



INSTITUTO TECNOLÓGICO DE IZTAPALAPA

ING. SISTEMAS COMPUTACIONALES

PROYECTO FINAL

TEMA:

CIRCT – “CIRCUIT IR COMPILERS AND TOOLS”

MAESTRO:

ABIEL TOMAS PARRA HERNÁNDEZ

PRESENTA:

ARREDONDO FLORES ALEXA KETZALI	181080005	33.33%
BOHORQUEZ LOPEZ MIGUEL ANGEL	181080006	33.33%
RODRIGUEZ GARCIA JESUS	181080027	33.33%

INDICE

Resumen.....	3
Abstract.....	3
Introducción.....	4
Objetivos	5
Objetivos Especificos	5
Justificación.....	6
Marco Teórico.....	7
Compilador	7
LLVM.....	8
MLIR.....	9
CIRCT.....	10
Aceleradores.....	10
Metodología	12
Desarrollo e Implementación	14
Windows.....	14
Linux – Debian.....	19
Resultados	22
Conclusión.....	23
Referencias.....	24

RESUMEN

En los últimos años la inclinación por la arquitectura de computadoras ha abierto el paso a grandes incógnitas con respecto a la fabricación de los chips complejos con el que se realizan las computadoras, en base a esto surgen dos cuestiones centrales, las cuales son. ¿Cómo se diseñan? y ¿cómo se programan? Por eso mismo buscamos demostrar cómo las herramientas que diseñan y dan la solución a estos problemas se centra en el LLVM / LMIR para la expresión de estas abstracciones y resolver los problemas de diseño a posteriori.

PALABRAS CLAVE

LLVM, LMIR, chips, herramientas

ABSTRACT

In recent years, the inclination for computer architecture has opened the way to great unknowns regarding the manufacture of complex chips with which computers are made, based on this two central questions arise, which are. How are they designed? and how are they programmed? For this reason, we seek to demonstrate how the tools that design and provide the solution to these problems focus on the LLVM / LMIR for the expression of these abstractions and solve the later design problems.

KEYWORDS:

LLVM, LMIR, chips, tools

INTRODUCCIÓN

El proyecto Circuit IR Compilers and Tools (CIRCT) busca aplicar MLIR y la metodología de desarrollo LLVM al dominio de las herramientas de diseño de hardware, tiene herramientas patentadas y de código abierto lo cual permite ser usadas por la comunidad interesada en este desarrollo, sin embargo, aún cuenta con problemas de usabilidad ya que no fueron diseñadas sobre una plataforma común.

Las nuevas tendencias en la arquitectura de PC'S han dado lugar a 2 inconvenientes centrales.

1. ¿Cómo diseñamos sistemas en chip complejos y heterogéneos que mezclan componentes de uso general y especializados?
2. ¿Cómo los programamos?

Creemos que los instrumentos de diseño que representan y manipulan una vasta diversidad de abstracciones son primordiales para solucionar dichos inconvenientes. Este plan se concentra en la utilización de LLVM / MLIR para manifestar estas abstracciones y edificar flujos de código abierto utilizables basados en aquellas abstracciones para solucionar los inconvenientes de diseño de la siguiente década.

Planear una vasta pluralidad de aceleradores de una vasta gama de lenguajes de diseño es un reto fundamental. Los lenguajes de diseño personales se asocian típicamente con sus propios dialectos, que principalmente se disminuyen a dialectos usuales o comunes. La aplicación de optimizaciones en dialectos usuales posibilita compartir el esfuerzo de desarrollar y conservar optimizaciones entre los lenguajes de diseño y las arquitecturas de destino.

OBJETIVOS

Identificar los beneficios que conlleva implementar “CIRCT” en el desarrollo de las herramientas de diseño de hardware abierto, con el fin de contextualizar a las personas interesadas en implementar dicho proyecto.

OBJETIVOS ESPECIFICOS

- Conocer a fondo MLIR y la metodología de desarrollo LLVM para ser aplicadas en el proyecto “CIRCT”.
- Identificar los beneficios de implementar el proyecto “CIRCT” para el dominio de las herramientas de diseño de hardware.
- Realizar un análisis sobre el estado actual del proyecto “CIRCT” / Circuit IR Compilers and Tools.
- Edificar un estudio sobre los beneficios de implementar el proyecto “CIRT” y saber con qué otras herramientas podemos utilizarlo.

JUSTIFICACIÓN

Implementar el proyecto “CIRCT” en la EDA (Electronic design automation) o bien en español “Automatización de diseño electrónico”, estimula el crecimiento de herramientas de diseño de hardware, el objetivo de estas herramientas de diseño es la aceleración de procesos. Cuando hablamos de aceleración, nos estamos refiriendo a aumentar la velocidad en la que recorremos una distancia en un determinado espacio de tiempo. En el mundo del hardware llamamos acelerador a todo tipo de unidad que realiza un trabajo concreto de forma más rápida y eficiente que un procesador complejo como por ejemplo una CPU o GPU.

En el desarrollo de estas herramientas de diseño podemos aplicar la metodología de desarrollo LLVM, también, será necesario aplicar MLIR para el correcto dominio de dichas herramientas de diseño de hardware.

En este sentido, la implementación del proyecto “CIRCT” en las herramientas de diseño de hardware ayuda a los desarrolladores en el desarrollo y creación de “aceleradores” mucho más rápidos y eficientes que los que ya existen.

MARCO TEÓRICO

Compilador

Un compilador es un traductor que se utiliza para convertir un lenguaje de programación de alto nivel en un lenguaje de programación de bajo nivel. Convierte todo el programa en una sesión e informa de los errores detectados después de la conversión.

Características principales:

- Necesita tiempo para hacer su trabajo, ya que traduce el código de alto nivel al código de nivel inferior de una vez y luego lo guarda en la memoria.
- Depende del procesador y de la plataforma. Se ha abordado con nombres alternativos como los siguientes: compilador especial, compilador cruzado y compilador de fuente a fuente.

Visto desde otro punto de vista, un compilador es una herramienta que traduce software escrito en un idioma a otro idioma. Para traducir texto de un idioma a otro, la herramienta debe comprender tanto la forma o la sintaxis como el contenido o el significado del idioma de entrada. Necesita comprender las reglas que gobiernan la sintaxis y el significado en el idioma de salida. Finalmente, necesita un esquema para mapear contenido del idioma de origen al idioma de destino.

La estructura de un compilador típico se deriva de estas simples observaciones. El compilador tiene una interfaz para manejar el lenguaje fuente, tiene una estructura formal para representar el programa en un nivel intermedio, forma cuyo significado es en gran medida independiente de cualquier idioma a mejorar la traducción, un compilador a menudo incluye un optimizador que analiza y reescribe esa forma intermedia.

Los principios fundamentales de la compilación: Los compiladores son objetos grandes, complejos y diseñados. Mientras que muchos de los problemas están en el diseño del compilador son susceptibles de múltiples soluciones e interpretaciones, hay dos principios fundamentales que un escritor de compiladores debe tener en cuenta en todo momento.

El primer principio es inviolable:

“El compilador debe preservar el significado del programa que se está

El segundo principio que debe observar un compilador es práctico:

“El compilador debe mejorar el programa de entrada de alguna manera discernible.”

LLVM

El proyecto LLVM tiene múltiples componentes. El núcleo del proyecto en sí mismo se llama "LLVM". Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para procesar representaciones intermedias y convertirlas en archivos de objeto. Las herramientas incluyen un ensamblador, un desensamblador, un analizador de código de bits y un optimizador de código de bits. También contiene pruebas de regresión básicas.

Los lenguajes similares a C usan la interfaz de Clang . Este componente compila código C, C ++, Objective C y Objective C ++ en código de bits LLVM y, desde allí, en archivos de objeto, utilizando LLVM.

Hay muchos proyectos diferentes que componen LLVM. La primera pieza es la suite LLVM. Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para usar LLVM. Contiene un ensamblador, desensamblador, analizador de código de bits y optimizador de código de bits. También contiene pruebas de regresión básicas que se pueden utilizar para probar las herramientas LLVM y la interfaz de Clang.

La segunda pieza es la parte delantera de Clang. Este componente compila código C, C ++, Objective C y Objective C ++ en código de bits LLVM. Clang generalmente usa bibliotecas LLVM para optimizar el código de bits y emitir código de máquina. LLVM es totalmente compatible con el formato de archivo de objeto COFF, que es compatible con todas las demás cadenas de herramientas de Windows existentes.

MLIR

El proyecto "Multi-Level Intermediate Representation (MLIR)" es un enfoque novedoso para construir una infraestructura de compilador extensible y reutilizable. MLIR tiene como objetivo abordar la fragmentación del software, mejorar la compilación para hardware heterogéneo, reducir significativamente el costo de construir compiladores específicos de dominio y ayudar a conectar compiladores existentes.

¿Para qué sirve MLIR?

MLIR está diseñado para ser un IR híbrido que puede soportar múltiples requisitos diferentes en una infraestructura unificada. Por ejemplo, esto incluye:

- La capacidad de representar gráficos de flujo de datos (como en TensorFlow),
- Las optimizaciones y transformaciones se realizan normalmente en tales gráficos (por ejemplo, en Grappler).
- Capacidad para albergar optimizaciones de bucle de estilo informático de alto rendimiento en los núcleos (fusión, intercambio de bucle, ordenamiento en teselas, etc.) y para transformar diseños de memoria de datos.
- Transformaciones de "reducción" de generación de código, como inserción DMA, gestión explícita de caché, ordenamiento en teselas de memoria y vectorización para arquitecturas de registro 1D y 2D.

- Capacidad para representar operaciones específicas de un objetivo, por ejemplo, operaciones de alto nivel específicas de un acelerador.
- Cuantización y otras transformaciones de gráficos realizadas en un gráfico de aprendizaje profundo.

MLIR es un IR común que también admite operaciones específicas de hardware. Por lo tanto, cualquier inversión en la infraestructura que rodea a MLIR (por ejemplo, el compilador transmite ese trabajo) debería producir buenos rendimientos; muchos objetivos pueden utilizar esa infraestructura y se beneficiarán de ella.

El marco MLIR fomenta las mejores prácticas existentes, por ejemplo, escribir y mantener una especificación de IR, construir un verificador de IR, brindar la capacidad de volcar y analizar archivos MLIR en texto, escribir pruebas unitarias extensas con la herramienta FileCheck y construir la infraestructura como un conjunto de bibliotecas modulares que se pueden combinar de nuevas formas.

CIRCT

“CIRCT” son las siglas de “Circuit IR Compilers and Tools”. También se podría interpretar de forma recursiva como “Compilador y herramientas CIRCT IR”. La T se puede expandir selectivamente como Herramienta, Traductor, Equipo, Tecnología, Destino, Árbol, Tipo.

Tiene como objetivo principal aplicar MLIR y la metodología de desarrollo LLVM al dominio de las herramientas de diseño de hardware para la creación de nuevos aceleradores.

Aceleradores

En el mundo del hardware llamamos acelerador a todo tipo de unidad que realiza un trabajo concreto de forma más rápida y eficiente que un procesador complejo: CPU, GPU, etc.

Todos los aceleradores cumplen las siguientes dos condiciones:

- Ocupa un espacio en el hardware que es varias órdenes de magnitud más pequeño en área en comparación con un procesador complejo.
- Su consumo energético, a la hora de realizar dicha tarea, es minúsculo en comparación con una CPU.

Es decir, los aceleradores ganan por mucho en la relación potencia/área y potencia/consumo a cualquier procesador de propósito general. De ahí a que sean utilizados en todo tipo de procesadores.

Estos aceleradores no solo están enfocados en el rendimiento en las computadoras de escritorio o laptops, también se enfocan en la implementación de Inteligencia Artificial, se están centrando en la inferencia y en integrarse en dispositivos finales como vehículos autónomos, cámaras, robótica y aviones teledirigidos para impulsar el procesamiento de datos y la toma de decisiones en tiempo real.

METODOLOGÍA

La metodología que se utilizará será la metodología ágil (SCRUM). Una de las características que la diferencia de otros enfoques de gestión de proyectos es el nivel de propiedad y responsabilidad que cada uno brinda a los miembros del equipo.

La metodología ágil tiene cuatro valores importantes:

- Mayor enfoque en individuos e interacciones que procesos y herramientas
- El software funcionando es más importante que una documentación extensa.
- La colaboración con el cliente es más importante que la negociación contractual.
- Responder al cambio en lugar de seguir ciegamente un plan.

Beneficios de la metodología ágil:

- Se establecen prioridades flexibles.
- Se empieza a entregar antes.
- Costes y plazos conocidos.
- Mejora la calidad final.
- Mayor transparencia.

“Agile” sigue un proceso iterativo en el que los proyectos se dividen en sprints de menor duración. A diferencia del enfoque tradicional, por ejemplo, se gasta menos tiempo en la planificación y la priorización por adelantado, ya que el enfoque ágil es más flexible en cuanto a cambios respecto a los requerimientos iniciales.

En la metodología ágil, cada miembro del equipo comparte la propiedad del proyecto. Cada uno de ellos juega un papel activo para completar el sprint dentro

del tiempo estimado. A diferencia de otros métodos, todos los involucrados en el proyecto pueden ver fácilmente el progreso desde el principio hasta el final.

La metodología ágil disfruta de una gran aceptación, se ha convertido en la primera opción para muchos gestores de proyectos porque pueden responder a las solicitudes requeridas a medida que validan cada iteración, esto les permite entregar un producto o servicio de alta calidad dentro del plazo establecido.

También nos hemos basado en la consultora The Standish Group, la cual, periódicamente publica el “CHAOS Report”, donde ilustran con datos el éxito o fracaso de los proyectos según la metodología utilizada, si comparamos ambas metodologías en un proyecto, vemos como según la metodología elegida tendremos mayor o menor probabilidad de éxito.

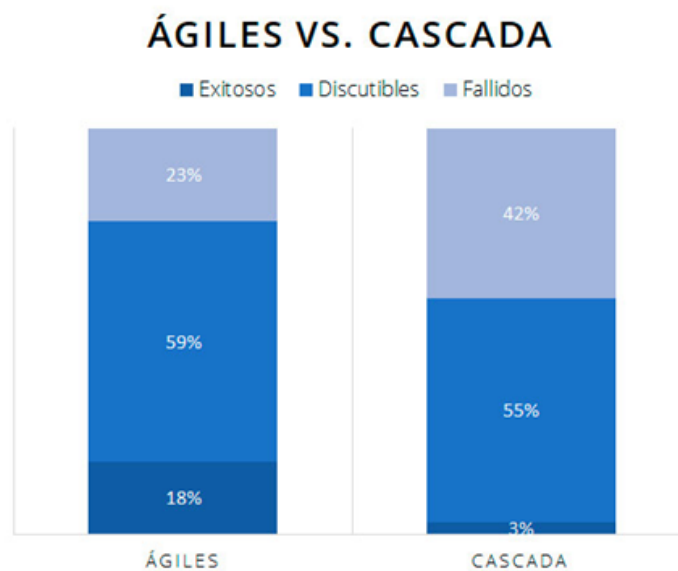


Figura 1. Gestión ágil vs gestión tradicional de proyectos. Fuente (Escuela de negocios, 2019).

DESARROLLO E IMPLEMENTACIÓN

Antes de construir cualquier cosa sobre el proyecto extraído del repositorio, hay que preparar el entorno de trabajo dependiendo del SO que estemos utilizando, en este caso usamos dos, Windows y Linux, la razón del cambio se debe a problemas con el compilador de C/C++ en Windows.

Windows

- Instalación de minGW como compilador de C/C++.

mingw-w64
GCC for Windows 64 & 32 bits

[Downloads](#) •
 [Documentation](#) •
 [Support](#) •
 [Contribute](#) •
 [Donate](#)

Downloads

The heart of the Mingw-w64 project is headers and support libraries to run the output of GCC on Windows. Since Mingw-w64 is neither the home of GCC nor of binutils, several sets of installation packages which combine them are available.

In addition, the sources are available but most people will want to grab binaries directly..

Pre-built toolchains and packages

	Version	Host	GCC / Mingw-w64 Version	Languages	Additional Software in Package Manager
Arch Linux	Arch Linux		8.2.0/5.0.4	Ada, C, C++, Fortran, Obj-C, Obj-C++	305, full list: Show
Cygwin	Rolling	Windows	5.4.0/5.0.2	Ada, C, C++, Fortran, Obj-C	5 (bzip2, libgcrypt, libgpg-error, minizip, xz, zlib)
Debian	Debian 7 (Wheezy)		4.6.3/2.0.3	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (gdb, nsis)
	Debian 8 (Jessie)		4.9.1/3.2.0		9 (gdb, libassuan, libgcrypt, libgpg-error, libksba, libnpt, nsis, win-iconv, zlib)
	Debian 9 (Stretch)		6.3.0/5.0.0		
	Debian 10 (Buster)		8.3.0/6.0.0		
Fedora	Fedora 19		4.8.1/7	Ada, C, C++, Fortran, Obj-C, Obj-C++	149, full list: Show
MacPorts	Rolling	macOS	8.2.0/5.0.4	C, C++, Fortran, Obj-C, Obj-C++	1 (nsis)
MingW-w64-builds	Rolling	Windows	7.2.0/5.0.3	C, C++, Fortran	4 (gdb, libiconv, python, zlib)
Msys2	Rolling	Windows	9.2.0/trunk	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	many
	12.04 Precise Pangolin		4.6.3/2.0.1	Ada, C, C++, Fortran, Obj-C, Obj-C++	2 (nsis, gdb)
	14.04 Trusty Tahr		4.8.2/3.1.0		
	14.10 Utopic Unicorn		4.9.1/3.1.0		

Figura 2. Instalación de MinGW.

- Instalación del gestor de paquetes para Windows, Chocolatey, el cual servirá para instalar ninja-build.

```

Administrador: Windows Power
PS C:\Users\jessg> choco install ninja-build
Chocolatey v0.10.15
2 validations performed. 1 success(es), 1 warning(s), and 0 error(s).

Validation Warnings:
- A pending system reboot request has been detected, however, this is
  being ignored due to the current Chocolatey configuration. If you
  want to halt when this occurs, then either set the global feature
  using:
    choco feature enable -name=exitOnRebootDetected
  or pass the option --exit-when-reboot-detected.

Installing the following packages:
ninja-build
By installing you accept licenses for the packages.
ninja-build not installed. The package was not found with the source(s) listed.
Source(s): 'https://chocolatey.org/api/v2/'
NOTE: When you specify explicit sources, it overrides default sources.
If the package version is a prerelease and you didn't specify '--pre',
the package may not be found.
Please see https://chocolatey.org/docs/troubleshooting for more
assistance.

Chocolatey installed 0/1 packages. 1 packages failed.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Failures
- ninja-build - ninja-build not installed. The package was not found with the source(s) listed.
Source(s): 'https://chocolatey.org/api/v2/'
NOTE: When you specify explicit sources, it overrides default sources.
If the package version is a prerelease and you didn't specify '--pre',
the package may not be found.
Please see https://chocolatey.org/docs/troubleshooting for more
assistance.
PS C:\Users\jessg>

Chocolatey installed 0/0 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Did you know the proceeds of Pro (and some proceeds from other
licensed editions) go into bettering the community infrastructure?
Your support ensures an active community, keeps Chocolatey tip top,
plus it nets you some awesome features!
https://chocolatey.org/compare
Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\chocolatey'.

```

Figura 3. Instalación de Chocolatey.

```

Windows PowerShell
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1052]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\ketza>choco install ninja
Chocolatey v0.10.15
Chocolatey detected you are not running from an elevated command shell
(cmd/powershell).

You may experience errors - many functions/packages
require admin rights. Only advanced users should run choco w/out an
elevated shell. When you open the command shell, you should ensure
that you do so with "Run as Administrator" selected. If you are
attempting to use Chocolatey in a non-administrator setting, you
must select a different location other than the default install
location. See
https://chocolatey.org/install#non-administrative-install for details.

```

```

Do you want to continue?([Y]es/[N]o): y

Installing the following packages:
ninja
By installing you accept licenses for the packages.
ninja not installed. An error occurred during installation:
Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\lib\ninja\legal'.
ninja package files install completed. Performing other installation steps.
This is try 1/3. Retrying after 300 milliseconds.
Error converted to warning:
Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\chocolatey'.
This is try 2/3. Retrying after 400 milliseconds.
Error converted to warning:
Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\chocolatey'.
Maximum tries of 3 reached. Throwing error.
Cannot create directory "C:\ProgramData\chocolatey\chocolatey". Error was:
System.UnauthorizedAccessException: Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\chocolatey'.
    en System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
    en System.IO.Directory.InternalCreateDirectory(String fullPath, String path, Object dirSecurityObj, Boolean checkHost)
    en System.IO.Directory.InternalCreateDirectoryHelper(String path, Boolean checkHost)
    en chocolatey.infrastructure.filesystem.DotNetFileSystem.<c__DisplayClass64.<create_directory>b__63()
    en chocolatey.infrastructure.tolerance.FaultTolerance.<c__DisplayClass1.<retry>b__0()
    en chocolatey.infrastructure.tolerance.FaultTolerance.retry[T](Int32 numberOfTries, Func`1 function, Int32 waitDurationMilliseconds, Int32 increaseRetryByMilliseconds, Boolean isSilent)
    en chocolatey.infrastructure.filesystem.DotNetFileSystem.create_directory(String directoryPath)
    en chocolatey.infrastructure.filesystem.DotNetFileSystem.create_directory_if_not_exists(String directoryPath, Boolean ignoreError)

Chocolatey installed 0/0 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Did you know the proceeds of Pro (and some proceeds from other
licensed editions) go into bettering the community infrastructure?
Your support ensures an active community, keeps Chocolatey tip top,
plus it nets you some awesome features!
https://chocolatey.org/compare
Acceso denegado a la ruta de acceso 'C:\ProgramData\chocolatey\chocolatey'.

```

Figura 4. Errores de instalación.

Algunos errores que pueden ocurrir durante la instalación de las dependencias se muestran en las fotos anteriores, esto se puede deber a los permisos de administrador o de usuario en nuestra computadora, rutas de acceso inválidas o fallas en los comandos.

- Instalación de Cmake gestor de paquetes.

CMake About ▾ Resources ▾ Developer Resources ▾ Download 🔍

Build the source distributions, unpack them into a zip or tar, and follow the instructions in README file at the top of the source tree. See also the [CMake 3.20 Release Notes](#).

Source distributions:

Platform	Files
Unix/Linux Source (has \n line feeds)	cmake-3.20.5.tar.gz
Windows Source (has \r\n line feeds)	cmake-3.20.5.zip

Binary distributions:

Platform	Files
Windows x64 Installer: Installer tool has changed. Uninstall CMake 3.4 or lower first!	cmake-3.20.5-windows-x86_64.msi
Windows x64 ZIP	cmake-3.20.5-windows-x86_64.zip
Windows i386 Installer: Installer tool has changed. Uninstall CMake 3.4 or lower first!	cmake-3.20.5-windows-i386.msi
Windows i386 ZIP	cmake-3.20.5-windows-i386.zip
macOS 10.13 or later	cmake-3.20.5-macos-universal.dmg
	cmake-3.20.5-macos-universal.tar.gz
macOS 10.10 or later	cmake-3.20.5-macos10.10-universal.dmg
	cmake-3.20.5-macos10.10-universal.tar.gz
Linux x86_64	cmake-3.20.5-linux-x86_64.sh
	cmake-3.20.5-linux-x86_64.tar.gz
Linux aarch64	cmake-3.20.5-linux-aarch64.sh
	cmake-3.20.5-linux-aarch64.tar.gz

Summary files:

Figura 5. Instalación de Cmake.

- Asignándolo al path como variable de entorno

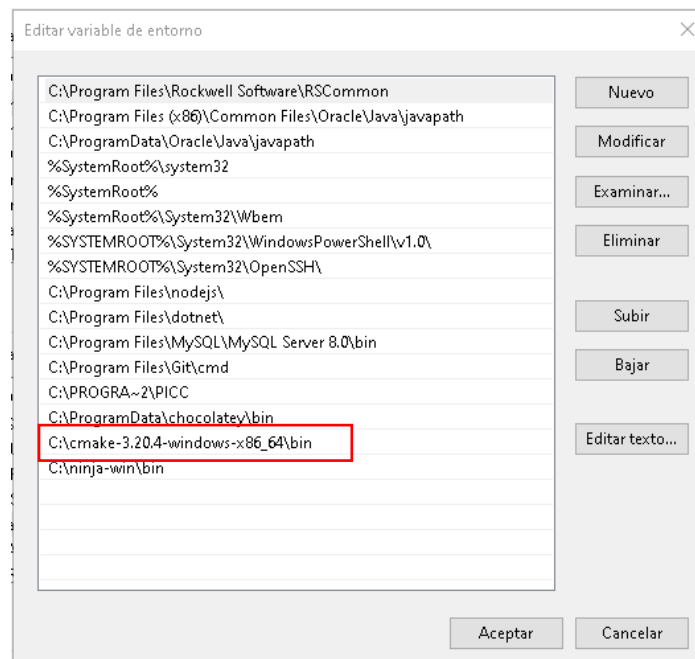


Figura 6. Agregando cmake a la variable de entorno Path.

- Clonamos el repositorio directamente de GitHub

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\ketza> git clone git@github.com:llvm/circt.git
Cloning into 'circt'...
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
PS C:\Users\ketza> |
```

Figura 7. Clonación del repositorio.

- Creamos una carpeta “llvm” para la construcción del test dentro del repositorio corremos la configuración de prueba para verificar que instalamos las dependencias de LLVM y MLIR.

```
remote: Enumerating objects: 27032, done.
remote: Counting objects: 100% (4068/4068), done.
remote: Compressing objects: 100% (1478/1478), done.
remote: Total 27032 (delta 2981), reused 3501 (delta 2558), pack-reused 22964
Receiving objects: 100% (27032/27032), 6.93 MiB | 2.40 MiB/s, done.
Resolving deltas: 100% (19234/19234), done.
PS D:\Escuela\LenguajesYAutomatasII> cd circt
PS D:\Escuela\LenguajesYAutomatasII\circt> mkdir llvm/build

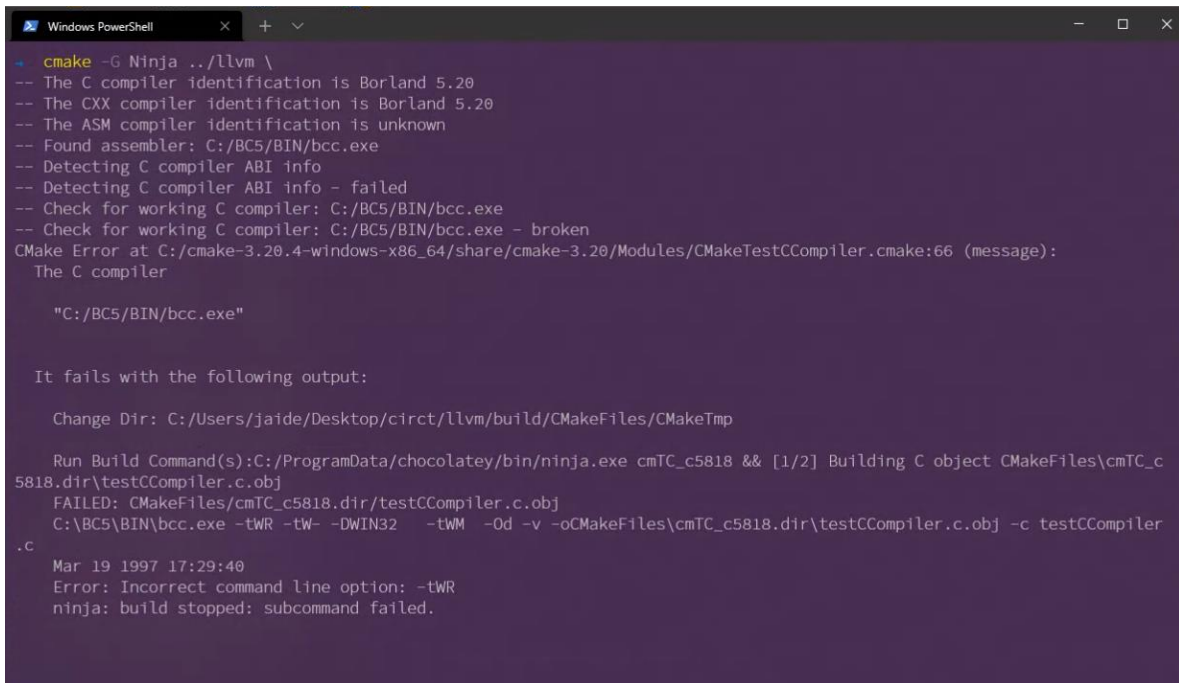
Directorio: D:\Escuela\LenguajesYAutomatasII\circt\llvm

Mode                LastWriteTime         Length Name
----                -
d-----          21/06/2021   03:36 p. m.             build

PS D:\Escuela\LenguajesYAutomatasII\circt> cd llvm/build
PS D:\Escuela\LenguajesYAutomatasII\circt\llvm\build> cmake -G Ninja ../llvm \
cmake : El término 'cmake' no se reconoce como nombre de un cmdlet, función, archivo de script o programa
ejecutable. Compruebe si escribió correctamente el nombre o, si incluyó una ruta de acceso, compruebe que
dicha ruta es correcta e inténtelo de nuevo.
En línea: 1 Carácter: 1
+ cmake -G Ninja ../llvm \
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (cmake:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

Figura 8. Creación de la carpeta llvm.

- Como resultado obtenemos un error de compilación.



```
Windows PowerShell
+ ~-
cmake -G Ninja ../llvm \
-- The C compiler identification is Borland 5.20
-- The CXX compiler identification is Borland 5.20
-- The ASM compiler identification is unknown
-- Found assembler: C:/BC5/BIN/bcc.exe
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - failed
-- Check for working C compiler: C:/BC5/BIN/bcc.exe
-- Check for working C compiler: C:/BC5/BIN/bcc.exe - broken
CMake Error at C:/cmake-3.20.4-windows-x86_64/share/cmake-3.20/Modules/CMakeTestCCompiler.cmake:66 (message):
  The C compiler

    "C:/BC5/BIN/bcc.exe"

It fails with the following output:

Change Dir: C:/Users/jaide/Desktop/circt/llvm/build/CMakeFiles/CMakeTmp

Run Build Command(s):C:/ProgramData/chocolatey/bin/ninja.exe cmTC_c5818 && [1/2] Building C object CMakeFiles\cmTC_c
5818.dir\testCCompiler.c.obj
FAILED: CMakeFiles\cmTC_c5818.dir\testCCompiler.c.obj
C:/BC5/BIN/bcc.exe -tWR -tW- -DWIN32 -tWM -Od -v -oCMakeFiles\cmTC_c5818.dir\testCCompiler.c.obj -c testCCompiler
.c
Mar 19 1997 17:29:40
Error: Incorrect command line option: -tWR
ninja: build stopped: subcommand failed.
```

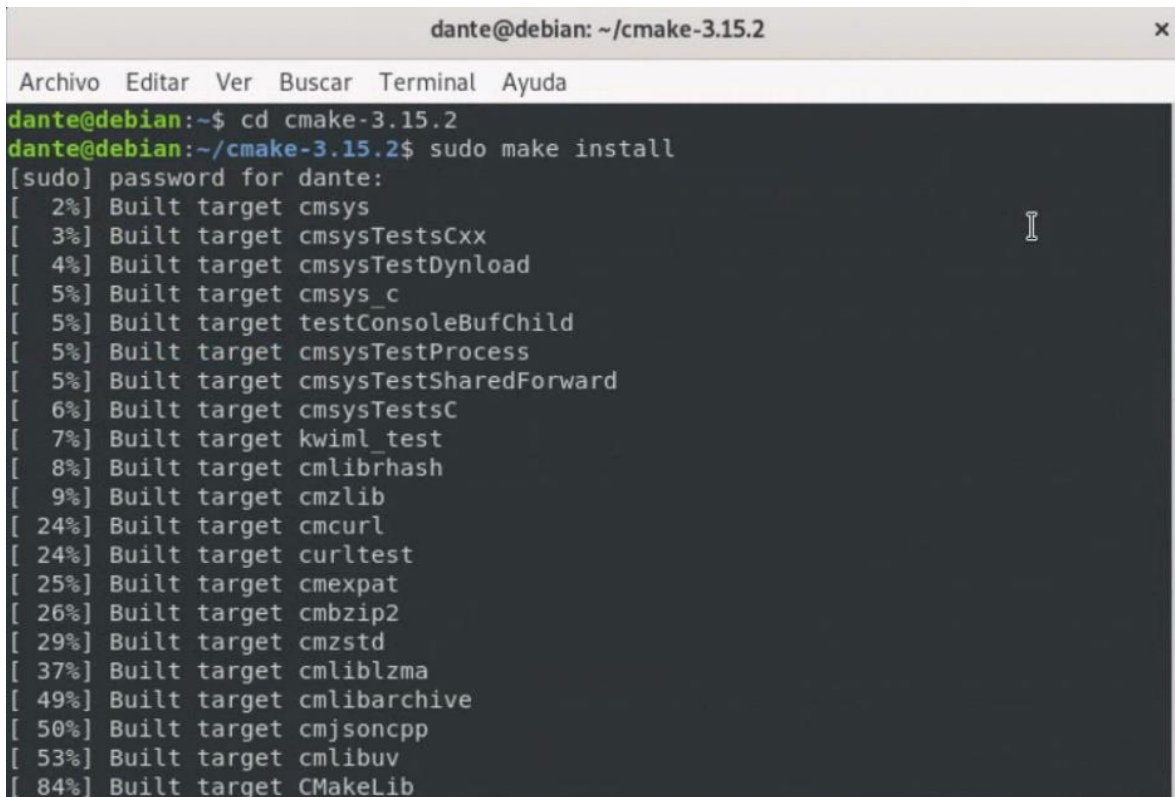
Figura 9. Error de compilación.

En base a este error decidimos cambiar de sistema operativo ya que no pudimos encontrar una solución óptima que nos funcionara.

Linux – Debian

Al estar en Linux no necesitamos instalar un compilador, ya que viene uno por defecto, tampoco instalamos ningún gestor de librerías porque Linux nos da esa libertad, lo único que necesitaremos serán dos librerías.

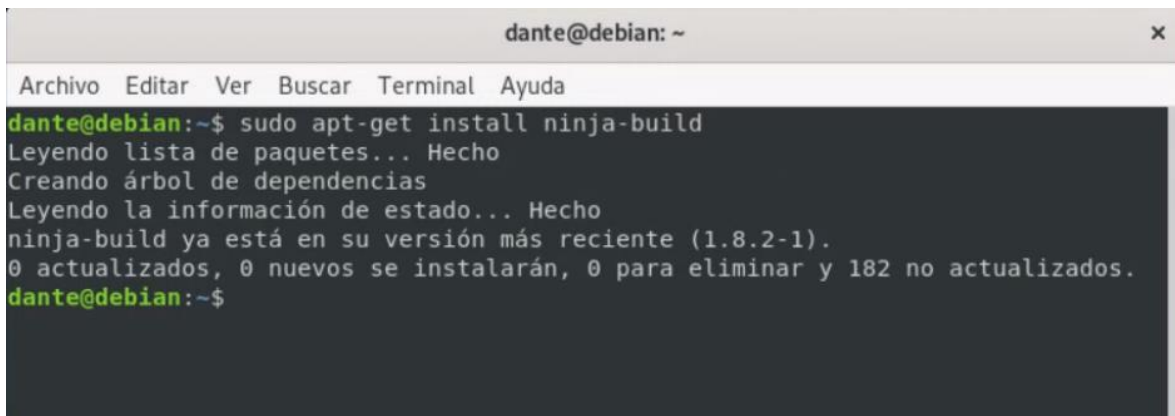
- Instalamos la librería Cmake, para poder operar con los comandos necesarios.



```
dante@debian: ~/cmake-3.15.2
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
dante@debian:~$ cd cmake-3.15.2
dante@debian:~/cmake-3.15.2$ sudo make install
[sudo] password for dante:
[ 2%] Built target cmsys
[ 3%] Built target cmsysTestsCxx
[ 4%] Built target cmsysTestDynload
[ 5%] Built target cmsys_c
[ 5%] Built target testConsoleBufChild
[ 5%] Built target cmsysTestProcess
[ 5%] Built target cmsysTestSharedForward
[ 6%] Built target cmsysTestsC
[ 7%] Built target kwiml_test
[ 8%] Built target cmlibrhash
[ 9%] Built target cmzlib
[24%] Built target cmcurl
[24%] Built target curltest
[25%] Built target cmexpat
[26%] Built target cmbzip2
[29%] Built target cmzstd
[37%] Built target cmliblzma
[49%] Built target cmlibarchive
[50%] Built target cmjsoncpp
[53%] Built target cmlibuv
[84%] Built target CMakeLib
```

Figura 10. Instalación de Cmake – Debian.

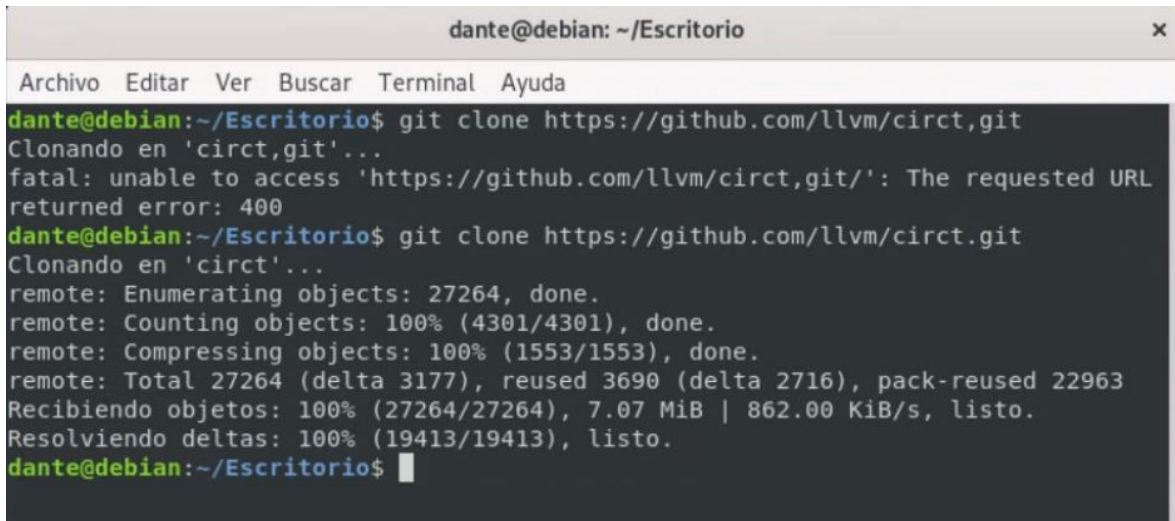
- Instalamos ninja-build.



```
dante@debian: ~
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
dante@debian:~$ sudo apt-get install ninja-build
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
ninja-build ya está en su versión más reciente (1.8.2-1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 182 no actualizados.
dante@debian:~$
```

Figura 11. Instalación de Ninja – Debian.

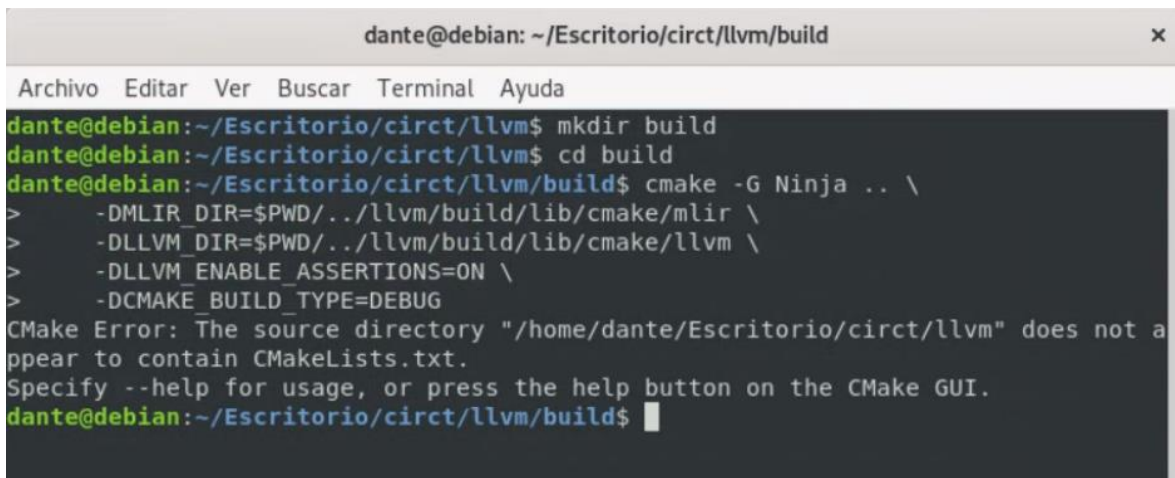
- Clonamos el repositorio de GitHub



```
dante@debian: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
dante@debian:~/Escritorio$ git clone https://github.com/llvm/circt.git
Clonando en 'circt.git'...
fatal: unable to access 'https://github.com/llvm/circt.git/': The requested URL
returned error: 400
dante@debian:~/Escritorio$ git clone https://github.com/llvm/circt.git
Clonando en 'circt'...
remote: Enumerating objects: 27264, done.
remote: Counting objects: 100% (4301/4301), done.
remote: Compressing objects: 100% (1553/1553), done.
remote: Total 27264 (delta 3177), reused 3690 (delta 2716), pack-reused 22963
Recibiendo objetos: 100% (27264/27264), 7.07 MiB | 862.00 KiB/s, listo.
Resolviendo deltas: 100% (19413/19413), listo.
dante@debian:~/Escritorio$
```

Figura 12. Clonación del repositorio – Debian.

- Creamos la carpeta de llvm/build y construimos el test de llvm/mlir



```
dante@debian: ~/Escritorio/circt/llvm/build
Archivo Editar Ver Buscar Terminal Ayuda
dante@debian:~/Escritorio/circt/llvm$ mkdir build
dante@debian:~/Escritorio/circt/llvm$ cd build
dante@debian:~/Escritorio/circt/llvm/build$ cmake -G Ninja .. \
> -DMLIR_DIR=$PWD/../../llvm/build/lib/cmake/mlir \
> -DLLVM_DIR=$PWD/../../llvm/build/lib/cmake/llvm \
> -DLLVM_ENABLE_ASSERTIONS=ON \
> -DCMAKE_BUILD_TYPE=DEBUG
CMake Error: The source directory "/home/dante/Escritorio/circt/llvm" does not a
ppear to contain CMakeLists.txt.
Specify --help for usage, or press the help button on the CMake GUI.
dante@debian:~/Escritorio/circt/llvm/build$
```

Figura 12. Creación de la carpeta LLVM – Debian.

RESULTADOS

```

Windows PowerShell
Directorio: D:\Escuela\LenguajesYAutomatasII\circt

Mode                LastWriteTime         Length Name
----                -
d----- 21/06/2021 03:36 p. m.          .github
d----- 21/06/2021 03:36 p. m.          .vscode
d----- 21/06/2021 03:36 p. m.          cmake
d----- 21/06/2021 03:36 p. m.          codeowners
d----- 21/06/2021 03:36 p. m.          docs
d----- 21/06/2021 03:36 p. m.          frontends
d----- 21/06/2021 03:36 p. m.          include
d----- 21/06/2021 03:36 p. m.          integration_test
d----- 21/06/2021 03:36 p. m.          lib
d----- 21/06/2021 03:36 p. m.          llvm
d----- 21/06/2021 03:36 p. m.          test
d----- 21/06/2021 03:36 p. m.          tools
d----- 21/06/2021 03:36 p. m.          utils
-a----- 21/06/2021 03:36 p. m.           56 .clang-format
-a----- 21/06/2021 03:36 p. m.          971 .clang-tidy
-a----- 21/06/2021 03:36 p. m.          111 .gitignore
-a----- 21/06/2021 03:36 p. m.           80 .gitmodules
-a----- 21/06/2021 03:36 p. m.           56 .style.yapf
-a----- 21/06/2021 03:36 p. m.           14 .yapfignore
-a----- 21/06/2021 03:36 p. m.        13088 CMakeLists.txt
-a----- 21/06/2021 03:36 p. m.        13469 LICENSE
-a----- 21/06/2021 03:36 p. m.        5654 README.md

PS D:\Escuela\LenguajesYAutomatasII\circt>

```

Figura 13. Resultado obtenido en Windows.

```

dante@debian: ~/Escritorio/circt
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
dante@debian:~/Escritorio/circt$ ls -a
.          CMakeLists.txt  .github          lib          test
..         codeowners      .gitignore       LICENSE       tools
.clang-format docs            .gitmodules      llvm         utils
.clang-tidy frontends       include          README.md    .vscode
cmake      .git           integration_test .style.yapf  .yapfignore
dante@debian:~/Escritorio/circt$

```

Figura 14. Resultado obtenido en Linux – Debian.

CONCLUSIÓN

Como podemos observar, tanto en la prueba de llvm/mlir en Windows como Debian, no se lograron los resultados esperados, esto debido a problemas de instalación de las diferentes dependencias con las que debemos trabajar.

Con Windows el problema fue directamente el compilador que utilizamos, instalamos WinMG, pero como ya teníamos el compilador de Borland C++ utilizó sus librerías y no nos dejó trabajar.

Con Linux el error en el directorio, específicamente donde se encontraba el archivo CMakeList.txt, donde se guardan las configuraciones para levantar un proyecto, por lo cual no nos dejó seguir con nuestra prueba, por esa misma razón el proyecto quedó inconcluso ya que los pasos son fundamentales para la arquitectura del proyecto.

El proyecto Circuit IR Compilers and Tools (CIRCT) tiene como fin utilizar MLIR y la metodología de desarrollo LLVM al dominio de los instrumentos de diseño de hardware para la creación de nuevos aceleradores.

REFERENCIAS

Chapyzhenka, A. (2021, enero 13). Llvm / cirtc-www. Github. Recuperado el 27 de mayo del 2021, de <https://github.com/llvm/cirtc-www>

CIRCT. (n.d.). Compiladores y herramientas de circuitos infrarrojos. CIRCT. Recuperado el 28 de mayo del 2021, de <https://cirtc.llvm.org/>

LLVM. (n.d.). The LLVM Compiler Infrastructure. LLVM. Recuperado el 28 de mayo de 2021, de <https://llvm.org/>

CIRCT. (n.d.). Documentación del código. CIRCT. Recuperado el 29 de mayo de 2021, de <https://cirtc.llvm.org/docs/>

CIRCT. (n.d.). CIRCT - DOXYGEN. CIRCT. Recuperado el 02 de junio del 2021, de <https://cirtc.llvm.org/doxygen/>

MLIR. (n.d.). Multi-Level Intermediate Representation. Multi-Level IR Compiler Framework. Recuperado el 02 de junio de 2021, de <https://mlir.llvm.org/>

Chand, G. (2020, marzo 28). Los aceleradores jugarán un papel clave en la computación 5G y Edge. DELL Technologies. Recuperado el 03 de junio del 2021, de <https://www.delltechnologies.com/es-es/blog/aceleradores-claves-computacion-5g-y-edge/>

Roca, J. (2020, noviembre 06). ¿Qué son y cómo funcionan los aceleradores o coprocesadores en tu PC? Hardzone. Recuperado el 03 de junio de 2021, de <https://hardzone.es/reportajes/que-es/aceleradores-proposito-especifico/>

Escuela de negocios. (2019, mayo 20). Gestión ágil vs gestión tradicional de proyectos ¿cómo elegir? Recuperado el 04 de junio del 2021, de <https://www.escueladenegociosfedacom/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir#:~:text=En%20la%20metodolog%C3%ADa%20C3%A1gil%2C%20cada.com>

[parte%20la%20propiedad%20del%20proyecto.&text=En%20el%20enfoque%20tradicional%2C%20cada.del%20tiempo%20y%20presupuesto%20estimados](#)

Wikipedia. (2021, junio 1). Acelerador de IA. Wikipedia. Recuperado el 06 de junio de 2021, de https://es.wikipedia.org/wiki/Acelerador_de_IA

Anónimo. (n.d.). Cómo instalar ninja-build en Ubuntu. Recuperado el 18 de junio del 2021, de <https://howtoinstall.co/es/ninja-build>

Buzdar, Karim. (2021, febrero). Install Cmake on Debian 10. Recuperado el 18 de junio del 2021, de <https://linuxhint.com/install-cmake-on-debian-10/>

Anónimo. (n.d.). Ubuntu – Cmake Error: Coud not find CMAKE_ROOT. Recuperado el 19 de junio del 2021, de https://itectec.com/ubuntu/ubuntu-cmake-error-could-not-find-cmake_root/