

RATHINAM TECHNICAL CAMPUS

RATHINAM TECHZONE

POLLACHI MAIN ROAD, EACHANARI, COIMBATORE-641021.



MASTER OF COMPUTER APPLICATION

RECORD NOTE BOOK

23MCP11 EMBEDDED LINUX PORTING WITH YOCTO LABORATORY

NAME :

REGISTER NUMBER :

YEAR/SEMESTER :

ACADEMIC YEAR :



RATHINAM TECHNICAL CAMPUS

RATHINAM TECHZONE

POLLACHI MAIN ROAD, EACHANARI, COIMBATORE-641021.

BONAFIDE CERTIFICATE

NAME :

ACADEMIC YEAR :

YEAR/SEMESTER :

BRANCH :

UNIVERSITY REGISTER NUMBER:

Certified that this is the bonafide record of work done by the above student in the
_____Laboratory during the year 2024-2025.

Head of the Department

Staff-in-Charge

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

INDEX

[illegible]

INDEX

[illegible]

EX NO: 01 DATE:	Define a simple recipe for a "Hello World" application. Understand the concept of Yocto layers and workflow. Use .bbappend files to modify existing recipes.
----------------------------------	---

AIM:

To create a simple recipe for a "Hello World" application.

ALGORITHM:

STEP 1: Start the process.

```
user :~$ docker pull ubuntu:16.04
user :~$ docker run -it ubuntu:16.04
root@584a332c0218:/# apt-get update
```

STEP 2: Pull and run an Ubuntu container.

STEP 3: Install the packages needed for yocto.

```
root@13f28da9360f:/home# cd yocto/
root@13f28da9360f:/home/yocto# cd sources/
```

STEP 4: Create a recipe directory in meta-rba5d2x (ex:recipe-hello).

```
root@13f28da9360f:/home/yocto/sources# cd meta-rba5d2x/
```

STEP 5: Go to the created recipe directory and create sub directory 'hello'. Go to the hello directory and create a 'file' directory go to the file directory and create a C file 'hello.c' and write into the C file simple 'hello world' program.

```
root@13f28da9360f:/home/yocto/sources/meta-rba5d2x# mkdir recipes-sample/
root@13f28da9360f:/home/yocto/sources/meta-rba5d2x/recipes-sample# mkdir
hello
root@13f28da9360f:/home/yocto/sources/meta-rba5d2x/recipes-sample# vim
hello_0.1.bb
```

STEP 6: Create bb file in 'hello' direcorey(ex:hello_0.1.bb) and write into the code
hello_0.1.bb

```
root@13f28da9360f:/home/yocto/sources/meta-rba5d2x/recipes-sample# mkdir
files
root@13f28da9360f:/home/yocto/sources/meta-rba5d2x/recipes-sample# vim
hello.c
```

STEP 7: Change the directory to conf and modify the local.conf.

```
root@584a332c0218:/home/yocto/sources# cd poky
```

```
root@584a332c0218:/home/yocto/sources/poky# source oe-init-build-env
```

```
root@584a332c0218:/home/yocto/build# cd conf
```

```
root@584a332c0218:/home/yocto/build/conf# vim local.conf
```

STEP 8: Compile the project.

STEP 9: Stop the process.

PROGRAM

local.conf:

This sets the default machine to be rugged-board-a5d2x(NOR Flash) if no other machine is selected:

```
MACHINE ?= "rugged-board-a5d2x-sd1"
```

Default policy config

```
DISTRO ?= "poky-tiny"
```

Package Management configuration

```
PACKAGE_CLASSES ?= "package_rpm"
```

Extra image configuration defaults

```
EXTRA_IMAGE_FEATURES ?= "debug-tweaks"
```

Interactive shell configuration

```
PATCHRESOLVE = "noop"
```

Disk Space Monitoring during the build

```
BB_DISKMON_DIRS ??= "\
```

```
  STOPTASKS,{TMPDIR},1G,100K \
```

```
  STOPTASKS,{DL_DIR},1G,100K \
```

```
  STOPTASKS,{SSTATE_DIR},1G,100K \
```

```
  STOPTASKS,/tmp,100M,100K \
```

```
  ABORT,{TMPDIR},100M,1K \
```

```
  ABORT,{DL_DIR},100M,1K \
```

```
  ABORT,{SSTATE_DIR},100M,1K \
```

```
  ABORT,/tmp,10M,1K"
```

Qemu configuration

```
PACKAGECONFIG_append_pn-qemu-native = " sdl"
```

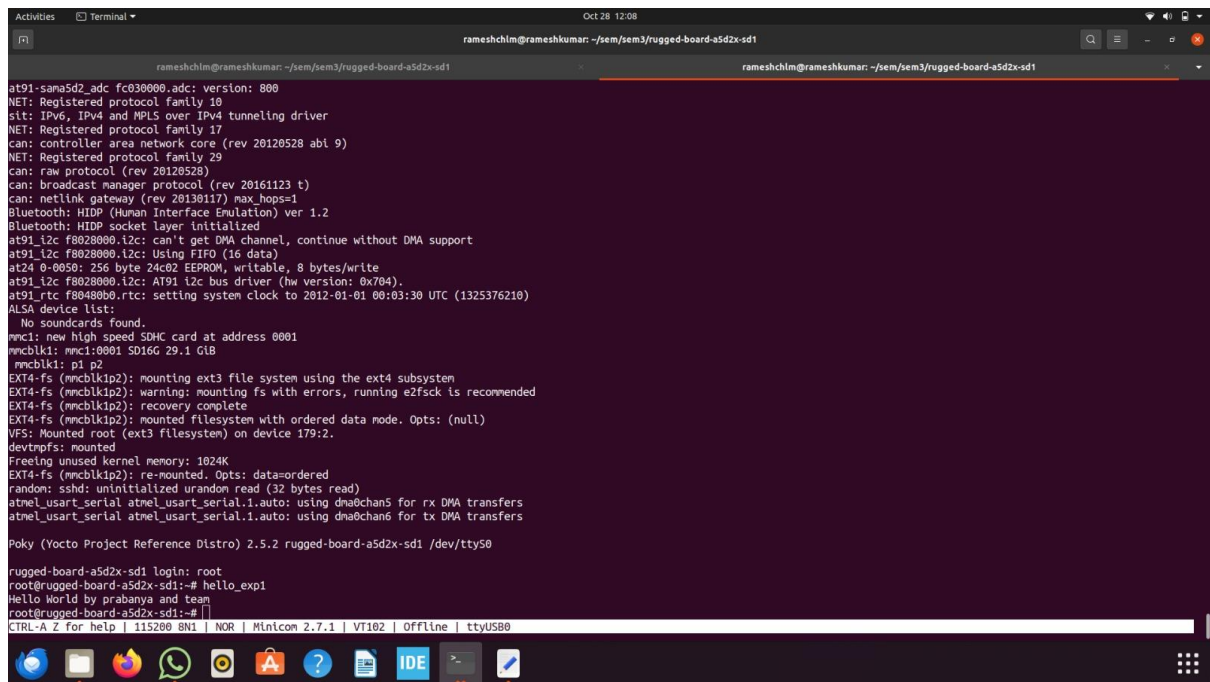
```
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
```

```
#ASSUME_PROVIDED += "libsdl-native"
```

```
export PYTHONDONTWRITEBYTECODE="0"
```

```
INCLUDE_PYCS=""
# CONF_VERSION is increased each time build/conf/ changes incompatibly and is used to
# track the version of this file when it was generated. This can safely be ignored if
# this doesn't mean anything to you.
CONF_VERSION = "1"
IMAGE_INSTALL_append = " hello"
```

OUTPUT:

A terminal window showing the boot process of a Poky (Vocto Project Reference Distro) 2.5.2 rugged-board-asd2x-sd1. The logs include network initialization (IPv6, MPLS, IPv4 tunneling), Bluetooth HIDP initialization, ALSA device list (no soundcards found), and filesystem mounting (EXT4-fs, VFS, devtmpfs). The boot process concludes with the login prompt 'root@rugged-board-asd2x-sd1:~#' and the message 'Hello World by prabanya and team'. The terminal window has a title bar with 'Activities', 'Terminal', and the date 'Oct 28 12:08'. The bottom of the window shows a taskbar with various application icons.

```
Oct 28 12:08
rameshchlm@rameshkumar: ~/sem/sem3/rugged-board-asd2x-sd1
rameshchlm@rameshkumar: ~/sem/sem3/rugged-board-asd2x-sd1
at91-sama5d2_adc: version: 800
NET: Registered protocol family 10
sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
NET: Registered protocol family 17
can: controller area network core (rev 20120528 abi 9)
NET: Registered protocol family 29
can: raw protocol (rev 20120528)
can: broadcast manager protocol (rev 20161123 t)
can: netlink gateway (rev 20130117) max_hops=1
Bluetooth: HIDP (Human Interface Emulation) ver 1.2
Bluetooth: HIDP socket layer initialized
at91_l2c_f8028000.l2c: can't get DMA channel, continue without DMA support
at91_l2c_f8028000.l2c: Using FIFO (16 data)
at24 0-0050: 256 byte 24c02 EEPROM, writable, 8 bytes/write
at91_l2c_f8028000.l2c: AT91 l2c bus driver (hw version: 0x704)
at91_rtc_f8048000.rtc: setting system clock to 2012-01-01 00:03:30 UTC (1325376210)
ALSA device list:
  No soundcards found.
mmc1: new high speed SDHC card at address 0001
mmcblk1: mmc1:0001 SD16G 29.1 GiB
mmcblk1: p1 p2
EXT4-fs (mmcblk1p2): mounting ext3 file system using the ext4 subsystem
EXT4-fs (mmcblk1p2): warning: mounting fs with errors, running e2fsck is recommended
EXT4-fs (mmcblk1p2): recovery complete
EXT4-fs (mmcblk1p2): mounted filesystem with ordered data mode. Opts: (null)
VFS: Mounted root (ext3 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 1024K
EXT4-fs (mmcblk1p2): re-mounted. Opts: data=ordered
random: sshd: uninitialized urandom read (32 bytes read)
atmel_usart_serial atmel_usart_serial.1.auto: using dma0chan5 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.1.auto: using dma0chan6 for tx DMA transfers
Poky (Vocto Project Reference Distro) 2.5.2 rugged-board-asd2x-sd1 /dev/ttyS0

rugged-board-asd2x-sd1 login: root
root@rugged-board-asd2x-sd1:~# hello_exp1
Hello World by prabanya and team
root@rugged-board-asd2x-sd1:~#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0
```

RESULT:

EX NO: 02
DATE:

Explore and modify local, machine, and distro configuration files.

AIM:

To create a simple Yocto project with basic configuration. Explore and modify local, machine, and distro configuration files.

ALGORITHM:

STEP 1: Start the process.

STEP 2: Pull and run an Ubuntu container.

STEP 3: Install the packages needed for yocto.

STEP 4: Create a directory 'yocto-a5d2x' in home directory, create a new directory 'source' and clone a git repositories,

```
# git clone https://github.com/rugged-board/meta-rba5d2x.git -b sumo-rba5d2x
```

```
# git clone https://github.com/rugged-board/poky.git -b sumo-rba5d2x
```

```
# git clone git://git.openembedded.org/meta-openembedded -b sumo
```

```
# git clone https://github.com/intel-iot-devkit/meta-iot-cloud.git -b sumo
```

STEP 5: Change the directory to poky and perform the command,

```
# source oe-init-build-env
```

STEP 6: SD Card Flash Images:

For rugged board a5d2x(SDCARD)

i) Change the machine name to "rugged-board-a5d2x-sd1" in "conf/local.conf" as below.

```
$ vi conf/local.conf
```

ii) Set the machine as below and save the file.

```
MACHINE ?= "rugged-board-a5d2x-sd1"
```

iii) Compile the images for **SDCARD Flash** using below command.

```
$ bitbake rb-sd-core-image-minimal
```


iv) After completion of this compiling please go to below path to get the **SDCARD Flash** images.

\$ cd tmp/deploy/images/rugged-board-a5d2x-sd1/

STEP 7: NOR Flash Images:

For rugged board a5d2x(NOR)

i) Change machine name to "rugged-board-a5d2x" in conf/local.conf as below.

\$ vi conf/local.conf

ii) Set the machine as below and save the file.

MACHINE ?= "rugged-board-a5d2x"

iii) Compile the images for **NOR Flash** using below command.

\$ bitbake rb-nor-core-image-minimal

iv) After completion of this compiling, please go to below path to get the **NOR Flash** images.

\$ cd tmp/deploy/images/rugged-board-a5d2x/

STEP 8: Change the Distro on local.conf,

DISTRO ?= "poky-tiny"

STEP 8: And finally build your image and flash the Rugged Board.

STEP 10: Stop the Process.

PROGRAM:

local.conf:

demonstration purposes:

Below machine is for rugged-board-a5d2x(SDCARD Flash)

MACHINE ?= "rugged-board-a5d2x-sd1"

#

This sets the default machine to be rugged-board-a5d2x(NOR Flash) if no other machine is selected:

#MACHINE ??= "rugged-board-a5d2x"

The distribution setting controls which policy settings are used as defaults.

The default value is fine for general Yocto project use, at least initially.

Ultimately when creating custom policy, people will likely end up subclassing

these defaults.

```
#
DISTRO ?= "poky-tiny"
# As an example of a subclass there is a "bleeding" edge policy configuration
# where many versions are set to the absolute latest code from the upstream
# source control systems. This is just mentioned here as an example, its not
# useful to most new users.
# DISTRO ?= "poky-bleeding"
```

OUTPUT:

```
Terminal
Sep 15 18:11
root@584a332c0218: /home/yocto_rba5d2x/sources/poky

rathinavel@rathinavel-ASUS-TUF-Gaming-A15-FA506IHRB-FA506IHRZ:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
31a15e21cccf   ubuntu:16.04   "/bin/bash"             31 minutes ago   Exited (129) 13 minutes ago           kind_tesla
0969f5786208   ubuntu:16.04   "/bin/bash"             26 hours ago    Up 2 hours                               agitated_booth
584a332c0218   ubuntu:16.04   "/bin/bash"             2 months ago    Up About an hour                       competent_gates

rathinavel@rathinavel-ASUS-TUF-Gaming-A15-FA506IHRB-FA506IHRZ:~$ sudo docker exec -it 584a332c0218 /bin/bash
root@584a332c0218:/# cd home/yocto_rba5d2x/sources/
root@584a332c0218:/home/yocto_rba5d2x/sources# ls
meta-tot-cloud meta-openembedded meta-rba5d2x poky
root@584a332c0218:/home/yocto_rba5d2x/sources# cd poky/
root@584a332c0218:/home/yocto_rba5d2x/sources/poky# ls
bitbake build documentation LICENSE meta meta-poky meta-selftest meta-skeleton meta-yocto-bsp oe-init-build-env
root@584a332c0218:/home/yocto_rba5d2x/sources/poky# source oe-init-build-env
### Welcome to the RUGGED BOARD A5D2X BSP-Yocto buildsyste.. ###

You can now run 'bitbake <target>'

for rugged board a5d2x(SDCARD)
change machine name to "rugged-board-a5d2x-sd1" in conf/local.conf

$ bitbake rb-sd-core-image-minimal

for rugged board a5d2x(NOR)
change machine name to "rugged-board-a5d2x" in conf/local.conf

$ bitbake rb-nor-core-image-minimal

Common targets are below:
core-image-minimal
core-image-sato
meta-toolchain
meta-ide-support
```

Rugged Board A5d2x

```
root@584a332c0218:/home/yocto_rba5d2x/sources/poky/build# ls
bitbake-cookerdaemon.log cache conf tmp
root@584a332c0218:/home/yocto_rba5d2x/sources/poky/build# cd conf
root@584a332c0218:/home/yocto_rba5d2x/sources/poky/build/conf# ls
bblayers.conf local.conf templateconf.cfg
root@584a332c0218:/home/yocto_rba5d2x/sources/poky/build/conf# vi local.conf
```

```
#
# Machine Selection
#
# You need to select a specific hardware board target machines included for
# demonstration purposes:
# Below machine is for rugged-board-a5d2x(SDCARD Flash)
#MACHINE ?= "rugged-board-a5d2x-sd1"
#
# This sets the default machine to be rugged-board-a5d2x(NOR Flash) if no other machine is selected:
MACHINE ??= "rugged-board-a5d2x"
#
# Where to place downloads
#
# During a first build the system will download many different source code tarballs
# from various upstream projects. This can take a while, particularly if your network
# connection is slow. These are all stored in DL_DIR. When wiping and rebuilding you
# can preserve this directory to speed up this part of subsequent builds. This directory
# is safe to share between multiple builds on the same machine too.
#
# The default is a downloads directory under TOPDIR which is the build directory.
#DL_DIR ?= "${TOPDIR}/downloads"
#
# Default policy config
#
# The distribution setting controls which policy settings are used as defaults.
# The default value is fine for general Yocto project use, at least initially.
# Ultimately when creating custom policy, people will likely end up subclassing
# these defaults.
DISTRO ?= "poky-tiny"
# As an example of a subclass there is a "bleeding" edge policy configuration
# where many versions are set to the absolute latest code from the upstream
# source control systems. This is just mentioned here as an example, its not
# useful to most new users.
# DISTRO ?= "poky-bleeding"
#
# Package Management configuration
#
# This variable lists which packaging formats to enable. Multiple package backends
# can be enabled at once and the first item listed in the variable will be used
```

RESULT:

EX NO: 03 DATE:	Create mycalci recipe in meta-rb layer having source files app.c, add.c, mul.c, div.c, sub.c and myheader.h (take 3 command line arguments <num1> <operator> <num2>) fetch these from local source and then add recipe as a package in phy-image after that build and test this package.
----------------------------------	--

AIM:

To create mycalci recipe in meta-rb layer having source files app.c, add.c, mul.c, div.c, sub.c and myheader.h fetch these from local source and then add recipe as a package in phy-image after that build and test this package.

ALGORITHM:

STEP 1: First we create the Recipe for your project.

STEP 2: There are some basic commands are used for create recipe,

STEP 3: Create the Recipe Directory:
`mkdir -p meta-custom/recipes-example/mycalci`

STEP 4: Next, we write the new Recipe File.
 Ex: mycalci_1.bb
 "app.c, add.c, sub.c, mul.c, div.c, myheader.h "

STEP 5: After write the recipe file, add the License File: LICENSE = "MIT".

STEP 6: Next, we Add the Recipe to Your Image.

STEP 7: After finish the all above process we should Build the Image by using the "bitbake core-image-minimal" command.

STEP 8: Once the build is complete, deploy your image to your target device or emulator.

STEP 9: After deploying the image, SSH into your target device or access the terminal of your emulator.

STEP 10: finally run the application.

STEP 11: Run the mycalci application to test it.

PROGRAM:

app.c

```
#include<stdio.h>
#include"myheader.h"
```

```

int main(int argc,char *argv[]){

    if(argc !=4){
        printf("Usage: %s <num1> <operator> <num2>\n",argv[0]);
        return 1;
    }

    double num1 = atof(argv[1]);
    char operator = argv[2][0];
    double num2 = atof(argv[3]);
    printf("A = %.2lf\n",num1);
    printf("B = %.2lf\n",num2);
    double result;

    switch(operator){
        case '+':
            result = add(num1, num2);
            break;
        case '-':
            result = sub(num1, num2);
            break;
        case '*':
            result = mul(num1, num2);
            break;
        case '/':
            result = division(num1, num2);
            break;
        default:
            printf("Invalid Operator");
            return 1;
    }

    printf("Result is '%c' operator :%.2lf\n",operator,result);
    return 0;
}

```

add.c

```

double add(double a,double b){
    return a+b;
}

```

sub.c

```

double sub(double a,double b){
    return a-b;
}

```

mul.c

```
double mul(double a,double b){  
    return a*b;  
}
```

div.c

```
#include<stdio.h>  
  
double divsion(double a,double b){  
  
    if(b !=0){  
        return a/b;  
    }  
    else{  
        printf("Can not Divisible by zero \n");  
        return 0;  
    }  
}
```

myheader.h

```
double add(double a,double b);  
double sub(double a,double b);  
double mul(double a,double b);  
double divsion(double a,double b);
```

OUTPUT:

```
rugged-board-a5d2x-sd1 login: root  
root@rugged-board-a5d2x-sd1:~# mycalci 20+30  
Usage: mycalci <num1> <operator> <num2>  
root@rugged-board-a5d2x-sd1:~# mycalci 20 + 30  
Result: 50.000000  
root@rugged-board-a5d2x-sd1:~# mycalci 20 - 30  
Result: -10.000000  
root@rugged-board-a5d2x-sd1:~# mycalci 20 "*" 30  
Result: 600.000000  
root@rugged-board-a5d2x-sd1:~# mycalci 20 / 30  
Result: 0.666667  
root@rugged-board-a5d2x-sd1:~# █
```

RESULT:

EX NO: 04 DATE:	Push your mycalci source to your github account then write the recipe mygit for fetching the source from git and then compile and install and generate a package and this package in phy-image and then build and test.
----------------------------------	--

AIM:

To create recipes in yocto and push the github account and compile the recipes.

ALGORITHM:

STEP 1: Create a C Code & Make file.

STEP 2: Run The Code.

STEP 3: Create a New Repository in Github Account.

STEP 4: Push the Recipe in Github Repository.

STEP 5: Set Up Yocto Environment.

STEP 6: Create a layer name meta-git by using command below.

Command 1.] bitbake-layers create-layer ../sources/meta-git

2.] bitbake-layers add-layer ../sources/meta-git

STEP 7: After create layer navigate to meta-git dir -> recipes-example -> create git directory

STEP 8: inside the git directory write the recipes with extension “.bb” inside the bb file write the code lines to process the c files inside the git repository.

STEP 9: Then compile the .bb file by using command “bitbake git”.

PROGRAM:

```
#include <stdio.h>
```

```
void add() {
```

```
    float a, b;
```

```
    printf("Enter two numbers: ");
```

```
    scanf("%f %f", &a, &b);
```

```
    printf("Result: %.2f\n", a + b);
```

```
}
```

```
void subtract() {
```

```

float a, b;

printf("Enter two numbers: ");
scanf("%f %f", &a, &b);
printf("Result: %.2f\n", a - b);
}

void multiply() {
    float a, b;
    printf("Enter two numbers: ");
    scanf("%f %f", &a, &b);
    printf("Result: %.2f\n", a * b);
}

void divide() {
    float a, b;
    printf("Enter two numbers: ");
    scanf("%f %f", &a, &b);
    if (b != 0) {
        printf("Result: %.2f\n", a / b);
    } else {
        printf("Error: Division by zero is not allowed.\n");
    }
}

int main() {
    int choice;
    do {
        printf("\nSimple Calculator\n");
        printf("1. Add\n");
        printf("2. Subtract\n");
        printf("3. Multiply\n");
        printf("4. Divide\n");
        printf("5. Exit\n");
        printf("Choose an option: ");

```

```

scanf("%d", &choice);
switch (choice) {
    case 1: add(); break;
    case 2: subtract(); break;
    case 3: multiply(); break;
    case 4: divide(); break;
    case 5: printf("Exiting...\n"); break;
    default: printf("Invalid choice. Please try again.\n");
}
} while (choice != 5);
return 0;
}

```

Make file:

```

CC = gcc
CFLAGS = -Wall -Werror
SRC = calculator.c
UT = calculator
all: $(OUT)
$(OUT): $(SRC)
        $(CC) $(CFLAGS) -o $(OUT) $(SRC)
clean:
        rm -f $(OUT)
.PHONY: all clean

```

.bb file:

```

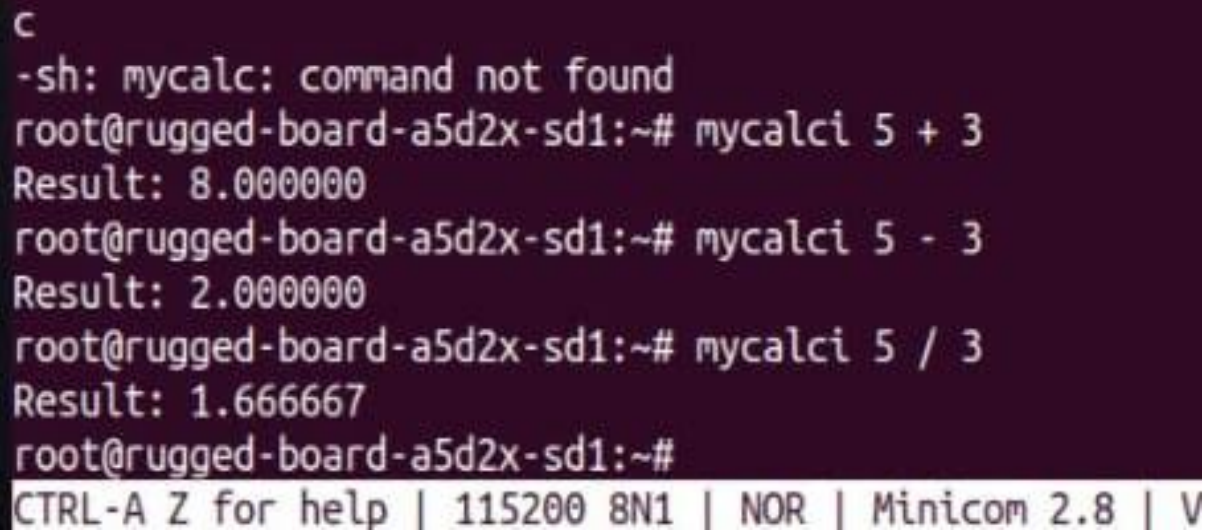
DESCRIPTION = "Simple calculator application"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
# Use SRC_URI for fetching the source
SRC_URI = "git://github.com/Surendar7550/mycalci.git;protocol=https;branch=main"

```



```
SRCREV = "e3e7af6c3974521864dfebe5d4e9b0d125024fc2"
S = "${WORKDIR}/git"
do_compile() {
    cd ${S}
    ${CC} ${CFLAGS} -Wl,--hash-style=gnu -o calculator calculator.c
}
do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${S}/calculator ${D}${bindir}/
}
FILES_${PN} = "${bindir}/calculator"
```

OUTPUT:

A terminal window with a dark purple background and light green text. The prompt is 'C'. The user enters '-sh: mycalc: command not found'. Then, the user enters 'root@rugged-board-a5d2x-sd1:~# mycalci 5 + 3' and the output is 'Result: 8.000000'. Next, the user enters 'root@rugged-board-a5d2x-sd1:~# mycalci 5 - 3' and the output is 'Result: 2.000000'. Then, the user enters 'root@rugged-board-a5d2x-sd1:~# mycalci 5 / 3' and the output is 'Result: 1.666667'. Finally, the user enters 'root@rugged-board-a5d2x-sd1:~#'. At the bottom, there is a status bar with the text 'CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.8 | V'.

```
C
-sh: mycalc: command not found
root@rugged-board-a5d2x-sd1:~# mycalci 5 + 3
Result: 8.000000
root@rugged-board-a5d2x-sd1:~# mycalci 5 - 3
Result: 2.000000
root@rugged-board-a5d2x-sd1:~# mycalci 5 / 3
Result: 1.666667
root@rugged-board-a5d2x-sd1:~#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.8 | V
```

RESULT:

EX NO: 05 DATE:	Create a Static Library recipe mystatic having a source and header files for basic calculator application. In do_install() static lib (.a) add in {libdir} and header file add in {includedir} after that add package in phy-image and then build & test.
----------------------------------	---

AIM:

To create a static library named mystatic containing source and header files for a basic calculator application. This library will be installed in the appropriate directories ({libdir} for the .a file and {includedir} for the header file), added to the target image (phy-image), and then built and tested.

ALGORITHM:

STEP 1: Create Source Files:

- Create the source file calculator.c for the basic calculator functions (e.g., add, subtract, multiply, divide).
- Create the header file calculator.h for function declarations and constants.

STEP 2: Create Recipe Directory:

- Create a new directory for the mystatic recipe (e.g., recipes-mystatic).

STEP 3: Create mystatic.bb File:

- Define the static library recipe (mystatic.bb), specifying the source files and build instructions.

STEP 4: Define do_compile():

- Use the appropriate compilation commands to build the static library (calculator.a).

STEP 5: Define do_install():

- Install the static library (calculator.a) to {libdir}.
- Install the header file (calculator.h) to {includedir}.

STEP 6: Add to phy-image:

- Add mystatic to the IMAGE_INSTALL in the target image recipe (phy-image).

STEP 7: Build the Image:

- Run bitbake phy-image to build the image with the static library.

STEP 8: Test the Library:

- Create a simple test application to link against the calculator.a static library and ensure it works.

STEP 9: Verify Installation:

- Ensure the static library and header file are correctly installed in {libdir} and {includedir} in the target image.

STEP 10: Run the Test:

- Test the target image to verify the calculator library functions are correctly integrated and working.

OUTPUT:

```
Summary: There were 7 WARNING messages shown.
root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/cortexa5hf-neon-poky-linux-musleabi/mystatic/0.0-r0#
ls
bitbake-cookerdaemon.log  pkgdata          sstate-install-packagedata  sysroot-destdir
deploy-rpms              pseudo          sstate-install-package_qa   temp
license-destdir         recipe-sysroot  sstate-install-package_write_rpm
package                 recipe-sysroot-native  sstate-install-populate_lic
packages-split          sstate-install-package  sstate-install-populate_sysroot
root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/cortexa5hf-neon-poky-linux-musleabi/mystatic/0.0-r0#
tree packages-split/
packages-split/
├── mystatic
├── mystatic-dbg
├── mystatic-dev
│   └── usr
│       └── include
│           └── calculator.h
├── mystatic-doc
├── mystatic-locale
├── mystatic-staticdev
│   └── usr
│       └── lib
│           └── libcalculator.a
10 directories, 2 files
root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/cortexa5hf-neon-poky-linux-musleabi/mystatic/0.0-r0#
^C
root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/cortexa5hf-neon-poky-linux-musleabi/mystatic/0.0-r0#
```

```
Activities Terminal Nov 6 15:49
rameshchlm@rameshkumar: ~/local/share/Trash/files/rugged-board-a5d2x-sd1.2
root@rugged-board-a5d2x-sd1:~# ls /usr/include/
aio.h          byteswap.h      dirent.h        features.h      h
alloca.h       c++             dlfcn.h        fenv.h         h
ar.h           calc.h          drm             float.h         o
arpa           calculator.h     elf.h          fmtmsg.h        h
asm            complex.h       endian.h        fnmatch.h       h
asm-generic    cpio.h          errno.h         ftw.h           h
assert.h       crypt.h          errno.h         getopt.h        h
bits           ctype.h          fcntl.h         glob.h          n
root@rugged-board-a5d2x-sd1:~# ls /usr/lib/libcalculator.a
/usr/lib/libcalculator.a
root@rugged-board-a5d2x-sd1:~#
```

```
Activities Terminal Nov 6 15:48
rameshchlm@rameshkumar: ~/local/share/Trash/files/rugged-board-a5d2x-sd1.2
root@rugged-board-a5d2x-sd1:~# ls /usr/include/
aio.h          byteswap.h      dirent.h        features.h      h
alloca.h       c++             dlfcn.h        fenv.h         h
ar.h           calc.h          drm             float.h         o
arpa           calculator.h     elf.h          fmtmsg.h        h
asm            complex.h       endian.h        fnmatch.h       h
asm-generic    cpio.h          errno.h         ftw.h           h
assert.h       crypt.h          errno.h         getopt.h        h
bits           ctype.h          fcntl.h         glob.h          n
root@rugged-board-a5d2x-sd1:~#
```

RESULT:

EX NO: 06 DATE:	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
----------------------------------	--

AIM:

To 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.

ALGORITHM:

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

STEP 3: 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 using vim editor ,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 ,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

STEP 10: Then start build process

home->yocto->build
give the commands:
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

STEP 16: - gcc test.c -o test -I/usr/include -L/usr/lib -lcalculator

./test

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

STEP 17: Stop the program and finished.

PROGRAM:

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;
}
```

mydynamic_0.1.bb:

```
DESCRIPTION = "A dynamic library for basic calculator functions"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2
f7b4f302"
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"

```

phy-image.bb:

```

DESCRIPTION = "Custom image including mydynamic"
LICENSE = "MIT"
IMAGE_INSTALL += "mydynamic"
IMAGE_INSTALL += "mydynamic mydynamic-dev"

```

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;
}

```


OUTPUT:

```
Activities Terminal Sep 15 19:27 sasar@sasar-HP-Laptop-15s-fq4xxx: ~/snap/snap-store
#endif

root@rugged-board-a5d2x-sd1:~# cd /usr/include
root@rugged-board-a5d2x-sd1:~# gcc test.c -o test -I/usr/include -L/usr/lib -lcalculator
root@rugged-board-a5d2x-sd1:~# ./test
Add: 15
Subtract: 5
Multiply: 50
Divide: 2
root@rugged-board-a5d2x-sd1:~#
```

RESULT:

EX NO: 07
DATE:

Write a GNU helloworld autotool recipe to add GNU helloworld package in phy-image and then build & test.

AIM:

To write a GNU helloworld autotool recipe to add GNU helloworld package in phy-image and then build & test.

ALGORITHM:

STEP 1: Start the process.

STEP 2: Pull and run an Ubuntu container.

STEP 3: Install the packages needed for yocto.

STEP 4: Create a directory(meta-example) which will house your BSP development.

STEP 5: Download and install GNU auto-tools config and intergrate to the layer

STEP 6: Create a recipe for the project

STEP 7: Change the directory to poky and Set up the build environment by sourcing the
“ oe-init-build-env ” script.

STEP 8: Run bitbake phy-image in build

STEP 9: Stop the process.

PROGRAM:

RECIPIE FILE

auto_hello.bb:

```
DESCRIPTION = "Simple helloworld application"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
SRC_URI = "file://auto-hello-1.0.tar.gz"
#S = "${WORKDIR}/auto-hello-1.0"
#PROVIDES += "myphy"
inherit autotools
```

The Major files inside auto hello files tar.gz

userprog.c

```
#include<stdio.h>
int main()
{
    printf("Hello world\n");
    return 0;
}
```

configure.ac:

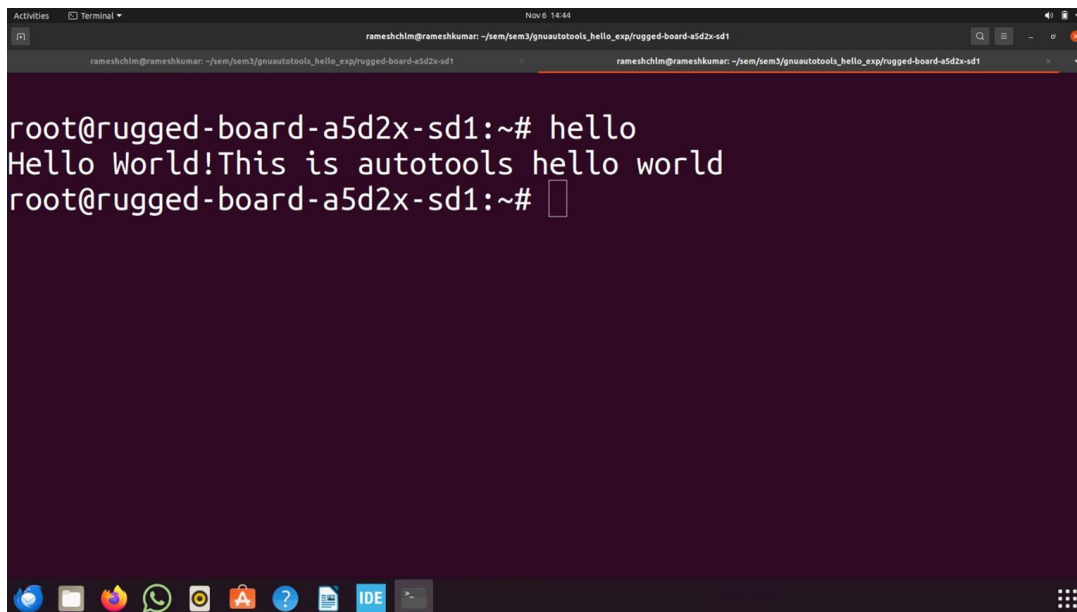
```
AC_INIT([auto-hello], [1.0], [panidharece2023@gmail.com])
AM_INIT_AUTOMAKE([-Wall -Werror foreign])
AC_PROG_CC
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Makefile.am:

```
bin_PROGRAMS = myhello
```

```
myhello_SOURCES = userprog.c
```

OUTPUT:

A screenshot of a terminal window with a dark purple background. The terminal shows the prompt 'root@rugged-board-a5d2x-sd1:~#' followed by the command 'hello'. The output is 'Hello World!This is autotools hello world'. The prompt is followed by a cursor. The terminal window has a title bar with 'Activities', 'Terminal', and the date 'Nov 6 14:44'. The bottom of the screen shows a dock with various application icons including a globe, a folder, a terminal, a chat bubble, a shopping bag, a question mark, a document, and an IDE icon.

```
root@rugged-board-a5d2x-sd1:~# hello
Hello World!This is autotools hello world
root@rugged-board-a5d2x-sd1:~#
```

RESULT:

EX NO: 08
DATE:

Experiment with character device driver in YOCTO

AIM:

Experiment with character device driver in YOCTO.

ALGORITHM:

STEP 1: Start the process.

STEP 2: path for kernel source in yocto to write a device driver code is is
root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/rugged_board_a5d2x_
sd1-poky-linux-musleabi/linux-rba5d2x/4.9-r0/git/drivers/char/mydriver/#

STEP 3: Create C file 'mychardriver.c' and write the code

'vim mychardriver.c'.

STEP 4: @Makefile -> add this

obj-\$(@Kconfig -> add the content

menuconfig MYCHARDRIVER

tristate "My Character Device Driver"

default n

help

Enable support for My Character Device Driver.

if MYCHARDRIVER

config MYCHARDRIVER_ENABLE

tristate "Enable My Character Device Driver"

default m

help

This enables the mychardriver character device driver.

endif

root@af642904d433:/home/yocto_rba5d2x/build/tmp/work/rugged_board_a5d2x_sd1-poky-linux-musleabi/linux-rba5d2x/4.9-r0/git/drivers/char/

@Makefile

obj-\$(CONFIG_MYCHARDRIVER) += mychardriver.o

@Kconfig

config MYCHARDRIVER

tristate "Enable My Character Device Driver"

depends on SOME_DEPENDENCY

default n

help

This option enables the My Character Device Driver.

The driver provides access to a simple character device.

```
# Source additional configurations from another Kconfig filesource
"drivers/char/mydriver/Kconfig" _MYCHARDRIVER_ENABLE) +=
my_char_device.o.
```

STEP 5: bitbake -c menuconfig virtual/kernel.

STEP 6: Stop the process.

PROGRAM:

mychardriver.c

```
#define MAX_DEV 2
```

```
#define BUFFER_SIZE 128
```

```
static int mychardev_open(struct inode *inode, struct file *file);
```

```
static int mychardev_release(struct inode *inode, struct file *file);
```

```
static long mychardev_ioctl(struct file *file, unsigned int cmd, unsigned long arg);
```

```
static ssize_t mychardev_read(struct file *file, char __user *buf, size_t count, loff_t *offset);
```

```
static ssize_t mychardev_write(struct file *file, const char __user *buf, size_t count, loff_t *offset);
```

```

static const struct file_operations mychardev_fops = {
    .owner      = THIS_MODULE,
    .open       = mychardev_open,
    .release    = mychardev_release,
    .unlocked_ioctl = mychardev_ioctl,
    .read       = mychardev_read,
    .write      = mychardev_write,
};

struct mychar_device_data {
    struct cdev cdev;
    char *buffer; // To hold user data
    struct mutex mutex; // For synchronization
};

static int dev_major = 0;
static struct class *mychardev_class = NULL;
static struct mychar_device_data mychardev_data[MAX_DEV];

static int mychardev_uevent(struct device *dev, struct kobj_uevent_env *env) {
    add_uevent_var(env, "DEVMODE=%#o", 0666);
    return 0;
}

static int __init mychardev_init(void) {
    int err, i;
    dev_t dev;

    err = alloc_chrdev_region(&dev, 0, MAX_DEV, "mychardev");
    if (err < 0) {
        printk(KERN_ERR "Failed to allocate char device region\n");
        return err;
    }
    dev_major = MAJOR(dev);

```

```

mychardev_class = class_create(THIS_MODULE, "mychardev");
if (IS_ERR(mychardev_class)) {
    unregister_chrdev_region(dev, MAX_DEV);
    return PTR_ERR(mychardev_class);
}
mychardev_class->dev_uevent = mychardev_uevent;
for (i = 0; i < MAX_DEV; i++) {
    cdev_init(&mychardev_data[i].cdev, &mychardev_fops);
    mychardev_data[i].cdev.owner = THIS_MODULE;

    mychardev_data[i].buffer = kmalloc(BUFFER_SIZE, GFP_KERNEL);
    if (!mychardev_data[i].buffer) {
        printk(KERN_ERR "Failed to allocate memory for device %d\n", i);
        while (--i >= 0) {
            kfree(mychardev_data[i].buffer);
            cdev_del(&mychardev_data[i].cdev);
        }
        class_destroy(mychardev_class);
        unregister_chrdev_region(dev, MAX_DEV);
        return -ENOMEM;
    }
    mutex_init(&mychardev_data[i].mutex);
    cdev_add(&mychardev_data[i].cdev, MKDEV(dev_major, i), 1);
    device_create(mychardev_class, NULL, MKDEV(dev_major, i), NULL, "mychardev-
%d", i);
}
printk(KERN_INFO "My character device driver initialized\n");
return 0;
}

static void __exit mychardev_exit(void) {
    int i;

```

```

for (i = 0; i < MAX_DEV; i++) {
    device_destroy(mychardev_class, MKDEV(dev_major, i));
    kfree(mychardev_data[i].buffer);
    cdev_del(&mychardev_data[i].cdev);
}
class_unregister(mychardev_class);
class_destroy(mychardev_class);
unregister_chrdev_region(MKDEV(dev_major, 0), MAX_DEV);

printk(KERN_INFO "My character device driver exited\n");
}

static int mychardev_open(struct inode *inode, struct file *file) {
    struct mychar_device_data *dev_data = container_of(inode->i_cdev, struct
mychar_device_data, cdev);
    file->private_data = dev_data;
    printk(KERN_INFO "MYCHARDEV: Device open\n");
    return 0;
}

static int mychardev_release(struct inode *inode, struct file *file) {
    printk(KERN_INFO "MYCHARDEV: Device close\n");
    return 0;
}

static long mychardev_ioctl(struct file *file, unsigned int cmd, unsigned long arg) {
    printk(KERN_INFO "MYCHARDEV: Device ioctl\n");
    return 0;
}

static ssize_t mychardev_read(struct file *file, char __user *buf, size_t count, loff_t *offset) {
    struct mychar_device_data *dev_data = file->private_data;
    size_t datalen = strlen(dev_data->buffer);

    if (*offset >= datalen) {

```



```

        return 0; // End of file
    }
    if (count > datalen - *offset) {
        count = datalen - *offset; // Adjust count to remaining data
    }
    if (copy_to_user(buf, dev_data->buffer + *offset, count)) {
        return -EFAULT;
    }
    *offset += count; // Update offset for the next read
    printk(KERN_INFO "MYCHARDEV: Read %zu bytes\n", count);
    return count;
}

static ssize_t mychardev_write(struct file *file, const char __user *buf, size_t count, loff_t
*offset) {
    struct mychar_device_data *dev_data = file->private_data;
    size_t maxdatalen = BUFFER_SIZE - 1; // Reserve space for null terminator
    if (count > maxdatalen) {
        count = maxdatalen; // Limit the number of bytes to write
    }
    mutex_lock(&dev_data->mutex); // Lock for synchronization

    // Clear the buffer before copying to prevent garbage values
    memset(dev_data->buffer, 0, BUFFER_SIZE);

    if (copy_from_user(dev_data->buffer, buf, count)) {
        mutex_unlock(&dev_data->mutex);
        return -EFAULT;
    }
    dev_data->buffer[count] = '\0'; // Null-terminate the string
    *offset += count; // Update offset
    printk(KERN_INFO "MYCHARDEV: Wrote %zu bytes: %s\n", count, dev_data->buffer);
}

```

```

mutex_unlock(&dev_data->mutex); // Unlock after the operation

return count;
}

MODULE_LICENSE("GPL");

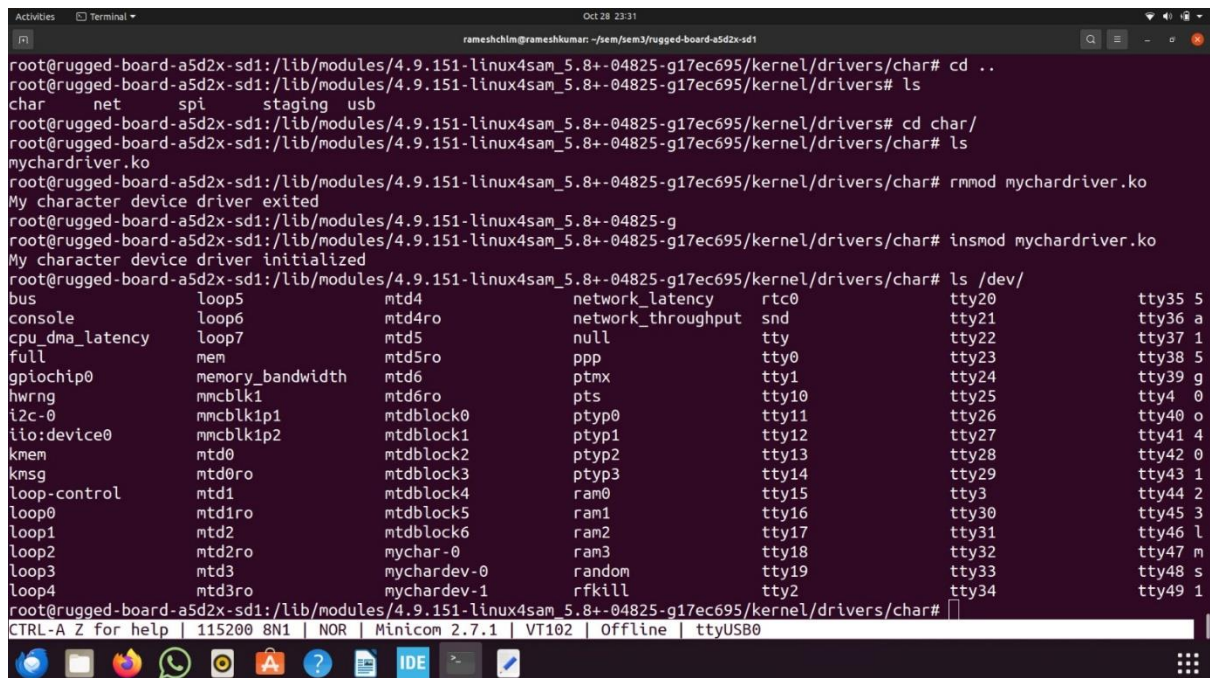
MODULE_AUTHOR("Rameshkumar_Hari_BelthaArthi");

module_init(mychardev_init);

module_exit(mychardev_exit);

```

OUTPUT:



```

root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char# cd ..
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers# ls
char  net  spi  staging  usb
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers# cd char/
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char# ls
mychardriver.ko
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char# rmmod mychardriver.ko
My character device driver exited
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char# insmod mychardriver.ko
My character device driver initialized
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char# ls /dev/
bus          loop5          mtd4          network_latency  rtc0          tty20          tty35 5
console      loop6          mtd4ro        network_throughput  snd          tty21          tty36 a
cpu_dma_latency  loop7          mtd5          null            tty          tty22          tty37 1
full         mem            mtd5ro        ppp             tty0         tty23          tty38 5
gpiochip0    memory_bandwidth  mtd6          ptmx            tty1         tty24          tty39 g
hwrng        mmcblk1        mtd6ro        pts             tty10        tty25          tty4 0
i2c-0        mmcblk1p1      mtdblock0     ptyp0           tty11        tty26          tty40 o
iio:device0  mmcblk1p2      mtdblock1     ptyp1           tty12        tty27          tty41 4
kmem         mtd0           mtdblock2     ptyp2           tty13        tty28          tty42 0
kmsg         mtd0ro         mtdblock3     ptyp3           tty14        tty29          tty43 1
loop-control  ntd1           mtdblock4     ram0            tty15        tty3           tty44 2
loop0        mtd1ro         mtdblock5     ram1            tty16        tty30          tty45 3
loop1        mtd2           mtdblock6     ram2            tty17        tty31          tty46 l
loop2        mtd2ro         mychar-0       ram3            tty18        tty32          tty47 m
loop3        mtd3           mychardev-0    random          tty19        tty33          tty48 s
loop4        mtd3ro         mychardev-1    rfkill          tty2         tty34          tty49 1
root@rugged-board-a5d2x-sd1:/lib/modules/4.9.151-linux4sam_5.8+-04825-g17ec695/kernel/drivers/char#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyUSB0

```

RESULT:

EX NO: 09 DATE:	In meta-rb layer create image recipe phy-image by inherit core-image bbclass and then add ssh-server-dropbear, read-only-rootfs features in your Image after adding build the image.
----------------------------------	---

AIM:

To create meta-rb layer create image recipe phy-image by inherit core-image bbclass and then add ssh-server-dropbear, read-only-rootfs features in your Image after adding build the image.

ALGORITHM:

STEP 1: Start the process.

STEP 2: Set Up the Docker container

STEP 3: In the meta-rb layer, create a directory for your custom image

STEP 4: touch meta-rb/recipes-core/images/phy-image.bb

\$ mkdir -p meta-rb/recipes-core/images

STEP 5: Create a new BitBake recipe file for phy-image

\$ touch meta-rb/recipes-core/images/phy-image_0.1.bb

STEP 6: Open phy-image.bb in a text editor and define the image, inherit core-image and Add image features “ssh-server-dropbear read-only-rootfs”

STEP 7: Creates a new recipe file for your custom image named phy-image.bb.

STEP 8: Open your build configuration file, typically conf/local.conf, in your Yocto build directory and Add phy-image as your default image to be built by setting

IMAGE_INSTALL

IMAGE_INSTALL += "phy-image"

STEP 9: Ensure that the meta-rb layer is included in your build configuration. If it's not, add it to bblayers.conf

**/home/yocto_rba5d2x/sources/meta-rb **

STEP 10: Change the directory to poky and perform the command,

```
# source oe-init-build-env
```

STEP 11: To build the custom image,

```
# bitbake phy-image
```

STEP 12: Copy the images in the docker to host PC

```
docker cp 584a332c0218:/home/yocto_rba5d2x/build/tmp/deploy  
/images/rugged-board-a5d2x-sd1 ~/phyimage
```

STEP 13: Copy that images to SD Card and flash the Rugged Board

STEP 14: Stop the Process

PROGRAM:

phy-image_0.1.bb

```
DESCRIPTION = "Custom image recipe for phy-image with SSH and read-only root  
filesystem features"
```

```
LICENSE = "MIT"
```

```
# Inherit core-image to base the image on standard core image structure  
inherit core-image
```

```
# Add the required packages and features
```

```
IMAGE_FEATURES += "ssh-server-dropbear read-only-rootfs"
```

OUTPUT:

```
ts Terminal Sep 23 15:30
root@S84a332c0218: /home/yocto_rba5d2x

You can now run 'bitbake <target>'

for rugged board a5d2x(SDCARD)
change machine name to "rugged-board-a5d2x-sd1" in conf/local.conf

$ bitbake rb-sd-core-image-minimal

for rugged board a5d2x(NOR)
change machine name to "rugged-board-a5d2x" in conf/local.conf

$ bitbake rb-nor-core-image-minimal

Common targets are below:
core-image-minimal
core-image-sato
meta-toolchain
meta-ide-support

Rugged Board A5d2x

root@S84a332c0218: /home/yocto_rba5d2x/sources/poky/build# bitbake phy-image
WARNING: Layer ../source/meta-rb should set LAYERSERIES_COMPAT_.../source/meta-rb in its conf/layer.conf file to
Loading cache: 100% |#####|
Loaded 3443 entries from dependency cache.
Parsing recipes: 100% |#####|
Parsing of 2462 .bb files complete (2461 cached, 1 parsed). 3444 targets, 358 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION      = "1.38.0"
BUILD_SYS      = "x86_64-linux"
NATIVELSBSTRING = "universal"
TARGET_SYS     = "arm-poky-linux-musleabi"
MACHINE        = "rugged-board-a5d2x"
DISTRO         = "poky-tiny"
DISTRO_VERSION  = "2.5.2"
TUNE_FEATURES  = "arm armv7a vfp thumb neon callconvention-hard cortexa5"
TARGET_FPU     = "hard"
meta
meta-poky
meta-yocto-bsp = "sumo-rba5d2x:e1960cc3bb6682f1f6d509f7d548cf0641e01063"
meta-rba5d2x   = "sumo-rba5d2x:effe26c91811a0ec3525af8c9d3cd7fa1df52390"
meta-oe
meta-python
meta-networking = "sumo:8760facba1bceb299b3613b8955621ddaa3d4c3f"
meta-iot-cloud  = "sumo:ee92127dc2107bed1532209c1d2a0e04860563de"
meta-rb         = "sumo-rba5d2x:e1960cc3bb6682f1f6d509f7d548cf0641e01063"

Initialising tasks: 100% |#####|
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2018 tasks of which 2018 didn't need to be rerun and all succeeded.
```

```
ts Terminal Sep 23 15:33
root@S84a332c0218: /home/yocto_rba5d2x/sources/poky/build/tmp/deploy/images/rugged-board-a5d2x

root@S84a332c0218: /home/yocto_rba5d2x/sources/poky/build/tmp/deploy/images/rugged-board-a5d2x# ls
a5d2x-rugged_board.dtb      phy-image-rugged-board-a5d2x-20240919072926.rootfs.tar.gz      u-boot-rugged-board-a5d2x.bin
at91bootstrap.bin          phy-image-rugged-board-a5d2x-20240919072926.testdata.json      u-boot-rugged-board-a5d2x-v2018.07-at91+gitAUTOINC+ae7f
at91bootstrap-rugged_board_a5d2x.bin  phy-image-rugged-board-a5d2x.cpio.gz                          zImage
BOOT.BIN                   phy-image-rugged-board-a5d2x.manifest                          zImage--4.9-r0-a5d2x-rugged_board-20240919072926.dtb
modules--4.9-r0-rugged-board-a5d2x-20240919072926.tgz  phy-image-rugged-board-a5d2x.squashfs                          zImage--4.9-r0-rugged-board-a5d2x-20240919072926.bin
modules-rugged-board-a5d2x.tgz        phy-image-rugged-board-a5d2x.tar.gz                            zImage-a5d2x-rugged_board.dtb
phy-image-rugged-board-a5d2x-20240919072926.rootfs.cpio.gz  phy-image-rugged-board-a5d2x.testdata.json                     zImage-rugged-board-a5d2x.bin
phy-image-rugged-board-a5d2x-20240919072926.rootfs.manifest  rugged_board_a5d2x-qspiboot-u-boot-3.8.12+gitAUTOINC+15d9f256b7.bin
phy-image-rugged-board-a5d2x-20240919072926.rootfs.squashfs  u-boot.bin
root@S84a332c0218: /home/yocto_rba5d2x/sources/poky/build/tmp/deploy/images/rugged-board-a5d2x#
```

RESULT:

EX NO: 10 DATE:	Create the Recipe an helloworld and to Patch the file on Git using Yocto.
----------------------------------	--

AIM:

To Create the Recipe an helloworld and to Patch the file on Git using Yocto

ALGORITHM:

STEP 1: First we Create the Recipe for your project.

STEP 2: There are some basic commands are used for create recipe,

STEP 3: Create the Recipe Directory:

```
mkdir -p meta-custom/recipes-example/mycalci
```

STEP 4: clone your Github To use Remote Repository in Yocto Project

```
"git clone https://github.com//username.git"
```

STEP 5: Next, we write the new Recipe File.

```
Ex: mycalci_1.bb
```

```
"helloworld.c"
```

STEP 6: To create the .bb file then include "do_patch(),do_compaile(),do_install()" commands

STEP 7: After write the recipe file, add the License File: LICENSE = "MIT".

STEP 8: Next, we add the Recipe to Your Image.

STEP 9: After finish the all above process we should Build the Image by using the "bitbake core-image-minimal" command.

STEP 10: Once the build is complete, deploy your image to your target device or emulator.

STEP 11: After deploying the image, SSH into your target device or access the terminal of your emulator.

STEP 12: finally run the appplication.

STEP 13: Run the patch_hello application to test it.

PROGRAM:

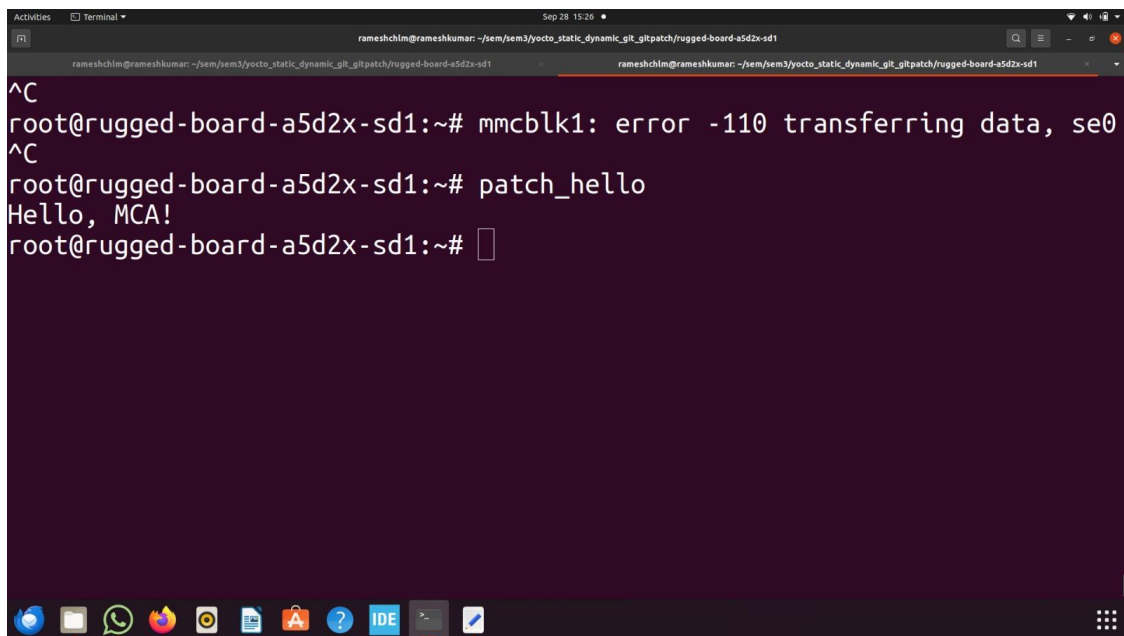
helloworld.c

```
#include<stdio.h>
int main(){
    printf("Hello, World!\n");
```

```
        return 0;
    }
patch_hello.patch

#include<stdio.h>
int main(){
    printf("Hello, World!\n");
    return 0;
}
```

OUTPUT:

A terminal window screenshot showing the execution of a patch. The terminal title is "Terminal" and the date/time is "Sep 28 15:26". The prompt is "root@rugged-board-a5d2x-sd1:~#". The user enters "mmcb1k1: error -110 transferring data, se0", which is followed by a carriage return (^C). The user then enters "patch_hello", which is followed by a carriage return (^C). The output is "Hello, MCA!". The prompt is "root@rugged-board-a5d2x-sd1:~#". The terminal has a dark background and a light-colored text. The bottom of the terminal shows a dock with various application icons.

```
^C
root@rugged-board-a5d2x-sd1:~# mmcb1k1: error -110 transferring data, se0
^C
root@rugged-board-a5d2x-sd1:~# patch_hello
Hello, MCA!
root@rugged-board-a5d2x-sd1:~#
```

RESULT: