Embedded Linux BSP U-Boot Porting

Organised & Supported by **RuggedBOARD**

Agenda

- U-BOOT Architecture
- U-BOOT Code Flow
- U-BOOT Porting on new Hardware
- U-BOOT Compilation & Flashing on RB-A5D2x (P)
- U-BOOT Commands (P)
- Adding new commands in U-BOOT (P)
- Custom Driver in U-BOOT (P)

Embedded Systems Classification

S1.0

MCU Based
Very Low Power
Small Code (KB’s)
Baremetal
Small RTOS

S2.0

MPU Based
High Speed (200MHz till 1GHz)
OS + Application Code

S3.0

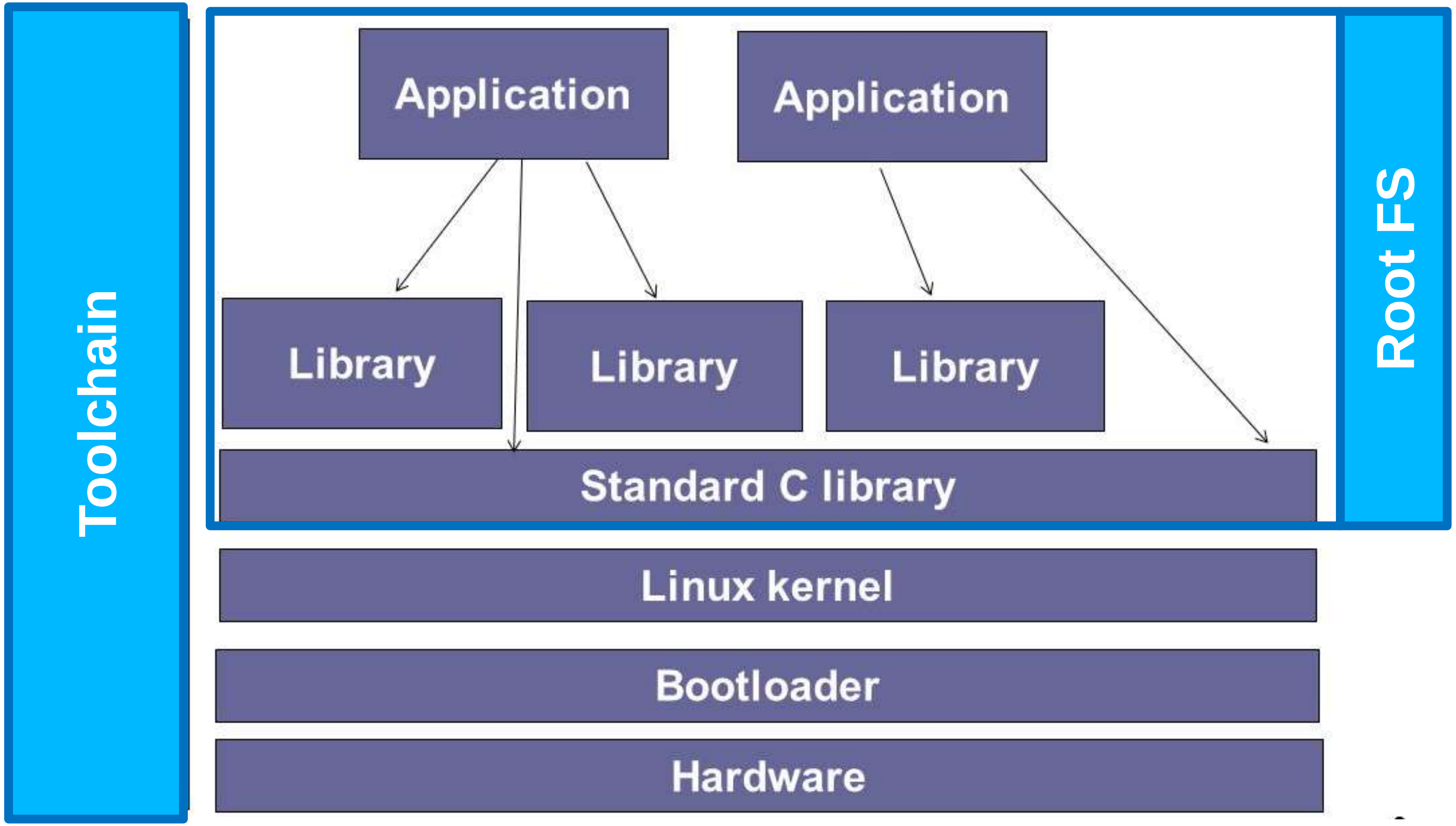
MPU+ Based
Special Co-Processors
Very High Computation Power
Special Hardware Accelerator Engines like TPU, VPU, GPU’s

Mostly uses Cortex-M4 having BLE comm and few sensors need companion mobile

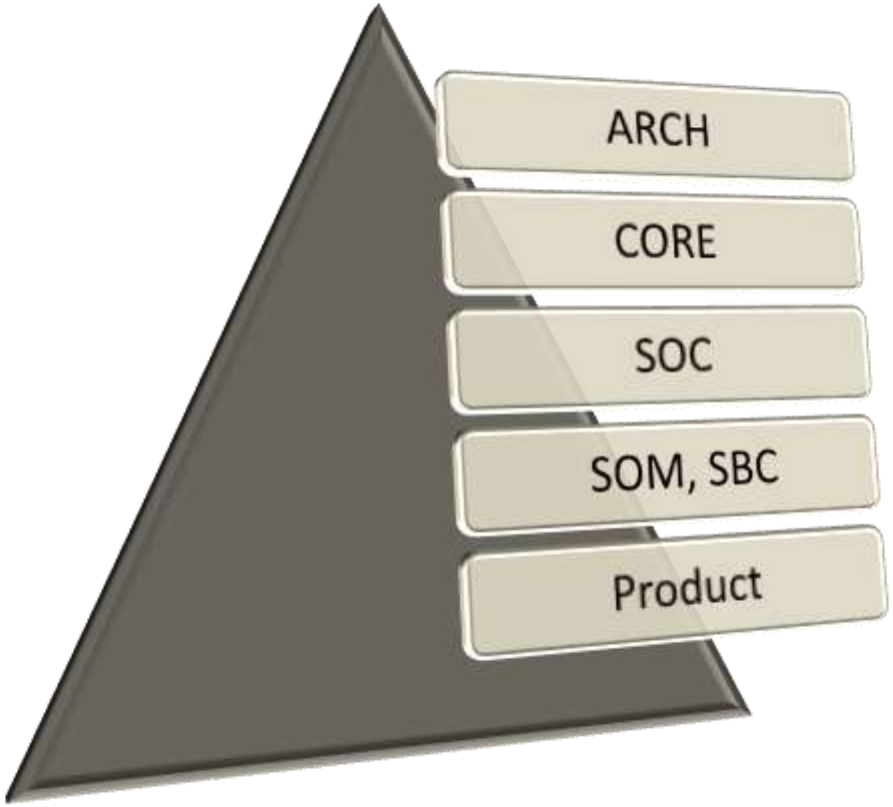
Mostly uses Cortex-A7, 4G comm friendly UI, make calls, check emails etc …

Mostly uses Cortex-53, 4G and advance AI/ML capabilities to process the data on-device and generate analytics & feedback

eLinux System ARCH



System Hardware



ARCH

CORE

SOC

SOM, SBC

Product

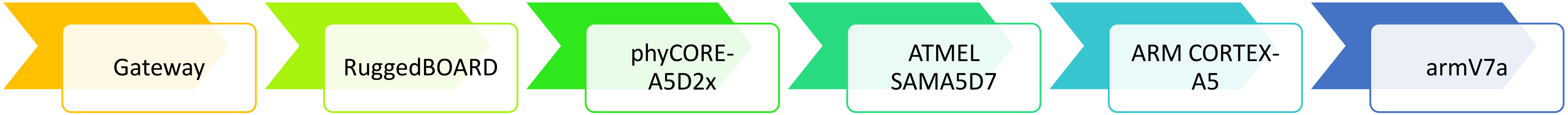
Processor Blueprint, defines IS & other hardware blocks of Processor

Processor design in VHDL / Verilog having ALU, Registors, TCU, Buses …

Silicon with Processor & peripherals like GPIO, UART, I2C, SPI, USB, Ethernet …

SOM = SOC+ RAM + Flash + PMIC, **SBC** = Board with SOM & interfacing devices like LCD, Connectors, Sensors & Communication modules

Product = SBC + Software + Housing/Mechanicals



Gateway

RuggedBOARD

phyCORE-A5D2x

ATMEL SAMA5D7

ARM CORTEX-A5

armV7a

RuggedBOARD

Rugged Board

- A5D2x @500MHz
Cortex - A5
64MB RAM
32MB Flash
- 2 x RS232
- 1x RS485
- 1 x CAN
- 1 x Ethernet
- TFT & Cap Touch
- 1 x MicroSD Slot
- 2 x USB
- DC & USB Power
- Expansion Header
- mikroBUS Conn.
- mPCIe conn.
- Micro SIM Slot

open source hardware

open source initiative®

Industrial Grade Hardware for IIoT

https://Community.ruggedboard.com

Boot Process

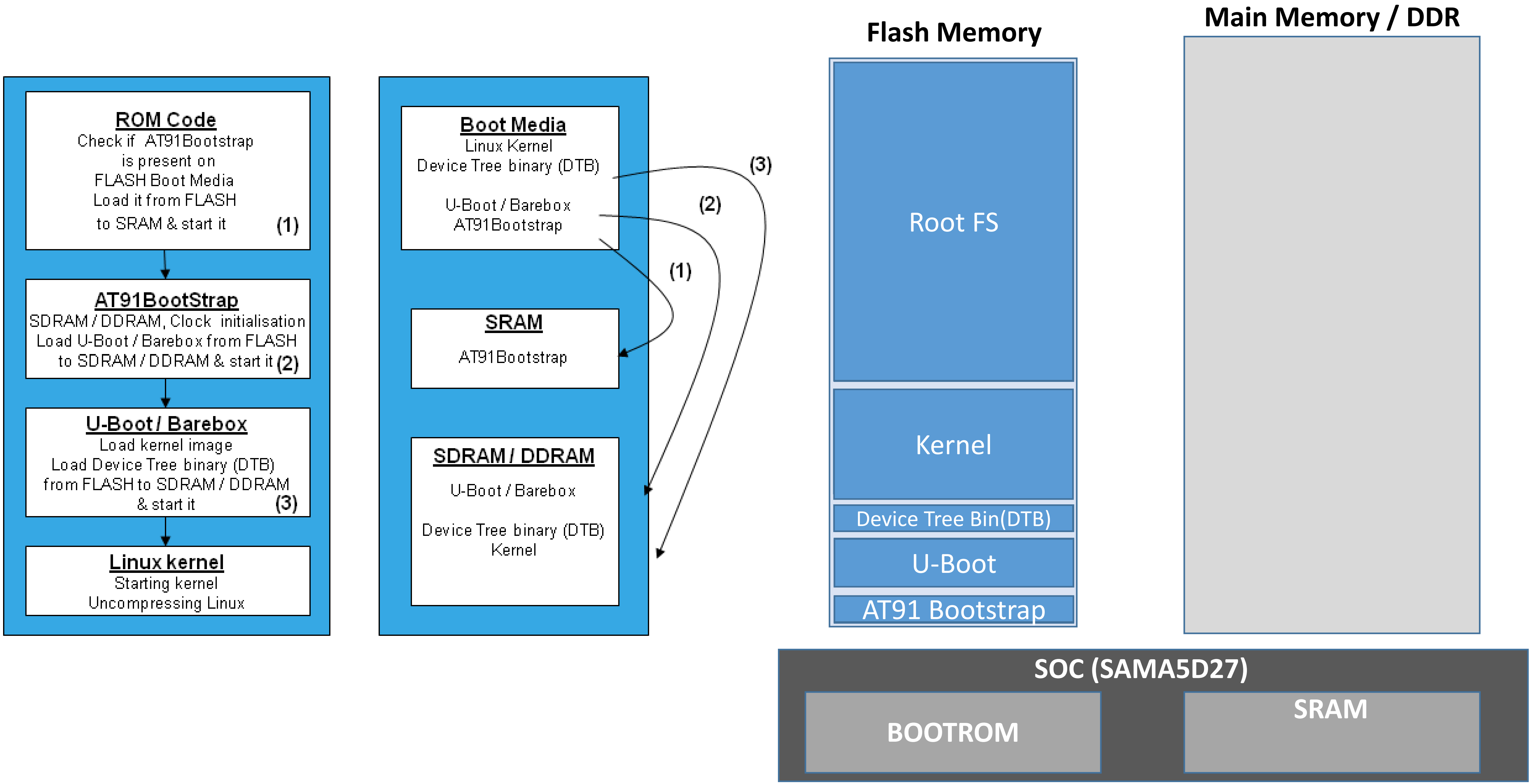
ON PC:

Power On-> BIOS (POST, Bootstraploader) -> MBR -> Bootloader -> Kernel -> RFS

ON RuggedBOARD:

1. Power On SBC
2. SOC BootROM Code will exec
3. BootCFG Pins will define the bootdevice (NAND, NOR, SDCARD .....)
4. From Bootdevice first piece of code (PBL) loaded in SRAM and executed
5. PBL responsible for External RAM Init and loads the BL to External RAM and execute.
6. BL will load the kernel and executes
7. Kernel boots and mounts the RootFS and finally executes the init binary
8. Init will follow init rc scripts to start services and applications

Boot Process



ROM Code
Check if AT91Bootstrap
is present on
FLASH Boot Media
Load it from FLASH
to SRAM & start it (1)

AT91BootStrap
SDRAM / DDRAM, Clock initialisation
Load U-Boot / Barebox from FLASH
to SDRAM / DDRAM & start it (2)

U-Boot / Barebox
Load kernel image
Load Device Tree binary (DTB)
from FLASH to SDRAM / DDRAM
& start it (3)

Linux kernel
Starting kernel
Uncompressing Linux

Boot Media
Linux Kernel
Device Tree binary (DTB)

U-Boot / Barebox
AT91Bootstrap

SRAM

AT91Bootstrap

SDRAM / DDRAM

U-Boot / Barebox

Device Tree binary (DTB)
Kernel

U-BOOT Source

Browse Source: https://github.com/rugged-board/uboot-rba5d2x

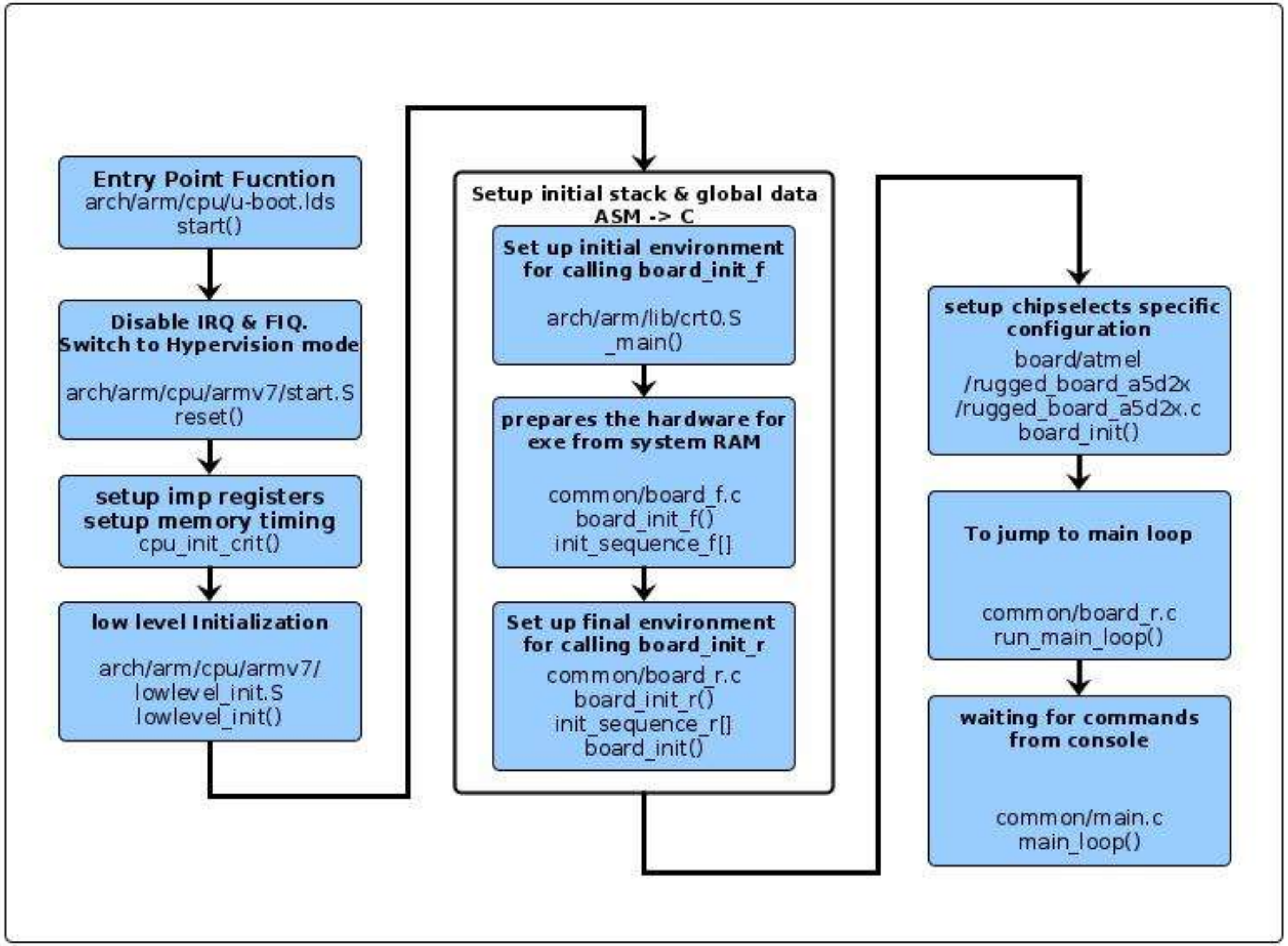
Download U-Boot for RuggedBOARD

```
$ wget https://github.com/rugged-board/uboot-rba5d2x/archive/uboot-rba5d2x.zip
Or
$ git clone https://github.com/rugged-board/uboot-rba5d2x.git
```

U-BOOT Dir Structure

uboot/arch/arm/cpu	Arch & Core specific code, u-boot.lds armv7/start.S, cpu.c
uboot/arch/arm/mach-at91	SOC specific code, armv7/sama5d2_devices.c
uboot/arch/arm/dts	device tree directory consists of device tree files for SOC, SOM, SBC *sama5d2.dsi , rb_a5d2x.dtsi, rugged_board_a5d2x.dts*
uboot/board/atmel/rugged_board_a5d2x	Board directory contains board files with syntax <vendor>/<boardname> boardname.c called board file.
uboot/configs	Contains board default configuration file used to configure uboot for a specific board. <boardname_defconfig> for ruggedboard we have two files for NOR: rugged_board_a5d2x_qspiflash_defconfig & for SDCARD: rugged_board_a5d2x_mmc1_defconfig
uboot/drivers	Contains bus drivers & device drivers (gpio, serial, i2c, spi, mmc, usb, net) at91_gpio.c, atmel_usart.c, at91_i2c.c, atmel_sdhci.c, atmel_spi/qspi.c, at91_emac.c Device Driver: rtc/ds1307.c, misc/i2c_eeprom.c …

U-BOOT Code Flow



Entry Point Fucntion
arch/arm/cpu/u-boot.lds
start()

Disable IRQ & FIQ.
Switch to Hypervision mode
arch/arm/cpu/armv7/start.S
reset()

setup imp registers
setup memory timing
cpu_init_crit()

low level Initialization
arch/arm/cpu/armv7/
lowlevel_init.S
lowlevel_init()

Setup initial stack & global data
ASM -> C

Set up initial environment for calling board_init_f
arch/arm/lib/crt0.S
_main()

prepares the hardware for exe from system RAM
common/board_f.c
board_init_f()
init_sequence_f[]

Set up final environment for calling board_init_r
common/board_r.c
board_init_r()
init_sequence_r[]
board_init()

setup chipselects specific configuration
board/atmel
/rugged_board_a5d2x
/rugged_board_a5d2x.c
board_init()

To jump to main loop
common/board_r.c
run_main_loop()

waiting for commands from console
common/main.c
main_loop()

U-BOOT Code Flow

```
board_init_f()
        - initf_bootstage          /* uses its own timer,so doesn’t need DM */
        - arch_cpu_init            /* basic arch cpu dependent setup */
        - mach_cpu_init            /* SoC/machine dependent CPU setup */
        - get_clocks               /* get CPU and bus clocks (etc.) */
        - timer_init               /* initialize timer */
        - env_init                 /* initialize environment */
        - init_baud_rate           /* initialze baudrate settings */
        - serial_init              /* serial communications setup */
        - console_init_f           /* stage 1 init of console */
        - dram_init                /* configure available RAM banks */
```

U-BOOT Code Flow

```
board_init_r()
        - board_init                    /* Setup chipselects */
        - set_cpu_clk_info              /* Setup clock information */
        - initr_nand                    /* initialize flash */
        - initr_mmc                     /* initialize fmmc */
        - console_init_r                /* fully init console as a device */
        - arch_misc_init                /* miscellaneous arch-dependent init */
        - misc_init_r                   /* misc platform-dependent init */
        - interrupt_init                /*set up exceptions */
        - initr_enable_interrupts       /* enable exceptions */
        - initr_ethaddr                 /* setup ethernet */
        - board_late_init               /* board late initialization */
        - run_main_loop                 /* jump to main loop & waiting for commands from console */
```

U-BOOT Porting

1. Identify the ARCH, CORE & SOC used in your board
2. Check the ARCH & Core support in u-boot location /arch/arm/cpu
3. Check the SOC support location uboot/arch/arm/mach-<soc_family>
4. Create new board folder in u-boot/boards/<board_name>
5. Take ref of existing boards in uboot and develop the code for your board

   Add board.c, modify Kconfig & Makefile
6. Create a default configuration file for your board in u-boot/configs
7. Driver level modification if required u-boot/drivers/
8. Make sure you did modified Makfiles corresponding to your code/file changes.

U-boot Compilation

Browse Source: https://github.com/rugged-board/uboot-rba5d2x

Compiling U-Boot for RuggedBOARD

```
#Set the toolchain path first
$ . env_setup.sh

# Download uboot Source
$ git clone https://github.com/rugged-board/uboot-rba5d2x.git
$ cd uboot-rba5d2x
$ git checkout origin/uboot-rba5d2x

# Configure u-boot bootloader for RB-A5D2x
$ make rugged_board_a5d2x_mmc1_defconfig            # For SD Card
Or
$ make rugged_board_a5d2x_qspiflash_defconfig       # For NOR Boot

# Compile u-boot bootloader
make
```

U-boot compiling using Yocto

#Configure for RuggedBOARD-A5D2x

```
$ source sources/poky/oe-init-build-env
$ vi conf/local.conf
# Edit   MACHINE ?= "rugged-board-a5d2x-sd1“
```

#Compile

```
$ bitbake u-boot
```

#Images for NOR

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

#Follow NOR Flashing Tutorial..

U-boot Flashing on RB-A5D2x (SDCARD)

```
# Power on board and stop at bootlaoder prompt

#check mmc card info
u-boot$ mmcinfo

# init serial flash
u-boot$ sf probe

#copy uboot image from mmc to RAM
u-boot$ fatload mmc 1 0x21FF0000 NOR/u-boot.bin

 #erase serial flash(NOR) u-boot partition
u-boot$ sf erase 0x20000 0x80000

# copy from uboot image from RAM to NOR Flash
u-boot$ sf write 0x21FF0000 0x20000 0x80000.
```

U-boot Flashing on RB-A5D2x (TFTP)

```
# Power on board and stop at bootlaoder prompt

#check network connection by pining host PC
u-boot$ ping <serverip>

# Download uboot image from PC to Board RAM
u-boot$ tftp 0x21FF0000 u-boot.bin

 #erase serial flash(NOR) u-boot partition
u-boot$ sf erase 0x20000 0x80000

# copy from uboot image from RAM to NOR Flash
u-boot$ sf write 0x21FF0000 0x20000 0x80000
```

U-boot Flashing on RB-A5D2x (Serial)

# Power on board in serial download mode by pressing the boot switch

U-Boot Commands

Information Commands	
help	*print online help*
bdinfo	*print Board Info structure*
coninfo	*print console devices and information*
flinfo	*print FLASH memory information*

Environment Variables Commands	
env	*environment handling commands*
printenv	*print environment variables*
setenv	*set environment variables*
editenv	*edit environment variable*
saveenv	*save environment variables to persistent storage*

Basic Commands	
version	*print monitor version*
echo	*echo arguments to console*
reset	*perform RESET of the CPU*
sleep	*delay execution for some time*
cls	*Clear screen*

Memory Commands	
mtest	*simple RAM test*
md	*echo arguments to console*
mm	*memory modify (auto incrementing)*
mw	*memory write (fill)*
nm	*memory modify (constant address)*
base	*print or set address offset*
crc32	*checksum calculation*
cp	*memory copy*

U-Boot Commands

Download & BOOT Commands	
loadb	*load binary file over serial line (kermit mode)*
loady	*load binary file over serial line (ymodem mode)*
loads	*load S-Record file over serial line*
Ping	*send ICMP ECHO REQUEST to network host*
bootp	*boot image via network using BOOTP/TFTP protocol*
dhcp	*invoke DHCP client to obtain IP/boot params*
tftpboot	*boot image via network using TFTP protocol*
nfs	*boot image via network using NFS protocol*
boot	*boot default, i.e., run 'bootcmd'*
bootm	boot application image from memory
Nboot	boot from NAND device
go	start application at address 'addr'
fatload	load binary file from a FAT file system
Ext2load	load binary file from a Ext2 filesystem

HW Subsytem	
gpio	manipulate gpios
i2c	I2C sub-system control
mmc	MMC sub system
usb	USB sub-system control
ftd	flattened device tree utility commands
mtdparts	define flash/nand partitions
eeprom	EEPROM sub-system control
nand	NAND sub-system control
flinfo	print FLASH memory information
erase	erase FLASH memory
sf	*Serial Flash sub-system*

Adding new command in U-Boot

U_BOOT_CMD() is the Macro used to add new command in u-boot.

```
U_BOOT_CMD(name, maxargs, repeatable, command, "usage", "help")
```

name:	is the name of the command. THIS IS NOT a string.
maxargs:	the maximum numbers of arguments this function takes
command:	Command implementation Function pointer (*cmd)(struct cmd_tbl_s *, int, int, char *[]);
Usage:	Short description. This is a string
help:	long description. This is a string

Command Function Prototype:

```
int do_funcname  (cmd_tbl_t *cmdtp, int flag, int argc, char *const  Argv[] )
```

cmdtp – Command table pointer (function vector table)

flag -- Unused

argc -- Argument count, including command name itself

argv[] -- Array of arguments (string).

Adding new command in U-Boot

Step-1 : create demo.c in u-boot/command folder

```
$ cd <uboot_path>/command
$ vim dummy.c
```

```
#include<common.h>
#include<command.h>

static int do_dummy(cmd_tbl_t *cmdtp, int flag, int argc,
char * const argv[])
{
        printf("Hello Rugged Board A5d2x\n");
        printf("This is dummy command implementation\n");
        return 0;
}

U_BOOT_CMD(dummy, 2, 1, do_dummy, "testing
hello","arg1 not needed");
```

Step-2: Modify Kconfig file under command folder

```
$vim Kconfig
```

```
config CMD_DUMMY
        bool “Dummy Command"
        default y
        help
          This is testing the new command in rugged board..
```

Step-3: Modify Makefile

```
$vim Makefile          //  bootloader/uboot-rba5d2x/cmd
```

```
obj-$(CONFIG_CMD_DUMMY) += dummy.o
```

Step-4: Compile & Flash

Step-5: Test the command on Target Bootloader prompt

```
=> hello
             Hello Rugged Board A5d2x
              This is dummy command implementation
```

Adding new Driver in U-Boot

#Step-1: Define your device in dts file

```
$ vim <uboot_path>/arch/arm/dts/rugged_board_a5d2x.dts

leds {
        compatible = "sled-testing";
        status = "okay";

        UserLed {
            label = "UserLed";
                        sled-default-state = "blink";
        };
    };
```

# Step-2: Define your driver sled.c in uboot/driver folder

```
$ vim <uboot_path>/driver/led/sled.c
# copy the sled.c code
```

# Step-3: Add sled configuration in Kconfig file

```
$ vim <uboot_path>/driver/led/Kconfig
```

```
config SLED
        bool "SLED support for LEDs"
        depends on LED
        help
          Sled driver on RuggedBOARD-A5D2x
```

# Step-4: Add sled configuration in Kconfig file

```
$ vim <uboot_path>/driver/led/Makefile
```

```
obj-$(CONFIG_SLED) += sled.o
```

# Step-5: Write a test code cmd_sled.c under command folder and which calls the driver functions

```
$ vim <uboot_path>/command/cmd_sled.c
#implement  do_sled() & register using U_BOOT_CMD
```

Driver Code Flow

File	Description
u-boot/driver/gpio/at91_gpio.c	Atmel GPIO Driver core bus driver
u-boot/driver/gpio/gpio-uclass.c	U-Boot GPIO Subsystem HAL
u-boot/driver/led/sled.c	Sled device driver which used gpio bus driver
u-boot/command/sled_cmd.c	Test app / command implemented to test sled driver

Demo’s

Next Sessions

1. Linux Kernel Porting using RuggedBOARD-A5D2x
   a. Source Code walkthrough & Code flow
   b. Kconfig Kernel Configuration System
   c. Adding Custom driver
2. Yocto BSP using RuggedBOARD-A5D2x
3. Design your own Single Board Computer using phyCORE-A5D2x. [HW Design]
4. Building Gateway Hardware and Open Source Linux Stacks for Gateway. [HW Design]

To get update's follow RuggedBOARD on LinkedIn, Youtube, Twitter, Facebook & Instagram links are on next slide …

Connect-us

in

YouTube

BLOG

FORUM

Developer Wiki

Follow us on twitter

Open Discussions

Presenter Profile

B Vasu Dev

Managing Director

PHYTEC Embedded Pvt Ltd

vasu.b@phytec.in

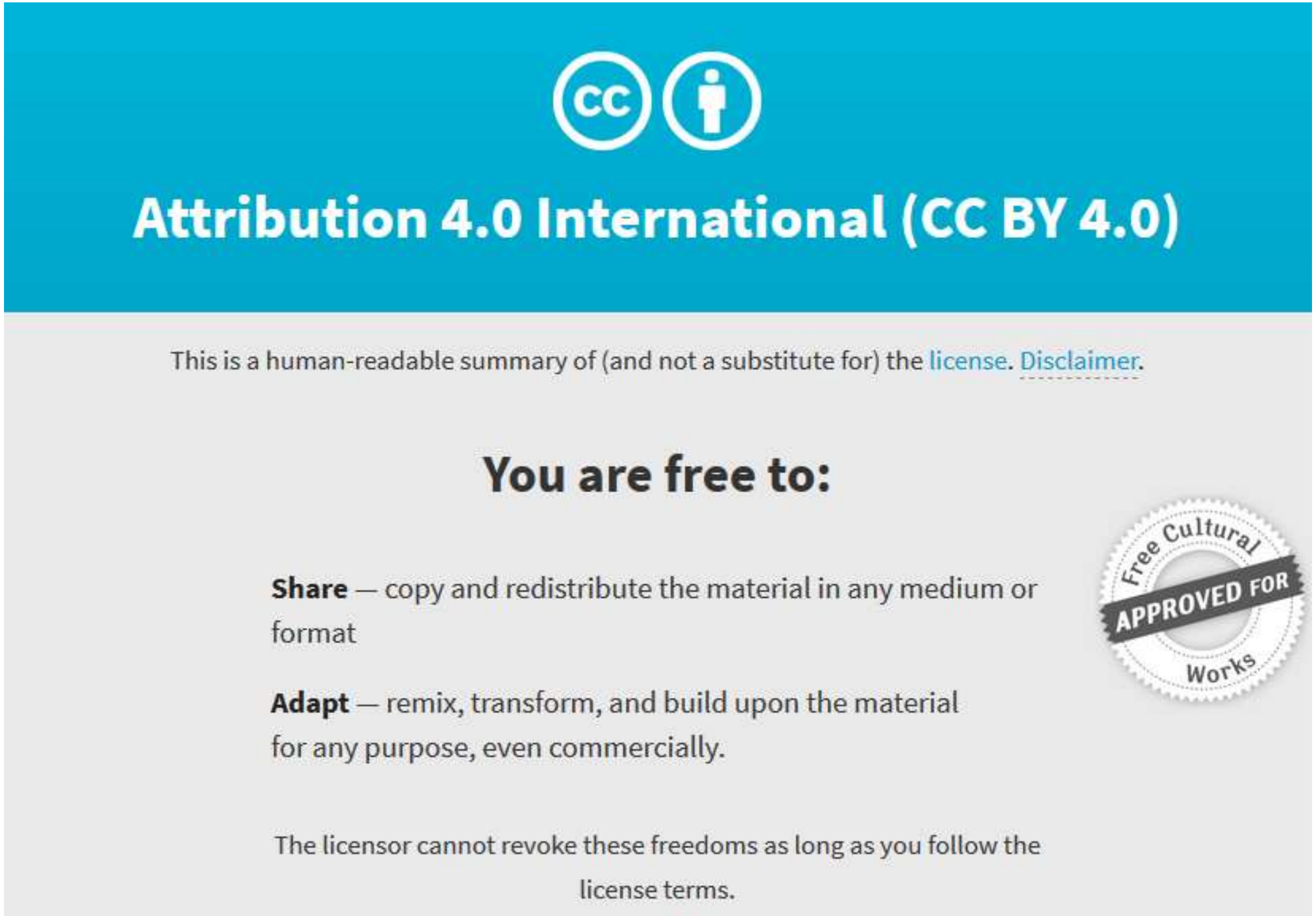
+91-9535504414

view my profile on Linked in.

ABOUT Vasu

Vasu has 15+ Years of industry experience in Embedded Technologies mainly on ARM & Linux, he has worked at major MNC's like LG, Wipro, MIC Electronics and is currently heading PHYTEC INDA, a subsidiary of PHYTEC Messtechnik GmbH GERMANY as Managing Director. PHYTEC serves as OEM for many electronic and embedded companies to develop and deploy their products at the lowest possible time with high reliability and quality using ARM based SOMs (System On Modules) & SBCs (Single Board Computers). The industry verticals he was engaged are Industrial Automation, Mobility & Energy, Medical/Healthcare, Retail market.

Apart from his technical work, he is an active coach & guide for Embedded developers and actively spend his time to train the developers on Embedded Linux, Yocto, IoT, Android System Development. He is the master mind behind RuggedBOARD Open Source Hardware Platform. Vasu as a mentor helped many start-ups to build their products and position them in market.



Attribution 4.0 International (CC BY 4.0)

This is a human-readable summary of (and not a substitute for) the license. Disclaimer.

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.