamental concept of using regular expression is the differentiation of each keyword pattern. If the regular port keyword is not unique and the keyword can be used which port vector, it means that the port vector is duplicated. One copy is stored in regular port storage and another copy store in vector port storage. It is significant to verify the type of ports before extracting it. For the MSB and LSB, it cannot use general string pattern which is (\w+) due to the MSB and LSB of ports can be other types not only the number. For example, MSB can be an equation that the equation computes MSB from the generic values. The example of generic value is data_width and port_count. In VHDL development, the developer uses data_width to set the width of vector port for sub-modules. This method assists the
developer to adjust the width of sub-module easier and reduce the complexity in modifying. It can change this parameter in top-module, and it will change the width in the sub-modules. It keeps the constant width ratio between all modules. This method prevents the error occurs from a mapping vector port that is larger than the width of another port.

![Diagram of Declaration Statement and Pattern]

**Figure 4.4:** The regular expression keyword of port MSB and port LSB

From figure 4.4, it shows the extension pattern for extracting port MSB and port LSB. It is used with a regular port pattern to get all five components. The vector port data collects port name, port direction, port data type from the regular port pattern and port MSB, port LSB from extension pattern.

The generic defining statement also has two types which are regular generic and vector generic. The essential components of the regular generic defining statement are a generic name, generic data type and generic value. The pattern of the generic defining statement is derived as shown in figure 4.5.

![Diagram of Declaration Statement and Pattern]

**Figure 4.5:** The regular expression keyword of regular generic.

From figure 4.5, it explains that regular expression of regular generic extract three data properties. The first round-bracket is a generic name, the second round-bracket is a generic data type, and the last round-bracket is generic value. The
extracted data stores in attributes of the regular generic object. The generic data extraction has more complicated than port data extraction due to generic value. The generic value can be an array of value as shown in figure 4.6. The array of generic values combines from many strings, and it also has blank space and comma between string. Therefore, it cannot use (\w+) to get all array. The (\w+) will get only the first string.

```plaintext
DST_MAC_A |: mac_array := (x"deadface000a", x"deadface0002");
DATA WIDTH: integer := 16;
```

Figure 4.6: A comparison of generic value and generic value array.

The unique keyword is defined to get aggregate values of generic value array. The derived keyword is [^();]+ that the keyword will continue collects all characters except ‘(‘, ‘)’ and ‘;’. When the keyword met one of those characters, it will stop reading the value and sum all collected characters as one of the result strings.

The vector generic has extra two parameters that are extended from regular generic. There are Most Significant Bit (MSB) and Less Significant Bit (LSB) similar to vector port. The positions of MSB and LSB are in the middle of the declaration statement as is shown in figure 4.7. It does not place at the end of the vector port declaration statement. The keyword is developed separately. It cannot extend the regular generic keyword. The regular generic keyword and the vector generic keyword are used separately to get generic object parameters.

The vector port and the vector generic have two methods in MSB and LSB declaration. The first method begins with MSB and LSB come after it. The declaration statement uses the word ‘downto’ to define that it begins with MSB.
When the MSB swaps the position with LSB that is the second method, it uses the word 'to' to differentiate the declaration statement from MSB first statement. In VHDL language, it uses ‘downto’ or ‘to’ depend on the sequence of data while sending which data send first MSB or LSB. When the ports and the generics of the sub-component, they should be kept systematically. The framework needs all data to generate the correct top-component file. It used object-oriented programming to keep the data.

4.3 The Data Storing Module

When the framework is developed to store the ports and the generics, the ports and the generics contain many parameters. Therefore, the framework stores the ports and the generics as instant objects. The object concept allows the parameters to store inside the object which is called object attributes. The module is implemented by using object-oriented programming style.

4.3.1 Object-Oriented Programming (OOP)

The object-oriented programming (OOP) was developed to solve the complication of structural programming. The two main problems of structural programming are the poor structural model of the real world and unrestricted access to global data [25]. This project uses object-oriented programming by intending to use the structural model benefit [25]. The second problem which is unrestricted access is not interested in this framework. Python language that is used to develop the framework does not provide the private attributes for an object. Therefore, it cannot restrict the global access to the attributes. All attributes in the class can access from the outside of the class.

The structural model problem is caused by the separation between data and functions in limitation of structural programming. There is no container for grouping the functions and the data. The OOP concept defines everything as objects to make it similar to a real-world object. For example, a car which is the real-object must have their properties that always have in all car such as wheels, engine and headlight. OOP programming has two essential properties which are attributes and behaviour to describe the object [25].

- **Attributes**: The attributes in OOP are the key properties that must exist in the same kind of object. All the car must have engine, wheel and power transmission as mentioned earlier. If the car does not have a wheel, it cannot
be classified as a car. The attributes of the objects are the properties that are shared in the same kind of object.

- **Behaviour:** The behaviours in OOP are the action of the object and the outcome of the action. The actions that are existed in the same kind of object. The car brake operates the car will stop. All car must have brake operation to be the car object.

The OOP programming concept concludes that all functions (Behaviour) and all parameters (Attributes) are generated inside the objects then generate the interaction between the objects to exchange information as shown in figure 4.8.

![Figure 4.8: The Object-Oriented Paradigm.](image)

The OOP is the interactions between the objects. Each object has its attribute and behaviour. The objects can be initialised, duplicated and destroyed. This exchange of the data between object constructs an application. The fundamental characteristics of the OOP that are compulsory in development are inheritance, reusability and polymorphism [25]. A concept of the class will be explained to make the characteristics more understandable. The class contains the attributes and the behaviours of the object. It seems as stamping mould to generate the copy of the object that is called instance object in OOP. The object that is generated by class has all attributes and all behaviours of its class.
1. **Inheritance:** The inheritance extends from the class idea. The inheritance uses the concept of many objects share the primary attributes and the primary behaviours; therefore, it can have top-class that contain the primary attributes and primary behaviour. For example, a car, a truck, a motorcycle; All of them have wheels. It can develop a top-class called vehicle, and it has the wheel attribute. The car, the truck and the motorcycle are the sub-class from vehicle class. The behaviours can also use the inheritance such as the brake operation that is mandatory for all vehicles (car, truck, motorcycle) therefore the brake operation can be placed in vehicle class. The inheritance provides an ability to reduce the code redundancy for writing common attributes and common behaviours over many sub-classes. It makes the class more organise and easy to modify. For the behaviours, the inheritance provides new algorithm call overriding. The overriding method is a method to overwriting or extending the inheritance behaviours such as brake behaviour of the truck has an extra function which is air brake to assist the brake. The inheritance behaviour of truck extends the braking behaviour of the vehicle by adding air brake function.

2. **Reusability:** From the overriding concept, it provides a capability to use the developed class without messing the class. To reuse the class, the new class inherits from the developed class then overriding or adding functions in the new class. This reusability assist developer in reusing the developed class and optimising it for a new application.

3. **Polymorphism and Overloading:** This Polymorphism concept is generating the different operations of the behaviour depends on the object that it operates. For example, the donation display is developed. It has two lines display. The first line shows the name of the last donator and the second line displays the amount of donation. It connects to the controller through one function (behaviour). If the controller sends a text object, the display will collect the name into a list and display the name. When the controller sends number object, the display verifies object, the object is number, not text object and operate with different action of behaviour. It will sum up the value and update total donation on the second line. The Overloading is the name of polymorphism concept that overriding the built-in function such as ‘+’ and ‘-’ operation to give them the capability to operate on the new object. For example, the ‘+’ operator can apply on string objects to concatenate the strings.
The object-oriented concept suits for our storage module. The framework has an uncountable number of ports and generics depends on the number of sub-components that are loaded and the number of ports and number of generics in each sub-component.

4.3.2 Implement Object-Oriented Programming to the Storage Module

The requirement of implementation is the storage that can store port and its properties. The storage must keep port properties appropriately that can call out with the port when the information is needed. It is decided to keep port information as attributes inside the port object. For the generic, the storage module keeps it similar to port by defining the generic to keep itself with its information. This ports and generics also receive the benefit of OOP. The storage module of the framework stores the ports and generics as diagram shown in figure 4.9. The storage module comprises of six classes. There are two classes of port, two classes of generic, one design class and one module class.

- **The ‘VHDLPort’ class:** It is the super-class for ports collecting. It consists of seven attributes that store information of that port. The first attribute is the ‘port_name’ that is used for storing port name string. It is a fundamental parameter when declaration in top-component VHDL file. This information is extracted from the port declarations of sub-components. The ‘port_type’ is the second parameter. It is used for keeping the variable type from port declarations of sub-components. It needs for top-component port declaration. The ‘port_dir’ attribute stands for the direction of the port. It is collected for top-component port declaration statement. The fourth attribute is ‘port_user’ that is used for collecting port connection type. The developer defines this parameter through the framework user-interface. This parameter is the most important for the framework. It is used for determining the connection type and where the codes are generated in top-component. For example, if a port is “global port” type, it will be declared in the top-component entity and be directly mapped to the same name port of sub-component. If a port is ‘internal port’ type, it will not be declared in the top-component entity. It will generate the internal signal in top-component architecture instead and connecting all the same name ports in sub-component to the signal. The ‘port_liked_module’ attribute contains the owner module of the port. This parameter also needs for differentiating port that has the same name. Some port type can have the ports which the same
name but different value of parameter inside. Therefore, the framework needs to know exactly where is that port belongs to connect the port correctly without error when synthesis. The next attribute is the ‘port_width’ that the value of it is always ‘1’. This class is used for collecting all the types of the port that are not the vector port, so the port width is the constant value ‘1’. The last attribute is ‘port_UCF’ that use for storing pin information of the port. This parameter has value when the port is ‘clock’, ‘reset’, ‘global port’ or ‘global split port’. This attribute is used for generating the mapping between the pin (output to an external device) to the port (the internal connection point of FPGA). There are three behaviours in the ‘VHDLPort’ class. The first behaviour is ‘__hash__()’ function. The ‘__hash__()’ function is the built-in function of Python. This behaviour is polymorphism to overloading the built-in function. This behaviour is necessary to define for producing the hashable object. The object uses this behaviour to hash the information when comparing the same type of object such as the equivalent of two objects which the same type. The second behaviour is ‘__eq__()’ that use for checking the equivalent of two objects. This behaviour needs the hash value to compare two objects. Therefore, using this behaviour the ‘__hash__()’ function needs to be defined. The ‘__eq__()’ behaviour is the overloading form build-in function to describe how to check the object equivalent. The ‘print_port()’ behaviour that is the last behaviour is used for showing the information of port. The information of port is used for checking the collected data and debugging the framework.

• The ‘VHDLPortVec’ class: It uses the object-oriented programming property by inheritance the ‘VHDLPort’ class. From the inheritance concept, the ‘VHDLPortVec’ class inherits all attributes and all behaviours from ‘VHDLPort’ class. The inheritance relationship is shown in figure 4.9 which the white-triangle arrow. Therefore, the ‘VHDLPortVec’ has attributes for storing port name, data type, port direction, port user-defined type, port owner, port width and pin information of port. It also has two additional attributes and one overriding the default value of the attribute. The two additional attributes are the most significant bit (MSB) of port and the less significant bit (LSB) of the port. These two attributes store data in integer or string format. If it is in string format, it will use the generic value to determine the MSB or the LSB of the port. The port width of this port class is computed by using MSB and LSB. When this type of module initiates, it will compute the port width and store its value instead of constant value ‘1’
of the ‘VHDLPort’ class. In this class, it has three behaviour defining statements. The three of them are the overriding behaviours from ‘VHDLPort’ class. The first behaviour is ‘__hash__()’ function. This function must be overwritten from ‘VHDLPort’ due to the difference in the number of attributes and the types of attributes inside. To differentiate these two types of object (VHDLPort, VHDLPortVec). The hash number must be generated from all attributes in those classes. VHDLPort uses all its attributes which has seven attributes. For ‘VHDLPortVec’, it adds two more attributes in generating a hash number which is MSB and LSB. The ‘__eq__()’ behaviour is overriding the ‘__eq__()’ function of ‘VHDLPort’ by adding MSB attribute and LSB attribute into equivalent checking method. The equivalent verification is developed to check that two objects of the same type are identical. The verification function needs to define separately due to the attributes inside each class are different. The new behaviour of this class is ‘compute_width’ function. This behaviour is used for computing width when the object is initial. This behaviour needs to receive the input argument from generic values. When the MSB and LSB are the strings, the values of them are evaluated from the generic value. Therefore, the ‘compute_width’ behaviour need to take all generic values into account. The last behaviour is ‘port_print’ that is used for debugging. It needs to override for presenting the additional attributes (MSB, LSB) of this class. The inheritance relation is shown in figure 4.9.
The ‘VHDLGenerics’ class: it is the main class of the generic storage. This class contains five attributes that are essential parameters of all generic types. The ‘gen_name’ is the first attribute that uses for containing a generic name for defining generic in top-component VHDL file. The second attribute is ‘gen_type’. It uses for collecting the data type of generic in string format. The generic defining statement uses a generic data type in the declaration.
statement. The ‘gen_value’ attribute is the attribute to store value of the generic. The generic value stores the value in integer format for decimal number and stores the other type of value in string format such as a hexadecimal number. The generic value is essential for generic declaration statement. The vector port also needs the generic value for computing the value of MSB variables and LSB variables. The four attribute is ‘gen_linked_module’ that contains the owner of the generics. The owner name of generic use for clarifying the same name generic from different sub-module. The different user defining the type of the module affects the same name generic values of sub-module differently. For example, If the developer defines one generic as ‘arbitrary’ (as shown in table 3.1 of chapter 3.3.2), that generic has different values depending on developer specify a value for each sub-module. Therefore, it needs the module owner information to specify value correctly. The ‘gen_user’ attribute is a user-defined type that it effects to the generic value of each sub-module. For example, if the ‘global’ type is selected, all generics get the identical value that developer specifies. Each sub-module receives different value when the developer selects ‘incrementing’, ‘decrementing’ or ‘arbitrary’. For the behaviour, the ‘VHDLGen’ class has only ‘print_gen()’ for displaying the stored data. The data is used for debugging the framework.

- **The ‘VHDLGenVec’ class:** it is an inheritance class from ‘VHDLGen’ class. The vector type of generic store in ‘VHDLGenVec’. It adds two essential attributes for storing the vector generic parameters. There are MSB and LSB of vector generics. The ‘gen_value’ attribute of this class store value in string format. It has one behaviour that overrides the behaviour of ‘VHDLGen’. It is ‘print_gen()’ that shows all generic information of vector generic type. The ‘print_gen()’ need to modify to show the MSB and LSB values of vector generic.

- **The ‘VHDLModule’ class:** The framework uses this module for collecting the port objects and the generic objects in each sub-module object. The class generates new objects when sub-component VHDL files are loaded into the system. The attributes of the class are used for storing the module information. The module has six attributes. The six attributes comprise of two module properties, two attributes for the port and another two attributes for the generic. The two properties attributes of the module are ‘mod_name’ and ‘mod_type’. The ‘mod_name’ attribute stores the name of the sub-component. The name of sub-component uses for determining the
usage of the module in the designs. The ‘mod_type’ attribute has two options which are ‘static’ and ‘dynamic’ for the developer to define. When the ‘mod_type’ attribute of the module is ‘static’ that module will be included in all design. When the framework computes the designs, and the port width of the main static module is available, the module which ‘mod_type’ is ‘dynamic’ will be placed. The two attributes for ports are ‘port_list’ and ‘port_list_vec’. The regular ports that belong to this sub-component object are stored inside ‘port_list’ attribute. The ‘port_list’ attribute keeps all ports in a list format. The ‘port_list_vec’ attribute use for collecting the vector ports of its sub-components. The attributes for generic are coincident with attributes for the port. The attributes divide to ‘gen_list’ and ‘gen_list_vec’ to store regular generic and vector generic separately. The ‘gen_list’ and ‘gen_list_vec’ store the generic in a list format. The ‘VHDLPort’ class, ‘VHDLPortVec’ class, ‘VHDLGen’ class and ‘VHDLGenVec’ class connect to this class by using aggregation relationship as shown in figure 4.9. The aggregation relationship main characteristics are the sub-class is part of super-class, and the sub-class and the super-class have a separate lifetime. The white-diamond arrow represents the aggregation relationship. The diamond arrow side is the super-class. The sub-class is the class on the tail of the arrow in figure 4.9. From the diagram, the super-class is ‘VHDLModule’ class, and the other four classes are sub-class.

- The ‘ParameterSet’ class: This framework uses this class to save parameter values that the developer filled into a set. The developer can use this class for choosing the configured set of parameters. This class consists of two attributes and two behaviours. The first attribute is ‘mod_list’ that stores all loaded modules. The modules place in this class has aggregation relationship with ‘ParameterSet’ class. The other attribute is ‘mod_usage’. It contains the usage status of the modules. The module usage has two options which are ‘use’ or ‘not use’. If the usage status is ‘not use’, the framework excludes that module when it generates a set of designs. The first attribute is ‘print_design’ that prints out the design data. The second attribute is ‘compute_mod_usage’ that it is used for defining the default value of ‘mod_usage’ when ‘ParameterSet’ initialises the design set.

The framework stores all input information from the developer input and the VHDL sub-module in the object hierarchy. The port object contains all information of port. The generic object has the identical characteristic to a port object which
contains all generic information. The module object is the conversion of the sub-component VHDL file. It contains sub-module information by storing port objects and generic objects of the sub-module. The parameter set object contains the set of module object and the configuration of each module. Although it is the same module, the developer can configure the module diversity by changing the parameter value of the generics. After the parameters were changed, the developer can store the different configuration in the set of parameters.

4.4 The Design Computation Module

The design computation generates a set of designs that ready for synthesis. A design comprises of two files which are top-component VHDL file and top-component UCF file. The top-component contains the sub-component declarations and the wiring information between the components. This part of the framework uses all information from parameter set object, and then the framework must compute MSB of ports and LSB of ports. The MSB and LSB that are extracted from sub-component VHDL files have two formats. The MSB and LSB formats are integer and equation string. The framework requires port width to compute the designs. The framework can compute port width directly for MSB integer type and LSB integer type. Instead of the equation string type, the framework must calculate the value from the string equation before it computes port width. The framework uses values of the generics that are defined by developer substitute into MSB and LSB equations. When there are no undefined variables of MSB and LSB equation left, the framework can compute the MSB and LSB values and use that values to calculate port width.

The computation process starts by loading the parameter set into the output generating module of the framework, and then the process verifies ‘mod_usage’ attribute of the module. If the ‘mod_usage’ is ‘use’, the module will keep it for computation further. It opposites to ‘not use’ status, the process will leave that module out of the computation. The next step, the process categorises the ports into groups by using ‘port_user’ attribute of the port. This attribute categorises the port into the groups of the same connection type. The process will focus on the group of ‘split port’, the process uses this port type to determine module combination of the design as shown in figure 3.17. The concept of design is explained in chapter 3.3.4. The set of designs computation is the essential process to define which sub-components placed in that design.
This module combination of the design process begins by sorting split ports which are the same name in each group. The split port can be one group or more. However, the rules of partial reconfiguration development are:

1. **All split ports come from the same group of sub-components:** When the framework group the split port by port name. The number of ports in all group must be equality, and each split port has the same sub-component ports. From figure 4.10 (1), the split port 1 has three ports which are component 1 port, component 2 port and component 3 port. The split port 2 also has three ports and same ports which are component 1 port, component 2 port and component 3 port. Therefore, the example (1) meets the rule condition. The (2) example, both split port 1 and split port 2 has 3 component port. However, split port 1 has different members from split port 1. The split port 1 has component 1, component 2 and component 3. The split port 2 has component 4 instead of component 1. Therefore, the condition does not meet.

2. **The width ratio of port between the same component from different split port must equal:** The port width form different split ports are not necessary to have equal width. However, it must keep the same ratio. From figure 4.10 (1), all components in split port keep 1:2 ratio. Component 1 of split port 1 is 4-bit width, and component 1 of split port 2 is 8-bit width. Therefore, the component 1 ratio is 1:2. The component 2 of split port 1 is 2-bit width and 4-bit width of split port 2. Therefore, the ratio is 1:2 ratio identical to component 1. Component 3 also has the same ratio. This condition fits the rule. For figure 4.10 (3), this condition does not meet the split port requirement of framework due to the different ratio between component. The component 1 of split port 1 is 4-bit width. The component 1 of split port 2 also has 4-bit width. The component 1 ratio is 1:1. However, the component 2 of split port 1 is 2-bit width, and the component 2 of split port 2 is 4-bit width. The ratio is 1:2. The ratio difference between components, therefore the condition does not meet the requirement.
The process will select one group of the same name split port. It uses this group for computing all possible the number of possible designs. All group of the split port has the same components and same component ratio. Therefore, the design that fits on computation group can fit on the other groups. The process will determine and take out the component port that has the largest width of the port. This component is the main static module due to the regulation of partial reconfiguration that dynamic module cannot connect to a dynamic module. It can only connect to the static module. Therefore, the port width of a static module must be the largest one for the other ports connect through its. After the process separates the main static module, the process will split the port into two groups by classifying with ‘mod_type’ attribute. The first group has ‘static’ value of ‘mod_type’ and the other group value is ‘dynamic’. The developer must set ‘static’ value for all static modules and can also set ‘static value’ for dynamic modules that are necessary to place in all designs. The ‘dynamic’ value is used for the dynamic modules that not need to be included in all designs.

After the process has two groups of split port, the process sums up the port width of the ports in ‘static’ group. The summation of ‘static’ group port width will
subtract from the port width of the main static module. The ‘static’ group is always in all designs; therefore, it must reserve permanent width of the main static module. The remaining width will use to compute the module of ‘dynamic’ group. The condition in placing dynamic are mentioned earlier in chapter 3.3.4. The framework does not limit the number of components that can be loaded, the number of dynamic modules (input) is unlimited. The algorithm that the framework uses for computing the design is a Depth-first search (DPS). The Depth-first search is the sub-algorithm of dynamic programming.

4.4.1 Depth-first Search (DFS)

Dynamic programming is the algorithm to solve the problem by dividing the problem into sub-problems. The solution of the problem combines from the recursion of the sub-problem [26]. This characteristic is called the optimal solution. The problem that can use dynamic programming must have the overlapping sub-program. The overlapping of sub-program is the sub-program repeating itself when it computes deeper into sub-program [26]. The dynamic program will compute the sub-program, store result of each level of sub-program and compute the next level of sub-program. The dynamic program will be repeating itself until it meets the stop condition. The problem that can apply the dynamic programming requires two conditions which are the optimal solution and the overlapping of sub-program [26]. The dynamic programming is the optimised version of recursive programming. It stores values of each sub-module level in table format. Therefore. It prevents it from recomputing it when it has the computed value.

The simple example of dynamic programming is the Fibonacci number. The Fibonacci number of 5 is calculated from the summation of the Fibonacci number of 4 and a Fibonacci number of 3. The Fibonacci number of 4 computes from the Fibonacci number of 3 and fin a Fibonacci number of 2. From this example, it shows that the sub-problem repeats itself. If the Fibonacci numbers are computed, they will store and load whenever the algorithm needs.

The depth-first search (DFS) is one of the graph algorithms. The DFS algorithm searches a tree graph in the vertical direction first [26]. It will go deeper until it found the end of a vertex that was not possible to continue [26]. It begins with choosing one side of the first vertex such as the left side of the tree graph. It will pass through each vertex on the left side until there is no left side of the vertex left, then it will turn back then go down to vertexes that it is not explored [26]. The DFS repeats the search until entire vertexes in tree graph are discovered. The DFS algorithm that is a part of dynamic programming computes the problem result from the result of the sub-problem. The DPS algorithm can use for finding the best
result from one branch or the results of the entire tree. The DPS can be optimised to go back faster when it determined that the branch is not the best solution. It jumps back immediately, and it does no go down until the last vertex. This algorithm is called backtracking algorithm. The backtracking concept is the algorithm for finding the best solution. It does not provide all solutions, unlike the DPS algorithm.

4.4.2 Implement the Module Combinations of the Designs

The process computes the designs by modifying the backtracking algorithm. The process sorts the port width of the components in the ‘dynamic’ group. It sorts the components in descending order. This ordering of the component can reduce the number of computing operation. After sorting, the process operates from top to bottom of the tree graph as shown in figure 4.11.

![Figure 4.11: The designs computation algorithm diagram](image)

The computation uses the port width of the main static module as the initial value (M in figure 4.12). The N0 is the largest port width of the components. The port widths of components are \( N_0 \geq N_1 \geq N_2 \geq N_3 \). When M goes into N0 level (1), the process subtracts N0 from M. If M is greater or equal 0, the process goes on left-side of the N0 vertex (1) to N1 vertex (2) and stores the remaining port width of M-N0. If the remaining port width from M-N0 is a negative value, the process will
move to the right side of the N0 vertex (1) to N1 vertex (3) and cut out all remaining vertexes on the left side. The N1 vertex (2) and the vertexes after will be removed when the remaining port width has a negative value. In the N1 level, it repeats the process. If the N1 vertex (2) is on the left-side of N0 vertex (1), the remaining port width is M - N0. The N1 vertex (2) subtracts port width of itself from the remaining port width. If the result from M - N0 - N1 is greater or equal to 0, it continues on N2 vertex (4) on the left side. If the result is less than 0, it goes into the N2 vertex (5) on the right side, and the remaining vertexes on the left side of N1 vertex (1) which are N2 vertex (4), N3 vertex (8) and N3 vertex (9) are removed from computation process. If the N1 vertex (3) is on the right-side, the remaining port width is M value (not M – N0). The operation of N1 vertex (3) is identical to N1 vertex (2). It goes into sub-level until it arrives at last level. It will go back to the vertex one level above (N-1 level). If this vertex did not go to the right-side before, it would go to the right side, and the repeating process continues. This algorithm will try to place the components into the design as many as possible and minimise the remaining port width of the main static module.

The framework count number of designs by checking the direction of the vertex when going into a lower level. From figure 4.12, it given the process is on N1 vertex. If the process goes into N2 vertex (1) on the left side, it means that component can fit into the design. The process adds the component into a component list of the first design. The first design has two components which are component N0 and component N1. If the process goes into N2 vertex (2) on the right-side, the process will generate a new design (design 2) and duplicate the components that stored until the one-level (N0 vertex) before N1 vertex. The design 2 has one member which is component N0. It continues the process until the entire tree graph is visited. The concept in generating design are right-side vertex stores the component and left-side generate new design list for storing component.
Figure 4.12: The algorithm for storing the component and generating a new design.

After it processed the tree graph, all designs that were generated are unique. However, it still has some designs that are the subset of another design. The framework concept places as many components as possible into the design. The subset design means it has some port width remain that why it has a superset of its. The process removes all subset designs. The process combines the group of ‘static’ component to all designs. At this state, the framework has all possible designs. Each of this design will generate file separately. The process sends the design one by one for computing the output file.

4.4.3 The Output File Generating

This process receives the design one-by-one. It repeats the process for each design. This step generates the code depends on ‘user type’ of ports and generics. The process begins by adding the required libraries into top-component VHDL file. The first line of the file is ‘library IEEE;’ and the second line is ‘use IEEE.STD_LOGIC_1164.ALL;’.

The framework divides the VHDL file into three parts. Each part has its function for generating the code. The three functions are the entity of top-component, the top part of architecture to the line before the VHDL structure word ‘begin’ and the bottom part of architecture from the word ‘begin’ to the end of architecture. The entity function uses to generate the ports and the generics of top-component. The top-part of architecture generate component instantiation code of the component in this design and the signal declaration statement. The last function generates the direct instantiation code of the components that declared
and the mapping code of ports and generics inside direct instantiation code of the components. The functions operate in a sequence by generating VHDL code from top to bottom. Therefore, the entity function must operate first. When entity function finish generates, the top part of the architecture function will initiate. After the two functions completed, the bottom part of architecture will begin. From this point, the report will explain the top-component VHDL code by dividing into the essential components.

The top part of the architecture function needs to be explained separately. However, the other two functions are explained in the characteristic of the ports and the generic. The top part function will generate component instantiation, it takes all module objects from the design computation and generates a component declaration statement. The component declaration statement contains all port objects and all generics object declaration statement as shown in figure 4.13.

```
architecture Behavioral of g1_dynamic_a_dynamic_c_dynamic_b is

component 'mod_name' is
    generic
    (
        'gen_name' : 'gen_type' := 'gen_valve';
    );
    port
    (
        'port_name' : 'port_dir' 'port_type';
    );
end component;
```

Figure 4.13: The example of component instantiation statement.

When the top-part function generates the component instantiation statement, it needs the module name information ('mod_name' attribute) as shown in figure 4.13. It also uses information of all port object and all generic object that left inside the module objects. The function needs to generate all ports of the module object, and it takes three attributes in a port object which are 'port_name', 'port_type' and 'port_dir' to initiate the port. The function uses three generic attributes which are 'gen_name', 'gen_type' and 'gen_valve' to generate a generic declaration statement. The other statements of VHDL fare explained in port cha

- **Clock port:** When a port is selected as clock port, the framework will send this port objects to the entity function. The entity found the clock port object. It will generate the port declaration statement which is 'clk : in STD_LOGIC;'. Although, it has many clock port objects, the entity will
generate only one declaration statement. The framework sends the module object to the bottom part of architecture function. The bottom part function will find all clock port objects and generating mapping code inside its component as shown in figure 4.14. The left side of the arrow is the port name of the port object on the right side is the word ‘clk’ that framework declared in top-entity.

```
dynamic_b_inst : dynamic_b
    generic map
    port map
    (\n        g_clk => clk,\n    );
```

Figure 4.14: The clock port mapping statement.

- **Reset Port**: The reset port uses an identical algorithm to the clock port. The entity function receives the reset port objects and generates a port declaration statement. The declaration statement is `'rst : in STD_LOGIC;` . The entity generates only one statement which the specific word ‘rst’. The bottom-part function will look for reset port objects, then generate the mapping statement inside its component. The mapping statement has port name of the port object on the right side and the right side is the word ‘rst’.

- **Global Port**: This port uses two functions which are the entity function and the bottom part of the architecture function. The framework sends all global port objects to the entity function. The entity function generates a personal port declaration statement for each port. The structure of the declaration statement is `'port_name' : 'port_dir' 'port_type';` . The entity gets the attributes of the global port object and generate them according to the position in the structure of declaration statement. For the bottom-part function, it will generate mapping statement inside the component direct initiation statement. The structure of mapping statement is `'a 'port_name' => 'port_name',` . The function uses ‘port_name’ attribute two time on both side of the arrow.

- **Internal Port**: This port uses two functions of architecture which are the top-part function and the bottom-part function. The top-part function obtains the internal port objects. The function will define an internal signal
for each internal port object. This signal is essential for the interconnection between internal ports. The structure of internal signal declaration is `signal 'port_name' + '_i' : 'port_type';`. The structure uses keyword 'signal' to state that it is a signal declaration. It uses the port name of internal port object with suffix '_i' as signal name. The 'port_type' of the signal is the port type of port object.

- **Unused Port:** The developer uses this port type for individual sub-component debugging. When top-component connect the sub-components, the unused ports are not used. Therefore, the unused port needs to send its information to the bottom part function only. The bottom part function will check two parameters of the port to specify the mapping statement. The function checks object types of port first. The port object has two types which are 'VHDLPort' and 'VHDLPortVec'. After the function known the object type, it checks the direction of port ('in' or 'out'). The function will generate three types of mapping statement depends on that two information. The first mapping statement is used for both port type which port directions are 'out'. The mapping statement is `port_name => open;`. The 'VHDLPortVec' object which its direction is 'in', the mapping statement is `port_name => (others => '0');`. The input port needs to get input signal. If it does not receive input signal, the Xilinx tools will error when synthesis. For the 'VHDLPort' object, the mapping statement is `port_name => '0';`.

- **Split port:** The framework sends this port objects to two functions. The top part of architecture function generates an internal signal for interconnecting sub-components. The internal signal port of the split port is the vector only. For internal signal declaration of split port, it requires the MSB information and LSB information. The MSB and LSB of the internal signal are the MSB and LSB of the main static module, as the port width of this module is largest. The signal declaration is signal `port_name' + '_i' : 'port_type'(port_msb downto 'port_lsb);'`. For the mapping statement in the bottom part of the architecture function, the function needs to calculate the bit positions for interconnecting dynamic modules to the main static module. The port object stores the MSB and LSB attributes of its port object individually. Therefore, the LSB attribute of the port object always stores integer '0'. When the port objects connect to the split port of the main static module, the MSB and LSB of the component must continue from the MSB and LSB of the component before its. The computation method is shown in table 4.4. The function will generate two variables that are called
interconnection MSB and interconnection LSB. These two variables are used for generating exact bit position when mapping. The function uses the MSB value and LSB value of the first connected sub-component as the initial values of interconnection MSB and interconnection LSB. The interconnection values of sub-component 1 are MSB = 3, LSB = 0. For the sub-component 2, the interconnection MSB and LSB are calculated from interconnection MSB value of component 1. The interconnection LSB of component 2 equals to the interconnection MSB increase by ‘1’. The MSB of component 2 computes from interconnection LSB of component 2 sum with itself port width. For the component 3, it uses MSB of component 2 as an initial value. Component 3 repeats the computation method. If the MSB of the port object is an equation, the computation method is the same. However, the function generates the MSB and LSB as the equation with an offset value. The example of mapping statement is shown in figure 4.15.

<table>
<thead>
<tr>
<th>Module</th>
<th>Port information</th>
<th>Interconnection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSB</td>
<td>LSB</td>
</tr>
<tr>
<td>Component 1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Component 2</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Component 3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Component 4</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: The method for calculating the port width.

- **Global Split port**: The global split port is similar to the split port; however, it does not have port width limit as a split port. The split port has a port width limitation which is the port width of the main static module. The framework needs to compute global split port to find exact bit positions, and it uses the same method to split port as shown in table 4.4. The difference is the global split port need to generate top component port for interconnection to sub-components. This top component port is the vector type port that can place all sub-component. The port width of the top component port is computed by the method in table 4.4. The MSB of this port is the interconnection MSB of the last component in the sequence (MSB of component 4 = 19). For the LSB, it is always ‘0’.
• **Global Generic:** The generic object that contains this type attribute. It uses two functions to generate its. The first function is the entity function, and it defines the generic declaration statement in entity function. The generic declaration statement pattern is ‘gen_name’ : ‘gen_type’ := ‘gen_value’;

The second function is the bottom part of architecture function; it defines the generic mapping statement inside each direct component instantiation. The pattern of generic mapping statement is ‘gen_name’ => ‘gen_name’, . It uses the ‘gen_name’ attribute two times. The left side of the arrow is the generic name of the sub-component, and the right side of the arrow is the generic name of top-component.

• **Incrementing Generic:** This incrementing generic object contains the ‘gen_value’ attribute that stores the value that developer input. This value is the initial value of this generic. The entity function generates the incrementing generic declaration statement similar to the global generic declaration. However, the incrementing generic mapping statement is different from the global generic mapping statement. The incrementing generic mapping statement has one extra parameter, which is an incrementing offset. The offset increases by one per each sub-component. The first component has no offset, the second component has offset equal 1, and the third component has offset equal 2. It continues until the last component. The structure of generic mapping statement is ‘gen_name’ => ‘gen_name’ + ‘offset value’,.

• **Decrementing Generic:** The decrementing generic has similar characteristics to the incrementing generic; however, the offset of decrementing generic will subtract from an initial value. Therefore, the generic declaration statement is identical to incrementing generic. The generic mapping statement uses the minus sign instead of plus sign.

• **Arbitrary Generic:** For this type of generic, each sub-component objects has its generic value. The arbitrary generic does not have a generic declaration statement, as the generic mapping directly to its value. The bottom part function obtains all arbitrary generic objects. It generates the arbitrary mapping statement which is ‘gen_name’ => ‘gen_value’.

Figure 4.15: An example of a port mapping statement.
When the process generated an entire VHDL code of one design, it will move to
next design and repeat VHDL code generator process until the last design is
generated. The next process is to generate a UCF file, and the UCF file contains
the mapping between global ports and pins (peripheral parts). The port types that
are considered for generating UCF file are ‘Clock port’, ‘Reset port’, ‘Global port’
and ‘Global Split port’. The first three ports are mapping the port directly to the
developer-defined pin. For the global port, it needs to compute the positions of sub-
components. When the developer defines the pin value, the developer sets the pin
value directly to the port of sub-component. The pin value will store in ‘port_UCF’
attribute. The principal of the VHDL language requires using a global port to map
with a pin. When mapping vector port, it needs to specify a bit position that
connects to the pin. When the design is generated, the location of sub-components
is flexible. For example, component 1 has MSB = 3 and LSB = 0 in design 1,
however, component 1 has MSB=11 and LSB=8 in design 2. In design 1, the pin
mapping statement uses port (3), port (2), port (1) and port (0). The number in
round bracket is the bit position. Design 2 uses port (11), port (10), port (9) and
port (8) for pin mapping statement. The ‘port_UCF’ attributes have the difference
mapping statement between design 1 and design 2. The framework will determine
the position of sub-component and map pin according to it.

4.5 The Graphical User Interface Module

This module uses for interacting with the developer, and it contains many
functions for the developer to use. This graphical user interface of the framework
is developed by using the ‘Tkinter’ package. Tkinter is built-in packages of Python
language. However, it has only essential libraries. The user interface is developed
in OOP pattern; therefore, the element can be reproduced. The concept of
implementing OOP into the interface, the new object types will be generated by
inheritance the ‘Tkinter’ class. This concept assists in regenerating the element.
For example, the input element of user-port type always has two elements inside
its which are ‘ttk.label’ for displaying port name and ‘ttk.combobx’ for choosing port
user type as shown in (2) of figure 4.16. All port elements in (2) always have the
identical sub-element. Therefore, the newly developed object type will store these
two sub-elements, then reusing it for each new port that is loaded into the
framework. The number of elements of the user interface varies on the number of
input files components. The user interface comprises of three sections, which are
the main frame (1), top-left frame (2) and bottom-left frame (3) as shown in figure 4.16.

The main idea of the main frame (1) is the area for developer assign the parameter values. It comprises of three tabs which are parameter set tab, arbitrary values tab and constrains file tab. The parameter set tab will show all generics excluding the arbitrary generic. Each generic consist of two elements which are ‘ttk.label’ for displaying generic name and ‘ttk.entry’ for obtaining generic value. The generics will move from the parameter set tab to the arbitrary value tab when the develop chooses a type of generic to arbitrary generic. For arbitrary generic, the developer must individually define a generic value for each sub-component. The arbitrary generic show the generic name and the component name by using ‘ttk.label’. It uses ‘ttk.entry’ for obtaining generic value from each sub-components. The last tab is the constraint file tab, this tab use for mapping the pin to the global port as shown in figure 4.16. On top of the tabs, it has parameter set control function. This control use for duplicating the parameter value to the new parameter set. The old parameter set and the parameter set that was generated are unrelated between them. This concept provides the capability to have more than one configuration set.

The top-left frame (2) is ‘modules loaded’ frame. It displays the modules that are loaded into the framework. It provides all module parameters for selecting. The parameters that can be config are the type of the module (‘static’, ‘dynamic’), module usage (‘use’, ‘unused’). The close button at the rightmost commands the function to remove the module from the framework. The leftmost of the frame has the checkbox object that controls the display of the bottom-left frame. The checkbox object is generated by using ‘ttk.Radiobutton’ class. The checkbox is used for choosing the module that the developer would like to configure. When the checkbox selects the module, the checkbox will destroy the bottom-left (3) frame object and generate the new bottom-left frame object. The bottom-left object will generate all aggregation objects that leave inside. The objects are generated from the changed information.
The bottom-left (3) frame is used for configuring the user type information of the ports and the generics. This frame object has two tabs. The first tab shows the ports of module object that was selected in the top-left frame. The second tab frame is the generic frame, and it also displays the generic from the selected module object. In this both tabs, it displays the name of objects followed by the dropdown box. The dropdown boxes obtain the user type information from the developer. The dropdown box is an instant object of ‘ttk.combobox’ class. When the information changed, the frame (3) monitors the changed parameters of the ports and the generics. It updates the information that is stored in all port objects and generic objects. The port objects that have the exact name (‘port_name’ attribute) will update ‘port_user’ attribute according to the changed information. The generic objects also use this algorithm, and it will update all ‘gen_user’ attribute of exact ‘gen_name’ generic object. This information also triggers the change of the main frame (1), the bottom-left frame will destroy and regenerate the main frame (3) when the generic changes to arbitrary generic from the others type. It also refreshes the main frame when the generic changes back from arbitrary to the others type. The generic moves form the parameter set tab to arbitrary value tab and moves back when the other user types of generic are selected. The ports are
only displayed in the main window when the port is ‘Clock port’, ‘Reset port’, ‘Global port’ or ‘Global Split port’, as this group of port type need to define UCF pin for connecting to the external device. Therefore, the main frame (1) will refresh when the port type alternates between this group of the port type to the other port type.

The last element is ‘Menu bar’, this element contains five functions to control the framework. The ‘Menu bar’ class is an inherent class from the ‘Menu’ class of ‘Tkinter’. The class store five functions as the objects inside it. The ‘New project’ function is used for loading VHDL files into the framework. This function uses ‘askopenfilenames’ built-in module of ‘Tkinter’ for loading files. The ‘askopenfilenames’ is configured to load only VHDL file (.vhd) for preventing wrong file type loading. The ‘Add modules’ function operates close to the ‘New project’ function, however, the ‘Add modules’ will append the new load files to the stored files. The ‘New project’ will remove all stored object from memory before loading new VHDL files. The ‘Save Project’ and ‘Load Project’ are used to save and load the configuration state of the framework parameters. These two functions use ‘Pickle’ which is a built-in python module with ‘askopenfilenames’ module. The ‘Pickle’ is used for converting the objects to a binary file, then use ‘asksaveasfilename’ to save it and ‘askopenfilenames’ to load it. The framework defines the new file type which is ‘virtual partial reconfiguration’ file (.vpr) for storing framework parameter. The last function is a shutdown function of the framework.

4.6 Summary

This chapter explained the implementation of the framework by dividing the explanation base on the function of the framework component modules. It begins which describe the structure of the framework and the component modules are assembled the framework. The framework comprises four modules, and each module is one python file. This chapter explains how each module is implemented; the techniques are used for the module.

The first module is ‘VHDL input file extraction’ module. This module uses ‘Regular Expression’ technique to determine the essential parameters from VHDL code. The essential parameters are the ports and the generics inside the file. It also needs to extract all port and generic information from the declaration statement of them. It explains how the ‘Regular Expression’ keyword development to get the essential information.

The second module is ‘Data storing’ module, that is used for storing the extracted data. This module applies the Object-Oriented programming (OOP)
principle to store the data as an object. The OOP principal is many objects has identical properties; therefore, the new superclass of that objects to store the identical properties. Each object will inherit the properties from the superclass. This idea is similar to the inheritance of animal in the same species. The OOP consists of two properties which are attribute and behaviour. The attribute is the property of the object, and the behaviour is the action of the object. The subclass always inherits the attribute and the behaviour of the superclass. For example, car class and truck class inherit from vehicle class (superclass), the car and the truck get the ‘number of wheels’ attribute and the ‘brake operation’ behaviour from vehicle class. This concept is applied to the framework; the storing module can organise the data orderly such as the ‘VHDLPortVec’ class inherit from the ‘VHDLPort’ class. The ‘VHDLPortVec’ does not need to declare attributes that are already in ‘VHDLPort’ class. It only needs to declare the additional attributes and the behaviours. If a function needs to apply to both of them, it can apply over superclass, and it also affects the sub-classes. It explains the storing in the class diagram to show the relation between the classes.

The third module is the ‘Design Computing’ module, and this module uses for computing the all possible designs from the information. The information is got from VHDL file and the developer input. The technique that is applied to the framework is the Depth-First search (DFS). The DFS technique is one of the tree graph search techniques. The DFS searches in the vertical first, and it goes down into the deeper vertex until no vertex remains, then it jumps back for the other branches of tree graph until all vertexes are explored. The module modifies DFS technique for finding all possible design. In this module also contain the function to generate the set of output which consist of VHDL files and UCF files.

The last module is the Graphical User Interface module, and it shows the components of the user interface module. It explains the functions of the components and how the components interact to other components.
5 Result and Evaluation

This framework is developed to assist the developer. The framework generated top-component VHDL files instead of the developer codes its. Therefore, it reduces the development time to developing the top-component file. This non-functional requirement cannot measure the exact values. It depends on the number of lines in top-component code which vary depending on the number of sub-components. The sub-component needs multiple lines of code in component instantiation in top-component. The development time depends on many variables which are the number of top component files, the number of lines in each top-component file and the coding speed of the developer. For the number of lines, it has the Source Lines of Code (SLOC) algorithm to measure the complexity of the program; however, it cannot measure the development time. Therefore, the evaluation of this framework will focus on the correctness of the output files.

5.1 Evaluation

To evaluate the framework, the different conditions are set to determine the correctness of the output files. The verifying points of the output file are:

- The framework generates the number of output files correctly. All possible designs must be generated.
- All Ports are connected correctly according to its user-defined types.
- All Generics are generated correctly according to its user-defined types.
- When the developer changes conditions of the module and the parameter of the ports and generics, the output files must change according to its.
- The functions of Graphical User interface work correctly.
- The top-component files can synthesis in Xilinx ISE tool without error.

Three conditions evaluate the framework. Due to the condition has many parameters need be set. Therefore, the pre-condition is saved as ‘evaluating_condition_1.vpr’ in the attached cd. The first condition uses all configurations according to pre-condition. The results of this condition are two design combinations the first design should have ‘dynamic_a’ and ‘dynamic_c’. The second design should have ‘dynamic_a’ and ‘dynamic_b’. The second condition changes the module usage parameter of ‘dynamic_c’ from ‘use’ to ‘unused’. The framework should generate only one design result which has ‘dynamic_a’ and
‘dynamic_b’. The third condition changes the module usage of ‘dynamic_c’ back to ‘use’ and changes module type of ‘dynamic_a’ from ‘static’ to ‘dynamic’. The third condition result should have three designs which are design1: ('dynamic_c' and 'dynamic_b'), design 2: ('dynamic_c' and 'dynamic_a') and design 3: ('dynamic_b' and 'dynamic_a'). The fourth condition modifies the ‘PORT_COUNT’ parameter from 2 to 3. The result of this condition is one design that contains ‘dynamic_a’, ‘dynamic_b’ and ‘dynamic_c’.

From these four conditions, it can check the number of designs that are generated and the sub-components that place in each design. The ports and the generic can verify by synthesis in Xilinx ISE. It should no error found.

Figure 5.1: The design consists of ‘dynamic_a’ and ‘dynamic_b’.

5.2 Result

The framework generates an exact number of designs and the sub-component inside according to all test conditions, and then manually checked the output file is the ports, and the generics generate according to the user-defined types that were defined. The Xilinx ISE synthesis result has no error found. Therefore it can generate RTL schematic as shown in figure 5.1. This RTL schematic that is in figure 5.1 is the example of one design. All designs that were generated from the
framework are verified. All of them can synthesise and can generate RTL schematic diagram. From the RTL schematic, the interconnections of the ports are manually verified. All the ports connect appropriately according to the defined user port type. The generic values are defined and mapped correctly. For the graphical user interface, all functions display the objects and the parameters according to the stored data. When the parameters are changed, it updates the stored data.
6 Conclusion

The virtual partial reconfiguration framework is developed to assist developer to develop many configuration files faster (bitstream file). Each configuration file contains the difference modules inside. The module represents the function of the application. The regular development method the developer must write the source code of top-component module, the top component module provides the sub-module information for FPGA board. The FPGA reconfiguration means a new configuration file loaded into FPGA board. When developer need the FPGA to have diversity functions, the developer needs to write top-component source code equals to the number of functions. The development time increase exponentially due to redundancy workload in developing top-component source code. The virtual partial reconfiguration framework provides capability to generating all top-component source codes according to all possible designs. The designs are the combination of sub-module. In order to use this framework, the developer need to develop the function of each sub-module separately according to the partial reconfiguration development principle. The framework will use the information from sub-module and some required information from the graphical user interface of the framework. It generates a set of designs that can synthesis in vendor development tool without coding. The framework reduces the development time and reduce confusing when has many sub-components and need to generate all design from them.

For future work, we plan to implement I-CAP generating function into the framework. The I-CAP function increases more development capability to the FPGA board. It provides potential to load new bitstream file from FPGA internal. Typically, it needs to connect to the external device for pushing new bitstream file into FPGA. Another future work is providing more function for a developer to use such as the incrementing generic is always increment by one. Next version will give the ability to the developer to define the incrementing value step.
Bibliography


Appendix A:

This appendix explains the contains of the reference CD:

- Evaluation file folder: it contains files that used for evaluating
  - evaluating_condition_1.vpr: it is pre-configured file
  - Input folder: it contains the initial files
- Source Code folder: it contains the source of program
  - mvc_view.py
  - mvc_module.py
  - mvc_output.py
  - mvc_extract.py
- Miscellaneous folder
  - Functional requirements of the framework
  - GUI class diagram