# 1.UEFI-edk2 development environment construction.

# EDK2 is a widely used open source implementation of UEFI, which supports development under a variety of operating system platforms. For the convenience of development and debugging, the development of UEFI is carried out with visual studio under win10. VS is known as the surface Top one's IDE, powerful and self-contained, often developed to do more with less. This blog we complete the edk2 source source development and compilation environment, while compiling and running a simulator to test the efi program.

# First, the installation of related tools.

# First download the installation vs2017 or vs2015, the installation component sinned "Desktop Development with C" item.

# Next, you'll download the NASM tool, which is recommended for installation in the "C:"NASM" directory.

# Finally, download and install the IASL tool, and after downloading it, it is recommended to install it under the "C: .IASL" directory.

# After the installation is complete, you need to configure the next environment variable and add the IASL installation path to the Path environment variable, as shown in the following image.

# You also need to create a new environment variable NASM\_PREFIX, and the value of the variable is set to the installation path of the NASM.

# 在这里插入图片描述

# 在这里插入图片描述

# Second, compile basetools.

# Download the source package of edk2 on the edk2 github project home page, open the command line under windows via Win-R -gt; cmd, cd to edk2's source directory, and enter the following command to compile the basetools.

# edksetup.bat Rebuild

# Once compiled, some tools will be generated in the $(edk2-base) - BaseTools ? Bin - Win32 directory, which will be used when you compile efi for subsequent

# copilations.

# 在这里插入图片描述

# Three, config build.

# Next, you need to compile the build config file, which is used to configure information such as the tool chain and project name for subsequent compilation projects, and enter the following command.

# edksetup.bat

# When the edksetup.bat command is executed, several .txt text files will be generated under the $(edk2-base)/Conf directory, where the Conf-target.txt file needs to be changed under the TOOL\_CHAIN\_TAG configuration, which can be written according to its own VS version, which can be VS2015/VS2017/VS2019, etc.

# ACTIVE\_PLATFORM configured as the default compiled project, the EmulatorPkg/EmulatorPkg.dsc is a simulator's engineering

# 在这里插入图片描述

# path.

# TARGET is configured as a compilation mode for the project, which can be RELEASE or DEBUG.

# TARGET\_ARCH is the target architecture, with X64 or IA32 optional in the case of windows plus VS.

# Fourth, compile the simulator.

# With the simulator, it is convenient to develop and debug EFI drive/ program, do not have to burn the firmware on the physical machine FLASH start test so cumbersome. The default configuration of Conf?target.txt is the engineering of the EmulatorPkg simulator, where only build commands can be compiled.

# edk2setup.bat

# build

# When compiled successfully, the following illustration shows.

# When the compilation is complete, we can run the simulator, we can cd to the output directory of the project, run the target file.

# 在这里插入图片描述

cd Build\EmulatorIA32\DEBUG\_VS2017\IA32\

# WinHost.exe

# After running WinHost.exe, the start-up interface of the UEFI firmware comes out and can be loaded by the efi program by selecting the button to enter the UEFI SHELL or the configuration interface.

# 在这里插入图片描述

# V. Debug the simulator.

# Open simulator engineering with visual studio: $(edk2-base) , EmulatorPkg, Win,VS2017, Win.sln, you can compile and run UEFI simulator project in visual software, you can also play break-ins on efi applications/drives, run simulator engineering in debug mode, you can debug efi applications/drivers, as shown in the following image, the specific debugging development follow-up process is slowly talked about.

# 在这里插入图片描述

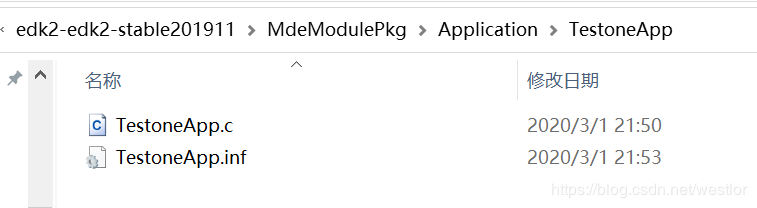
# 2.UEFI-edk2 writes the first application.

The previous chapter set up the compilation development and simulator environment of UEFI, here began to write the first application. Before writing, i briefly introduce the source directory structure of UEFI-edk2, which mainly has the following sub-directories under the source directory:

| **The directory name.** | **Description.** |
| --- | --- |
| BaseTools. | Contains the set of binary compilation tools and the compilation environment profiles required for code compilation. |
| MdePkg. | Contains basic underlying library functions, protocols, and industrial standards that are common to each platform. |
| MdeModulePkg. | Contains a range of modules that are common to each platform, including an example of an application module for a public library in MdePkg. |
| Conf. | Save the compilation environment information, compile the target path, and compiler parameters, under which the tool generates three profiles. |
| Edk Shell Pkg, Shell Pkg. | Provides a platform-generic UEFI Shell application development environment. |
| Edk FatBin Pkg. | Contains the original FAT driverforthed CPU architecture. |
| EmulatorPkg. | A 32/64-bit simulator can be loaded under the Windows operating system, providing a platform for the UEFI operating environment. |
| ArmPkg, ArmPlatform Pkg. | For the implementation of ARM platform, related to the specific platform hardware. |
| NetWorkPkg, Uefi CpuPkg. | Network, CPU-driven reference implementation. |
| DuetPkg. | Provides a support library based on the traditional BIOS operating environment. |
| Option RomPkg. | Provides examples of compiling PCI-compatible images for different CPU architectures. |

Each directory ending with Pkg is a project package, compiled by the "build -p-Pkg?Pkg.dsc" command to compile the corresponding engineering package, such as compiling the simulator's engineering package can be used to "build porPkg?Emulator?Pkg.dsc" command.  
Each engineering package can reference the applications and libraries and driver modules of other engineering packages by changing the configuration of the configuration of the spkg?Pkg.inf file, and some engineering packages that generate The UEFI startup firmware can be added to the application/library/driver module of itself or other engineering packages by changing the configuration of the configuration.

## First, the application code writing.

Write the application module first to create a new module directory, as described in the table foreword, we are here to create a simple test program, not for a specific platform, the new application module in the MdeModulePkg? The app module is tentatively named TestoneApp, and two new files, TestoneApp.c and TestoneApp.inf, are created in this directory, as shown in the following image.  
TestoneApp.inf is an engineering profile of the module, somewhat similar to Makefile under a Linux   
system. The contents of the TestoneApp.inf file are as follows:

[Defines]

INF\_VERSION = 0x00010005

BASE\_NAME = TestoneApp

FILE\_GUID = 12345678-ABCD-EF01-2345-123456789012

MODULE\_TYPE = UEFI\_APPLICATION

VERSION\_STRING = 1.0

ENTRY\_POINT = UefiMain

[Sources]

TestoneApp.c

[Packages]

MdePkg/MdePkg.dec

[LibraryClasses]

UefiApplicationEntryPoint

UefiLib

There are some definition blocks in the TestoneApp.inf file, each of which functions as follows.

[Defines] : Used to define the properties of the module and some custom variables.，BASE\_NAME configures the file name of the EFI module generated after the application is compiled, MODULE\_TYPE specifies that this is a UEFI application module, and ENTRY\_POINT defines the entry function name of the module , FILE\_GUID is the only code of the module, this can be written as long as it is not the same as other modules.

[Sources] :

[Packages] : List the package declaration files of the packages referenced by this module.

[LibraryClasses] : Need to list the libraries used by this module.

Then look at the TestoneApp.c source file, the source function is very simple, is to print a "Hello, world!" the string of . Note that the string needs to be preceded by an L letter, and the surface is a Unicode string.

#include <Uefi.h>

#include <Library/UefiLib.h>

EFI\_STATUS EFIAPI UefiMain(

IN EFI\_HANDLE ImageHandle,

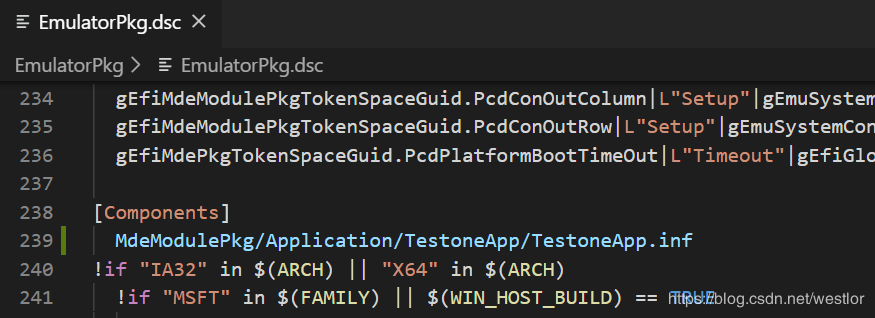
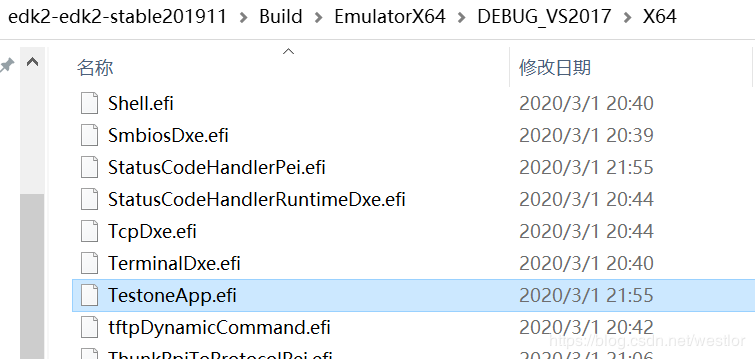
IN EFI\_SYSTEM\_TABLE \*SystemTable){

Print(L"Hello, world!\r\n");

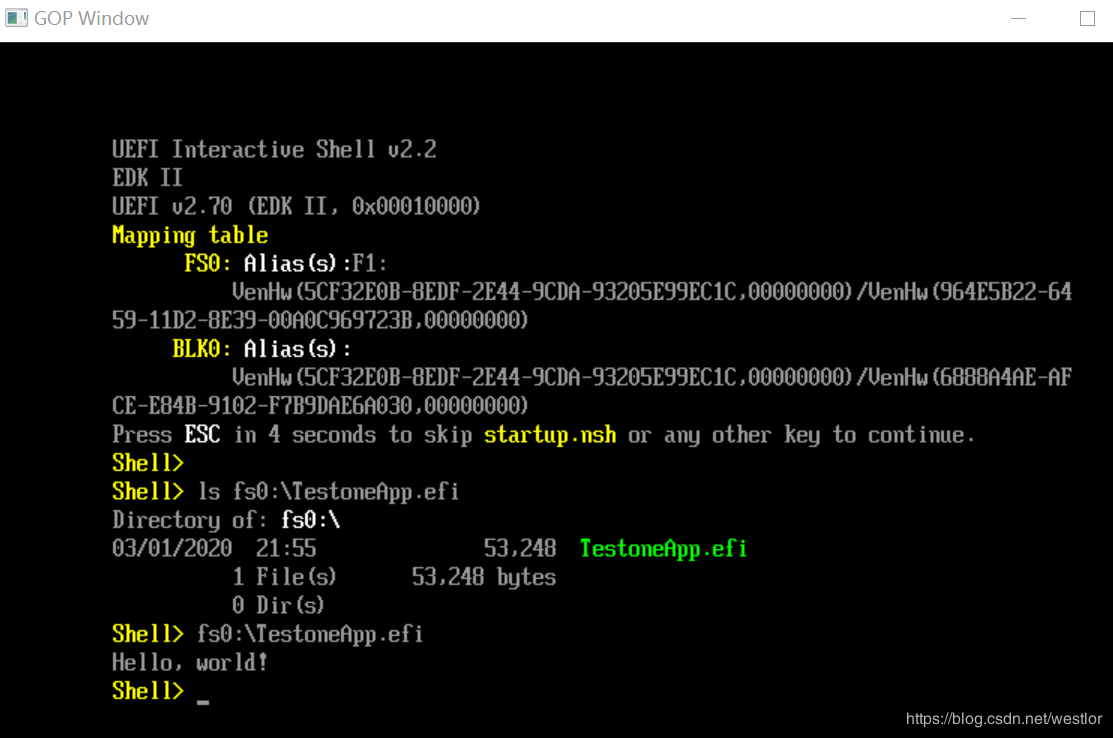
return EFI\_SUCCESS;

}

## Second, compile custom applications.

If you compile the test application module, add the module's profile path to the engineering profile of the EmulatorPkg package. Open the EmulatorPkg.Emulatsc file and add "MdeModulePkg/Application/TestoneApp/TestoneApp.inf" under its "Components" block.  
The EmulatorPkg project package can now be compiled, build commands executed on the cmd command line, or the EmulatorPkg project can be compiled through visual stdio.  After the compilation is complete, in addition to generating the default WinHost.exe (hypervisor) and FV\_RECOVERY.fd (UEFI firmware), we also generate the application module TestoneApp.efi that we added, as shown in the following image.  
When compiling the EmulatorPkg project through visual studio, here will encounter the error of the newspaper "'gbk' codec can'encode character' and 'u2cbf' in position" because the TestoneApp.c file L "Hello,   
world!" Strings are Unicode encoding, gdk can not be resolved, this error can be resolved by changing the source file or configuring the system coding, I suggest that the default code of the direct windows system changed to UTF-8, so that the code can be easily migrated under the windows and Linux platform.

## Third, run the app.

Start the simulator, wait until the firmware runs to the UEFI shell, the ls command can view the generated efi module file location, and then directly execute the module file, as shown in the following image.  


## Fourth, debug the application.

Also open the $(edk2-base) and the "EmulatorPkg"Win?WIN?VS2017?win.sln engineering file, and then open the $(edk2-base) in the visual studio project," the MdeModulePkg?Application?TestoneApp?testoneApp.c file, at "Print (L" Hello, world!?r?n"); This line of code is broken to compile and run the project in debug mode, and the app is ready to be debugged.  
