Politecnico di Torino

Master's Degree Thesis

Routing Congestion Tracing in High-Level Synthesis Flow of FPGA based Systems

Supervisor

Prof. Luciano Lavagno

Dept of Electronics and Telecommunications

Candidate

Muhammad Tahir Rafiq

Master's Degree in Electronic Engineering

Oct 2019

Abstract

In the current electronic design, logic synthesis that starts from an RTL description of the design has been the dominant method to implement digital systems on both FPGAs and application specific chips. But in the recent times, High-Level Synthesis (HLS) has become a preferred choice of hardware designers and engineers for implementing complex digital designs. State of the art EDA flows have also incorporated HLS based design techniques.

High-Level Synthesis or HLS is an automated process that accepts synthesizable code written using high-level languages e.g. C, C++, SystemC and OpenCL (Open Computing Language) and transforms them to an RTL design. This design is then implemented on hardware devices e.g. FPGAs. FPGAs have limited hardware resources in terms of logic cells, and interconnects that contain wires that are routed to implement power supply, clock and signal nets.

During the routing process in the design implementation flow, congestion is generated if resource utilization is high or the design is very complex. This routing congestion forces router to detour the tracks thus increasing the clock period and in some cases the tool is even unable to route the design and the implementation process fails. This situation leads to difficult timing closure of design and longer design cycles. Error messages and reports that indicate routing congestion contain information only about the congested cells and congestion windows. Unfortunately there is no simple traceable link available that can help designer comprehend what section of high level code is the main source of this routing congestion.

Design tools like Xilinx Vivado Design Suite contain some information to avoid congestion but it is more relevant to the RTL descriptions and is focused on iterative patterns of RTL design cycles to alleviate congestion. Congestion report generated by Vivado indicates enormous number of complex RTL net names that are automatically assigned to the nets during HLS and that are responsible for the congestion present in the design. Although present in the auto-generated RTL descriptions of design, these complex net names are not explicitly related with the high level instructions responsible for the creation of these nets.

The main aim of this research work is to analyze the routing congestion phenomena in FPGAs and to generate a correlation between the HLS code and the congested nets and windows information reported by Vivado Design Suite during the placement and implementation phase of design flow. A novel technique has been devised that collects data generated by the tools in various files during the design flow and the result is correlation information between high-level code and the congested windows on the FPGA. This correlation information indicates the specific high-level instructions responsible for routing congestion in a quantitative manner, and is very useful for designers to eliminate congestion in early design stages without digging deeply into the auto generated complex RTL descriptions of designs. Based on this high-level congestion information, some counter measures like modifying the source code without losing functionality and efficiency of

design and the use of some suitable HLS directives, are also proposed in the end. The effectiveness of this technique is demonstrated using a Complex Discrete Fourier Transform Design to eliminate congestion at the C++ source level.

ACKNOWLEDGEMENTS

I am thankful for the cooperation and support from the professors and students of our research group, particularly head of research group and my thesis supervisor Prof. Luciano Lavagno. Without his guidance and patient supervision, this work was not accomplishable for me.

I dedicate this work to all my teachers and professors from all stages of my academic career; they were always there to rescue me from all the crisis and problems related to the academia. Learning without moral character building is useless from the perspective of contribution made towards society. I was blessed to have the mentors who were very focused on both academic aspects and the moral uprising of their students.

Table of Contents

1 High-Level Synthesis

1.1 Introduction

High-Level Synthesis transform a high level language (C, C++ or SystemC) design specifications into an RTL implementation that can be further synthesized for hardware construction on ASIC or FPGA device. High-Level Synthesis is an automated design process, to better understand this process; some basic concepts of digital design are first produced here.

1.1.1 VLSI Design of Digital Systems

To well understand the nature and evolution of High-Level Synthesis it would be quite advantageous to consider the VLSI design of digital systems. VLSI or very large scale integration is a process that yields a hardware integrated chip by combining a large numbers of transistors present on that chip. The process of VLSI began in the early 1970s when technologies related to semiconductors were evolving. From its earliest implementation compared with current high density chips, number of transistors on a unit area has increased in almost an approximate order. The very first step in the VLSI Design flow is 'Design Specifications'. In some cases Design Specifications is a simple written document but to implement a real word digital system, most often an executable model based on C, C++ or MATLAB is generated. SystmeC is a new addition in this list to create an executable model. At this stage no hardware information is included and focus is mainly to validate and verify the behavior of system. After success fully testing this model for functional accuracy, further design flow includes multiple steps that map this basic design to the actual hardware implementation. Now comes the architectural part, where first step before any optimization is to implement the desired functionality. *If the functionality defines "what" the system does, the architecture defines "how" the system does it, with direct consequences on performance, area, and power consumption.*[1]

Figure 1-1 shows a typical VLSI circuit design flow with block representing different levels of design and placed in a design flow sequence.

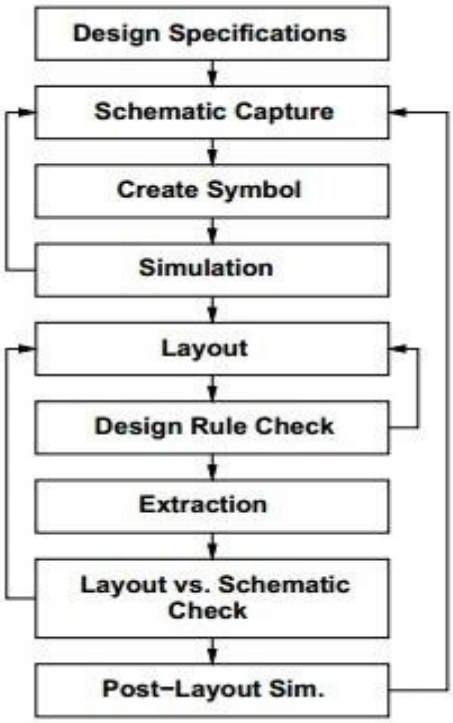


Figure 1-1: **VLSI IC Circuits Design Flow**

After the formal definition of architecture, the next step in the design flow is to generate an RTL description in Verilog or VHDL. This is a non-trivial, cyclic and extremely time consuming process in which this manually coded piece of HDL language is tested, bugs are sorted and fixed and ultimately verified through an HDL based test bench.

1.1.2 Y-Chart Based Design Methodology

Another interesting approach to view the VLSI design technique is Y-Chart Based Design Methodology. Gajski and Kuhn presented a Y-chart[2] that shows three different representations of the same system from three different angles. Y-Chart has three branches each representing Behavioral, structural and physical views. Then there are circles crossing each branch and shows the specific abstraction level.

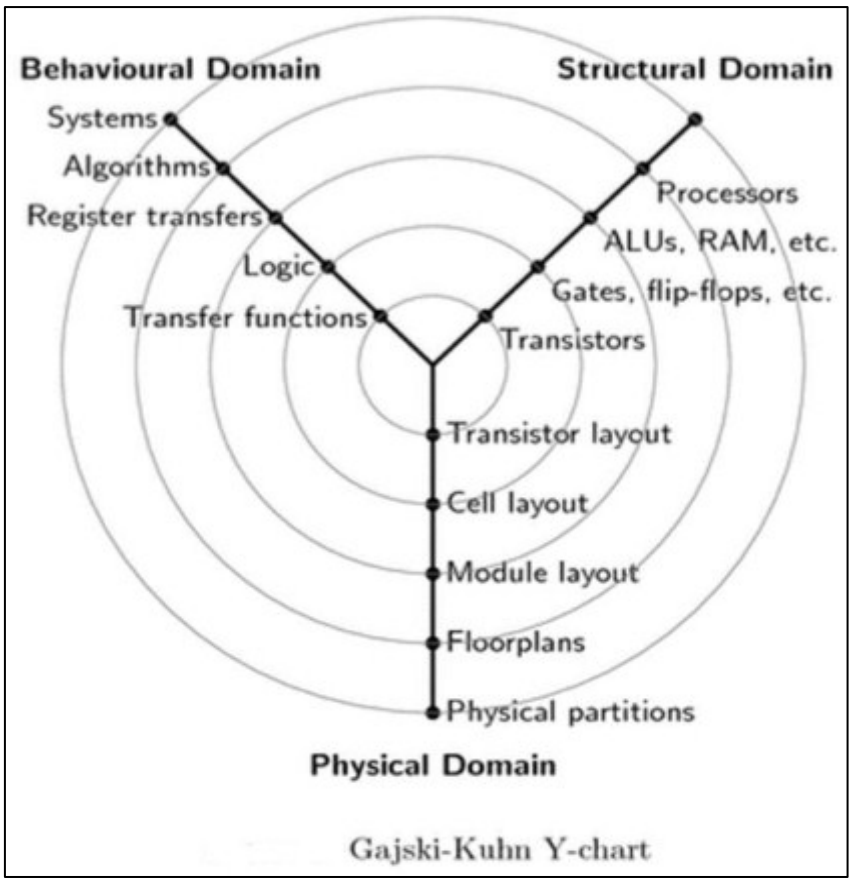


Figure 1-2: **Gajski-Kuhn Y-chart** [2]

1.1.3 Drawback of Traditional RTL Approach

With the increase in the functionality and complexity of digital designs, manually generating RTLs from behavioral models became tedious as more bigger is the system and higher is the complexity of application, there are more chances of errors and design cycle can have possibly more iteration and hence larger design time. This was not a very good aspect of modern market driven digital projects where if deadlines or not met, the only utilization of the design and project is to dispose it into the waist bin. It is very hard to design a 5G chip using the methods and tools that were evolved and been used in the past century.

Ever increasing complexity and more and more number of required transistors for a modern day digital design are a harsh reality for the designers. One more aspect that has emerged as a hurdle for the traditional VLSI approach is the low power requirement in design. Low power design for the portable systems is a major requirement and popularity and use of low power portable devices is increasing day by day. Low power design is also a must for the systems implementing IOT (Internet of Things) networks.

Higher complexity of systems, dealing with billions of transistors on a very small chip area, multi-core chip design, high frequency and low power requirements, mixed signal SOC's

with both digital and analog parts and the use of on chip testing circuitry have been proving a serious blow to the classic VLSI design approach of modern digital systems.

1.1.4 Automation of RTL Generation

To cope up the problems faced by designers in modern day digital design, High Level Synthesis (HLS) has emerged as a strong alternative of traditional manual RTL generation technique. The main task of High-Level Synthesis is to convert a high-level description of a design to an RTL netlist in an automated way. There has been devised a large variety of tools that accepts as input designs based on High-level languages, Behavioral hardware description languages and state diagrams and generates structural RTLs based on the constraints like area and delay defined by the designer.

The evolution of High-Level Synthesis can be divided into three generations, plus a prehistoric period. Prehistoric period of HLS evolution was in 1970s, first generation spans over 1980s and earlier 1990s, second generation covers time from mid 1990s to early 2000s, after second generations end to current day, we are seeing the third generation of HLS.[3] In [3] it is also forecasted that a four generation would possibly come after this third generation. Currently the dominant HLS approach is C-based and centered on data path oriented applications. Many recent EDA tools are using this approach.

1.2 Advantages of HLS

There is a gradual increase in the sale of HLS tools in the commercial market. It is shown in the figure 1-3.

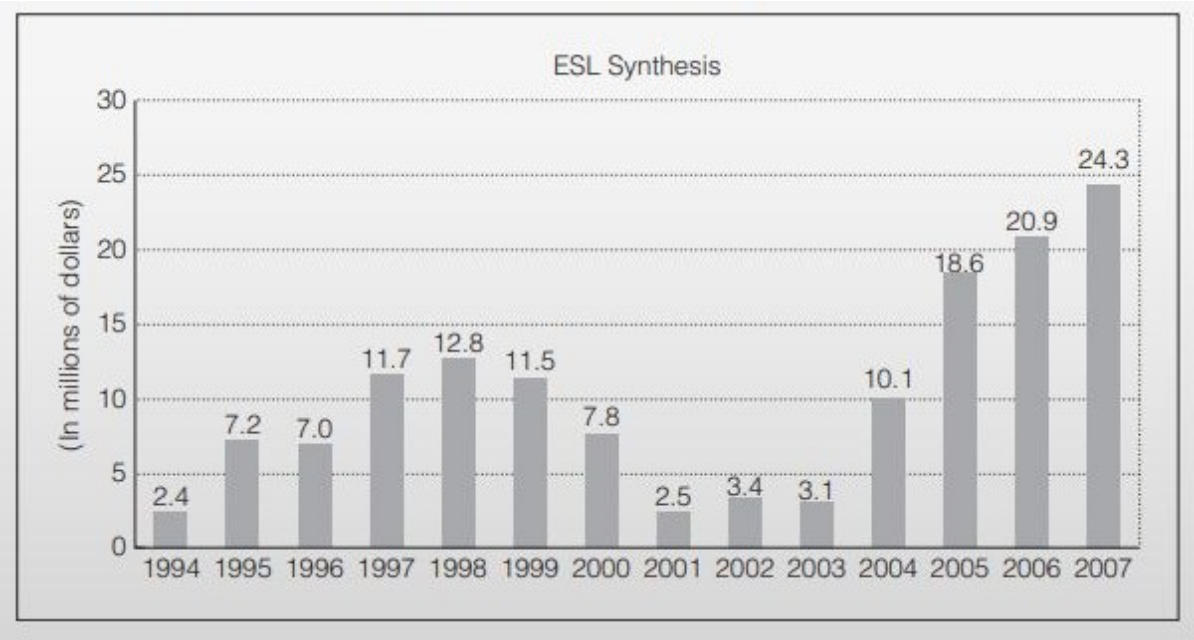


Figure 1-3: **Sales of Electronic System-Level Synthesis Tools** [Source: Gary Smith EDA statistics]

HLS is gaining a market success and popularity among designers due to its various distinct features. The main task of High-Level Synthesis is to generate error free RTL from input abstract specifications. *By using HLS, design teams greatly accelerate design time while also reducing the overall verification effort.*[1] *With High-Level Synthesis, code can be relatively easily ported from software to a hardware implementation.*[4]

The main advantages of High-Level Synthesis can be summarized in the following points:

- High-level synthesis has the ability to perform significant changes in the system in early stages of design cycle.
- A related advantage of High-level synthesis is ease of maintenance.
- Design through High-Level Synthesis offers huge savings in terms of time-to-market.
- High-level synthesis uses popular and well established high-level languages, like C and C++.
- In many cases, design of control path is mostly implicit in the language representation.
- High-Level Synthesis tools analyze the structures of algorithms like loops and branches to extract and build the control path in an automated way
- Latest state of the art High-Level Synthesis tools are capable to exploit the parallelism in high level code in the following main methods
    - Pipelining can be implemented in the design to full fill the timing constraints
    - Loop unrolling technique is applied
    - Many parallel hard ware units are built to parallelized the iterations
- Trade-off between speed and hardware resources can be made by design space exploration with different combinations
- In High-level synthesis domain, there are software profiling tools that are proved helpful in identifying the bottlenecks in the design phase
- High-level synthesis tools are capable of providing the resource estimates without actually synthesizing the resulting RTL. These estimates are reasonably accurate.
- Multiple designs are generated and simulation or verification for each design pretty fast when using HLS tools.
- *In High-Level Synthesis verification takes place at a higher level*[5]
- Some High-Level Synthesis tools are capable of generating the RTL test-benches automatically from the high level verification code.
- When code is efficiently structured, state of the art HLS tools can produce designs that are comparable to hand-coded RTL in terms of speed and resources.
- High-level synthesis has proved to be more efficient in terms of reducing design and verification efforts and effective reuse
- As High-Level Synthesis flows can save time in R&D effort when properly utilized, this saving of resources can be utilized where it really matters
- Now a day, High-Level Synthesis tools are developed and equipped with the necessary technology to make them truly production-worthy

- Earlier HLS tools were limited to data path designs but now they are capable of dealing with designs of complete systems, with control logic and complex SoC interconnects.

1.3 Limitations of HLS

While the advantages offered by High-Level Synthesis are quiet obvious, there is also the other side of the picture. Unluckily High-Level Synthesis is not as simple as compiling a code for hardware. With most tools, the algorithm must be written in a particular style to enable the synthesis tools to identify and exploit parallelism.[1] In this case code has to be restructured, in some cases even without this restructuring; HLS tools can yield a hardware design but in most cases performance of this system is poor. HLS tools have some serious limitations, use of High level languages is restricted as there are many non-synthesizable constructs that needs to be addressed. When these are handled to make them synthesizable, in many cases quality of the design is compromised. A single high level code usually generates multiple RTL, so designer has to look for the optimal choice among these designs.

When it comes to FPGA's, the ultimate design is obviously hard ware, not a software design. The synthesis languages, both HDLs and C based, describes a hard ware regardless of the fact that these languages can be adopted from software design roots. Every statement in the code yields a hardware unit that must be physically built; it is not like an instruction that is meant to be executed on a processor. When designer has a software background, he can treat the synthesis languages in software manner and it leads to inefficient use of hardware. While algorithmic representation for software is mature, for hardware realization it is still in its relative infancy in spite of ongoing research in this area.[4]

There are certain issues in hardware that is resulted through design by HLS languages. *Algorithms based strongly on pointers and pointer arithmetic do not synthesize well to hardware.*[6] Recursion, a well-established technique in software design, is not well interpreted into hardware designs. High-Level Synthesis tools yields RTL that are not well readable by humans except is case of very simple designs. Although HLS alleviates the need of RTL programming but HLS generated RTL needs to be verified. If verification fails it is hard to settle this issue. Routing congestion in FPGA based designs, when resulted using High-Level Synthesis design tools, is extremely hard to address at the High-level code. Addressing this issue is the overall topic of this thesis work.

1.4 High Level Languages used for HLS

Most prominent feature of High-Level Synthesis is that it enables the designer to program hardware systems like FPGAs using high level languages. Tools from different EDA vendors

use different high level languages to generate RTL. Following are the high level languages that are currently being used by EDA tools.

- C
- C++
- SystemC
- OpenCL
- C#
- Matlab
- Java

Output of the tools using above languages is mostly based on Verilog, VHDL, SystemVerilog or bitstream.

1.5 Tools implementing High-Level Synthesis

In the field of digital design, HLS design is turning to be a big success, that's why all major players of EDA arena have introduced their HLS tools to get the chunk of their market share. Some of these tools are Vivado HLS by Xilinx, Catapult by Mentor, HercuLeS by Ajax Compilers, Synphony-C by Synopsis, Stratus by Cadence, HLS Compiler for Intel Quartus Prime Design Software by Intel and the list continues. Three of these tools are briefly described below.

1.5.1 Vivado HLS

Vivado HLS tool by Xilinx is currently the most used High-Level Synthesis tools for FPGA based designs. *High-Level Synthesis transforms a C, C++ or SystemC design specification into a Register Transfer Level (RTL) implementation which in turn can be synthesized into a Xilinx Field Programmable Gate Array (FPGA)*.[7] Vivado HLS helps designer in focusing only the design functionality while RTL design is automatically created by tool using this functional specification. This feature of Vivado HLS is extremely beneficial in terms of design optimization and verification.

1.5.2 Mentor Catapult

Mentor has introduced its high level tool as 'Catapult C Synthesis tool'. This tool generates control based algorithm based RTL designs using C++ and SystemC source codes. These RTL can then be used for the designs meant for both FPGAs and ASICs and the verification of designs. Catapult is also equipped with High-Level Verification (HLV) tools that help designers in verification process at higher abstract level of design phase. Catapult High-Level Synthesis Platform is depicted in figure 1-4.[8]

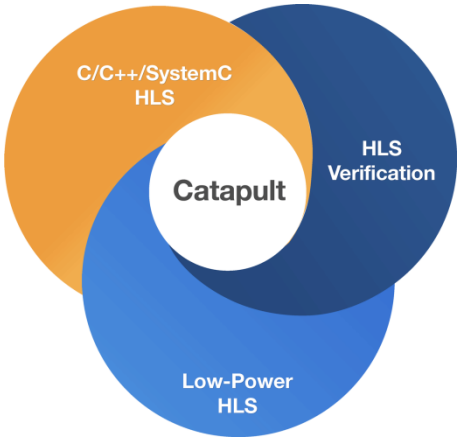


Figure 1-4: **Catapult High-Level Synthesis Platform** [8]

1.5.3 Intel HLS Compiler for Intel Quartus Prime Design Software

Altera has been the major competitor of Xilinx in FPGA based design market until recent past. In 2015, Intel Corporation acquired Altera. Intel design Suite 'Intel Quartus Prime Design Software' contains a High-Level Synthesis tool 'Intel HLS Compiler'. This tool receives untimed C++ source code on its input and produces an RTL code that is of production-quality and is fully optimized for Intel FPGA based designs. This tool also reduces verification time needed for RTL verification by taking the abstraction level for verification at a higher level for FPGA hardware design.

1.6 HLS design Flow Description

High-Level Synthesis generates RTL implementation from high-level language source code. Control and data flow is extracted from the source code and the implementation of design is carried on hardware based on defaults and on directives used by the designer. This is a very generic description of High-level synthesis. While general design flow is quite similar, commercial tools from different EDA vendors implement HLS design flow in different patterns. In figure 1-5[7] High-Level Synthesis overview of Vivado-HLS is shown.

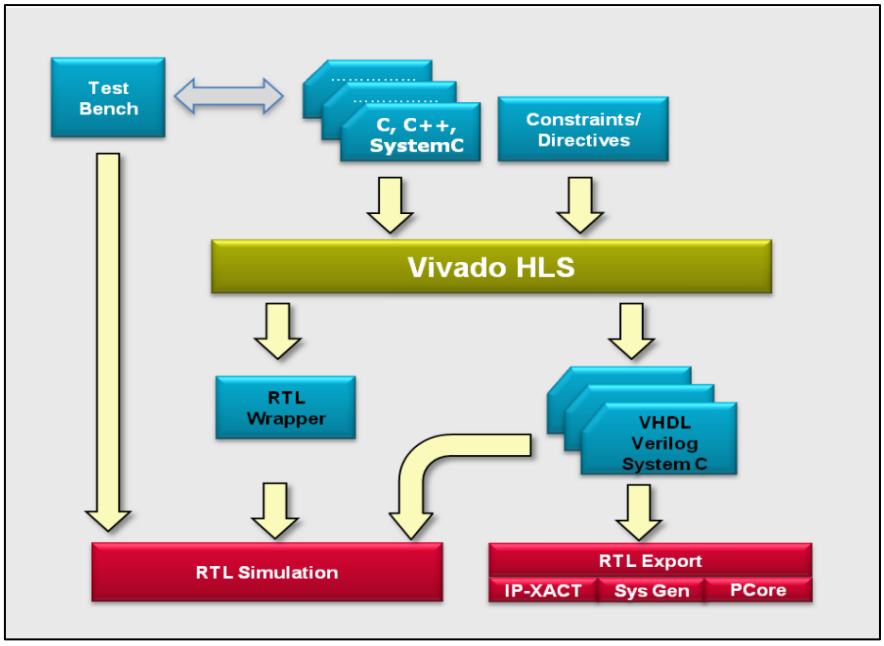


Figure 1-5: **High-Level Synthesis Use Model** [7]

Using Catapult HLS simplifies the traditional design flow by automating the RTL generation based on a higher level functional description and architectural constraints. Using C++/SystemC, compared to RTL, reduces the number of lines of code up to 80%, making HLS code significantly easier to write and debug.[8] *The HLS technique adopted by Mentor Catapult is bit different and is summarized in the following figure 1-6.*[9]

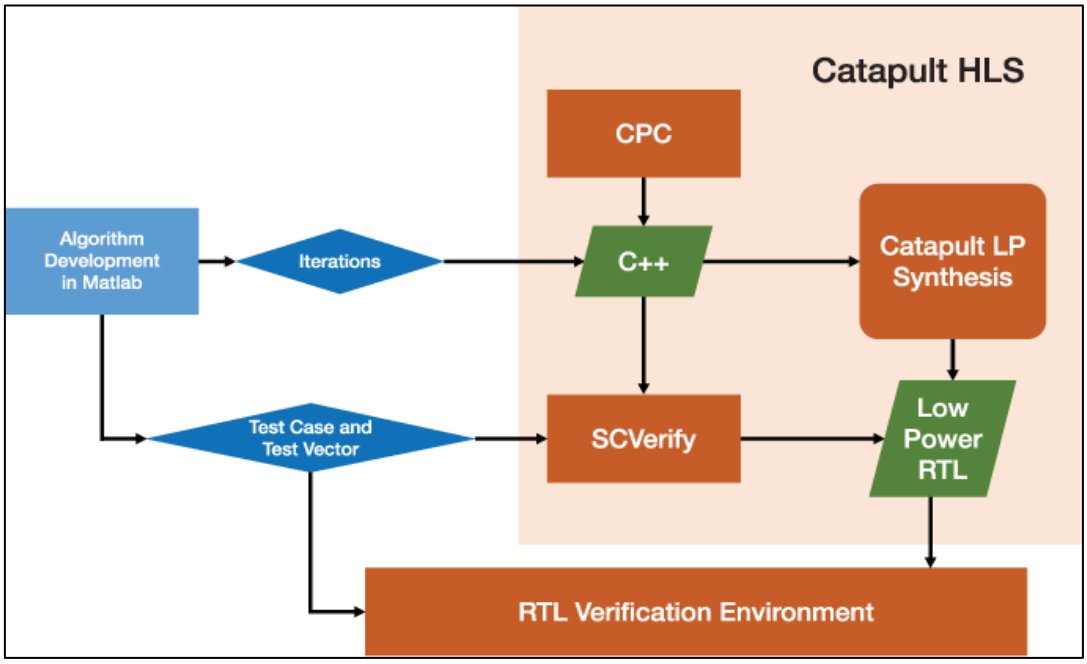


Figure 1-6: **HLS Technique adopted by Mentor Catapult**

Catapult has the distinct capability of native support for both ANSI C++ and SystemC, this feature gives designers the freedom for choosing their preferred high-level language. *The database and smart caching techniques provide at least a 10X capacity improvement, making the synthesis of large subsystems possible.*[8] *Catapult has micro-architectural exploration, which enables the designers to quickly produce higher quality designs through continuous refinement.*[9]

2 Xilinx Vivado HLS based RTL design for FPGAs

2.1 Introduction

Programming model is of paramount importance for designs based on a hardware platform. Software algorithms are generally described through C/C++ or any other high level language. These algorithms are mainly used for the development of processor based systems using software compilers. There is a huge line of processors available in the market, both general purpose and specialized processors. Specialized processors include digital signal processor (DSP) and graphics processing unit (GPU). These processors execute codes from high level languages that are based on algorithms with required functional requirements. Increasing the clock frequency has remained the key parameter to enhance the speed of software executing in these processors. Then we saw a regime shift from increasing clock frequency to adding more processing cores per chip to achieve high performance from software designs. To get full benefits from multicore processors, designer needs to be capable of efficiently using the parallelization techniques.

Historically, the programming model of an FPGA was centered on register-transfer level (RTL) descriptions instead of C/C++. Although this model of design capture is completely compatible with ASIC design, it is analogous to assembly language programming in software engineering.[10]

At earlier times, design effort to implement designs based on FPGAs was well above the typical software based systems. So FPGAs were used to implement high performance designs, which were hard to implement on traditional processors. Xilinx now claims to level this difference by the use of Xilinx Vivado High-Level Synthesis (HLS) compiler, which uses C/C++ programs to implement design on FPGAs. The comparison of RTL based FPGA design time and HLS based design time with their counterpart processor is depicted in following two figures 2-1 and 2-2.[10]

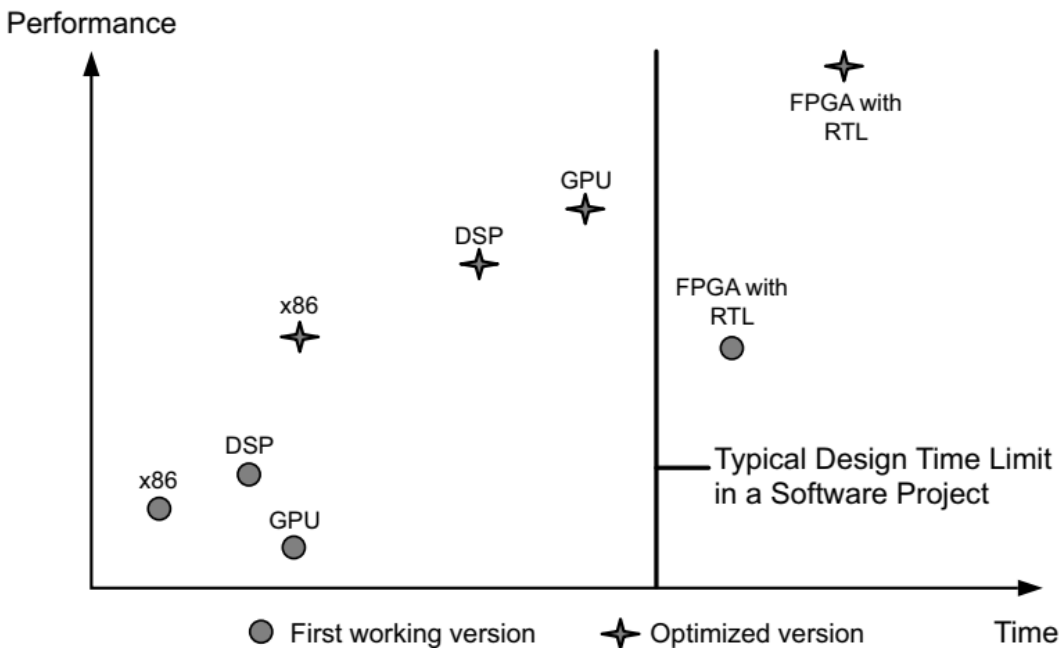


Figure 2-1: **Design Time vs. Application Performance with RTL Design Entry**

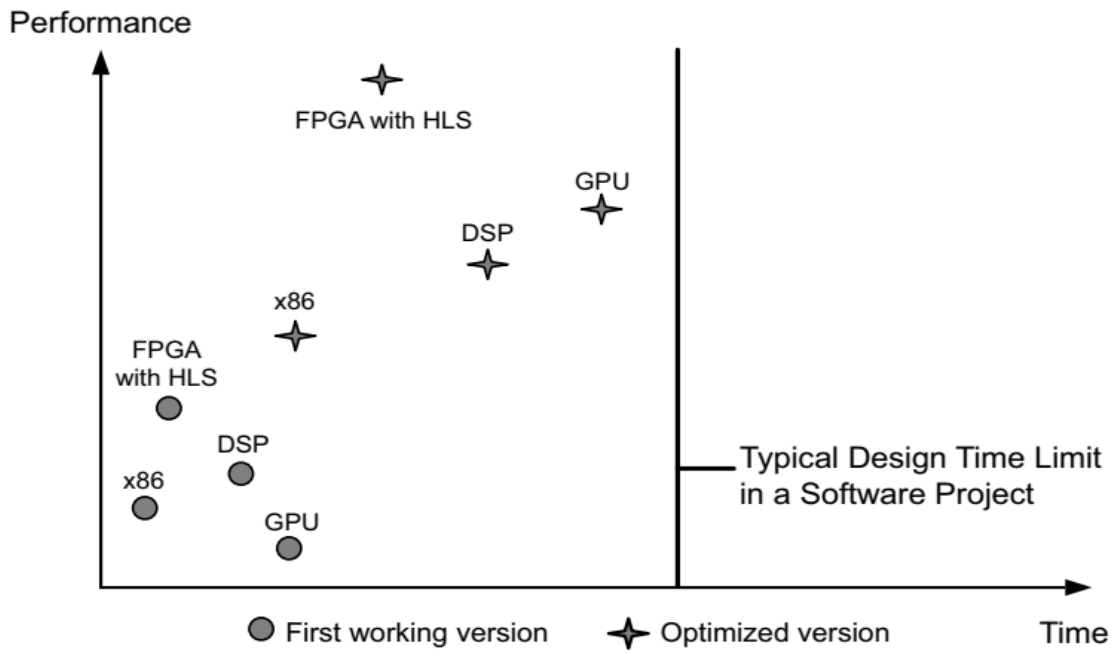


Figure 2-2: **Design Time vs. Application Performance with Vivado HLS Compiler**

2.2 Design Flow in Vivado HLS [11]

Vivado HLS flow starts with creation of a project with a source C/C++ file or files, and/or any header files and a test bench top level file. Test bench is not synthesized but used for the verification of design. Following are the design steps in Vivado HLS project design.

2.2.1 Create a New Project

- Open tool in GUI mode and create a new project and enter some suitable name
- Create/select the directory where project files are to be placed; A directly can be selected that is already having C/C++ files
- Add source/test bench/data files to the project that are present in the project directory, all header files present in the directory are automatically added to the project
- Specify the top-level function that is the main function of the synthesizable C/C++ file
- Each design can have different solutions, so select a solution name and set parameters like clock period and uncertainty. Select target Xilinx FPGA device/board

2.2.2 Validate the C Source Code

Next step in the design flow is to validate the C source code so that any error present is to be detected at the earliest stage. An advantage of Vivado HLS tool is that it can use high level test bench to verify the RTL. There is no need to specifically create a test bench at RTL level.

- Open test bench folder from explorer and double-click the top level test bench file, it is opened in the information pane
- Select the main() function in the test bench file that subsequently calls the other files to be synthesized at the end, for simulation/verification.
- Run C Simulation through button or menu.
- Result of simulation is shown on the screen

The main () function in the test bench must return some value in case of Vivado HLS. In case of successful verification of C code, the return value by test bench is zero. If any other value or no value is returned, this indicates that simulation has failed. In this case debugging of source code can be carried out. If return value is zero and simulation is successful, design is ready for high-level synthesis.

2.2.3 High-level synthesis

At this step, the design in C/C++ is synthesized to RTL design and a synthesis report is generated. This is done through Run C Synthesis tool button or from menu. On this code is synthesized and the generated report states parameters like Timing, Latency and usage of resources in full detail. Point to be noted here is that, these parameters are only approximations although very close to the final values that are obtained after place and route in the further design steps. During synthesis, functions/instructions of C code are transformed to hard ware units like Block RAMs, DSPs, FFs and LUTs. Total available hard ware resources in the device and approximated used by the synthesized design is also reported at this stage. It is helpful in selecting a device with requisite resources. As now design is at RTL level and not a mere C/C++ code, so interfaces are created and singles/control ports like clock, reset, idle, ready, start, stop, valid etc. are added automatically to the design. Based on the reported parameters, design optimizations can be incorporated in the design at this stage and design can be re-synthesized to get the optimum version.

2.2.4 RTL Verification

This is the step where advantages of HLS are more apparent. As mentioned earlier, Vivado HLS re-use the test bench that was for C/C++ design also for generated RTL design verification. C test bench also generates input vectors for the generated RTL design. RTL design is simulated during this verification step. Output vectors from RTL design are then fed back again to test bench to verify the functionality of design if there exist such signals. Otherwise again the return value of test bench main function is evaluated for verification of RTL design. If the returned value by test bench is zero, RTL design is verified and evaluated positively, otherwise verification fails. Test bench should be carefully written, so that it returns zero value only in the case when all the functionality of the design is verified successfully and all the results are correct. To execute RTL verification, use toolbar button or menu bar.

2.2.5 IP Block Generation

The end result of Vivado HLS flow is to convert the design (RTLs) into an IP block that can be further used with other tools available in the Vivado Design Suite. To accomplish this task use Export RTL button or menu bar from solution menu. IP packager generates a package that is then included and used with Vivado IP Catalog. Some other options are also available at this step. Here project can also be finished along with incorporating ‘place and route’ option in this step. IP and project files are generated in the ‘impl folder’ containing ‘IP folder’ and .zip file for IP block and Verilog or VHDL folder with project.xpr file to be used as a project. Vivado HLS can generate RTLs in both Verilog and VHDL as per the choice of

designer. After this, project can be exported to other tools like Vivado Design Suite for placing this design on a physical FPGA device.

2.3 Software Compilation Process [10]

The FPGA is an inherently parallel processing fabric capable of implementing any logical and arithmetic function that can run on a processor.[10] While considering throughput device of an FPGA device and the memory bandwidth, to fully explore the capabilities of the device, Vivado HLS uses three distinct processes. To get best optimized hardware level solution of software based design, these three processes are the integral stages of compilation process. Subsequent are these three processes.

2.3.1 Scheduling

In scheduling, different control and data dependencies between different operations are explored. Vivado HLS explores dependencies among the operations based on their relevance in time and space. Through scheduling, compiler groups multiple operations to be executed in a single clock cycle. This allows the overlapping of function calls. This overlap is also termed as pipeline.

2.3.2 Pipelining

Theoretically, Pipelining is a subject of digital design, through which data dependencies are avoided and the level of parallelism is increased while performing a hardware implementation of an algorithm. During pipelining, hardware design is divided into independent stages, and all these stages run in parallel in one single clock cycle. Data received by each stage is computed from the results of preceding stage in the previous clock cycle.

2.3.3 Dataflow

Data flow is another technique to explore parallelisms, conceptually it is closer to pipelining, but dataflow exploits the parallelism present at coarse-grain level. It is linked with functions executing in parallel within a single clock cycle in terms of software execution. Interactions between different functions of a program are evaluated to get parallelism by Vivado HLS. Simple example to understand this concept is parallelism present between functions that work individually on different data sets and there is no communication between them. FPGA resources are allocated to each function distinctly and then these hardware blocks run with having any dependency among them.

2.4 Vivado HLS TCL Command Interface

The above procedure was based on GUI of Vivado HLS. For quick iterations and for the purpose of optimization of a design, TCL command file can be used. This TCL command file is generated for all projects created in Vivado HLS and can be further modified and used with TCL interface. This file contains all commands corresponding to the design steps performed in the above discussion for the creation of a Vivado HLS project. Parameters and commands can be added/removed and parameters can be varied and this file can be executed on command line interface of Vivado HLS to generate IP blocks or HLS design projects that are exactly the same as generated in the GUI version. During GUI based design, this file is generated automatically by tool and is placed in the solution directory with the name 'script.tcl'.

3 Xilinx Vivado Design Suite

3.1 Introduction

Xilinx Vivado Design Suite is the next tool to finish an FPGA based design started from Vivado HLS, although not limited to only HLS based designs. In Vivado HLS, high-Level Synthesis was carried out from C/C++ code while in Vivado Design Suite, logical synthesis is carried out and the actual numbers of resources are reported, in case of Vivado HLS, the reported resources were just a good approximation. This is the place where the final design is integrated, we can make a larger design using IP generated through Vivado HLS and integrating it with custom IPs provided by Xilinx, third party IPs and IPs created by tools other than Vivado HLS. *A common use of High-Level Synthesis design is to create an accelerator for a CPU – to move code that executes on the CPU into the FPGA programmable logic to improve performance*.[11] In this design scheme, SoC design is implemented on Xilinx Zynq series FPGAs. HLS based IP generated by Vivado HLS can also be used inside, for system generator of DSP applications.

3.2 Xilinx FPGA Architecture [10]

Before going for implementation of a design on a Xilinx based FPGA, some information about the architecture and available resources on an FPGA is essential. An FPGA is a specific type of Integrated Circuits that can be used for multiple designs and different algorithms can be implemented on it. Capacity of FPGAs in terms of number of logic cells has increased tremendously with the improvement in semiconductor technologies and currently an FPGA can contain logics cells as high as two millions. FPGA is a cost effective solution for different designs as compared to developing a specific IC for that design. Another major advantage of FPGAs over conventional ICs is that it can be configured dynamically. This process is quite similar to loading an embedded software code on a general purpose processor.

3.2.1 Main Elements of a Xilinx FPGA

FPGA consists of following main components:

- Look-up table (LUT): For logic operations
- Flip-Flop (FF): Store data to be used by LUT
- Wires: To provide interconnections
- Input/Output (I/O) pads: Physically available ports to the exchange of Data

A basic FPGA architecture consisting of these elements is shown figure 3-1.[10]

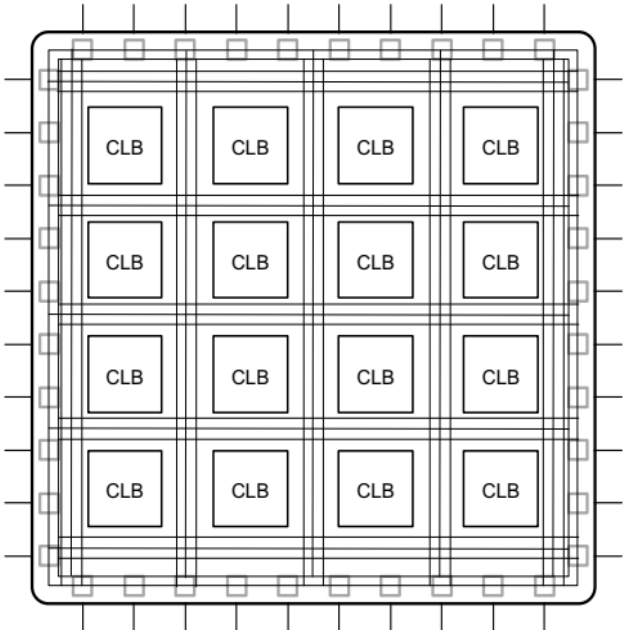


Figure 3-1: **Basic FPGA Architecture**

3.2.2 Advanced Resources on a Xilinx FPGA

FPGA with only basic elements is not that efficient in terms of throughput and high clock frequency. To address these limitations, some other computational and data storage blocks are introduce on FPGAs to enhance the efficiency and computational power. These blocks are:

- Embedded memories for distributed data storage
- Phase-locked loops (PLLs) for driving the FPGA fabric at different clock rates
- High-speed serial transceivers
- Off-chip memory controllers
- Multiply-accumulate blocks

FPGA having these elements is termed as contemporary architecture FPGA and is more flexible and capable of implementing any software algorithm. This contemporary architecture is shown in figure 3-2.[10]

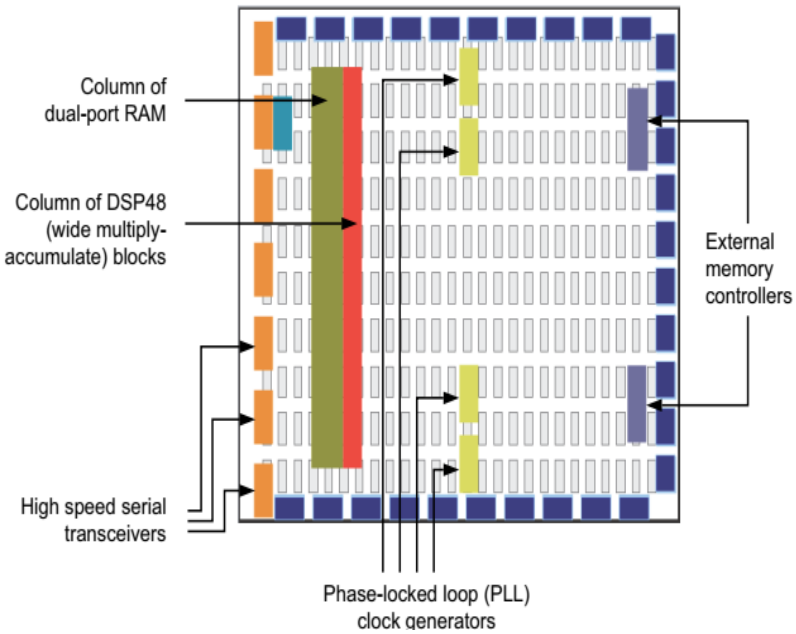


Figure 3-2: **Contemporary FPGA Architecture**

3.3 HLS IP to Final FPGA Design [11]

As discussed above, Vivado Design Suite can use HLS IPs in multiple ways to implement the final design on an FPGA. To address the congestion problem in the design, two methods can be used. First one is to use HLS IP created by Vivado HLS in IP Integrator in Vivado Design Suite. The other way is to directly open the design by selecting project.xpr file in Vivado that was created by Vivado HLS. In the first case, other IPs blocks can be integrated to get a larger design along with HLS IP created in Vivado HLS. Remaining process is same for both cases when dealing with the congestion problem in the designs created by Vivado HLS. Following is a quick overview of implementation of Vivado HLS IP based design on an FPGA using Vivado Design Suite.

3.3.1 Generate Vivado HLS IP Block

Create HLS IP block of the design using RTLs generated by Vivado HLS using C/C++ codes as described in chapter no.2 of this document.

3.3.2 Create Vivado Design Suite Project

When HLS IP block is ready, open Vivado Design Suite and create new project using new project wizard, select directory for project files, which is already containing HLS IP block to be implemented. Select project type as RTL in this case and then select the target

device/board, the same one that was used in Vivado HLS. Click finish to complete the setup of the program.

3.3.3 Add HLS IP to IP Repository

This can be done in the project manager area by clicking IP catalog, IP settings and add repository. Next browse to directory containing IP block created by Vivado HLS and select the IP from that folder. Now the added IP should be visible in the IP catalog block.

3.3.4 Create a Block Design

Create the block design using IP integrator. At this point other IP blocks can be added within this block to create an integrated Vivado design. The added customized IP blocks can be configured at this stage as per the design requirements. Then IP block are connected as per the design using external IO ports of IP blocks. External connectors are places that are to be used for the communication of this integrated design with outside word and internally these are connected the IP block ports. These connectors are very similar to physical connectors used on a hard ware design. Connect control signals within the modules and with the external connectors. When block design is complete is complete, save this design and create output products of this design by Generate option of .bd file options in the Project Manager Window.

3.3.5 Verification of Design

At this stage the integrated design can be verified by the use of an HDL test bench. Unlike the verification process in Vivado HLS, here an RTL level test bench is needed. For this verification step an HDL wrapper is created and the integrated design is enclosed within this wrapper. The HDL test bench can be included in the design by using Add or Create Simulation Sources option. After this Run Simulation option can be used to verify the integrated design that contains IP Block imported from Vivado HLS and other IPs block added with it.

3.3.6 RTL Analysis

Next step in design flow is the RTL analysis of the design. At this stage, the schematic of design is generated and can be viewed along with some reports like DRC report and Noise Report. The schematic generated for a DFT module is shown in figure 3-3.

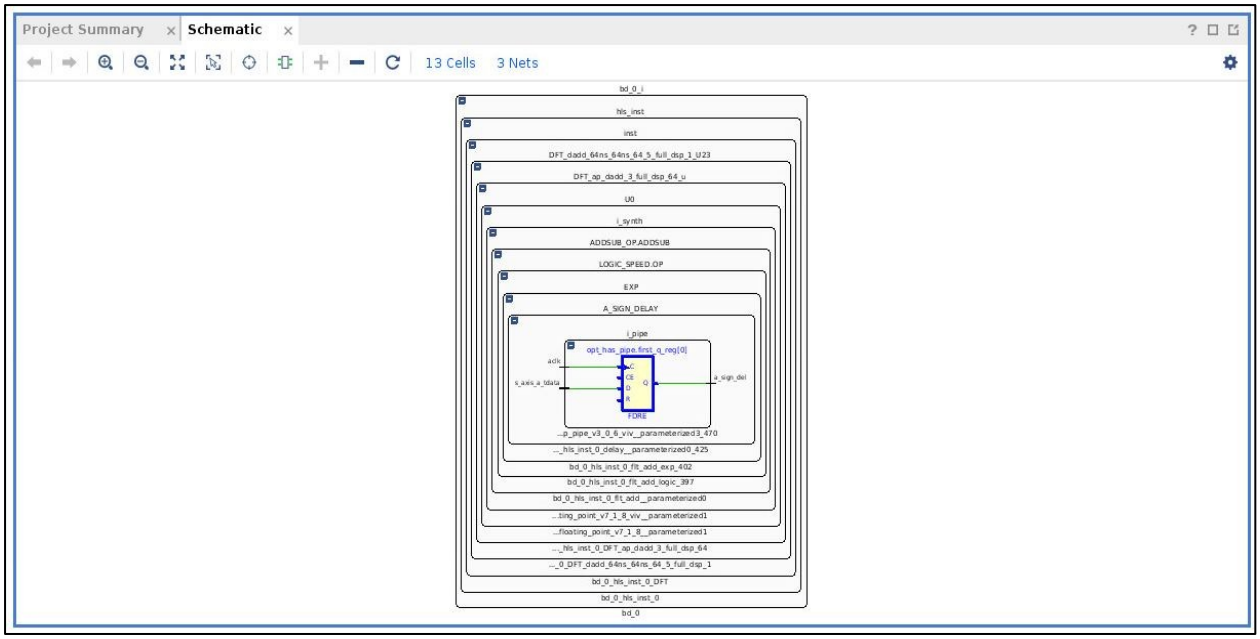


Figure 3-3: **Schematic generated by Vivado**

3.3.7 Final Steps in Design Flow

When the project in Vivado Design Suite is simulated, analyzed and verified, next steps in the design flow are Synthesis, Implementation, Program and Debug. Synthesis here is logical synthesis and not the HLS synthesis that was carried out in Vivado HLS. Here the actual resource usage is reported the utilization report. Next step is implementation; at the end design analysis report is generated that contains parameter like Timing, Complexity and Congestion. For addressing congestion problem it is the final step needed. Final step in design flow is the Program and Debug in which Bitstream is generated that is used to construct the design on the physical FPGA device.

3.4 TCL Console Based Flow

The Tool Command Language (Tcl) is the scripting language integrated in the Vivado® tool environment. TCL is a standard language in the semiconductor industry for application programming interfaces, and is used by Synopsys® Design Constraints (SDC).[12] All the design steps, tools, menus and icons present in Vivado Design Suite GUI, like the ones described above can also be used in the command line mode using TCL commands. A TCL console is available in the Vivado Design Suite GUI where these commands can be inserted for execution. In case command is generated using any menu, tool etc. TCL console automatically generates the corresponding command and the corresponding processing is also reported on the console containing operations performed, errors, warnings and creation of files reports etc.

4 Routing Congestion in FPGA Based Designs

4.1 Introduction

High-Level Synthesis is becoming industry standard for VLSI design but it also have some limitations. One of these issues is congestion during the routing process of custom chips and FPGA based designs. Routing congestion is not a new concept specific to VLSI design. It has been a problem with the traditional HDL based designs but the level of severity of problem is case of High-Level Synthesis is very high. *Although routing congestion manifests itself only at the very end of the typical synthesis-to-layout flow, it can lead to unacceptable design quality and lack of design closure.*[13] In this scenario, the best choice is to predict this issue of congestion at higher levels in design flow. In case of a congested design, automated router is left with a very few choices to route the design while minimizing the large wire delays in congested nets and achieving the stringent timing constraints. Routing congestion can lead to very long automated routing process durations, degradations in the performance of the systems, decrease in the yield of the final products and failure of routing process in designs where routing congestion of very high degree is present. On one side the negative impact of routing congestion is on the shoot due to the increase in the complexity of digital designs and technology scaling, on the other hand modern state of the art EDA tools are unable to fully address this problem.

4.2 Congestion in Routing Process

As explained in the introduction of architecture of FPGA, standard cells are present on the chips for the implementation of synthesized designs generated through RTLs or High Level based designs. In a standard cell there are wires present to implement clocks, signal tracks, and power supply lines. To route all these signals, only a limited set of wire resources is available. With improvements in technologies, number of standard cells on a unit area is increasing but on the other hand electrical characteristics of metals used for wires are not coming up to the same level. In FPGAs, standard tracks are available for the global clock signals and power supply lines. In a full custom design, clock and supply tracks are routed first and then signal nets are routed, that's why signal nets are more prone to routing congestion. *A design is said to exhibit routing congestion when the demand for the routing resources in some region within the design exceeds their supply*.[3]

There are routing tracks grouped together and contained in a bin. Routing congestion can be avoided if the signals to be routed through a bin do not exceed the number of tracks available in the bin for routing. Normally is routing is carried out in two stages, a global routing, that is applicable to the entire design and this stage is followed by detailed routing that deals with small regions having few bins within its jurisdiction at a time. In this process,

nets of design are assigned to tracks in the bins. During the routing process, router tries to accommodate wires on the tracks available in the bins. During this process, if there arises a situation when router has to accommodate more wires than the available tracks in the bins, it attempts to find alternate bin with some free tracks available to route these excessive wires. If router fails to accommodate all wires even after trying to fit them on alternate bins, the routing of all wires may not be completed and not all nets are connected as per the design. The occurrence of this situation is termed as routing congestion. This is the indication of limitation of tracks in some regions to successfully route the all signal nets in those blocks.

4.3 Drawbacks of Routing Congestion

During a design, effort is put to minimize the routing congestion for the successful implementation of design. Routing Congestion in a design can lead to following problems.

- Decrease in the performance of the design
- Increase in the uncertainty in the closure procedure of design
- Decrease in the yield of IC manufacturing process in terms of functions and parameters
- Failure of router in the final routing process
- Difficult assignment of memory interfaces in FPGAs
- Degradation in optimal quality of results (QoR)
- Tight floorplan constraints
- Incorrect estimation of net delays
- Reduction in slack available
- Clock skew and uncertainty issues
- Sub-optimal Placement

4.4 Routing Congestion Estimation

The accurate measurement of routing congestion can only be computed after the routing process has finished. The congestion reported at this final step of routing is problematic as now there would be a need of new design iteration with the necessary changes to cater the problem of routing. Even at this stage, for the designer to be sure that the modified version of design is capable of addressing the congestion issue of previous design, he requires some information before hand to make things working in the desired way. To fulfill this requirement, *several congestion estimation metrics and schemes applicable to different stages of the design flow have been developed over the years*.[14] These metrics are very handy for the designer to make a prediction of final routing beforehand and helps him in a routing-congestion free design flow. These metrics are generated by the EDA tools on the different stages of design cycle. During optimization stage of a design cycle, these metrics are very useful for the designer in making decisions.

Following is a list of contents of a congestion report generated by Vivado v.2018.2.1

1. Placed Maximum Level Congestion Reporting
2. Initial Estimated Router Congestion Reporting
3. Routed Maximum Level Congestion Reporting
4. SLR Net Crossing Reporting
5. Placed Tile Based Congestion Metric (Vertical)
6. Placed Tile Based Congestion Metric (Horizontal)

4.5 Diagnosing Congestion in Xilinx FPGA Based Design [14]

During the routing process, if critical paths are to be routed in a congested region or even near to it, it becomes difficult to meet the timing constraints. This issue also flags when device resources are used up to a higher level and thus after placement, it is really difficult for the user to route the device. Placement and routing are the most critical steps in a design implementation of an FPGA after successful synthesis process. Another problem is the time it takes router to finish the routing process in case of a high level of congestion is present in an FPGA based design. *If a path shows routed delays that are longer than expected, Xilinx recommends analyzing the congestion of the design and identifying the best congestion alleviation technique*.[14]

The architecture of a Xilinx FPGA device contains interconnects that are of various lengths and are spanned in each direction; East, West, North, South. Congestion is flagged for an area defined in a square shape that consist of interconnect tiles (INT_XnYm) that are adjacent or CLB tiles (CLE_M_XnYm) where the usage of interconnect resources is near or above 100% in some particular direction. Vivado reports a congestion parameter that is called congestion level; it is always a positive integer that indicates the side length of the congested square. In figure 4-1[14] sizes of congested areas are reported on a Xilinx UltraScale device against clock regions.

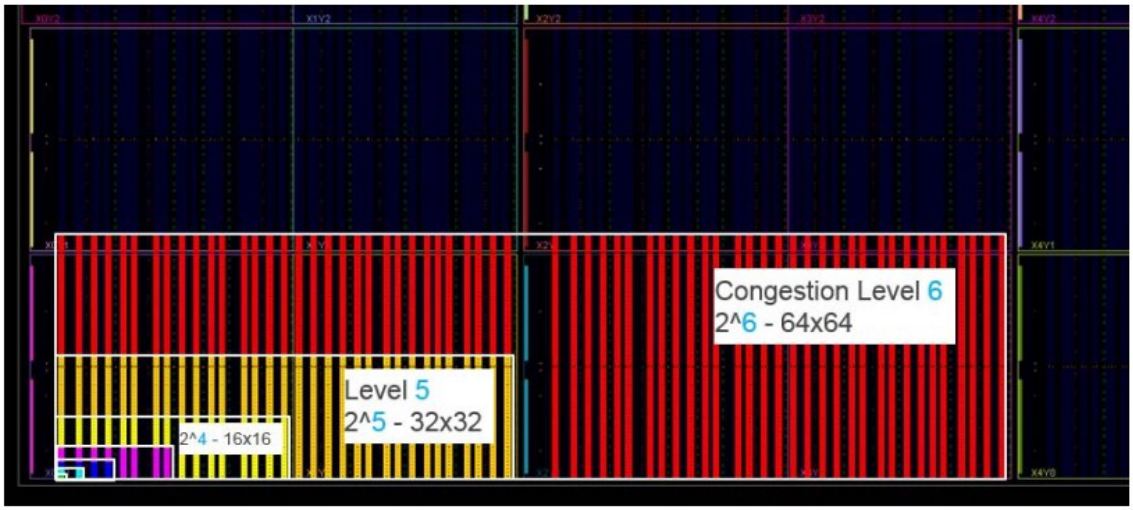


Figure 4-1: **Congestion Levels and Areas in an UltraScale Device View** [14]

Congestion level is defined above and Xilinx tools report congestion pattern based on a specific pattern, this pattern is reported in figure 4-2

If the reported congestion level is 5 or higher then, QoR would always be degraded and router would take longer times to finish its job.

Level	Area	Congestion	QoR Impact
1, 2	2x2, 4x4	None	None
3, 4	8x8, 16x16	Mild	Possible QoR degradation
5	32x32	Moderate	Likely QoR degradation
6	64x64	High	Difficulty routing
7, 8	128x128, 256x256	Impossible	Likely unroutable

Figure 4-2: **Congestion Patterns** [14]

During design implementation process, routing congestion per CLB is also reported, but this is based on estimation and not on the actual routing process. This congestion parameter is reported is Vertical and Horizontal Routing Congestion per CLB. This parameter provides a quick graphical view of congested spots on the device layout. For a specific design with a high level of utilization of resources and net-list complexity, after placement it contains many congested areas and these are shown in figure 4-3. [14]

This congestion parameter of Vertical and Horizontal Routing Congestion per CLB is reported as a device metric and was a part of congestion report generated by Vivado Design Suite versions up till 2018 but in 2019 version this parameter is not reported in the congestion report.

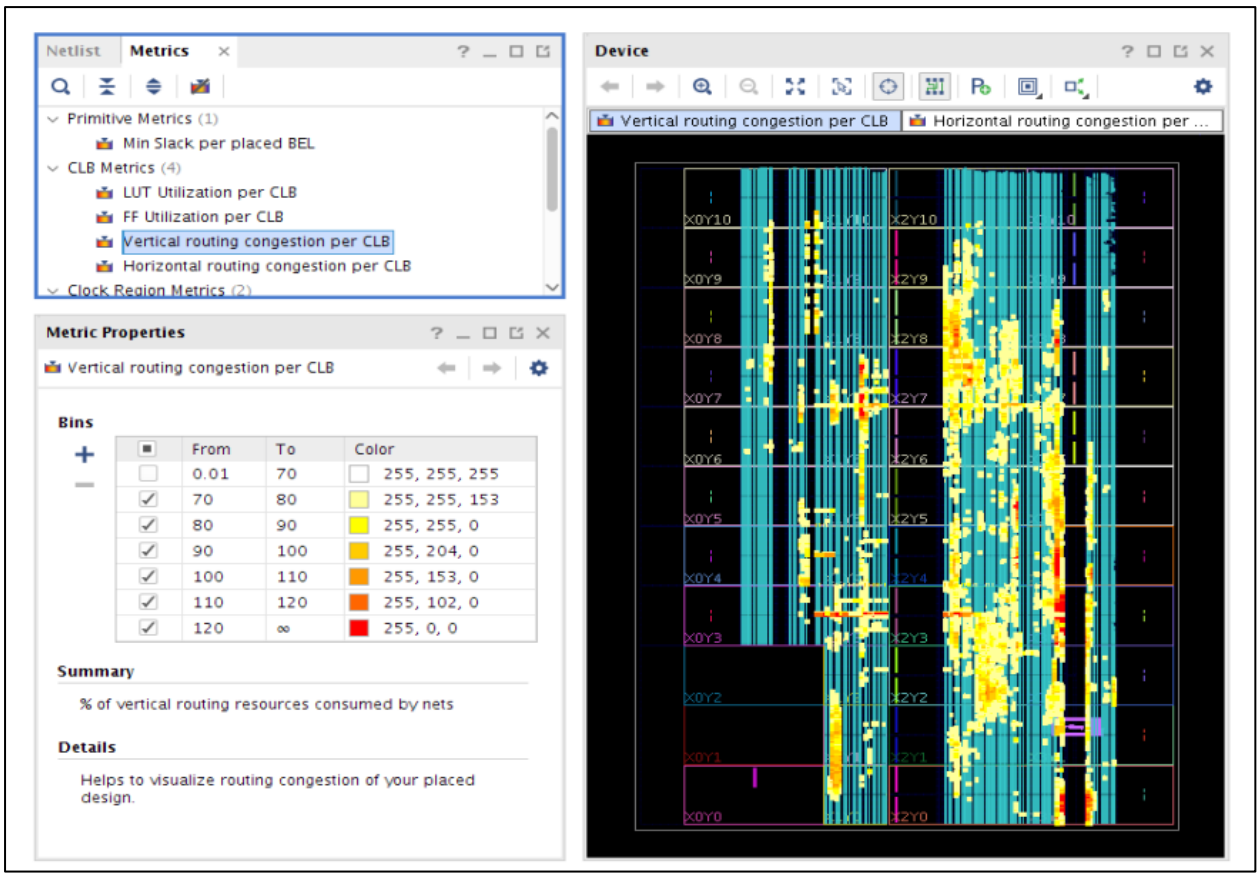


Figure 4-3: **Example of Congestion in the Device Window** [14]

4.6 Methods for Reducing Congestion [14]

Congestion is the routing process is generated due to multiple factors and it is a complex issue that does not always have a trivial solution. In case of Xilinx FPGA based designs, routing can be addressed using the same techniques that are applied to resolve the complexity issue. This issue becomes more difficult to address when complex modules are placed in the congested windows. Xilinx tools recommend various techniques to address the issue of congested coupled with complexity of the designs implemented on FPGAs.

To improve congestion issue, it is recommended to have well defined and observed constraints in the design. Over lapping of Pblocks, which results during floor planning if same region of chip is used for multiple components e.g DSPs, can be a source of congestion and must be avoided. Excessive hold time failures or negative hold slack causes the router to detour and can be resulting to congestion. When resource utilization is of higher level like 75% or above, the placement becomes more difficult if the complexity if netlist is also at higher level. Placement of high performance design is also a challenging task. In this case, recommended strategy is to review the design features and remove the modules that are not critical until the resource utilization is reduced to a suitable level. If this process of logic

reduction is not possible, a larger FPGA can be selected, or other congestion alleviation techniques mentioned at the end of this section, can be used in this scenario.

High fanout nets are a major source of congestion and tracing them can be helpful for fighting congestion. Particularly control signals other than clocks with high fan-out can cause congestion. Some other techniques that are applicable in some specific situations are suggested in UltraFast Design Methodology Guide [14] and are reported below. Further detail can be found in this Xilinx guide.

- Use Alternate Placer and Router Directives
- Turn Off Cross-Boundary Optimization
- Reduce MUXF Mapping
- Disable LUT Combining
- Limit High-Fanout Nets in Congested Areas
- Use Cell Bloating

5 Tracing Congestion back to High-Level Design

5.1 Introduction

Up-till now a brief introduction of a complete HLS design flow is presented, taking HLS based design implementation on a Xilinx FPGAs as a templetae using tools like Vivado HLS and Vivado Design Suit. In this chapter the complete flow is reiterated based on a practical HLS design example that results in a final congested implementation on the selected FPGA. Then the congestion in the implemented design is discussed and a practical approach is presented that traces back this congestion in the design to the high-level code in a quantitative way. The technique that is used to evaluate this linkage is in evolution phase and it is applicable to designs with only one high-level code file to be synthesized and without any external library functions to be used in this code file. The second condition can be relaxed with the use of Vivado HLS pragmas.

5.2 Discrete Fourier Transform Implementation

Test example design that is used for the demonstration of the back tracing of congestion to HLS level is based on implementation of a two dimensional Discrete Fourier Transform (DFT). Discrete Fourier Transform is extensively used in digital signal processing systems and scientific computing designs. *More specifically, multidimensional (MD) DFT is used in imaging applications which need frequency-domain analysis, such as image watermarking, finger print recognition, synthetic aperture radar (SAR) processing and medical imaging.*[15] A very basic and simple implementation of 2-D DFT [15]algorithm is implemented in HLS and its code, NDFT.cpp is attached as Appendix A in this document. The letter N in the file name indicates Numerical DFT. During processing for DFT calculation of a complex sampled signal, sin and cosine functions are required to be calculated. For finding these functions, generally libraries like math.h of C++ and hls_math.h of Vivado_HLS are used. In this case custom functions are written based on the numerical evaluation of sin and cosine functions using Maclaurin Expansion these functions. Results obtained using these functions are not very accurate but the major aim of this design implementation is to obtain correlation between implemented congested design and its high-level code, for this task the used functions are reasonably accurate. An in-depth analysis of DFT algorithm and its implementation can be found in the book Parallel Programming for FPGAs.[16]

NDFT.cpp is the base for this congestion analysis. The method that is reported in this section can be applied to any high-level design for getting the linkage between high-level code and the routing congestion in the final design after synthesis, placement and routing. In NDFT.cpp three HLS pragmas are used. #pragma HLS PIPELINE and #pragma HLS UNROLL generate pipelined and unrolled synthesized design at RTL level corresponding to code blocks where they are used to get higher level of parallelism and resource sharing. #pragma

HLS INLINE is used to *remove a function as a separate entity in the hierarchy. After inlining, the function is dissolved into the calling function and no longer appears as a separate level of hierarchy in the register transfer level (RTL). In some cases, inlining a function allows operations within the function to be shared and optimized more effectively with surrounding operations. An inlined function cannot be shared. This can increase area required for implementing the RTL.*[17]] Some redundant code is also a part of this example to obtain the required level of utilization of resources available on the selected FPGA, to get a congested implementation of synthesized design.

5.3 Back-Annotation Flow

Before starting the practical work on tools, here is a high-level brief descrioption of what we are going to do next in this section to achieve out task. First the design is synthesized in the Vivado HLS and source level correlation information is extracted from Vivado HLS database. A list of all nets is formulated based on this extracted information and is termed as HLS_Nets. Then design is exported and physically synthesized and implemented and routing congestion profile is reported. Based on this congestion information, CLBs above to a certain congestion threshold are selected and the nets corresponding to these CLBs are extracted and termed as Congested_Nets. Then a correlation is made between these two lists of nets and back-annotation information is formulated, describing the high-level source code responsible for congestion. Finally the design is modified based on this information and the process is repeated on the modified design to analyze the effectiveness of this routing congestion estimation technique.

5.4 High-Level Synthesis of NDFT

In the following, complete step by step procedure for the High-Level Synthesis is described for the DFT design using “Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC v2019.1 (64-bit)” which is a tool for high-level synthesis in Xilinx “Vivado v2019.1 (64-bit)”design suite. The Xilinx FPGA that is selected for this congestion analysis is xc7z020clg484-1 that is a part of Zynq-7000 series. This description is based on a system equipped with Linux version of Vivado v2019.1 but it is equally applicable to Windows version of Vivado with some minor step modifications related to launching the tool. For further help in this regard, Vivado User Guide High-Level Synthesis, UG871[11] can be consulted.

5.4.1 HLS Project Creation

Create a directory named DFT in the Home directory of a Linux based system having Vivado v2019.1 installed with relevant licenses. In the directory DFT, place the file NDFT.cpp or

create a new text file there, paste the code and save that file as NDFT.cpp. Then open a Linux terminal in the same directory by right click mouse and select “Open in Terminal”. Terminal opens with bash prompt.

In the terminal issue following command;

bash-4.2$ source /tools/xilinx/Vivado/2019.1/.settings64-Vivado.sh

Consider the path of Vivado directory as per the location of Vivado setup in the system. This command initializes the Vivado v2019.1 design suite in the system. Next step is to launch the GUI of Vivado HLS. This is done by issuing the following command in the terminal.

bash-4.2$ vivado_hls

Launch message of initialization of Vivado HLS GUI is displayed in the terminal as shown in figure 5-1.

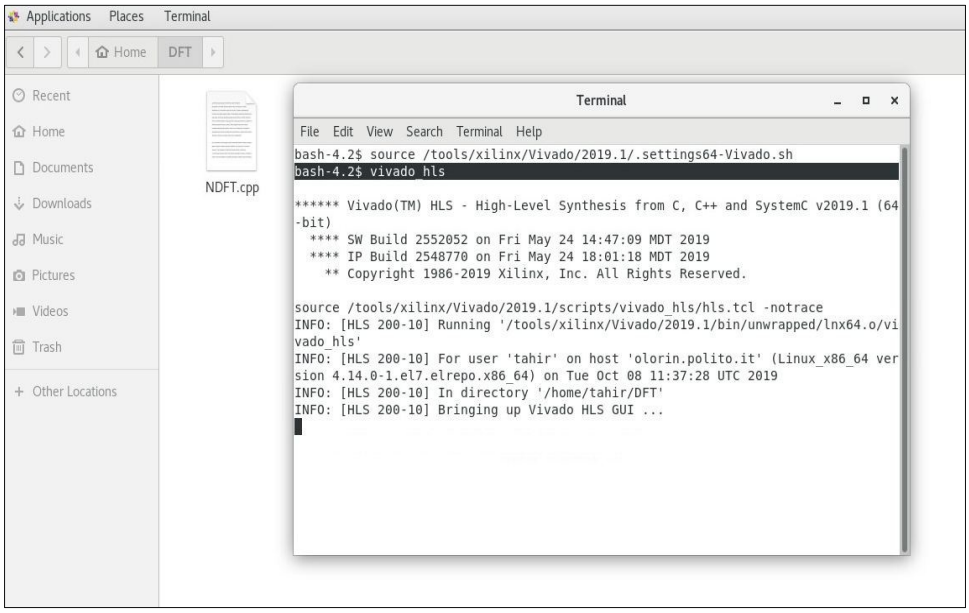


Figure 5-1: **Launching of Vivado HLS**

A new window opens with Vivao HLS Welcome Page that also contains a list of previous recent projects; this window is shown in figure 5-2.

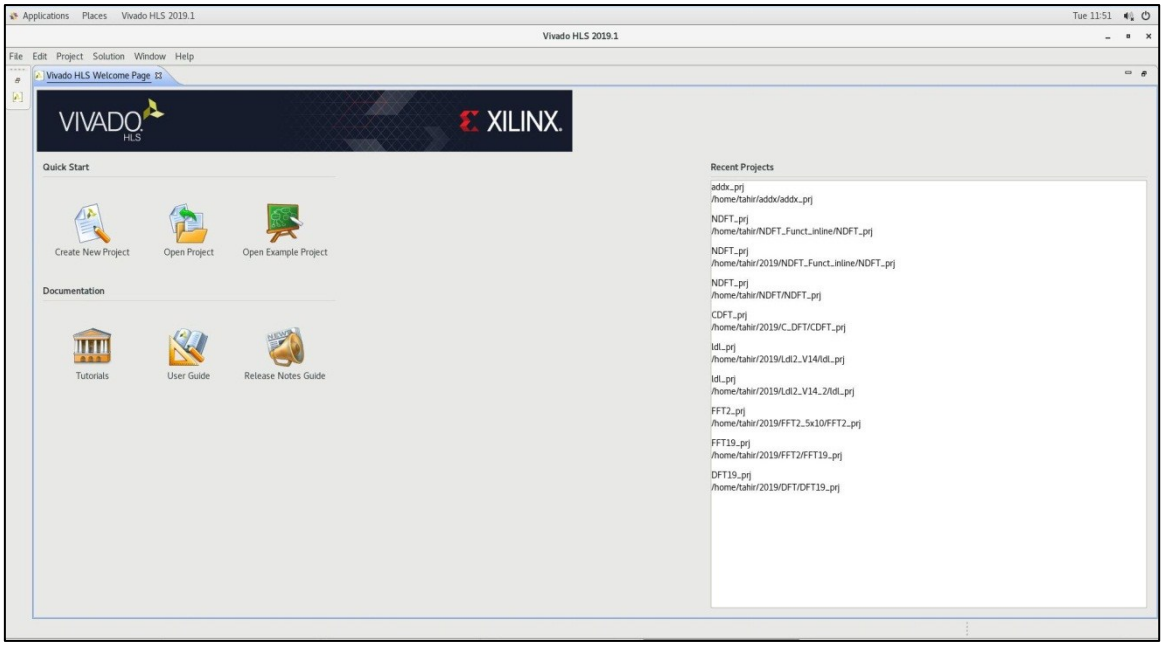


Figure 5-2: **Vivado HLS Welcome Page**

Click the icon “Create New Project”, a new window pops up for project configuration. Write ‘DFT_prj’ in the project name field and set the location same as the location of DFT directory and click next. A new window opens that is used to add files in the design project. Press ‘Add Files’ and the DFT directory opens. Click NDFT.cpp and press OK. If some other directory is opened due to opening terminal in any other directory or due to any previous project that was opened in Vivado HLS, browse to directory DFT and select file NDFT.cpp and press OK. Add/Remove Files window is displayed again with NDFT.cpp included in the Design Files section. Now click Browse button in front of Top Function field and select NDFT(NDFT.cpp) as top function and press OK. Add/Remove Files window contains now NDFT as top function and NDFT.cpp under Design Files. This is shown in figure 5-3. For larger designs having multiple files, all files can be added in the project in this way. The design under consideration contains only one design file. Header files with extension .h are not required to be added as design files here but needed to be placed in the same directory where other design files to be synthesized are located.

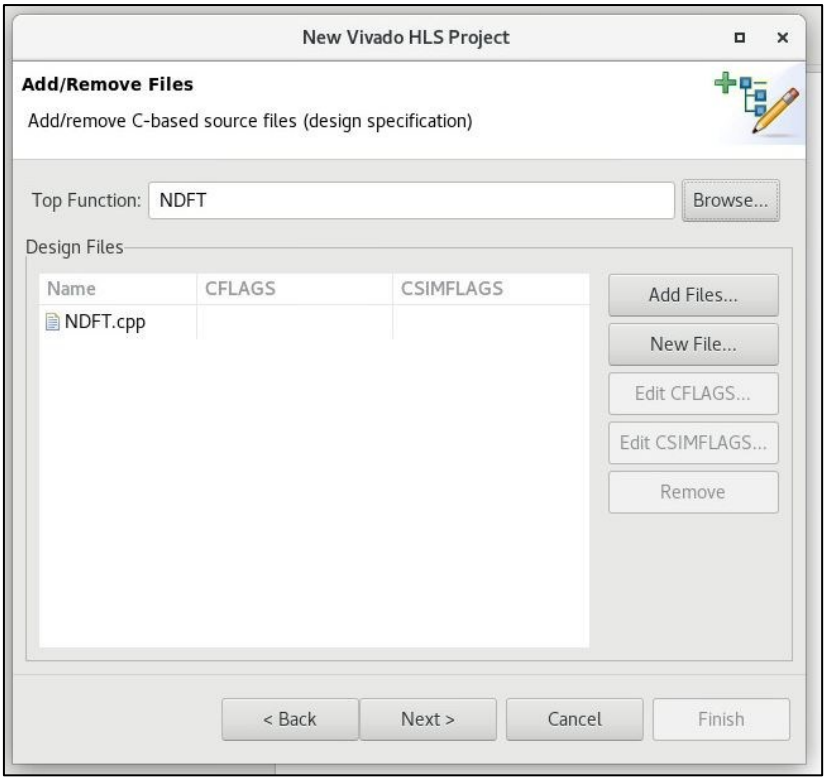


Figure 5-3: **Add/Remove Files Window**

Click next to move to next window that asks for C-based test bench file to be used for design test. For the current case no files needed to be added as the main task of this work is to evaluate the routing congestion of design and its correlation with the HLS code. If the design is to be tested before and after synthesis, than C-based test bench files can be added here. These files are not synthesized in the final design, and are sufficient for testing at high-level code and RTL level code. Click next to move to solution configuration window. Here solution name can be assigned to the design, default is soution1 and for the current case leave it as it is. Next field is the Clock Period, where this parameter is to be inserted for the design, in our case, set it to 5. There is also a field of Clock Uncertainty, leave it blank (default). Then there is the option of part selection, a default device is pre-selected, click button next to it and from the 'Device Selection Dialog' select device xc7z020clg484-1 by scrolling the list or by using the search option for device selection. Click OK and Solution Configuration is complete and as shown in the figure 5-4. Press finish and the HLS project creation is complete.

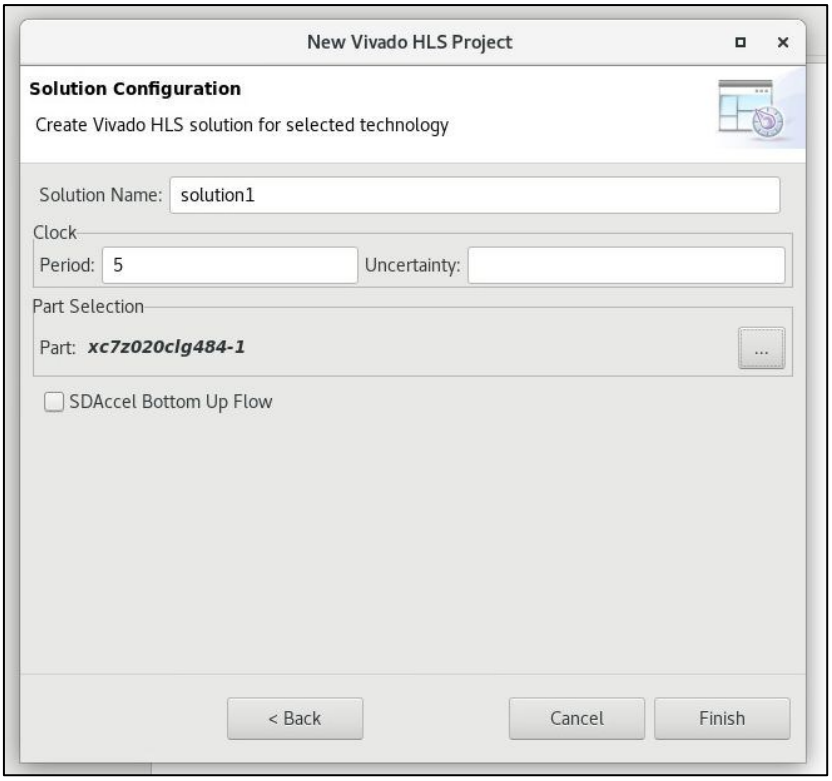


Figure 5-4: **Solution Configuration**

5.4.2 HLS Project Synthesis

After completing the setup for HLS project creation, the project window opens with the project name and its path on top. On the left side of window, there is Explorer section, in that, double click on Source and under it, file NDFT.cpp appears. On the file name double click and the file opens in the middle section for any possible changes. Make sure that on the top right side, Synthesis option is selected among the modes Debug, Synthesis and Analysis. In the outline section on the right of project window, click on the NDFT( ), i.e. on the top level selection function. At this stage C Simulation can be run for the verification of HLS code but this step needs test bench file with a main( ) function and is not applicable to our case. Project window with the stated configuration is shown in figure 5-5.

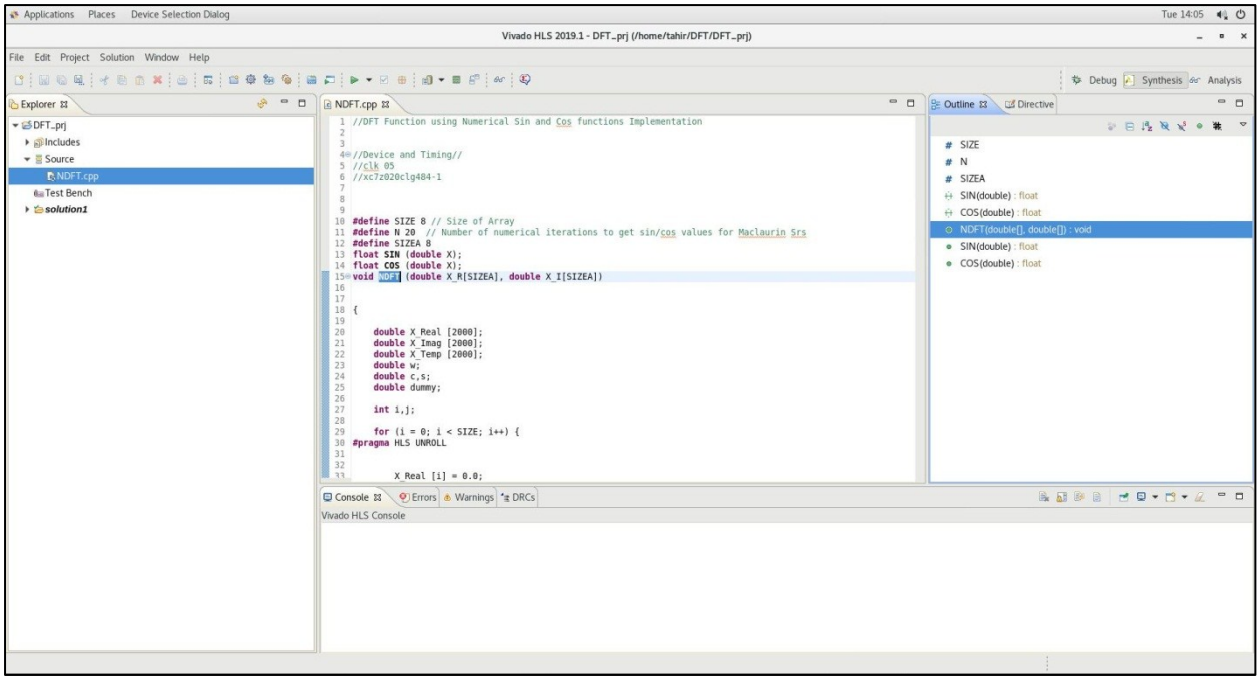


Figure 5-5: **Vivado HLS Project Window**

Click Solution menu, select 'Run C Synthesis' under its options and select 'Active Solution' to start C Synthesis of the design. This can also be run using tool button for 'Run C Synthesis'. C synthesis of HLS code is started and its progress can be shown in the Vivado HLS Console in the lower middle end of the project window. When synthesis is completed, Synthesis Report for 'NDFT' opens in the new window as shown in figure 5-6. If there are errors/warnings in the design, they are also reported next to Vivado HLS Console. This file is also present in the DFT directory in DFT_prj/solution1/syn/report, as NDFT_csynth.rpt. This report is included as Appendix B in this document.

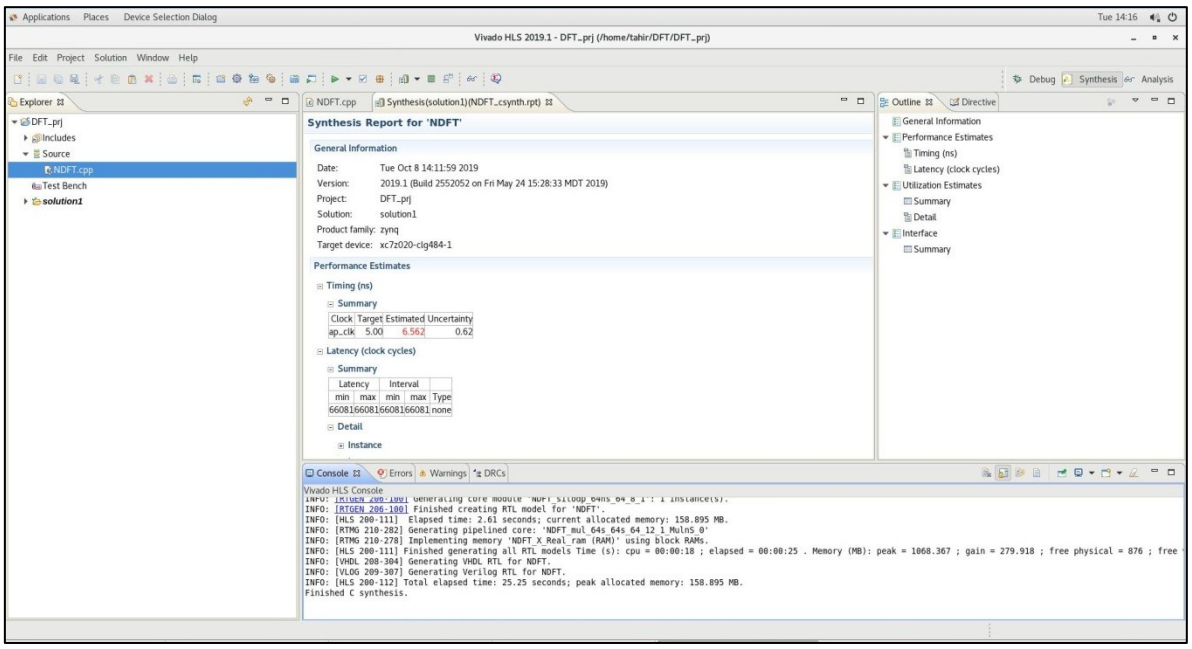


Figure 5-6: **Synthesis Report**

5.4.3 Export RTL for Vivado

Next step in design flow is to export RTL generated by High-Level Synthesis for further processing in Vivado Design Suite. Design could be exported as Xilinx IP or it can be directly opened in Vivado. Here the second approach is adopted and placement and routing of the design is also carried out during this Export RTL step, it can also be done later in Vivado. To start exporting RTL, click menu 'Solution' and press Export RTL, it can also be accomplished by clicking the tool button for Export RTL. Export RTL dialogue opens, select options as shown in figure 5-7 and press OK. Export RTL process is initialized and the progress of flow is displayed in the Vivado HLS Console. As synthesis, place and route are all carried out in this step so this step takes some time to complete. When export is complete, Export Report for 'NDFT' is shown containing post-implementation resource usage and final timing information as shown in figure 5-8. This report can also be found in directory path DFT/DFT_prj/solution1/impl/report/Verilog as NDFT_export.rpt. This is all what is required to be done in Vivaod HLS. At this stage, Vivado HLS can be exited. As described earlier, this complete Vivado HLS flow can also be run in command line mode using a script written in tcl format containing all the commands executed for the high-level flow. Vivado HLS also creates a script.tcl file that can be used to run complete flow as explained above in command line mode. This file can be found in the solution1 directory in the project space.

Figure 5-7: **Export RTL Dialogue**

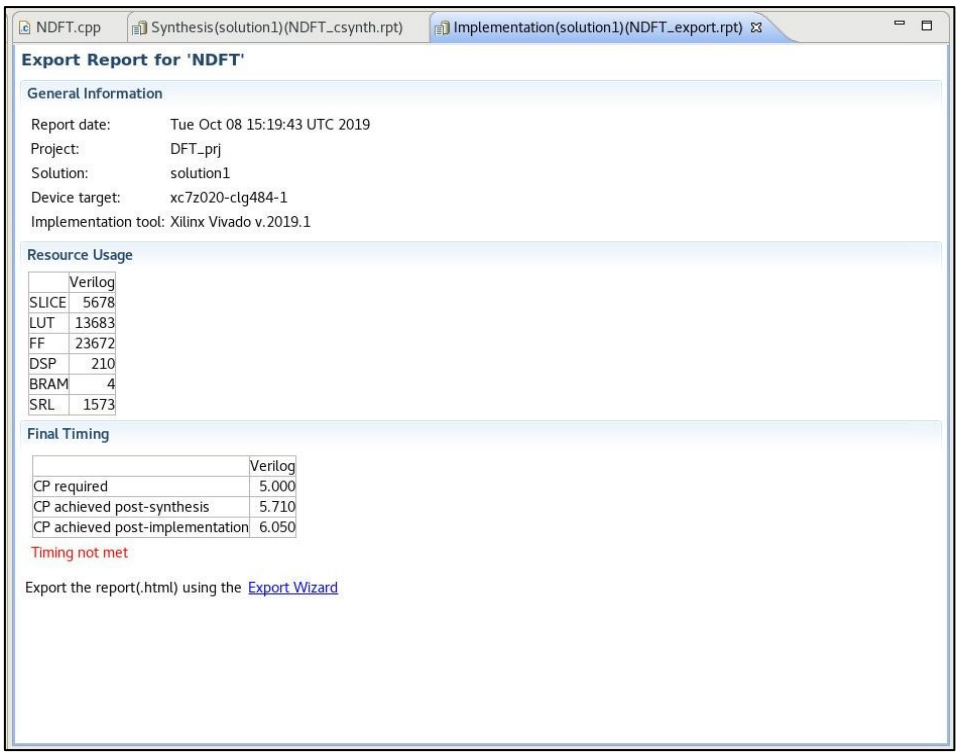


Figure 5-8: **Export Report**

5.5 Vivado HLS Database Files

During High-Level Synthesis flow, Vivado HLS generates many intermediate files. Database or .adb files are also generated during this process. The .adb files contain a control flow graph (CFG) that describes the design at high level, close to the source code. Vivado HLS provides debugging information for every CFG node, representing an LLVM instruction producing a value flowing through the data path, and hence closely associated with RTL nets and registers. The high level source code information in the .adb file is further linked with hard ware resource level information that is present in a .rpt file of the same name. One .adb and .rpt file is created for every distinct function present in the high-level source code.

5.6 Processing of .adb Files

Vivado HLS generates RTL of the design using high-level code. In the RTL design different net names are created corresponding to the high-level code. These net names are linked to high-level code lines and this information is present in .adb files generated by Vivado HLS in a very crude form. These files contain information regarding source code line number, bit width and delay related to each RTL net created during high-level synthesis flow. A hidden directory is created by Vivado HLS named .autopilot that contains a directory named 'db' and it contains files containing information related to net names in the synthesized RTL design and their correlation with the line numbers of NDFT.cpp file. Path of this directory can be traced as DFT/DFT_prj/solution1/.autopilot/db. Sometimes hidden directories are

not accessible in graphical mode, command line mode can be used to access db directory and the concerned .adb files in that directory. A Parser is used to extract the requisite information from these .adb files present in the db directory.

5.7 Parsing .adb Files

A parser has been developed by Christos Sotiriou and Yorgos Floros from EECE Department, University of Thessaly, Greece. This parser, named 'adb_parser' is a customized piece of software that is developed to get net names that Vivado HLS assigns to nets in the generated RTLs during High-Level Synthesis process corresponding to line numbers of high-level code file, in this case NDFT.cpp. It takes .adb files as input, process them for the requisite data and generates some files at the end of the process that are then used for the purpose of obtaining the net names that are responsible for the routing congestion at the end of Vivado design flow. The process to get files corresponding to this parsing process is following.

Create a directory in the project folder and assign it a suitable name, in this a directory named Parser_DFT is created in the DFT directory. Copy 'adb_parser' in this directory. Open a Linux terminal in this directory and run following command.

```
bash-4.2$ ./adb_parser ../DFT_prj/solution1/.autopilot/db/
```

During running this type of commands, set the correct paths for relevant directories/files, as the commands that are reported here are specific to DFT project and the structure of its subdirectories/files. At the end of parsing process, there are many files generated in Parser_DFT directory as shown in figure 5-9.

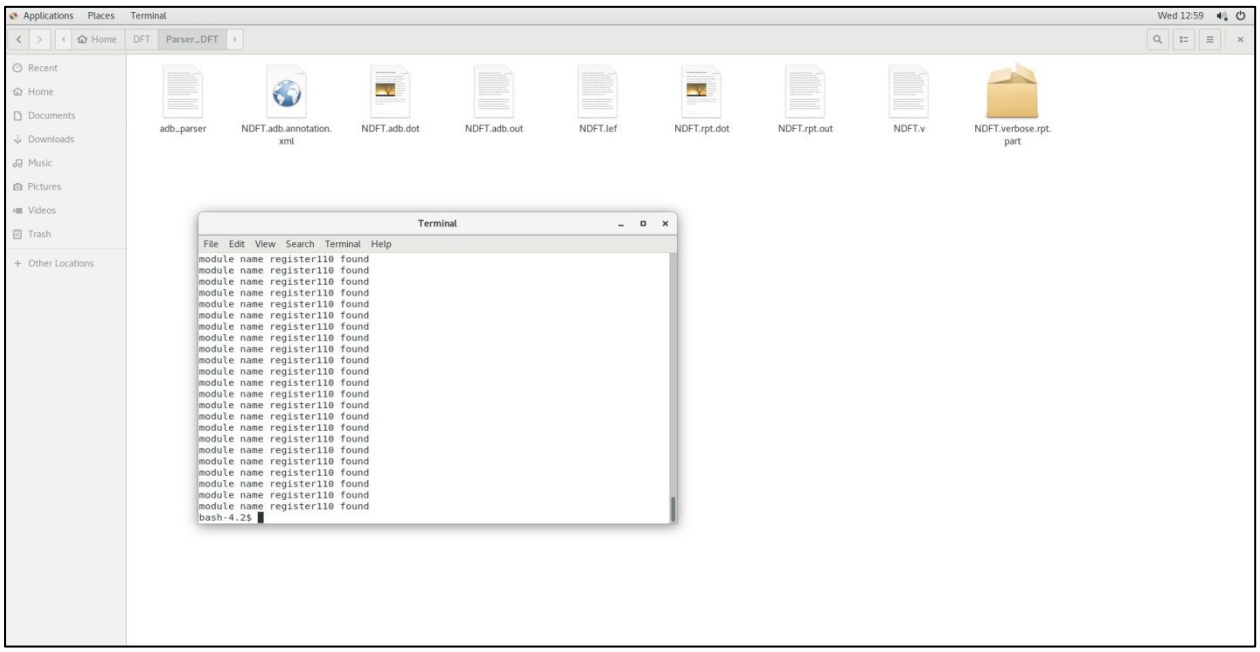
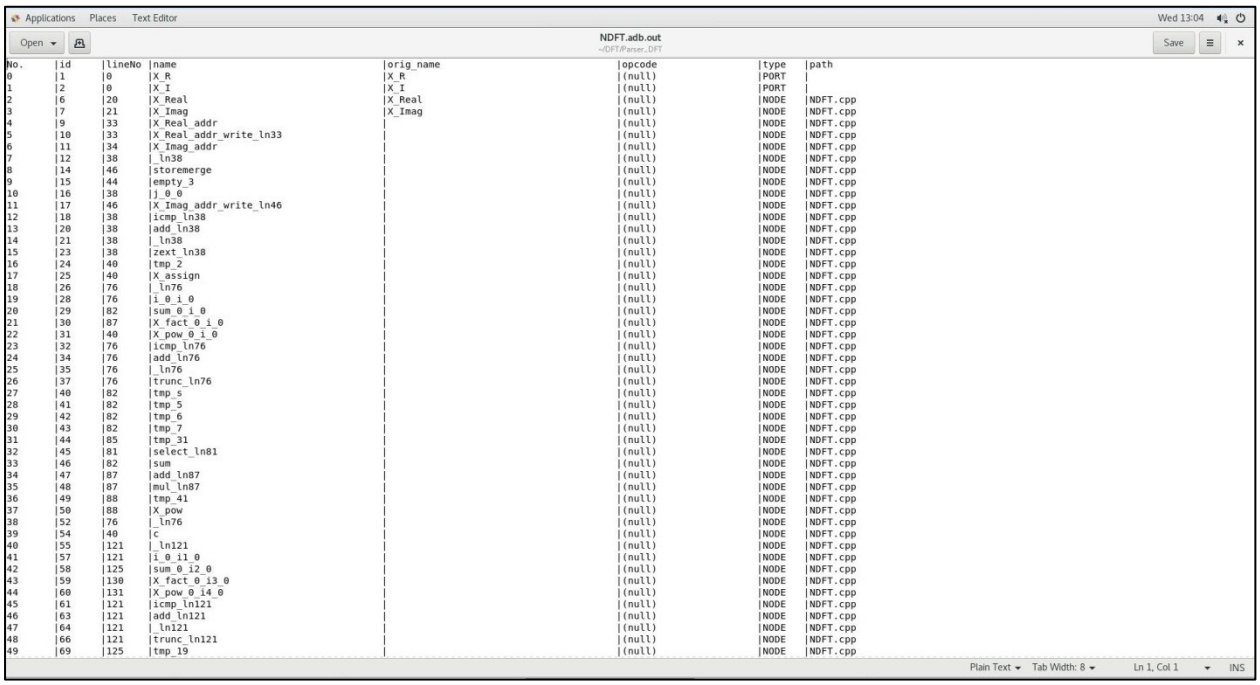


Figure 5-9: **Files generated after Parsing**

Among all these files, NDFT.adb.out is of our interest for the movement. Some part of this file is shown in figure 5-10.



No.	id	lineNo	name	orig_name	opcode	type	path
0	1	0	X_R	X_R	(null)	PORT	
1	2	0	X_I	X_I	(null)	PORT	
2	6	20	X_Real	X_Real	(null)	NODE	NDFT.cpp
3	7	21	X_Imag	X_Imag	(null)	NODE	NDFT.cpp
4	9	33	X_Real_addr		(null)	NODE	NDFT.cpp
5	10	33	X_Real_addr_write_ln33		(null)	NODE	NDFT.cpp
6	11	34	X_Imag_addr		(null)	NODE	NDFT.cpp
7	12	38	_ln38		(null)	NODE	NDFT.cpp
8	14	46	storemerge		(null)	NODE	NDFT.cpp
9	15	44	empty_3		(null)	NODE	NDFT.cpp
10	16	38	j_0_0		(null)	NODE	NDFT.cpp
11	17	46	X_Imag_addr_write_ln46		(null)	NODE	NDFT.cpp
12	18	38	icmp_ln38		(null)	NODE	NDFT.cpp
13	20	38	add_ln38		(null)	NODE	NDFT.cpp
14	21	38	_ln38		(null)	NODE	NDFT.cpp
15	23	38	zext_ln38		(null)	NODE	NDFT.cpp
16	24	40	tmp_2		(null)	NODE	NDFT.cpp
17	25	40	X_assign		(null)	NODE	NDFT.cpp
18	26	76	_ln76		(null)	NODE	NDFT.cpp
19	28	76	i_0_i_0		(null)	NODE	NDFT.cpp
20	29	82	sum_0_i_0		(null)	NODE	NDFT.cpp
21	30	87	X_fact_0_i_0		(null)	NODE	NDFT.cpp
22	31	40	X_pow_0_i_0		(null)	NODE	NDFT.cpp
23	32	76	icmp_ln76		(null)	NODE	NDFT.cpp
24	34	76	add_ln76		(null)	NODE	NDFT.cpp
25	35	76	_ln76		(null)	NODE	NDFT.cpp
26	37	76	trunc_ln76		(null)	NODE	NDFT.cpp
27	40	82	tmp_s		(null)	NODE	NDFT.cpp
28	41	82	tmp_5		(null)	NODE	NDFT.cpp
29	42	82	tmp_6		(null)	NODE	NDFT.cpp
30	43	82	tmp_7		(null)	NODE	NDFT.cpp
31	44	85	tmp_31		(null)	NODE	NDFT.cpp
32	45	81	select_ln81		(null)	NODE	NDFT.cpp
33	46	82	sum		(null)	NODE	NDFT.cpp
34	47	87	add_ln87		(null)	NODE	NDFT.cpp
35	48	87	mul_ln87		(null)	NODE	NDFT.cpp
36	49	88	tmp_41		(null)	NODE	NDFT.cpp
37	50	88	X_pow		(null)	NODE	NDFT.cpp
38	52	76	_ln76		(null)	NODE	NDFT.cpp
39	54	40	c		(null)	NODE	NDFT.cpp
40	55	121	_ln121		(null)	NODE	NDFT.cpp
41	57	121	i_0_i1_0		(null)	NODE	NDFT.cpp
42	58	125	sum_0_i2_0		(null)	NODE	NDFT.cpp
43	59	130	X_fact_0_i3_0		(null)	NODE	NDFT.cpp
44	60	131	X_pow_0_i4_0		(null)	NODE	NDFT.cpp
45	61	121	icmp_ln121		(null)	NODE	NDFT.cpp
46	63	121	add_ln121		(null)	NODE	NDFT.cpp
47	64	121	_ln121		(null)	NODE	NDFT.cpp
48	66	121	trunc_ln121		(null)	NODE	NDFT.cpp
49	69	125	tmp_19		(null)	NODE	NDFT.cpp

Figure 5-10: **NDFT.adb.out**

In this file, column 'lineNo' contains line numbers of NDFT.cpp that are responsible for the creation of RTL nets, reported in the column 'name'. Every net name is part of RTLs that are generated by Vivado HLS during High-Level Synthesis process and are the results of statements of high-level code. At this step it is useful to convert NDFT.adb.out file into a more useful form for further processing. Open this file in some text editor and save it as .txt file. Import this file into some spreadsheet software like MS Excel and keep only two columns with line numbers and corresponding net names. Delete rows with line numbers as 0 or 99999, that are logically incorrect. Save this file as HLS_nets in a CSV format or tab delimited text format. Adb_parser is yet in evolving phase so these changes are to be incorporated manually. Later these modifications can be automated.

5.8 Congestion Reporting in Vivado Design Suite

Next we import the RTLs generated by Vivado HLS in to Vivado Design Suit V.2019 and analyze the design for any possible routing congestion present in the design at the end of design flow. Open terminal in project directory and issue following two commands to start Vivado v2019.1 (64-bit) in GUI mode.

```
bash-4.2$ source /tools/xilinx/Vivado/2019.1/.settings64-Vivado.sh

bash-4.2$ vivado
```

Vivado starts and Quick Start window is displayed. Click Open Project and select project.xpr file present in Verilog directory as shown in figure 5-11 and press OK.

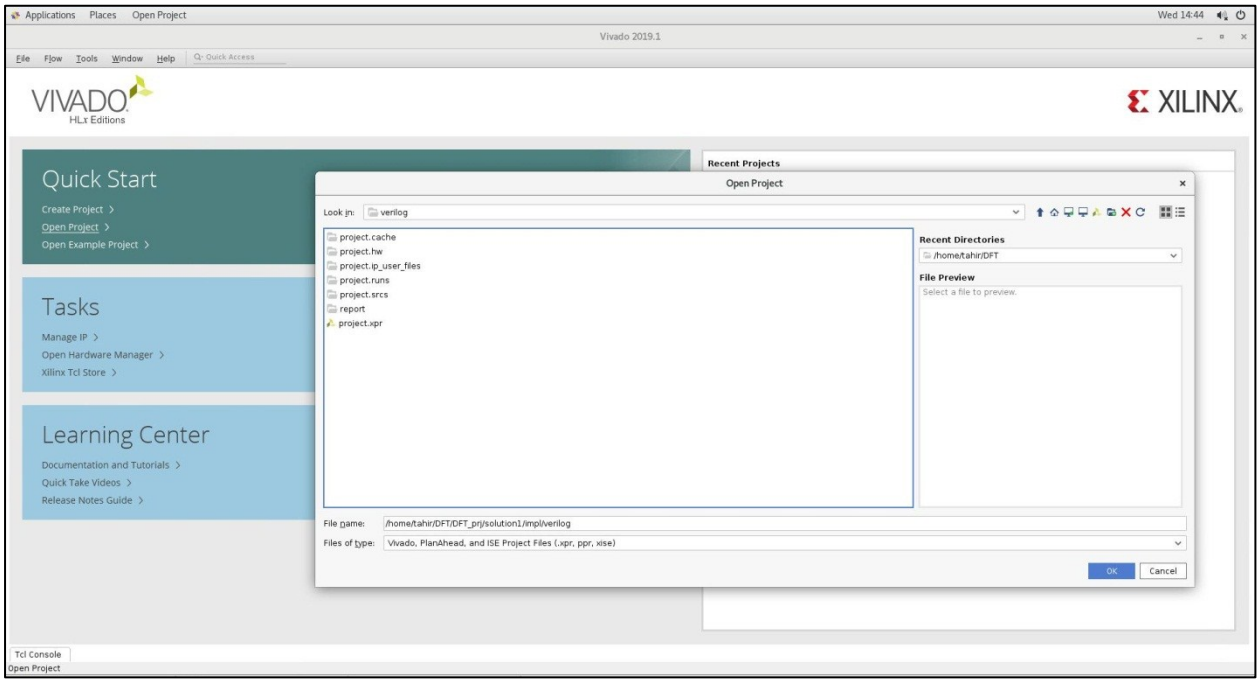


Figure 5-11: **Vivado Open Project Window**

Now project opens in Vivado 2019.1 with Synthesis and Implementation steps already done during export RTL process in Vivado HLS. Click on Open Implemented Design in Flow Navigator pan on left side of project window. Implemented design opens in Device Window next to Project Summary.

Vivado gives two types of congestion information; one describes congestion windows having multiple tiles and the other type states the congested tiles in horizontal and vertical direction. This has been already described in the congestion topic. Both congestion windows information and congested tiles information can be used to further analyze congestion in design and its correlation to high-level code. In this work, the analysis is based on the congested tiles. To view and report congested tiles, right-click on device view, select Metric and click Vertical routing congestion per CLB and Horizontal routing congestion per CLB one by one. Congested tiles are highlighted in the device view in the colors as per the level of congestion and in the lower side of window, Metric Results tab opens, that contains the tile names, their position and vertical and horizontal congestion per CLB in % form, as shown in figure 5-12.

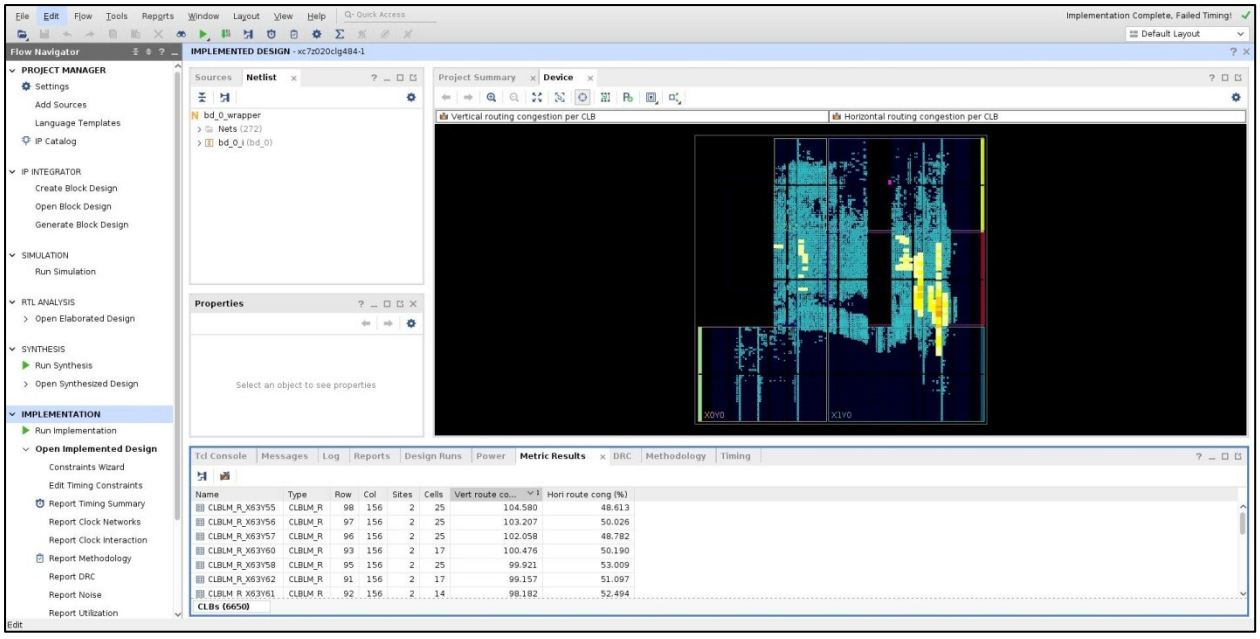


Figure 5-12: **Routing Congestion per CLB**

Metric results with congestion information can be exported to spreadsheet for further processing. Right click on Metric Results view and press Export to Spreadsheet. Give file name and path of project directory, in this case file name is congested_CLBs.xlsx

5.9 Reporting Nets of Congested CLBs

Selection of percentage congestion present in CLBs for further analysis is an optimization problem. In literature, generally congestion above 80% present in CLBs is suggested to be leading to issues in design. For this particular case, CLBs with congestion percentage above 85% are selected for further analysis. So next step, in this analysis is to extract CLBs with congestion above 85% from both horizontal and vertical direction. For this purpose, create a directory named 'Results' in the project space and copy and copy files 'congested_CLBs.xlsx' and 'HLS_nets.csv' in this directory. Open file 'congested_CLBs.xlsx' and copy only names of CLBs with congestion above 85% and write them in a separate file and save it as high_cong_CLB.csv in the Results directory.

We have now CLBs with required level of congestion written in a separate file. Now we need the net name corresponding to these CLBs. For this we need to run Vivado in batch script mode. Write the following small piece of code in a text file and save it as 'clb_to_net.tcl' in Results directory. This file is used to run Vivado in batch script mode, it reads CLBs from 'high_cong_CLB.csv' and writes the all corresponding nets in a newly created file 'Congested_nets.csv', when completed, Vivado exits and the file is printed in Reults directory.

```
open_project /home/tahir/DFT/DFT_prj/solution1/impl/verilog/project.xpr
update_compile_order -fileset sources_1
open_run impl_1
set fp [open /home/tahir/DFT/Results/Congested_nets.csv w]
set fpr [open /home/tahir/DFT/Results/high_cong_CLB.csv r]
fconfigure $fpr -buffering line
gets $fpr data

while {$data != ""} {
        set all_nets [get_nets -of_objects [get_tiles  $data]]
        foreach net $all_nets {puts $fp $net}
        gets $fpr data
        }
close $fp
close $fpr
```

Now open a terminal in the Results directory and issue the following commands to run Vivado and execute clb_to_net.tcl script to get requisite net names. Make necessary changes in the directory names and file paths accordingly.

```
bash-4.2$ source /tools/xilinx/Vivado/2019.1/.settings64-Vivado.sh
bash-4.2$ vivado -mode batch -source clb_to_net.tcl
```

Vivado runs in background and the progress of process is shown on the terminal as shown in figure 5-13.

```
File  Edit  View  Search  Terminal  Help
bash-4.2$ vivado -mode batch -source clb_to_net.tcl

****** Vivado v2019.1 (64-bit)
  **** SW Build 2552052 on Fri May 24 14:47:09 MDT 2019
  **** IP Build 2548770 on Fri May 24 18:01:18 MDT 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

source clb_to_net.tcl
# open_project /home/tahir/DFT/DFT_prj/solution1/impl/verilog/project.xpr
Scanning sources...
Finished scanning sources
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1700] Loaded user IP repository '/home/tahir/DFT/DFT_prj/solution1/impl/ip'.
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/tools/xilinx/Vivado/2019.1/data/ip'.
open_project: Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 1530.613 ; gain = 70.738 ; free physical = 2152 ; free virtual = 117088
# update_compile_order -fileset sources_1
# open_run impl_1
INFO: [Device 21-403] Loading part xc7z020clg484-1
INFO: [Netlist 29-17] Analyzing 1544 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-479] Netlist was created with Vivado 2019.1
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Timing 38-478] Restoring timing data from binary archive.
INFO: [Timing 38-479] Binary timing data restore complete.
INFO: [Project 1-856] Restoring constraints from binary archive.
INFO: [Project 1-853] Binary constraint restore complete.
Reading XDEF placement.
Reading placer database...
Reading XDEF routing.
Read XDEF File: Time (s): cpu = 00:00:02 ; elapsed = 00:00:02 . Memory (MB): peak = 2562.449 ; gain = 34.004 ; free physical = 1050 ; free virtual = 116078
Restored from archive | CPU: 2.190000 secs | Memory: 42.595322 MB |
Finished XDEF File Restore: Time (s): cpu = 00:00:02 ; elapsed = 00:00:02 . Memory (MB): peak = 2562.449 ; gain = 34.004 ; free physical = 1050 ; free virtual = 116078
Netlist sorting complete. Time (s): cpu = 00:00:00.01 ; elapsed = 00:00:00 . Memory (MB): peak = 2562.449 ; gain = 0.000 ; free physical = 1054 ; free virtual = 116083
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 16 instances were transformed.
  SRLC32E => SRL16E: 16 instances

open_run: Time (s): cpu = 00:00:17 ; elapsed = 00:00:22 . Memory (MB): peak = 2562.449 ; gain = 959.805 ; free physical = 1054 ; free virtual = 116083
# set fp [open /home/tahir/DFT/Results/Congested_nets.csv w]
# set fpr [open /home/tahir/DFT/Results/high_cong_CLB.csv r]
# fconfigure $fpr -buffering line
# gets $fpr data
# while {$data != ""} {
#       set all_nets [get_nets -of_objects [get_tiles  $data]]
#       foreach net $all_nets {puts $fp $net}
#       gets $fpr data
#       }
# close $fp
# close $fpr
INFO: [Common 17-206] Exiting Vivado at Thu Oct 10 11:56:52 2019...
bash-4.2$
```

Figure 5-13: **Vivado in Batch Script Mode**

5.10 Correlation between High-Level Code and Congested Nets

At this stage we have obtained net names of RTL generated as a result of high-level synthesis, corresponding to high-level code of file NDFT.cpp along with line numbers in file 'HLS_nts.csv' using adb_parser and net names extracted from congested CLBs from Vivado after physical synthesis, placement and routing is completed and congestion is reported, in file 'Congested_nets.csv' in the results folder. Top results of both these two files are shown in figure 5-14 side by side. Next step is to trace out HLS reported nets in the total congested nets and the corresponding code lines to get the precise information that what lines of code in the high-level code are responsible for the nets in the congested CLBs and make a qualitative analysis about the nets reported after High-Level Synthesis and the ones that belongs to congested CLBs.

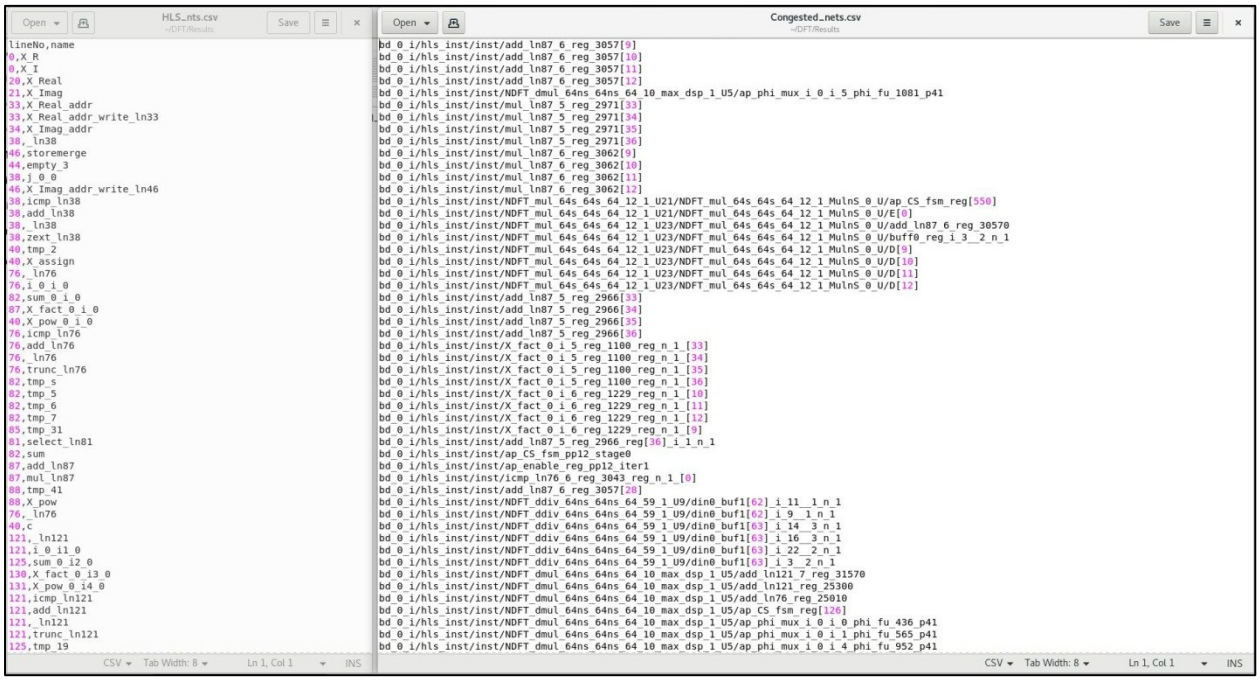


Figure 5-14: **HLS RTL and Congested Nets**

To trace out HLS nets in congested nets, a small script written in python is used. Create a directory named Python and copy files HLS_nts.csv and Congested_nets.csv in it. Write following python code in a file and save it as correlate.py in the 'Python' directory.

```
import csv
#  reading csv to list [['code_lineNo', 'hls_net_name'] e.g. ['0', 'X_R']]
with open('HLS_nts.csv', 'r') as f:
  reader = csv.reader(f)
  net_names = list(reader)

#print HLS_Nets info on console
```

```
print("line_No , hls_net_name : ",net_names[1])
print("total hls_net_names : ",len(net_names))

# reading all congested net names to list
with open('Congested_nets.csv', 'r') as f:
    lines = f.read().splitlines()

#print Congested_Nets info on console
print("congested_net_1: ",lines[0])
print("total congested_net_names : ",len(lines))

#count for each HLS_net, occuring how many times in congested nets
for i in range(len(net_names)):
    word = net_names[i][1]     # get name of single HLS_Net
    count = (sum(word in line for line in lines))
    net_names[i].append(count)

#print all hls_nets and line number with total  no of match in all congested nets
for i in range(len(net_names)):
            print("line No,hls_net,congested_nets count",net_names[i])

#filter out HLS_nets with zero match in congested nets
net_names_filter = []
sum = 0
already_present = False;
for word in net_names:
    if(word[2]!=0):  #   filter for value not 0
        already_present = False  #  check for already existing in net_names_filter array
        for o in net_names_filter:    # if already present in net_names_filter, then add in same
row
            if o[0] == word[0] and o[1] == word[1]:
                o[2] = o[2]+word[2]
                already_present = True
                break
        if not already_present:
            net_names_filter.append(word)
        sum = sum + word[2]
#print and write in file cumulative sum of all HLS_net found in Congested_nets
print("cumulative sum of all hls_nets found in congested_nets : " + str(sum) )
net_names_filter.append(["Total Number of ","HLS_Nets",len(net_names)])    # write in file
net_names_filter.append(["Total Number of ","Congested_Nets",len(lines)])  # write in file
net_names_filter.append(["Total Number of ","Match Found",sum])    # write in file

# saving results in  csv file name Match
#with open("Match.csv", 'w', newline="") as myfile:
with open("Match.csv", 'wb') as myfile:
```

wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
wr.writerows(net_names_filter)

Open terminal in the same directory and run correlate.py using following command:

bash-4.2$ python correlate.py

On the console, HLS net names along with line numbers of HLS code from file NDFT.cpp are reported along with total number of these HLS_net names matched in the file containing list of all congested nets, as shown in figure 5-15.

```
File  Edit  View  Search  Terminal  Help
bash-4.2$ python correlate.py
('line_No , hls_net_name : ', ['0', 'X_R'])
('total hls_net_names : ', 605)
('congested_net_1: ', 'bd_0_i/hls_inst/inst/add_ln87_6_reg_3057[9]')
('total congested_net_names : ', 3230)
('line No,hls_net,congested_nets count', ['lineNo', 'name', 0])
('line No,hls_net,congested_nets count', ['0', 'X_R', 0])
('line No,hls_net,congested_nets count', ['0', 'X_I', 0])
('line No,hls_net,congested_nets count', ['20', 'X_Real', 0])
('line No,hls_net,congested_nets count', ['21', 'X_Imag', 0])
('line No,hls_net,congested_nets count', ['33', 'X_Real_addr', 0])
('line No,hls_net,congested_nets count', ['33', 'X_Real_addr_write_ln33', 0])
('line No,hls_net,congested_nets count', ['34', 'X_Imag_addr', 0])
('line No,hls_net,congested_nets count', ['38', '_ln38', 0])
('line No,hls_net,congested_nets count', ['46', 'storemerge', 0])
('line No,hls_net,congested_nets count', ['44', 'empty_3', 0])
('line No,hls_net,congested_nets count', ['38', 'j_0_0', 0])
('line No,hls_net,congested_nets count', ['46', 'X_Imag_addr_write_ln46', 0])
('line No,hls_net,congested_nets count', ['38', 'icmp_ln38', 0])
('line No,hls_net,congested_nets count', ['38', 'add_ln38', 0])
('line No,hls_net,congested_nets count', ['38', '_ln38', 0])
('line No,hls_net,congested_nets count', ['38', 'zext_ln38', 0])
('line No,hls_net,congested_nets count', ['40', 'tmp_2', 0])
('line No,hls_net,congested_nets count', ['40', 'X_assign', 0])
('line No,hls_net,congested_nets count', ['76', '_ln76', 22])
('line No,hls_net,congested_nets count', ['76', 'i_0_i_0', 6])
('line No,hls_net,congested_nets count', ['82', 'sum_0_i_0', 0])
('line No,hls_net,congested_nets count', ['87', 'X_fact_0_i_0', 20])
('line No,hls_net,congested_nets count', ['40', 'X_pow_0_i_0', 6])
('line No,hls_net,congested_nets count', ['76', 'icmp_ln76', 20])
('line No,hls_net,congested_nets count', ['76', 'add_ln76', 2])
('line No,hls_net,congested_nets count', ['76', '_ln76', 22])
('line No,hls_net,congested_nets count', ['76', 'trunc_ln76', 0])
('line No,hls_net,congested_nets count', ['82', 'tmp_s', 0])
('line No,hls_net,congested_nets count', ['82', 'tmp_5', 0])
('line No,hls_net,congested_nets count', ['82', 'tmp_6', 0])
('line No,hls_net,congested_nets count', ['82', 'tmp_7', 0])
('line No,hls_net,congested_nets count', ['85', 'tmp_31', 0])
('line No,hls_net,congested_nets count', ['81', 'select_ln81', 0])
('line No,hls_net,congested_nets count', ['82', 'sum', 0])
('line No,hls_net,congested_nets count', ['87', 'add_ln87', 77])
('line No,hls_net,congested_nets count', ['87', 'mul_ln87', 132])
('line No,hls_net,congested_nets count', ['88', 'tmp_41', 0])
('line No,hls_net,congested_nets count', ['88', 'X_pow', 79])
('line No,hls_net,congested_nets count', ['76', '_ln76', 22])
('line No,hls_net,congested_nets count', ['40', 'c', 481])
('line No,hls_net,congested_nets count', ['121', '_ln121', 83])
('line No,hls_net,congested_nets count', ['121', 'i_0_i1_0', 15])
('line No,hls_net,congested_nets count', ['125', 'sum_0_i2_0', 0])
('line No,hls_net,congested_nets count', ['130', 'X_fact_0_i3_0', 128])
('line No,hls_net,congested_nets count', ['131', 'X_pow_0_i4_0', 3])
('line No,hls_net,congested_nets count', ['121', 'icmp_ln121', 40])
Python   correlate.py (~/DFT/Python) - gedit   Terminal
```

Figure 5-15: **HLS Nets matched in Congested Nets**

This script also creates a file named Match.csv containing similar results reported in the above figure with the HLS nets not reported in congested nets filtered out. This file Match.csv is written here with each row having one HLS net found in congested nets, along with its line number and number of 'hits' in congested nets. In the final three rows, total number of nets and cumulative hits found are printed.

```
"76","_ln76","528"
"76","i_0_i_0","6"
"87","X_fact_0_i_0","20"
"40","X_pow_0_i_0","6"
"76","icmp_ln76","20"
"76","add_ln76","2"
"87","add_ln87","77"
"87","mul_ln87","132"
"88","X_pow","79"
"40","c","481"
"121","_ln121","1992"
"121","i_0_i1_0","15"
"130","X_fact_0_i3_0","128"
"131","X_pow_0_i4_0","3"
"121","icmp_ln121","40"
"121","add_ln121","43"
"130","add_ln130","144"
"130","mul_ln130","173"
"41","s","3230"
"76","i_0_i_1","1"
"40","X_pow_0_i_1","2"
"131","X_pow_0_i4_1","4"
"121","add_ln121_1","1"
"40","X_pow_0_i_2","6"
"130","X_fact_0_i3_2","1"
"131","X_pow_0_i4_2","8"
"40","X_pow_0_i_3","5"
"130","X_fact_0_i3_3","32"
"131","X_pow_0_i4_3","2"
"130","add_ln130_3","18"
"76","i_0_i_4","13"
"87","X_fact_0_i_4","84"
"40","X_pow_0_i_4","10"
"76","icmp_ln76_4","19"
"87","add_ln87_4","47"
"87","mul_ln87_4","67"
"121","i_0_i1_4","10"
"130","X_fact_0_i3_4","42"
"131","X_pow_0_i4_4","1"
"121","icmp_ln121_4","13"
"121","add_ln121_4","22"
"130","add_ln130_4","9"
"130","mul_ln130_4","24"
"76","i_0_i_5","1"
"87","X_fact_0_i_5","33"
"40","X_pow_0_i_5","3"
"87","add_ln87_5","17"
```

"87","mul_ln87_5","40"
"130","X_fact_0_i3_5","1"
"131","X_pow_0_i4_5","7"
"121","icmp_ln121_5","1"
"87","X_fact_0_i_6","28"
"40","X_pow_0_i_6","9"
"76","icmp_ln76_6","1"
"87","add_ln87_6","13"
"87","mul_ln87_6","4"
"131","X_pow_0_i4_6","3"
"41","s_6","1979"
"40","X_pow_0_i_7","5"
"131","X_pow_0_i4_7","5"
"121","add_ln121_7","1"
"54","b","3230"
"Total Number of ","HLS_Nets","605"
"Total Number of ","Congested_Nets","3230"
"Total Number of ","Match Found","12941"

After this above written python script is modified to get total number of hits found corresponding to each line of code with all HLS_net names written along with the code line number and in the third column cumulative sum of all hits corresponding to the line number and it's all HLS_nets in a newly created file 'Match_cumulated.csv'. Code lines with maximum number hits are reported in ascending order. The content of this file is shown in figure 5-16. Python script file is attached as correlate_cumulated.py in Appendix C.

	A	B	C
1	41	s,s_6	5209
2	54	b	3230
3	121	_ln121,i_0_i1_0,icmp_ln121,add_ln121,add_ln121_1,i_0_i1_4,icmp_ln121_4,add_ln121_4,icmp_ln121_5,add_ln121_7	2138
4	76	_ln76,i_0_i_0,icmp_ln76,add_ln76,i_0_i_1,i_0_i_4,icmp_ln76_4,i_0_i_5,icmp_ln76_6	591
5	130	X_fact_0_i3_0,add_ln130,mul_ln130,X_fact_0_i3_2,X_fact_0_i3_3,add_ln130_3,X_fact_0_i3_4,add_ln130_4,mul_ln130_4,X_fact_0_i3_5	572
6	87	X_fact_0_i_0,add_ln87,mul_ln87,X_fact_0_i_4,add_ln87_4,mul_ln87_4,X_fact_0_i_5,add_ln87_5,mul_ln87_5,X_fact_0_i_6,add_ln87_6,mul_ln87_6	562
7	40	X_pow_0_i_0,c,X_pow_0_i_1,X_pow_0_i_2,X_pow_0_i_3,X_pow_0_i_4,X_pow_0_i_5,X_pow_0_i_6,X_pow_0_i_7	527
8	88	X_pow	79
9	131	X_pow_0_i4_0,X_pow_0_i4_1,X_pow_0_i4_2,X_pow_0_i4_3,X_pow_0_i4_4,X_pow_0_i4_5,X_pow_0_i4_6,X_pow_0_i4_7	33
10			

Figure 5-16: **HLS Nets matched per Line of Code**

Now in place of net names in figure 5-16 we can put the actual lines of code to observe that what high-level line of code is responsible for creating these RTL nets after High-Level Synthesis that are causing congestion in the design. The resultant congestion creating lines from NDFT.cpp are shown in figure 5-17. Care must be taken during all this flow that the high-level design file is not modified after it has been synthesized by Vivado HLS otherwise there could be some difference in the actual congestion responsible lines and the lines reported by the analysis.

Line No	Code Line in NDFT.cpp	Nets Matched
41	s = COS(j * w);	5209
54	for (int b =0; b<SIZE; b++)	3230
121	for (i = 0; i < N; i++)	2138
76	for (i = 0; i < N; i++)	591
130	X_fact = X_fact * (X_fact+1);	572
87	X_fact = X_fact * (X_fact+1);	562
40	s = SIN(j * w);	527
88	X_pow = X_pow * X * X;	79
131	X_pow = X_pow * X * X;	33

Figure 5-17: **HLS Lines of Code with Congested Nets**

5.11 Changes in High-Level Design

Now the design is modified based on the results reported in figure 5-17. The usefulness of these changes would be described in the next conclusive section. The design file NDFT.cpp is saved as MDFT.cpp in a new directory, where letter 'M' is used to indicate that it is a modified version of original file. From figure 5-17 we observe that line 40 and line 41 are sources of congestion. These lines are:

```
40.    s = SIN(j * w);
41.    c = COS(j * w);
```

Here SIN and COS functions are called and pragma HLS INLINE are used that creates a new instance for each call to these functions in the hardware, so this pragma is commented in the MDFT.cpp file.

Next we observe that line 76 and 121 contributes to congestion in the design. These lines are:

```
76.    for (i = 0; i < N; i++) {
121.   for (i = 0; i < N; i++) {
```

These two lines are for loops and in the top of for loop body pragma HLS PIPELINE is used to pipeline the design corresponding to these loops to get higher level of parallelism, we also comment this pragma for both the for loops.

Next code lines 87, 88, 130 and 131 are mathematical statements that are synthesized to a design containing adder and multipliers implemented using DSPs. To retain the functionality

of the design, we are not making any changes in these lines. Line 54 is again a ‘for’ loop statement with ‘b’ as loop variable and this ‘b’ is a part of lot of net names that are presented in the congested nets, as it is not itself a physical RTL net so we can ignore it.

After making all these changes in the MDFT.cpp, we save it in a new directory ‘MDFT’ and apply the complete design flow as reported in the previous chapter to obtain the modified implement design on the same FPGA device, report congestion CLBs and windows and compare the congestion profile of both the original and modified design.

6 Conclusions and Recommendations

As reported in figure 5-17, this work is very useful for designer in case of High-Level design which results in a congested implementation. At the final stage of design flow, when significant routing congestion is found in the design, the procedure described in chapter 5 can be consulted to trace out the exact statements in the High-Level code that are responsible for the congestion issue in the implemented design. Then this information can be used to modify the High-Level code and other changes like use of high resources device for design, can be incorporated in the design flow to reduce the routing congestion up to an acceptable level.

To prove the useful of this technique, the example design is modified based on the correlation between high-level code of NDFT.cpp and routing congestion reported for the original basic design. First the complete design flow is repeated after making necessary changes in the High-Level code based on modified statements responsible for congestion as done in the last section of previous chapter and then here a comparison is made between the congestion profile of original design and modified design.

6.1 Comparison of Routing Congestion

After implementing modified design of DFT we analyze metric results with congestion information in vertical and horizontal direction as described in section 5-6. Comparison for the congestion in CLBs for the case of original and modified design is reported in table 6-1.

Congestion Parameter	Original Design	Modified Design
CLBs with Horizontal Routing Congestion above 85%	1	0
CLBs with Vertical Routing Congestion above 85%	37	0
Maximum Horizontal Routing Congestion per CLB	85.045%	67.987
Maximum Vertical Routing Congestion per CLB	104.580%	73.265%

Table 6-1: **Comparison of Routing Congestion per CLB**

Device views with heat maps corresponding to routing congestion per CLBs are shown in figure 6-1 and 6-2 for the original design implementation and for its modified version respectively.

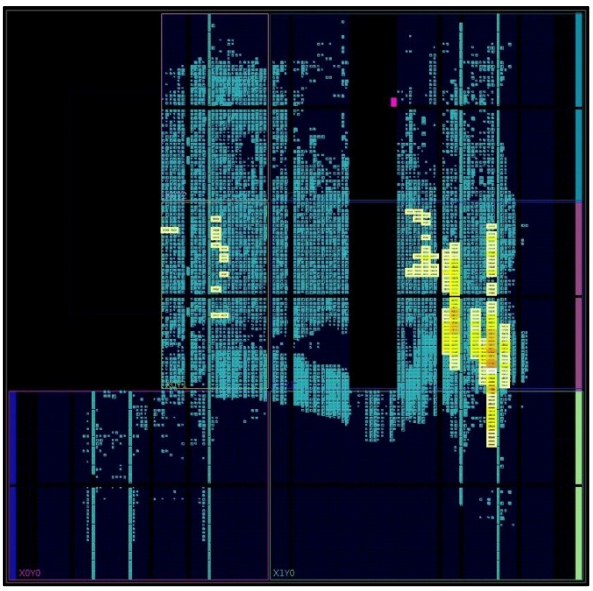


Figure 6-1: **Device view of Original Design**

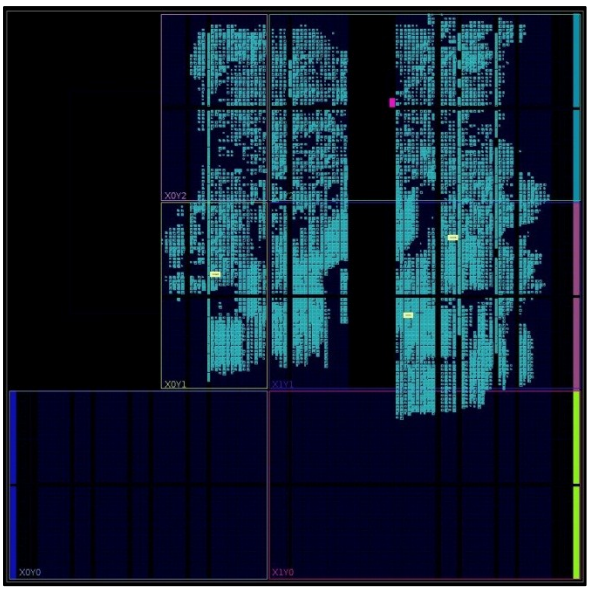
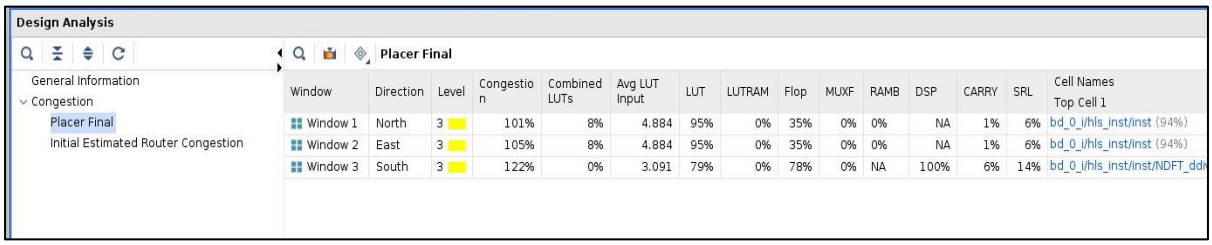


Figure 6-2: **Device view of Modified Design**

Next we analyze the Congestion Report generated by Vivado v.2019.1 as a part of 'Report Design Analysis' feature. Issue following command, in the Vivado Tcl Console to print congestion report in a text file in project directory.

```
report_design_analysis -congestion -min_congestion_level 3 -file
/home/tahir/DFT/Congestion_Report.txt
```

Repeat the above command for the case of congestion report for the modified design of DFT. Congestion report for the original design contains three congestion windows of level-3 in 'Placer Final Level Congestion Reporting' and two congestion windows of level 3 and 4 in 'Initial Estimated Router Congestion Reporting'. Placer Final Level Congestion Reporting and Initial Estimated Router Congestion Reporting by Vivado are shown in figures 6-3 and 6-4 respectively.



Design Analysis

General Information
Congestion
Placer Final
Initial Estimated Router Congestion

Placer Final

Window	Direction	Level	Congestion	Combined LUTs	Avg LUT Input	LUT	LUTRAM	Flop	MUXF	RAMB	DSP	CARRY	SRL	Cell Names Top Cell 1
Window 1	North	3	101%	8%	4.884	95%	0%	35%	0%	0%	NA	1%	6%	bd_0_i/hls_inst/inst (94%)
Window 2	East	3	105%	8%	4.884	95%	0%	35%	0%	0%	NA	1%	6%	bd_0_i/hls_inst/inst (94%)
Window 3	South	3	122%	0%	3.091	79%	0%	78%	0%	NA	100%	6%	14%	bd_0_i/hls_inst/inst/NDFT_ddr

Figure 6-3: **Place Final Congestion**

Design Analysis

Initial Estimated Router Congestion

General Information
Congestion
Placer Final
Initial Estimated Router Congestion

Window	Direction	Type	Level	Percentage Tiles	Combined LUTs	Avg LUT Input	LUT	LUTRAM	Flop	MUXF	RAMB	DSP	CARRY	SRL	Cell Names Top Cell 1	Top Cell 2	Top Cell 3
Window 1	East	Global	3	0.514%	11%	4.896	95%	0%	38%	0%	0%	NA	1%	9%	bd_0_i/hls_inst/inst (92%)	bd_0_i/hls_inst/inst/NDFT_ddiv_64ns_64ns_64_59_1_U9 (7%)	
Window 2	North	Short	4	5.027%	14%	4.05	80%	0%	47%	0%	0%	100%	1%	15%	bd_0_i/hls_inst/inst (77%)	bd_0_i/hls_inst/inst/NDFT_ddiv_64ns_64ns_64_59_1_U9 (10%)	bd_0_i/hls_inst/i

Figure 6-4: **Initial Estimated Router Congestion**

In case of modified design, congestion report states that “No effective congestion windows are found above level 3” as shown in figure 6-5. This result can also be anticipated from the routing congestion reporting per CLBs.

Tcl Console | Messages | Log | Reports | Design Runs | Power | Metric Results | DRC | Design Analysis | Methodology | Timing

Router Initial Congestion

General Information
Congestion
Router Initial Congestion

No effective congestion windows are found above level 3

Figure 6-5: **Congestion Report for Modified Design**

Above comparison suggests the effectiveness of technique for routing congestion tracing in High-Level Synthesis flow devised in this research work in a quantitative manner.

6.2 Recommendations for Future Work

While High-Level Synthesis has become the preferred choice for digital designs, there is a lake of availability of information that can correlate the issues that arise at the end of design flow and high-level code, routing congestion is one of these issues that can in severe cases adversely affects the design flow. The research work presented above can be further improved to include designs with multiple synthesizable high-level code files. This technique can be automated in a way that all the suggested tools are applied to the congested design in a sequence with one single command and the results are reported to the designer at the end of congestion analysis in a form similar to figure 5-17 describing lines of high-level code responsible for congestion. For extracting congested nets from the implemented design, nets corresponding to all tiles present in the congested windows can also be considered along with the nets of CLBs with congestion above a certain level. A further step towards the automation of above suggested routing congestion tracing technique could be that at the final step, some processing be carried on the high-level code based on the information related to the lines of code responsible for the congestion in the final stages to minimize this congestion. Some suggestions could also be reported to the designer at the end of congestion tracing analysis on how to improve the high-level design for lowering routing congestion in the implemented design at the final stages of design flow.

APPENDIX A

NDFT.cpp

```
//DFT Function Implementation using Numerical Sin and Cosine Functions

//Device and Timing//
//clk 05
//xc7z020clg484-1

#define SIZE 8 // Size of Array
#define N 20  // Number of numerical iterations to get sin/cos values for Maclaurin Srs
#define SIZEA 8
float SIN (double X);
float COS (double X);
void NDFT (double X_R[SIZEA], double X_I[SIZEA])
{
	double X_Real [2000];
	double X_Imag [2000];
	double X_Temp [2000];
	double w;
	double c,s;
	double dummy;
	int i,j;
	for (i = 0; i < SIZE; i++) {
#pragma HLS UNROLL

		X_Real [i] = 0.0;
		X_Imag [i] = 0.0;
		w = -(((2.0 * 3.141592653589) / SIZE) * (i));
		for (j=0; j< SIZE; j++) {

			s = SIN(j * w);
			c = COS(j * w);
			X_Real[i] += (X_R[j] * c - X_I[j] * s);
			X_Imag[i] += (X_R[j] * s + X_I[j] * c);
			dummy = c * s;
			X_Temp[i] = (dummy * i * j);
		}
	}
	for (int b =0; b<SIZE; b++) {

		X_R[b] =  X_Real[b];
		X_I[b] =  X_Imag[b];
	}
```

```
}

float SIN (double X)

{
#pragma HLS INLINE
        double X_pow;
        long X_fact;
        float sum;
        X_pow  = X;
        X_fact = 1;
        sum    = 0;
        int i;
        int count = 0;
        int dummy;
        for (i = 0; i < N; i++) {
#pragma HLS PIPELINE

                if(i%2==0)
                        sum += X_pow/X_fact;

                else
                        sum -= X_pow/X_fact;

                X_fact = X_fact * (X_fact+1);
                X_pow  = X_pow  * X * X;
                count = count+i*2;
                dummy = count-i/2;
                if(i%2==0)
                        count = count+i*2;

                else
                dummy = count-i/2;

        }
return sum;
}

float COS (double X)
{
#pragma HLS INLINE
        double X_pow;
        long X_fact;
        float sum;
        X_pow  = 1;
        X_fact = 1;
        sum    = 0;
        int i;
```

```
        int count = 0;
        int dummy ;
        for (i = 0; i < N; i++) {
#pragma HLS PIPELINE
                if(i%2==0)
                        sum -= X_pow/X_fact;

                else
                sum += X_pow/X_fact;
                X_fact = X_fact * (X_fact+1);
                X_pow  = X_pow  * X * X;
                count = count+i/2;
                dummy = count-i*2;
                if(i%2==0)
                        count = count+i/2;

                else
                        dummy = count-i*2;


        }
return sum;

}
```

APPENDIX B

NDFT_csynth.rpt

```
================================================================
== Vivado HLS Report for 'NDFT'
================================================================
* Date:           Tue Oct  8 14:11:58 2019

* Version:        2019.1 (Build 2552052 on Fri May 24 15:28:33 MDT 2019)
* Project:        DFT_prj
* Solution:       solution1
* Product family: zynq
* Target device:  xc7z020-clg484-1


================================================================
== Performance Estimates
================================================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   5.00|     6.562|        0.62|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-------+-------+-------+-------+---------+
    |    Latency    |    Interval   | Pipeline|
    |  min  |  max  |  min  |  max  |   Type  |
    +-------+-------+-------+-------+---------+
    |  66081|  66081|  66081|  66081|   none  |
    +-------+-------+-------+-------+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        +-------------+------+------+----------+-----------+-----------+------+----------+
        |             |   Latency   | Iteration|  Initiation Interval  | Trip |          |
        |  Loop Name  |  min |  max |  Latency |  achieved |   target  | Count| Pipelined|
        +-------------+------+------+----------+-----------+-----------+------+----------+
        |- Loop 1     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 1.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 1.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 2     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 2.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 2.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 3     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 3.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 3.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 4     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 4.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 4.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 5     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 5.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 5.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 6     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 6.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 6.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 7     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 7.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 7.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 8     |  8256|  8256|      1032|          -|          -|     8|    no    |
        | + Loop 8.1  |   483|   483|        85|         21|          1|    20|    yes   |
        | + Loop 8.2  |   483|   483|        85|         21|          1|    20|    yes   |
        |- Loop 9     |    24|    24|         3|          -|          -|     8|    no    |
        +-------------+------+------+----------+-----------+-----------+------+----------+


================================================================
== Utilization Estimates
```

```
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|      -|       0|   2824|    -|
|FIFO             |        -|      -|       -|      -|    -|
|Instance         |        -|    210|   25607|  11687|    -|
|Memory           |       16|      -|       0|      0|    0|
|Multiplexer      |        -|      -|       -|   4599|    -|
|Register         |        0|      -|   11120|   1024|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |       16|    210|   36727|  20134|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |        5|     95|      34|     37|    0|
+-----------------+---------+-------+--------+-------+-----+

+ Detail:
    * Instance:
    +-----------------------------------------------+-----------------------------------------+------
---+-------+------+------+-----+
    |                  Instance                   |                  Module                 |
BRAM_18K| DSP48E|  FF  |  LUT | URAM|
    +-----------------------------------------------+-----------------------------------------+------
---+-------+------+------+-----+
    |NDFT_dadd_64ns_64ns_64_14_full_dsp_1_U4      |NDFT_dadd_64ns_64ns_64_14_full_dsp_1      |
0|      3|  1047|  1102|    0|
    |NDFT_dadddsub_64ns_64ns_64_14_full_dsp_1_U3  |NDFT_dadddsub_64ns_64ns_64_14_full_dsp_1  |
0|      3|  1047|  1102|    0|
    |NDFT_ddiv_64ns_64ns_64_59_1_U9               |NDFT_ddiv_64ns_64ns_64_59_1               |
0|      0|  6160|  3871|    0|
    |NDFT_dmul_64ns_64ns_64_10_max_dsp_1_U5       |NDFT_dmul_64ns_64ns_64_10_max_dsp_1       |
0|     11|   456|   603|    0|
    |NDFT_dmul_64ns_64ns_64_10_max_dsp_1_U6       |NDFT_dmul_64ns_64ns_64_10_max_dsp_1       |
0|     11|   456|   603|    0|
    |NDFT_dmul_64ns_64ns_64_10_max_dsp_1_U7       |NDFT_dmul_64ns_64ns_64_10_max_dsp_1       |
0|     11|   456|   603|    0|
    |NDFT_dmul_64ns_64ns_64_10_max_dsp_1_U8       |NDFT_dmul_64ns_64ns_64_10_max_dsp_1       |
0|     11|   456|   603|    0|
    |NDFT_fpext_32ns_64_3_1_U2                    |NDFT_fpext_32ns_64_3_1                    |
0|      0|   100|   138|    0|
    |NDFT_fptrunc_64ns_32_3_1_U1                  |NDFT_fptrunc_64ns_32_3_1                  |
0|      0|   128|   277|    0|
    |NDFT_mul_64s_64s_64_12_1_U11                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U12                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U13                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U14                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U15                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U16                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U17                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U18                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U19                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U20                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U21                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U22                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U23                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U24                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U25                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_mul_64s_64s_64_12_1_U26                 |NDFT_mul_64s_64s_64_12_1                  |
0|     10|   922|   135|    0|
    |NDFT_sitodp_64ns_64_8_1_U10                  |NDFT_sitodp_64ns_64_8_1                   |
0|      0|   549|   625|    0|
    +-----------------------------------------------+-----------------------------------------+------
---+-------+------+------+-----+
```

```
    |Total                                          |                                                        |
0|    210| 25607| 11687|    0|
    +-----------------------------------------------+----------------------------------------------+------
---+-------+------+------+-----+

    * DSP48E:
    N/A

    * Memory:
    +----------+-------------+---------+---+----+-----+------+-----+------+-------------+
    |  Memory  |    Module   | BRAM_18K| FF| LUT| URAM| Words| Bits| Banks| W*Bits*Banks|
    +----------+-------------+---------+---+----+-----+------+-----+------+-------------+
    |X_Real_U  |NDFT_X_Real  |        8|  0|   0|    0|  2000|   64|     1|       128000|
    |X_Imag_U  |NDFT_X_Real  |        8|  0|   0|    0|  2000|   64|     1|       128000|
    +----------+-------------+---------+---+----+-----+------+-----+------+-------------+
    |Total     |             |       16|  0|   0|    0|  4000|  128|     2|       256000|
    +----------+-------------+---------+---+----+-----+------+-----+------+-------------+

    * FIFO:
    N/A

    * Expression:
    +---------------------------+----------+-------+---+----+------------+------------+
    |       Variable Name       | Operation| DSP48E| FF| LUT| Bitwidth P0| Bitwidth P1|
    +---------------------------+----------+-------+---+----+------------+------------+
    |add_ln121_1_fu_1866_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_2_fu_1957_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_3_fu_2048_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_4_fu_2139_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_5_fu_2230_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_6_fu_2321_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_7_fu_2412_p2     |     +    |      0|  0|  15|           5|           1|
    |add_ln121_fu_1775_p2       |     +    |      0|  0|  15|           5|           1|
    |add_ln130_1_fu_1876_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_2_fu_1967_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_3_fu_2058_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_4_fu_2149_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_5_fu_2240_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_6_fu_2331_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_7_fu_2422_p2     |     +    |      0|  0|  71|           1|          64|
    |add_ln130_fu_1785_p2       |     +    |      0|  0|  71|           1|          64|
    |add_ln38_1_fu_1815_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_2_fu_1906_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_3_fu_1997_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_4_fu_2088_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_5_fu_2179_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_6_fu_2270_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_7_fu_2361_p2      |     +    |      0|  0|  13|           4|           1|
    |add_ln38_fu_1724_p2        |     +    |      0|  0|  13|           4|           1|
    |add_ln76_1_fu_1832_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_2_fu_1923_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_3_fu_2014_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_4_fu_2105_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_5_fu_2196_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_6_fu_2287_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_7_fu_2378_p2      |     +    |      0|  0|  15|           5|           1|
    |add_ln76_fu_1741_p2        |     +    |      0|  0|  15|           5|           1|
    |add_ln87_1_fu_1842_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_2_fu_1933_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_3_fu_2024_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_4_fu_2115_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_5_fu_2206_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_6_fu_2297_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_7_fu_2388_p2      |     +    |      0|  0|  71|           1|          64|
    |add_ln87_fu_1751_p2        |     +    |      0|  0|  71|           1|          64|
    |b_fu_2452_p2               |     +    |      0|  0|  13|           4|           1|
    |icmp_ln121_1_fu_1860_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_2_fu_1951_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_3_fu_2042_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_4_fu_2133_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_5_fu_2224_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_6_fu_2315_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_7_fu_2406_p2    |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln121_fu_1769_p2      |   icmp   |      0|  0|  11|           5|           5|
    |icmp_ln38_1_fu_1809_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_2_fu_1900_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_3_fu_1991_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_4_fu_2082_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_5_fu_2173_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_6_fu_2264_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_7_fu_2355_p2     |   icmp   |      0|  0|  11|           4|           5|
    |icmp_ln38_fu_1718_p2       |   icmp   |      0|  0|  11|           4|           5|
```

icmp_ln54_fu_2446_p2	icmp	0	0	11	4	5
icmp_ln76_1_fu_1826_p2	icmp	0	0	11	5	5
icmp_ln76_2_fu_1917_p2	icmp	0	0	11	5	5
icmp_ln76_3_fu_2008_p2	icmp	0	0	11	5	5
icmp_ln76_4_fu_2099_p2	icmp	0	0	11	5	5
icmp_ln76_5_fu_2190_p2	icmp	0	0	11	5	5
icmp_ln76_6_fu_2281_p2	icmp	0	0	11	5	5
icmp_ln76_7_fu_2372_p2	icmp	0	0	11	5	5
icmp_ln76_fu_1735_p2	icmp	0	0	11	5	5
select_ln124_1_fu_1887_p3	select	0	0	64	1	64
select_ln124_2_fu_1978_p3	select	0	0	64	1	64
select_ln124_3_fu_2069_p3	select	0	0	64	1	64
select_ln124_4_fu_2160_p3	select	0	0	64	1	64
select_ln124_5_fu_2251_p3	select	0	0	64	1	64
select_ln124_6_fu_2342_p3	select	0	0	64	1	64
select_ln124_7_fu_2433_p3	select	0	0	64	1	64
select_ln124_fu_1796_p3	select	0	0	64	1	64
select_ln81_1_fu_1853_p3	select	0	0	64	1	64
select_ln81_2_fu_1944_p3	select	0	0	64	1	64
select_ln81_3_fu_2035_p3	select	0	0	64	1	64
select_ln81_4_fu_2126_p3	select	0	0	64	1	64
select_ln81_5_fu_2217_p3	select	0	0	64	1	64
select_ln81_6_fu_2308_p3	select	0	0	64	1	64
select_ln81_7_fu_2399_p3	select	0	0	64	1	64
select_ln81_fu_1762_p3	select	0	0	64	1	64
ap_enable_pp0	xor	0	0	2	1	2
ap_enable_pp1	xor	0	0	2	1	2
ap_enable_pp10	xor	0	0	2	1	2
ap_enable_pp11	xor	0	0	2	1	2
ap_enable_pp12	xor	0	0	2	1	2
ap_enable_pp13	xor	0	0	2	1	2
ap_enable_pp14	xor	0	0	2	1	2
ap_enable_pp15	xor	0	0	2	1	2
ap_enable_pp2	xor	0	0	2	1	2
ap_enable_pp3	xor	0	0	2	1	2
ap_enable_pp4	xor	0	0	2	1	2
ap_enable_pp5	xor	0	0	2	1	2
ap_enable_pp6	xor	0	0	2	1	2
ap_enable_pp7	xor	0	0	2	1	2
ap_enable_pp8	xor	0	0	2	1	2
ap_enable_pp9	xor	0	0	2	1	2
Total		0	0	2824	280	2230

* Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
X_I_address0	47	10	3	30
X_Imag_address0	47	10	11	110
X_Imag_d0	44	9	64	576
X_R_address0	47	10	3	30
X_Real_address0	89	18	11	198
X_Real_d0	21	4	64	256
X_fact_0_i3_0_reg_500	9	2	64	128
X_fact_0_i3_1_reg_629	9	2	64	128
X_fact_0_i3_2_reg_758	9	2	64	128
X_fact_0_i3_3_reg_887	9	2	64	128
X_fact_0_i3_4_reg_1016	9	2	64	128
X_fact_0_i3_5_reg_1145	9	2	64	128
X_fact_0_i3_6_reg_1274	9	2	64	128
X_fact_0_i3_7_reg_1403	9	2	64	128
X_fact_0_i_0_reg_455	9	2	64	128
X_fact_0_i_1_reg_584	9	2	64	128
X_fact_0_i_2_reg_713	9	2	64	128
X_fact_0_i_3_reg_842	9	2	64	128
X_fact_0_i_4_reg_971	9	2	64	128
X_fact_0_i_5_reg_1100	9	2	64	128
X_fact_0_i_6_reg_1229	9	2	64	128
X_fact_0_i_7_reg_1358	9	2	64	128
X_pow_0_i4_0_reg_512	9	2	64	128
X_pow_0_i4_1_reg_641	9	2	64	128
X_pow_0_i4_2_reg_770	9	2	64	128
X_pow_0_i4_3_reg_899	9	2	64	128
X_pow_0_i4_4_reg_1028	9	2	64	128
X_pow_0_i4_5_reg_1157	9	2	64	128
X_pow_0_i4_6_reg_1286	9	2	64	128
X_pow_0_i4_7_reg_1415	9	2	64	128
X_pow_0_i_0_reg_467	9	2	64	128
X_pow_0_i_1_reg_596	9	2	64	128
X_pow_0_i_2_reg_725	9	2	64	128

X_pow_0_i_3_reg_854	9	2	64	128
X_pow_0_i_4_reg_983	9	2	64	128
X_pow_0_i_5_reg_1112	9	2	64	128
X_pow_0_i_6_reg_1241	9	2	64	128
X_pow_0_i_7_reg_1370	9	2	64	128
ap_NS_fsm	2193	853	1	853
ap_enable_reg_pp0_iter4	9	2	1	2
ap_enable_reg_pp10_iter4	9	2	1	2
ap_enable_reg_pp11_iter4	9	2	1	2
ap_enable_reg_pp12_iter4	9	2	1	2
ap_enable_reg_pp13_iter4	9	2	1	2
ap_enable_reg_pp14_iter4	9	2	1	2
ap_enable_reg_pp15_iter4	9	2	1	2
ap_enable_reg_pp1_iter4	9	2	1	2
ap_enable_reg_pp2_iter4	9	2	1	2
ap_enable_reg_pp3_iter4	9	2	1	2
ap_enable_reg_pp4_iter4	9	2	1	2
ap_enable_reg_pp5_iter4	9	2	1	2
ap_enable_reg_pp6_iter4	9	2	1	2
ap_enable_reg_pp7_iter4	9	2	1	2
ap_enable_reg_pp8_iter4	9	2	1	2
ap_enable_reg_pp9_iter4	9	2	1	2
ap_phi_mux_X_fact_0_i3_0_phi_fu_504_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_1_phi_fu_633_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_2_phi_fu_762_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_3_phi_fu_891_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_4_phi_fu_1020_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_5_phi_fu_1149_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_6_phi_fu_1278_p4	9	2	64	128
ap_phi_mux_X_fact_0_i3_7_phi_fu_1407_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_0_phi_fu_459_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_1_phi_fu_588_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_2_phi_fu_717_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_3_phi_fu_846_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_4_phi_fu_975_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_5_phi_fu_1104_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_6_phi_fu_1233_p4	9	2	64	128
ap_phi_mux_X_fact_0_i_7_phi_fu_1362_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_0_phi_fu_516_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_1_phi_fu_645_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_2_phi_fu_774_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_3_phi_fu_903_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_4_phi_fu_1032_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_5_phi_fu_1161_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_6_phi_fu_1290_p4	9	2	64	128
ap_phi_mux_X_pow_0_i4_7_phi_fu_1419_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_0_phi_fu_470_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_1_phi_fu_599_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_2_phi_fu_728_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_3_phi_fu_857_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_4_phi_fu_986_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_5_phi_fu_1115_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_6_phi_fu_1244_p4	9	2	64	128
ap_phi_mux_X_pow_0_i_7_phi_fu_1373_p4	9	2	64	128
ap_phi_mux_i_0_i1_0_phi_fu_481_p4	9	2	5	10
ap_phi_mux_i_0_i1_1_phi_fu_610_p4	9	2	5	10
ap_phi_mux_i_0_i1_2_phi_fu_739_p4	9	2	5	10
ap_phi_mux_i_0_i1_3_phi_fu_868_p4	9	2	5	10
ap_phi_mux_i_0_i1_4_phi_fu_997_p4	9	2	5	10
ap_phi_mux_i_0_i1_5_phi_fu_1126_p4	9	2	5	10
ap_phi_mux_i_0_i1_6_phi_fu_1255_p4	9	2	5	10
ap_phi_mux_i_0_i1_7_phi_fu_1384_p4	9	2	5	10
ap_phi_mux_i_0_i_0_phi_fu_436_p4	9	2	5	10
ap_phi_mux_i_0_i_1_phi_fu_565_p4	9	2	5	10
ap_phi_mux_i_0_i_2_phi_fu_694_p4	9	2	5	10
ap_phi_mux_i_0_i_3_phi_fu_823_p4	9	2	5	10
ap_phi_mux_i_0_i_4_phi_fu_952_p4	9	2	5	10
ap_phi_mux_i_0_i_5_phi_fu_1081_p4	9	2	5	10
ap_phi_mux_i_0_i_6_phi_fu_1210_p4	9	2	5	10
ap_phi_mux_i_0_i_7_phi_fu_1339_p4	9	2	5	10
b_0_reg_1427	9	2	4	8
empty_10_reg_537	9	2	64	128
empty_17_reg_666	9	2	64	128
empty_24_reg_795	9	2	64	128
empty_31_reg_924	9	2	64	128
empty_38_reg_1053	9	2	64	128
empty_3_reg_408	9	2	64	128
empty_45_reg_1182	9	2	64	128
empty_52_reg_1311	9	2	64	128
grp_fu_1438_p0	85	17	64	1088
grp_fu_1441_p0	149	33	32	1056
grp_fu_1460_opcode	15	3	2	6

grp_fu_1460_p0	53	12	64	768
grp_fu_1460_p1	21	4	64	256
grp_fu_1464_p0	47	10	64	640
grp_fu_1464_p1	15	3	64	192
grp_fu_1484_p0	97	20	64	1280
grp_fu_1484_p1	50	11	64	704
grp_fu_1524_p0	85	17	64	1088
grp_fu_1544_p0	117	25	64	1600
i_0_i1_0_reg_477	9	2	5	10
i_0_i1_1_reg_606	9	2	5	10
i_0_i1_2_reg_735	9	2	5	10
i_0_i1_3_reg_864	9	2	5	10
i_0_i1_4_reg_993	9	2	5	10
i_0_i1_5_reg_1122	9	2	5	10
i_0_i1_6_reg_1251	9	2	5	10
i_0_i1_7_reg_1380	9	2	5	10
i_0_i_0_reg_432	9	2	5	10
i_0_i_1_reg_561	9	2	5	10
i_0_i_2_reg_690	9	2	5	10
i_0_i_3_reg_819	9	2	5	10
i_0_i_4_reg_948	9	2	5	10
i_0_i_5_reg_1077	9	2	5	10
i_0_i_6_reg_1206	9	2	5	10
i_0_i_7_reg_1335	9	2	5	10
j_0_0_reg_420	9	2	4	8
j_0_1_reg_549	9	2	4	8
j_0_2_reg_678	9	2	4	8
j_0_3_reg_807	9	2	4	8
j_0_4_reg_936	9	2	4	8
j_0_5_reg_1065	9	2	4	8
j_0_6_reg_1194	9	2	4	8
j_0_7_reg_1323	9	2	4	8
storemerge1_reg_524	9	2	64	128
storemerge2_reg_653	9	2	64	128
storemerge3_reg_782	9	2	64	128
storemerge4_reg_911	9	2	64	128
storemerge5_reg_1040	9	2	64	128
storemerge6_reg_1169	9	2	64	128
storemerge7_reg_1298	9	2	64	128
storemerge_reg_395	9	2	64	128
sum_0_i2_0_reg_488	9	2	32	64
sum_0_i2_1_reg_617	9	2	32	64
sum_0_i2_2_reg_746	9	2	32	64
sum_0_i2_3_reg_875	9	2	32	64
sum_0_i2_4_reg_1004	9	2	32	64
sum_0_i2_5_reg_1133	9	2	32	64
sum_0_i2_6_reg_1262	9	2	32	64
sum_0_i2_7_reg_1391	9	2	32	64
sum_0_i_0_reg_443	9	2	32	64
sum_0_i_1_reg_572	9	2	32	64
sum_0_i_2_reg_701	9	2	32	64
sum_0_i_3_reg_830	9	2	32	64
sum_0_i_4_reg_959	9	2	32	64
sum_0_i_5_reg_1088	9	2	32	64
sum_0_i_6_reg_1217	9	2	32	64
sum_0_i_7_reg_1346	9	2	32	64
Total	4599	1375	6611	22419

* Register:

Name	FF	LUT	Bits	Const Bits
X_Imag_load_reg_3221	64	0	64	0
X_Real_load_reg_3216	64	0	64	0
X_fact_0_i3_0_reg_500	64	0	64	0
X_fact_0_i3_1_reg_629	64	0	64	0
X_fact_0_i3_2_reg_758	64	0	64	0
X_fact_0_i3_3_reg_887	64	0	64	0
X_fact_0_i3_4_reg_1016	64	0	64	0
X_fact_0_i3_5_reg_1145	64	0	64	0
X_fact_0_i3_6_reg_1274	64	0	64	0
X_fact_0_i3_7_reg_1403	64	0	64	0
X_fact_0_i_0_reg_455	64	0	64	0
X_fact_0_i_1_reg_584	64	0	64	0
X_fact_0_i_2_reg_713	64	0	64	0
X_fact_0_i_3_reg_842	64	0	64	0
X_fact_0_i_4_reg_971	64	0	64	0
X_fact_0_i_5_reg_1100	64	0	64	0
X_fact_0_i_6_reg_1229	64	0	64	0
X_fact_0_i_7_reg_1358	64	0	64	0

```
|X_pow_0_i4_0_reg_512      |   64|   0|   64|          0|
|X_pow_0_i4_1_reg_641      |   64|   0|   64|          0|
|X_pow_0_i4_2_reg_770      |   64|   0|   64|          0|
|X_pow_0_i4_3_reg_899      |   64|   0|   64|          0|
|X_pow_0_i4_4_reg_1028     |   64|   0|   64|          0|
|X_pow_0_i4_5_reg_1157     |   64|   0|   64|          0|
|X_pow_0_i4_6_reg_1286     |   64|   0|   64|          0|
|X_pow_0_i4_7_reg_1415     |   64|   0|   64|          0|
|X_pow_0_i_0_reg_467       |   64|   0|   64|          0|
|X_pow_0_i_1_reg_596       |   64|   0|   64|          0|
|X_pow_0_i_2_reg_725       |   64|   0|   64|          0|
|X_pow_0_i_3_reg_854       |   64|   0|   64|          0|
|X_pow_0_i_4_reg_983       |   64|   0|   64|          0|
|X_pow_0_i_5_reg_1112      |   64|   0|   64|          0|
|X_pow_0_i_6_reg_1241      |   64|   0|   64|          0|
|X_pow_0_i_7_reg_1370      |   64|   0|   64|          0|
|add_ln121_1_reg_2621      |    5|   0|    5|          0|
|add_ln121_2_reg_2712      |    5|   0|    5|          0|
|add_ln121_3_reg_2803      |    5|   0|    5|          0|
|add_ln121_4_reg_2894      |    5|   0|    5|          0|
|add_ln121_5_reg_2985      |    5|   0|    5|          0|
|add_ln121_6_reg_3076      |    5|   0|    5|          0|
|add_ln121_7_reg_3157      |    5|   0|    5|          0|
|add_ln121_reg_2530        |    5|   0|    5|          0|
|add_ln130_1_reg_2631      |   64|   0|   64|          0|
|add_ln130_2_reg_2722      |   64|   0|   64|          0|
|add_ln130_3_reg_2813      |   64|   0|   64|          0|
|add_ln130_4_reg_2904      |   64|   0|   64|          0|
|add_ln130_5_reg_2995      |   64|   0|   64|          0|
|add_ln130_6_reg_3086      |   64|   0|   64|          0|
|add_ln130_7_reg_3167      |   64|   0|   64|          0|
|add_ln130_reg_2540        |   64|   0|   64|          0|
|add_ln38_1_reg_2568       |    4|   0|    4|          0|
|add_ln38_2_reg_2659       |    4|   0|    4|          0|
|add_ln38_3_reg_2750       |    4|   0|    4|          0|
|add_ln38_4_reg_2841       |    4|   0|    4|          0|
|add_ln38_5_reg_2932       |    4|   0|    4|          0|
|add_ln38_6_reg_3023       |    4|   0|    4|          0|
|add_ln38_7_reg_3114       |    4|   0|    4|          0|
|add_ln38_reg_2477         |    4|   0|    4|          0|
|add_ln76_1_reg_2592       |    5|   0|    5|          0|
|add_ln76_2_reg_2683       |    5|   0|    5|          0|
|add_ln76_3_reg_2774       |    5|   0|    5|          0|
|add_ln76_4_reg_2865       |    5|   0|    5|          0|
|add_ln76_5_reg_2956       |    5|   0|    5|          0|
|add_ln76_6_reg_3047       |    5|   0|    5|          0|
|add_ln76_7_reg_3128       |    5|   0|    5|          0|
|add_ln76_reg_2501         |    5|   0|    5|          0|
|add_ln87_1_reg_2602       |   64|   0|   64|          0|
|add_ln87_2_reg_2693       |   64|   0|   64|          0|
|add_ln87_3_reg_2784       |   64|   0|   64|          0|
|add_ln87_4_reg_2875       |   64|   0|   64|          0|
|add_ln87_5_reg_2966       |   64|   0|   64|          0|
|add_ln87_6_reg_3057       |   64|   0|   64|          0|
|add_ln87_7_reg_3138       |   64|   0|   64|          0|
|add_ln87_reg_2511         |   64|   0|   64|          0|
|ap_CS_fsm                 |  852|   0|  852|          0|
|ap_enable_reg_pp0_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp0_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp0_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp0_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp0_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp10_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp10_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp10_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp10_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp10_iter4  |    1|   0|    1|          0|
|ap_enable_reg_pp11_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp11_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp11_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp11_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp11_iter4  |    1|   0|    1|          0|
|ap_enable_reg_pp12_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp12_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp12_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp12_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp12_iter4  |    1|   0|    1|          0|
|ap_enable_reg_pp13_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp13_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp13_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp13_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp13_iter4  |    1|   0|    1|          0|
```

```
|ap_enable_reg_pp14_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp14_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp14_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp14_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp14_iter4  |    1|   0|    1|          0|
|ap_enable_reg_pp15_iter0  |    1|   0|    1|          0|
|ap_enable_reg_pp15_iter1  |    1|   0|    1|          0|
|ap_enable_reg_pp15_iter2  |    1|   0|    1|          0|
|ap_enable_reg_pp15_iter3  |    1|   0|    1|          0|
|ap_enable_reg_pp15_iter4  |    1|   0|    1|          0|
|ap_enable_reg_pp1_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp1_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp1_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp1_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp1_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp2_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp2_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp2_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp2_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp2_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp3_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp3_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp3_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp3_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp3_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp4_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp4_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp4_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp4_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp4_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp5_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp5_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp5_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp5_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp5_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp6_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp6_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp6_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp6_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp6_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp7_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp7_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp7_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp7_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp7_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp8_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp8_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp8_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp8_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp8_iter4   |    1|   0|    1|          0|
|ap_enable_reg_pp9_iter0   |    1|   0|    1|          0|
|ap_enable_reg_pp9_iter1   |    1|   0|    1|          0|
|ap_enable_reg_pp9_iter2   |    1|   0|    1|          0|
|ap_enable_reg_pp9_iter3   |    1|   0|    1|          0|
|ap_enable_reg_pp9_iter4   |    1|   0|    1|          0|
|b_0_reg_1427              |    4|   0|    4|          0|
|b_reg_3195                |    4|   0|    4|          0|
|empty_10_reg_537          |   64|   0|   64|          0|
|empty_17_reg_666          |   64|   0|   64|          0|
|empty_24_reg_795          |   64|   0|   64|          0|
|empty_31_reg_924          |   64|   0|   64|          0|
|empty_38_reg_1053         |   64|   0|   64|          0|
|empty_3_reg_408           |   64|   0|   64|          0|
|empty_45_reg_1182         |   64|   0|   64|          0|
|empty_52_reg_1311         |   64|   0|   64|          0|
|i_0_i1_0_reg_477          |    5|   0|    5|          0|
|i_0_i1_1_reg_606          |    5|   0|    5|          0|
|i_0_i1_2_reg_735          |    5|   0|    5|          0|
|i_0_i1_3_reg_864          |    5|   0|    5|          0|
|i_0_i1_4_reg_993          |    5|   0|    5|          0|
|i_0_i1_5_reg_1122         |    5|   0|    5|          0|
|i_0_i1_6_reg_1251         |    5|   0|    5|          0|
|i_0_i1_7_reg_1380         |    5|   0|    5|          0|
|i_0_i_0_reg_432           |    5|   0|    5|          0|
|i_0_i_1_reg_561           |    5|   0|    5|          0|
|i_0_i_2_reg_690           |    5|   0|    5|          0|
|i_0_i_3_reg_819           |    5|   0|    5|          0|
|i_0_i_4_reg_948           |    5|   0|    5|          0|
|i_0_i_5_reg_1077          |    5|   0|    5|          0|
|i_0_i_6_reg_1206          |    5|   0|    5|          0|
|i_0_i_7_reg_1335          |    5|   0|    5|          0|
|icmp_ln121_1_reg_2617     |    1|   0|    1|          0|
```

icmp_ln121_2_reg_2708	1	0	1	0
icmp_ln121_3_reg_2799	1	0	1	0
icmp_ln121_4_reg_2890	1	0	1	0
icmp_ln121_5_reg_2981	1	0	1	0
icmp_ln121_6_reg_3072	1	0	1	0
icmp_ln121_7_reg_3153	1	0	1	0
icmp_ln121_reg_2526	1	0	1	0
icmp_ln76_1_reg_2588	1	0	1	0
icmp_ln76_2_reg_2679	1	0	1	0
icmp_ln76_3_reg_2770	1	0	1	0
icmp_ln76_4_reg_2861	1	0	1	0
icmp_ln76_5_reg_2952	1	0	1	0
icmp_ln76_6_reg_3043	1	0	1	0
icmp_ln76_7_reg_3124	1	0	1	0
icmp_ln76_reg_2497	1	0	1	0
j_0_0_reg_420	4	0	4	0
j_0_1_reg_549	4	0	4	0
j_0_2_reg_678	4	0	4	0
j_0_3_reg_807	4	0	4	0
j_0_4_reg_936	4	0	4	0
j_0_5_reg_1065	4	0	4	0
j_0_6_reg_1194	4	0	4	0
j_0_7_reg_1323	4	0	4	0
mul_ln130_1_reg_2636	64	0	64	0
mul_ln130_2_reg_2727	64	0	64	0
mul_ln130_3_reg_2818	64	0	64	0
mul_ln130_4_reg_2909	64	0	64	0
mul_ln130_5_reg_3000	64	0	64	0
mul_ln130_6_reg_3091	64	0	64	0
mul_ln130_7_reg_3172	64	0	64	0
mul_ln130_reg_2545	64	0	64	0
mul_ln87_1_reg_2607	64	0	64	0
mul_ln87_2_reg_2698	64	0	64	0
mul_ln87_3_reg_2789	64	0	64	0
mul_ln87_4_reg_2880	64	0	64	0
mul_ln87_5_reg_2971	64	0	64	0
mul_ln87_6_reg_3062	64	0	64	0
mul_ln87_7_reg_3143	64	0	64	0
mul_ln87_reg_2516	64	0	64	0
reg_1563	64	0	64	0
reg_1569	64	0	64	0
reg_1583	64	0	64	0
reg_1588	64	0	64	0
reg_1600	64	0	64	0
reg_1605	64	0	64	0
reg_1612	64	0	64	0
reg_1634	64	0	64	0
reg_1646	64	0	64	0
reg_1665	64	0	64	0
reg_1671	64	0	64	0
reg_1677	64	0	64	0
reg_1682	64	0	64	0
reg_1687	64	0	64	0
reg_1692	64	0	64	0
reg_1701	64	0	64	0
reg_1710	64	0	64	0
select_ln124_1_reg_2641	64	0	64	0
select_ln124_2_reg_2732	64	0	64	0
select_ln124_3_reg_2823	64	0	64	0
select_ln124_4_reg_2914	64	0	64	0
select_ln124_5_reg_3005	64	0	64	0
select_ln124_6_reg_3096	64	0	64	0
select_ln124_7_reg_3177	64	0	64	0
select_ln124_reg_2550	64	0	64	0
select_ln81_1_reg_2612	64	0	64	0
select_ln81_2_reg_2703	64	0	64	0
select_ln81_3_reg_2794	64	0	64	0
select_ln81_4_reg_2885	64	0	64	0
select_ln81_5_reg_2976	64	0	64	0
select_ln81_6_reg_3067	64	0	64	0
select_ln81_7_reg_3148	64	0	64	0
select_ln81_reg_2521	64	0	64	0
storemerge1_reg_524	64	0	64	0
storemerge2_reg_653	64	0	64	0
storemerge3_reg_782	64	0	64	0
storemerge4_reg_911	64	0	64	0
storemerge5_reg_1040	64	0	64	0
storemerge6_reg_1169	64	0	64	0
storemerge7_reg_1298	64	0	64	0
storemerge_reg_395	64	0	64	0
sum_0_i2_0_reg_488	32	0	32	0
sum_0_i2_1_reg_617	32	0	32	0

```
    |sum_0_i2_2_reg_746        |   32|   0|   32|          0|
    |sum_0_i2_3_reg_875        |   32|   0|   32|          0|
    |sum_0_i2_4_reg_1004       |   32|   0|   32|          0|
    |sum_0_i2_5_reg_1133       |   32|   0|   32|          0|
    |sum_0_i2_6_reg_1262       |   32|   0|   32|          0|
    |sum_0_i2_7_reg_1391       |   32|   0|   32|          0|
    |sum_0_i_0_reg_443         |   32|   0|   32|          0|
    |sum_0_i_1_reg_572         |   32|   0|   32|          0|
    |sum_0_i_2_reg_701         |   32|   0|   32|          0|
    |sum_0_i_3_reg_830         |   32|   0|   32|          0|
    |sum_0_i_4_reg_959         |   32|   0|   32|          0|
    |sum_0_i_5_reg_1088        |   32|   0|   32|          0|
    |sum_0_i_6_reg_1217        |   32|   0|   32|          0|
    |sum_0_i_7_reg_1346        |   32|   0|   32|          0|
    |trunc_ln121_1_reg_2626    |    1|   0|    1|          0|
    |trunc_ln121_2_reg_2717    |    1|   0|    1|          0|
    |trunc_ln121_3_reg_2808    |    1|   0|    1|          0|
    |trunc_ln121_4_reg_2899    |    1|   0|    1|          0|
    |trunc_ln121_5_reg_2990    |    1|   0|    1|          0|
    |trunc_ln121_6_reg_3081    |    1|   0|    1|          0|
    |trunc_ln121_7_reg_3162    |    1|   0|    1|          0|
    |trunc_ln121_reg_2535      |    1|   0|    1|          0|
    |trunc_ln76_1_reg_2597     |    1|   0|    1|          0|
    |trunc_ln76_2_reg_2688     |    1|   0|    1|          0|
    |trunc_ln76_3_reg_2779     |    1|   0|    1|          0|
    |trunc_ln76_4_reg_2870     |    1|   0|    1|          0|
    |trunc_ln76_5_reg_2961     |    1|   0|    1|          0|
    |trunc_ln76_6_reg_3052     |    1|   0|    1|          0|
    |trunc_ln76_7_reg_3133     |    1|   0|    1|          0|
    |trunc_ln76_reg_2506       |    1|   0|    1|          0|
    |zext_ln56_reg_3200        |    4|   0|   64|         60|
    |icmp_ln121_1_reg_2617     |   64|  32|    1|          0|
    |icmp_ln121_2_reg_2708     |   64|  32|    1|          0|
    |icmp_ln121_3_reg_2799     |   64|  32|    1|          0|
    |icmp_ln121_4_reg_2890     |   64|  32|    1|          0|
    |icmp_ln121_5_reg_2981     |   64|  32|    1|          0|
    |icmp_ln121_6_reg_3072     |   64|  32|    1|          0|
    |icmp_ln121_7_reg_3153     |   64|  32|    1|          0|
    |icmp_ln121_reg_2526       |   64|  32|    1|          0|
    |icmp_ln76_1_reg_2588      |   64|  32|    1|          0|
    |icmp_ln76_2_reg_2679      |   64|  32|    1|          0|
    |icmp_ln76_3_reg_2770      |   64|  32|    1|          0|
    |icmp_ln76_4_reg_2861      |   64|  32|    1|          0|
    |icmp_ln76_5_reg_2952      |   64|  32|    1|          0|
    |icmp_ln76_6_reg_3043      |   64|  32|    1|          0|
    |icmp_ln76_7_reg_3124      |   64|  32|    1|          0|
    |icmp_ln76_reg_2497        |   64|  32|    1|          0|
    |trunc_ln121_1_reg_2626    |   64|  32|    1|          0|
    |trunc_ln121_2_reg_2717    |   64|  32|    1|          0|
    |trunc_ln121_3_reg_2808    |   64|  32|    1|          0|
    |trunc_ln121_4_reg_2899    |   64|  32|    1|          0|
    |trunc_ln121_5_reg_2990    |   64|  32|    1|          0|
    |trunc_ln121_6_reg_3081    |   64|  32|    1|          0|
    |trunc_ln121_7_reg_3162    |   64|  32|    1|          0|
    |trunc_ln121_reg_2535      |   64|  32|    1|          0|
    |trunc_ln76_1_reg_2597     |   64|  32|    1|          0|
    |trunc_ln76_2_reg_2688     |   64|  32|    1|          0|
    |trunc_ln76_3_reg_2779     |   64|  32|    1|          0|
    |trunc_ln76_4_reg_2870     |   64|  32|    1|          0|
    |trunc_ln76_5_reg_2961     |   64|  32|    1|          0|
    |trunc_ln76_6_reg_3052     |   64|  32|    1|          0|
    |trunc_ln76_7_reg_3133     |   64|  32|    1|          0|
    |trunc_ln76_reg_2506       |   64|  32|    1|          0|
    +--------------------------+-----+----+-----+-----------+
    |Total                     |11120|1024| 9164|         60|
    +--------------------------+-----+----+-----+-----------+


================================================================
== Interface
================================================================
* Summary:
+--------------+-----+-----+------------+--------------+--------------+
|   RTL Ports  | Dir | Bits|  Protocol  | Source Object|    C Type    |
+--------------+-----+-----+------------+--------------+--------------+
|ap_clk        |  in |    1| ap_ctrl_hs |     NDFT     | return value |
|ap_rst        |  in |    1| ap_ctrl_hs |     NDFT     | return value |
|ap_start      |  in |    1| ap_ctrl_hs |     NDFT     | return value |
|ap_done       | out |    1| ap_ctrl_hs |     NDFT     | return value |
|ap_idle       | out |    1| ap_ctrl_hs |     NDFT     | return value |
|ap_ready      | out |    1| ap_ctrl_hs |     NDFT     | return value |
```

```
|X_R_address0  | out |    3|  ap_memory |      X_R     |     array    |
|X_R_ce0       | out |    1|  ap_memory |      X_R     |     array    |
|X_R_we0       | out |    1|  ap_memory |      X_R     |     array    |
|X_R_d0        | out |   64|  ap_memory |      X_R     |     array    |
|X_R_q0        |  in |   64|  ap_memory |      X_R     |     array    |
|X_I_address0  | out |    3|  ap_memory |      X_I     |     array    |
|X_I_ce0       | out |    1|  ap_memory |      X_I     |     array    |
|X_I_we0       | out |    1|  ap_memory |      X_I     |     array    |
|X_I_d0        | out |   64|  ap_memory |      X_I     |     array    |
|X_I_q0        |  in |   64|  ap_memory |      X_I     |     array    |
+--------------+-----+-----+------------+--------------+--------------+
```

APPENDIX C

correlate_cumulated.py

```
import csv

#  reading csv to list [['code_lineNo', 'hls_net_name'] e.g. ['0', 'X_R']]
with open('HLS_nts.csv', 'r') as f:
  reader = csv.reader(f)
  net_names = list(reader)

# reading all congested net names to list
with open('Congested_nets.csv', 'r') as f:
    lines = f.read().splitlines()


#count for each HLS_net, occuring how many times in congested nets
for i in range(len(net_names)):
    word = net_names[i][1]    # get name of single HLS_Net
    count = (sum(word in line for line in lines))
    net_names[i].append(count)


#filter out HLS_nets with zero match in congested nets
net_names_filter = []
sum = 0
already_present = False;

for word in net_names:
    if(word[2]!=0):  #   filter for value not 0
        already_present = False  #  check for already existing in net_names_filter array
        for o in net_names_filter:    # if already present in net_names_filter, then add in same row
            if o[0] == word[0] and o[1] == word[1]:
                o[2] = o[2]+word[2]
                already_present = True
                break
        if not already_present:
            net_names_filter.append(word)
        sum = sum + word[2]

already_present = False
# concatenate row number and sum
words_filter_v1 = []
for word in net_names_filter:
    already_present = False
    for o in words_filter_v1:    # if already present in words_filter, then add in same row
        if o[0] == word[0]: # if row number already present in words_filter, then add in same row
```

```
            o[1] = o[1] + "," + word[1]
            o[2] = o[2] + word[2]
            already_present = True
            break
    if not already_present:
        words_filter_v1.append(word)
words_filter_v1.sort(key=lambda x: x[2], reverse=True)

# saving to csv
with open("Match_cumulated.csv", 'wb') as myfile:
                    wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
                    wr.writerows(words_filter_v1)
```

Bibliography

[1] M. Fingeroff and T. Bollaert. *High-Level Synthesis Blue Book*. Mentor Graphics Corporation, 2010

[2] D. D. Gajski and R. H. Kuhn. *Guest Editor's Introduction: New VLSI Tools*. IEEE Computer, December 1983

[3] Grant Martin, Gary Smith. *High-Level Synthesis: Past, Present, and Future*, IEEE Design & Test of Computers, July/August 2009

[4] Donald G. Bailey School of Engineering and Advanced Technology Massey University Palmerston North, New Zealand. *The Advantages and Limitations of High-level synthesis for FPGA Based Image Processing*

[5] J. Sanguinetti. *Understanding high-level synthesis design's advantages*. EE Times Asia, 26 April 2010

[6] F. Winterstein, S. Bayliss, and G. A. Constantinides. *High-level synthesis of dynamic data structures: A case study using Vivado HLS*. In International Conference on Field Programmable Technology, 2013

[7] Vivado Design Suite, User Guide. *High-Level Synthesis, UG902 (v2012.4).* Xilinx, Inc. December 18, 2012

[8] *Catapult® High-Level Synthesis data sheet*. Mentor

[9] *https://www.mentor.com/hls-lp/success/bosch-visiontec*

[10] *Introduction to FPGA Design with Vivado HLS, UG998 (v1.1).* Xilinx, Inc. January 22, 2019

[11] *High-Level Synthesis, UG871 (v2017.1).* Xilinx, Inc. May 5, 2017

[12] *Vivado Design Suite Tcl Command Reference Guide UG835 (v2017.3).* Xilinx, Inc. October 04, 2017

[13] Prashant Saxena, Rupesh S. Shelar, Sachin S. Sapatnekar. *Routing Congestion in VLSI Circuits: Estimation and Optimization.* Springer, Boston, MA

[14] UltraFast Design Methodology Guide, UG949 (v2018.2). Xilinx, Inc. June 7, 2017

[15] Chi-Li Yu, Kevin Irick. *Multidimensional DFT IP Generator for FPGA Platforms*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, VOL. 58, NO. 4, APRIL 2011

[16] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. *Parallel Programming for FPGAs.* http://hlsbook.ucsd.edu. Copyright 2011-2018.

[17] *https://www.xilinx.com/html_docs/xilinx2019_1/sdsoc_doc/hls-pragmas* (SDSoC Development Environment Help)