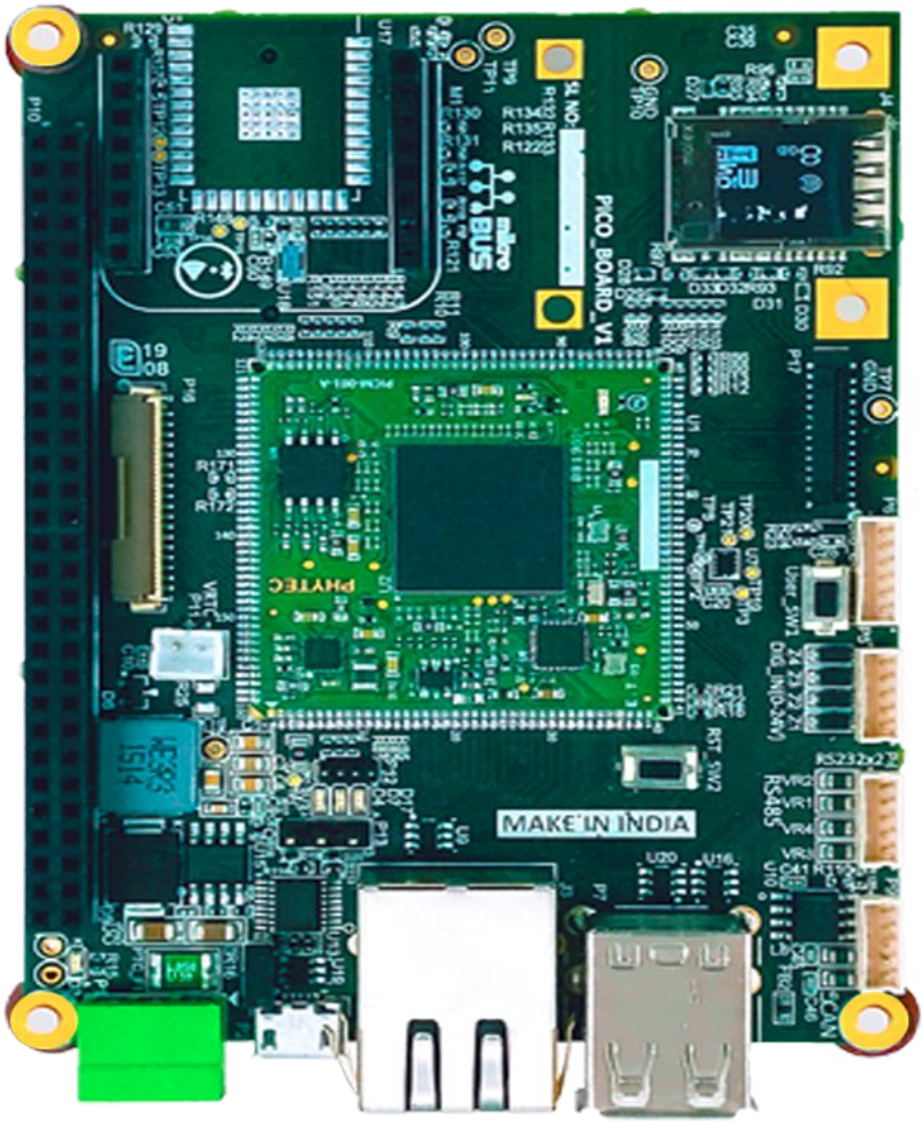
Rugged Board A5D2X - Training Manual



contents:

Linux Introduction

🌐 Historical Context: UNIX and C

Key Milestones:

- **1969**: Ken Thompson develops the first version of UNIX in assembly language on a DEC PDP-7.
- **1972-1973**: Dennis Ritchie develops the **C programming language**, and UNIX is **rewritten in C**, making it one of the first portable operating systems.

This portability was groundbreaking, as it allowed UNIX to be adapted across different hardware platforms, which was not possible with most operating systems at the time.

Growth and Fragmentation of UNIX

Since the UNIX **source code was available to universities and research institutions**, many variants were developed:

Notable UNIX Derivatives:

1. **BSD (Berkeley Software Distribution)** – From the University of California, Berkeley.
2. **System V** – Developed by AT&T as the commercial version of UNIX.
3. **SunOS**, **HP-UX**, **AIX** – Proprietary UNIX variants by Sun Microsystems, HP, and IBM respectively.

This **proliferation** led to:

- **Fragmentation** in command behavior, filesystem layout, and system APIs.
- **Incompatibilities** between systems calling themselves "UNIX".

Standardization Efforts

To reduce chaos, standards were introduced:

🛠 Major Standards:

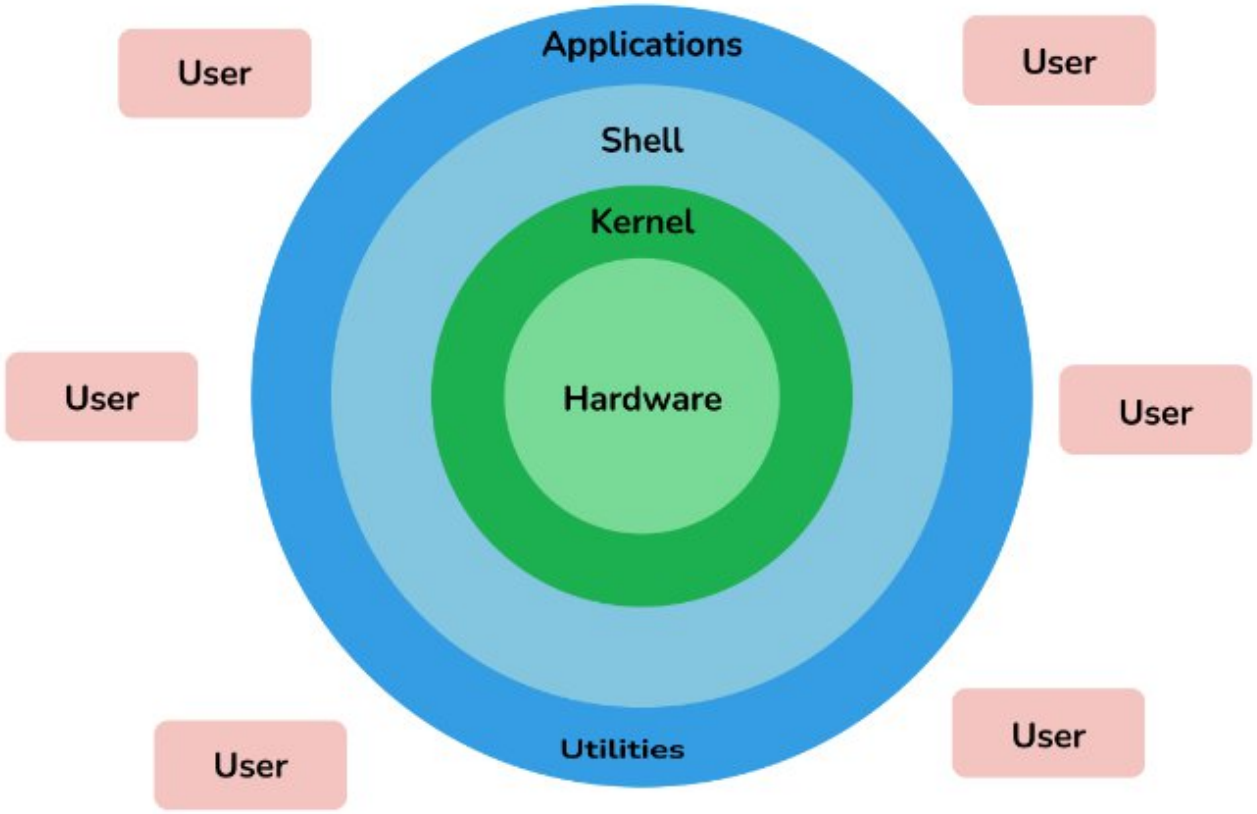
- **POSIX (Portable Operating System Interface)** – A set of IEEE standards to unify UNIX variants.
- **Single UNIX Specification (SUS)** – Maintained by The Open Group to certify compliant systems as "UNIX".

Influence on Modern OSes

Today's major operating systems are either direct descendants of UNIX or strongly influenced by it:

Operating System	UNIX Lineage
Linux	UNIX-like, inspired by UNIX but not derived from its code.
macOS	Based on BSD (specifically FreeBSD and NeXTSTEP). Certified UNIX.
Android	Based on Linux kernel (UNIX-like).

Operating System	UNIX Lineage
BSD Variants	Direct descendants of UNIX (FreeBSD, OpenBSD, NetBSD).
Solaris	Derived from System V. Certified UNIX.



Origin of the Linux Kernel

ℹ Timeline:

- **1991**: Linus Torvalds, a student at the University of Helsinki, posted a message on the **comp.os.minix** newsgroup announcing a personal project — a new free operating system kernel.
- He was initially inspired by **MINIX** (a teaching OS by Andrew Tanenbaum) but wanted something more powerful and flexible.

"Just a hobby, won't be big and professional..." — Linus Torvalds, 1991

Design Goals:

- POSIX compliance
- Free and open-source
- Portability across hardware platforms
- Monolithic kernel architecture

📈 Impact of Linux Today

Linux is now a **dominant force** across many computing domains:

Domain	Linux Penetration
Supercomputers	~97% of the **Top 500** supercomputers use Linux
Mobile Devices	~80% via **Android**, which runs on the Linux kernel
Web Servers	~70% of global web servers run Linux (Apache, Nginx)
Desktops	Millions of users (especially developers, engineers)
Embedded Systems	Routers, IoT, Rugged Board , drones, smart TVs, etc.
Automotive	Linux-based systems in **self-driving** and connected cars

Projects like **Automotive Grade Linux (AGL)** are backed by Toyota, Honda, and others.

Linux Architecture (High-Level View)

- **Monolithic kernel** (not microkernel)
- Core components include:
    - Process management
    - Memory management
    - Device drivers
    - File system support
    - Networking stack
- Modular: supports **loadable kernel modules (LKM)** for extending capabilities dynamically

✅ Linux is a *kernel* — not a complete operating system by itself.

- The **kernel** is the **core component** of an operating system.
- It manages:
    - **Processes**
    - **Memory**
    - **File systems**
    - **Device I/O**
    - **Networking**
- Linux specifically uses a **monolithic kernel** architecture with support for loadable kernel modules (LKMs).

So, when people say “Linux”, they’re often referring to much more than just the kernel.

What Most People Call “Linux” Is Actually a Linux Distribution (Distro)

A **Linux distribution (distro)** is:

> **Linux kernel + GNU userland tools + libraries + package manager + desktop environment (optional) + system utilities**

Examples of Popular Linux Distros:

Distro Name	Kernel	Userland Tools	Purpose/Use Case
Ubuntu	Linux	GNU + Debian tools	Desktop/server
Debian	Linux	GNU	Stable/server
Arch Linux	Linux	GNU	Rolling release
Fedora	Linux	GNU + Red Hat tools	Developer/testbed
Android	Linux	Non-GNU (AOSP)	Mobile OS

Analogy

Imagine the **kernel** as the **engine** of a car — it powers the system.

- The **Linux kernel** is just the engine.
- A **Linux distribution** is the **whole car** — engine + frame + seats + dashboard + wheels.

Clarifying Definitions

Term	Definition
Kernel	The core system software that manages hardware and system resources. Linux is this.
Operating System	The full stack of software that allows users to interact with a computer. This includes the kernel, system libraries, shell, etc.
Linux OS	Usually refers to a Linux-based distro (e.g., Ubuntu, Fedora).

GNU stands for **"GNU's Not Unix"** — a **recursive acronym**, which is common in computer science. It is a **free and open-source software project** launched in **1983** by **Richard Stallman**, aimed at creating a **Unix-compatible operating system** composed entirely of **free software**.

Purpose of the GNU Project

- Provide a **free Unix-like OS** (as in freedom, not necessarily price)
- Develop **free alternatives** to proprietary UNIX tools and utilities
- Promote the philosophy of **software freedom**: users should be able to **run, study, modify, and share** software freely

Components of the GNU Operating System

The GNU project developed most components of a complete OS — except for the kernel.

Component	Description
GNU Core Utilities (coreutils)	Basic file, shell, and text manipulation commands (ls, cp, rm, etc.)
Bash	GNU Bourne Again SHell
GCC	GNU Compiler Collection (C, C++, etc.)
glibc	GNU C standard library
GDB	GNU Debugger
Make	Build automation tool
Emacs	Extensible text editor

GNU + Linux = GNU/Linux

By the early 1990s, GNU had nearly all the pieces for a full operating system **except the kernel**. In 1991, **Linus Torvalds** released the **Linux kernel**, which filled that gap.

> This combination — **GNU tools + Linux kernel** — is what most people refer to simply as **Linux**.

Key Components of a Linux Distribution

Every Linux distro typically includes:

Component	Purpose
Linux Kernel	The core OS managing hardware, processes, memory, etc.
Init System	Starts and manages services at boot (e.g., systemd, SysV)
Package Manager	Handles software install/update (e.g., apt, yum, dnf)
System Tools	File managers, shells (bash), text editors, etc.
Server Software	Web servers (Apache, Nginx), databases, mail servers
Desktop Environment	GUI interface (e.g., GNOME, KDE, XFCE)
Documentation	Man pages, help files, admin guides

Popular Linux Distributions (with Use Cases)

Here's a categorized view of the ones you listed, with technical context:

Distro	Website	Focus/Use Case	Package Manager
Kali Linux	kali.org	Penetration testing, ethical hacking	apt (Debian-based)
Red Hat Enterprise Linux (RHEL)	redhat.com	Enterprise-grade servers, support & stability	dnf/yum (RPM-based)

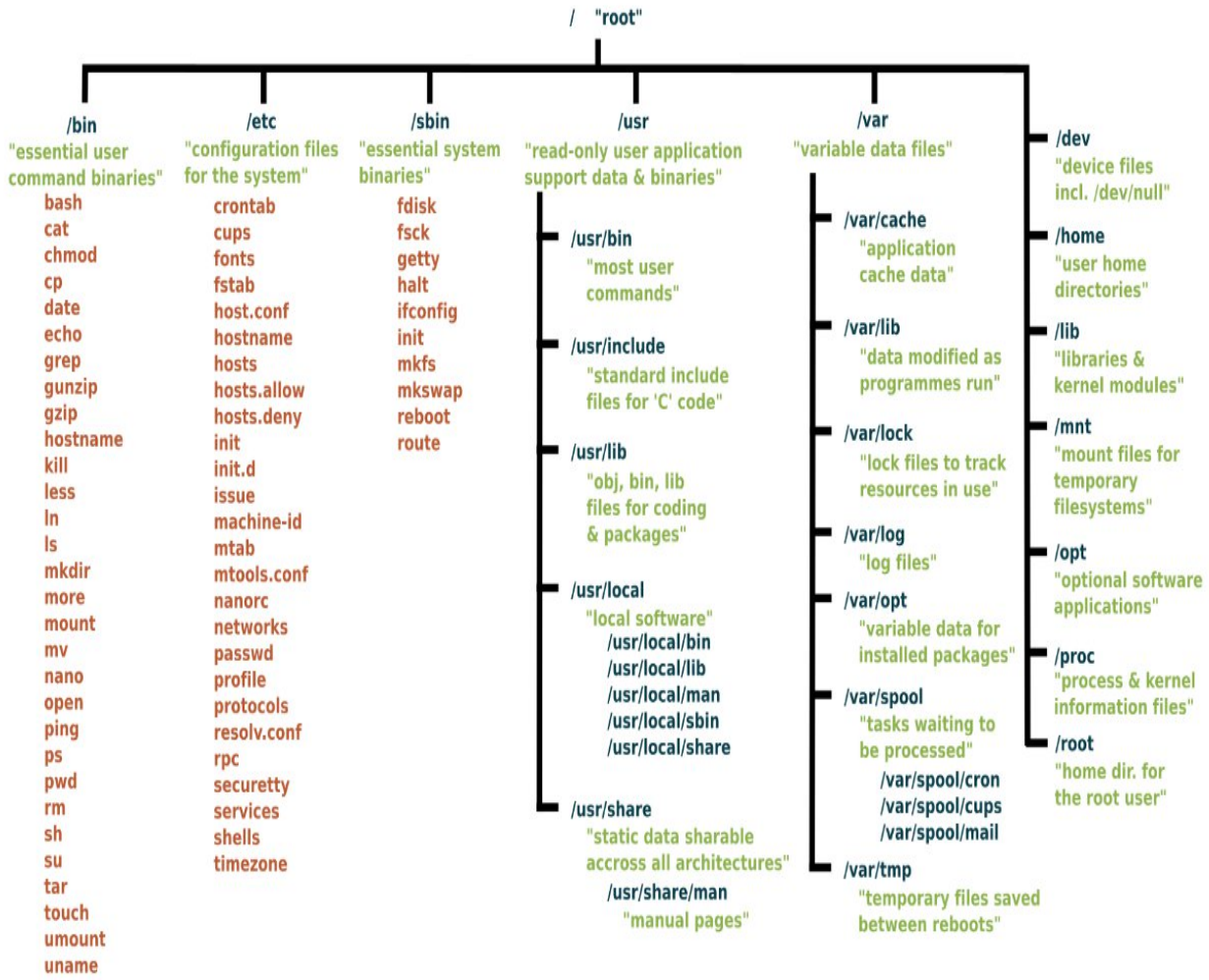
Distro	Website	Focus/Use Case	Package Manager
Ubuntu	ubuntu.com	Desktop, cloud, server, IoT	apt
CentOS *(now replaced by CentOS Stream)*	centos.org	Former RHEL clone, now rolling pre-RHEL testbed	dnf/yum
Debian	debian.org	Parent of Ubuntu, extremely stable	apt
Linux Mint	linuxmint.com	User-friendly desktop (based on Ubuntu)	apt
OpenSUSE	opensuse.org	Desktop/server, dev-focused (Tumbleweed & Leap)	zypper

Package Management Overview

Each distro has its own **package manager** and **format**:

Format	Used By	Example Command
.deb	Debian, Ubuntu	sudo apt install nginx
.rpm	RHEL, CentOS, Fedora	sudo dnf install nginx

Linux-File Structure :



1. / – Root Directory

The top-level directory of the Linux filesystem. All other directories and files branch from here.

2. /bin – Essential User Binaries

Contains basic command-line utilities needed by all users, like ls, cp, mv, and bash. These commands are essential for system operation and repair.

3. /sbin – System Binaries

Holds critical system administration binaries used mostly by the root user. Includes commands like fsck, reboot, and ifconfig.

4. /etc – Configuration Files

Stores all system-wide configuration files and shell scripts used to boot and configure the system. Examples: passwd, hosts, fstab.

5. /usr – User Applications & Data

Contains read-only user applications and support files like libraries, binaries, and documentation. Often includes subdirectories like /usr/bin, /usr/lib.

6. /var – Variable Files

Stores files that change during system operation, such as logs, mail, and spool files. Common subdirectories include /var/log and /var/cache.

7. /dev – Device Files

Contains special files that represent system devices like hard drives, USBs, and terminals. Enables interaction with hardware via files.

8. /home – User Home Directories

Holds personal directories for each user (/home/username). Users store their files, configs, and documents here.

9. /lib – Essential Shared Libraries

Contains essential shared libraries and kernel modules needed by the binaries in /bin and /sbin.

10. /mnt – Temporary Mount Points

A standard mount point for temporarily attaching filesystems like USBs, DVDs, or remote drives.

11. /opt – Optional Software

Used for installing optional or third-party application packages. Software here doesn't interfere with core system files.

12. /proc – Process & Kernel Info

A virtual filesystem showing real-time system and process information. Files like /proc/cpuinfo provide system data.

13. /root – Root User's Home Directory

Home directory for the root (superuser) account. Separate from regular users' home directories in /home.

14. /tmp – Temporary Files

Used to store temporary files created by programs. Files here are typically deleted on system reboot.

Interacting with Linux: Two Main Interfaces

1. Graphical User Interface (GUI)

A **GUI** is a **visual interface** allowing users to interact with the system using **windows, icons, menus, and pointing devices (WIMP)** like a mouse.

Popular Linux Desktop Environments:

Environment	Description	Lightweight?	Common Distros
GNOME	Default on Fedora, Ubuntu (modern look, simplified)	✗	Fedora, Ubuntu
KDE Plasma	Highly customizable, feature-rich	✓ (with	Kubuntu, openSUSE

Environment	Description	Lightweight?	Common Distros
		tweaks)	
Xfce	Fast and lightweight	✓✓	Xubuntu, Manjaro
LXQt/LXDE	Very lightweight, good for low-end systems	✓✓✓	Lubuntu
Unity	Former default for Ubuntu (now discontinued officially)	✗	Ubuntu (14.04–17.04)
Enlightenment (E)	Eye-candy and light resource use	✓	Bodhi Linux
Sugar	Educational environment (used in OLPC)	✓	Sugar-on-a-Stick

GUIs are launched by a **Display Manager** (e.g., GDM, LightDM, SDDM) and run atop an **X server** (X11) or **Wayland** display server.

2. 💻 Command Line Interface (CLI) / Shell

The CLI provides a **text-based interface** for precise and scriptable interactions with the OS.

What is a Shell?

A **shell** is a command-line interpreter that:

- Accepts keyboard input (commands)
- Parses them **left to right**
- Executes them using system calls
- Returns output to the screen

Common Linux Shells

Shell Name	Executable	Description
Bourne Shell	/bin/sh	Original UNIX shell; basis for scripting
Bourne Again Shell (Bash)	/bin/bash	Default on most Linux distros; supports scripting, command history, job control
C Shell (csh/tcsh)	/bin/csh, /bin/tcsh	C-like syntax; includes features like aliases, job control
Korn Shell (ksh)	/bin/ksh	Combines features of sh, csh; used in enterprise systems
Z Shell (zsh)	/bin/zsh	Advanced features: auto-completion, spell correction
Fish Shell	/usr/bin/fish	User-friendly, interactive, modern syntax

Most scripting and system administration in Linux is done using **Bash**, although sh-compatible scripts are most portable.

1. Built-in Commands

These are commands that are **integrated directly into the shell** (e.g., bash, zsh, sh). They are executed **within the shell process itself** — no separate executable file is called.

✅ **Characteristics:**

- **Faster**: No need to fork a new process
- Used for **shell-specific functions** (e.g., variable management, control flow, environment setup)
- Cannot be overridden by external binaries with the same name (unless explicitly prefixed)

📦 **Examples of Built-in Commands:**

Command	Description
cd	Change directory
echo	Print a string to stdout
export	Set environment variables
alias	Create shortcuts for commands
history	Show command history
read	Read user input into variables
exit	Exit the shell
pwd	Print working directory
type	Identify if a command is built-in or external

🔍 **Check if a command is built-in:**

```
type cd
# Output: cd is a shell builtin
```

2. External Commands

These are **separate executable files** stored on the filesystem. When you run them, the shell **searches the directories listed in your** $PATH and executes the binary if found.

✅ **Characteristics:**

- Can be **shell-independent**
- Located in directories like /bin, /usr/bin, /usr/local/bin, etc.
- Can be **user-defined scripts or compiled binaries**

c

📦 Examples of External Commands:

Command	Path (typical)	Description

Command	Path (typical)	Description
ls	/bin/ls	List directory contents
grep	/bin/grep	Search file content
cat	/bin/cat	Concatenate file contents
find	/usr/bin/find	Find files and directories
wget	/usr/bin/wget	Download files from the internet
python3	/usr/bin/python3	Run Python interpreter

🔍 Check the path of an external command:

```
which ls
# Output: /bin/ls
```

Summary Comparison

Feature	Built-in Command	External Command
Location	Inside the shell	Separate executable file
Execution	Runs in shell process	New process (fork/exec)
Performance	Faster	Slightly slower
Portability	May vary by shell (e.g., bash vs zsh)	Shell-independent
Customization	Harder to replace/override	Can be aliased, wrapped, etc.

Basic Commands

Commn ands	Description
ls	List the files and directories stored in the current directory = > ls
mkdir	Creat directory => mkdir Work
rmdir	Remove directory => rmdir Work
cd	Change the directory => cd work
pwd	Present working directory => pwd
mv	Rename the file => mv hello.txt hai.txt
cp	Copy the files from source to destination => cp <source file > <target file>
touch	Creat empty file => touch demo.txt
find	Used to find files or folders matching a particular search pattern => find . -type d -name src
ln	Used to create links => ln -s recipes.txt newrecipes.txt
gzip	Compress a file using the gzip compression => gzip -c filename > filename.gz
tar	Used to create an archive,grouping multiple files in a single file. => tar -xf archive.tar -C directory
cat	Prints a file's content to the standard output => cat file

Commands	Description
echo	Prints to the output the argument passed to it => echo "Hai"
chown	Change the owner to another user => chown flavio test.txt
chmod	Change the permitions of the file or directory => chmod +x demo.c
export	Command is used to export variables to child processes => export PATH=$PATH:/new/path
uname	Uname - print system information => uname –a
du	Command will calculate the size of a directors a whole: => du –ah work
df	Command is used to get disk usage df information => df -h
ps	list of user-initiated processes currently running in the current session = > ps -Af
kill	Send a signal to a process => kill –9 demo.c
diff	send a signal to a process => diff file1.txt file2.txt
Vim	Vim test editor => vim demo.c
Who	show who is logged on => who
Sudo	Commonly used to run a command as root. => sudo -i
history	Many programs read input from the user a line at a time => history

Commands	Description
gzip	Compress a file using the gzip compression => gzip -c filename > filename.gz Decompress -> gzip -d filename.gz Gunzip filename.gz
tar	Used to create an archive,grouping multiple files in a single file. => tar -xf archive.tar -C directory
cat	Prints a file's content to the standard output => cat file
wc	Gives us useful information about a file or input it receives via pipes => wc -l test.txt
grep	To search in files or combine it with pipes to filter the output of another command => grep –nir "ex" list.txt
printenv	Print all or part of environment => printenv

Creating Folders and Files :

Create folders using the **mkdir command**:

Sy: mkdir <Directory Name>
Ex: mkdir Work

Create multiple folders :

Sy:$ mkdir -p dir1/dir2/dir3
Ex: mkdirWork linuxSource

bootloader toolchain

Creating files redirection: Concatenate FILE(s) to standard output

Sy: cat >> file.txt
Ex: cat >> hello.txt

ls command :

Lists the files in the current directory, in alphanumeric order.

ls -a (all): Lists all the files (including.*files)

ls -l (long): Long listing (type, date, size, owner, permissions)

ls -t (time): Lists the most recent files first

ls -S (size): Lists the biggest files first

ls -r (reverse): Reverses the sort order

ls -ltr (options can becombined): Long listing, most recent files at the end

Cp command:

cp <source_file> <target_file>Copies the source file to the target.

Sy: cp file1 file2
Ex: cp hello.txt hello1.txt

cp file1 file2 file3 ... dir Copies the files to the target directory (last argument).

Sy: cp file1 file2 file3 <demo>
Ex: cp gsm.c gps.c Comm

cp r <source_dir> <target_dir>(recursive) : Copies the whole directory.

ex: cp –rf LinuxSource Work

cd and pwd :

Cd : change directory

Ex : mkdir work

Change directory to ``work`` : Cd work

Move up one directory : $ cd ..

Move up two directories : $ cd ../..

Change to home directory : $ cd ~

Get current directory : pwd

ln file1 file2: Creates a physical link.

Create soft links using the -s

ln -s file1 file2: Creates a symbolic link.

locate : It is used to locate a file in Linux System

echo : This command helps us move some data, usually text into a file.

df: It is used to see the available disk space in each of the partitions in your system.

tar : Used to work with tarballs (or files compressed in a tarball archive)

Changing file access rights :

The chmod and chown commands are used to control access to files in UNIX and Linux systems.

- chown: Used to change the owner of the file.
- chgrp: Used to change the group owner of the file.
- chmod: Used to modify the access/permission of a user.

- Add write permissions to the current user => chmod u+w
- FileAdd read permissions to users in the file group: => chmod g+r
- FileAdd execute permissions to other users => chmod o+x
- FileAdd read + write permissions to all users => chmod a+rw
- FileMake executable files executable by all => chmod a+rX *
- Make the whole directory and its contents accessible by all users
  => chmod -R a+rX dir (recursive)

Handling file contents :

grep <pattern> <files>Scans the given files and displays the lines which match the given pattern

$grep error *.log => Displays all the lines containing error in the *.log files

$grep i error *.log => Same, but case insensitive

$grep ri error .Same, but recursively in all the files in. and its subdirectories

$grep v info *.log => Outputs all the lines in the files except those containing info.

$grep -nri "example" hello.txt //scan string example in hello.txt

The " find " command can be used to find files or folders matching a particular search pattern

Find all the files under the current tree that have the .js extension and print the relative path of each file matching:

Ex : find . -name '*.js'

Use "-name" is case sensitive.
Use "-iname" to perform a case-insensitive search

Find directories under the current tree matching the name "modules" or 'public':

Ex: find . -type d -name modules -or -name public

- $ echo "export PATH="<path>:$PATH"" >> ~/.bashrc && source ~/.bashrc
- $ printenv PATH
- <path>:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
-

find . -type d -iname / (dir name)
find . -type f -iname a.c (“file name”)

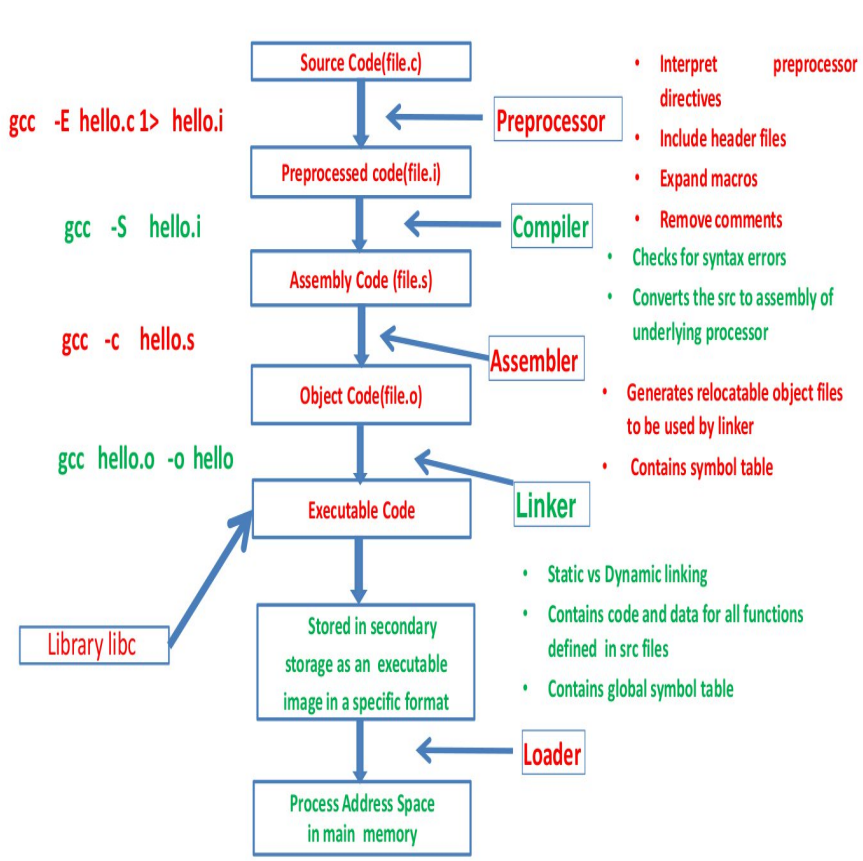
Grep -lnir –include=*.c* “searching string”

Cat file.txt
Cat file1.txt file2.txt file3.txt
Cat -n file.txt // with line no
Cat > file.txt // create file and make you to write
Cat file1.txt>file2.txt // copy one file into second, append, overwrite
Cat file1.txt>>file2.txt // append at end line .
Cat -E file.txt // will highlight the end of line

C Programming in Linux :

File

C Compiling Process In Linux



management in Linux :

Following are the four key system calls for performing file I/O (programming languages and software packages typically employ these calls indirectly via I/O libraries):

• **fd = open(pathname, flags, mode)** opens the file identified by pathname, returning a file descriptor used to refer to the open file in subsequent calls. If the file doesn't exist, open() may create it, depending on the settings of the flags bit. The flags argument also specifies whether the file is to be opened for reading, writing, or both. The mode argument specifies the permissions to be placed on the file if it is created by this call. If the open() call is not being used to create a file, this argument is ignored and can be omitted

• **numread = read(fd, buffer, count)** reads at most count bytes from the open file referred to by fd and stores them in buffer. The read() call returns the number of bytes actually read. On eof, read() returns 0.

• **numwritten = write(fd, buffer, count)** writes up to count bytes from buffer to the open file referred to by fd. The write() call returns the number of bytes actually written, which may be less than count

• **status = close(fd)** is called after all I/O has been completed, in order to release the file descriptor fd and its associated kernel resources

File Descriptor	Purpose	POSIX Name	stdio Stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

File Descriptor to File Contents :

1. Each process in UNIX has an associated PPFDT, whose size is equal to the number of files that a process can open simultaneously

2. File descriptor is an integer returned by open() system call, and is used as an index in the PPFDT

3. File descriptor is used in read(), write() and close() system call

4. Kernel uses this descriptor to index the PPFDT, which contain a pointer to another table called System Wide File Table

5. In the System Wide File Table, other than some information there is another pointer to a table called Inode Table

6. The inode table contains a unique Inode to every unique file on disk

Standard Descriptors in Linux :

Three files are automatically opened for every process to read its input from and to send its output and error messages to.
• Theses files are called standard files:

- **0 – Standard Input (stdin)**. Default input to a program is from the user terminal (keyboard), if no file name is given.
- **1 – Standard Output (stdout)**. A simple program's output normally goes to the user terminal (monitor), if no file name is given.
- **2 – Standard Error (stderr)**. Default output of error messages from a program normally goes to the user terminal, if no file name is given.

• These numbers are called File Descriptors – System calls use them to refer to files.

Examples – File handling

Open terminal using **<ctrl> + <alt> + t**
in terminal open vim editor using command

$ vim file1.c

its opened in command mode **press i** to go in editor mode and then write the below code

```
int main()
{
        char buff[256];
        read(0, buff, 255);
        write(1, buff, 255);
        return 0;
}
```

after code is writen save the file to come out from editor mode to command mode **press <esc>**
and then to save and comming back to shell **write :wq!** And then **press <enter>**
after that compile the code using command

$ gcc file1.c -o file1

then run the binary file using command

$ ./file1

```
int main()
{
        char buff[256];
        while(1)
                {
                int n = read(0, buff, 255);
                write(1, buff, n);
                }
        return 0;
}

int main()
{
        char buff[2000];
        int fd = open ("/etc/passwd", O_RDONLY);
        int n;
        for(;;)
        {
        n = read(fd, buff, 1000);
        if (n <= 0)
                {
                close(fd);
                exit(-n);
                }
        write(1, buff, n);
        }
        return 0;
}

int main(){
        int n;
        char buff[1024];
        int fd=open("file.txt",O_CREAT|O_TRUNC|O_RDWR,0666);
        for(;;){
                n = read(0, buff, 1023);
                if (n <= 0){
                        printf("Error in reading kb.\n");
                        exit(-n);
                        }
                write(fd, buff, n);
                }
        close(fd);
        return 0;
```

}

Makefile :

UNIX make utility

1.Imagine you write a program and divide it into hundred .c files and some header files

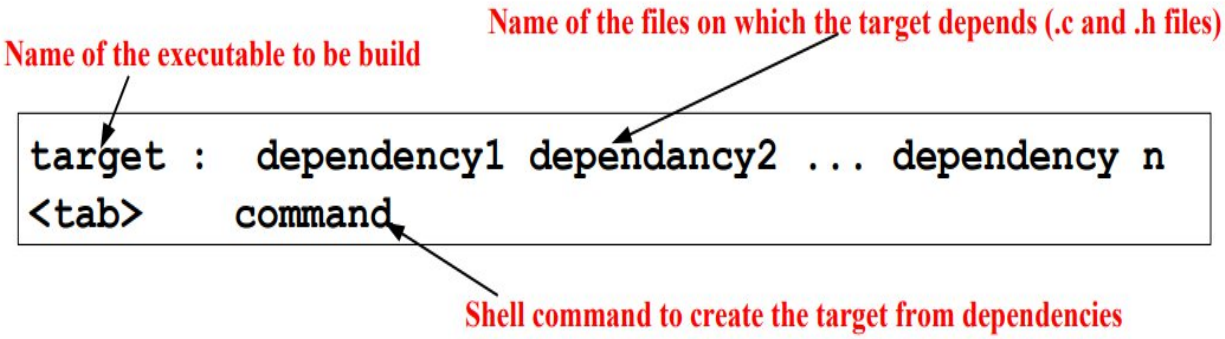
2. To make the executable you need to compile those hundred source files to create hundred relocatable object files and then you need to link those object files to final executable

3. What happens if we make changes to one of these files:
   Recompile all the files and then link all of them
   Recompile only the file which has changed and then link
   What if instead of .c file a .h file has changed ?
   Solution: Recompile only those .c files that include this header file and then link

4. UNIX make utility is a powerful tool that allows you to manage compilation of multiple modules into an executable

5. It reads a specification file called "makefile" or "Makefile", that describes how the modules of a s/w system depend on each other.
   If you want to use a nonstandard name you can specify that name to make using -f option

6. Make utility uses this dependency specification in the makefile and the time when various components were modified, in order to minimize the amount of recompilation



1.This is one dependency rule in a makefile
2. A makefile may have several such rules.
   Every make rule describes the dependency relationship
3. Advantages of make utility:
 (a) Makes management of large s/w projects with multiple source files easy
 (b) No need to recompile a source file that has not been modified, only those files that have been changed are recompiled, others are simply relinked

Make [options]

The most commonly used are:

-f By default make looks for a file "makefile" in the current directory. If doesn't exist, it looks for "Makefile".

To tell make to use a different file, user -f option followed by filename

-k Tells make to keep going when an error is found, rather than stopping as soon as the first problem is detected. You can use this to find out in one go which source files fail to compile

Multiple Targets in a Makefile

A makefile can have multiple targets.

We can call a make file with the name of a particular target

- To tell make to build a particular target, you can pass the target name to make as parameter (By default, make will try to make the first target listed in makefile)
- Many programmers specify all as the first target in their makefile and then list the other targets as being dependencies for all
- A phony target is a target without dependency list. Some important phony targets are all, clean, install

```
clean:
        -@rm  -f  *.o
```

- If there is no .o file in the current working directory, make will return an error. If we want make to ignore error while executing a command we proceed the command with a hyphen as done above. Moreover, make print the command to stdout before executing. If we want to tell make not to print the command to stdout before executing we use @ character

Multiple Makefiles in a Project

Project source divided in multiple directories

- Different developers involved
- Multiple makefiles
- Top level makefile use include directive
- Include Directive:

  Tells make to suspend reading the current makefile and read one or more other makefiles before continuing.

  include ./d2/makefile ./d3/makefile

Use of Macros in a Makefile

A Makefile allows us to use macros or variables, so that we can write it in a more generalized form.

Variables allow a text string to be defined once and substituted in multiple places later

- We can define macros/variables in a makefile as: MACRONAME=value
- We can access the macros as $(MACRONAME)
- Example: We can use a macro to give options to the compiler,

  e.g., while an application is being developed, it will be compiled with no optimization but with debugging information included.

  So we declare a macro CFLAGS

  CFLAGS = -std=c11 -O0 -ggdb -Wall

  and later can use it with all compilation commands like

  gcc -c file.c $(CFLAGS)

simple makefile to do generate all intermediate files

in terminal **$ vim makefile**

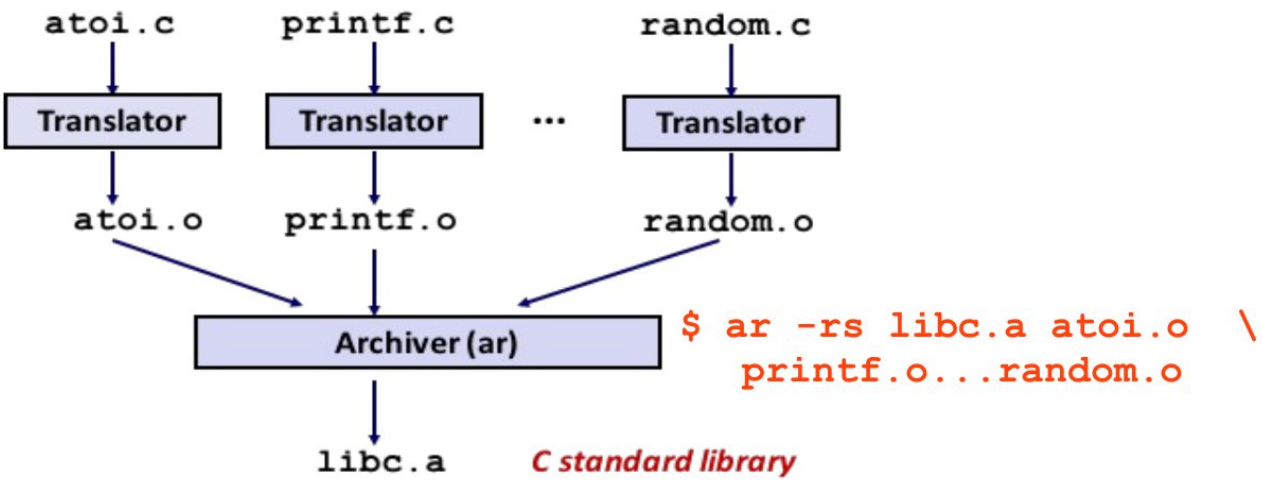
```
app:hello.o
	gcc hello.o -o app
hello.o:hello.s
	gcc -c hello.s
hello.s:hello.i
	gcc -S hello.i
hello.i:hello.c
	gcc -E hello.c >hello.i
```

$ make

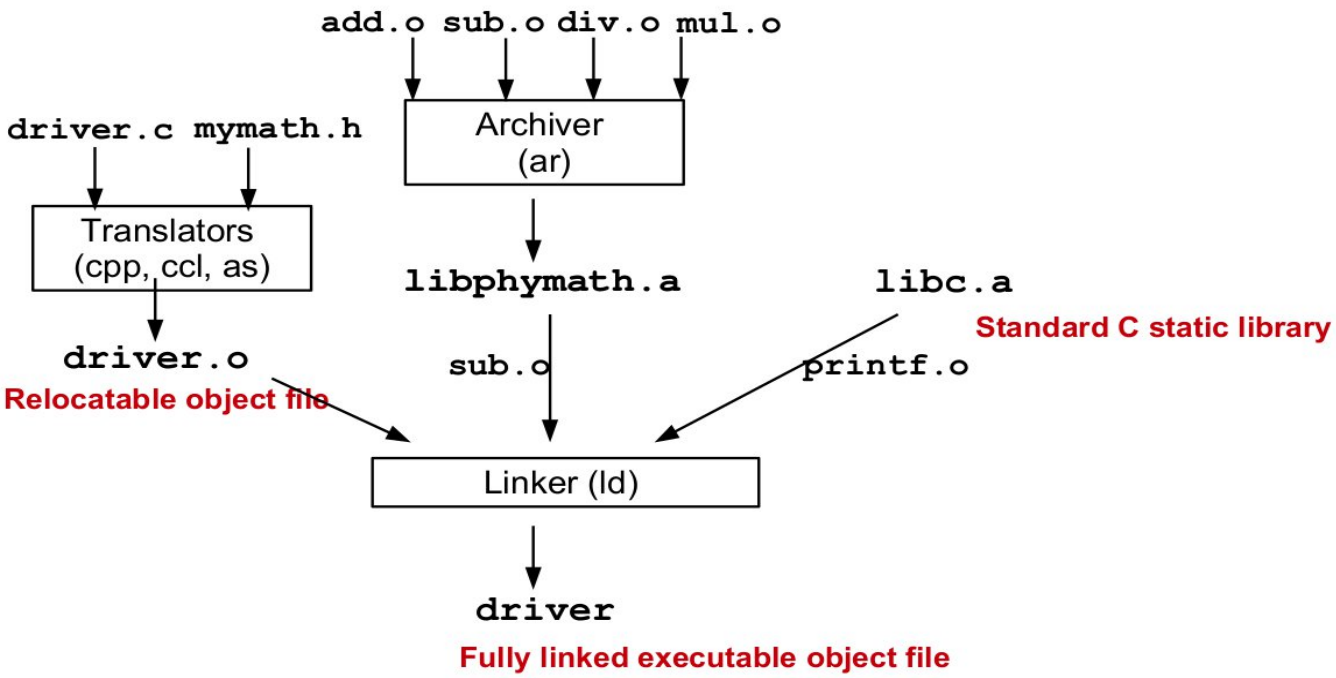
Static and Dynamic Libraries :

Static Libraries :

- Concatenate related relocatable object files into a single file with an index called an archive
- Enhance the linker so that it tries to resolve unresolved external references by looking for the symbols in one or more archives
- If an archive member file resolve reference, link it to the executable
- To make the processes fast **.a** files contains an index for the symbols in all files



Linking with Static Library :



The Librarian (ar utility) :

ar is a Unix tool also called librarian that allows us to :

1) Create

-r → create a new archive

$ ar -r libfirst.a file1.o file2.o

-q → append an object file to an existing archive

$ ar -q libfirst.a file3.o

-d → delete object modules from an existing archive

$ ar -d libfirst.a file2.o

2)Extract:

-x → extract object modules in your PWD

$ ar -x /usr/lib/libm.a

3)Display:

-t → display table of contents of an archive

$ ar -t /usr/lib/libm.a

Linking with Static Library :

Linker's algorithm for resolving external references:

- Scan .o files and .a files in the command line order
- During the scan, keep a list of the current unresolved references
- As each new .o or .a file, obj, is encountered, try to resolve each unresolved reference in the list against the symbols defined in obj

● If any entries in the unresolved list at end of scan, then error

Problem:
● Command line order matters
● Put libraries at the end of the command line

Limitations of Static Libraries :

Limitations of static linking:
● The size of executable is large
● Duplication in the executables stored on disk
● Duplication in the executables running in memory. Suppose you are executing ten C-programs all of them using the scanf functions. So ten copies of scanf functions will be there in memory.
● Minor bug fixes of system libraries require each application to be explicitly relinked

Modern solution: Shared Libraries
● Object files that contain code and data that are loaded and linked into an application dynamically, at either load-time or run-time
● In UNIX world they are called shared objects (.so)
● In MS world they are called dynamic link libraries or dlls

Shared libraries :

● A shared library is similar to static library because it is also a group of object files however a shared library is different from a static library as the linker and loader both behave differently to a dynamic library.
● A code that can be loaded and executed at any address without being modified by the linker is known as position-independent code. The –fPIC option to gcc specifies that the compiler should generate position- independent code. This is necessary for shared libraries, since there is no way of knowing at link time where the shared library code will be located in memory.

● Steps to create a shared library are given below:

Step1: Compile each .c file with -fPIC flag to create object files.

```
$gcc -c -fPIC myadd.c mysub.c mydiv.c mymul.c
```

Step2: Produce a shared object which can then be linked with other objects to form an executable

```
$gcc -shared myadd.o mysub.o mydiv.o mymul.o -o libphymath.so
$gcc -c -fPIC myadd.c mysub.c mydiv.c mymul.c
$gcc -shared myadd.o mysub.o mydiv.o mymul.o -o libphymath.so
```

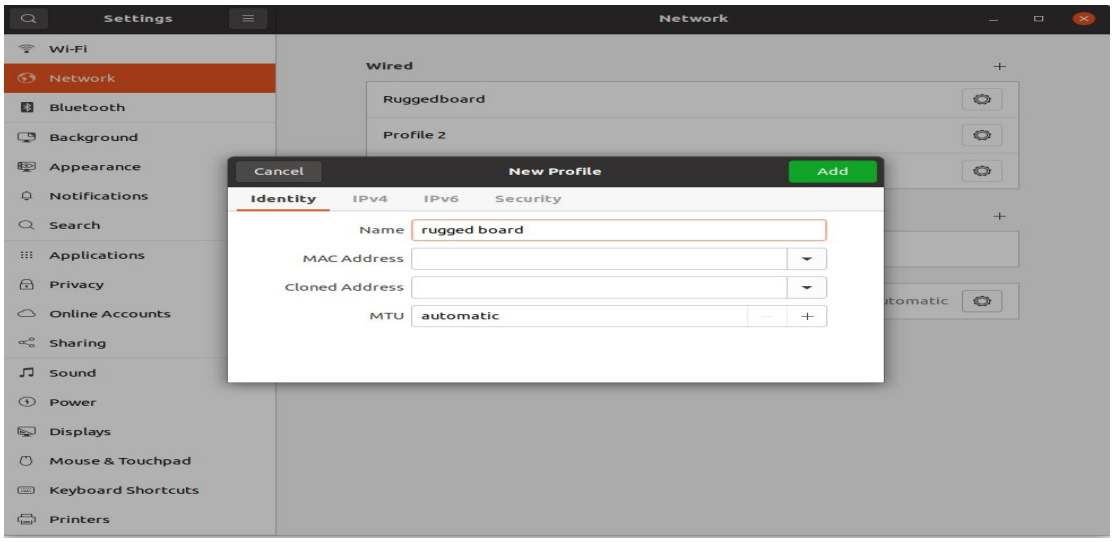
PC environment Setup :

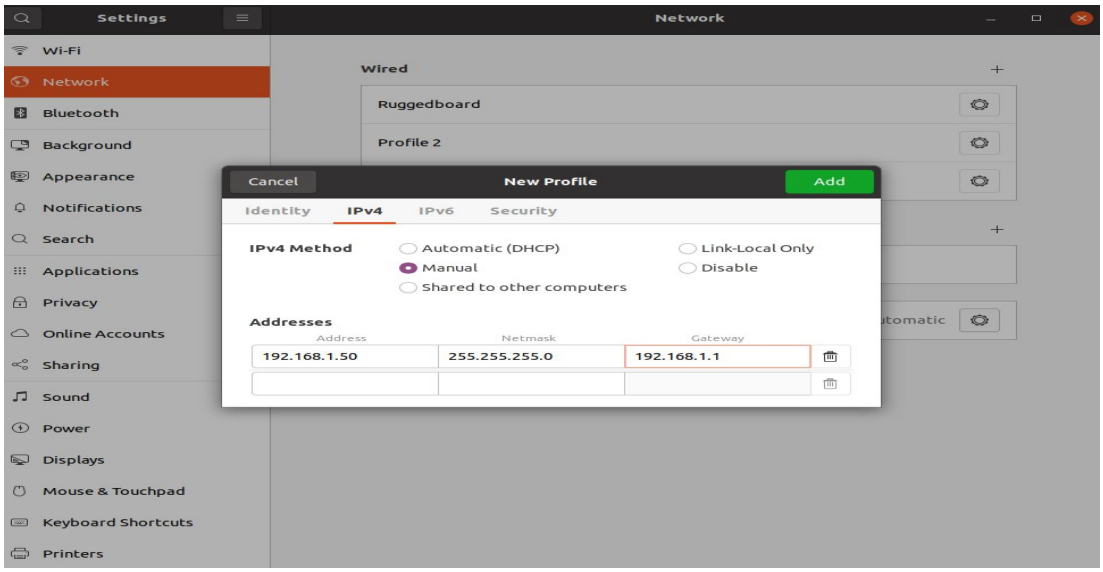
Installing Additional Packages :

go to below link and download the elinux_pkg.sh script
https://developer.ruggedboard.com/g5-system-development-guide/host-setup

set pc ip :

go to setting->network->add wire ->enter name





Toolchain Downloading:

Step1: Make directory to download the toolchain.

$ mkdir toolchain

Step2: Download the Toolchain for Rugged Board A5d2x in your pc using below link:
https://drive.google.com/drive/folders/1JLbgs26aEYKNDR98raaD51BKLMbMLQ85

Step3: unzip the downloaded file. Using below command

$ unzip RB-Toolchain.20.0.0.zip

$ cd RB-Toolchain.20.0.0/

Step4: Give the permissions to the toolchain script file as below and install it.

$ chmod 777 poky-tiny-musl-x86_64-core-image-minimal-cortexa5hf-neon-toolchain-2.5.2.sh
$ ./poky-tiny-musl-x86_64-core-image-minimal-cortexa5hf-neon-toolchain-2.5.2.sh

Step5: To cross compile any sample application run the below command to get the arm environment.

$. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

Step6: Cross compiling the c application as below

$ $CC hello_world.c -o hello_world
$ file hello_world

Example C program

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

Using http server file transfor from pc to board :

step1: Install http sever in your pc use this command

$ sudo snap install http

step2: run the server using command in pc

$ python3 -m http.server 8080

```
chaman@chaman:~/work/SIT/MPU/RB_programs$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.8.31 - - [09/Jun/2023 15:04:41] "GET /test_mpu HTTP/1.1" 200 -
```

Setup for Rugged Board :

connect the USB cable from Board to PC and in terminal use command

$ sudo dmesg | tail

FTDI USB Serial Device converter now attached to ttyUSB0

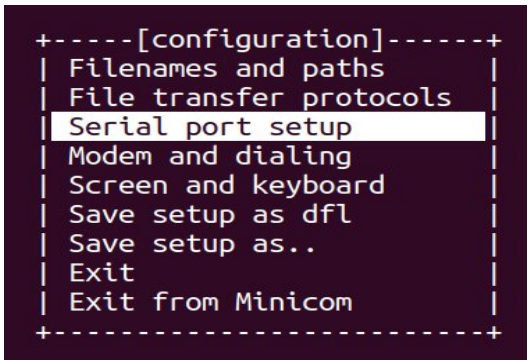
Linux-SerialPort :

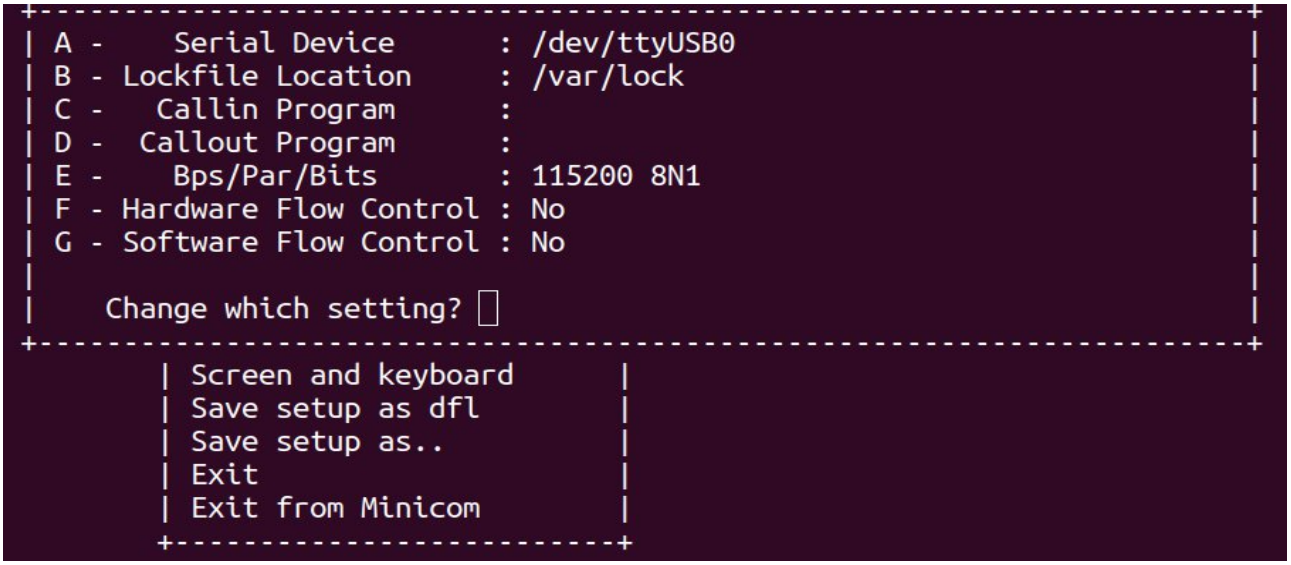
minicom is a serial communication program that connects to devices through a GNU/Linux PC's serial ports.
If run by calling its name without any additional arguments, it uses whatever settings have been saved for its defaults in /etc/minicom/minirc.dfl.

$ sudo apt-get install minicom
$ man minicom

Then use command

$ sudo minicom -s





```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyUSB0, 17:14:17

Press CTRL-A Z for help on special keys


Poky (Yocto Project Reference Distro) 2.5.2 rugged-board-a5d2x-sd1 /dev/ttyS0

rugged-board-a5d2x-sd1 login: root
root@rugged-board-a5d2x-sd1:~#
```

Rugged Board side :

Booting Rugged Board from NOR :

Step1: set the ip address by using below command on the target board

$ ifconfig eth0 192.168.1.15

Step2: Next ping to your host ip address by using below command.

$ ping 192.168.1.50

```
VFS: Mounted root (ext3 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 1024K
EXT4-fs (mmcblk1p2): re-mounted. Opts: (null)
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 (Yocto Project Reference Distro) 2.5.2 rugged-board-a5d2x-sd1 /dev/ttyS0

rugged-board-a5d2x-sd1 login: root
root@rugged-board-a5d2x-sd1:~# ifconfig eth0 192.168.1.15
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
root@rugged-board-a5d2x-sd1:~# macb f8008000.ethernet eth0: link up (100/Full)
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready

root@rugged-board-a5d2x-sd1:~# ping 192.168.1.50
PING 192.168.1.50 (192.168.1.50): 56 data bytes
64 bytes from 192.168.1.50: seq=0 ttl=64 time=1.331 ms
64 bytes from 192.168.1.50: seq=1 ttl=64 time=0.753 ms
64 bytes from 192.168.1.50: seq=2 ttl=64 time=0.728 ms
64 bytes from 192.168.1.50: seq=3 ttl=64 time=0.516 ms
64 bytes from 192.168.1.50: seq=4 ttl=64 time=0.712 ms
64 bytes from 192.168.1.50: seq=5 ttl=64 time=0.717 ms
64 bytes from 192.168.1.50: seq=6 ttl=64 time=0.695 ms
64 bytes from 192.168.1.50: seq=7 ttl=64 time=0.749 ms
64 bytes from 192.168.1.50: seq=8 ttl=64 time=0.727 ms
64 bytes from 192.168.1.50: seq=9 ttl=64 time=0.500 ms
64 bytes from 192.168.1.50: seq=10 ttl=64 time=0.713 ms
64 bytes from 192.168.1.50: seq=11 ttl=64 time=0.699 ms
64 bytes from 192.168.1.50: seq=12 ttl=64 time=0.542 ms
64 bytes from 192.168.1.50: seq=13 ttl=64 time=0.708 ms
^Z
[1]+  Stopped(SIGTSTP)        ping 192.168.1.50
root@rugged-board-a5d2x-sd1:~#
```

$ ./hello_world

Rugged Board Application :

GPIO :

Title : LED Blinking

Aim: Test LEDs (PC13,PC17,PC19)
Equipment Requirements:
A) Hardware Requirements :

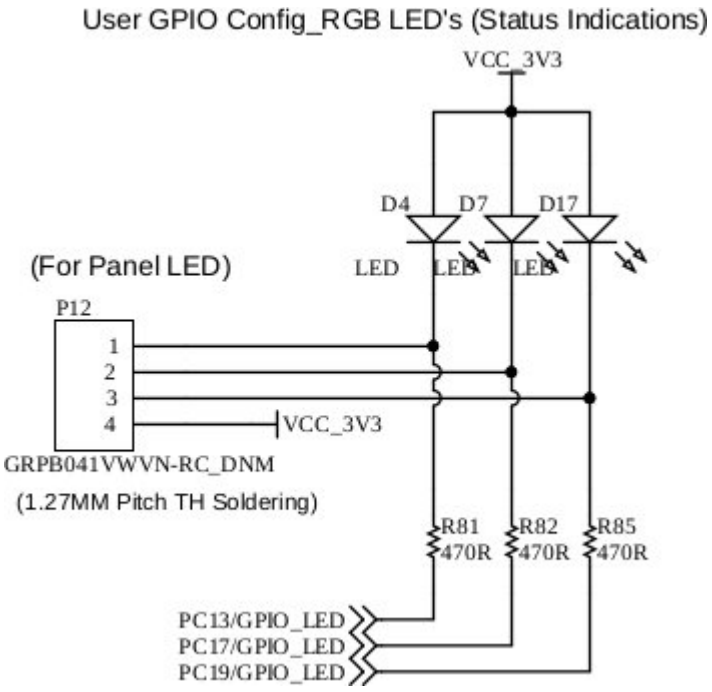
1. RB-A5D2X Development board
2. USB cable

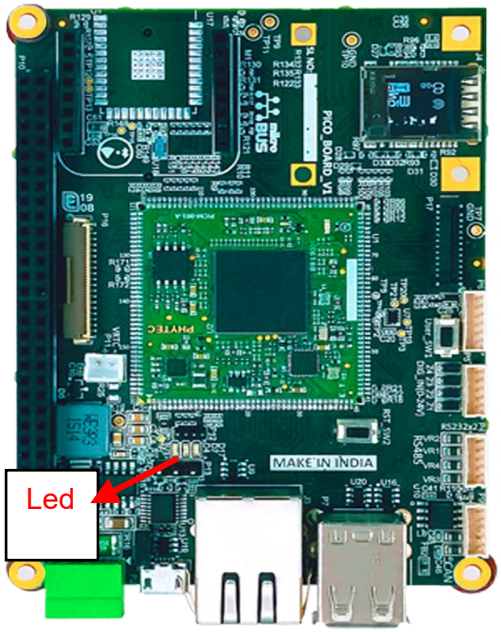
B) Software Requirements :

1. SDK

Connections:

1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X
3. Leds are already on board.





(command line):

PC13: (pin number 77):

root@ruggedboard-imx6ul: echo 77 > /sys/class/gpio/export
root@ruggedboard-imx6ul: echo out > /sys/class/gpio/gpio77/direction
root@ruggedboard-imx6ul: echo 0 > /sys/class/gpio/gpio77/value
root@ruggedboard-imx6ul: echo 1 > /sys/class/gpio/gpio77/value

PC17: (pin number 81):

root@ruggedboard-imx6ul: echo 81 > /sys/class/gpio/export
root@ruggedboard-imx6ul: echo out > /sys/class/gpio/PC17/direction

```
root@ruggedboard-imx6ul: echo 0 > /sys/class/gpio/PC17/value
root@ruggedboard-imx6ul: echo 1 > /sys/class/gpio/PC17/value
```

PC19: (pin number 83):

```
root@ruggedboard-imx6ul: echo 83 > /sys/class/gpio/export
root@ruggedboard-imx6ul: echo out > /sys/class/gpio/PC19/direction
root@ruggedboard-imx6ul: echo 0 > /sys/class/gpio/PC19/value
root@ruggedboard-imx6ul: echo 1 > /sys/class/gpio/PC19/value
```

SYSFS:

GPIO:

$Vi led_blinky.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#define GPIO_EN "/sys/class/gpio/export"

#define LED1_DIR "/sys/class/gpio/PC13/direction"
#define LED1_Val "/sys/class/gpio/PC13/value"

#define LED2_DIR "/sys/class/gpio/PC17/direction"
#define LED2_Val "/sys/class/gpio/PC17/value"

#define LED3_DIR "/sys/class/gpio/PC19/direction"
#define LED3_Val "/sys/class/gpio/PC19/value"


int led1_fd,led2_fd,led3_fd;
char gpio_buf[30];
int led1_num = 77,led2_num = 81, led3_num = 83;


void leds_init()
{
   //led1 export
   led1_fd = open(GPIO_EN,O_WRONLY);
   if(led1_fd < 0)
```

```
{
   printf("Unable to open the file %s\n",GPIO_EN);
   exit(0);
}
printf("Enable LED1\n");
sprintf(gpio_buf,"%d",led1_num);
write(led1_fd,gpio_buf,strlen(gpio_buf));
close(led1_fd);

//led2 export
led2_fd = open(GPIO_EN,O_WRONLY);
if(led2_fd < 0)
{
   printf("Unable to open the file %s\n",GPIO_EN);
   exit(0);
}
printf("Enable LED2\n");
sprintf(gpio_buf,"%d",led2_num);
write(led2_fd,gpio_buf,strlen(gpio_buf));
close(led2_fd);

//led3 export
led3_fd = open(GPIO_EN,O_WRONLY);
if(led3_fd < 0)
{
   printf("Unable to open the file %s\n",GPIO_EN);
   exit(0);
}
printf("Enable LED3\n");
sprintf(gpio_buf,"%d",led3_num);
write(led3_fd,gpio_buf,strlen(gpio_buf));
close(led3_fd);

//led1 direction
printf("Configuring LED1 direction \n");
led1_fd = open(LED1_DIR,O_WRONLY);
if(led1_fd < 0)
{
   printf("Unable to open the file %s",LED1_DIR);
   exit(0);
}

write(led1_fd,"out",3);
close(led1_fd);
```

```
//led2 direction
printf("Configuring LED2 direction \n");
led2_fd = open(LED2_DIR,O_WRONLY);
if(led2_fd < 0)
{
   printf("Unable to open the file %s",LED2_DIR);
   exit(0);
}

write(led2_fd,"out",3);
close(led2_fd);

//led3 direction
printf("Configuring LED3 direction \n");
led3_fd = open(LED3_DIR,O_WRONLY);
if(led3_fd < 0)
{
   printf("Unable to open the file %s",LED3_DIR);
   exit(0);
}

write(led3_fd,"out",3);
close(led3_fd);

//LED1 value
printf("Set the value into LED1_Val \n");

led1_fd = open( LED1_Val,O_WRONLY);
if(led1_fd < 0)
{
   printf("Unable to open the file %s",LED1_Val);
   exit(0);
}

//LED2 value
printf("Set the value into LED2_Val \n");

led2_fd = open( LED2_Val,O_WRONLY);
if(led2_fd < 0)
{
   printf("Unable to open the file %s",LED2_Val);
   exit(0);
}
```

```
    //LED3 value
    printf("Set the value into LED3_Val \n");

    led3_fd = open( LED3_Val,O_WRONLY);
    if(led3_fd < 0)
    {
        printf("Unable to open the file %s",LED3_Val);
        exit(0);
    }
}


int main()
{   int count=20;
    leds_init();

    printf("Toggling the LEDs! \n");
    while(count--)
    {
        write(led1_fd,"0" ,1);
        write(led2_fd,"0" ,1);
        write(led3_fd,"0" ,1);
        usleep(500000);
        write(led1_fd,"1" ,1);
        write(led2_fd,"1" ,1);
        write(led3_fd,"1" ,1);
        usleep(500000);
    }
    close(led1_fd);
    close(led2_fd);
    close(led3_fd);
}
```

compile :(HOST)

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} led_blinky.c -o blinky

execute:(BOARD)

root@rugged-board-a5d2x-sd1:/home# ./blinky

Enable LED1

Enable LED2

Enable LED3

Configuring LED1 direction

Configuring LED2 direction

Configuring LED3 direction

Set the value into LED1_Val

Set the value into LED2_Val

Set the value into LED3_Val

Toggling the LEDs!

MRAA

Vi leds_mraa.c

```
#include <unistd.h>
#include <mraa/gpio.h>

int main() {
// Initialize GPIO pins for the LEDs
mraa_gpio_context led1, led2, led3;
led1 = mraa_gpio_init(61); // GPIO pin 61 for LED 1
led2 = mraa_gpio_init(62); // GPIO pin 62 for LED 2
led3 = mraa_gpio_init(63); // GPIO pin 63 for LED 3

if (led1 == NULL || led2 == NULL || led3 == NULL) {
// Failed to initialize GPIO
return 1;
}

// Set the GPIO direction to output for all LEDs
if (mraa_gpio_dir(led1, MRAA_GPIO_OUT) != MRAA_SUCCESS ||
mraa_gpio_dir(led2, MRAA_GPIO_OUT) != MRAA_SUCCESS ||
mraa_gpio_dir(led3, MRAA_GPIO_OUT) != MRAA_SUCCESS) {
// Failed to set GPIO direction
mraa_gpio_close(led1);
mraa_gpio_close(led2);
mraa_gpio_close(led3);
return 1;
}

// Turn on and off LEDs with different patterns
```

```
while (1) {
// Pattern 1: Turn on LED 1, off LED 2 and LED 3
mraa_gpio_write(led1, 0);
mraa_gpio_write(led2, 1);
mraa_gpio_write(led3, 1);
usleep(500000);

// Pattern 2: Turn off LED 1, on LED 2, off LED 3
mraa_gpio_write(led1, 1);
mraa_gpio_write(led2, 0);
mraa_gpio_write(led3, 1);
usleep(500000);

// Pattern 3: Turn off LED 1, off LED 2, on LED 3
mraa_gpio_write(led1, 1);
mraa_gpio_write(led2, 1);
mraa_gpio_write(led3, 0);
usleep(500000);
}

// Close the GPIO connections
mraa_gpio_close(led1);
mraa_gpio_close(led2);
mraa_gpio_close(led3);

return 0;
}
```

compile:(HOST)

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

$CC leds_mraa.c -o led_mraa –lmraa

BOARD:

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./led_mraa
gpio 77
gpio pin 77
gpio 81
gpio pin 81
gpio 83
gpio pin 83
```

Python:

Source code:

```
import mraa

import time

gpio_1 = mraa.Gpio(61)

 gpio_2 = mraa.Gpio(62)

gpio_3 = mraa.Gpio(63)

gpio_1.dir(mraa.DIR_OUT)

 gpio_2.dir(mraa.DIR_OUT)

gpio_3.dir(mraa.DIR_OUT)

while True:

gpio_1.write(1)

gpio_2.write(0)

gpio_3.write(1)

time.sleep(1)

gpio_1.write(0)

gpio_2.write(1)

 gpio_3.write(0)

time.sleep(1)

gpio_1.write(0)

gpio_2.write(0)

gpio_3.write(1)

time.sleep(1)
```

Output:

```
root@rugged-board-a5d2x-sd1:~# python3 01_gpio_blink.py
gpio 77
gpio pin 77
gpio 81
gpio pin 81
gpio 83
gpio pin 83
```

Title: LED Blink with User Switch

Aim: Test switch PC13 with User Switch.

Equipment Requirements:

A) Hardware Requirements :

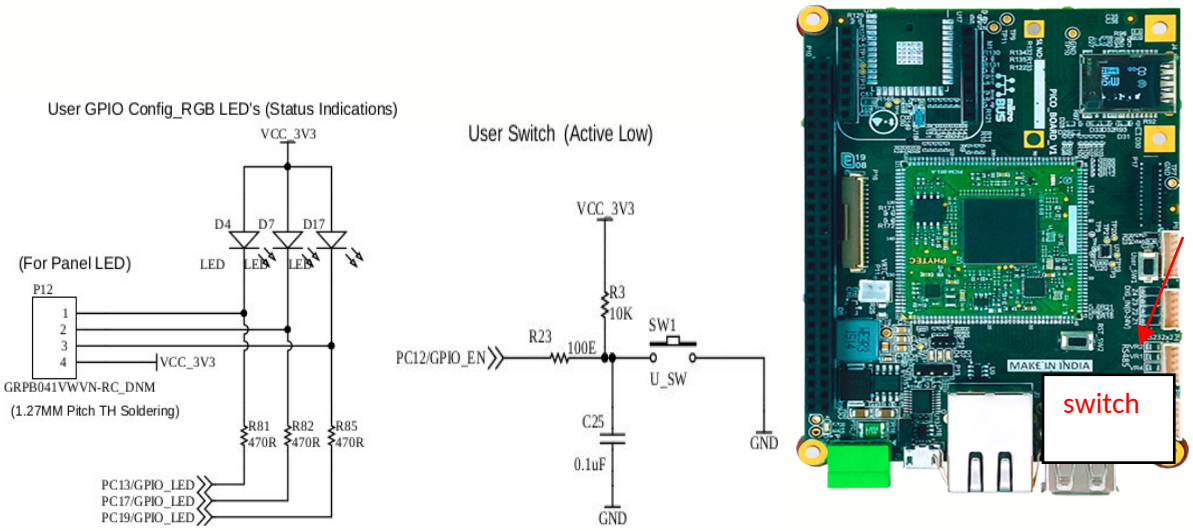
1. RB-A5D2X Development board
2. USB cable

B) Software Requirements:

1. SDK

Connections:

1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X
3. Leds are already on board.
4. User Switch is present on board.



Export PC12

root@rugged-board-a5d2x-sd1:~# echo 76 > /sys/class/gpio/export

Set PC12 as input pin

root@rugged-board-a5d2x-sd1:~# echo in > /sys/class/gpio/PC12/direction

Keep pressing user switch

root@rugged-board-a5d2x-sd1:~# cat /sys/class/gpio/PC12/value

0

Switch not pressed

```
root@rugged-board-a5d2x-sd1:~# cat /sys/class/gpio/PC12/value
1
```

Unexport PC13

```
root@rugged-board-a5d2x-sd1:~# echo 76 > /sys/class/gpio/unexport
```

SYSFS:

(HOST):

```
$ vi switch.c

#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>

#define GPIO_EN "/sys/class/gpio/export"

#define LED_DIR "/sys/class/gpio/PC13/direction"
#define LED_VAL "/sys/class/gpio/PC13/value"

#define SW_DIR "/sys/class/gpio/PC12/direction"
#define SW_VAL "/sys/class/gpio/PC12/value"

int led_init(int);
int sw_init(int);

int sw_fd,led_fd;

int main() {
int sw_num = 76, led_num = 77;
char sw_val;

led_fd = led_init(led_num);
sw_init(sw_num);
while (1)
{
//printf("Set the value into GPIO_Val \n");
sw_fd = open(SW_VAL, O_RDONLY);
```

```
if (sw_fd < 0) {
printf("Unable to open the file %s", SW_VAL);
exit(0);
}

read(sw_fd, &sw_val, 1);

if(sw_val == '1'){
write(led_fd, "1", 1);
usleep(50);
}
else{
write(led_fd, "0", 1);
usleep(50);
}
usleep(5000);
close(sw_fd);
}
return 0;
}

int led_init(int led_num){

char led_buf[30];

led_fd = open(GPIO_EN, O_WRONLY);

if (led_fd < 0) {
printf("Unable to open the file %s\n", GPIO_EN);
exit(0);
}

//printf("Enable the GPIO");
sprintf(led_buf, "%d", led_num);
write(led_fd, led_buf, strlen(led_buf));
close(led_fd);

//printf("Configuring GPIO direction \n");
led_fd = open(LED_DIR, O_WRONLY);

if (led_fd < 0) {
printf("Unable to open the file %s", LED_DIR);
exit(0);
}
```

```
write(led_fd, "out", 3);
close(led_fd);

//printf("Set the value into GPIO_Val \n");

led_fd = open(LED_VAL, O_WRONLY);
if (led_fd < 0) {
printf("Unable to open the file %s", LED_VAL);
exit(0);
}
return led_fd;

}


int sw_init(int sw_num){

char sw_buf[30];

sw_fd = open(GPIO_EN, O_WRONLY);

if (sw_fd < 0) {
printf("Unable to open the file %s\n", GPIO_EN);
exit(0);
}

//printf("Enable the GPIO");
sprintf(sw_buf, "%d", sw_num);
write(sw_fd, sw_buf, strlen(sw_buf));
close(sw_fd);

//printf("Configuring GPIO direction \n");
sw_fd = open(SW_DIR, O_WRONLY);

if (sw_fd < 0) {
printf("Unable to open the file %s", SW_DIR);
exit(0);
}

write(sw_fd, "in", 2);
close(sw_fd);

}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} button.c -o switch

Board:

root@rugged-board-a5d2x-sd1:/home# ./switch
{ Press the user switch led will blink}.

```
gpio 76
gpio pin 76
gpio 83
gpio pin 83
```

(HOST):

vi switch_mraa.c

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

#define SW_PIN 35
#define LED_PIN 61

int main() {
mraa_init();

mraa_gpio_context sw = mraa_gpio_init(SW_PIN);
mraa_gpio_context led = mraa_gpio_init(LED_PIN);

mraa_gpio_dir(sw, MRAA_GPIO_IN);
mraa_gpio_dir(led, MRAA_GPIO_OUT);

while (1) {
if (mraa_gpio_read(sw) == 0)
mraa_gpio_write(led, 0);
else
mraa_gpio_write(led, 1); // Turn off the LED

}
}
```

Compile:

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} switch.c -o switch_mraa -lmraa

BOARD:

{ Press the user switch led will blink}.

```
gpio 76
gpio pin 76
gpio 83
gpio pin 83
```

Python:

```
import mraa

gpio_1 = mraa.Gpio(35)

gpio_2 = mraa.Gpio(63)

gpio_1.dir(mraa.DIR_IN)

gpio_2.dir(mraa.DIR_OUT)

while True:

sw_input=gpio_1.read()

if sw_input == 1:

gpio_2.write(1)

else :
gpio_2.write(0)
```

Output:

```
root@rugged-board-a5d2x-sd1:~#
root@rugged-board-a5d2x-sd1:~# python3 02_LED_SWITCH.py
gpio 76
gpio pin 76
gpio 83
gpio pin 83
```

Title: UART Loopback

Aim: Write a Program to send and Transmit data using UART Protocol.

Equipment Requirements:

A) Hardware Requirements:

1. RB-A5D2X Development board

2. USB cable

3. Male to Male Jumper

B) Software Requirements:

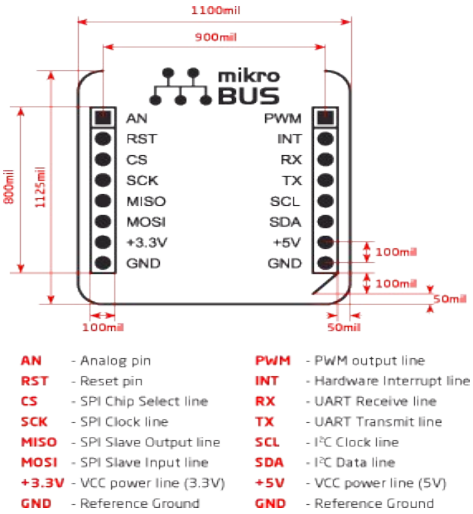
1. SDK

Connections:

1. connect Usb cable between RB-A5D2X and PC

2. use minicom to access RB-A5D2X

3. Connect jumper between TX and RX.



Loopback through command Line:
IN BOARD:
Short TX and RX pins
----> open microcom

```
root@rugged-board-a5d2x-sd1:~# microcom -s 115200 /dev/ttyS3
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
vcvucvubciacoam
```

SYSFS:

Vi uart.c

HOST:

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>

int set_interface_attribs(int fd, int speed)
{
struct termios tty;
if (tcgetattr(fd, &tty) < 0)
{
printf("Error from tcgetattr: %s\n", strerror(errno));
```

```
return -1;
}
cfsetispeed(&tty, (speed_t)speed);
cfsetospeed(&tty, (speed_t)speed);

tty.c_cflag |= (CLOCAL | CREAD); /* ignore modem controls */
tty.c_cflag &= ~CSIZE;
tty.c_cflag |= CS8; /* 8-bit characters */
tty.c_cflag &= ~PARENB; /* no parity bit */
tty.c_cflag &= ~CSTOPB; /* only need 1 stop bit */
tty.c_cflag &= ~CRTSCTS; /* no hardware flowcontrol */

tty.c_iflag = IGNPAR;
tty.c_lflag = 0; /* no signaling chars, no echo, no canonical processing */
tty.c_oflag = 0; /* no remapping, no delays */
tty.c_cc[VMIN] = 0; /* non-blocking read */
tty.c_cc[VTIME] = 10; /* 1 second read timeout */

if (tcsetattr(fd, TCSANOW, &tty) != 0)
{
printf("Error from tcsetattr: %s\n", strerror(errno));
return -1;
}
return 0;
}

int main()
{
char *portname = "/dev/ttyS3";
int fd;
ssize_t wlen, rdlen;
unsigned char buf[100];
char input[100];

fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
printf("Error opening %s: %s\n", portname, strerror(errno));
return -1;
}

if (set_interface_attribs(fd, B9600) < 0)
{
close(fd);
return -1;
}

printf("Enter text to send via serial (press ENTER to send, Ctrl+D to quit):\n");

while (fgets(input, sizeof(input), stdin) != NULL)
```

```
{
size_t len = strlen(input);

// Remove newline character from input if present
if (len > 0 && input[len - 1] == '\n')
{
input[len - 1] = '\0';
len--;
}

if (len == 0)
{
printf("Empty input, please enter something.\n");
continue;
}

// Write input to serial port
wlen = write(fd, input, len);
if (wlen != len)
{
printf("Error from write: %ld, %s\n", wlen, strerror(errno));
}

// Try reading response from serial port (non-blocking)
rdlen = read(fd, buf, sizeof(buf) - 1);
if (rdlen > 0)
{
buf[rdlen] = '\0'; // Null-terminate
printf("Received: %s\n", buf);
}
else if (rdlen < 0)
{
printf("Error from read: %s\n", strerror(errno));
}
}

printf("Exiting...\n");
close(fd);
return 0;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} uart.c -o uart

IN BOARD:

```
root@rugged-board-a5d2x-sd1:/home# ./uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
Enter text to send via serial (press ENTER to send, Ctrl+D to quit):
Hello All How are you!
Received: Hello All How are you!
```

Vi uart_mraa.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mraa.h>

#define RFID_UART_PORT "/dev/ttyS3"

int main() {
mraa_init();
char buffer[100]={0};
mraa_uart_context uart = mraa_uart_init_raw(RFID_UART_PORT);

while (1) {

    fgets(buffer,sizeof(buffer),stdin);
/* send data through UART */
mraa_uart_write(uart,buffer,sizeof(buffer));

usleep(10000);
mraa_uart_read(uart, buffer,sizeof(buffer));
    printf("%s\n",buffer);

}
}
```

compile:

```
. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} uart_mraa.c -o mraa_uart
```

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./mraa_uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
hello all
hello all

how are you
how are you
```

Python:

```
import mraa
import time
import sys

# serial port
port = "/dev/ttyS3"

data = "Greetings from PHYTEC!"
```

```
# initialise UART
uart = mraa.Uart(port)

# send data through UART
uart.write(bytearray(data, 'utf-8'))
time.sleep(1)

while True:
    if uart.dataAvailable():
        data_byte = uart.readStr(1)
        print(data_byte,end=" ")
    if data_byte == "!":
        print("")
        break
```

IN Board:

```
root@rugged-board-a5d2x-sd1:~# python3 uart.py
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
G r e e t i n g s   f r o m   P H Y T E C !
root@rugged-board-a5d2x-sd1:~#
```

Title : RS232 LoopBack (ttyS1)

Aim: Write a Program to send and Transmit data using RS232 Protocol.

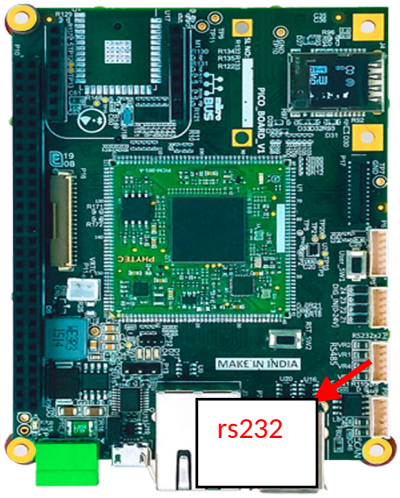
Equipment Requirements:

A) Hardware Requirements :

1. RB-A5D2X Development board
2. USB cable
3. 5-pins Patch cord and male to male jumperB)

B) Connections:

4. connect Usb cable between RB-A5D2X and PC
5. use minicom to access RB-A5D2X
6. Connect jumper between RS232_TX_1 and RS232_RX_1.



Command Line:(ttyS1)

```
root@rugged-board-a5d2x-sd1:/home# microcom -s 115200 /dev/ttyS1
atmel_usart_serial atmel_usart_serial.0.auto: using dma0chan7 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.0.auto: using dma0chan8 for tx DMA transfers
Hello all!
```

HOST:

Vi rs232.c

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>

int set_interface_attribs(int fd, int speed)
{
struct termios tty;
if (tcgetattr(fd, &tty) < 0)
{
printf("Error from tcgetattr: %s\n", strerror(errno));
return -1;
}
cfsetispeed(&tty, (speed_t)speed);
cfsetospeed(&tty, (speed_t)speed);

tty.c_cflag |= (CLOCAL | CREAD); /* ignore modem controls */
tty.c_cflag &= ~CSIZE;
tty.c_cflag |= CS8; /* 8-bit characters */
tty.c_cflag &= ~PARENB; /* no parity bit */
tty.c_cflag &= ~CSTOPB; /* only need 1 stop bit */
tty.c_cflag &= ~CRTSCTS; /* no hardware flowcontrol */

tty.c_iflag = IGNPAR;
tty.c_lflag = 0; /* no signaling chars, no echo, no canonical processing */
```

```
tty.c_oflag = 0; /* no remapping, no delays */
tty.c_cc[VMIN] = 0; /* non-blocking read */
tty.c_cc[VTIME] = 10; /* 1 second read timeout */

if (tcsetattr(fd, TCSANOW, &tty) != 0)
{
printf("Error from tcsetattr: %s\n", strerror(errno));
return -1;
}
return 0;
}

int main()
{
char *portname = "/dev/ttyS1";
int fd;
ssize_t wlen, rdlen;
unsigned char buf[100];
char input[100];

fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
printf("Error opening %s: %s\n", portname, strerror(errno));
return -1;
}

if (set_interface_attribs(fd, B9600) < 0)
{
close(fd);
return -1;
}

printf("Enter text to send via serial (press ENTER to send, Ctrl+D to quit):\n");

while (fgets(input, sizeof(input), stdin) != NULL)
{
size_t len = strlen(input);

// Remove newline character from input if present
if (len > 0 && input[len - 1] == '\n')
{
input[len - 1] = '\0';
len--;
}

if (len == 0)
{
printf("Empty input, please enter something.\n");
continue;
```

```
}

// Write input to serial port
wlen = write(fd, input, len);
if (wlen != len)
{
printf("Error from write: %ld, %s\n", wlen, strerror(errno));
}

// Try reading response from serial port (non-blocking)
rdlen = read(fd, buf, sizeof(buf) - 1);
if (rdlen > 0)
{
buf[rdlen] = '\0'; // Null-terminate
printf("Received: %s\n", buf);
}
else if (rdlen < 0)
{
printf("Error from read: %s\n", strerror(errno));
}
}

printf("Exiting...\n");
close(fd);
return 0;
}
```

compile :
. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} rs232.c -o uart

IN BOARD:

```
root@rugged-board-a5d2x-sd1:/home# ./uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
Enter text to send via serial (press ENTER to send, Ctrl+D to quit):
Hello All How are you!
Received: Hello All How are you!
```

MRAA:

Vi rs232_mraa.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mraa.h>

#define RFID_UART_PORT "/dev/ttyS1"

int main() {
mraa_init();
char buffer[100]={0};
```

```
mraa_uart_context uart = mraa_uart_init_raw(RFID_UART_PORT);

while (1) {

    fgets(buffer,sizeof(buffer),stdin);
/* send data through UART */
mraa_uart_write(uart,buffer,sizeof(buffer));

usleep(10000);
mraa_uart_read(uart, buffer,sizeof(buffer));
    printf("%s\n",buffer);

}
}
```

compile:

```
. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} rs232_mraa.c -o mraa_uart
```

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./mraa_uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
hello all
hello all

how are you
how are you
```

Python:

```
import mraa
import time
import sys

# serial port
port = "/dev/ttyS1"

data = "Greetings from PHYTEC!"

# initialise UART
uart = mraa.Uart(port)

# send data through UART
uart.write(bytearray(data, 'utf-8'))
time.sleep(1)

while True:
    if uart.dataAvailable():
        data_byte = uart.readStr(1)
```

```
        print(data_byte,end=" ")
    if data_byte == "!":
        print("")
        break
```

IN Board:

```
root@rugged-board-a5d2x-sd1:~# python3 uart.py
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
G r e e t i n g s   f r o m   P H Y T E C !
root@rugged-board-a5d2x-sd1:~#
```

Title : RS232 LoopBack (ttyS4)

Aim: Write a Program to send and Transmit data using RS232 Protocol.

Equipment Requirements:

A) Hardware Requirements :

1. RB-A5D2X Development board
2. USB cable
3. 5-pins Patch cord and male to male jumper

B) Software Requirements:

Python3

Connections:

4. connect Usb cable between RB-A5D2X and PC
5. use minicom to access RB-A5D2X
6. Connect jumper between RS232_TX_2 and RS232_RX_2

Command Line:(ttyS4)

```
root@rugged-board-a5d2x-sd1:/home# microcom -s 115200 /dev/ttyS1
atmel_usart_serial atmel_usart_serial.0.auto: using dma0chan7 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.0.auto: using dma0chan8 for tx DMA transfers
Hello all!
```

Sysfs:

HOST:

Vi rs232.c

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
```

```
int set_interface_attribs(int fd, int speed)
{
struct termios tty;
if (tcgetattr(fd, &tty) < 0)
{
printf("Error from tcgetattr: %s\n", strerror(errno));
return -1;
}
cfsetispeed(&tty, (speed_t)speed);
cfsetospeed(&tty, (speed_t)speed);

tty.c_cflag |= (CLOCAL | CREAD); /* ignore modem controls */
tty.c_cflag &= ~CSIZE;
tty.c_cflag |= CS8; /* 8-bit characters */
tty.c_cflag &= ~PARENB; /* no parity bit */
tty.c_cflag &= ~CSTOPB; /* only need 1 stop bit */
tty.c_cflag &= ~CRTSCTS; /* no hardware flowcontrol */

tty.c_iflag = IGNPAR;
tty.c_lflag = 0; /* no signaling chars, no echo, no canonical processing */
tty.c_oflag = 0; /* no remapping, no delays */
tty.c_cc[VMIN] = 0; /* non-blocking read */
tty.c_cc[VTIME] = 10; /* 1 second read timeout */

if (tcsetattr(fd, TCSANOW, &tty) != 0)
{
printf("Error from tcsetattr: %s\n", strerror(errno));
return -1;
}
return 0;
}

int main()
{
char *portname = "/dev/ttyS4";
int fd;
ssize_t wlen, rdlen;
unsigned char buf[100];
char input[100];

fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
printf("Error opening %s: %s\n", portname, strerror(errno));
return -1;
}

if (set_interface_attribs(fd, B9600) < 0)
{
close(fd);
```

```
return -1;
}

printf("Enter text to send via serial (press ENTER to send, Ctrl+D to quit):\n");

while (fgets(input, sizeof(input), stdin) != NULL)
{
size_t len = strlen(input);

// Remove newline character from input if present
if (len > 0 && input[len - 1] == '\n')
{
input[len - 1] = '\0';
len--;
}

if (len == 0)
{
printf("Empty input, please enter something.\n");
continue;
}

// Write input to serial port
wlen = write(fd, input, len);
if (wlen != len)
{
printf("Error from write: %ld, %s\n", wlen, strerror(errno));
}

// Try reading response from serial port (non-blocking)
rdlen = read(fd, buf, sizeof(buf) - 1);
if (rdlen > 0)
{
buf[rdlen] = '\0'; // Null-terminate
printf("Received: %s\n", buf);
}
else if (rdlen < 0)
{
printf("Error from read: %s\n", strerror(errno));
}
}

printf("Exiting...\n");
close(fd);
return 0;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} rs232.c -o uart

IN BOARD:

```
root@rugged-board-a5d2x-sd1:/home# ./uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
Enter text to send via serial (press ENTER to send, Ctrl+D to quit):
Hello All How are you!
Received: Hello All How are you!
```

MRAA:

Vi rs232_mraa.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mraa.h>

#define RFID_UART_PORT "/dev/ttyS4"

int main() {
mraa_init();
char buffer[100]={0};
mraa_uart_context uart = mraa_uart_init_raw(RFID_UART_PORT);

while (1) {

    fgets(buffer,sizeof(buffer),stdin);
/* send data through UART */
mraa_uart_write(uart,buffer,sizeof(buffer));

usleep(10000);
mraa_uart_read(uart, buffer,sizeof(buffer));
    printf("%s\n",buffer);

}
}
```

compile:

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} rs232_mraa.c -o mraa_uart

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./mraa_uart
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
hello all
hello all

how are you
how are you
```

Python:

```
import mraa
import time
```

```
import sys

# serial port
port = "/dev/ttyS4"

data = "Greetings from PHYTEC!"

# initialise UART
uart = mraa.Uart(port)

# send data through UART
uart.write(bytearray(data, 'utf-8'))
time.sleep(1)

while True:
    if uart.dataAvailable():
        data_byte = uart.readStr(1)
        print(data_byte,end=" ")
    if data_byte == "!":
        print("")
        break
```

IN Board:

```
root@rugged-board-a5d2x-sd1:~# python3 uart.py
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan0 for rx DMA transfers
atmel_usart_serial atmel_usart_serial.3.auto: using dma1chan1 for tx DMA transfers
G r e e t i n g s   f r o m   P H Y T E C !
root@rugged-board-a5d2x-sd1:~#
```

Title : PWM interfacing

Aim: Write a Program to control brightness of led using PWM .

Equipment Requirements:

A) Hardware Requirements:

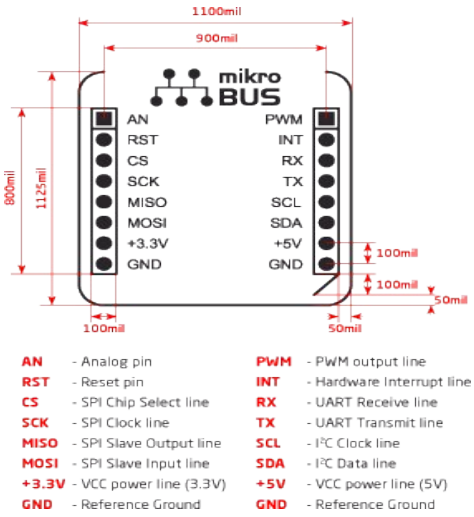
1. RB-A5D2X Development board
2. USB cable
3. Led

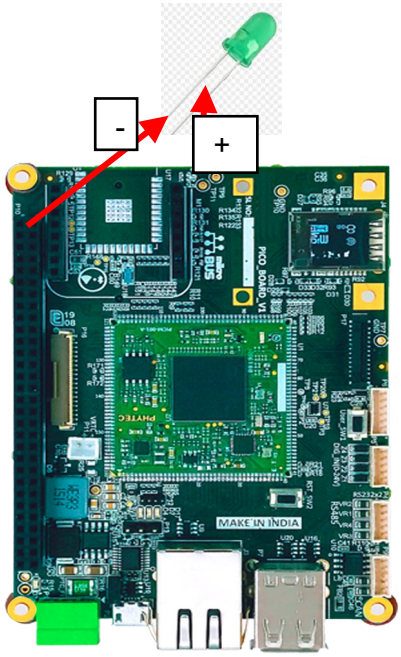
B) Software Requirements :

1. python3

Connections:

1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X





Expected output::

you will see brightness of led getting Low from HIGH brightness.

Export PWM1

echo 1 > /sys/class/pwm/pwmchip0/export

Set the period (must be set before duty_cycle)

echo 500000 > /sys/class/pwm/pwmchip0/pwm1/period

Enable PWM1

echo 1 > /sys/class/pwm/pwmchip0/pwm1/enable

Set duty cycle (start with 100)

echo 50 > /sys/class/pwm/pwmchip0/pwm1/duty_cycle

SYSFS:

In HOST(side) :
Vi pwm.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<fcntl.h>
#include<unistd.h>
#include<time.h>


char export_path[] = "/sys/class/pwm/pwmchip0/export";
```

```
char pwm_periods[] = "/sys/class/pwm/pwmchip0/pwm1/period";
char pwm_enable[] = "/sys/class/pwm/pwmchip0/pwm1/enable";
char duty_cycle[] = "/sys/class/pwm/pwmchip0/pwm1/duty_cycle";


// ./a.out 100 1
// period duty cycle
int main(int argc,char **argv)
{
    int fd;
    int value,period;
    char *value_str;
    value_str=malloc(4);
    fd = open(export_path,O_WRONLY);
    if(fd < 0)
        perror("export path error \n");
    write(fd,"1",2);
    close(fd);

    fd = open(pwm_periods,O_WRONLY);
    if(fd < 0)
        perror("period error \n");
    write(fd,argv[1],4);
    close(fd);

    fd = open(pwm_enable,O_WRONLY);
    if(fd < 0)
        perror("enable error \n");
    write(fd,"1",2);
    close(fd);

    period = atoi(argv[1]);

    value_str = argv[2];
    value = atoi(argv[2]);

    fd = open(duty_cycle,O_WRONLY);

    while(1)
    {
    start : sprintf(value_str,"%d",value);
        write(fd,value_str,sizeof(value_str));
        printf("pwm value : %d \n",value);
        value++;
        usleep(100000);
        if(value >= period)
        {
            value=0;
            goto start;
        }
```

```
    }
    close(fd);
}
```

How to compile Application:

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} pwm.c -o pwm

How to execute:

./pwm 100 1
pwm --> executable file
100 --> no of periods
1
--> no of duty cycle.

```
root@rugged-board-a5d2x-sd1:~# ./pwm  100 1
value : 1
value : 2
value : 3
value : 4
value : 5
```

MRAA:

Vi pwm_mraa.c

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

#define PWM_PIN 72 // PWM pin number
#define PWM_PERIOD 20 // PWM period in microseconds

int main() {
mraa_pwm_context pwm;
float output;
// Initialize PWM pin
pwm = mraa_pwm_init(PWM_PIN);
if (pwm == NULL) {
printf("Failed to initialize PWM\n");
return 1;
}
// Set PWM period
mraa_pwm_period_ms(pwm, PWM_PERIOD);
mraa_pwm_enable(pwm, 1);
float value = 1.0;
while (1) {
```

```
// Write PWM value
mraa_pwm_write(pwm, value);
usleep(70000);
value -= 0.01;
if (value <= 0.0)
value = 1;
/* read PWM duty cyle */
output = mraa_pwm_read(pwm);
printf("PWM value : %f\n", output);
}
// Cleanup and close PWM pin
mraa_pwm_enable(pwm, 0);
mraa_pwm_close(pwm);
return 0;
}
```

How to compile Application:

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} pwm_mraa.c -o mraa_pwm –lmraa

Output:

```
root@rugged-board-a5d2x-sd1:/home# ./mraa_pwm
PWM value : 1.000000
PWM value : 0.990000
PWM value : 0.980000
PWM value : 0.970000
PWM value : 0.960000
PWM value : 0.950000
PWM value : 0.940000
PWM value : 0.930000
PWM value : 0.920000
PWM value : 0.910000
PWM value : 0.900000
PWM value : 0.890000
PWM value : 0.880000
PWM value : 0.870000
PWM value : 0.860000
PWM value : 0.850000
PWM value : 0.840000
PWM value : 0.830000
PWM value : 0.820000
PWM value : 0.810000
PWM value : 0.800000
```

Python:

```
Vi pwm.py
import mraa
import time

# initialise PWM
x = mraa.Pwm(72)

# set PWM period
```

```
x.period_us(700)

# enable PWM
x.enable(True)

value= 0.0

while True:
    # write PWM value
    x.write(value)

    time.sleep(0.05)

    value = value + 0.01
    if value >= 1:
        value = 0.0
```

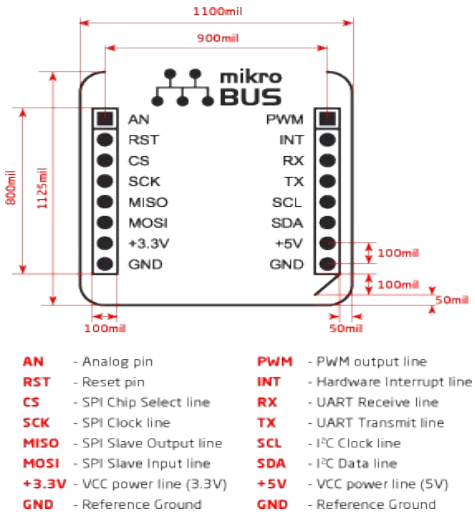
IN Board:

root@rugged-board-a5d2x-sd1:/home# python3 pwm.py
{see the led }

Title : ADC interfacing

Aim: Write a Program to get analog input using ADC.
Equipment Requirements:
A) Hardware Requirements :
1. RB-A5D2X Development board
2. USB cable
3. Potentiometer

Connections:
1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X



Expected output::

As you move potentiometer values, the output data will be vary.

Output:(In Board)

cat /sys/bus/iio/devices/iio:device0/in_voltage3_raw

root@ruggedboard-imx6ul:iio:device0# cat in_voltage3_raw

4095

root@ruggedboard-imx6ul:iio:device0# cat in_voltage3_raw

2083

root@ruggedboard-imx6ul:iio:device0# cat in_voltage3_raw

1055

SYSFS:

Vi adc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

// Read ADC value from given channel
int read_adc(int channel)
{
char path[64];
char value_str[8];
int fd;

// Construct sysfs path
snprintf(path, sizeof(path),
"/sys/bus/iio/devices/iio:device0/in_voltage%d_raw", channel);

// Open ADC sysfs file
```

```
fd = open(path, O_RDONLY);
if (fd < 0) {
perror("Failed to open ADC file");
return -1;
}

// Read the value
if (read(fd, value_str, sizeof(value_str)) < 0) {
perror("Failed to read ADC value");
close(fd);
return -1;
}

close(fd);
return atoi(value_str); // Convert string to integer
}

int main(int argc, char *argv[])
{
if (argc != 2) {
printf("Usage: %s <adc_channel>\n", argv[0]);
return 1;
}

int channel = atoi(argv[1]);

while (1) {
int value = read_adc(channel);
if (value >= 0) {
printf("ADC Channel %d Value: %d\n", channel, value);
}

usleep(500000); // 0.5 second delay
}

return 0;
}
```

compile :
. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi
${CC} adc.c -o sysfs_adc
Output:

```
root@rugged-board-a5d2x-sd1:/home# ./sysfs_adc 6
ADC Channel 6 Value: 1790
ADC Channel 6 Value: 1792
ADC Channel 6 Value: 1793
ADC Channel 6 Value: 1794
ADC Channel 6 Value: 1791
ADC Channel 6 Value: 1795
ADC Channel 6 Value: 1796
ADC Channel 6 Value: 1802
ADC Channel 6 Value: 1798
ADC Channel 6 Value: 1792
ADC Channel 6 Value: 1796
ADC Channel 6 Value: 1795
ADC Channel 6 Value: 1792
ADC Channel 6 Value: 1794
ADC Channel 6 Value: 1784
ADC Channel 6 Value: 1768
ADC Channel 6 Value: 1724
ADC Channel 6 Value: 1708
ADC Channel 6 Value: 1682
ADC Channel 6 Value: 1655
ADC Channel 6 Value: 1631
ADC Channel 6 Value: 1617
ADC Channel 6 Value: 1585
ADC Channel 6 Value: 1555
```

MRAA:

Vi adc_mraa.c

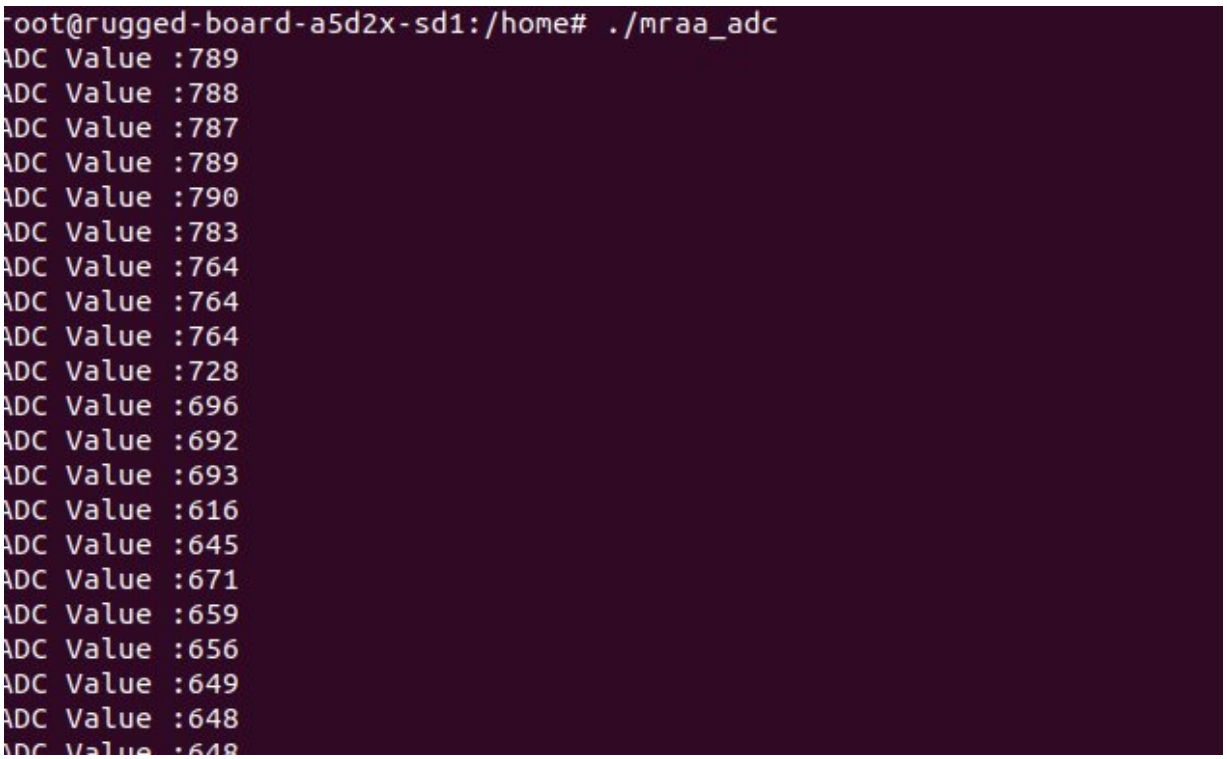
```
#include <stdlib.h>
#include <unistd.h>
/* mraa header */
#include "mraa.h"
/* AIO port */
#define AIO_PORT 6
int main()
{
mraa_init();
mraa_aio_context aio;
uint16_t value = 0;
/* initialize AIO */
aio = mraa_aio_init(AIO_PORT);
while (1) {
value = mraa_aio_read(aio);
printf("ADC Value :%d\n",value);
sleep(2);
}
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} adc_mraa.c -o mraa_adc –lmraa

Output:



Python:

Vi adc.py

```
import mraa
import time
print(mraa.getVersion())

while True:
    # initialise AIO
    x = mraa.Aio(6)

    # read integer value
    print(x.read())

    # read float value
    print("%.5f" % x.readFloat())
    time.sleep(1)
```

```
root@rugged-board-a5d2x-sd1:/home# python3 adc.py
v2.0.0
656
0.64223
658
0.64223
659
0.64516
667
0.65103
683
0.66667
696
0.67937
713
0.69697
717
0.69990
722
0.70577
739
0.72239
755
0.73900
757
0.73900
```

Title : I2C EEPROM Reading memory

Aim: Write a Program to read data from EEPROM using I2C Protocol.

Equipment Requirements:

A) Hardware Requirements :

1. RB-A5D2X Development board
2. USB cable
3. EEPROM is present on board.

B) Software Requirements :

1. python3

Connections:

1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X

Command line:

root@ruggedboard-imx6ul : echo phytec_hello > /sys/bus/i2c/devices/0-0050/eeprom

root@ruggedboard-imx6ul: cat /sys/bus/i2c/devices/0-0050/eeprom

```
root@rugged-board-a5d2x-sd1:~# echo phytec_hello >  /sys/bus/i2c/devices/0-0050/eeprom
root@rugged-board-a5d2x-sd1:~# cat /sys/bus/i2c/devices/0-0050/eeprom
phytec_hello
♦♦♦
```

IN host :

Vi i2c.c

```
#include <linux/types.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include "i2c-dev.h"

char buffer[20];

int main() {
char i2c_dev_node_path[] = "/dev/i2c-0";
int i2c_dev_node;
int ret_val = 0;

i2c_dev_node = open(i2c_dev_node_path, O_RDWR);
if (i2c_dev_node < 0) {
perror("Unable to open device node.");
exit(1);
}

int i2c_dev_address = 0x50; // EEPROM address
ret_val = ioctl(i2c_dev_node, I2C_SLAVE, i2c_dev_address);
if (ret_val < 0) {
perror("Could not set I2C_SLAVE.");
exit(2);
}

int i2c_dev_reg_addr = 0x00; // EEPROM start location
unsigned int i2c_val_to_write = 0x40; // Writing '@' (ASCII 64)

ret_val = i2c_smbus_write_byte_data(i2c_dev_node, i2c_dev_reg_addr, i2c_val_to_write);
if (ret_val < 0) {
perror("I2C Write Operation failed.");
exit(4);
}

sleep(1); // Wait before reading back

int i2c_dev_reg_x_acc = 0x00; // Start reading from 0x00

while (1) {
s32 read_value = i2c_smbus_read_byte_data(i2c_dev_node, i2c_dev_reg_x_acc);
if (read_value < 0) {
perror("I2C Read operation failed.");
exit(3);
```

```
}

// Print result: raw value, char, address
printf("Addr 0x%02X = 0x%02X '%c'\n", i2c_dev_reg_x_acc, read_value,
(read_value >= 32 && read_value <= 126) ? read_value : '.');

i2c_dev_reg_x_acc++;
if (i2c_dev_reg_x_acc >= 256) break; // stop after full EEPROM read
sleep(1);
}

close(i2c_dev_node);
return 0;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} i2c.c -o eeprom_c

```
root@rugged-board-a5d2x-sd1:/home# ./eeprom_c
Addr 0x00 = 0x40  '@'
Addr 0x01 = 0x65  'e'
Addr 0x02 = 0x6C  'l'
Addr 0x03 = 0x6C  'l'
Addr 0x04 = 0x6F  'o'
Addr 0x05 = 0x5F  '_'
Addr 0x06 = 0x61  'a'
Addr 0x07 = 0x6C  'l'
Addr 0x08 = 0x6C  'l'
Addr 0x09 = 0x0A  '.'
Addr 0x0A = 0x68  'h'
Addr 0x0B = 0x6F  'o'
Addr 0x0C = 0x77  'w'
Addr 0x0D = 0x20  ' '
Addr 0x0E = 0x61  'a'
Addr 0x0F = 0x72  'r'
Addr 0x10 = 0x65  'e'
Addr 0x11 = 0x20  ' '
Addr 0x12 = 0x79  'y'
Addr 0x13 = 0x6F  'o'
Addr 0x14 = 0x75  'u'
Addr 0x15 = 0x0A  '.'
Addr 0x16 = 0xFF  '.'
```

MRAA:

Vi mraa_adc.c

```
#include <stdio.h>
#include <unistd.h>
#include <mraa/i2c.h>

#define I2C_BUS 0 // I2C bus number, usually 0 or 1
```

```
#define EEPROM_ADDR 0x50 // EEPROM I2C address
#define START_ADDR 0x00 // EEPROM memory address to start reading
#define READ_LEN 32 // Number of bytes to read

int main() {
mraa_i2c_context i2c;
int i;
char buffer[READ_LEN + 1] = {0}; // +1 for null terminator

// Initialize I2C
i2c = mraa_i2c_init(I2C_BUS);
if (!i2c) {
fprintf(stderr, "Failed to initialize I2C\n");
return 1;
}

// Set EEPROM I2C address
if (mraa_i2c_address(i2c, EEPROM_ADDR) != MRAA_SUCCESS) {
fprintf(stderr, "Failed to set I2C address\n");
mraa_i2c_stop(i2c);
return 1;
}

// Read bytes from EEPROM
for (i = 0; i < READ_LEN; i++) {
int val = mraa_i2c_read_byte_data(i2c, START_ADDR + i);
if (val < 0) {
fprintf(stderr, "Read failed at offset 0x%02X\n", START_ADDR + i);
mraa_i2c_stop(i2c);
return 1;
}
buffer[i] = (char)val;
}

buffer[READ_LEN] = '\0'; // Null terminate for safe string output
printf("EEPROM Data: \"%s\"\n", buffer);

// Clean up
mraa_i2c_stop(i2c);
mraa_deinit();

return 0;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} mraa_eeprom.c -o mraa_eeprom –lmraa

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./mraa_eeprom
EEPROM Data: "Hello_all how are you
◆◆◆◆◆◆◆◆◆◆◆"
root@rugged-board-a5d2x-sd1:/home#
```

Python:

Vi i2c1.py

```
import mraa
import time

x = mraa.I2c(0)        # Initialize I2C on bus 0
x.address(0x50)        # Set EEPROM I2C address

for i in range(256):    # Limit to 256 bytes
    value = x.readReg(i)
    if 32 <= value <= 126:    # Printable ASCII range
        print(chr(value), end="")
    else:
        print(".", end="")    # Show . for non-printable chars
    time.sleep(0.005)        # Small delay for stability

print()
```

In Board :

Python3 i2c1.py

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# python3 i2c1.py
Hello_all.how are you.....N
root@rugged-board-a5d2x-sd1:/home#
```

Title : SPI LoopBack interfacing

Aim: Write a Program to send and Transmit data using SPI.

Equipment Requirements:

A) Hardware Requirements :

1. RB-A5D2X Development board
2. USB cable
3. male to male jumper wire.

B) Software Requirements :

1. python3

Connections:

1. connect Usb cable between RB-A5D2X and PC
2. use minicom to access RB-A5D2X

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./spi
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
RX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF F0 0D  | ......@....•.................•.
root@rugged-board-a5d2x-sd1:/home#
```

In host :

SYSFS:

Vi spidev.c

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/ioctl.h>
#include <sys/stat.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
static void pabort(const char *s)
{
perror(s);
abort();
}
static const char *device = "/dev/spidev3.0";
static uint32_t mode;
static uint8_t bits = 8;
static char *input_file;
static char *output_file;
static uint32_t speed = 500000;
static uint16_t delay;
static int verbose;
uint8_t default_tx[] = {
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xF0, 0x0D,
};
uint8_t default_rx[ARRAY_SIZE(default_tx)] = {0, };
char *input_tx;
static void hex_dump(const void *src, size_t length, size_t line_size,
char *prefix)
{
int i = 0;
const unsigned char *address = src;
const unsigned char *line = address;
```

```
unsigned char c;
printf("%s | ", prefix);
while (length-- > 0) {
printf("%02X ", *address++);
if (!(++i % line_size) || (length == 0 && i % line_size)) {
if (length == 0) {
while (i++ % line_size)
printf("__ ");
}
printf(" | "); /* right close */
while (line < address) {
c = *line++;
printf("%c", (c < 33 || c == 255) ? 0x2E : c);
}
printf("\n");
if (length > 0)
printf("%s | ", prefix);
}
}
}
/*
* Unescape - process hexadecimal escape character
* converts shell input "\x23" -> 0x23
*/
static int unescape(char *_dst, char *_src, size_t len)
{
int ret = 0;
int match;
char *src = _src;
char *dst = _dst;
unsigned int ch;
while (*src) {
if (*src == '\\' && *(src+1) == 'x') {
match = sscanf(src + 2, "%2x", &ch);
if (!match)
pabort("malformed input string");
src += 4;
*dst++ = (unsigned char)ch;
} else {
*dst++ = *src++;
}
ret++;
}
return ret;
}
static void transfer(int fd, uint8_t const *tx, uint8_t const *rx, size_t len)
{
int ret;
int out_fd;
struct spi_ioc_transfer tr = {
.tx_buf = (unsigned long)tx,
```

```
.rx_buf = (unsigned long)rx,
.len = len,
.delay_usecs = delay,
.speed_hz = speed,
.bits_per_word = bits,
};
if (mode & SPI_TX_QUAD)
tr.tx_nbits = 4;
else if (mode & SPI_TX_DUAL)
tr.tx_nbits = 2;
if (mode & SPI_RX_QUAD)
tr.rx_nbits = 4;
else if (mode & SPI_RX_DUAL)
tr.rx_nbits = 2;
if (!(mode & SPI_LOOP)) {
if (mode & (SPI_TX_QUAD | SPI_TX_DUAL))
tr.rx_buf = 0;
else if (mode & (SPI_RX_QUAD | SPI_RX_DUAL))
tr.tx_buf = 0;
}
ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
if (ret < 1)
pabort("can't send spi message");
if (verbose)
hex_dump(tx, len, 32, "TX");
if (output_file) {
out_fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if (out_fd < 0)
pabort("could not open output file");
ret = write(out_fd, rx, len);
if (ret != len)
pabort("not all bytes written to output file");
close(out_fd);
}
if (verbose || !output_file)
hex_dump(rx, len, 32, "RX");
}
static void print_usage(const char *prog)
{
printf("Usage: %s [-DsbdlHOLC3]\n", prog);
puts(" -D --device device to use (default /dev/spidev1.1)\n"
" -s --speed max speed (Hz)\n"
" -d --delay delay (usec)\n"
" -b --bpw bits per word\n"
" -i --input input data from a file (e.g. \"test.bin\")\n"
" -o --output output data to a file (e.g. \"results.bin\")\n"
" -l --loop loopback\n"
" -H --cpha clock phase\n"
" -O --cpol clock polarity\n"
" -L --lsb least significant bit first\n"
" -C --cs-high chip select active high\n"
```

```
" -3 --3wire SI/SO signals shared\n"
" -v --verbose Verbose (show tx buffer)\n"
" -p Send data (e.g. \"1234\\xde\\xad\")\n"
" -N --no-cs no chip select\n"
" -R --ready slave pulls low to pause\n"
" -2 --dual dual transfer\n"
" -4 --quad quad transfer\n");
exit(1);
}
static void parse_opts(int argc, char *argv[])
{
while (1) {
static const struct option lopts[] = {
{ "device", 1, 0, 'D' },
{ "speed", 1, 0, 's' },
{ "delay", 1, 0, 'd' },
{ "bpw", 1, 0, 'b' },
{ "input", 1, 0, 'i' },
{ "output", 1, 0, 'o' },
{ "loop", 0, 0, 'l' },
{ "cpha", 0, 0, 'H' },
{ "cpol", 0, 0, 'O' },
{ "lsb", 0, 0, 'L' },
{ "cs-high", 0, 0, 'C' },
{ "3wire", 0, 0, '3' },
{ "no-cs", 0, 0, 'N' },
{ "ready", 0, 0, 'R' },
{ "dual", 0, 0, '2' },
{ "verbose", 0, 0, 'v' },
{ "quad", 0, 0, '4' },
{ NULL, 0, 0, 0 },
};
int c;
c = getopt_long(argc, argv, "D:s:d:b:i:o:lHOLC3NR24p:v",
lopts, NULL);
if (c == -1)
break;
switch (c) {
case 'D':
device = optarg;
break;
case 's':
speed = atoi(optarg);
break;
case 'd':
delay = atoi(optarg);
break;
case 'b':
bits = atoi(optarg);
break;
case 'i':
```

```
input_file = optarg;
break;
case 'o':
output_file = optarg;
break;
case 'l':
mode |= SPI_LOOP;
break;
case 'H':
mode |= SPI_CPHA;
break;
case 'O':
mode |= SPI_CPOL;
break;
case 'L':
mode |= SPI_LSB_FIRST;
break;
case 'C':
mode |= SPI_CS_HIGH;
break;
case '3':
mode |= SPI_3WIRE;
break;
case 'N':
mode |= SPI_NO_CS;
break;
case 'v':
verbose = 1;
break;
case 'R':
mode |= SPI_READY;
break;
case 'p':
input_tx = optarg;
break;
case '2':
mode |= SPI_TX_DUAL;
break;
case '4':
mode |= SPI_TX_QUAD;
break;
default:
print_usage(argv[0]);
break;
}
}
if (mode & SPI_LOOP) {
if (mode & SPI_TX_DUAL)
mode |= SPI_RX_DUAL;
if (mode & SPI_TX_QUAD)
mode |= SPI_RX_QUAD;
```

```
}
}
static void transfer_escaped_string(int fd, char *str)
{
size_t size = strlen(str);
uint8_t *tx;
uint8_t *rx;
tx = malloc(size);
if (!tx)
pabort("can't allocate tx buffer");
rx = malloc(size);
if (!rx)
pabort("can't allocate rx buffer");
size = unescape((char *)tx, str, size);
transfer(fd, tx, rx, size);
free(rx);
free(tx);
}
static void transfer_file(int fd, char *filename)
{
ssize_t bytes;
struct stat sb;
int tx_fd;
uint8_t *tx;
uint8_t *rx;
if (stat(filename, &sb) == -1)
pabort("can't stat input file");
tx_fd = open(filename, O_RDONLY);
if (fd < 0)
pabort("can't open input file");
tx = malloc(sb.st_size);
if (!tx)
pabort("can't allocate tx buffer");
rx = malloc(sb.st_size);
if (!rx)
pabort("can't allocate rx buffer");
bytes = read(tx_fd, tx, sb.st_size);
if (bytes != sb.st_size)
pabort("failed to read input file");
transfer(fd, tx, rx, sb.st_size);
free(rx);
free(tx);
close(tx_fd);
}
int main(int argc, char *argv[])
{
int ret = 0;
int fd;
parse_opts(argc, argv);
fd = open(device, O_RDWR);
if (fd < 0)
```

```
pabort("can't open device");
/*
* spi mode
*/
ret = ioctl(fd, SPI_IOC_WR_MODE32, &mode);
if (ret == -1)
pabort("can't set spi mode");
ret = ioctl(fd, SPI_IOC_RD_MODE32, &mode);
if (ret == -1)
pabort("can't get spi mode");
/*
* bits per word
*/
ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
if (ret == -1)
pabort("can't set bits per word");
ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
if (ret == -1)
pabort("can't get bits per word");
/*
* max speed hz
*/
ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
pabort("can't set max speed hz");
ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
if (ret == -1)
pabort("can't get max speed hz");
printf("spi mode: 0x%x\n", mode);
printf("bits per word: %d\n", bits);
printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);
if (input_tx && input_file)
pabort("only one of -p and --input may be selected");
if (input_tx)
transfer_escaped_string(fd, input_tx);
else if (input_file)
transfer_file(fd, input_file);
else
transfer(fd, default_tx, default_rx, sizeof(default_tx));
close(fd);
return ret;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} spidev.c -o spi

```
root@rugged-board-a5d2x-sd1:/home#
root@rugged-board-a5d2x-sd1:/home# ./spi
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
RX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF F0 0D  | ......@.....................
root@rugged-board-a5d2x-sd1:/home#
```

MRAA:

vi mraa_spi.c

```
#include <mraa/spi.h>
#include <stdio.h>
#include <unistd.h>

int main() {
// Initialize SPI on bus 0
mraa_spi_context spi;
spi = mraa_spi_init(0);

if (spi == NULL) {
fprintf(stderr, "Failed to initialize SPI\n");
return 1;
}

// Prepare data to send
uint8_t txbuf[3] = {100, 123, 75};
uint8_t rxbuf[3];

while (1) {
// Perform SPI transaction (write and read)
for (int i = 0; i < 3; i++) {
rxbuf[i] = mraa_spi_write(spi, txbuf[i]);
}

// Print received data
printf("%d %d %d\n", rxbuf[0], rxbuf[1], rxbuf[2]);

// Sleep for 0.5 seconds
usleep(500000);
}

// Never reached, but good practice
mraa_spi_stop(spi);
mraa_deinit();

return 0;
}
```

compile :

. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi

${CC} mraa_spi.c -o mraa_spi -lmraa

```
root@rugged-board-a5d2x-sd1:/home# ./mraa_spi
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
```

Python:

Vi spi.py

```
import mraa
import time

# Initialize SPI on bus 0
spi = mraa.Spi(0)

# Prepare data to send (3 bytes)
txbuf = bytearray([100, 123, 75])

while True:
    # Send and receive data over SPI
    rxbuf = spi.write(txbuf)

    # Print received bytes
    print(rxbuf[0], end=" ")
    print(rxbuf[1], end=" ")
    print(rxbuf[2])

    # Wait for 0.5 seconds
    time.sleep(0.5)
```

In Board:

```
root@rugged-board-a5d2x-sd1:/home# python3 spi.py
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
100 123 75
```