#LEARN IN DEPTH

#Be professional in embedded system

Embedded Linux (PART 8)

- LAB6: BUILD/RUN UBOOT FOR RASPBERRY PI2
- LAB7: BUILD/RUN UBOOT FOR BEAGLEBONE BLACK
- LAB8: DEBUG THE U-BOOT CODE
- PORT U-BOOT TO A NEW BOARD (BASICS)
- PROJECT "SIMPLE CONCEPT IMPLEMENTATION FOR OTA SW"

ENG.KEROLES SHENOUDA

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

You have to review Part 7 first

LEARN IN DEPTH ☺

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Lab6: Build/Run Uboot for Raspberry Pi2

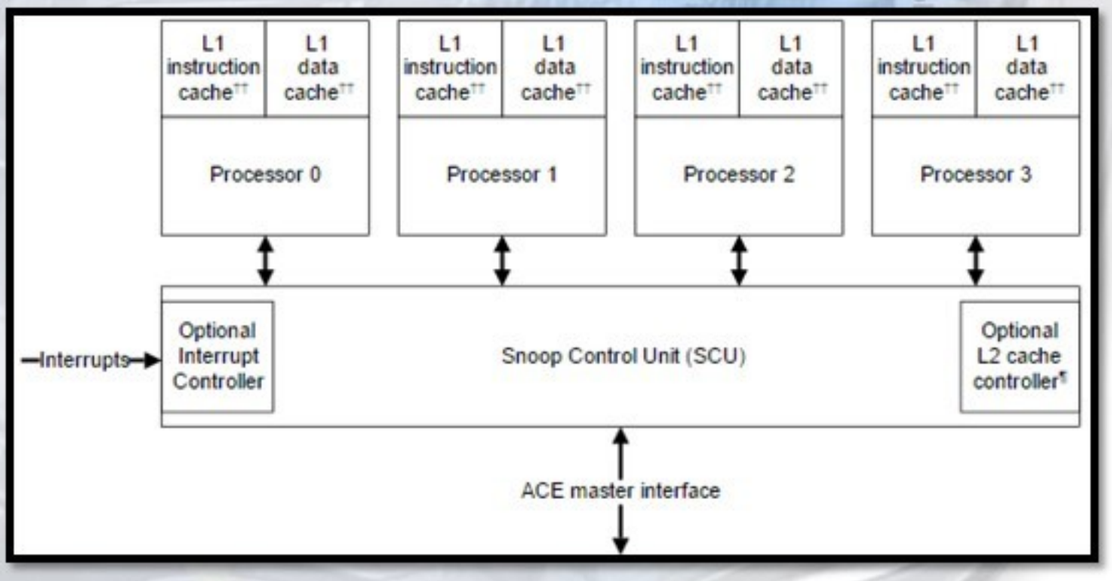
Raspberry Pi 2

- Released in Feb 2015
- SoC – Broadcom BCM2836 quad core Cortex A7 processor @ 900MHz with VideoCore IV GPU
- System Memory – 1GB LPDDR2 (PoP)
- Storage – micro SD card slot (push release type)
- Video & Audio Output – HDMI and AV via 3.5mm jack.
- Connectivity – 10/100M Ethernet
- USB – 4x USB 2.0 ports, 1x micro USB for power
- Expansion
  - 2×20 pin header for GPIOs
  - Camera header
  - Display header
- Power – 5V via micro USB port.
- Dimensions – 85 x 56 mm

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

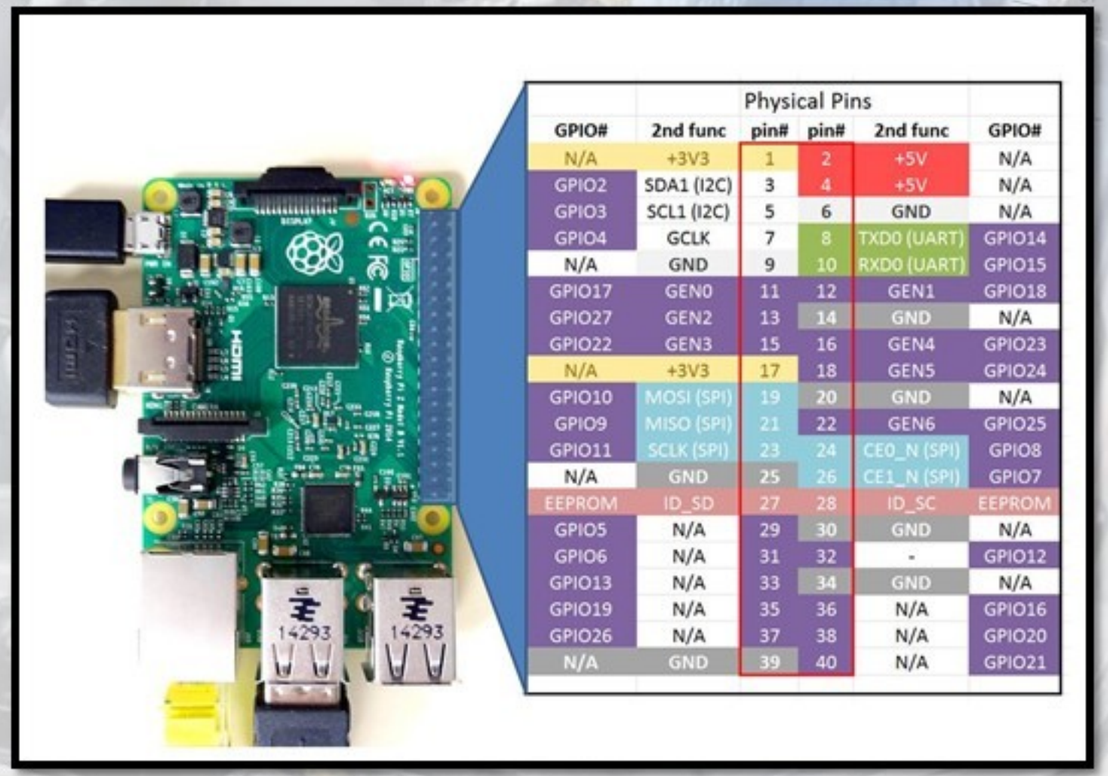
System on Chip (SoC). BCM2836

- The Raspberry Pi 2 main chip is the Broadcom BCM2836 System on a Chip (SoC); it is running at 900Mhz, based on the quad-core ARM Cortex-A7
- The ARM Cortex-A7 processor
  - is a 32-bit processor core
  - implementing the ARMv7-A architecture.



RASPBERRY PI 2 PINOUT

- This diagram shows the pinout of the raspberry pi. You can see some pins are multipurpose. How the pins are referenced will depend on how you configure them in your software.



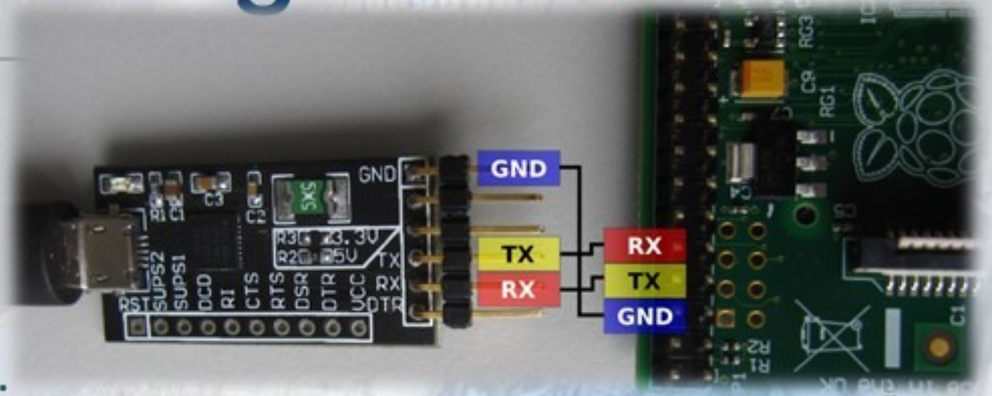
		Physical Pins			
GPIO#	2nd func	pin#	pin#	2nd func	GPIO#
N/A	+3V3	1	2	+5V	N/A
GPIO2	SDA1 (I2C)	3	4	+5V	N/A
GPIO3	SCL1 (I2C)	5	6	GND	N/A
GPIO4	GCLK	7	8	TXD0 (UART)	GPIO14
N/A	GND	9	10	RXD0 (UART)	GPIO15
GPIO17	GEN0	11	12	GEN1	GPIO18
GPIO27	GEN2	13	14	GND	N/A
GPIO22	GEN3	15	16	GEN4	GPIO23
N/A	+3V3	17	18	GEN5	GPIO24
GPIO10	MOSI (SPI)	19	20	GND	N/A
GPIO9	MISO (SPI)	21	22	GEN6	GPIO25
GPIO11	SCLK (SPI)	23	24	CE0_N (SPI)	GPIO8
N/A	GND	25	26	CE1_N (SPI)	GPIO7
EEPROM	ID_SD	27	28	ID_SC	EEPROM
GPIO5	N/A	29	30	GND	N/A
GPIO6	N/A	31	32	-	GPIO12
GPIO13	N/A	33	34	GND	N/A
GPIO19	N/A	35	36	N/A	GPIO16
GPIO26	N/A	37	38	N/A	GPIO20
N/A	GND	39	40	N/A	GPIO21

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Raspberry Pi2 Intrface through UART

1M CP2102 USB to UART TTL Cable Module Serial Adapter

- Use USB to TTL serial converters.



DAFLABS CP2102 USB 2.0 to TTL UART Serial Converter

OLatus MAX3232 RS232 to TTL Serial Port Converter

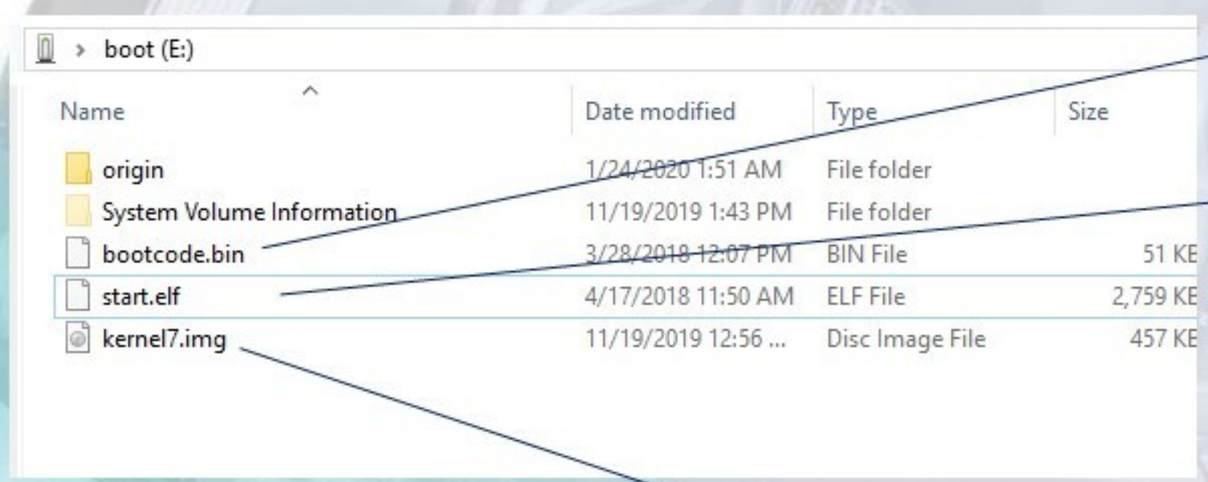
USB To RS232 PL2303 TTL Converter

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Raspberry Pi2 Booting Sequence

#LEARN IN DEPTH

#Be professional in embedded system



bootcode.bin

start.elf

Kernel7.img = Uboot.bin

GPU turns ON (CPU is OFF)

GPU executes ROMCode

ROMCODE reads the 1 stage bootlader from SDCARD, loads it to L2 cache and runs it

The 1st bootloader enable SDRAM loads the 2 stage bootloader to RAM and runs it

The 2st bootloader

Load the kernel image from SDCARD to RAM

Release the reset signal on ARM

ARM Booting the uboot

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

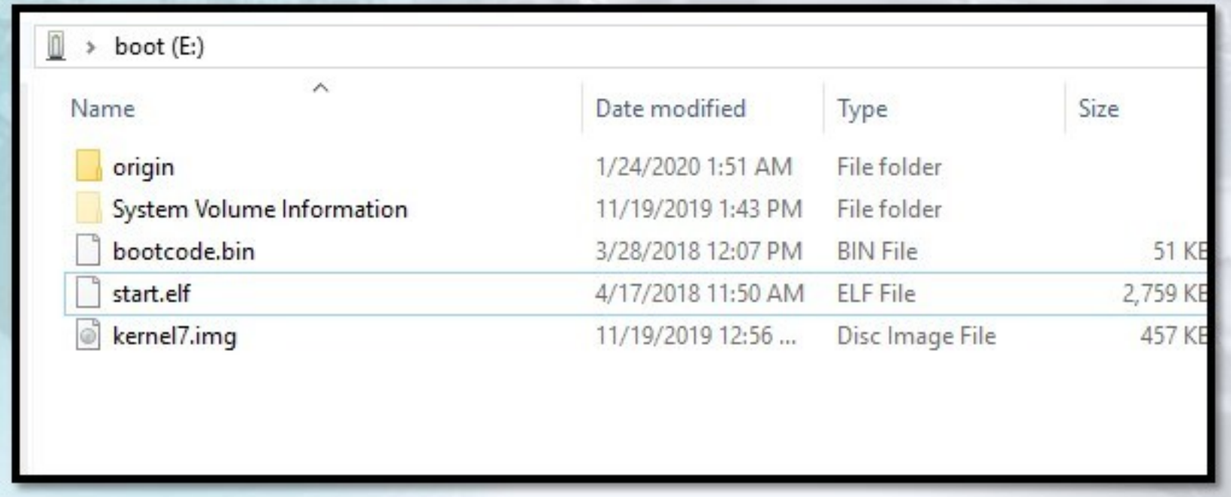
Build u-Boot fot RPI2

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ export ARCH=arm
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ export CROSS_COMPILE=arm-linux-gnueabi-
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make clean
  CLEAN   u-boot.cfg
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make rpi_2_config
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make -j 16
```

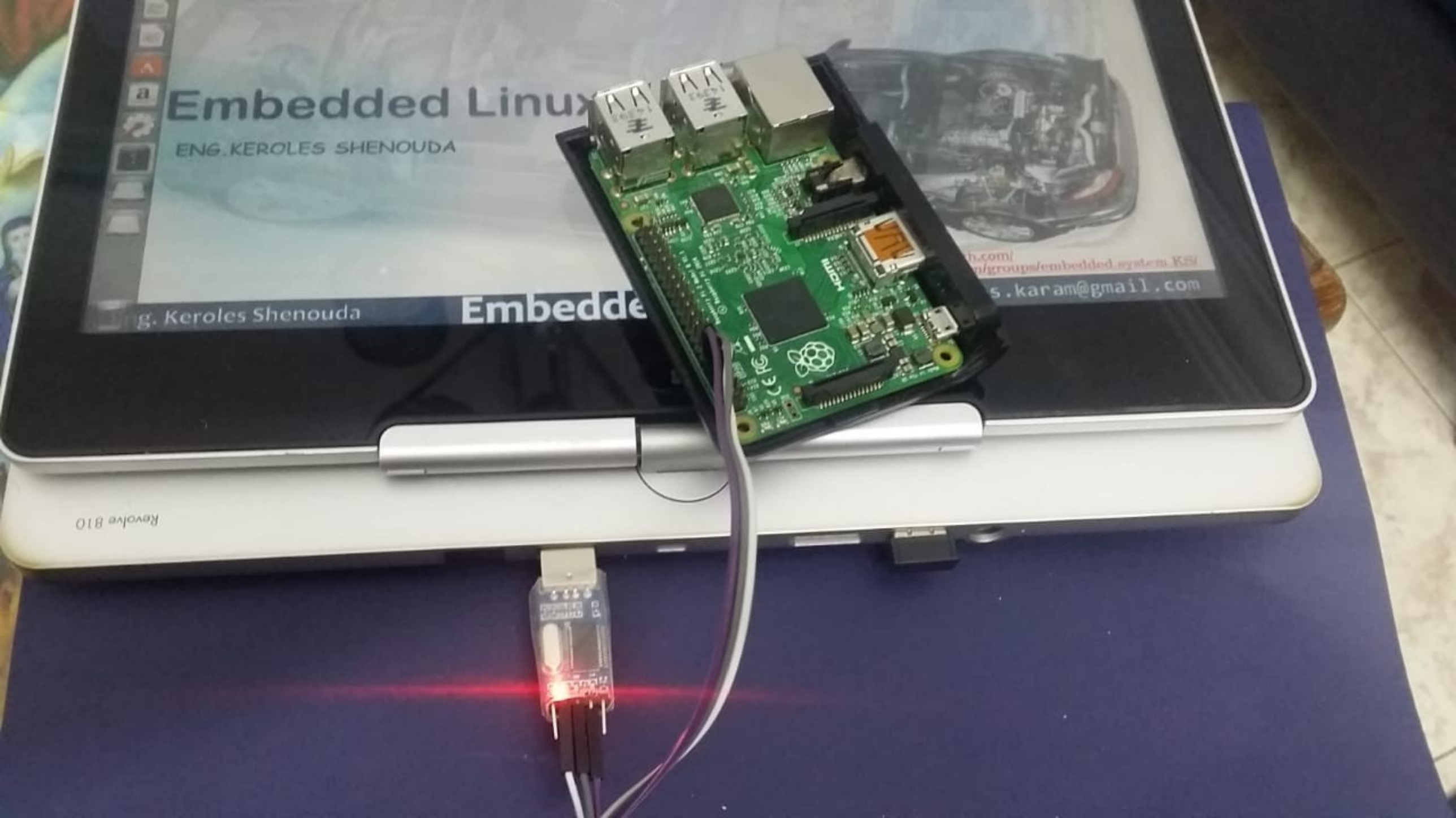

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Copy the images on SDCARD p1 FAT

- rename u-boot.bin to kernel7.img
- copy kernel7.img to P1
- Copy start.elf to P1
- Copy bootcode.bin to p1



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/


Embedded Linux
ENG.KEROLES SHENOUDA
g. Keroles Shenouda
Embedde
s.karam@gmail.com
Revolve 810

Run it by picocom in linux

```
keroles@karas:~$ sudo picocom /dev/ttyUSB0 -b 115200
picocom v1.7

port is        : /dev/ttyUSB0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready
MMC:   mmc@7e202000: 0
Loading Environment from FAT... *** Warning - bad CRC, using default environment

In:    serial
Out:   vidconsole
Err:   vidconsole
Net:   No ethernet found.
starting USB...
Bus usb@7e980000: scanning bus usb@7e980000 for devices... 3 USB Device(s) found
       scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot:  0
U-Boot>
U-Boot> fatls mmc 0:1
            System Volume Information/
            origin/
    52064   bootcode.bin
  2825124   start.elf
   467140   kernel7.img

3 file(s), 2 dir(s)

U-Boot>
```


Kernel7.img = Uboot.bin

bootcode.bin

start.elf

GPU turns ON (CPU is OFF)

GPU executes ROMCode

ROMCODE reads the 1 stage bootlader from SDCARD, loads it to L2 cache and runs it

The 1st bootloader enable SDRAM loads the 2 stage bootloader to RAM and runs it

The 2st bootloader

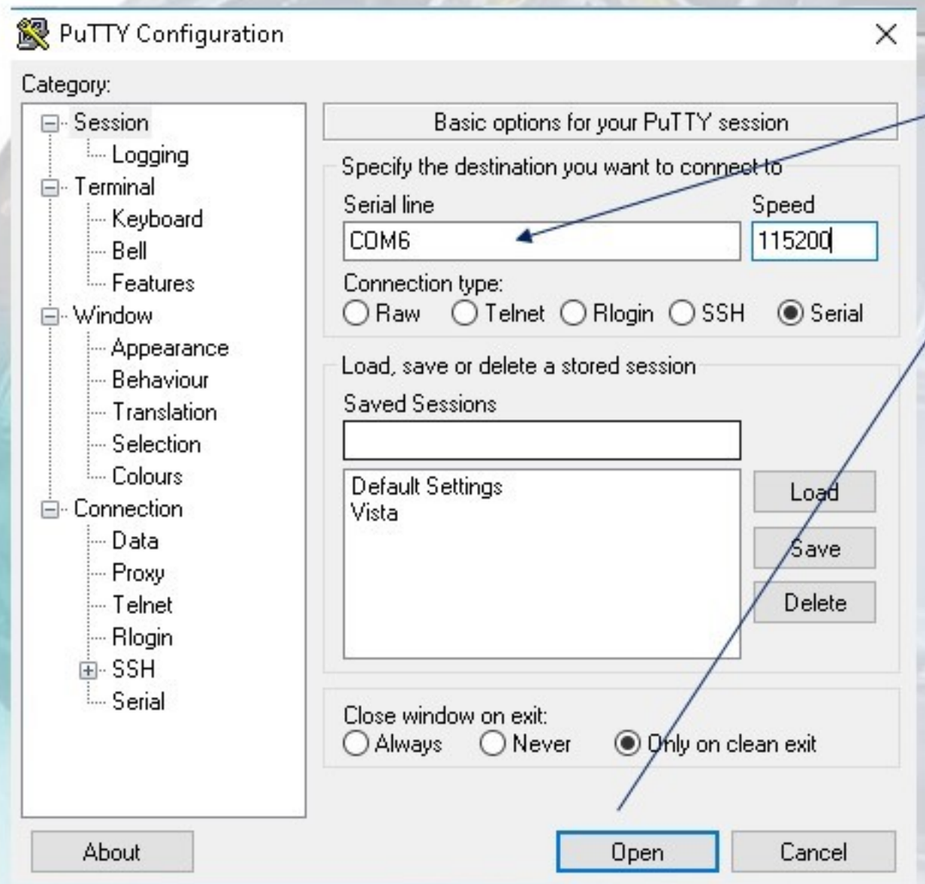
Load the kernel image from SDCARD to RAM

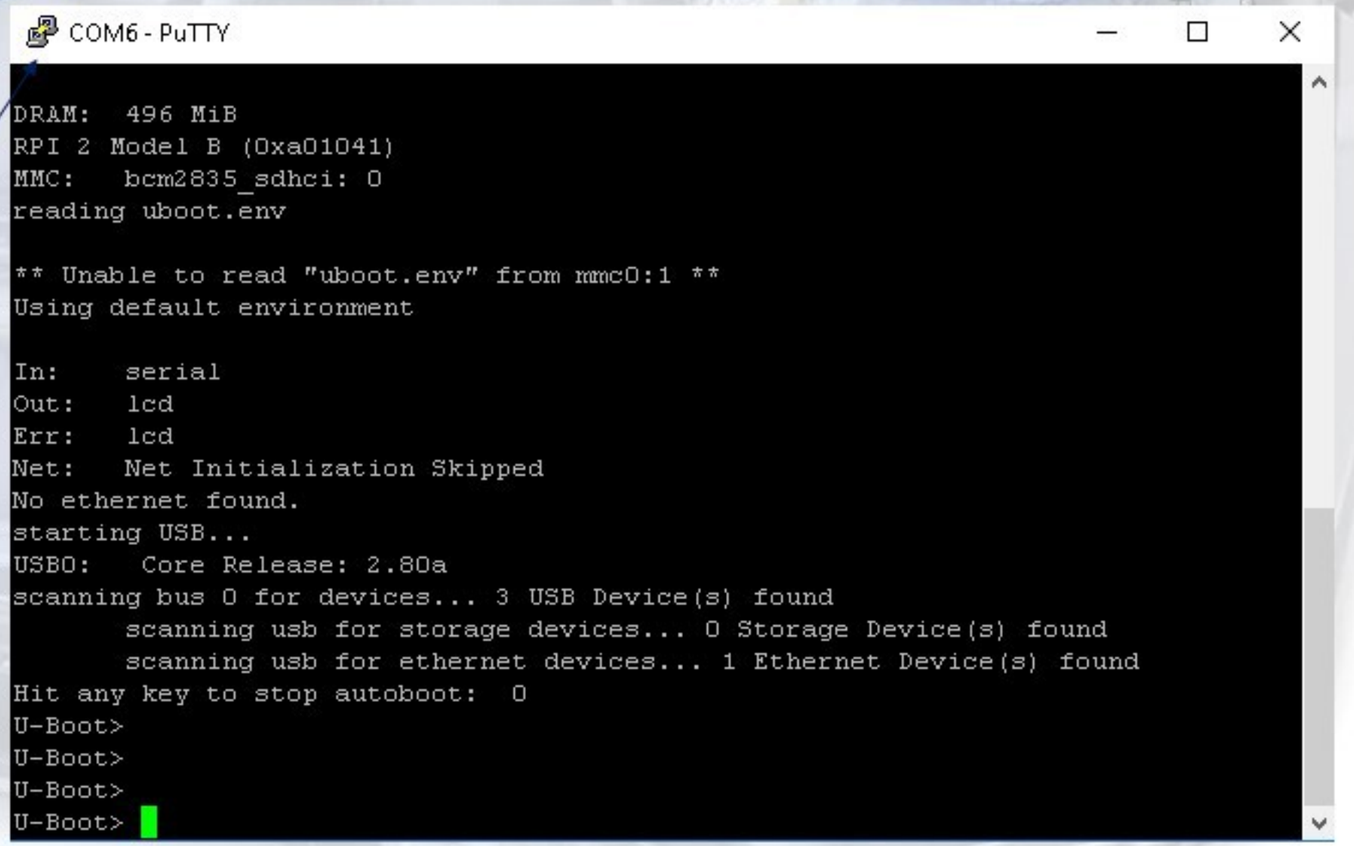
Release the reset signal on ARM

ARM Booting the uboot

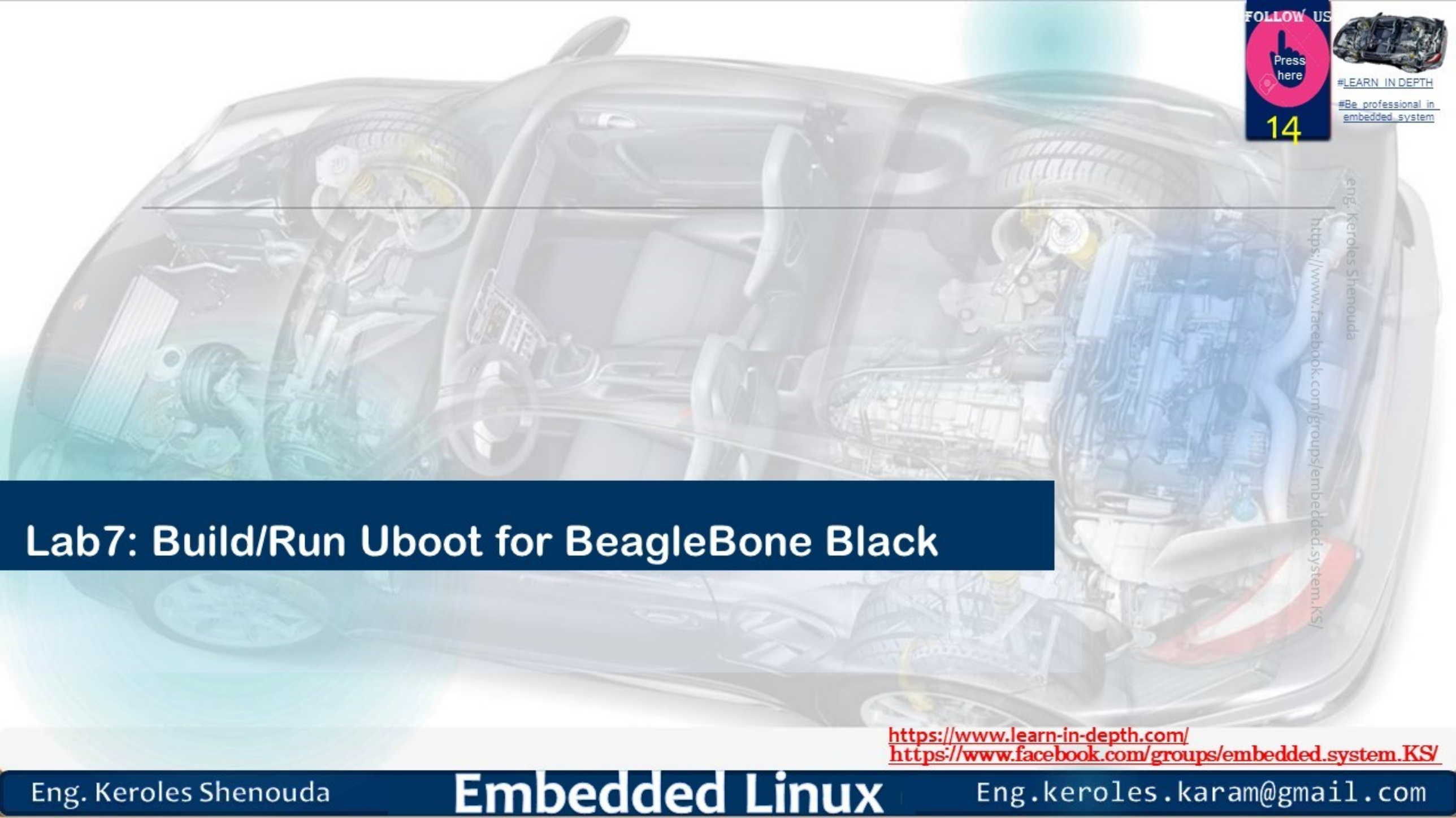
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Also you can use putty in windows





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

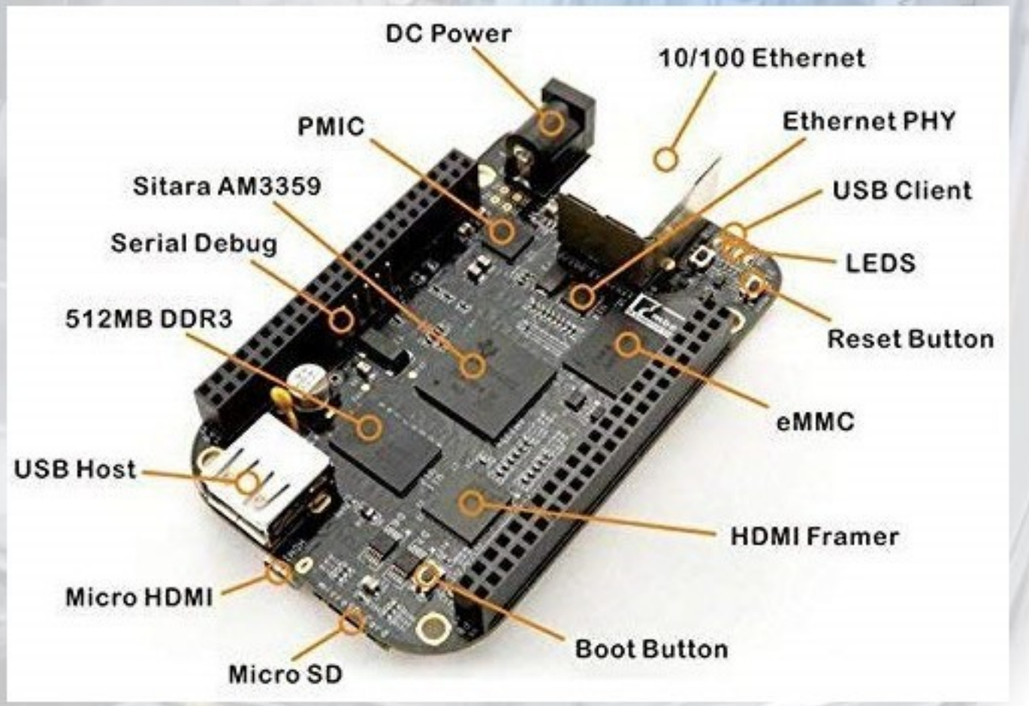


Lab7: Build/Run Uboot for BeagleBone Black

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

BeagleBone Black RevC

- Sitara AM3359AZCZ **ARM Cortex-A8** 32-Bit RISC microprocessor, up to 1GHz
- Onboard HDMI to connect directly to TVs and monitors
- More and faster memory now with 512MB DDR3L 400MHz SDRAM, 4GB eMMC Memory
- **Onboard** storage frees up the microSD card slot for greater expansion
- Support for existing Cape plug-in boards
- HS USB 2.0 Client and Host ports
- Serial port: UART0 access via 6-pin 3.3V TTL header
- 10/100 RJ45 Ethernet



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

AM3359 Sitara processor: Arm Cortex-A8

TEXAS INSTRUMENTS

Search

Products | Applications | Design resources | Quality & reliability | Support & training | Order now | About TI

TI Home > Semiconductors > Processors > Sitara processors > AM335x Arm Cortex-A8 >

AM3359 ACTIVE

In English | Alert me

Sitara processor: Arm Cortex-A8, EtherCAT, 3D, PRU-ICSS, CAN

DATASHEET

AM335x Sitara™ Processors datasheet (Rev. K)

View now | Download

USER GUIDES

AM335x and AMIC110 Sitara™ Processors Technical Reference Manual (Rev. Q)

ERRATA

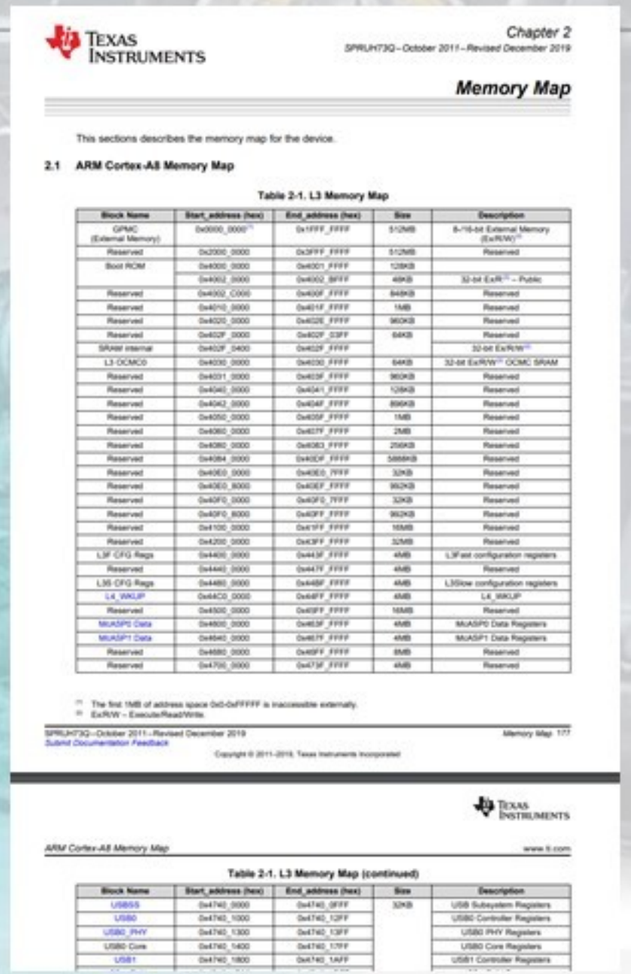
AM335x Sitara Processors Silicon Errata (Revs 2.1, 2.0, 1.0) (Rev. I)

Description & parametrics | Technical documentation | Design & development | Order now | Quality & packaging | Support & training

Description | Features | Diagram | Design resources for you | Parametrics

http://www.ti.com/product/AM3359

AM335X SoC



TEXAS INSTRUMENTS

Chapter 2
SPRUH73Q–October 2011–Revised December 2019

Memory Map

This sections describes the memory map for the device.

2.1 ARM Cortex-A8 Memory Map

Table 2-1. L3 Memory Map

Block Name	Start_address (hex)	End_address (hex)	Size	Description
GPMC (External Memory)	0x0000_0000[1]	0x1FFF_FFFF	512MB	8-/16-bit External Memory (Ex/R/W)[2]
Reserved	0x2000_0000	0x3FFF_FFFF	512MB	Reserved
Boot ROM	0x4000_0000	0x4001_FFFF	128KB	
	0x4002_0000	0x4002_BFFF	48KB	32-bit Ex/R[2] – Public
Reserved	0x4002_C000	0x400F_FFFF	848KB	Reserved
Reserved	0x4010_0000	0x401F_FFFF	1MB	Reserved
Reserved	0x4020_0000	0x402E_FFFF	960KB	Reserved
Reserved	0x402F_0000	0x402F_03FF	64KB	Reserved
SRAM internal	0x402F_0400	0x402F_FFFF		32-bit Ex/R/W[2]
L3 OCMC0	0x4030_0000	0x4030_FFFF	64KB	32-bit Ex/R/W[2] OCMC SRAM
Reserved	0x4031_0000	0x403F_FFFF	960KB	Reserved
Reserved	0x4040_0000	0x4041_FFFF	128KB	Reserved
Reserved	0x4042_0000	0x404F_FFFF	896KB	Reserved
Reserved	0x4050_0000	0x405F_FFFF	1MB	Reserved
Reserved	0x4060_0000	0x407F_FFFF	2MB	Reserved
Reserved	0x4080_0000	0x4083_FFFF	256KB	Reserved
Reserved	0x4084_0000	0x40DF_FFFF	5888KB	Reserved
Reserved	0x40E0_0000	0x40E0_7FFF	32KB	Reserved
Reserved	0x40E0_8000	0x40EF_FFFF	992KB	Reserved
Reserved	0x40F0_0000	0x40F0_7FFF	32KB	Reserved
Reserved	0x40F0_8000	0x40FF_FFFF	992KB	Reserved
Reserved	0x4100_0000	0x41FF_FFFF	16MB	Reserved
Reserved	0x4200_0000	0x43FF_FFFF	32MB	Reserved
L3F CFG Regs	0x4400_0000	0x443F_FFFF	4MB	L3Fast configuration registers
Reserved	0x4440_0000	0x447F_FFFF	4MB	Reserved
L3S CFG Regs	0x4480_0000	0x48BF_FFFF	4MB	L3Slow configuration registers
L4_WKUP	0x44C0_0000	0x44FF_FFFF	4MB	L4_WKUP
Reserved	0x4500_0000	0x45FF_FFFF	16MB	Reserved
McASP0 Data	0x4600_0000	0x463F_FFFF	4MB	McASP0 Data Registers
McASP1 Data	0x4640_0000	0x467F_FFFF	4MB	McASP1 Data Registers
Reserved	0x4680_0000	0x46FF_FFFF	8MB	Reserved
Reserved	0x4700_0000	0x473F_FFFF	4MB	Reserved

[1] The first 1MB of address space 0x0-0xFFFFF is inaccessible externally.
[2] Ex/R/W – Execute/Read/Write.

Table 2-1. L3 Memory Map (continued)

Block Name	Start_address (hex)	End_address (hex)	Size	Description
USBSS	0x4740_0000	0x4740_0FFF	32KB	USB Subsystem Registers
USB0	0x4740_1000	0x4740_12FF		USB0 Controller Registers
USB0_PHY	0x4740_1300	0x4740_13FF		USB0 PHY Registers
USB0 Core	0x4740_1400	0x4740_17FF		USB0 Core Registers
USB1	0x4740_1800	0x4740_1AFF		USB1 Controller Registers

1.4 Functional Block Diagram

Figure 1-1 shows the AM335x microprocessor functional block diagram.

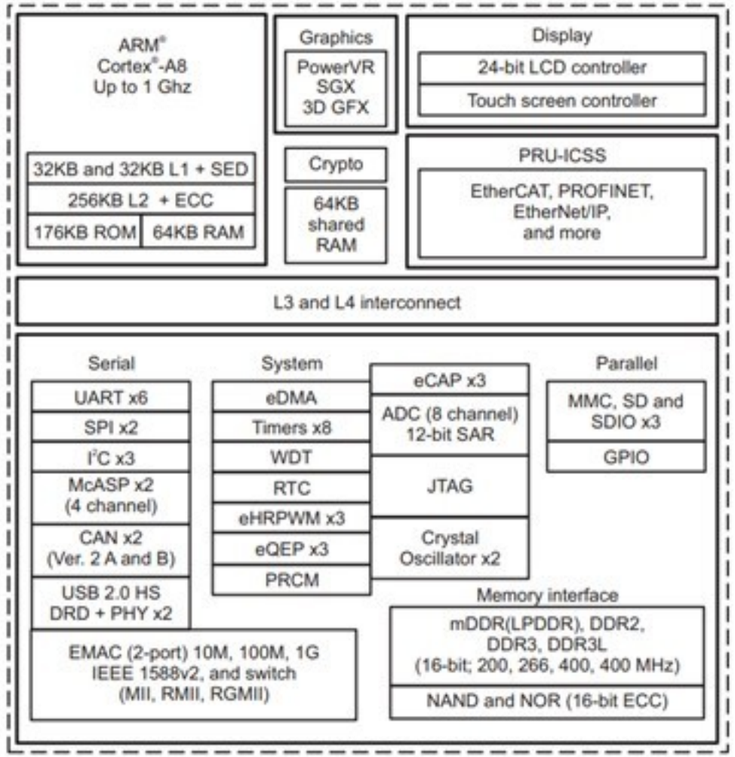


Figure 1-1. AM335x Functional Block Diagram

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Beagle Bone Black RevC INOUT

Beaglebone Black Pinout Diagram

P9

Function	Physical Pins		Function
DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V
VDD 5V	5	6	VDD 5V
SYS 5V	7	8	SYS 5V
PWR_BUT	9	10	SYS_RESET
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPIO_CSO	17	18	SPIO_D1
I2C2_SCL	19	20	I2C_SDA
SPIO_DO	21	22	SPIO_SLCK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SP11_CSO
SP11_DO	29	30	GPIO_112
SP11_SCLK	31	32	VDD_ADC
AIN4	33	34	GND_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPWMO
DGND	43	44	DGND
DGND	45	46	DGND

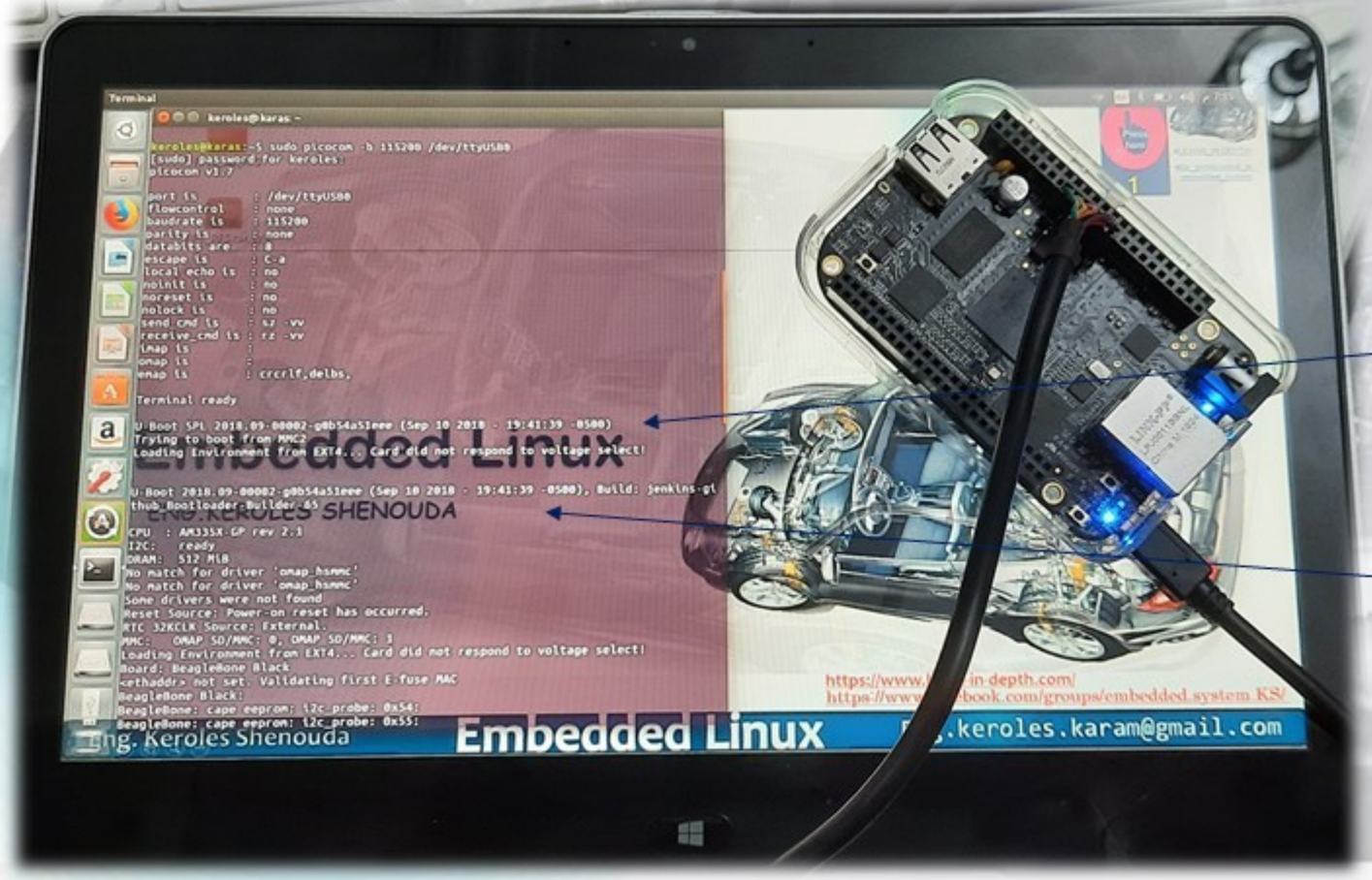
LEGEND
Power, Ground, Reset
Digital Pins
PWM Output
1.8 Volt Analog Inputs
Shared I2C Bus
Reconfigurable Digital

P8

Function	Physical Pins		Function
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DATO	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Beagle Bone Black RevC Boot Sequence



- ROM Bootloader decides boot device order based on hardware Pin settings
- ROM bootloader loads the SPL from bootdevice into SoC memory and pass control to it

BootRoom

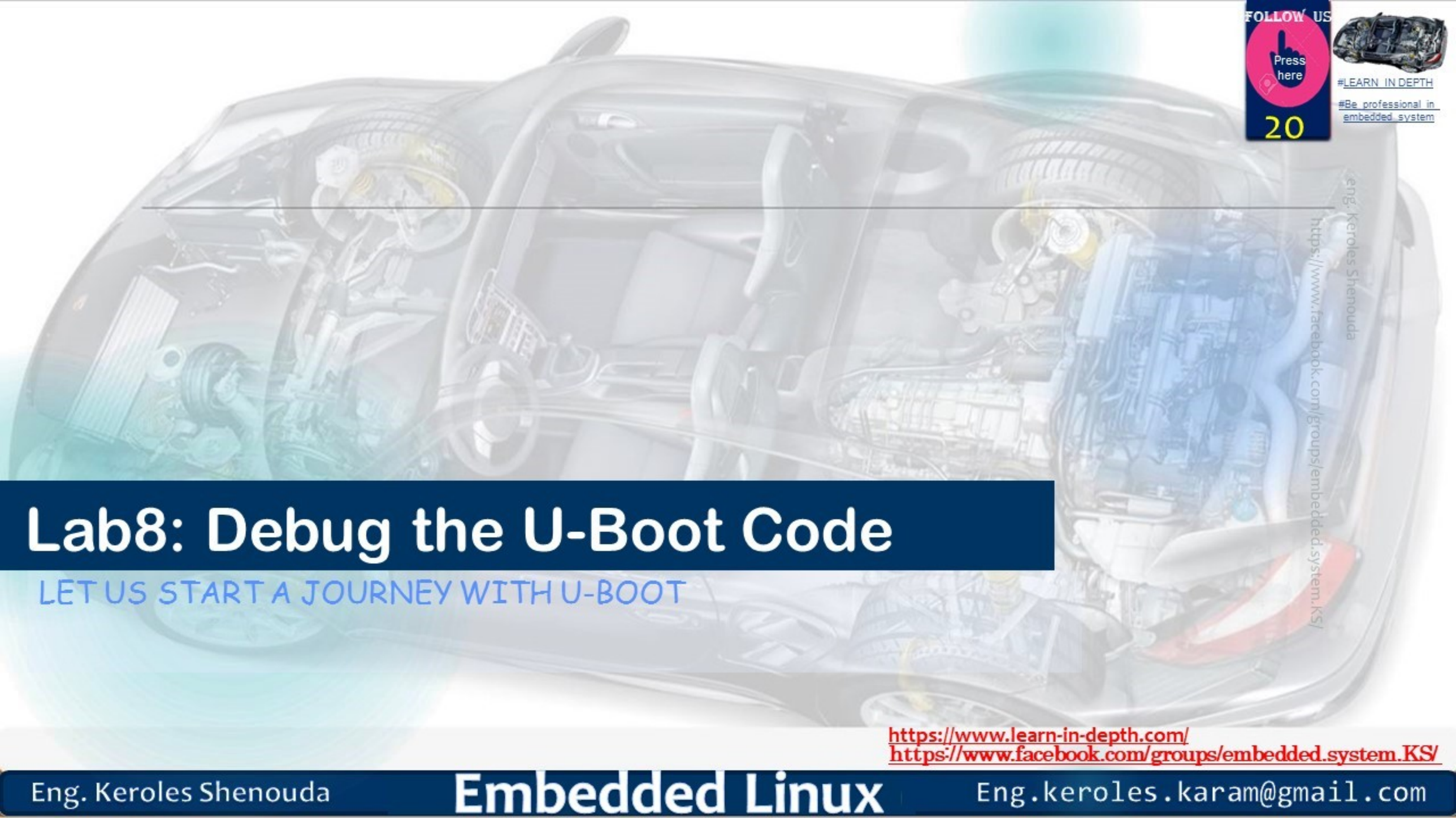
- The SPL copies the UBOOT into RAM and passes control to IT

First Stage Bootloader "SPL"

- The UBOOT loads the Kernel image from different location as you want then boot it

Second Stage Bootloader "Uboot"

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Lab8: Debug the U-Boot Code

LET US START A JOURNEY WITH U-BOOT

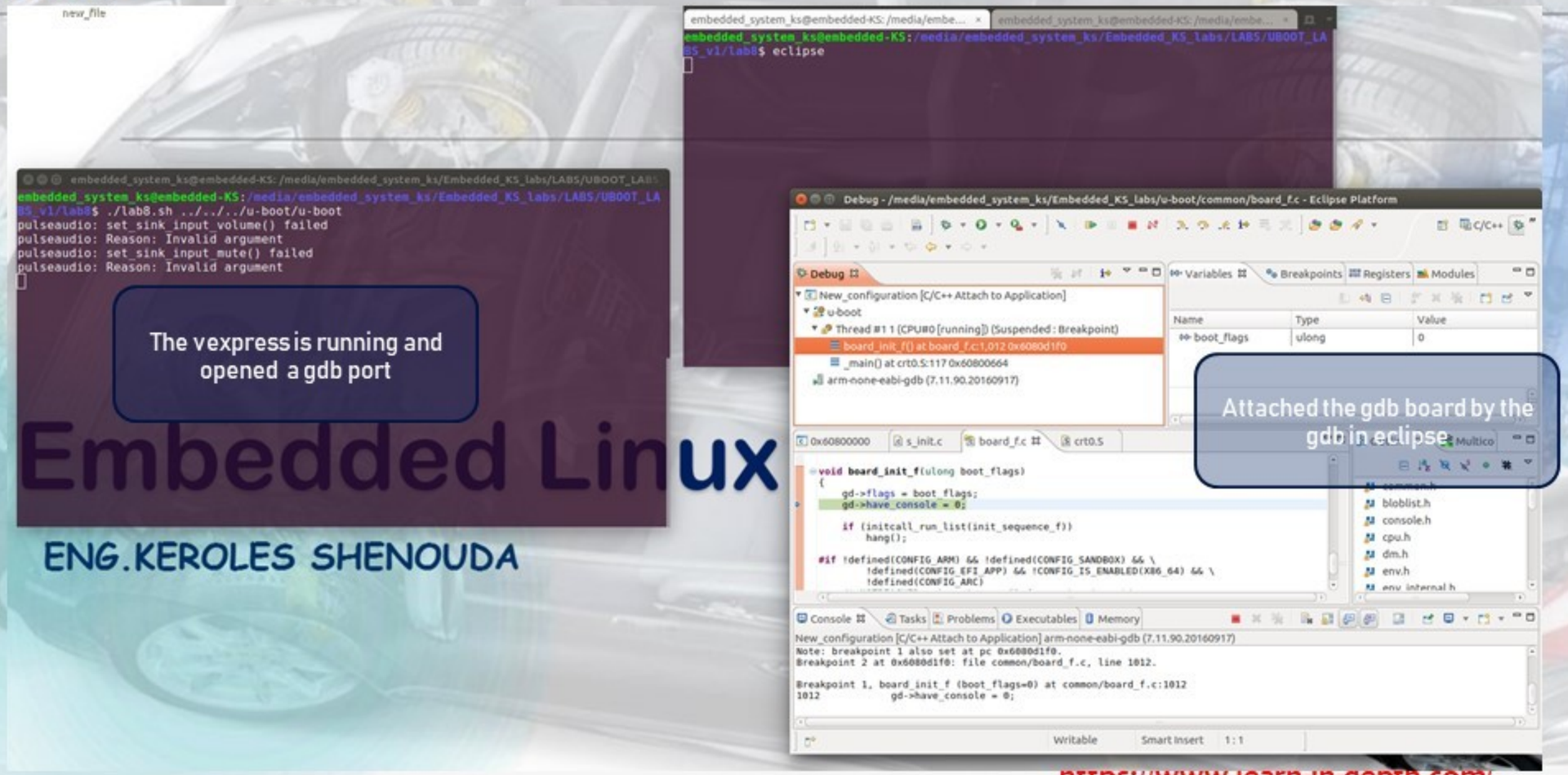
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Build u-boot for vexpress cortex A9 SoC a

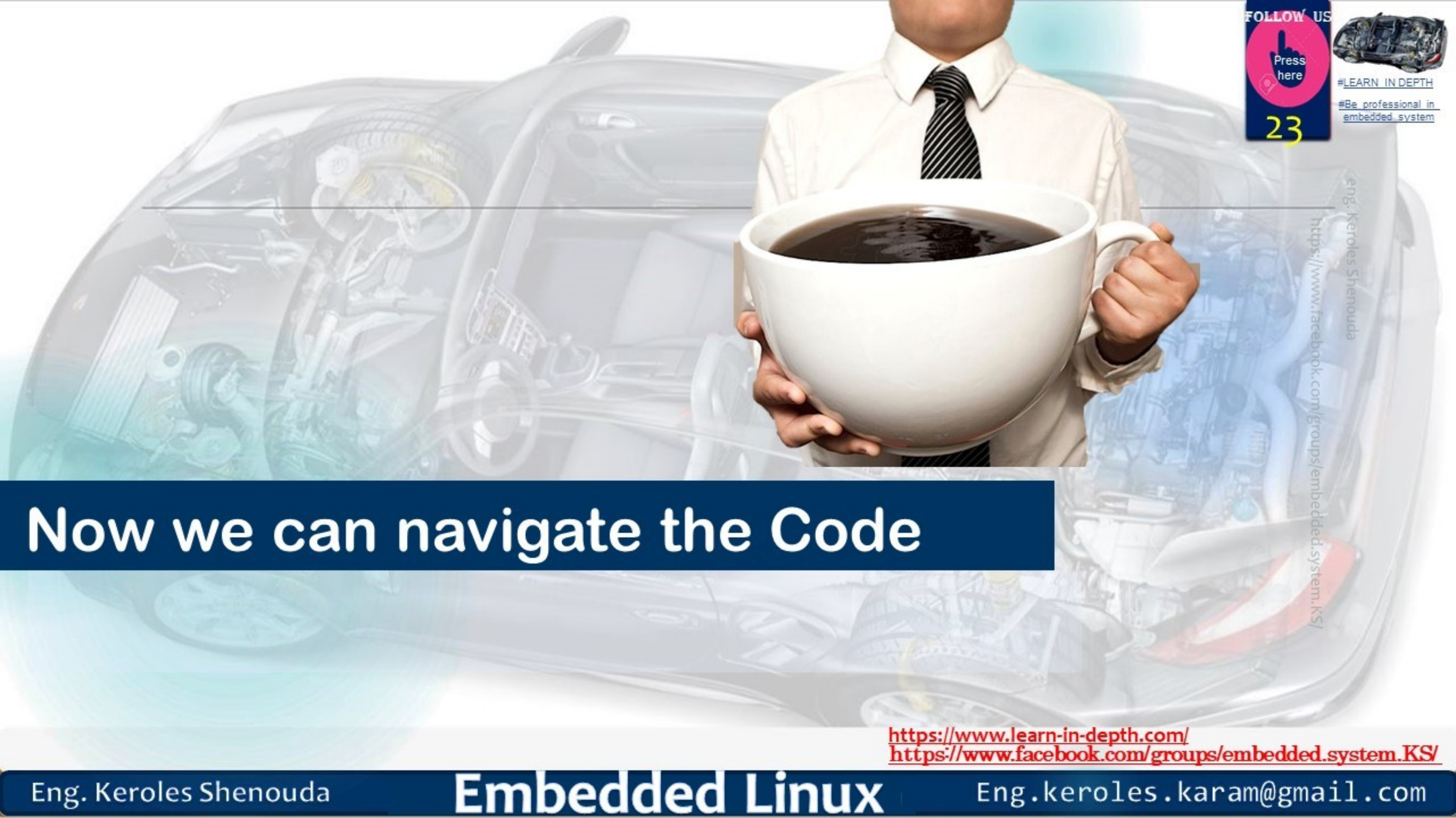
```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/u-boot
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ export ARCH=arm
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ export CROSS_COMPILE=arm-linux-gnueabi-
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make clean
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make vexpress_ca9x4_config
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make -j 16
scripts/kconfig/conf  --syncconfig Kconfig
  CHK     include/config.h
  UPD     include/config.h
  CFG     u-boot.cfg
  GEN     include/autoconf.mk.dep
  GEN     include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  CHK     include/generated/version_autogenerated.h
  CC      lib/asm-offsets.s
  CC      arch/arm/lib/asm-offsets.s
  CHK     include/generated/generic-asm-offsets.h
  UPD     include/generated/generic-asm-offsets.h
  CHK     include/generated/asm-offsets.h
  LDS     u-boot.lds
  WRAP    tools/lib/crc32.c
  HOSTCC  tools/env/embedded.o
  WRAP    tools/lib/sha1.c
  HOSTCC  tools/gen_eth_addr
  WRAP    tools/lib/crc8.c
  HOSTCC  tools/mkenvimage.o
  HOSTCC  tools/os_support.o
  HOSTCC  tools/gen_ethaddr_crc.o
  HOSTCC  tools/lib/crc32.o
  HOSTCC  tools/aisimage.o
  HOSTCC  tools/default_image.o
  HOSTCC  tools/atmelimage.o
  HOSTCC  tools/imagetool.o
  WRAP    tools/common/bootm.c
  WRAP    tools/lib/fdtdec.c
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Run the Qemu board and connect the gdb to the board gdb number



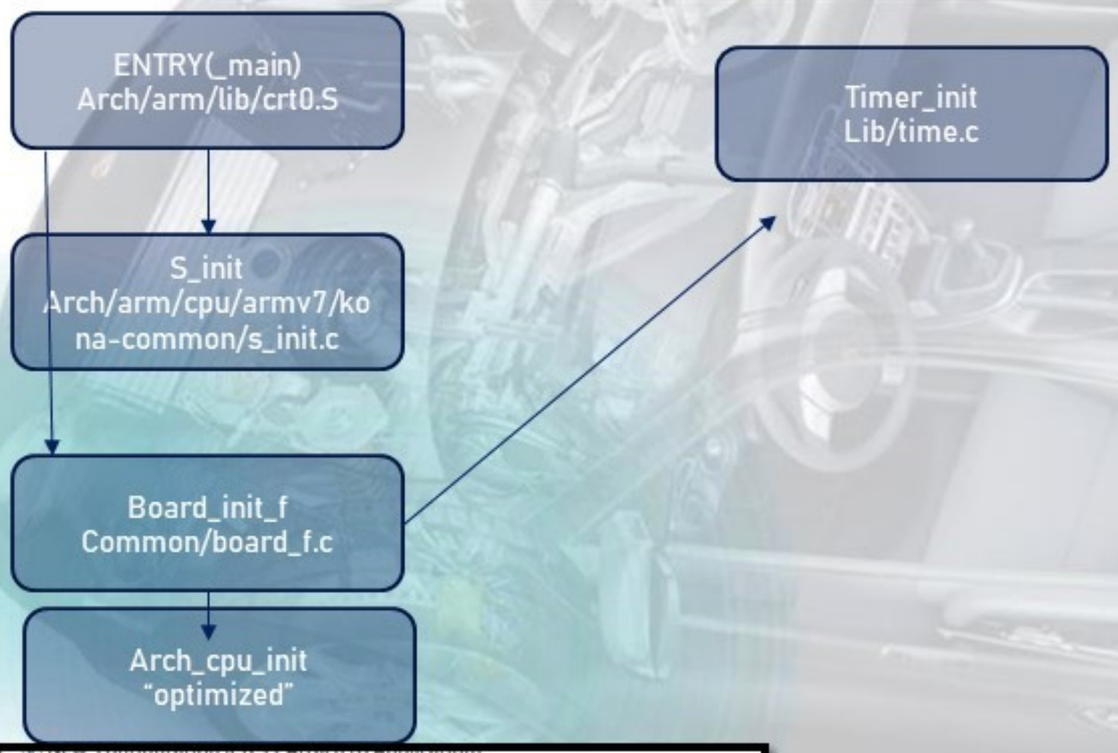
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

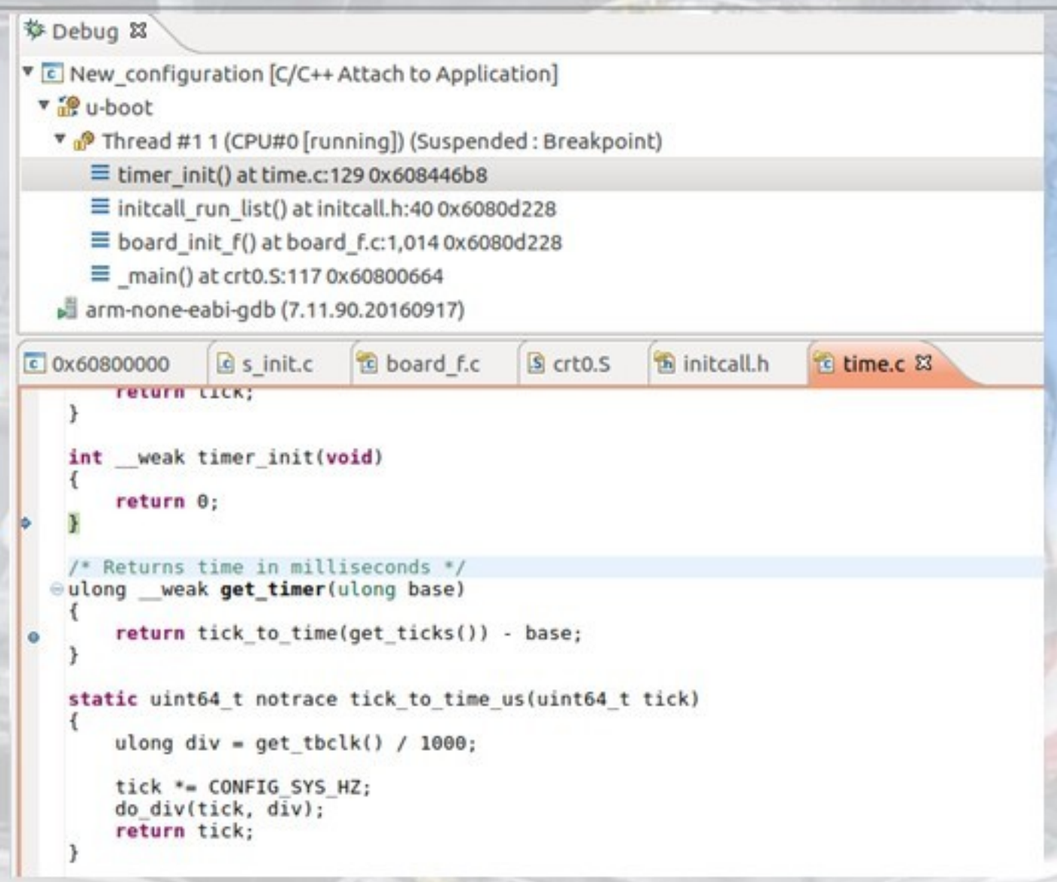


Now we can navigate the Code

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

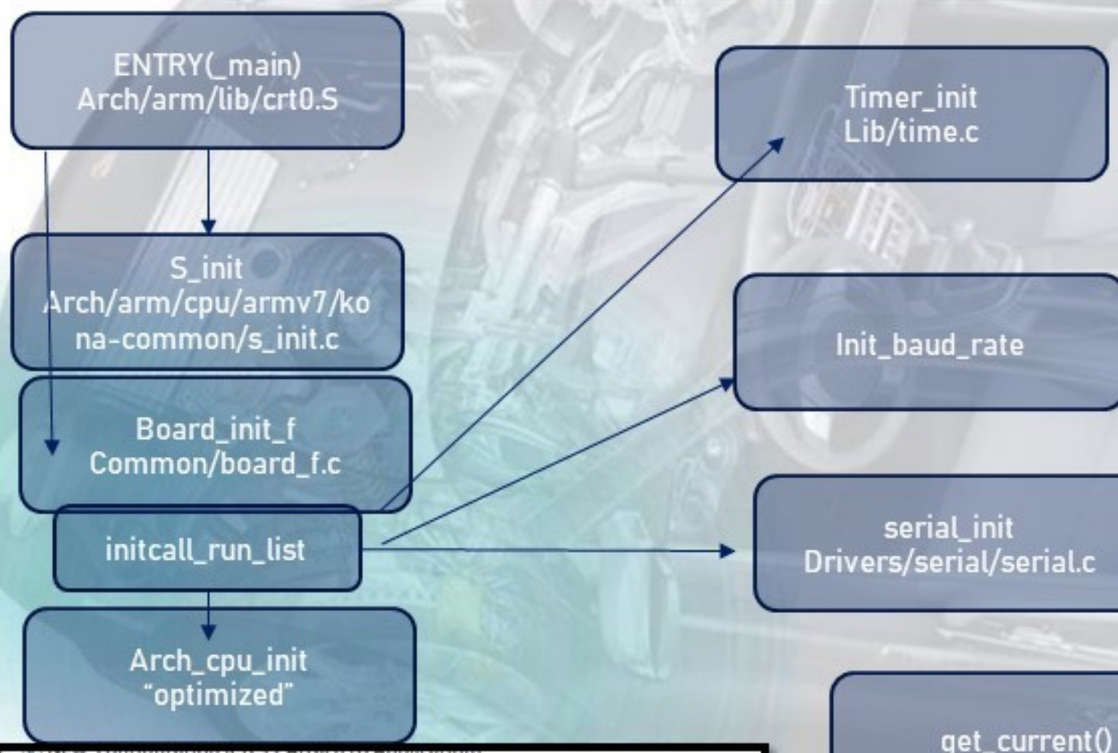
Journey on the Code

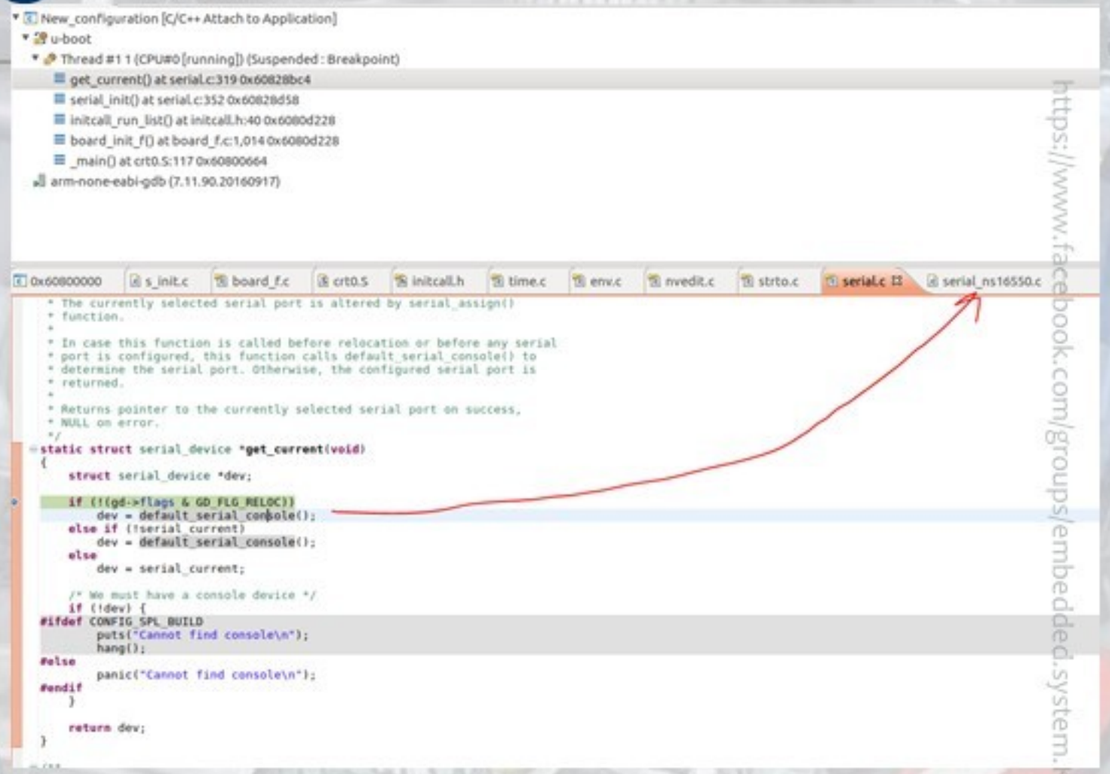




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Journey on the Code





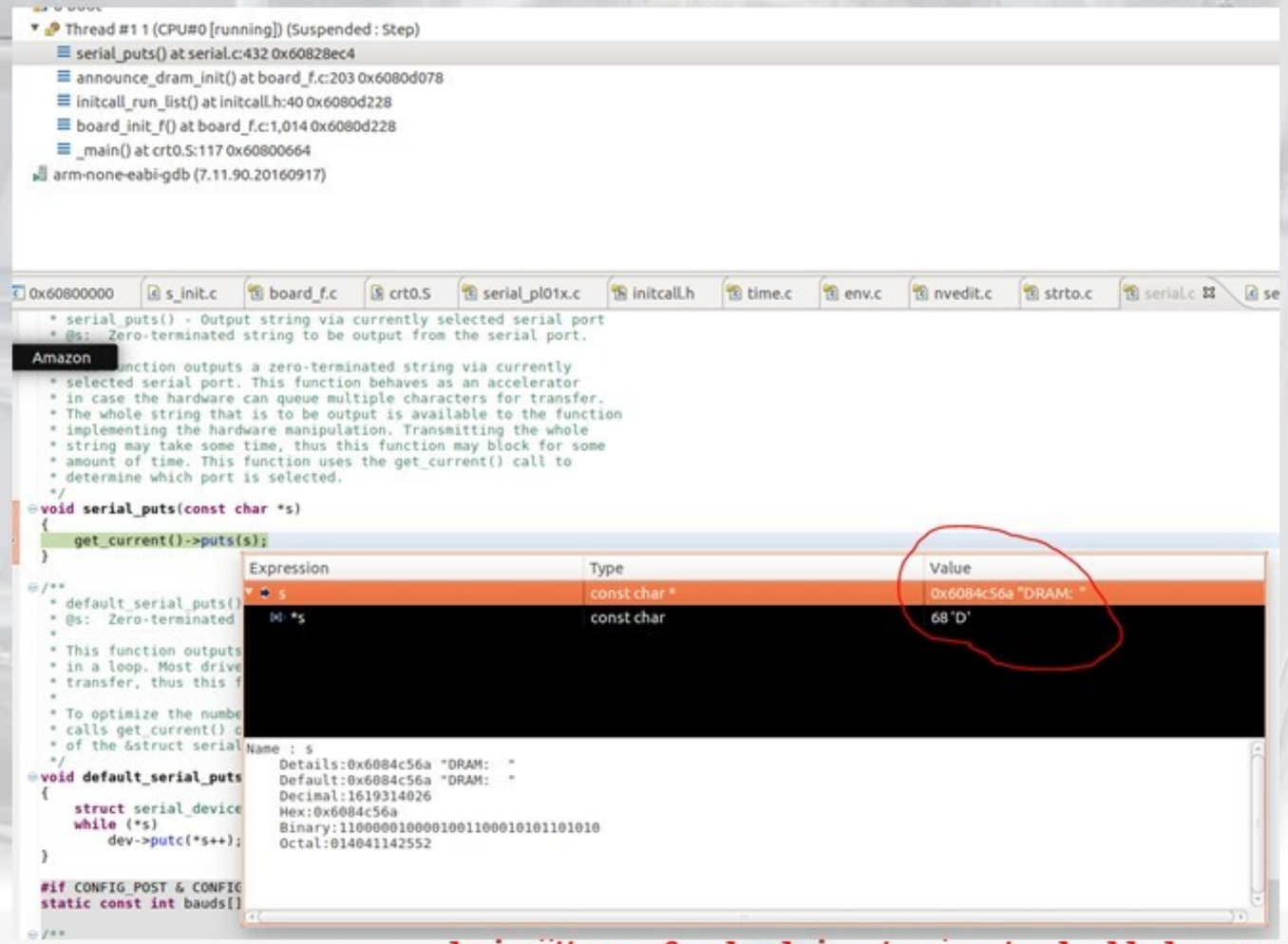
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Now the uboot can print a messages

```
static int announce_dram_init(void)
{
    puts("DRAM:  ");
    return 0;
}
```

```
 * string may take some time, thus this fu
 * amount of time. This function uses the
 * determine which port is selected.
 */
void serial_puts(const char *s)
{
    get_current()->puts(s);
}

/**
 * default_serial_puts() - Output string b
 * @s:  Zero-terminated string to be outpu
```



#LEARN IN DEPTH

#Be professional in embedded system

Now the uboot can print a messages Cont.

```
static int announce_dram_init(void)
{
    puts("DRAM:  ");
    return 0;
}
```

```
 * string may take some time, thus this fu
 * amount of time. This function uses the
 * determine which port is selected.
 */
void serial_puts(const char *s)
{
    get_current()->puts(s);
}

/**
 * default_serial_puts() - Output string b
 * @s:  Zero-terminated string to be outpu
```

```
void default_serial_puts(const char *s)
{
    struct serial_device *dev = get_current();
    while (*s)
        dev-
}

#if CONFIG_F
static const

/**
 * uart_post
 * @flags:
 *
 * Do a loop
```

Expression	Type	Value
s	const char *	0x6084c56a "DRAM: "
*s	const char	68 'D'

```
embedded_system_ks@embedded-KS: /media/embe...  ×  embedded_system_ks@embedded-KS: /media/embe...  ×
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/UBOOT_LA
BS_v1/lab8$ ./lab8.sh ../../../u-boot/u-boot
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument

U-Boot 2020.01-rc2-00035-g3ff1ff3ff7-dirty (Jan 31 2020 - 18:16:16 +0200)

DRAM:
```



```
static int pl01x_putc(struct pl01x_regs *regs, char c)
{
    /* Wait until there is space in the FIFO */
    if (readl(&regs->fr) & UART_PL01x_FR_TXFF)
        return -EAGAIN;

    /* Send the character */
    writel(c, &regs->dr);

    return 0;
}
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

Dram_size

Debug

New_configuration [C/C++ Attach to Application]
u-boot
Thread #1 1 (CPU#0 [running]) (Suspended : Step)
default_serial_puts() at serial.c:450 0x60828edc
printf() at vsprintf.c:807 0x6084584c
print_size() at display_options.c:126 0x60842a68
show_dram_config() at board_f.c:228 0x6080d164
initcall_run_list() at initcall.h:40 0x6080d228
board_init_f() at board_f.c:1,014 0x6080d228
_main() at crt0.S:117 0x60800664
arm-none-eabi-gdb (7.11.90.20160917)

0x60800000 | board_f.c | crt0.S | serial_pl01x.c | initcall.h | nvedit.c | strto.c | serial.c | serial_ns16550.c

```
 * in case the hardware can queue multiple characters for transfer.
 * The whole string that is to be output is available to the function
 * implementing the hardware manipulation. Transmitting the whole
 * string may take some time, thus this function may block for some
 * amount of time. This function uses the get_current() call to
 * determine which port is selected.
 */
void serial_puts(const char *s)
{
    get_current()->puts(s);
}

/**
 * default_serial_puts() - Output string by calling serial_putc() in loop
 * @s:  Zero-terminated string to be output from the serial port.
 *
 * This function outputs a zero-terminated string by calling serial_putc()
 * in a loop. Most drivers do not support queueing more than one byte for
 * transfer, thus this function precisely implements their serial_puts().
 *
 * To optimize the number of get_current() calls, this function only
 * calls get_current() once and then directly accesses the putc() call
 * of the &struct serial_device .
 */
void default_serial_puts(const char *s)
{
    struct serial_device *dev = get_current();
    while (*s)
        dev->putc(*s++);
}

#if CONFIG_POST & CONFIG_SYS_POST_UART
static const int bauds[] = CONFIG_SYS_BAUDRATE_TABLE;
```

embedded_system_ks@embedded

```
embedded_system_ks@embedd
BS_v1/lab8$ ./lab8.sh ../
pulseaudio: set_sink_inpu
pulseaudio: Reason: Inval
pulseaudio: set_sink_inpu
pulseaudio: Reason: Inval

U-Boot 2020.01-rc2-00035-

DRAM:  51
```

Debug

New_configuration [C/C++ Attach to Application]
u-boot
Thread #1 1 (CPU#0 [running]) (Suspended : Step)
default_serial_puts() at serial.c:452 0x60828ee8
printf() at vsprintf.c:807 0x6084584c
print_size() at display_options.c:126 0x60842a68
show_dram_config() at board_f.c:228 0x6080d164
initcall_run_list() at initcall.h:40 0x6080d228
board_init_f() at board_f.c:1,014 0x6080d228
_main() at crt0.S:117 0x60800664
arm-none-eabi-gdb (7.11.90.20160917)

0x60800000 | board_f.c | crt0.S | serial_pl01x.c | initcall.h | nvedit.c | strto.c | serial.c | serial_ns16550.c | vexpres

```
 * The whole string that is to be output is available to the function
 * implementing the hardware manipulation. Transmitting the whole
 * string may take some time, thus this function may block for some
 * amount of time. This function uses the get_current() call to
 * determine which port is selected.
 */
void serial_puts(const char *s)
{
    get_current()->puts(s);
}

/**
 * default_serial_puts() - Output string by calling serial_putc() in loop
 * @s:  Zero-terminated string to be output from the serial port.
 *
 * This function outputs a zero-terminated string by calling serial_putc()
 * in a loop. Most drivers do not support queueing more than one byte for
 * transfer, thus this function precisely implements their serial_puts().
 *
 * To optimize the number of get_current() calls, this function only
 * calls get_current() once and then directly accesses the putc() call
 * of the &struct serial_device .
 */
void default_serial_puts(const char *s)
{
    struct serial_device *dev = get_current();
    while (*s)
        dev->putc(*s++);
}

#if CONFIG_POST & CONFIG_SYS_POST_UART
static const int bauds[] = CONFIG_SYS_BAUDRATE_TABLE;

/**
 * uart_post_test() - Test the currently selected serial port using POST
 * @flags:  POST framework flags
```

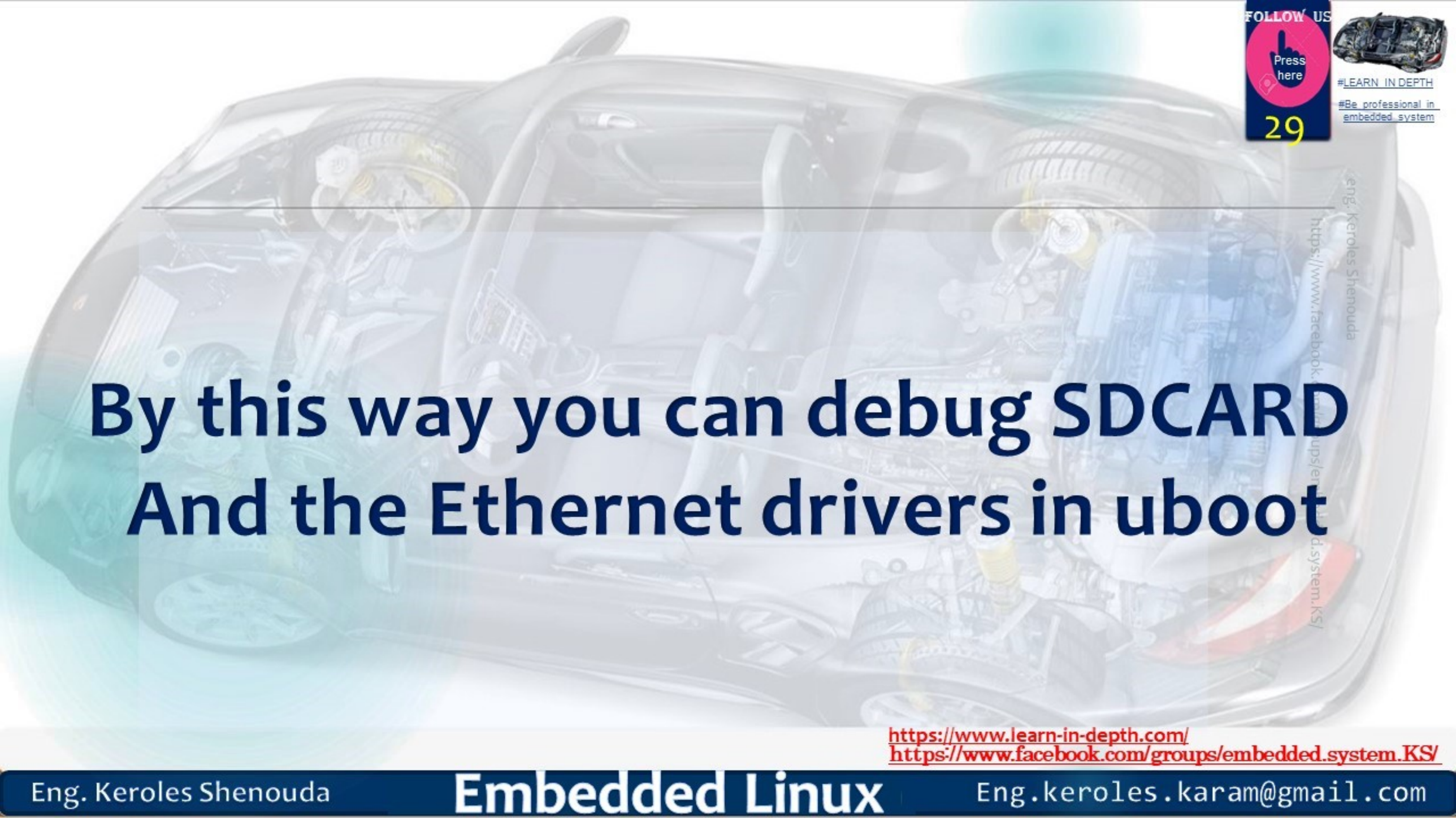
embedded_system_ks@embedded-KS: /media/emb

```
embedded_system_ks@embedded-KS:/med
BS_v1/lab8$ ./lab8.sh ../../../u-boo
pulseaudio: set_sink_input_volume()
pulseaudio: Reason: Invalid argumen
pulseaudio: set_sink_input_mute() f
pulseaudio: Reason: Invalid argumen

U-Boot 2020.01-rc2-00035-g3ff1ff3ff

DRAM:  512
```

https://www.facebook.com/groups/embedded.system.KS/



By this way you can debug SDCARD And the Ethernet drivers in uboot

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

port U-boot to a new board (Basics)

OBJECTIVE: TO UNDERSTAND INDEPTH THE UBOOT INFRASTRUCTURE

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

U-Boot directories

- **Arch**: anything arch or platform related: DTS, CPU init, pinmux controller, DRAM, clocks, ...
- **Board**: code board specific (init, pinmuxing configuration, etc), Kconfig file specifying board header file, board file, paths, Makefile for board file,
- **Configs**: all boards' defconfigs
- **Drivers**: Device drivers implemented in drivers/
  A lot of reuse from Linux drivers!
- **include**: all headers
- **include/configs**: all boards' header files

#LEARN IN DEPTH

#Be professional in embedded system

new board support workflow

- 1. Create the board file,

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/
```

- 2. Create the board Kconfig file,

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/Kconfig
```

- 3. Create the board Makefile,

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/Makefile
```

- 4. Create the board defconfig,

```
/Embedded_KS_labs/u-boot$ ls configs/my_KS_board_defconfig
```

- 5. Create the board header file,

```
Embedded_KS_labs/u-boot$ ls include/configs/my_KS_board.h
```

- 6. Source board's Kconfig in the architecture's Kconfig,

```
/Embedded_KS_labs/u-boot$ ls arch/arm/mach-imx/Kconfig
```

- 7. Define the TARGET Kconfig option in its CPU's Kconfig

```
Embedded_KS_labs/u-boot$ ls arch/arm/Kconfig
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

1. Create the board file

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/
```

- DECLARE_GLOBAL_DATA_PTR,
- usable in code with gd global variable,
- on ARM, equals to hardware register r9 for ARM32 and x18 for ARM64,

```
//www.learn-in-depth.com
//be professional in embedded system
//Board name is my_KS_board
#include <common.h>
#include <malloc.h>
#include <errno.h>
#include <netdev.h>
#include <asm/io.h>
#include <asm/mach-types.h>


// include all the Drivers
//#include "../drivers/mmc/arm_pl180_mmci.h"

//Decleare Uboot Global variables

DECLARE_GLOBAL_DATA_PTR;


int board_init(void)
{
    return 0;
}


int board_eth_init(bd_t *bis)
{
    int rc = 0;
#ifdef CONFIG_SMC911X

#endif
    return rc;
}

int cpu_mmc_init(bd_t *bis)
{
    int rc = 0;

#ifdef CONFIG_ARM_PL180_MMCI


#endif
    return rc;
}
static void flash__init(void)
{

}


int dram_init(void)
{

//  for example if we need to get IMX6Q ddr size under arch/arm/mach-imx/mmcd_size.c
//  gd->ram_size = imx_ddr_size();
    return 0;
}

int dram_init_banksize(void)
{


    return 0;
}

/*
 * Start timer:
 *    Setup a 32 bit timer, running at 1KHz
 */
void my_timer_init(void)
{

}
```

Based on Specific driver configuration

2. Create the board Kconfig file

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/Kconfig
```

- SYS_VENDOR and SYS_BOARD are used to identify the directory where make find the files it needs to compile,
- if both are present,
  - board/SYS_VENDOR/SYS_BOARD/
- if SYS_VENDOR is omitted,
  - board/SYS_BOARD/
- SYS_CONFIG_NAME is used to identify the board header file,
  - include/configs/SYS_CONFIG_NAME.h

Kconfig | my_KS_board.c

```
1 if TARGET_MY_KS_BOARD
2
3 config SYS_BOARD
4         default "my_KS_board"
5
6 config SYS_VENDOR
7         default "my_KS_vendor"
8
9 config SYS_CONFIG_NAME
10        default "my_KS_board"
11 endif
12
```

3. Create the board Makefile

```
/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/Makefile
```

Makefile ×

```
1 obj-y := my_KS_board.o
```

Kconfig ×

4. Create the board defconfig

```
Embedded_KS_labs/u-boot$ ls configs/my_KS_board_defconfig
```

- put here anything that is selectable in Kconfig (menuconfig),
- drivers, features, U-Boot behaviour, libs, etc.

```
1 CONFIG_ARM=y
2 CONFIG_ARCH_MX6=y
3 CONFIG_TARGET_MY_KS_BOARD=y
4 CONFIG_MXC_UART=y
5
```

5. Create the board header file

```
Embedded_KS_labs/u-boot$ ls include/configs/my_KS_board.h
```

my_KS_board.h | Makefile | Kconfig | my_KS_board.c | my_KS_board_defconfig

```
1 #ifndef __MY_KS_BOARD_CONFIG_H__
2 #define __MY_KS_BOARD_CONFIG_H__
3
4
5 //fill each base address for each embedded module in your board as this example
6 #define CONFIG_MXC_UART_BASE UART5_BASE
7
8 #include "mx6_common.h"
9
10 #define CONFIG_NR_DRAM_BANKS 1
11 #define CONFIG_SYS_MAX_FLASH_BANKS 1
12
13
14 #define CONFIG_ENV_SIZE                    0x3000
15 #define CONFIG_SYS_INIT_SP_ADDR            0x60100000
16
17
18 #endif
```

6. Source board's Kconfig in the architecture's Kconfig

```
Embedded_KS_labs/u-boot$ ls arch/arm/mach-imx/Kconfig
```

```
1834 source "board/variscite/dart_6ul/Kconfig"
1835 source "board/vscom/baltos/Kconfig"
1836 source "board/woodburn/Kconfig"
1837 source "board/xilinx/Kconfig"
1838 source "board/xilinx/zynq/Kconfig"
1839 source "board/xilinx/zynqmp/Kconfig"
1840 source "board/phytium/durian/Kconfig"
1841
1842 source "board/my_KS_vendor/my_KS_board/Kconfig"
1843
1844 source "arch/arm/Kconfig.debug"
```

7. Define the TARGET Kconfig option in its CPU's Kconfig

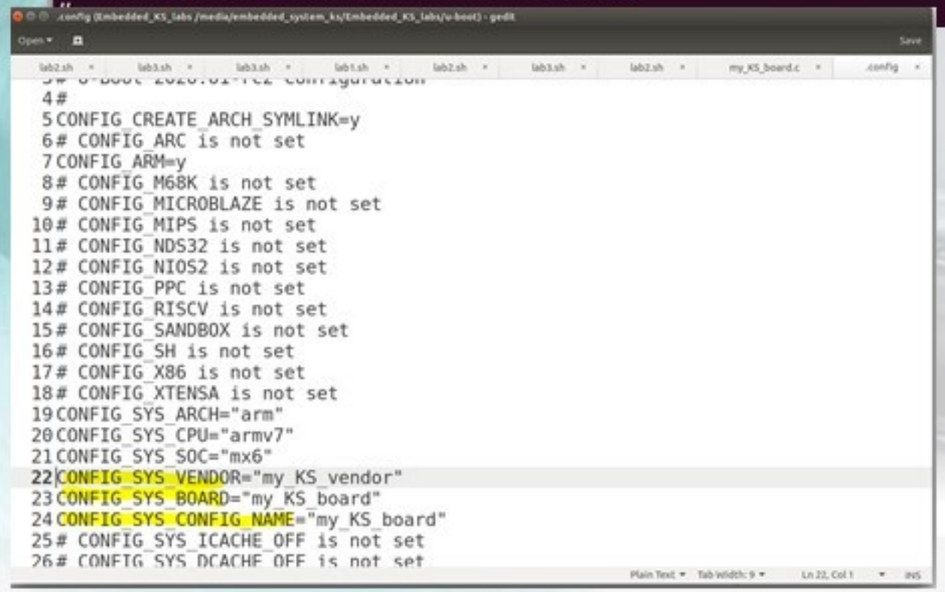
Kconfig × | Kconfig × | my_KS_I

```
1 if ARCH_MX6
2
3 config TARGET_MY_KS_BOARD
4         bool "My KS board"
5         select MX6S
6
7 config MX6_SMP
8         bool
9         select ARM_ERRATA_751472
10        select ARM_ERRATA_761320
11        select ARM_ERRATA_794072
12        select ARM_ERRATA_845369
13        select MP
14
```

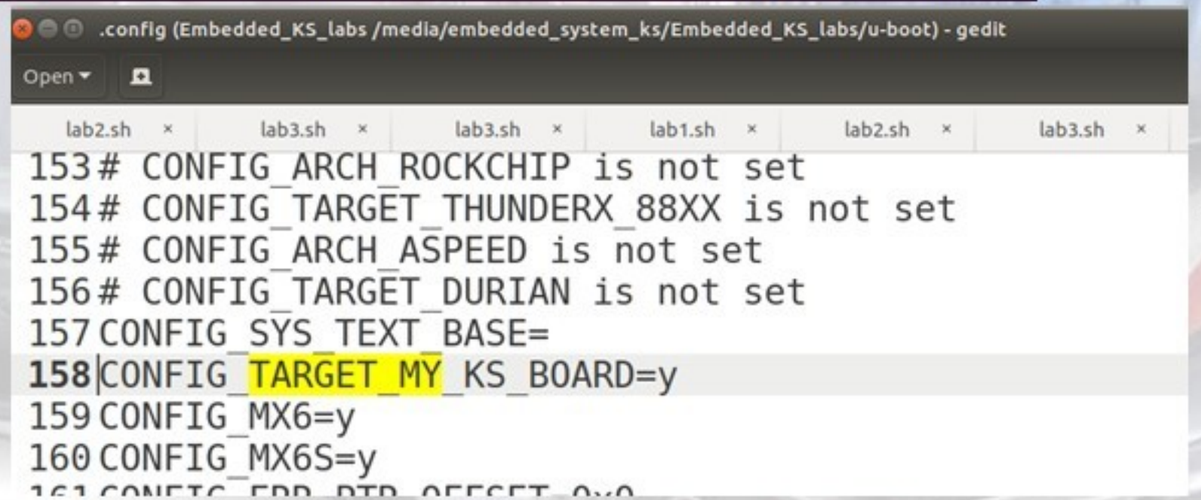
Configure the u-boot

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ ls configs/my_KS_board_defconfig
configs/my_KS_board_defconfig
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ make my_KS_board_config
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
```

You can see the .config file target the new board



```
4#
5 CONFIG_CREATE_ARCH_SYMLINK=y
6 # CONFIG_ARC is not set
7 CONFIG_ARM=y
8 # CONFIG_M68K is not set
9 # CONFIG_MICROBLAZE is not set
10 # CONFIG_MIPS is not set
11 # CONFIG_NDS32 is not set
12 # CONFIG_NIOS2 is not set
13 # CONFIG_PPC is not set
14 # CONFIG_RISCV is not set
15 # CONFIG_SANDBOX is not set
16 # CONFIG_SH is not set
17 # CONFIG_X86 is not set
18 # CONFIG_XTENSA is not set
19 CONFIG_SYS_ARCH="arm"
20 CONFIG_SYS_CPU="armv7"
21 CONFIG_SYS_SOC="mx6"
22 CONFIG_SYS_VENDOR="my_KS_vendor"
23 CONFIG_SYS_BOARD="my_KS_board"
24 CONFIG_SYS_CONFIG_NAME="my_KS_board"
25 # CONFIG_SYS_ICACHE_OFF is not set
26 # CONFIG_SYS_DCACHE_OFF is not set
```



.config (Embedded_KS_labs /media/embedded_system_ks/Embedded_KS_labs/u-boot) - gedit

Open

lab2.sh × lab3.sh × lab3.sh × lab1.sh × lab2.sh × lab3.sh ×

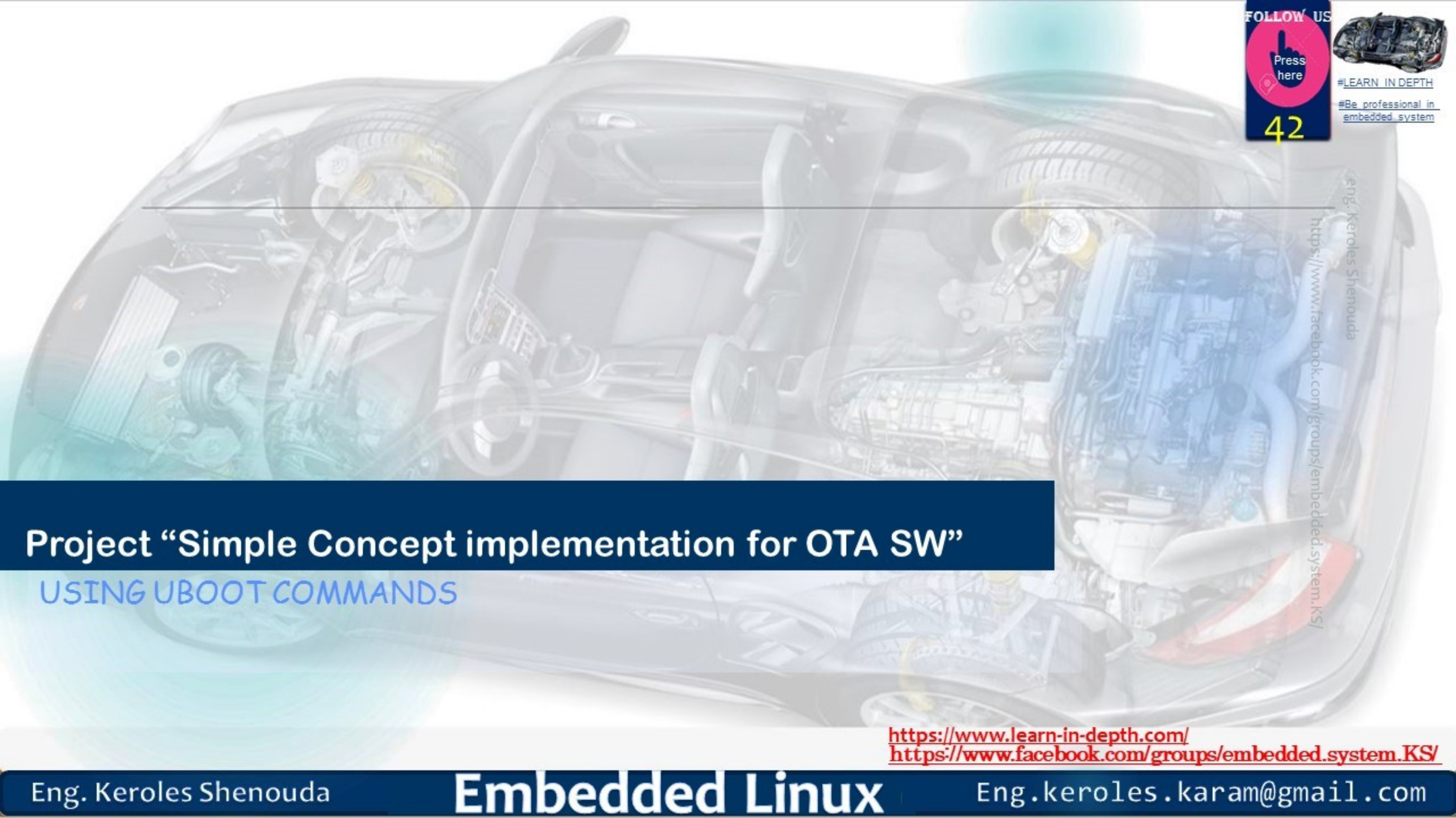
```
153 # CONFIG_ARCH_ROCKCHIP is not set
154 # CONFIG_TARGET_THUNDERX_88XX is not set
155 # CONFIG_ARCH_ASPEED is not set
156 # CONFIG_TARGET_DURIAN is not set
157 CONFIG_SYS_TEXT_BASE=
158 CONFIG_TARGET_MY_KS_BOARD=y
159 CONFIG_MX6=y
160 CONFIG_MX6S=y
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Then build it

After the build you can see the my_KS_board.o object file is created

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/
built-in.o          Kconfig             my_KS_board.c       .my_KS_board.o.cmd
.built-in.o.cmd     Makefile            my_KS_board.o       my_KS_board.su
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/u-boot$ ls board/my_KS_vendor/my_KS_board/
```

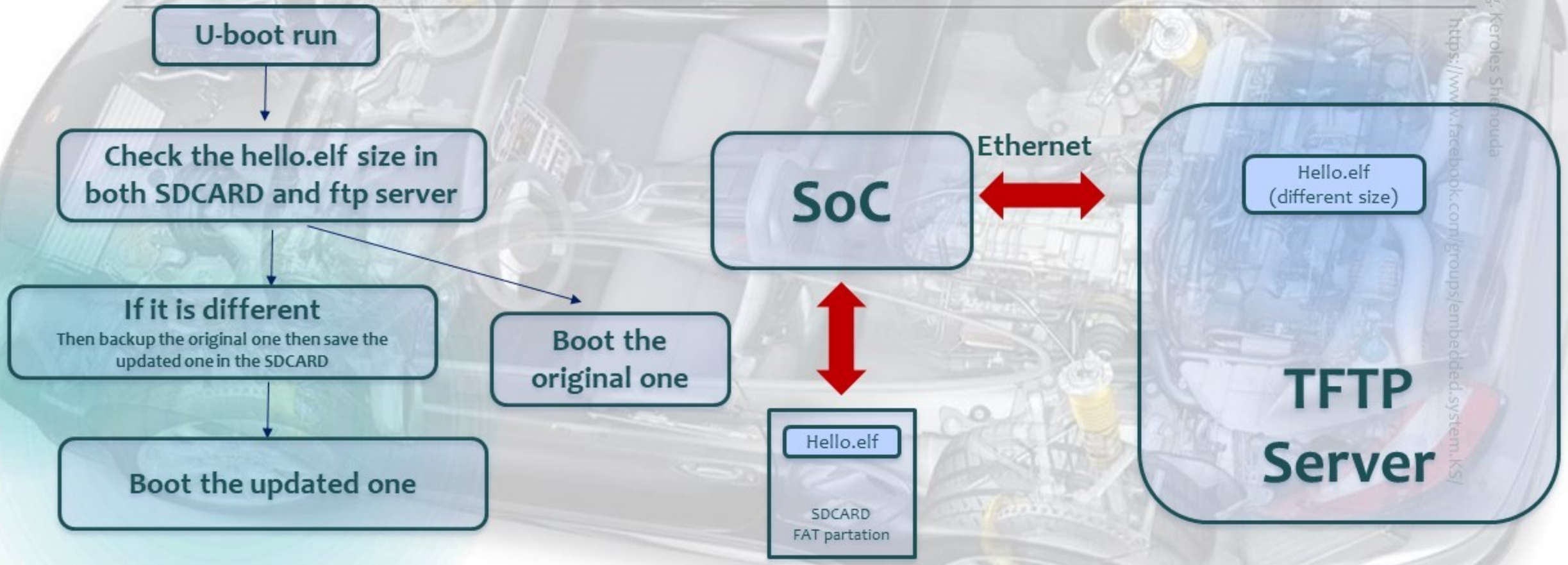


Project "Simple Concept implementation for OTA SW"

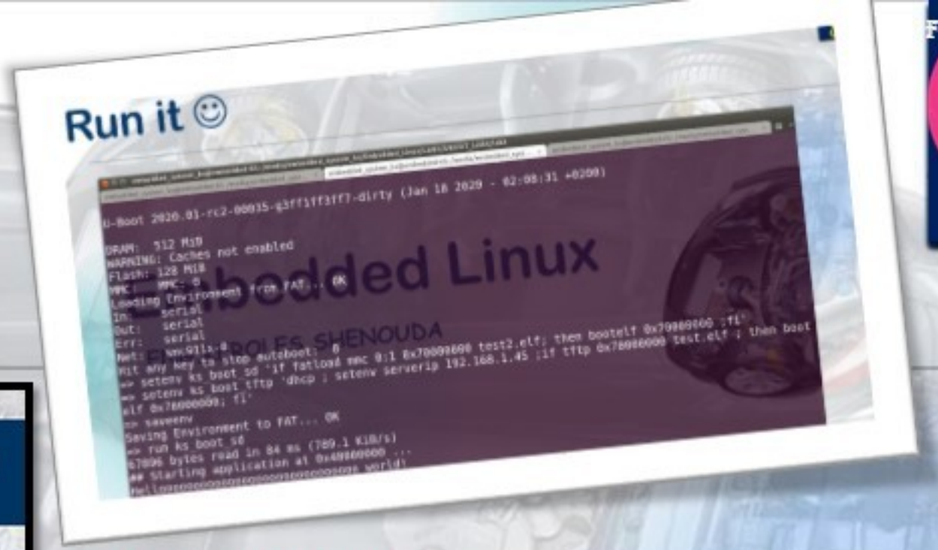
USING UBOOT COMMANDS

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Simple Concept implementation for OTA SW



Topics can help you
So investigate on it well



Lab5:make the tftp automatically run, if it is failed then boot from sdcard automatically.

KINDLY NOTE ALL OF THOSE AFTER DELAY 7 SEC

```
# Update Linux kernel  from usb flash to nand flash
fatload usb 0:1 ${loadaddr} uImage;
nand erase.part kernel;
nand write ${loadaddr} kernel ${filesize};
```

```
=> tftp $loadaddr rootfs.img
Using FEC device
TFTP from server 192.168.1.1; our IP address is 192.168.20.56
Filename 'rootfs.img'.
Load address: 0x70800000
Loading: #################################################################
         #################################################################
         ##############################################################
763.7 KiB/s
done
Bytes transferred = 16385024 (fa0400 hex)
=> mmc part

Partition Map for MMC device 0  --   Partition Type: DOS

Part    Start Sector    Num Sectors    UUID        Type
  1     8192            131072         00000000-01    83
  2     139264          7753728        00000000-02    83
=> mmc write $loadaddr 0x22000 $cnt

MMC write: dev # 0, block # 139264, count size ...   (0x22000  == 139264 )
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Learn-in-depth.com

References

- Embedded Linux training
  - https://bootlin.com/training/
- Linux kernel and driver development training
- Yocto Project and OpenEmbedded development training
- Buildroot development training
- Building Embedded Linux Systems, 2nd Edition
- Mastering Embedded Linux Programming - Second Edition
- Christopher Hallinan, **Embedded Linux Primer,** Prentice Hall, 2006.
- **Silberschatz**, Galvin, and Gagne's **Operating System Concepts**, **Eighth Edition**.
- Robert love, linux kernel develoepemnt 3 rd edition 2010
- **Advanced Linux Programming By Mark Mitchell, Jeffrey Oldham, Alex Samuel**
- **Linux OS in Embedded Systems & Linux Kernel Internals(2/2)**
  - Memory Management, Paging, Virtual Memory, File system and its implementation, Secondary Storage(HDD), I/O systems

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- Memory Management article
- Introduction to Operating Systems Class 9 - Memory Management
- Virtual Memory lecture for Introduction to Computer Architecture at Uppsala University.
- OS Lecture Note 13 - Address translation, Caches and TLBs
- CSE 331 Operating Systems Design lectures
- CSE 331 Virtual Memory
- CSE 332 Computer Organization and Architecture
- File Systems: Fundamentals.
- Linux System Programming
- System Calls, POSIX I/O
  CSE 333 Spring 2019, Justin Hsia

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- **File Permissions in Linux/Unix with Example**
- Introduction to UNIX / Linux – 4
- **12.2 Basic I/O Concepts**
- Communicating with Hardware
- I/O Systems I/O Hardware Application I/O Interface
- COMPUTER ARCHITECTURE
- **OS**
- Linux Operating System
- W4118 Operating Systems, Instructor: Junfeng Yang
- CSNB334 Advanced Operating Systems 4. Process & Resource Management.
- Using the POSIX API
- **Linux Memory Management**
- Porting U-Boot and Linux on new ARM boards
- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- U-Boot Environment Variables

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

References Cont.

- Using U-boot as production test strategy -- really?
- TFTP Boot using u-boot
- Loading the kernel with TFTP and U-boot
- https://blog.3mdeb.com/2013/2013-06-07-0x5-qemu-network-configuration-and-tftp-for-virtual-development-board/
- Installing and Testing TFTP Server in Ubuntu
- Pthreads: A shared memory programming model https://slideplayer.com/slide/8734550/
- Signal Handling in Linux Tushar B. Kute
- ARM case-study: the raspberry pi
- Introduction to Sockets Programming in C using TCP/IP
- The Broadcom chip used in the Raspberry Pi 2 Model B