**Ex:no-06**

**Create a Dynamic Library recipe mydynamic having a source and header files for basic calculator application. In do\_install( ) dynamic lib (.so) add in {libdir} and header file add in {includedir} after that add package in phy-image and then build & test**

This question process of creating a dynamic library (mydynamic) that implements basic calculator functionality and packaging it within a Yocto image (phy-image), ensuring that the dynamic library and its header file are properly installed into their respective directories.

1. **Create the Dynamic Library:**
   * Develop a dynamic library (.so file) that implements a basic calculator.
   * The library source (mydynamic.c) contains the functionality, and the header file (mydynamic.h) provides function declarations.
2. **Write the Yocto Recipe:**
   * Create a Yocto recipe for the dynamic library.
   * The recipe should handle the compilation of the library and installation of the shared library (libmydynamic.so) into the correct system directory (libdir).
   * The header file should be installed into the system's includedir so other programs can access the function declarations from the library.
3. **Modify the phy-image:**
   * Add the mydynamic library to the custom Yocto image (phy-image) so that the image includes this dynamic library as part of the final build.
4. **Build and Test:**
   * Use Bitbake to build the custom image (phy-image) that includes the dynamic library.
   * Test the library on the target board to ensure it functions correctly, verifying that both the dynamic library and the header file are correctly installed and usable.

This entire process ensures that the library is dynamically linked, modular, and reusable, and it is packaged within a Yocto image for deployment on embedded systems or rugged boards.

Step by step procedure

Step 1

Open the docker and move to home->yocto,then give the build command it is move move to

to home->yocto->build.

Step 2:

Then creating layer using this command

bitbake-layers create-layer ../sources/meta-mydynamic

then add the layer ,add layer command show an under the layer creation

step3:

then move to home->yocto->sources->meta-layer->recipe-example

create inside two folder

mkdir mydynamic

mkdir phy-image

step 4:

go to home->yocto->sources->meta-layer->recipe-example->mydynamic

create a folder

mkdir files

step 5:

go to home->yocto->sources->meta-layer->recipe-example->mydynamic->files

create the two files calculator.c and calculator.h

step 6:

writing the code

**calculator.h**

#ifndef MYCALCULATOR\_H

#define MYCALCULATOR\_H

int add(int a, int b);

int subtract(int a, int b);

int multiply(int a, int b);

int divide(int a, int b);

#endif

**calculator.c**

#include "mycalculator.h"

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b == 0) return 0; // Simple error handling for division by zero

 return a / b;

}

Then save the files and quit

Step 7:

Come out from the files

Cd ..

home->yocto->sources->meta-layer->recipe-example->mydynamic

create a recipe file

mydynamic\_0.1.bb

step 8:

write the code inside mydynamic\_0.1.bb

DESCRIPTION = "A dynamic library for basic calculator functions"

LICENSE = "MIT"

LIC\_FILES\_CHKSUM = "file://${COMMON\_LICENSE\_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC\_URI = "file://calculator.c \

file://calculator.h"

S = "${WORKDIR}"

do\_compile() {

# Compile the source into a shared library

${CC} ${CFLAGS} ${LDFLAGS} -shared -fPIC -o libcalculator.so calculator.c

}

do\_install() {

# Install the dynamic library (.so) into the libdir

install -d ${D}${libdir}

install -m 0755 libcalculator.so ${D}${libdir}/libcalculator.so

# Install the header file into the includedir

install -d ${D}${includedir}

install -m 0644 calculator.h ${D}${includedir}/calculator.h

}

FILES\_${PN} = "${libdir}/libcalculator.so"

FILES\_${PN}-dev = "${includedir}/calculator.h"

Then save and quit.

This file compile and generate library file into usr/lib and header file into usr/include

Step 9:

Come out from the mydynamic and go inside the phy-image folder s

home->yocto->sources->meta-layer->recipe-example->phy-image

create the inside phy-image.bb file

**write the code inside phy-image.bb**

DESCRIPTION = "Custom image including mydynamic"

LICENSE = "MIT"

IMAGE\_INSTALL += "mydynamic"

IMAGE\_INSTALL += "mydynamic mydynamic-dev"

**Purpose of phy-image**

The phy-image in your Yocto project is a custom image recipe that defines the packages and components that will be included in the final system image. This image is what will be built, deployed, and run on your rugged board or target device.

1. **Custom Image Definition**: The phy-image.bb file describes what software, libraries, and tools will be included in the final image that runs on the rugged board. This includes basic system utilities, libraries, and any custom or third-party software required for your application.
2. **Package Selection**: The IMAGE\_INSTALL variable in the phy-image recipe lists all the packages that will be installed in the image. For example, you have:

IMAGE\_INSTALL += "mydynamic usbutils sasar gcc packagegroup-core-buildessential"

This includes your mydynamic package (which contains your dynamic library and header files), usbutils, and other packages that are needed for your system to function.

1. **Include Custom Software**: You are adding your custom-built library mydynamic (which includes libcalculator.so and calculator.h) to the image. This allows you to develop and run applications that use this library on the target board.
2. **System Configuration**: The phy-image also defines the system's behavior by specifying which packages or configurations are required for the system to boot and function. This could include necessary drivers, utilities, networking tools, and other system-level packages.
3. **Portability**: The image recipe ensures that you have a reproducible build that can be deployed on different hardware of the same type (e.g., rugged boards or other embedded systems). It simplifies the deployment process as the same image can be used across similar devices.

In your project, the phy-image serves as the foundation for deploying your dynamic library (mydynamic) and any other tools or utilities you need for your application to run directly on the rugged board.

Step 10:

Then start build process

home->yocto->build

give the command

bitbake phy-image

bitbake mydynamic(optional)

step 11:

home->yocto->build->conf

**local.conf**

then include the mydynamic package into local.conf and packagegroup-core-buildessential this package used to gcc command working on rugged board

IMAGE\_INSTALL\_append = " mydynamic mydynamic-dev"

IMAGE\_INSTALL\_append = " packagegroup-core-buildessential"

Step 12:

Then give the bitbake command

Step 13:

home->yocto->build->tmp-deploy->image>rugged-board

copy the rugged board folder from the docker

docker cp <container\_id\_or\_name>:<path\_in\_container> <path\_on\_host>

example:

docker cp abc123:/usr/src/app/file.txt ~/home/user/

that file are contain boot files and rootfs

step 14 :

change the file names into the rugged board folder for boot files

then boot the sd card using this boot files and rootfs

then insert the sd card into rugged board start the boot process

step 15:

after booting process create a c file

vi test.c

#include <stdio.h>

#include "calculator.h"

int main() {

int a = 10, b = 5;

printf("Add: %d\n", add(a, b));

printf("Subtract: %d\n", subtract(a, b));

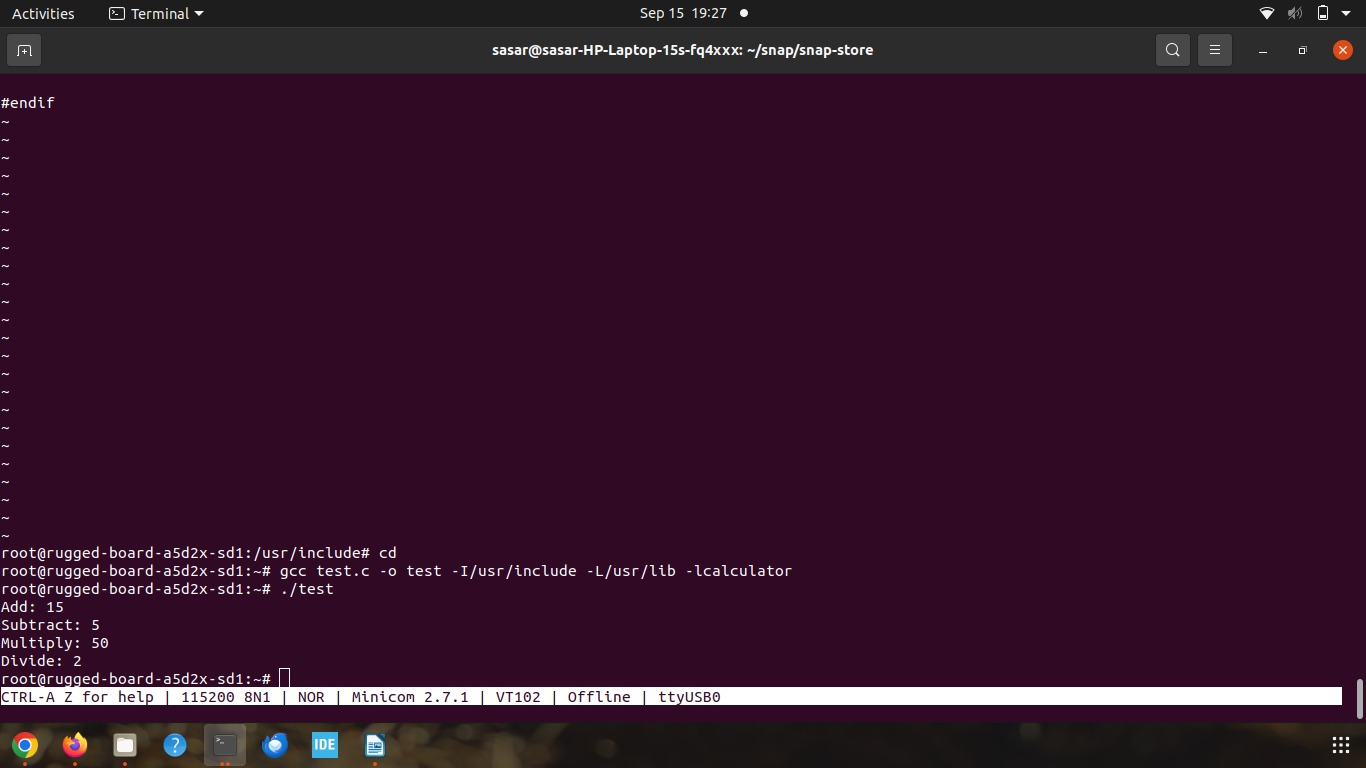
printf("Multiply: %d\n", multiply(a, b));

printf("Divide: %d\n", divide(a, b));

return 0;

}

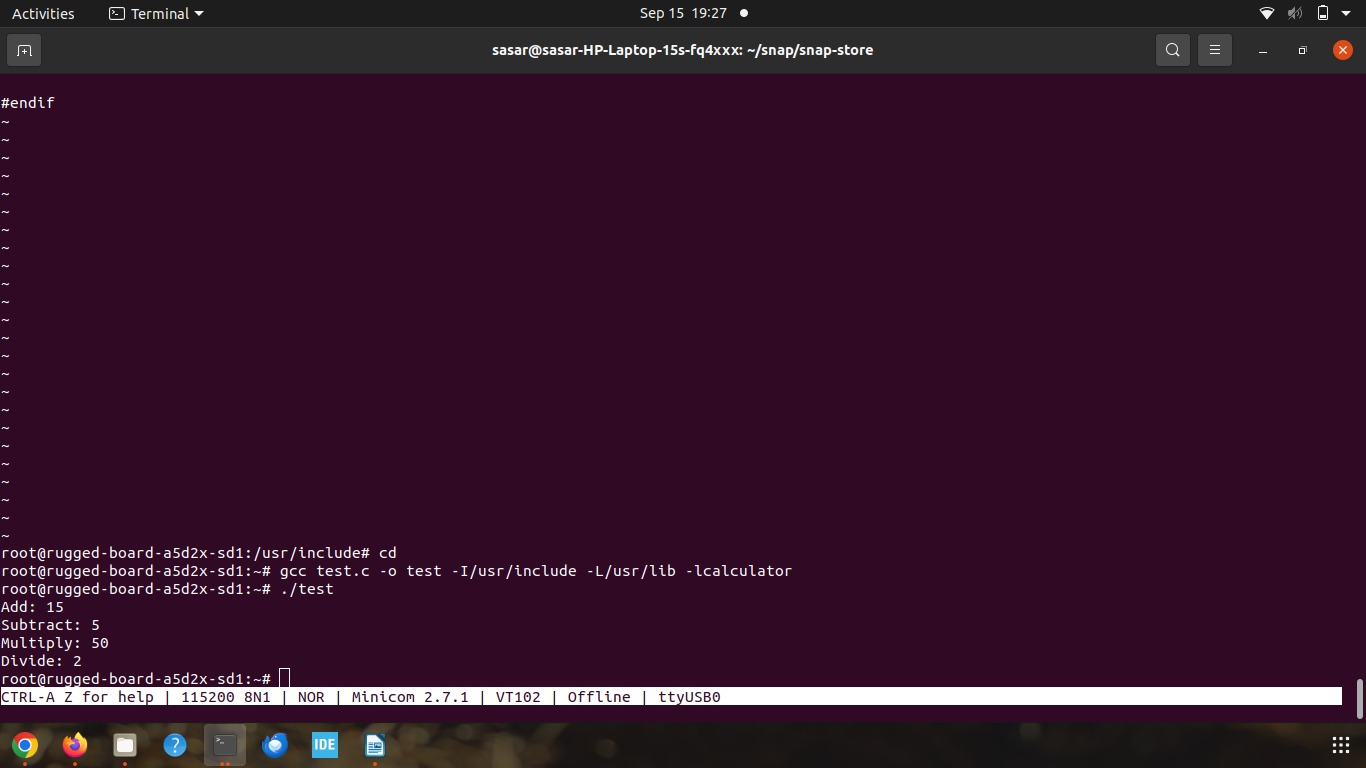
Step 16:



-I refer the header file path –L refer the library path

Step 17:

After compile the program like



Step 18:

Stop the program and finished…..