"Vitis – Xilinx Command Line Terminal Development"

Major Project Report – Part 2

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

(Embedded Systems)

By

Urvish Kosta Nirmal

(21MECE04)

Department of Electronics and Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

May 2023

CERTIFICATE

This is to certify that the Major Project Report entitled "**Vitis – Xilinx Command Line Terminal Development**" submitted by Mr. Urvish Kosta Nirmal (21MECE04) towards the partial fulfilment of the requirements for the award of degree in Master of Technology in the field of Electronics & Communication Engineering of Nirma University is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for the award of any degree or diploma.

Date: 11/05/2023 **Place: Ahmedabad**

Internal Guide

Dr. Ruchi Gajjar
Assistant Professor
Electronics and Communication Engineering Department
Institute of Technology, Nirma University

PG Coordinator

Dr. Nagendra Gajjar
Professor & PG Coordinator
Electronics & Communication Engineering Department
Institute of Technology, Nirma University

Head of Department

Dr. Usha Mehta
Professor and Head
Department of Electronics & Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

Director

Dr. Rajesh N Patel
Director (i/c)-Institute of Technology,
Dean(i/c) - Faculty of Technology & Engineering
Nirma University,
Ahmedabad 382 481

Acknowledgement

When you travel with someone, the trip is easier. Independence is unquestionably less important than interdependence. The work that went into this thesis was done with the help and support of numerous people. The fact that I can now thank them all is a good element.

With immense pleasure, I express my sincere gratitude, regards and thanks to **Dr Ruchi Gajjar** Assistant Professor, EC Department, Institute of Technology, Nirma University for her excellent guidance, invaluable suggestions and continuous encouragement at all the stages of my research work. Her interest and confidence in me were the reason for all the success I have made. I have been fortunate to have her as my guide as she has been a great influence on me, both as a person and as a professional. I would like to express my gratitude and sincere thanks to **Dr Nagendra Gajjar**, PG Coordinator of MTech Embedded Systems, **Dr. Usha Mehta** Professor Head, Department of Electronics & Communication for guidance and encouragement during the project and the entire Department of Electronics & Communication, Nirma University for a great and memorable learning experience.

It was a pleasure to be associated with Data Center and Communications Group - Software Programmable Acceleration AMD-Xilinx Inc. Hyderabad and I would like to thank the entire team. I would like to thank **Mr. Sadanand Mutyala** for continuous support and guidance. I would also like to thank **Mr. Vidhumouli Hunsigida** for mentoring and helping me throughout and your support and advice have helped in shaping my professional career. **Mr. Prashant Malladi** for mentoring and helping me with the project work. To acknowledge help taken from seniors is always a joy.

- **Urvish Kosta**
  (21MECE04)

Abstract

The fourth industrial revolution is all about accelerating digital transformation. Organizations of every type are creating data-driven business models which look to derive maximum value out of what they now see as their most asset. Data collection and analysis mean more data centers, which in turn require skilled professionals to run them. Today, data centers are turning to AI to autonomously manage various tasks, from server optimization to equipment monitoring.

Field-programmable gate arrays (FPGAs) are a class of silicon devices that can be configured by the end-user to serve a variety of purposes. flexible nature, FPGAs have found applications in countless industries, including (but not limited to) defense, manufacturing, supercomputing, telecommunications, and healthcare. More recently, they have been positioned as the answer to the challenges of running artificial intelligence (AI) at scale. Within the data center, FPGAs are particularly suitable for building hardware accelerators.

For all their benefits, one enduring criticism levelled at FPGAs is the complexity of the software development process, and the company is attempting to solve this with a free developer environment called Vitis. Using Vitis, users can write software for FPGAs in familiar high-level languages such as C, C++ and OpenCL. Meanwhile, ACAP devices enable ML inference to be coded at the framework level – using familiar tools like Caffe, PyTorch, or TensorFlow.

So, the work focuses on embedded software development for Vitis unified software by Xilinx Inc. Major work has been carried out on commands for Xilinx Software Command-Line Tool (XSCT). Xilinx Software Command-line Tool is an interactive and scriptable command-lineinterface to Xilinx SDK. As the outcome, this Internship work carried out at AMD Xilinx Inc. Contribution has been done for developing commands for the XSCT i.e., Vitis CLI depends onthe customer requirements and company product roadmaps and in the area of Vitis Software backend tools. Backend tools consist of various embedded development tools like debuggers, bootable image creators etc.

Declaration

This is to certify that:

a. The thesis comprises my original work toward the degree of Master of Technology in Embedded System at Nirma University and has not been submitted elsewhere for a degree.
b. Due acknowledgement has been made in the text to all other material used.

- Urvish Kosta
(21MECE04)

Table of Contents

Table of Figure

List of Tables

Abbreviations

Table 1 Abbreviations Table

Abbreviation	Description
ACAP	Adaptive Compute Acceleration Platform
AIE	AI Engine
BSP	Board Support Package
ELF	Executable And Linkable Format
FPGA	Field Programmable Gate Arrays
FSBL	First Stage Bootloader
GDB	Gnu Project Debugger
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
QEMU	Quick Emulator
SDK	Software Development Kit
SoC	System On a Chip
TCF	Target Communication Framework
WDB	Waveform Database
XSA	Xilinx Shell Archive

AMD Xilinx - Company Profile

Xilinx is the inventor of the FPGA, programmable SoCs, and now, the ACAP. Our highly flexible programmable silicon, enabled by a suite of advanced software and tools, drives rapid innovation across a wide span of industries and technologies - from consumers to cars to the cloud. Xilinx delivers the most dynamic processing technology in the industry, enabling rapid innovation with its adaptable, intelligent computing. Ross Freeman, Bernard Overestimate, and James V Barnett II co-founded the company in 1984. Xilinx is in San Jose, California,and was founded in Silicon Valley.

Xilinx created the first FPGA in 1984, spawning a new industry, enabling custom solutions for any variety of applications and has got a track record of innovative products over years. In the hardware space, Xilinx moved beyond the FPGA to deliver a new set of adaptive processors with Versal in 2019 as the First Adaptive Compute Acceleration Platform. Our software suite drives rapid innovation across a wide span of industries and technologies. Xilinx provides a range of development tools to enable a programming environment for every developer.

Figure 1 Xilinx Hardware Profile [1]

Domains where Xilinx Products are used widely are:

- Aerospace & Defense
- ProAV & Broadcast
- Automotive
- Data Center
- Industrial & Vision
- Healthcare & Sciences
- Test & Measurement, and Emulation

Group Profile

Xilinx Inc has many groups and is at many locations like San Jose, USA, with additional offices in Longmont, USA; Dublin, Ireland; Singapore; Hyderabad, India; Beijing, China; Shanghai, China; Brisbane, Australia and Tokyo, Japan.

Data Center and Communications Group - Software Programmable Acceleration

Software Programmable Acceleration works on VITIS IDE, Backend tools like Bootgen, prep target, Traditional compilers for ARM and MB, integration to IDE, and All the works related to AIE Cardano toolchains.

Data levels are growing exponentially, not just in the data center but at the edge and various endpoints. As we enter the fourth industrial revolution, advances in technology are changing the way we live, work, and relate to each other. For businesses, automation of traditional manufacturing and industrial practices means more data produced and consumed; and the secret to success often lies in squeezing the maximum value out of the oceans of data at their disposal.

The so-called fourth industrial revolution is all about accelerating digital transformation. Organizations of every type are creating data-driven business models which look to derive maximum value out of what they now see as their most valuable asset. Data collection and analysis mean more data centers, which in turn require skilled professionals to run them. Today, data centers are turning to AI to autonomously manage various tasks, from server optimization to equipment monitoring. Figure 2 Team Overview shows team hierarchy under the data center group.

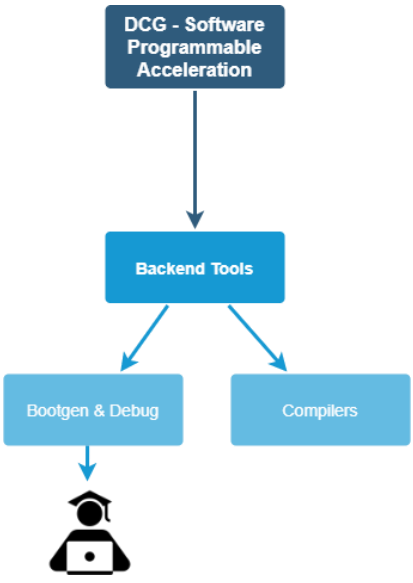

Figure 2 Team Overview

Chapter 1. Introduction

1.1. Motivation

Data centers are evolving rapidly due to exploration in AI, increasingly complex loads, and the explosive growth of random data. The Xilinx platform drives this revolution with an adaptive acceleration of computing, storage and networking. Accelerate your entire application – Use a domain-specific architecture to accelerate inference for AI, processing, and other critical workloads. For low-latency AI inference, Xilinx provides the high data throughput with the lowest latency across a wide range of networks and data types. FPGA-based adaptive computing has often proven to be an efficient and cheap solution for running complex AI workloads. The Vitis is a development kit for seamlessly building accelerated applications. A full set of graphical and CLI, including compilers, analyzers, and debuggers for creating, analyzing, and debugging performance bottlenecks, acceleration algorithms developed in C, C ++, or OpenCL. Use these features in your IDE or use the standalone Vitis IDE.

1.2. Organization of Report

This report covers Company and Group Information as of the preface to the report. Chapter 1 describes the basic introduction and timeline of the tasks performed and the table of the tasks assigned during the course work of the internship. Chapter 2 covers the details of the literature survey about the Vitis Unified software. Vitis CLI ie. XSCT and the Xilinx system debugger on which the team works. Chapter 3 deals with the tools used by Xilinx Inc. and the technologies used during this internship course work for the software development work. Later Chapter 4 explains the development work contains majorly the Stories that have been submitted to the 2022.1 release for Vitis unified software and the contribution. concluding statements are shown in Chapter 5. Chapter 6 shows the extra learning and certification carried out along with the internship coursework.

1.3. Timeline

Table 2 Project Timeline shows the project timeline which started on 7 June 2022 and will end on 7 June 2022. The timeline has been divided into mainly 3 sections as per the software release cycle of Xilinx Inc. with 2 release per year so major contribution for 2021.2, 2022.1 and 2022.2 has been added to a timeline for detailed work please refer to Table 3 List of CRs & Stories which has details of Change Requests and stories assigned.

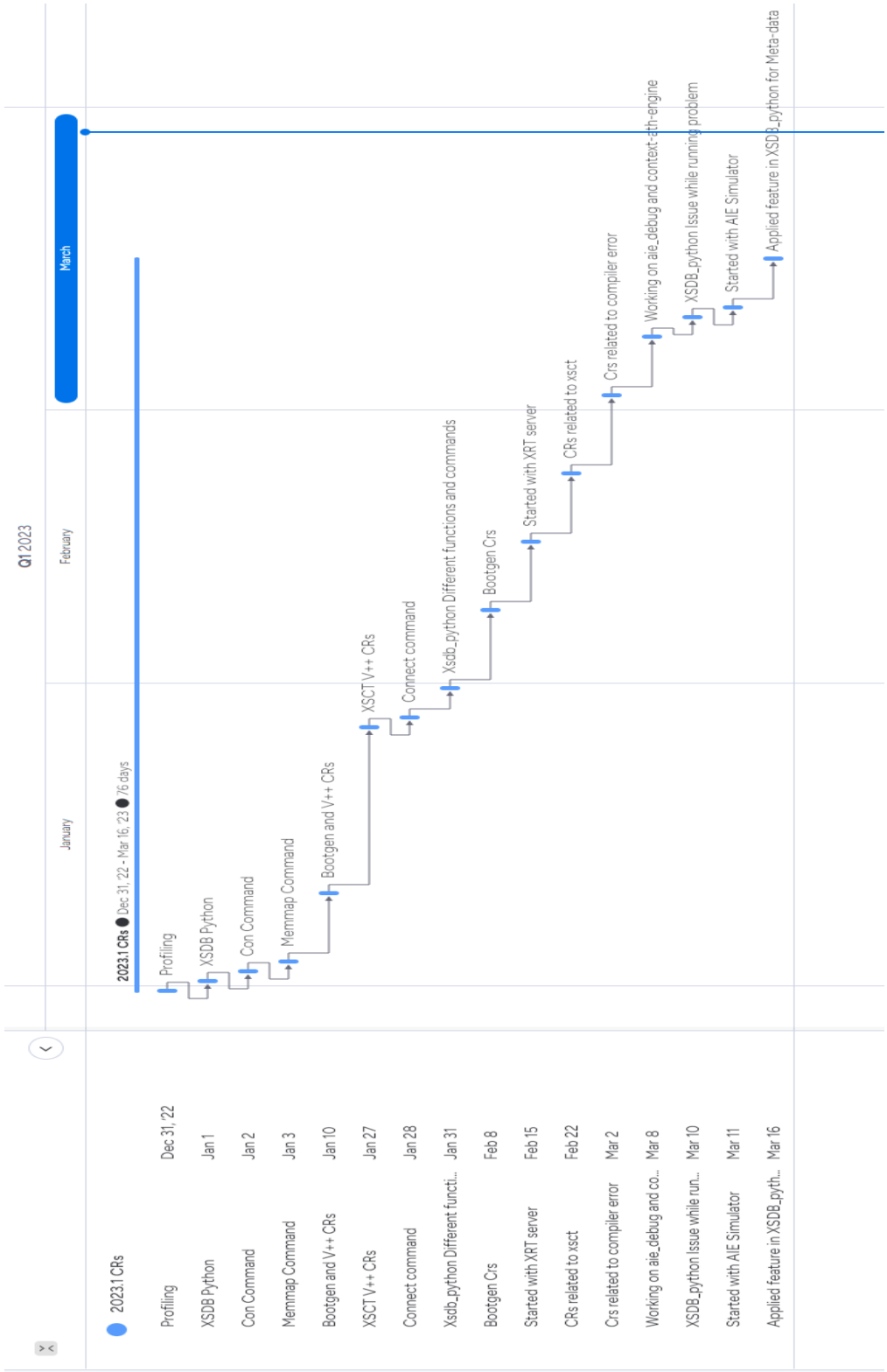

Figure – 2: List of CRs and Stories

Chapter 2. Literature Review

2.1. Vitis Unified Software Platform

The Vitis is a new tool that integrates all aspects of software development for Xilinx hardware into an integrated environment. The Vitis software is an embedded software development process suitable for users who want to migrate to the next-generation technology Xilinx Software Development Kit (SDK) and an acceleration development process suitable for software developers using the latest Xilinx. Supports. Faster FPGA software.

2.2. Vitis Software Development Workflow

Figure 4 Vitis Development Tool Shows the Vitis software can be used to develop applications above Heterogeneous Compute, Cloud to AI deployment & AI Enablement over Hardwareprofile.

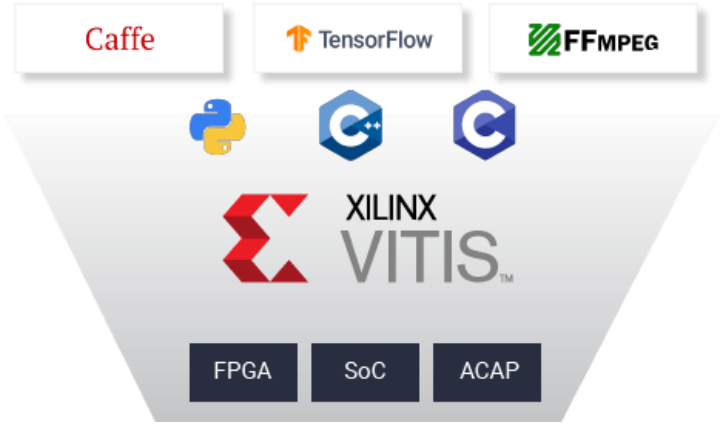

Figure 3 Vitis - Software Development Tool [1]

Figure 3 Vitis - Software Development Tool shows the overview of the Vitis Development tool that is used for Xilinx hardware to make applications leveraging C, C++, and Python environments to use higher-level frameworks [2].

The following figure shows the workflow for developing a software application for the Vitis.

- A hardware engineer designs the logic and exports the information needed for development from the Vivado Design Suite to an xsa file.
- Developers import XSA to the Vitis by creating a platform. This platform is frequently used in projects. To integrate the Vitis workspace architecture for in all types of applications, software development projects are currently migrating to platform and application architectures. The platform contains hardware specifications and software preferences.
- Software preferences are called domains and are part of the platform.

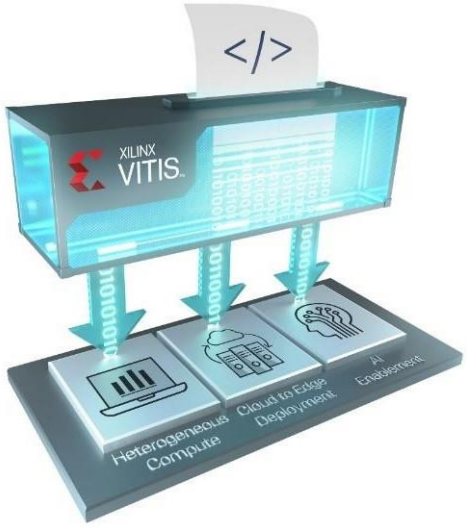

Figure 4 Vitis Development Tool [1]

- Software developers create applications based on platform and domain.
- Applications can be debugged.
- In complex systems, multiple applications can run simultaneously and communicate with each other. Therefore, system level validation should also be performed.
- When everything is ready, the IDE will help you create a boot image that initializes your system and launches your application [3].

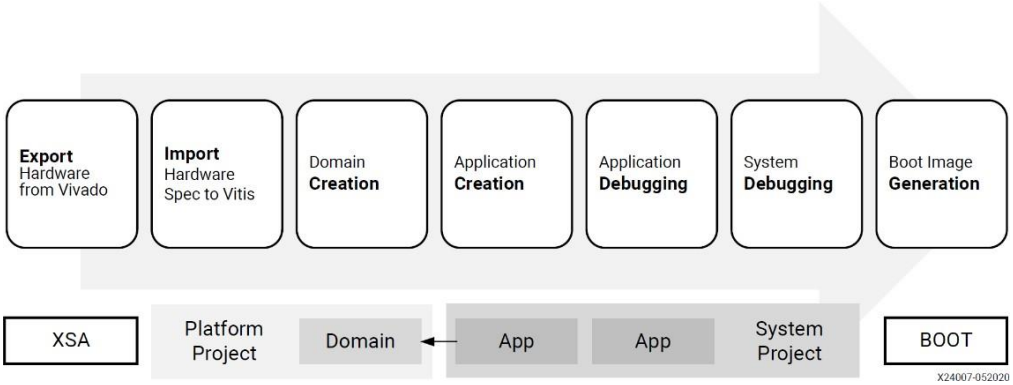

Figure 5 Vitis Design Flow [3]

Figure 5 Vitis Design Flow shows the pictorial representation of the flow explained above.

2.3. Workspace Structure in the Vitis Software Platform

There are two types of projects in the Vitis workspace.

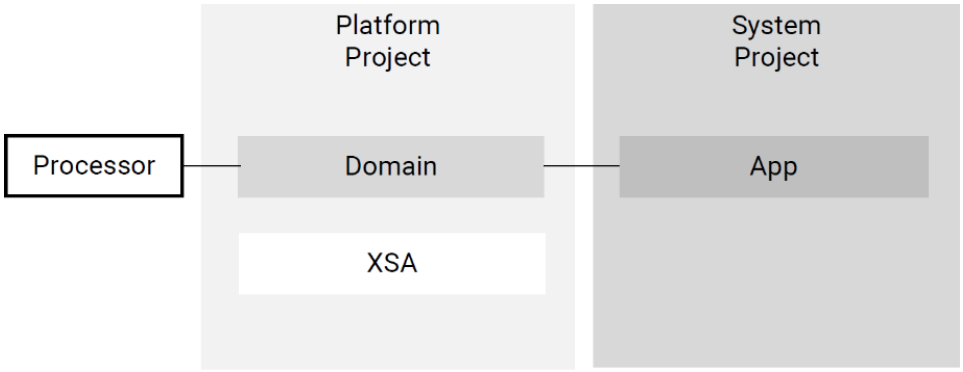

Figure 6 Vitis Workspace Structure [3]

Platform Project: The platform project provides information regarding hardware and a software runtime environment.

System project: Combination of applications running concurrently on a device.

Workspace: After opening the Vitis, create a workspace. The workspace is the location that the Vitis uses to store project data and metadata. You must specify the initial workspace when launching the Vitis.

Platform: A target platform or platform is a combination of hardware components (XSA) and software components (such as domain / BSP, boot components such as FSBL).

Platform project: You can customize the platform project to add domains & change domain settings. Projects can be created by importing XSA or by importing an existing platform. You cannot edit the platform in the repository. Only the platforms in the workspace are calledplatform projects. You can edit the platform project to create more domains. You can create multiple system projects on the same platform project to share hardware and software preferences.

System project: A system project is a combination of applications running concurrently on a device. You cannot put two standalone apps for the processor that are similar together in a system project. You can combine two Linux applications into one system project. A workspace can contain multiple system projects.

Application: A project can have many applications. Each software project requires a respective domain.

XSA: XSA is exported from the Vivado Design Suite. It includes hardware specifications such as processor configuration, peripheral connections, mapping address, and initialization of the device. You need to provide the XSA when you create the platform project.

Domain: The domain is a BSP or OS. Use a collection of software drivers to build yourapplication. The software image created contains only part of the Xilinx library used by the embedded theme. You can create multiple applications that run in your domain. A domain is associated with a single processor or a group of isomorphic processors on the platform (such as A53_0 or A53).

Flow: In the Vitis integrated software platform, software development workflows and application acceleration share a major part, but there are still subtle differences. The Vitisplatform has different configurations to support different use cases.

Embedded: This platform supports software development for embedded Arm processors. And MicroBlaze processor.
Embedded Acceleration: In addition to embedded SD, application for the accelerated use case is also supported by the platform. This platform provides a clock, bus interface, and an interrupt controller for using the acceleration kernel.
Data Center Acceleration: Acceleration kernels and x86 host applications can be developed on this platform. The kernel is controlled via the PCIe bus.

2.4. Vitis - Software Stack

Figure 7 Vitis Software Stack [1]

Figure *7* Vitis Software Stack shows the software stack for the Vitis Unified software, Xilinx Runtime Server runs above the hardware which communicates between the processor cores and FGPA fabric.

The Vitis platform is based on a stack-based architecture and can be seamlessly inserted into development systems and standard open-source build environments, but most importantly, it contains a stylish set of standard libraries. rich. The base layer is the target platform of Vitis, consisting of a board and pre-programmed I/O. The second layer is called Vitis Core SDK and contains the open source Xilinx runtime library to manage the movement of data betweendifferent domains, including subsystems, artificial intelligence engine in Versal Upcoming ACAP and servers. outside. This class also includes basic development tools such as a compiler,

an analyzer, and a debugger. Offers a design environment, the tools are designed to seamlessly work with industry requirements development environments and builds [3].

There are around 400 or more open source applications that are optimized in 8 Vitis libraries on the third layer. These include the Vitis Basic Linear Algebra Subroutine Library (BLAS), the Vitis Solver Library, the Vitis Security Library, and the Vitis Vision Library, the Vitis Data Compression Library, the Vitis Quantitative Finance Library, Vitis DB Library and Vitis AI Library These allow developers to use accelerated functionality before standard API calls [2][3
].

2.5. Xilinx Software Command-Line Tool (XSCT)

GUI development tools, such as the Vitis unified software, can help you get up to speed on new processor architecture development. It aids in abstracting and grouping the majority of the information.

The common functions have been transformed into logical wizards that can be used by even the most inexperienced user. However, a tool's script ability is also critical for allowing users to expand what they can accomplish with it. It's very beneficial when writing nightly regressiontests or when running commands that the developer uses frequently. XSCT is a command-line interface to the Vitis IDE that is interactive and scriptable. The programming language for XSCT,like those of other Xilinx tools, is based on the Tcl. XSCT commands can be run in an interactiveway or as programs for automation [4]. The following are supported by XSCT:

- Creating domains, platform projects, system projects, and application projects
- Managing repositories
- Changing toolchain preferences
- Configuring and building domains/BSPs and applications
- Downloading and running applications on hardware targets
- Using the Bootgen and program flash tools to create and flash boot images

2.6. Xilinx System Debugger (XSDB)

System Debugger uses hw server as the basic debugging tool. The SDK converts each UI action into several TCF commands. It makes the system debugger output show the part of the code being debugged. It makes communication with the hardware using hw server. Figure 8 Debug Workflow shows the XSDB working with the hardware server [4].

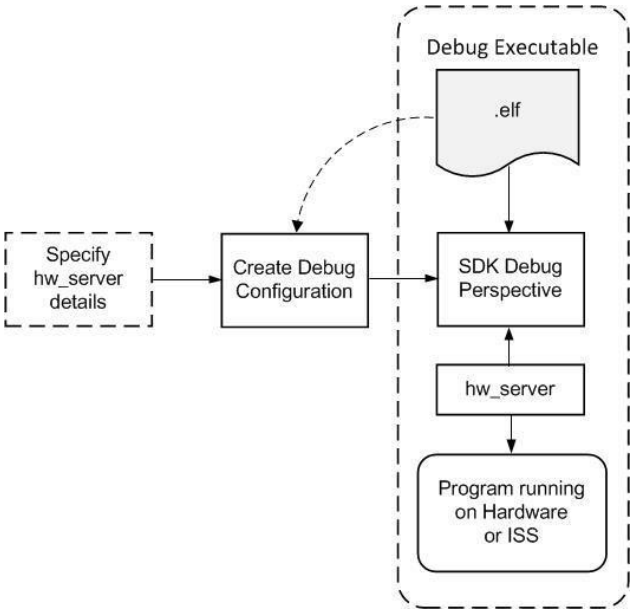

Figure 8 Debug Workflow [4]

The workflow contains the following components:

- **Executable ELF file:** To debug an application, you must debug it using a compiled Executable and Linkable Format (ELF) file. The Debug ELF file contains additional debugging information that allows the debugger to create a direct link between thesource code and the binaries built from this source. See Build Configuration for more information.

- **Debug Configuration:** To start a debug session, you need to create a debugconfiguration in the SDK. This configuration captures the options needed to start adebug session, including the name of the executable file, the CPU target to debug, and other information. For more information, see Performing Configuration.

- **SDK Debug Perspective:** The Debug Perspective allows you to manage the debugging of or the execution of programs in the workbench. You can control program execution by setting breakpoints, pausing a running program, stepping through your code, and examining the contents of variables.

Chapter 3. Tools & Technology

3.1. Target Communication Framework

TCF is a vendor-independent, lightweight, extensible network protocol primarily used to communicate with embedded systems (targets). The most notable feature is that TCF aims to seamlessly connect value servers between tools and targets. However, even without added value, the protocol has the potential to integrate many of today's independent communication links, save resources, and be easier to install and configure than currently integrated development scenarios. Specific transmissions such as TCP / IP, serial lines, SSH tunnels, etc. Third-party providers are ready to use all services from standard TCP / IP channels to custom channels (JTAG (Joint Test Action Group) or proprietary hardware connectivity), adapting new transports and adding value right away. [5].

For OSI (Open Systems Interconnection) models, TCF supports layers 5-7. It is currently assumed that the underlying bearer's reliable end-to-end transport is available. TCF handles (5) session-layer host-to-host communication. (6) JSON-based presentation layer data representation and encryption. (7) The TCF service in the application layer can be regarded as an application [5]. The tool communication model is shown in Figure 9 TCF Protocol Components.

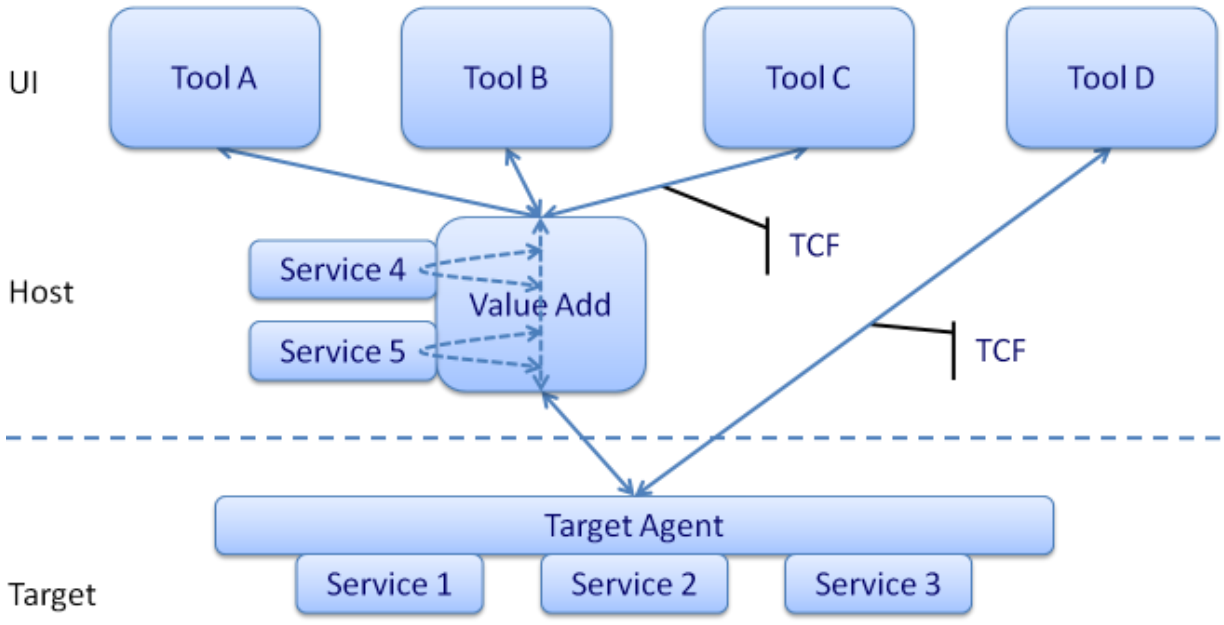

Figure 9 TCF Protocol Components [5]

3.2. GRADLE

Gradle is an Associate in open supply build automation tool that's versatile enough to make virtually any sort of code. Below could be a general summary of a number of the foremost vital options [6].

[1] High performance
Gradle avoids gratuitous work by doing solely what must be done as a result of the input or output has modified. you'll be able to conjointly use the build cache to employ task output from previous runs or from different systems (using a shared build cache). There area unit several different optimizations that Gradle makes, and therefore the development team is consistently operating to boost Gradle's performance.
[2] JVM Foundation
Gradle runs on prime of the JVM and needs the Java Development Kit (JDK) to be put into use it. this is often a bonus for users at home with the Java platform, because it permits you to use customary Java Apis in your build logic, like custom action sorts and plugins. It conjointly makes it easier to run Gradle on all platforms.
[3] Conventions
Gradle area unit is taken from the mavin book and implements rules to form it easier to form common styles of comes as Java comes. With the correct plugin applied, you'll be able to simply get lightweight build scripts for several comes. However, these rules don't limit you. Gradle permits you to override rules, add your actions, and build several different rule-based changes to your build.
[4] Extensibility
you'll be able to conjointly simply extend Gradle to produce your drawback sorts or to form templates. See automaton build support samples. It adds several new construction ideas like flavors and builds designs.
[5] IDE support
you'll be able to import and work with Gradle builds exploitation many major days like automaton Studio, IntelliJ plan, Eclipse and NetBeans. Gradle conjointly supports generating the answer files required to load the project in Visual Studio.
[6] Insight
Build scans give in-depth info regarding build execution that you just will use to spot build issues. This area unit is particularly sensible at distinguishing build performance problems. you'll be able to conjointly share the build scan with different users. this is often particularly helpful if you wish for recommendations on coping with construction problems.

3.2.1. Build phases
A Gradle build has three distinct phases:

[1] Initialization

Gradle supports single project builds and multi-project builds. During the initialization phase, Gradle decides which projects will participate in the build process and creates a project instance for each of those projects.

[2] Settings

In this phase, project objects are configured. The build script for all projects that are part of the build will be executed.

[3] Running

Gradle defines a subset of the tasks created and configured during the configuration phase. The subset is determined by the task name arguments passed to the Gradle command and the current directory. Gradle then performs each of the selected tasks.

Figure 10 Gradle Build Execution shows the build execution snippet.

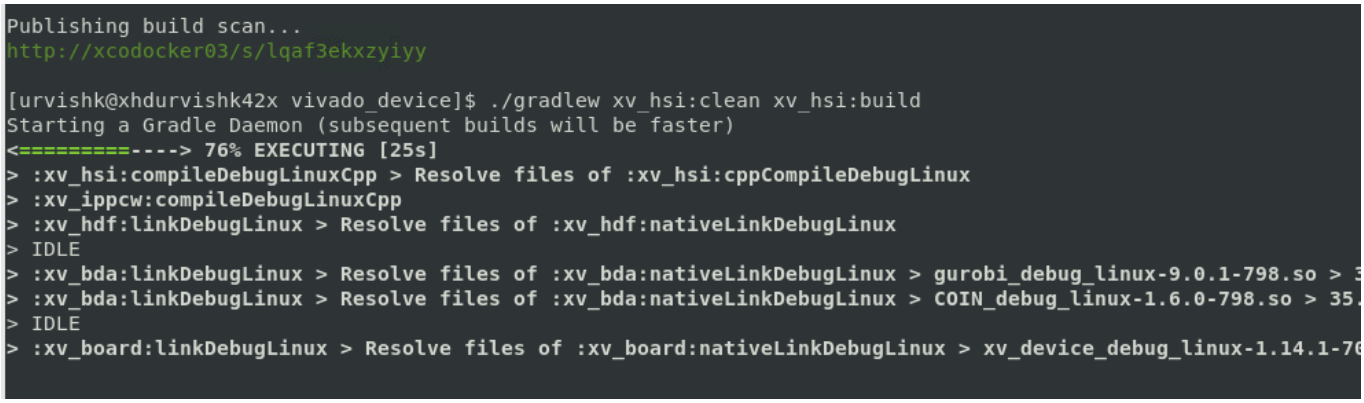

Figure 10 Gradle Build Execution [6]

3.3. JIRA

From concept development to client work, Jira is a set of flexible work management solutions that empower all teams to work together to do the best work of their lives. Jira offers several products and deployment options designed specifically for software developers, IT, business, operations, and more. Jira enables teams to plan, allocate, track, report and manage tasks and connect teams to deliver and support clients to startups and enterprises in agile software development [7].

Plan
Create user stories and issues, plan sprints, and distribute tasks across the team.
Track
Prioritize and discuss work in full context.
Release
Release to customers with confidence when the software is up to date.
Report
Improve team performance based on real-time, visual data that your team can put to use.

Figure 11 Jira Development Flow shows the development cycle for the change request on the JIRA platform.

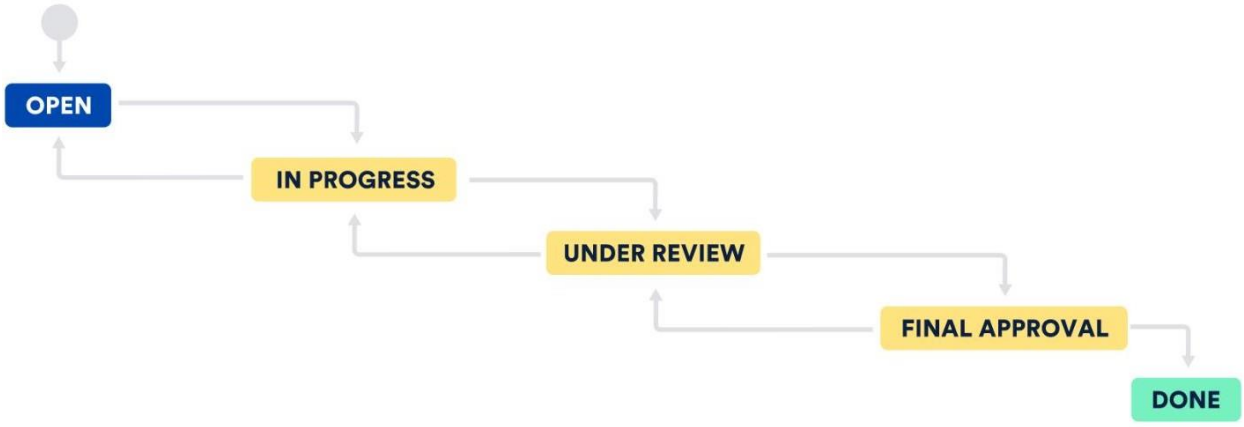

Figure 11 Jira Development Flow [7]

3.4. Total View

Total View is a software debugger for complex C, C++ and Fortran applications running hundreds to thousands of parallel processes. Intuitively diagnose and understand complex code so you can troubleshoot bugs, memory issues, and runtime crashes faster [11][12][8].

Reverse debugging:
Work backwards from failure and eliminate the need to repeatedly restart the application.
Multi-language applications debugging:
Easily analyze and debug apps written in both Python and C/C++.
Simultaneous debugging:
Get complete control over program execution within a single thread or within groups of processes or threads.
Pinpointing and fixing bugs:
Troubleshoot difficult problems that occur in concurrent programs that take advantage of threads such as OpenMP, MPI, GPUs, or coprocessors.

Designed specifically for multi-core and parallel computing, TotalView provides a set of tools that provide unprecedented control over process and thread execution, as well as deep visibility into program and data state.

3.5. Perforce P4V

The Helix Visual Client, P4V, is a cross-platform graphical user interface for Helix core servers, also known as Helix servers. Helix Core Server is an employer model controller that you can use to work with delivery documents and other documents that contain multiple revisions of manuals, web pages, or work machine management documents. Documents managed using Helix Core Server can be found in the depot. To paint a document, open the document and edit it in the workspace. When you're done, use the change list to file the modified document to the depot. The depot continues the songs of all the latest and previous revisions of the record [11].

Customers of Helix Core Server combine the use of consumer software such as P4V with a shared dataset repository. P4V connects your PC to Helix Server, allowing you to exchange documents between your Helix Server depot and your workspace.

Workspace: A folder or directory on your workstation that works with revisions of files managed by Helix Core Server.
Helix Core App: P4V (or another Helix Core application such as a command-line client or P4VS, Helix plugin for Visual Studio) running on your workstation makes requests from the Helix Core server and makes requests for these requests. Deliver the results (file, status). Information etc.) to you.
Helix Server: A program that responds to requests from Helix Core applications, maintains vault files, and tracks workspace status.
Depot: File repository hosted by Helix server. Contains all existing versions of all files submitted so far. The Helix server can host multiple depots, but the examples in this guide show a single depot.
Figure 12 Perforce P4v Overview presents the tool overview and the space for the deport and workspace in P4V software.

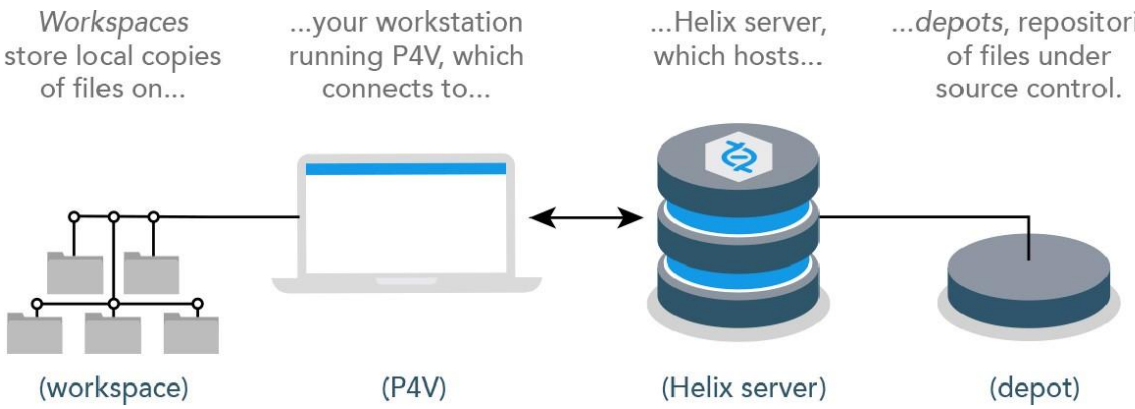

Figure 12 Perforce P4v Overview [11]

3.6. C++

The C++ programming language was created by Bjarne Stroustrup and his team at Bell Laboratories (AT&T, USA) to implement modeling projects in an object-oriented and efficient way. An early version, originally called "C with Classes," dates back to 1980. As the name C++ suggests, C++ was a derivative of the C programming language. ++ is the increment operator in C. In 1989, the American National Standards Institute (ANSI) was established to standardize the C++ programming language. The goal was to get as many compiler vendors and software developers as possible to agree on a single language description to avoid confusion due to multiple dialects. In 1998, the International Organization for Standardization (ISO) approved the standard for C++ (ISO/IEC 14882)[9].

C++ is not a purely object-oriented language but a hybrid that contains the functionality of the C programming language. This means that you have all the features that are available in C:

- universally usable modular programs
- efficient, close to the machine programming
- portable programs for various platforms.

The large quantities of existing C source code can also be used in C++ programs. C++ supports the concepts of object-oriented which are:

- data abstraction, that is, the creation of classes to describe objects
- data encapsulation for controlled access to object data
- inheritance by creating derived classes (including multiple derived classes)
- polymorphism (Greek for multiform), that is, the implementation of instructions that can have varying effects during program execution.

Object-oriented programming offers several major advantages to software development:

- reduced susceptibility to errors: an object controls access to its own data. More specifically, an object can reject erroneous access attempts
- easy re-use: objects maintain themselves and can therefore be used as building blocks for other programs
- low maintenance requirement: an object type can modify its own internal data representation without requiring changes to the application.

3.7. TCL

Tcl stands for Tools Command Language. This name reflects the strengths of Tcl as a scripting language for merging other applications into new applications. Tcl was developed by Dr. John Ousterhout while at UC Berkeley. He and his group developed a modeling package that required a macro language to manage. After creating several only languages that are suitable for one application and not suitable for another, I decided to create an interpreter library that could be combined with other projects. This provided a common parsing package that could be used with a common base language for all projects and applications [10].

Tcl is a multifaceted language. You can use Tcl as a command-scripting language, a powerful multi-platform interpretation language, a rapid prototyping platform, or an interpreter call library for other projects. Tcl's simple syntax makes it quick and easy to write one-time scripts [to replace repetitive sets of commands or graphical user interface (GUI) clicks]. Tcl's modularityand encapsulation capabilities help you develop large projects (more than 100,000 lines of code). Tcl's extensibility makes it easy to use Tcl as the default language for a wide range of projects, from machine control to database applications, electronic design applications, network test devices, and more. Tcl is a free software and commercially supported package. The core Tcl language is maintained by a group of volunteers around the world, and support is available for purchase from Active State, Noumena Corporation, Cygnus, Proc Place, and more. The current central site for information on Tcl/Tk is www.tcl.tk. The source code repository and some binary snapshots are at *http://sourceforge.net/projects/tcl/* [10].

Chapter 4. Work and Results

AMD Xilinx Inc. follows agile model of Software development for Vitis Software. This approach is based on iterative model. Agile methods divide the task into smaller iterations. The part does not require direct long-term planning. The scope of the project is defined at the beginning of the development process. The plan for the number of iterations, the duration and extent of each iteration is well defined depicted in Figure 13 Agile Software Development methodology [7].

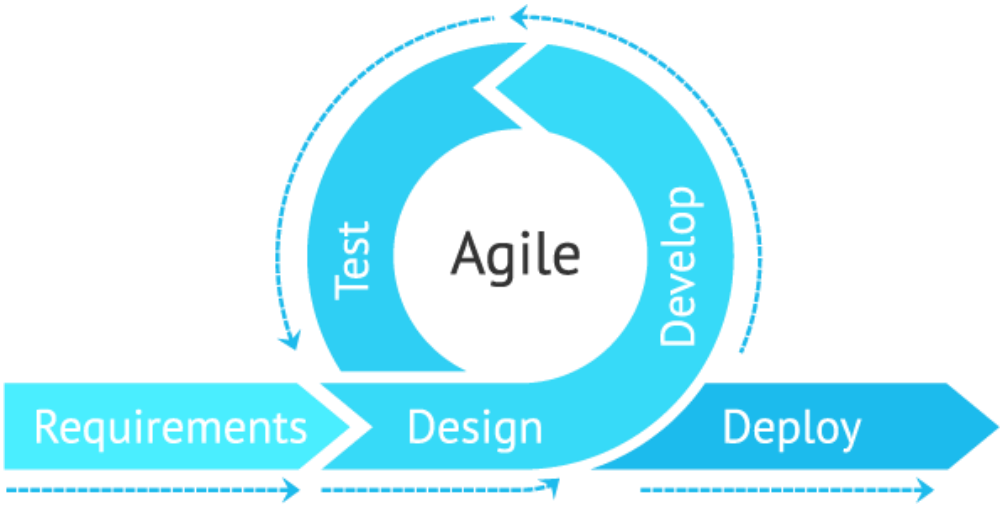

Figure 13 Agile Software Development methodology [14]

- In Jira as shown in Figure 14 Agile epic vs. initiative vs. story A story, also known as a "user story," is a short requirement or requirement created from the end-user's perspective [7].
- An epic is a large unit of work that can be divided into a series of small tasks (called stories).
- Initiatives are a collection of epics working towards a common goal.

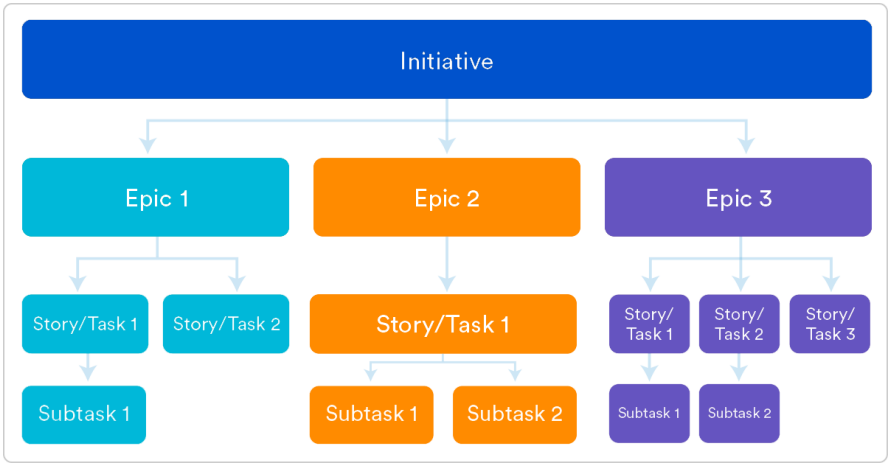

Figure 14 Agile epic vs. initiative vs. story [7]

Figure 15 Work data chart and Figure 16 Contributed Tools shows the amount of work carried out and the backed tools contributed during the course work.

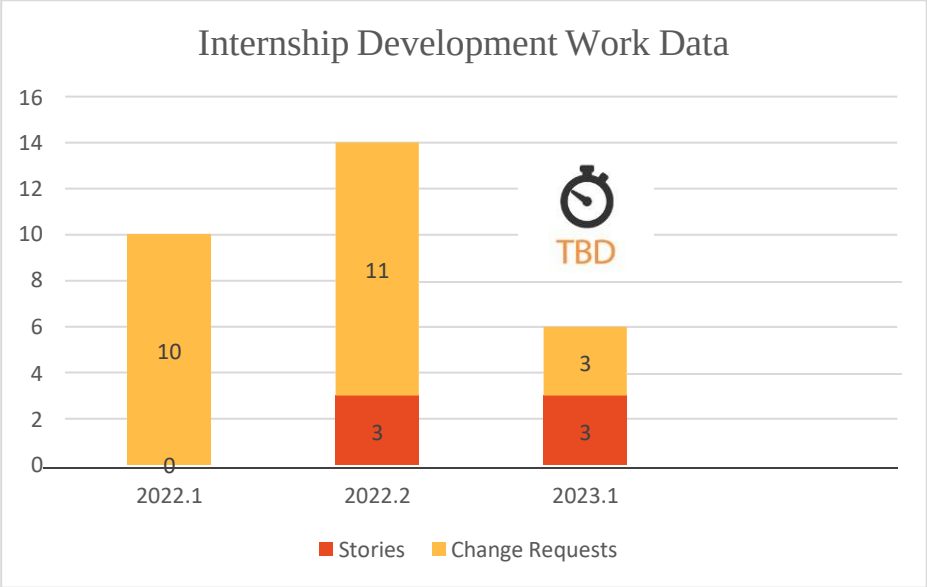

Figure 15 Work data chart

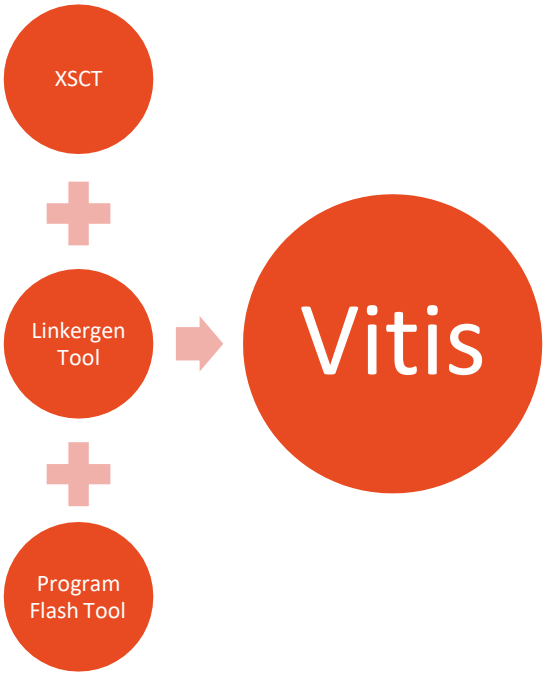

Figure 16 Contributed Tools

4.1. Story to generate device tree

4.1.1. Introduction

Using XSCT to generate device tree is a common task in Vitis platform creation. For now, it needs to use a set of XSCT commands by downloading dtg, creating platform, setting OS, etc. This story will contribute to develop a command to generate Vitis acceleration required dts, dtsi, dtb and dtbo. Figure 17 Device tree Example is a example of how a device tree looks like the Node is the core the peripherals are the leaves and the branches are the data paths.

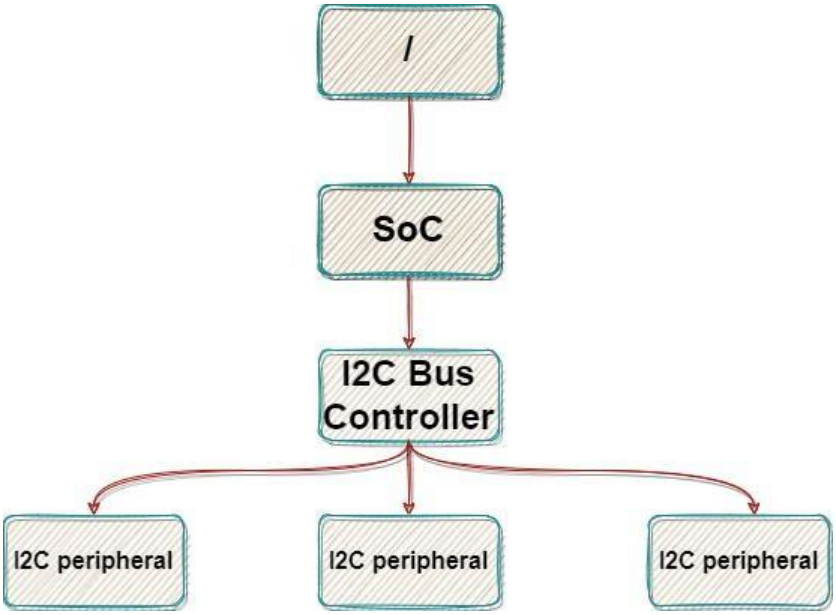

Figure 17 Device tree Example

```
/ {
        SOC {
                i2c controller {
                        i2c peripheral-1 {
                        };
                        i2c peripheral-2 {
                        };
                        i2c peripheral-3 {
                        };
                };
        };
};
```

A device driver is a software that tells the OS and other software running on the hardware how to make communicate with hardware and making things work.

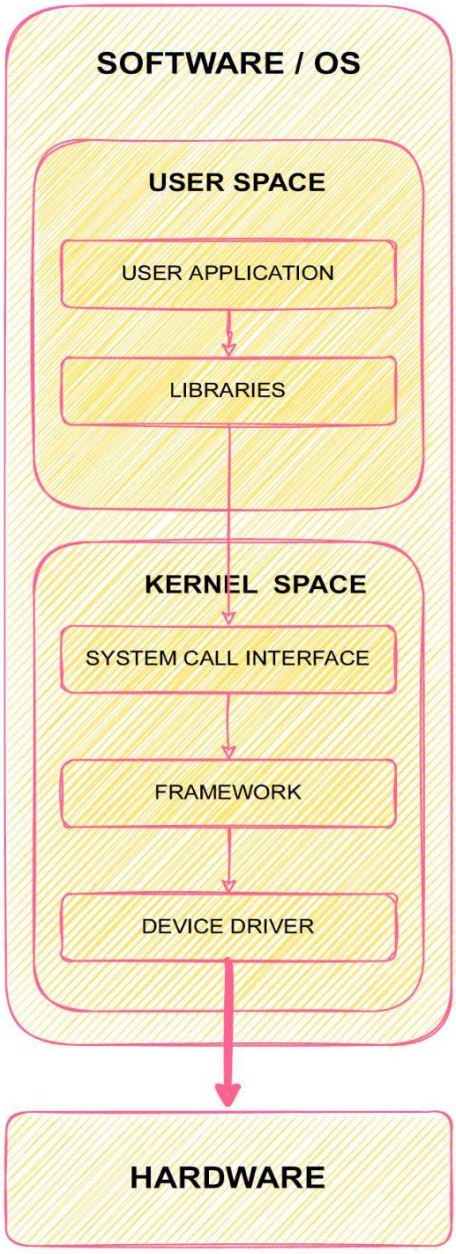

Figure 18 Application Stack

Figure 18 Application Stack shows that device driver communicates with the hardware, but the Linux kernel is same for all the drivers. So only the device tree changes which helps device driver to communicate between hardware and the OS kernel. Device tree is a data structure that tells abouth the hardware on which the software is running. This description readable by an OS like Linux_OS so that it does not need to hard code details of the machine when the hardware changes.

Linux uses the DT basically for platform identification, run-time configuration like bootargs and the device node population.

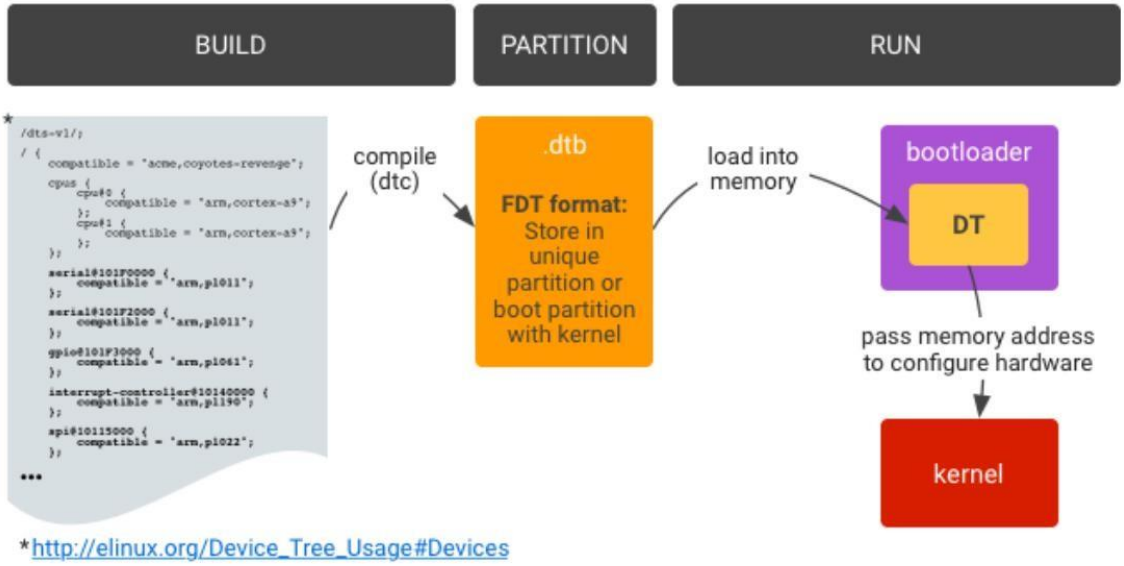

Figure 19 DTB kernel Interface [12]

Figure 19 DTB kernel Interface shows that a device tree is compiled into a device tree blob by the device tree compiler & which is added to the memory of the hardware, with help of this device tree blob the OS kernel talks with the hardware using the device driver.

Each driver or a module in the device tree is defined by the node and all its properties are defined under that node. Based on the driver it can have child nodes or parent node.

Under the root node typically consists of
1) CPUs node details
2) Memory details
3) Have information data like the kernel parameters and the location of an image
4) Aliases
5) Nodes that define the buses details

Figure 20 Device Tree Flows shows the device tree flow sequence diagram with Xilinx specific flow from Vivado design suite to the Linux kernel operating on the Xilinx hardware.

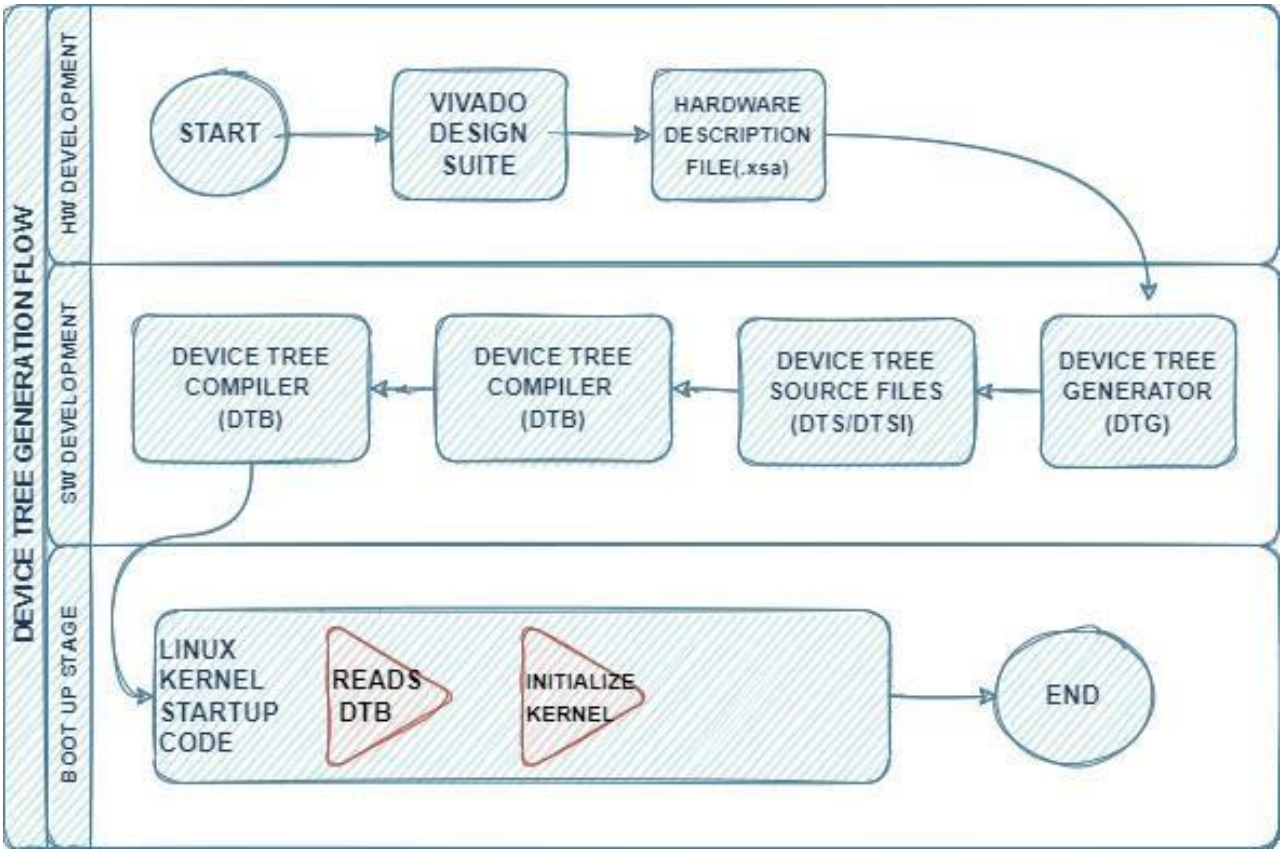

Figure 20 Device Tree Flow

4.1.2. CLI Commands developed

```
createdts
NAME
    createdts - creates device tree.

SYNOPSIS
    createdts [options]
        Create a device tree for the hardware definition file.

OPTIONS
    -platform-name <software-platform name>
        Name of the software platform to be generated.

    -board <board name>
        Board name for device tree to be generated.
        Board names available at <DTG Repo>/device_tree/data/kernel_dtsi.

    -hw <handoff-file>
        Hardware description file to be used to create the device tree.

    -out <output-directory>
```

```
        The directory where the software platform needs to be created.
        Workspace will be default directory, if this option is not specified.

    -local-repo <directory location>
        Location of the directory were bsp for git repo is available.
        Device-tree repo will be cloned from git,
        if this option is not specified.

    -git-url <Git URL>
        Git URL of the dtg repo to be cloned.
        Default repo is https://github.com/Xilinx/device-tree-xlnx.git.

    -git-branch <Git Branch>
        Git branch to be checked out. Master branch selected default.

    -zocl
        Set zocl flag to enable zocl driver support, default set to False.
        zocl should only be used when the designs are PL enabled.
        Only master and xlnx_rel_v2021.2 branch supports zocl property.

    -overlay
        Set overlay flag to enable device-tree overlay support,
        default set to False.

    -compile
        Specify this option to compile the generated dts to create dtb.
        If this option is not specified, users can manually use dts
        to compile dtb.
        For example,
        dtc -I dts -O dtb -o <file_name>.dtb <file_name>.dts
            Compile dts(device tree source) or
            dtsi(device tree source include) files.

        dtc -I dts -O dtb -f <file_name>.dts -o <file_name>.dtb
            Convert dts(device tree source) to dtb(device tree blob).

        dtc -I dtb -O dts -f <file_name>.dtb -o <file_name>.dts
            Convert dtb(device tree blob) to dts(device tree source).

    -update
        Set update flag to enable existing device tree platform
        to update with new xsa.

NOTE
    This command is a shortcut of creating a device tree domain and
    generate the device tree. It clones device-tree repo, creates a
    platform with device_tree as OS, configure and generate the
    platform to create dts.
    -zocl should only be used when the designs are PL enabled.
    Only master and xlnx_rel_v2021.2 branch supports zocl property.
```

```
EXAMPLE
    createdts -hw zcu102.xsa -platform-name my_devicetree
        Create a device tree for the handoff-file with
        Default repo as "https://github.com/Xilinx/device-tree-xlnx.git"
        and deault branch as "master".

    createdts -hw zcu102.xsa -platform-name my_devicetree -git-url <Git URL>
            -git-branch <Git Branch>
        Create a device tree for the handoff-file with
        user repo as repo mentioned in <Git URL>
        and user branch as <Git Branch>.

    createdts -hw zc702.xsa -platform-name my_devicetree
            -local-repo /my_local_git_repo
        Create a device tree for the handoff-file and use the local repo.

    createdts -hw vck190.xsa -platform-name my_devicetree
            -out /device-tree_output_directory
        Create a device tree at the out directory specified
        by device-tre output directory.

    createdts -hw zcu102.xsa -platform-name my_devicetree -overlay
            -zocl -compile
        Create device tree for the handoff-file with overlay and zocl node.
        Compile flag compiles the Devicetree Blob file from the DTS.

    createdts -hw zcu102.xsa -platform-name my_devicetree -board <Board Name>
        Creates a device tree adding board value to the library,
        Board names available at <DTG Repo>/device_tree/data/kernel_dtsi.

    createdts -update -hw newdesign.xsa
        Updates existing device tree platform with new xsa.

RETURNS
    None.
```

4.2. Story to generate linker script from XSA

4.2.1. Introduction:

Vitis IDE has the feature to generate linker script for BareMetal applications. Similarly, we must add the same feature to XSCT which will enable it generate linker script.

A linker script is a text file consisting of a set of linker directives that tells the linker where to use memory and how to use it. Therefore, it accurately reflects the memory resources and memory map of the target device. A linker script controls each link. Such scripts are written in the linker command language. The main purpose of the linker script is to explain how to map sections of

the input file to the output file and to makes decision about the layout of memory. The linker combines the input files into a single output file. The output file and each input file have a special data format called the object file format. Each file is called an object file. Output files are often referred to as executable files. Each object file specifically has a list of sections. The section of the input file is sometimes called the input section. Similarly, the section of the outputfile is the output section.

Each section of the object file has a name and size. Most sections also have a block of data associated with them, called section content. Sections can be marked as loadable. That is, the content must be loaded into memory when the output file is run. Sections with no content may be assignable. That is, the area in memory should be reserved, but nothing should be loaded there (in some cases, this memory should be zero). Sections that are neither loadable nor assignable usually contain some kind of debug information.

Each loadable or assignable output section has two addresses. The first is the VMA or virtual memory address. This is the address that the section has when the output file is executed. The second is the loadable memory address or load memory address. This is the address where the section will be loaded. In most cases the two addresses will be the same. An example of when they might be different is when a data section is loaded into Read Only Memory, and then copied into Random Access Memory when the program starts up (this technique is often used to initialize global variables in a Read Only Memory based system). In this case the Read Only Memory address would be the Loadable Memory Address, and the Random Access Memory address would be the virtual memory address.

4.2.2. Commands Developed:

Figure 21 Build Flow shows the build flow for xilinx devices Figure 22 Linker script GUI Basic Window available under Xilinx > option in the toolbar of the Vitis Unified software shows the available GUI tool to create and edit a linker script which has the memory map view of the memory available in the .xsa design option to change memory of the section and stack & heap size. Figure 23 Linker script GUI Advanced Window is the advance window for the advance userto change the subsection of the linker script to specific size and memory.

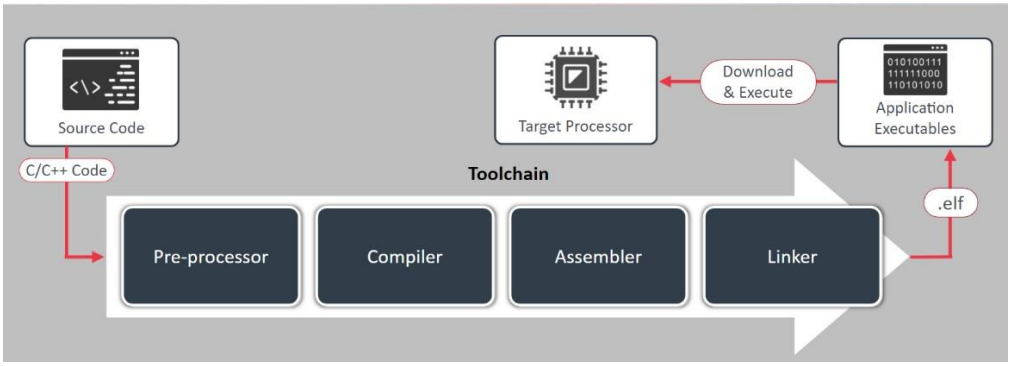

Figure 21 Build

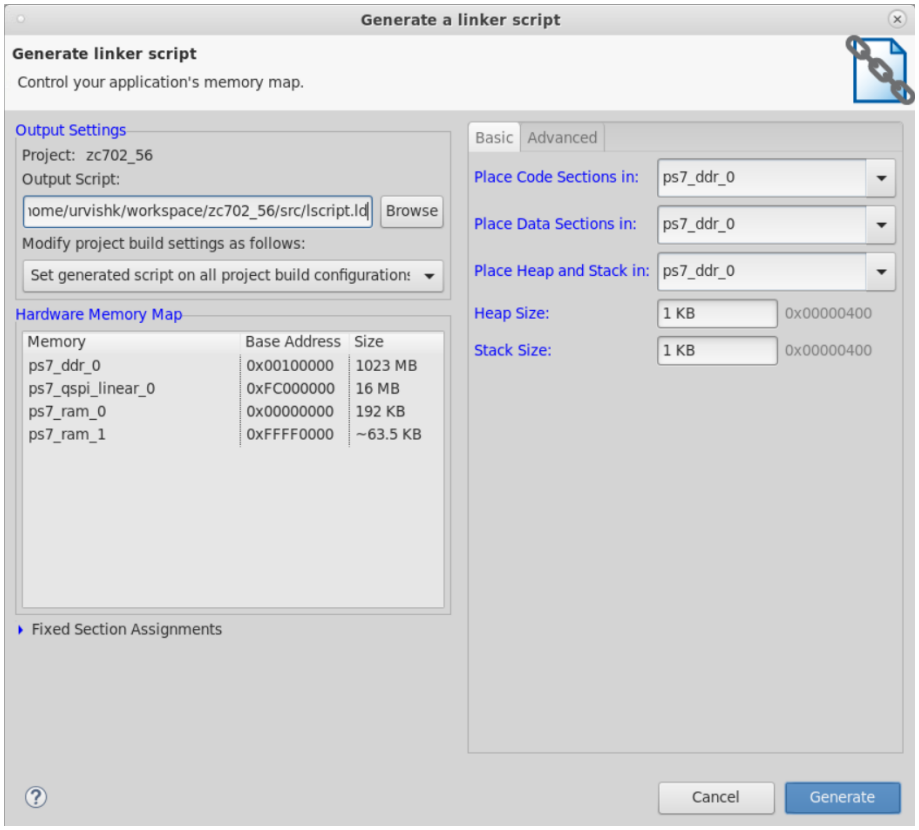

Figure 22 Linker script GUI Basic Window

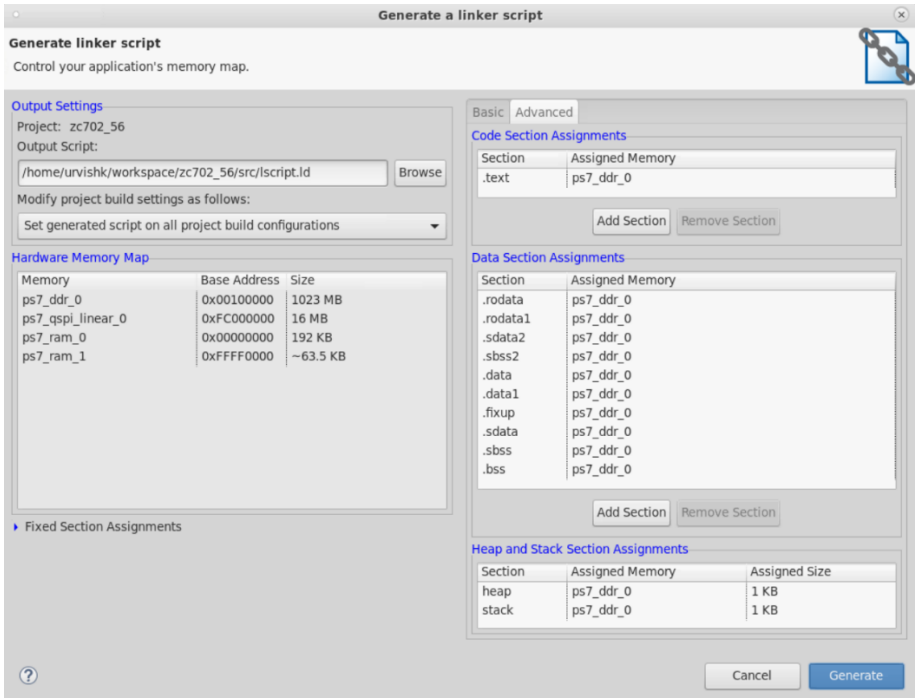

Figure 23 Linker script GUI Advanced Window

lscript

NAME

Lscript – Create linkerscript

SYNOPSIS

lscript <sub-command> [options]

Create a Linkerscript, or perform various other operations on the linkerscript, based on <sub-command> specified.

Following sub-commands are supported.

memory - List of the memories supported by the active domain.

section - Lists and edit the sections available.

def-mem - Returns default memory for the section type.

generate - Generate a linkerscript.

Type "help" followed by "lscript sub-command", or "lscript sub-command" followed by "-help" for more details.

OPTIONS

Depends on the sub-command. Please refer to sub-command help for details.

EXAMPLE

Please refer to sub-command help for details.

RETURNS

Depends on the sub-command. Please refer to sub-command help for details.

lscript memory

NAME

lscript memory - List supported memory.

SYNOPSIS

lscript memory [options]

List of the memories supported by the active domain.

OPTIONS

-supported-mem

Returns supported memory regions for each section.

EXAMPLE

lscript memory

This command returns the list of memories available in the active domain.

lscript memory -supported-mem

Returns the section wise supported memories.

RETURNS

List of the memories supported by the active domain in tabular format.

lscript section

```
NAME
    lscript section - List the sections available.

SYNOPSIS
    lscript section [options]
        List, add, edit the sections available in the active domain.

OPTIONS
    -name <section-name>
        Name of the section to be edited.

    -mem <memory-region>
        Name of the memory region to be used for the section.

    -size <section-size>
        Size of the section.

    -add
        Add a new section.

    -type
        Type of new section to be added.
        Supported types are CODE, DATA, STACK, HEAP.

EXAMPLE
    lscript section
        List of the sections available in the active domain along with the
type,
        size and assigned memory.

    lscript section -name <section-name> -mem <memory-region> -size <section-
size>
        Edit the section-name with memory and size.

    lscript section -mem <memory-region> -size <section-size>
        Edit all the sections with memory and size.

    lscript section -add -name <section-name> -mem <memory-region> -size
<section-size>
            -type <section-type>
        Add a new section with section-name, memory and size.

RETURNS
    List of the sections with corresponding memory and size in tabular format,
    when no options or args are specified.
    Nothing, if a section sucessfully edited or added.
    Error if the section cannot be edited or added.
```

lscript def-mem

```
NAME
    lscript def-mem - Returns the default memory region for the section type.

SYNOPSIS
    lscript def-mem <memory-type>
        Return the default memory-region of the section type.

OPTIONS
    -code
        Return default code memory.

    -data
        Return default data memory.

    -stack
        Return default stack & heap memory.

EXAMPLE
    lscript def-mem -stack
        Return default stack & heap memory-region.

RETURNS
    Return the default memory-region of the section type.
```

lscript generate

```
NAME
    lscript generate - Generate a linkerscript.

SYNOPSIS
    lscript generate [options]
        Generate a linkerscript.

OPTIONS
    -name <linkerscript name>
        Name of the linkerscript file.
        Default linkerscript will be "newlscript.ld" if -name not provided.

    -path <path>
        The directory where the linkerscript needs to be created,
        Default path will be pwd if -path not provided.

EXAMPLE
    lscript generate -name <linkerscript name> -path <path>
        This command generate a linkerscript with the changes at path
provided,
        Otherwise generate default linkerscript with name "newlscript.ld".

RETURN
```

```
xsct% lscript memory
================================================================
NAME                    BASE_ADDR              SIZE
================================================================
psu_ddr_0_MEM_0         0x0                    0x7FF00000
psu_ddr_1_MEM_0         0x800000000            0x80000000
psu_ocm_ram_0_MEM_0     0xFFFC0000             0x40000
psu_qspi_linear_0_MEM_0 0xC0000000              0x20000000


xsct% lscript memory -supported-mem
================================================================================
TYPE                 SUPPORTED MEMORY
================================================================================
code_sections        psu_ddr_0_MEM_0
                     psu_ddr_1_MEM_0
                     psu_ocm_ram_0_MEM_0
                     psu_qspi_linear_0_MEM_0
--------------------------------------------------------------------------------------------------------
data_sections        psu_ddr_0_MEM_0
                     psu_ddr_1_MEM_0
                     psu_ocm_ram_0_MEM_0
--------------------------------------------------------------------------------------------------------
stack_section        psu_ddr_0_MEM_0
                     psu_ddr_1_MEM_0
                     psu_ocm_ram_0_MEM_0
--------------------------------------------------------------------------------------------------------
heap_section         psu_ddr_0_MEM_0
                     psu_ddr_1_MEM_0
                     psu_ocm_ram_0_MEM_0
--------------------------------------------------------------------------------------------------------

xsct% lscript section
=================================================
TYPE       NAME       SIZE       ASSIGNED MEMORY
=================================================
CODE       text       0x0        default
DATA       rodata     0x0        default
DATA       rodata1    0x0        default
DATA       sdata2     0x0        default
DATA       sbss2      0x0        default
DATA       data       0x0        default
DATA       data1      0x0        default
DATA       fixup      0x0        default
DATA       sdata      0x0        default
DATA       sbss       0x0        default
DATA       bss        0x0        default
STACK      stack      0x2000     default
```

```
HEAP       heap       0x2000     default

xsct% lscript section -name text -size 15 -mem psu_ocm_ram_0_MEM_0
xsct% lscript section
================================================================
TYPE       NAME       SIZE       ASSIGNED MEMORY
================================================================
CODE       text       0xF        psu_ocm_ram_0_MEM_0
DATA       rodata     0x0        default
DATA       rodata1    0x0        default
DATA       sdata2     0x0        default
DATA       sbss2      0x0        default
DATA       data       0x0        default
DATA       data1      0x0        default
DATA       fixup      0x0        default
DATA       sdata      0x0        default
DATA       sbss       0x0        default
DATA       bss        0x0        default
STACK      stack      0x2000     default
HEAP       heap       0x2000     default
xsct% lscript section -add -name nirma_university -size 15 -mem
psu_ddr_1_MEM_0

xsct% lscript section
================================================================
TYPE       NAME       SIZE       ASSIGNED MEMORY
================================================================
CODE       text       0xF        psu_ocm_ram_0_MEM_0
CODE       nirma_university 0xF       psu_ddr_1_MEM_0
DATA       rodata     0x0        default
DATA       rodata1    0x0        default
DATA       sdata2     0x0        default
DATA       sbss2      0x0        default
DATA       data       0x0        default
DATA       data1      0x0        default
DATA       fixup      0x0        default
DATA       sdata      0x0        default
DATA       sbss       0x0        default
DATA       bss        0x0        default
STACK      stack      0x2000     default
HEAP       heap       0x2000     default

xsct% lscript section -size 14 -mem psu_ddr_1_MEM_0
xsct% lscript section
============================================================
TYPE       NAME       SIZE       ASSIGNED MEMORY
============================================================
CODE       text       0xE        psu_ddr_1_MEM_0
CODE       nirma_university 0xE       psu_ddr_1_MEM_0
DATA       rodata     0xE        psu_ddr_1_MEM_0
```

```
DATA        rodata1     0xE         psu_ddr_1_MEM_0
DATA        sdata2      0xE         psu_ddr_1_MEM_0
DATA        sbss2       0xE         psu_ddr_1_MEM_0
DATA        data        0xE         psu_ddr_1_MEM_0
DATA        data1       0xE         psu_ddr_1_MEM_0
DATA        fixup       0xE         psu_ddr_1_MEM_0
DATA        sdata       0xE         psu_ddr_1_MEM_0
DATA        sbss        0xE         psu_ddr_1_MEM_0
DATA        bss         0xE         psu_ddr_1_MEM_0
STACK       stack       0xE         psu_ddr_1_MEM_0
HEAP        heap        0xE         psu_ddr_1_MEM_0
```

```
[urvishk@xhdurvishk42x ~ ] xsdb

****** Xilinx System Debugger (XSDB) v2023.1.0
  **** Build date : Dec  4 2022-18:42:01
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.


xsdb% help
Available Help Categories

breakpoints    - Target Breakpoints/Watchpoints.
connections    - Target Connection Management.
device         - Device Configuration System.
download       - Target Download FPGA/BINARY.
ipi            - IPI commands to Versal PMC.
jtag           - JTAG Access.
memory         - Target Memory.
miscellaneous  - Miscellaneous.
registers      - Target Registers.
reset          - Target Reset.
running        - Program Execution.
sdk            -
stapl          - STAPL Operations.
streams        - Jtag UART.
svf            - SVF Operations.
tfile          - Target File System.

Type "help" followed by above "category" for more details or
help" followed by the keyword "commands" to list all the commands
```

Figure 24 Basic startup commands for XSDB

Chapter 5. Conclusion & Future Scope

This Internship aimed to develop embedded software for the Vitis Unified software by Xilinx Inc. Work focuses on commands for Xilinx Software Command-Line Tool (XSCT). XSCT is an scriptable CLI to Xilinx Vitis.

The work that has been done in the software has given me a better understanding about the components of the Vitis Unified Software and products and technologies that Xilinx has been working. There are many customers who are using the Xilinx devices to develop their applications and now Xilinx has moved from a FPGA company to a Solution company that has created many opportunities and paths that can be explored. The product profile of Xilinx has a solution as per the needs of the customers. With Increase in the demands and users there are future scope, and many features can be added to Vitis that can improve its functionality. Contribution has been done for developing commands for the XSCT i.e., Vitis CLI depending on the customer requirements and company product roadmaps and in area of Vitis Software backendtools. Backend tools consist of various embedded development tools like debuggers, bootable image creators etc.

As Future scope I will be working for the remaining period of the internship to support Xilinx next generation Versal device which will be launching in future. Along with adding a mechanismto platform create command to add properties to the platform that can be used by the internal team. For 2023.1 release work will be carried out on to provide part-specific AIE hardware metadata in XSA - Vitis_XSCT.

References

[1] Xilinx. 2022. Xilinx - Adaptable. Intelligent.. [online] Available at: <https://www.xilinx.com/> [Accessed 19 july 2022].

[2] "Xilinx Developer Forum (XDF) 2019 Silicon Valley", 2019 [Online]. Available: https://www.xilinx.com/content/dam/xilinx/imgs/press/media-kits/vitis-pr-release.pdf. [Accessed: 05- Aug- 2022]

[3] "Vitis Unified Software Platform Documentation", Xilinx.com, 2021. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf. [Accessed: 05- Aug- 2022]

[4] "Xilinx Software Command-Line Tool (XSCT) Reference Guide UG1208 (v2018.2) June 6, 2018", Xilinx.com, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1208-xsct-reference-guide.pdf. [Accessed: 05- Aug- 2022]

[5] "TCF - Eclipsepedia", Wiki.eclipse.org, 2021. [Online]. Available: https://wiki.eclipse.org/TCF. [Accessed: 05- Dec- 2022]

[6] "Gradle Build Tool", Gradle, 2021. [Online]. Available: https://gradle.org/. [Accessed: 05- Dec- 2022]

[7] "Jira | Issue & Project Tracking Software | Atlassian", Atlassian, 2021. [Online]. Available: https://www.atlassian.com/software/jira. [Accessed: 05- Dec- 2022]

[8] "QEMU documentation - QEMU", Qemu.org, 2021. [Online]. Available: https://www.qemu.org/documentation/. [Accessed: 05- Aug- 2022]

[9] D.Schmidt and S. Huston, C++ network programming. Boston: Addison-Wesley, 2002.

[10] "Tcl Developer Site", Tcl.tk, 2021. [Online]. Available: https://www.tcl.tk/. [Accessed: 05- September- 2022]

[11] "Helix Visual Client (P4V) | Perforce", Perforce.com, 2021. [Online]. Available: https://www.perforce.com/downloads/helix-visual-client-p4v. [Accessed: 05- Septmeber- 2022]

[12] "The Most Advanced Debugger for HPC Computing | TotalView by Perforce", Totalview.io, 2021. [Online]. Available: https://totalview.io/. [Accessed: 05- Oct- 2022]

[13] Android Open Source Project. 2022. Device Tree Overlays | Android Open Source Project. [online] Available at: https://source.android.com/devices/architecture/dto [Accessed 13 November 2022]

[14] "Agile Advantages for Software Development and Your Business", devcom.com, 2022. [Online]. Available: https://devcom.com/tech-blog/agile-advantages-for-business/. [Accessed: 13- November- 2022]