#LEARN IN DEPTH

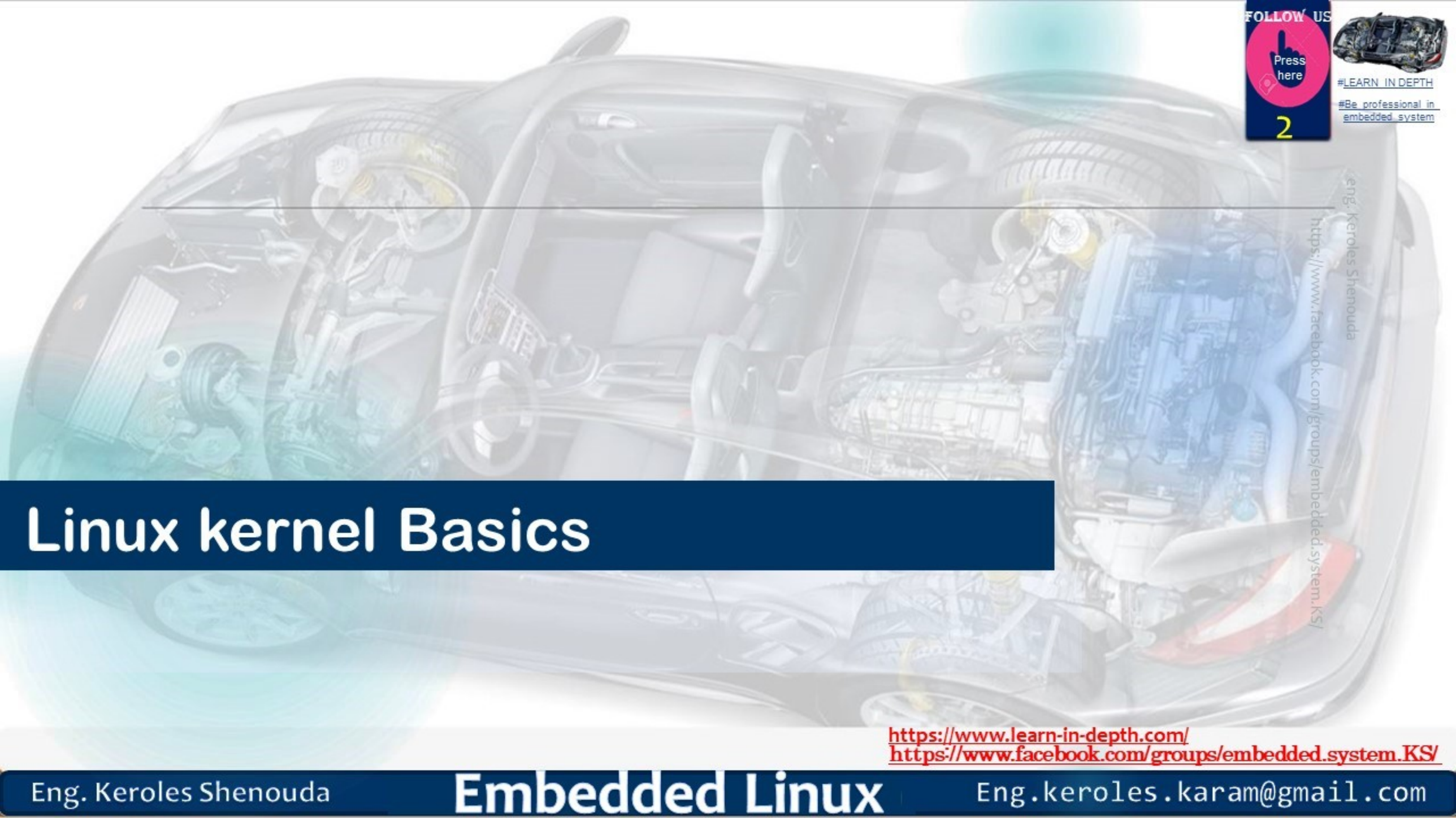
#Be professional in embedded system

Embedded Linux (PART 9)

- LINUX KERNEL BASICS
- BUILDING THE LINUX KERNEL
- LAB1:BUILDNG/RUNNING LINUX KERNEL ON VEXPRESS A9
- LAB2: BUILD/RUN LINUX ON RPI2
- DEVICE TREES
- LINUX START-UP SEQUENCE

ENG.KEROLES SHENOUDA

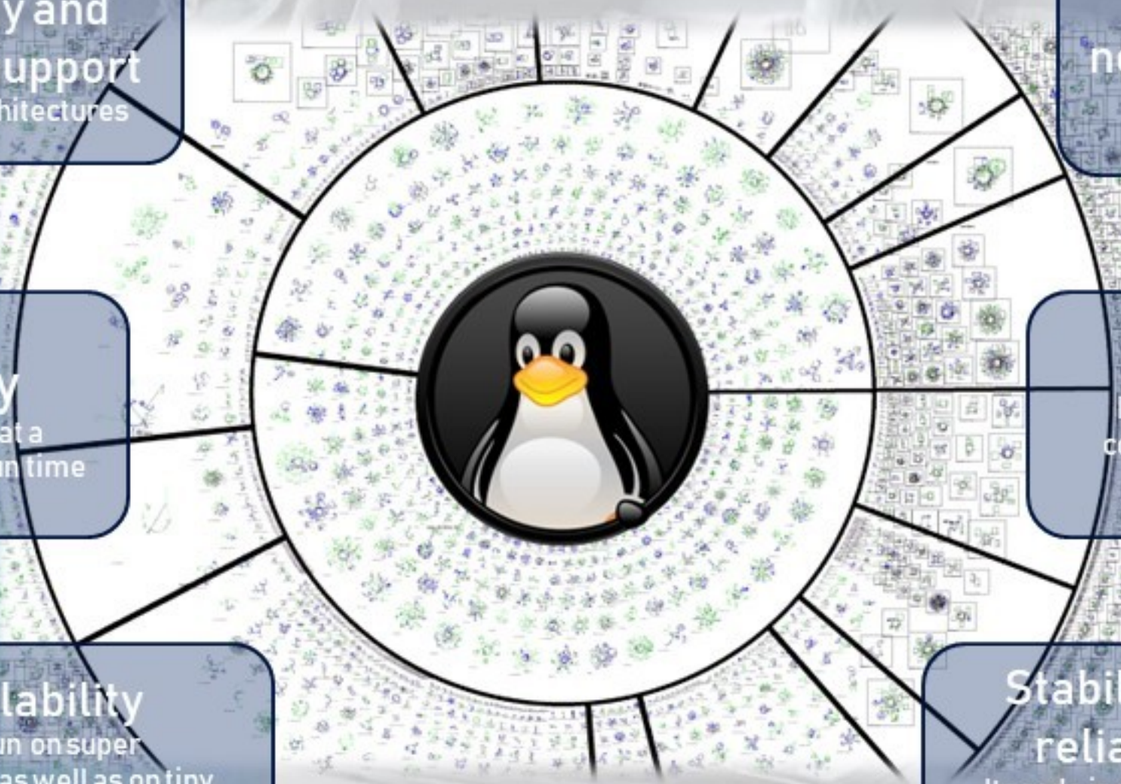
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Linux kernel Basics

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

Linux kernel key features



Portability and hardware support
Runs on most architectures

networking support

Modularity
Can include only what a system needs even at run time

Security
It can't hide its flaws. Its code is reviewed by many experts

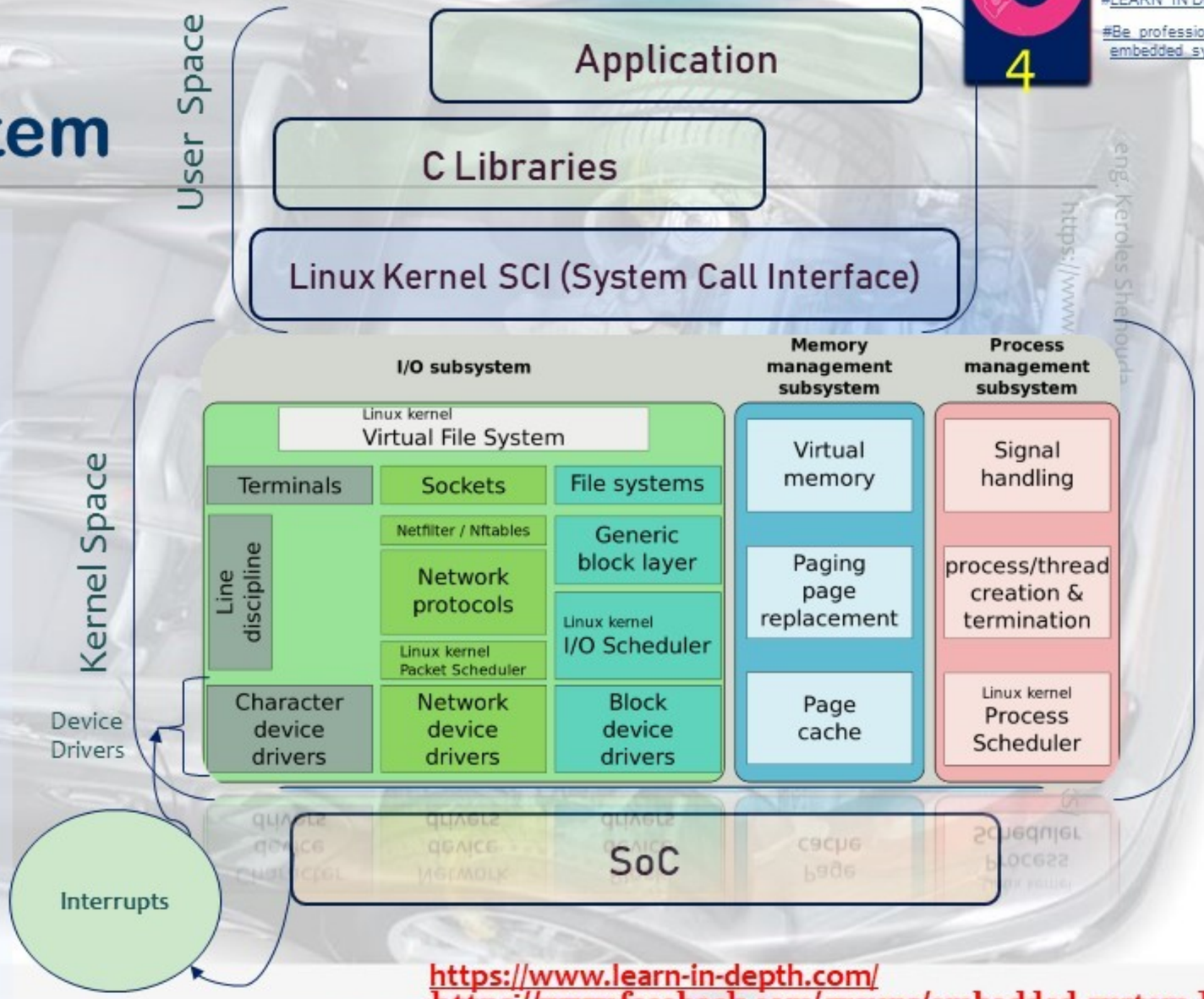
Scalability
Can run on super computers as well as on tiny devices
(4 MB of RAM is enough).

Stability and reliability
Its code is developed by more than 800 Engineers from different companies

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

Linux kernel in the system

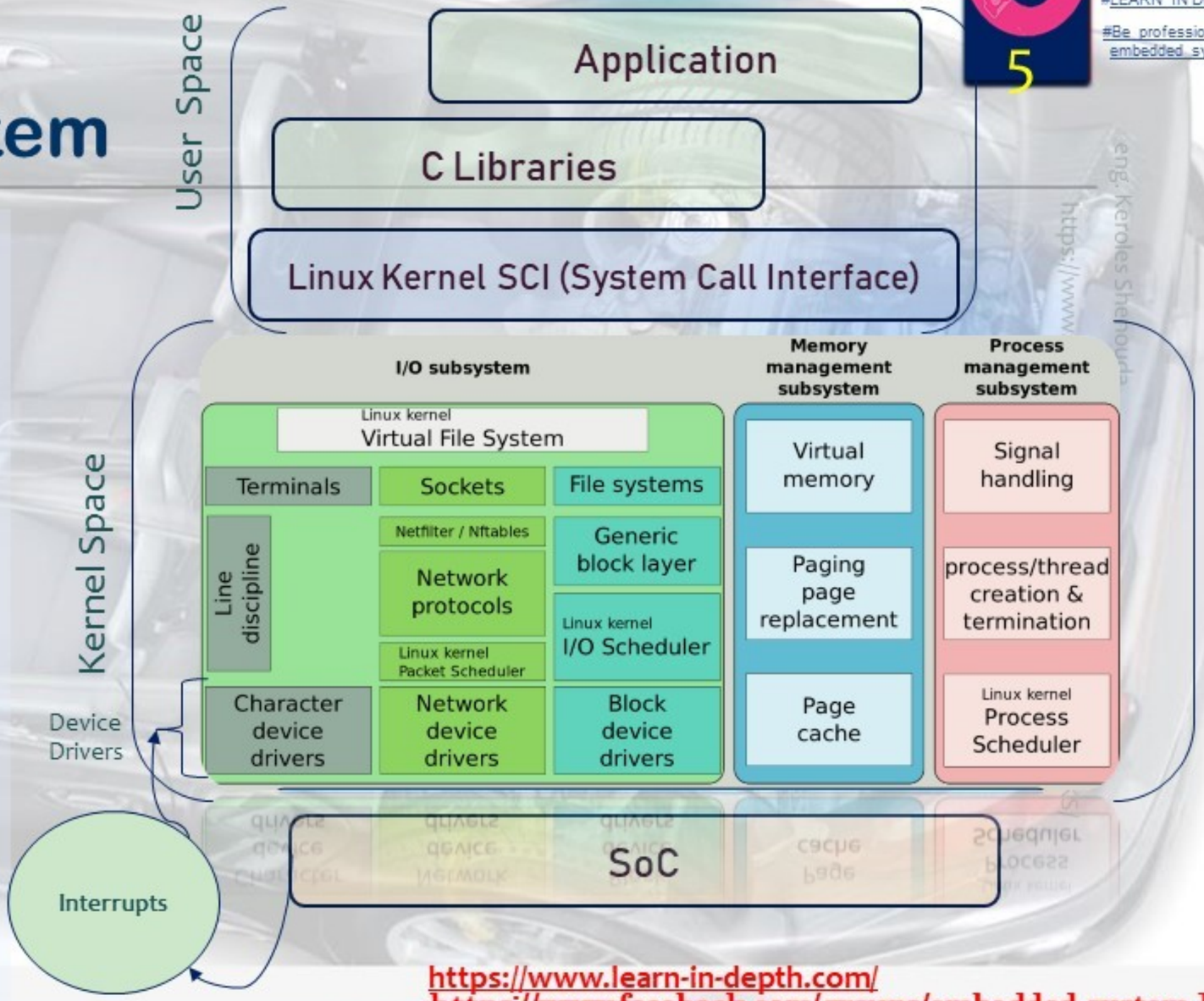
- The kernel has three main jobs:
  - to manage resources
  - to interface with hardware
  - to provide an API that offers a useful level of abstraction to user space programs
- The primary interface between the User space and the Kernel space is the C library, which translates user level functions, such as those defined by **POSIX**, into kernel system calls.
- The system call interface uses an architecture-specific method, such as a trap or a software interrupt, to switch the CPU from low privilege user mode to high privilege kernel mode, which allows access to all memory addresses and CPU registers.



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

Linux kernel in the system

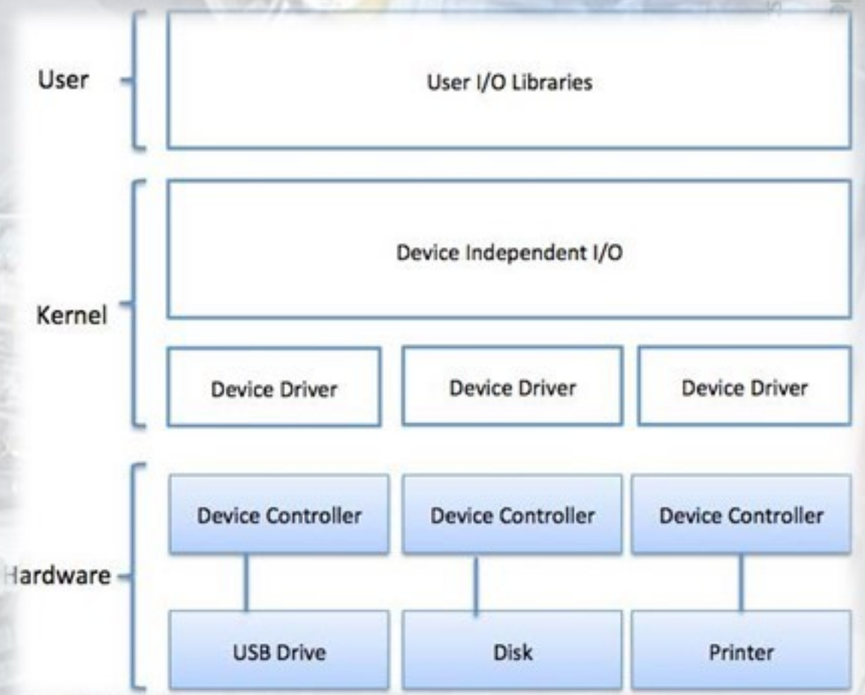
- The System call handler dispatches the call to the suitable kernel subsystem:
  - memory allocation calls go to the memory manager,
  - filesystem calls to the filesystem code, and so on.
- Some of those calls require input from the hardware and will be passed down to a device driver.
- In some cases, the hardware itself invokes a kernel function by raising an interrupt.



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

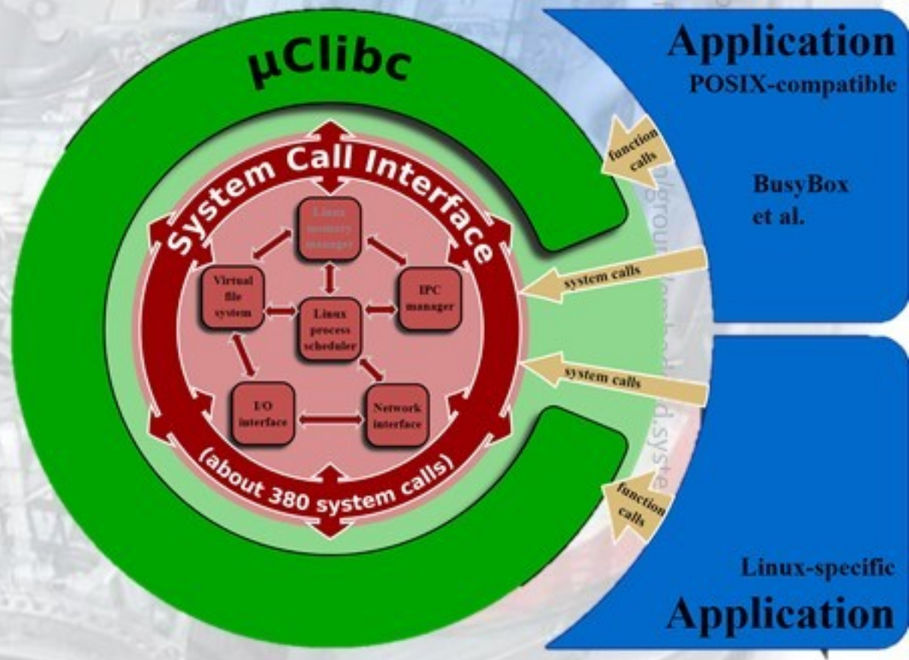
Linux kernel main roles

- **Manage all the hardware resources:** CPU, memory, I/O.
- Provide **a set of portable, architecture and hardware independent APIs** to allow user space applications and libraries to use the hardware resources.
- **Handle concurrent accesses and usage** of hardware resources from different applications.
  - Example: a single network interface is used by multiple user space applications through various network connections. The kernel is responsible to "multiplex" the hardware resource



Linux Kernel SCI (System Call Interface)

- The main interface between the kernel and user space is the set of system calls
  - About 400 system calls that provide the main kernel services
    - File and device operations, networking operations, inter-process communication, process management, memory mapping, timers, threads, synchronization primitives, etc

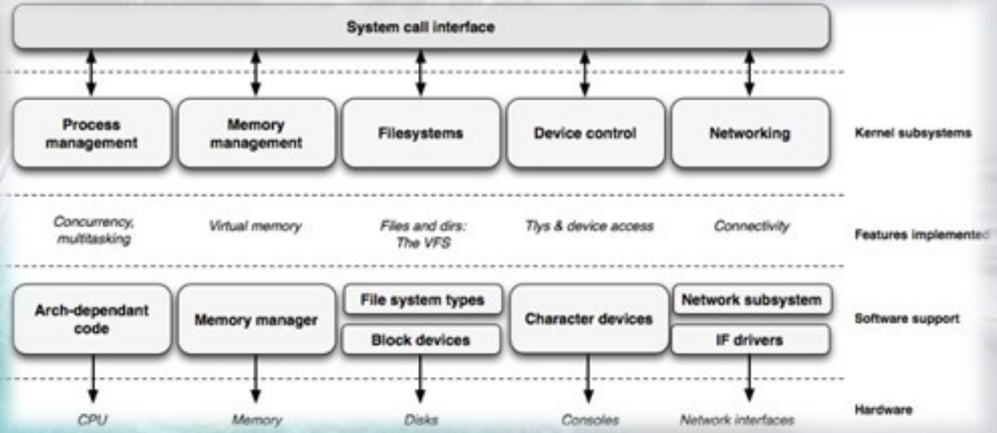


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Pseudo filesystems

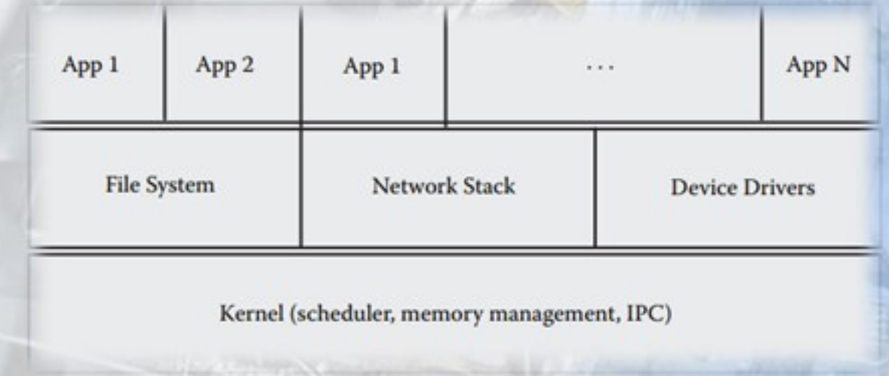
- Linux makes **system** and **kernel information** available in **user space** through pseudo filesystems, sometimes also called virtual filesystems
- Pseudo filesystems allow **applications** to see directories and files that **do not exist** on any real storage,
  - **they are created and updated on the fly by the kernel**
- The two most important pseudo filesystems are
  - **proc**, usually mounted on /proc: Operating system related information (processes, memory management parameters...)
  - **sysfs**, usually mounted on /sys: Representation of the system as a set of devices and buses. Information about these devices.

Linux Vs RTOS (Architecture)



Architecture of Linux kernel

The user space programs operate on a virtual address so that they cannot corrupt another application's or the kernel's memory. However, the kernel components share the same address space; so a badly written driver or module can cause the system to crash.

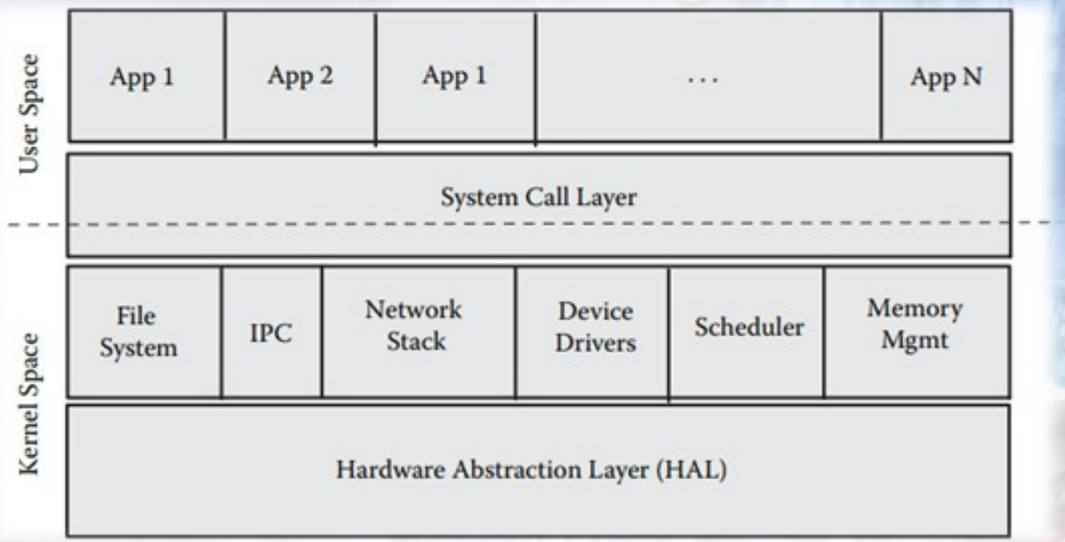


Architecture of traditional RTOS.

The traditional embedded system model was based on having tightly controlled software running on the boards; the cost of memory and storage space further restricted the amount of software that could be run on the system

Inside the Linux kernel

- the basic architecture of the Linux kernel can be split into the following subsystems.
  - The hardware abstraction layer
  - Memory manager
  - Scheduler
  - File system
  - IO subsystem
  - Networking subsystem
  - IPC



Hardware Abstraction Layer (HAL)

- The hardware abstraction layer (HAL) virtualizes the platform hardware so that the different drivers can be ported easily on any hardware.
- The HAL is equivalent to the BSP provided on most of the RTOSs
- The HAL has support for the following hardware components. Processor, cache, and MMU
  - Setting up the memory map
  - Exception and interrupt handling support
  - DMA
  - Timers
  - System console
  - Bus management
  - Power management

Scheduler

- The Linux scheduler provides the **multitasking** capabilities and is evolving over the kernel releases with the aim of providing a deterministic scheduling policy.
- **Kernel thread**: These are processes that do not have a user context. They execute in the kernel space as long as they live.
- **User process**: Each user process has **its own address space** by the **virtual memory**.
  - They enter into the kernel mode when an interrupt, exception, or a system call is executed. Note that when a process enters the kernel mode, it uses a totally different stack. This is referred to as the kernel stack **and each process has its own kernel stack**
- **User thread:** The threads are different execution entities **that are mapped to a single user process.** The user space threads **share a common text**, **data, and heap space**.
  - **They have separate stack addresses**. Other resources such as open files and signal handlers are also shared across the threads.

File System

- On Linux, the various file systems **are managed by** a layer called the **VFS** or the **Virtual File System**.
- The virtual file system provides a consistent view of data as stored on various devices on the system.
- The following are some of the commonly used embedded file systems.
  - EXT2: A classical Linux file system that has a broad user base
  - CRAMFS: A compressed read-only file system
  - ROMFS: A read-only file system
  - RAMFS: A read-write, memory-based file system
  - JFFS2: A journaling file system built specifically for storage on flash
  - PROCFS: A pseudo file system used for getting system information
  - DEVFS: A pseudo file system for maintaining the device files

Will explain it more in The future parts

IO Subsystem

- **The IO subsystem** on Linux provides **a simple** and **uniform interface** to onboard devices.
- Three kinds of devices are supported by the IO subsystem.
  - **Character devices** for supporting sequential devices.
  - **Block devices** for supporting randomly accessible devices.
    - **Block devices** are essential for implementing file systems.
  - **Network devices** that support a variety of link layer devices.

Will explain it on Linux Device Driver Session

Networking Subsystems

- One of the major strengths of Linux has been its robust support for various networking protocols.

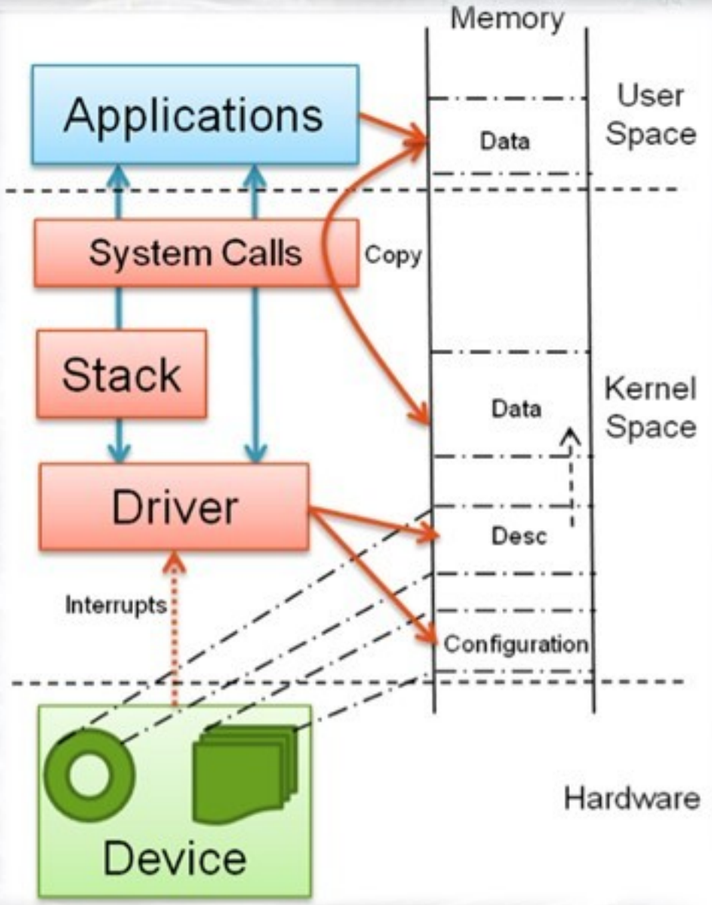
	Kernel Availability		
Feature	*2.2*	*2.4*	*2.6*
Layer 2			
Support for bridging	Yes	Yes	Yes
X.25	Yes	Yes	Yes
LAPB	Experimental	Yes	Yes
PPP	Yes	Yes	Yes
SLIP	Yes	Yes	Yes
Ethernet	Yes	Yes	Yes
ATM	No	Yes	Yes
Bluetooth	No	Yes	Yes
Layer 3			
IPV4	Yes	Yes	Yes
IPV6	No	Yes	Yes
IP forwarding	Yes	Yes	Yes
IP multicasting	Yes	Yes	Yes
IP firewalling	Yes	Yes	Yes
IP tunneling	Yes	Yes	Yes
ICMP	Yes	Yes	Yes
ARP	Yes	Yes	Yes
NAT	Yes	Yes	Yes
IPSEC	No	No	Yes
Layer 4 (and above)			
UDP and TCP	Yes	Yes	Yes
BOOTP/RARP/DHCP	Yes	Yes	Yes

IPC

- The interprocess communication on Linux includes **signals** (for asynchronous communication), **pipes**, and **sockets** as well as the System V IPC mechanisms such as **shared memory**, **message queues**, and **semaphores**. The 2.6 kernel has the additional support for **POSIX-type message queues**.

User Space

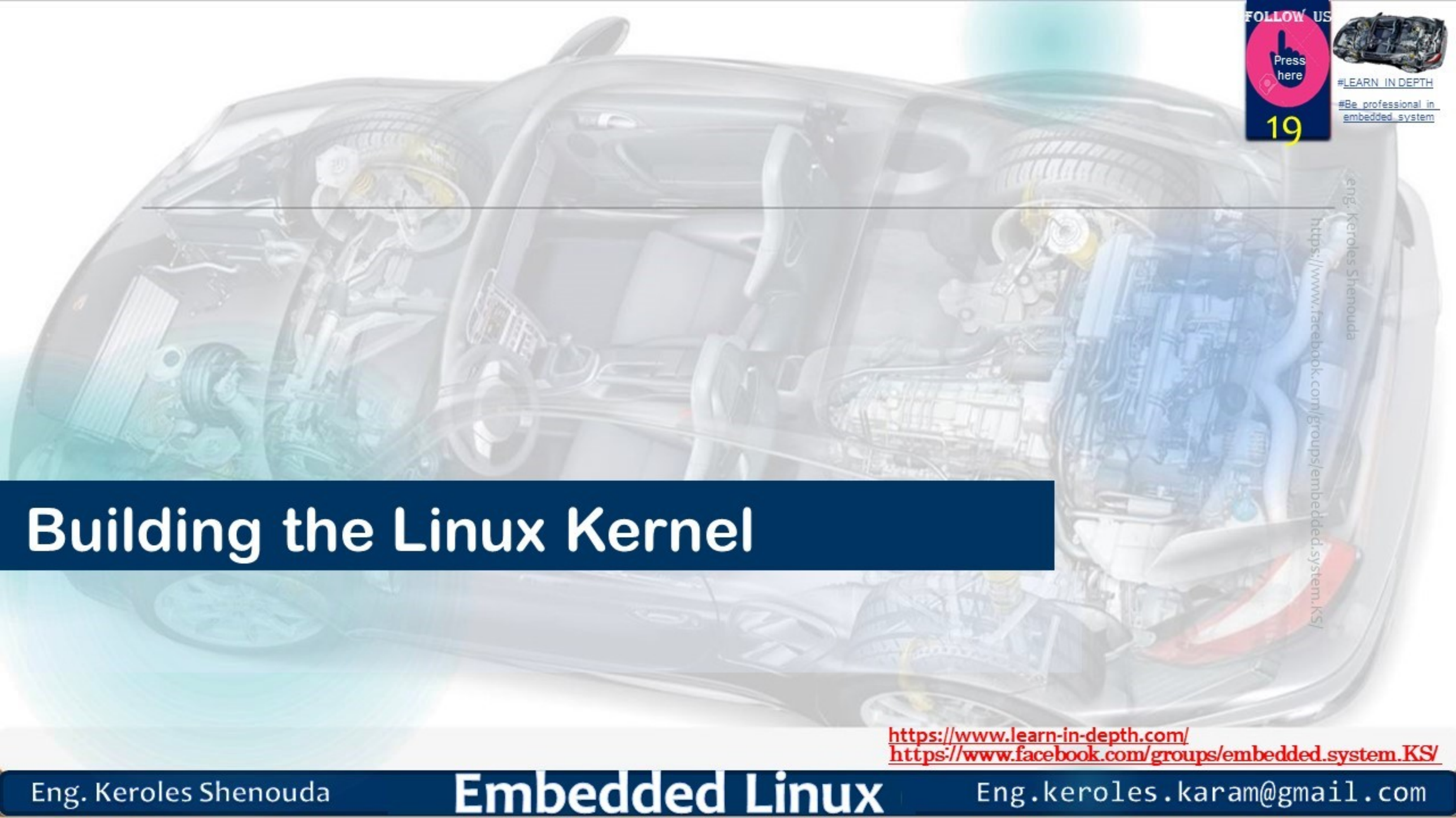
- The user space on Linux is based on the following concepts.
- **Program**: This is the image of an application. It resides on a file system. When an application needs to be run, the **image is loaded into memory and run**. Note that because of **virtual memory** the entire process image is not loaded into memory **but only the required memory pages are loaded**.
- **Virtual memory**: This allows each process to have its own address space. Virtual memory allows for advanced features such as shared libraries. Each process has its own memory map in the virtual address space; this is unique for any process **and is totally independent of the kernel memory map**.
- **System calls**: These are entry points into the kernel so that the kernel can execute services on behalf of the application.



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

Kernel sources

- Go to the Linux kernel web site (https://kernel.org/) and identify the latest stable version.



Building the Linux Kernel

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

The main directories

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ tree -L 1
.
├── arch
├── block
├── certs
├── COPYING
├── CREDITS
├── crypto
├── Documentation
├── drivers
├── fs
├── include
├── init
├── ipc
├── Kbuild
├── Kconfig
├── kernel
├── lib
├── LICENSES
├── MAINTAINERS
├── Makefile
├── mm
├── Module.symvers
├── net
├── README
├── samples
├── scripts
├── security
├── sound
├── tools
├── usr
└── virt

22 directories, 8 files
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```

Contains architecture-specific files. There is one subdirectory per architecture.

Contains kernel documentation. Always look here first if you want to find more information about an aspect of Linux.

Contains device drivers, thousands of them. There is a subdirectory for each type of driver.

Contains filesystem code.

Contains kernel header files, including those required when building the toolchain.

Contains the kernel start-up code.

Contains core functions, including scheduling, locking, timers, power management, and debug/trace code.

Contains memory management.

Contains network protocols

Contains many useful tools and including the Linux performance counters tool

Contains many useful scripts, including the device tree compiler,DTC

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

Kernel configuration

- The kernel contains **thousands** of device drivers, filesystem drivers, network protocols and other configurable items
- **Thousands** of **options are available**, that are used to selectively compile parts of the kernel source code
- The kernel configuration is the process of defining the set of options with which you want your kernel to be compiled
- The set of options depends
  - **On the target architecture and on your hardware** (for device drivers, etc.)
  - **On the capabilities you would like to give to your kernel (network capabilities, filesystems, real-time, etc.)**. Such generic options are available in all architectures.

Specifying the target architecture

- First, specify the architecture for the kernel to build
- Set it to the name of a directory under arch/:
  - export ARCH=arm
- By default, the kernel build system assumes that the kernel is configured and built for the host architecture (x86 in our case, native kernel compiling)
- The kernel build system will use this setting to:
  - Use the configuration options for the target architecture.
  - Compile the kernel with source code and headers for the target architecture.

Kernel configuration and build system

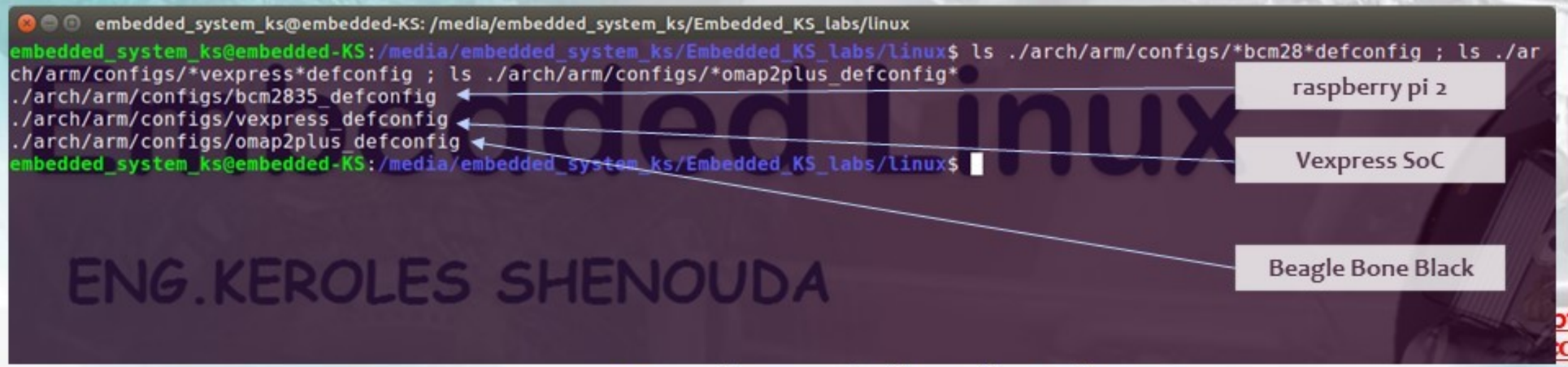
- The kernel configuration and build system is based on multiple Makefiles
- One only interacts with the main Makefile, present at the top directory of the kernel source tree
- Interaction takes place
  - using the make tool, which parses the Makefile
  - through various targets, defining which action should be done (configuration, compilation, installation, etc.). Run make help to see all available targets.

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
22 directories, 8 files
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make help
Cleaning targets:
  clean           - Remove most generated files but keep the config and
                    enough build support to build external modules
  mrproper        - Remove all generated files + config + various backup files
  distclean       - mrproper + remove editor backup and patch files

Configuration targets:
  config          - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based program
  menuconfig      - Update current config utilising a menu based program
  xconfig         - Update current config utilising a Qt based front-end
  gconfig         - Update current config utilising a GTK+ based front-end
  oldconfig       - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  defconfig       - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig     - New config where all options are answered with no
  allyesconfig    - New config where all options are accepted with yes
  allmodconfig    - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig      - New config with random answer to all options
  listnewconfig   - List new options
  olddefconfig    - Same as oldconfig but sets new symbols to their
                    default value without prompting
  kvmconfig       - Enable additional options for kvm guest kernel support
  xenconfig       - Enable additional options for xen dom0 and guest kernel support
  tinyconfig      - Configure the tiniest possible kernel
  testconfig      - Run Kconfig unit tests (requires python3 and pytest)
```

```
Other generic targets:
  all             - Build all targets marked with [*]
* vmlinux         - Build the bare kernel
* modules         - Build all modules
  modules_install - Install all modules to INSTALL_MOD_PATH (default: /)
  dir/            - Build all files in dir and below
  dir/file.[ois]  - Build specified target only
  dir/file.ll     - Build the LLVM assembly file
                    (requires compiler support for LLVM assembly generation)
  dir/file.lst    - Build specified mixed source/assembly target only
                    (requires a recent binutils and recent build (System.map))
  dir/file.ko     - Build module including final link
  modules_prepare - Set up for building external modules
  tags/TAGS       - Generate tags file for editors
  cscope          - Generate cscope index
  gtags           - Generate GNU GLOBAL index
  kernelrelease   - Output the release version string (use with make -s)
  kernelversion   - Output the version stored in Makefile (use with make -s)
  image_name      - Output the image name (use with make -s)
  headers_install - Install sanitised kernel headers to INSTALL_HDR_PATH
                    (default: ./usr)
```

Kernel configuration details

- The configuration is stored in the .config file at the root of kernel sources
  - Simple text file, CONFIG_PARAM=value (included by the kernel Makefile)
- As options have dependencies, typically never edited by hand, but through graphical or text interfaces:
  - make xconfig, make gconfig (graphical)
  - make menuconfig, make nconfig (text)
- Default configuration files are available, usually for each CPU family.
  - They are stored in **arch/configs/**, and are just minimal .config files (only settings different from default ones).



Create your own default configuration

- To create your own default configuration file:
  - **make savedefconfig** This creates a minimal configuration (non-default settings)
  - mv defconfig arch/configs/myown_defconfig This way, you can share a reference configuration inside the kernel sources.

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$  make savedefconfig
scripts/kconfig/conf  --savedefconfig=defconfig Kconfig
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ ls defconfig
defconfig
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```

Kernel option types

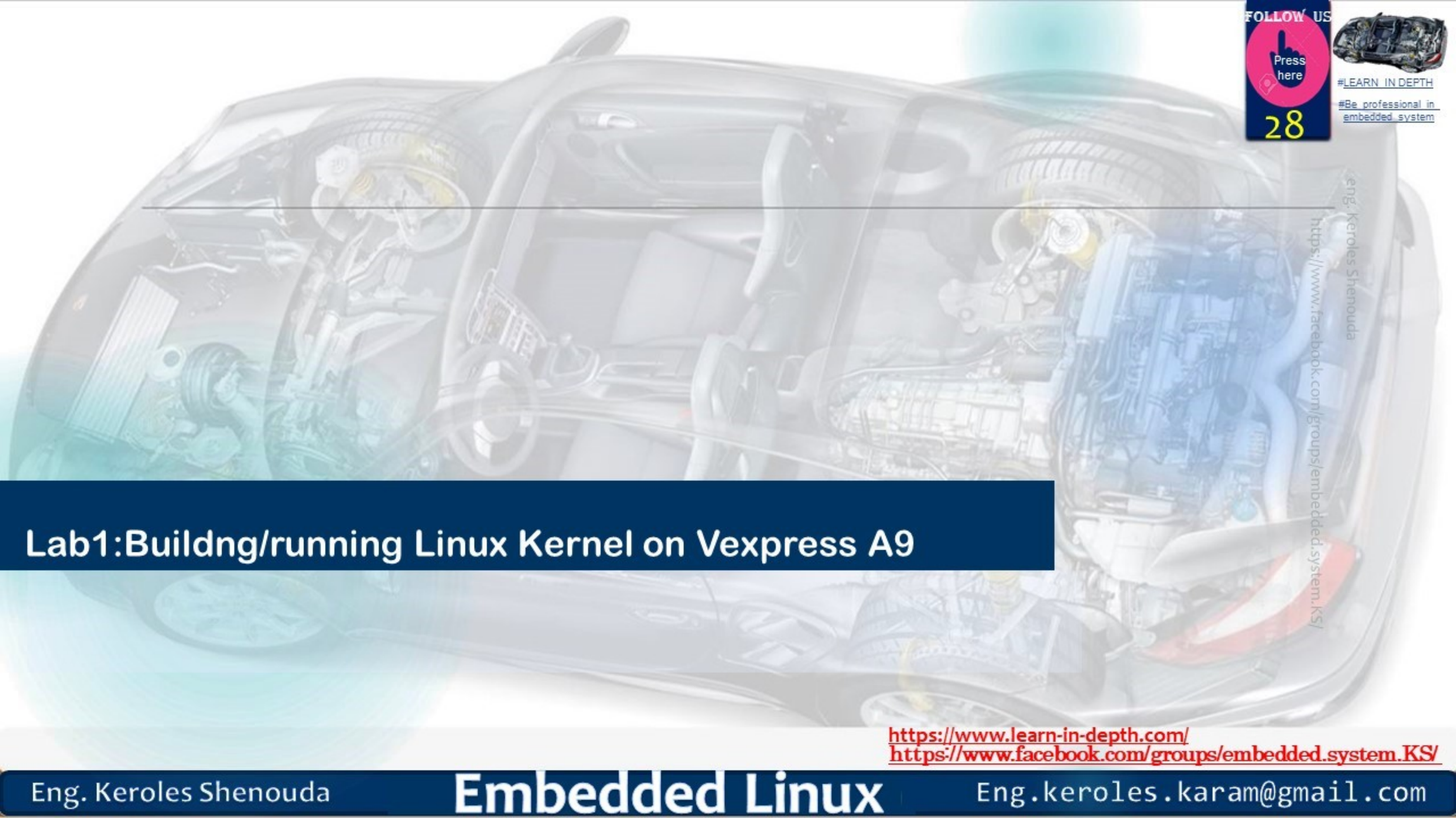
- There are different types of options
  - **bool** options, they are either
    - true (to include the feature in the kernel) or
    - false (to exclude the feature from the kernel)
  - **tristate** options, they are either
    - true (to include the feature in the kernel image) or
    - module (to include the feature as a kernel module) or
    - false (to exclude the feature)
  - **int** options, to specify integer values
  - **hex** options, to specify hexadecimal values
  - **string** options, to specify string values

Kernel option dependencies

- There are dependencies between kernel options
- For example, enabling a network driver requires the network stack to be enabled

```
Option
  [x] Network File Systems
  [ ] Timestamping in PHY devices
  [ ] Security Marking
      Socket and Networking Security Hooks
      XFRM (IPSec) Networking Security Hooks

Network File Systems (NETWORK_FILESYSTEMS)

CONFIG_NETWORK_FILESYSTEMS:

Say Y here to get to see options for network filesystems and
filesystem-related networking code, such as NFS daemon and
RPCSEC security modules.

This option alone does not add any kernel code.

If you say N, all options in this submenu will be skipped and
disabled; if unsure, say Y here.

Symbol: NETWORK_FILESYSTEMS [=y]
Type : bool
Prompt: Network File Systems
Location:
-> File systems
Defined at fs/Kconfig:270
Depends on: NET [=y]
```

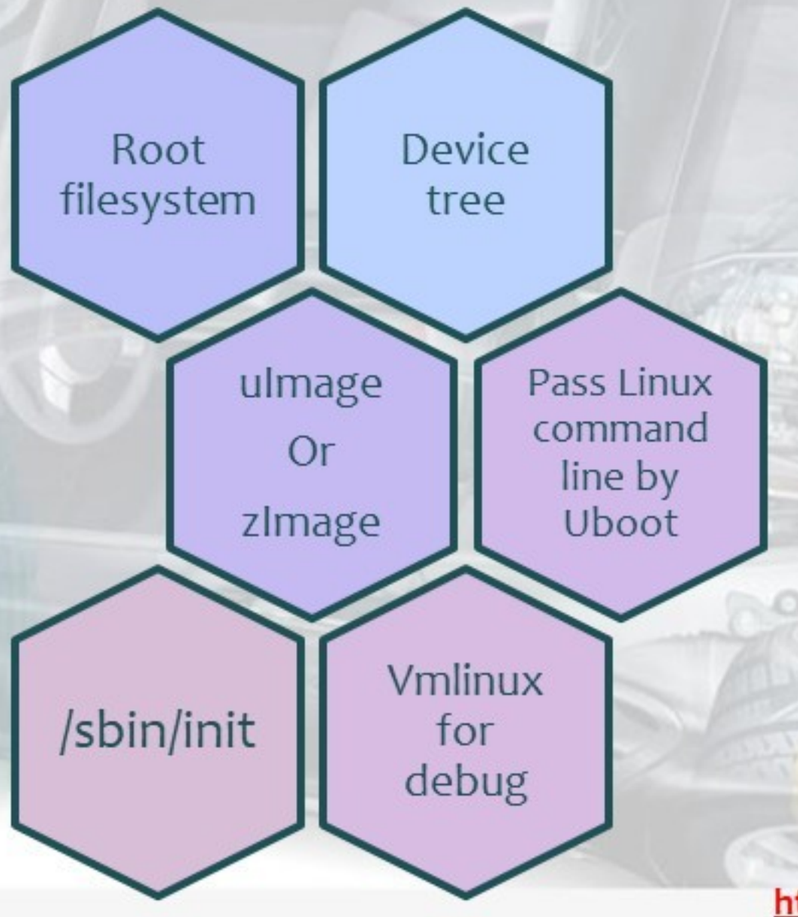

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Lab1:Buildng/running Linux Kernel on Vexpress A9

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

You should to have



Configure Linux kernel

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ export CROSS_COMPILE=arm-none-eabi-
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ export ARCH=arm
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make vexpress_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTLD  scripts/kconfig/conf
#
# No change to .config
#
```

Generate the Device tree

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make dtbs
scripts/kconfig/conf  --syncconfig Kconfig
  UPD     include/config/kernel.release
  HOSTCC  scripts/dtc/dtc.o
  HOSTCC  scripts/dtc/flattree.o
  HOSTCC  scripts/dtc/fstree.o
  HOSTCC  scripts/dtc/data.o
  HOSTCC  scripts/dtc/livetree.o
  HOSTCC  scripts/dtc/treesource.o
  HOSTCC  scripts/dtc/srcpos.o
  HOSTCC  scripts/dtc/checks.o
  HOSTCC  scripts/dtc/util.o
  LEX     scripts/dtc/dtc-lexer.lex.c
  YACC    scripts/dtc/dtc-parser.tab.[ch]
  HOSTCC  scripts/dtc/dtc-lexer.lex.o
  HOSTCC  scripts/dtc/dtc-parser.tab.o
  HOSTLD  scripts/dtc/dtc
  DTC     arch/arm/boot/dts/vexpress-v2p-ca5s.dtb
  DTC     arch/arm/boot/dts/vexpress-v2p-ca9.dtb
  DTC     arch/arm/boot/dts/vexpress-v2p-ca15-tc1.dtb
  DTC     arch/arm/boot/dts/vexpress-v2p-ca15_a7.dtb
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```

Generate the zImage

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make zImage -j 16
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  CC      arch/arm/mm/extable.o
  CC      arch/arm/mm/iomap.o
  CC      arch/arm/mm/fault.o
  CC      arch/arm/mm/init.o
  AS      arch/arm/common/secure_cntvoff.o
  AS      arch/arm/common/mcpm_head.o
  CC      arch/arm/mm/dma-mapping.o
  CC      arch/arm/common/firmware.o
  CC      arch/arm/mm/mmap.o
  CC      arch/arm/mm/fault-armv.o
  CC      arch/arm/mm/flush.o
  CC      arch/arm/mm/idmap.o
  CC      arch/arm/mm/ioremap.o
  CC      arch/arm/probes/uprobes/core.o
  CC      arch/arm/mm/pgd.o
  CC      arch/arm/probes/uprobes/actions-arm.o
  CC      arch/arm/mm/mmu.o
  CC      arch/arm/kernel/atags_parse.o
  AS      arch/arm/kernel/entry-armv.o
  CC      arch/arm/mm/pageattr.o
  CC      arch/arm/common/mcpm_entry.o
  AR      arch/arm/net/built-in.a
  CC      arch/arm/common/mcpm_platsmp.o
  AS      arch/arm/common/vlock.o
  CC      arch/arm/mm/proc-syms.o
  CC      arch/arm/probes/decode.o
  CC      arch/arm/kernel/bugs.o
  CC      arch/arm/mm/alignment.o
  AR      arch/arm/probes/uprobes/built-in.a
  CC      arch/arm/probes/decode-arm.o
  AS      arch/arm/mm/abort-ev7.o
  AS      arch/arm/mm/cache-v7.o
  AS      arch/arm/mm/pabort-v7.o
  CC      arch/arm/mm/copypage-v6.o
  CC      arch/arm/mm/context.o
  AS      arch/arm/mm/tlb-v7.o
```

```
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/bswapsdi2.o
  AS      arch/arm/boot/compressed/piggy.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
```

You can also use make all

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make all  -j 16
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  Kernel: arch/arm/boot/Image is ready
  Building modules, stage 2.
  MODPOST 0 modules
  Kernel: arch/arm/boot/zImage is ready
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```

What is the difference between **Image vs zImage vs uImage** ?

Image: the generic Linux kernel binary image file.
zImage: a compressed version of the Linux kernel image that is self-extracting.
uImage: an image file that has a U-Boot wrapper (installed by the **mkimage** utility) that includes the OS type and loader information.

A very common practice (e.g. the typical Linux kernel Makefile) is to use **a zImage file**. Since a zImage file is self-extracting (i.e. needs no external decompressors

Linux Low-Level Architecture Objects

- vmlinux
  - Kernel proper, in ELF format, including symbols, comments, debug info (if compiled with -g)
- System.map
  - Text-based kernel symbol table for vmlinux module.

```
Device          Boot Start    End Sectors Size Id Type
KS_SD_70M.img1 *      2048  26623   24576  12M  c W95 FAT32 (LBA)
KS_SD_70M.img2       26624 143359  116736  57M 83 Linux
```

Prepare the SDCARD

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_Labs/lab1$ ./lab1.sh u-boot KS_SD_30M.img
qemu-system-arm: -redir tcp:5555::22: The -redir option is deprecated. Please use '-netdev user,hostfwd=...' instead.
WARNING: Image format was not specified for 'KS_SD_30M.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.


U-Boot 2020.01-rc2-00035-g3ff1ff3ff7-dirty (Nov 21 2019 - 16:47:34 +0200)

DRAM:  512 MiB
WARNING: Caches not enabled
Flash: 128 MiB
MMC:   MMC: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   smc911x-0
Hit any key to stop autoboot:  0
=> setenv bootargs  'rw earlyprintk loglevel=8 root=/dev/mmcblk0p2  console=ttyAMA0 dhcp'
=> fatload mmc 0:1 0x60000000 zImage
4594264 bytes read in 2707 ms (1.6 MiB/s)
=> fatload mmc  0:1 0x63000000 vexpress-v2p-ca9.dtb
14143 bytes read in 36 ms (382.8 KiB/s)
=> bootz 0x60000000 - 0x63000000
```

Set the linux Command Line

Load the zImage

Load the device tree

Boot the Linux kernel

P1 (Fat32)
vexpress-v2p-ca9.dtb
zImage

P2 (ext4)
Root FileSystem

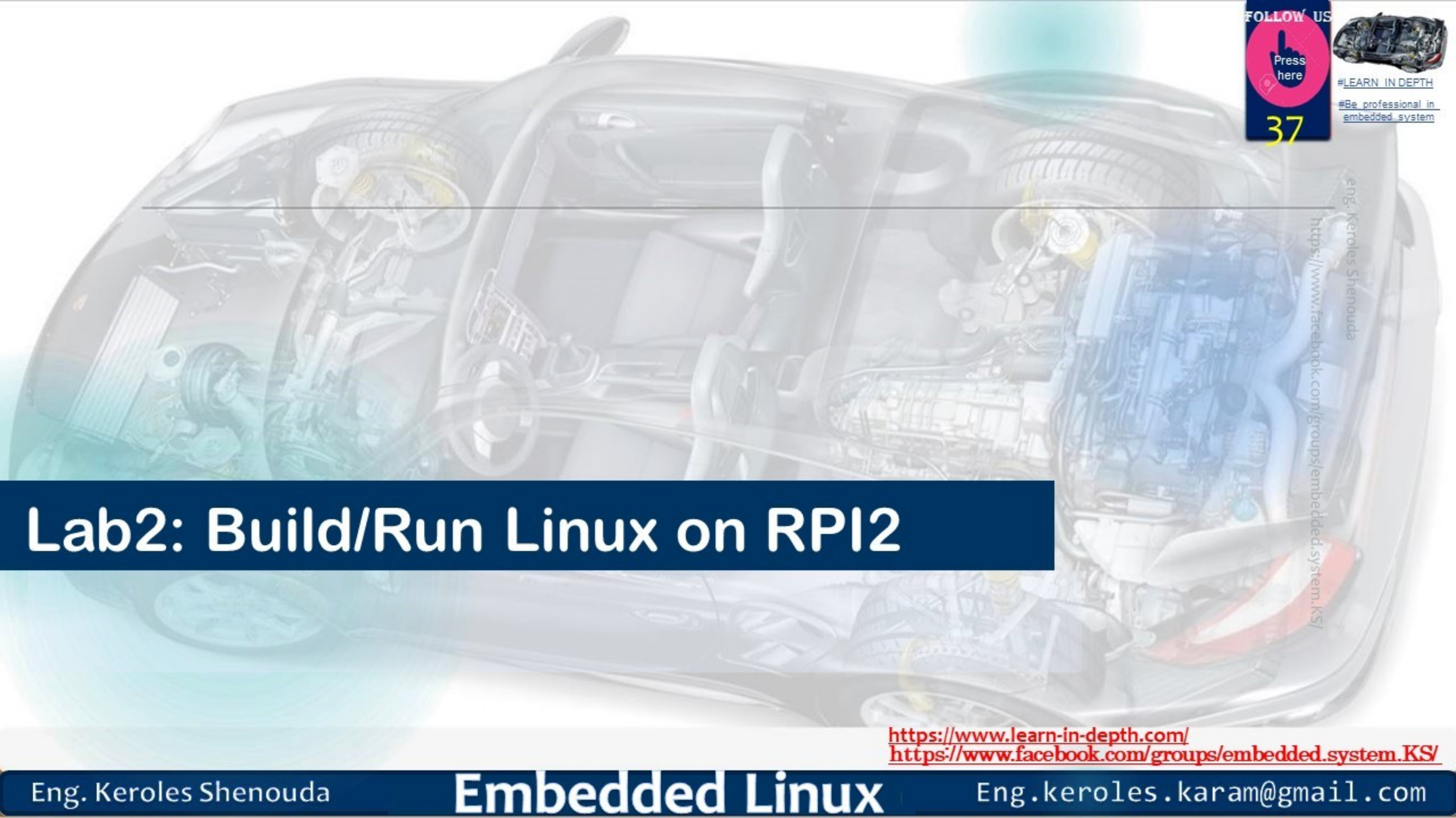
SDCARD

The linux is reached to the login

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_Labs/lab1
embedded_system_ks@embedded-KS: /media/e...  embedded_system_ks@embedded-KS: /media/e...  embedded_system_ks@embedded-KS: /media/e...

U-Boot 2020.01-rc2-00035-g3ff1ff3ff7-dirty (Nov 21 2019 - 16:47:34 +0200)

DRAM:  512 MiB
WARNING: Caches not enabled
Flash: 128 MiB
MMC:   MMC: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   smc911x-0
Hit any key to stop autoboot:  0
=> setenv bootargs  'rw earlyprintk loglevel=8 root=/dev/mmcblk0p2  console=ttyAMA0 dhcp'
=> fatload mmc 0:1 0x60000000 zImage
4594264 bytes read in 2421 ms (1.8 MiB/s)
=> fatload mmc  0:1 0x63000000 vexpress-v2p-ca9.dtb
14143 bytes read in 34 ms (405.3 KiB/s)
=> bootz 0x60000000 - 0x63000000
Kernel image @ 0x60000000 [ 0x000000 - 0x461a58 ]
## Flattened Device Tree blob at 63000000
   Booting using the fdt blob at 0x63000000
   Loading Device Tree to 7fe80000, end 7fe8673e ... OK

Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 5.4.0-rc8+ (embedded_system_ks@embedded-KS) (gcc version 5.4.1 20160919 (15:5.4.1+svn241155-1)) #4 SMP Fr
eb 14 22:49:50 EET 2020
CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
OF: fdt: Machine model: V2P-CA9
OF: fdt: Ignoring memory block 0x80000000 - 0x80000004
Memory policy: Data cache writeback
Reserved memory: created DMA memory pool at 0x4c000000, size 8 MiB
OF: reserved mem: initialized node vram@4c000000, compatible id shared-dma-pool
cma: Reserved 16 MiB at 0x7ec00000
On node 0 totalpages: 131072
```

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_Labs/lab1
embedded_system_ks@embedded-KS: /media/e...  embedded_system_ks@embedded-KS: /media/e...  embedded_system_ks@embedded-KS: /media/e...

input: AT Raw Set 2 keyboard as /devices/platform/smb@4000000/smb@4000000:motherboard/smb@4000000:motherboard:iofpga@7,
00000/10006000.kmi/serio0/input/input0
mmc0: new SD card at address 4567
mmcblk0: mmc0:4567 QEMU! 70.0 MiB
aaci-pl041 10004000.aaci: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 20
aaci-pl041 10004000.aaci: FIFO 512 entries
oprofile: using arm/armv7-ca9
NET: Registered protocol family 17
9pnet: Installing 9P2000 support
Registering SWP/SWPB emulation handler
 mmcblk0: p1 p2
Error: Driver 'vexpress-muxfpga' is already registered, aborting...
drm-clcd-pl111 10020000.clcd: initializing Versatile Express PL111
drm-clcd-pl111 10020000.clcd: DVI muxed to daughterboard 1 (core tile) CLCD
Error: Driver 'vexpress-muxfpga' is already registered, aborting...
drm-clcd-pl111 10020000.clcd: initializing Versatile Express PL111
drm-clcd-pl111 10020000.clcd: DVI muxed to daughterboard 1 (core tile) CLCD
rtc-pl031 10017000.rtc: setting system clock to 2020-02-22T01:48:51 UTC (1582336131)
ALSA device list:
  #0: ARM AC'97 Interface PL041 rev0 at 0x10004000, irq 20
input: ImExPS/2 Generic Explorer Mouse as /devices/platform/smb@4000000/smb@4000000:motherboard/smb@4000000:motherboard
fpga@7,00000000/10007000.kmi/serio1/input/input2
Error: Driver 'vexpress-muxfpga' is already registered, aborting...
drm-clcd-pl111 10020000.clcd: initializing Versatile Express PL111
drm-clcd-pl111 10020000.clcd: DVI muxed to daughterboard 1 (core tile) CLCD
EXT4-fs (mmcblk0p2): mounting ext3 file system using the ext4 subsystem
random: fast init done
EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
ext3 filesystem being mounted at /root supports timestamps until 2038 (0x7fffffff)
VFS: Mounted root (ext3 filesystem) on device 179:2.
Freeing unused kernel memory: 1024K
Run /sbin/init as init process
random: crng init done
/etc/init.d/rcS: line 14: can't create /proc/sys/kernel/hotplug: nonexistent directory

Please press Enter to activate this console.
[root@vexpress ]#
[root@vexpress ]# uname -a
Linux vexpress 5.4.0-rc8+ #4 SMP Fri Feb 14 22:49:50 EET 2020 armv7l GNU/Linux
[root@vexpress ]#
```


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



Lab2: Build/Run Linux on RPI2

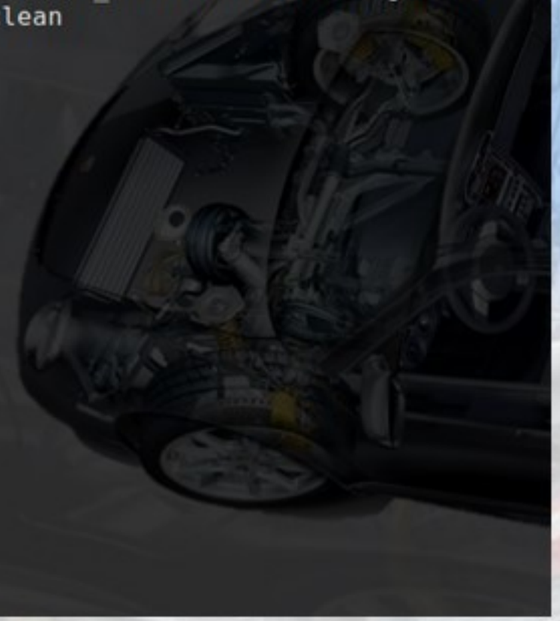
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

Cross-compiling environment setup

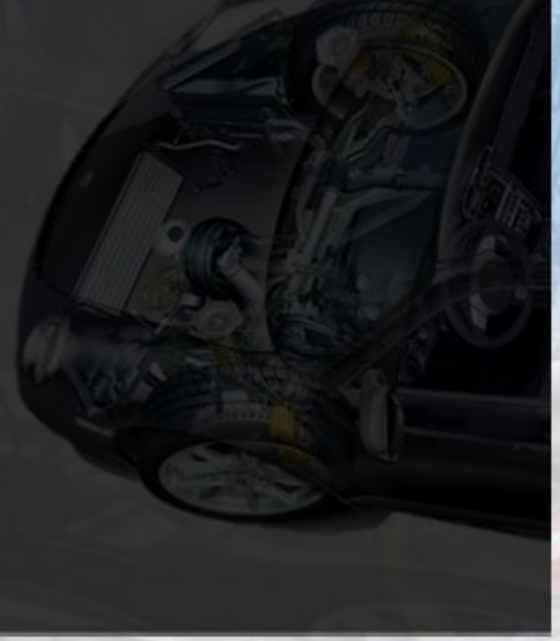
```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ export ARCH=arm
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ export CROSS_COMPILE=arm-linux-gnueabi-
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make clean
  CLEAN   arch/arm/kernel
  CLEAN   arch/arm/vdso
  CLEAN   drivers/scsi
  CLEAN   drivers/tty/vt
  CLEAN   drivers/video/logo
  CLEAN   kernel
  CLEAN   lib
  CLEAN   usr
  CLEAN   arch/arm/boot/compressed
  CLEAN   arch/arm/boot
  CLEAN   modules.builtin.modinfo
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```



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

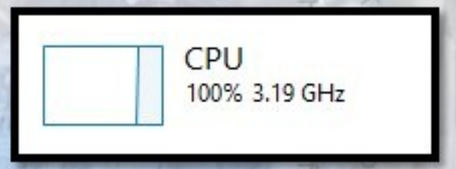
Linux kernel configuration

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/linux
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make bcm2835_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```



Build Linux Kernel

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$ make all -j 16
scripts/kconfig/conf  --syncconfig Kconfig
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-eabi.h
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-common.h
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-oabi.h
  HOSTCC  scripts/dtc/dtc.o
  HOSTCC  scripts/dtc/data.o
  HOSTCC  scripts/dtc/fstree.o
  HOSTCC  scripts/dtc/treesource.o
  HOSTCC  scripts/dtc/livetree.o
  HOSTCC  scripts/dtc/checks.o
  HOSTCC  scripts/dtc/util.o
  HOSTCC  scripts/dtc/flattree.o
  HOSTCC  scripts/dtc/srcpos.o
  LEX     scripts/dtc/dtc-lexer.lex.c
  YACC    scripts/dtc/dtc-parser.tab.[ch]
  HOSTCC  scripts/dtc/dtc-lexer.lex.o
  HOSTCC  scripts/dtc/dtc-parser.tab.o
  HOSTLD  scripts/dtc/dtc
  HOSTCC  scripts/kallsyms
```



Take care
-j 16
Will force CPU to run 100% utilization ☺

```
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/bswapsdi2.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_KS_labs/linux$
```

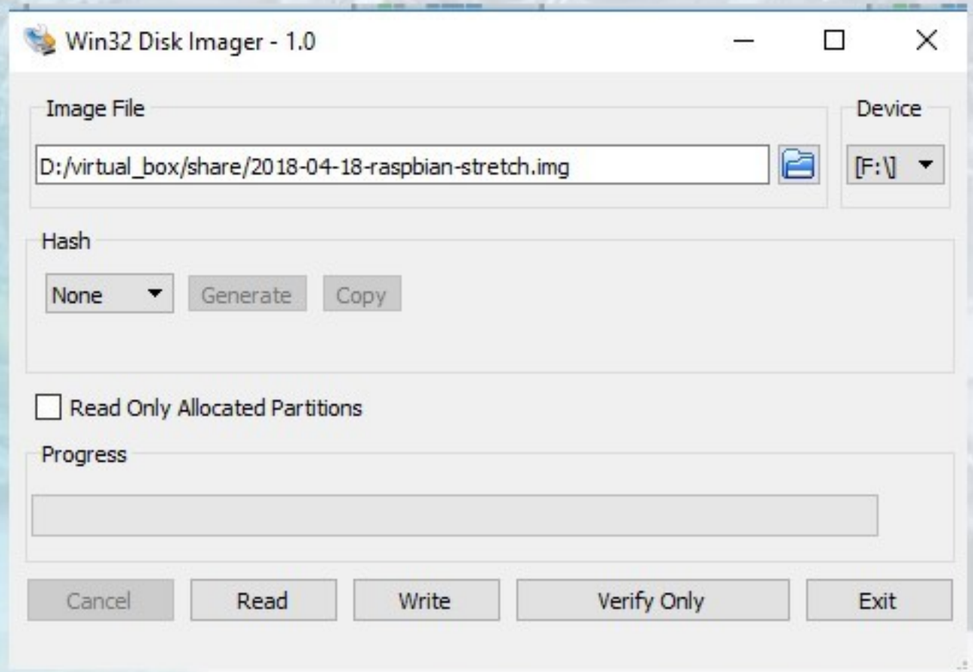
arch/arm/boot/zImage >> Linux Kernel image
arch/arm/boot/dts/bcm2836-rpi-2-b.dtb >> device tree

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

Prepare the SDCARD

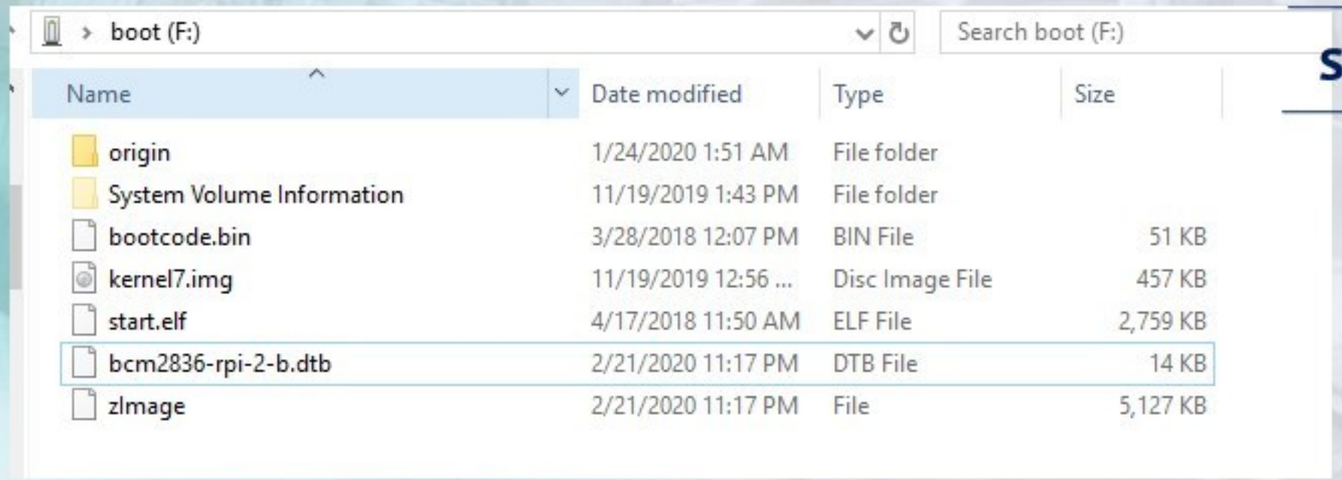
- For now, we didn't build the RootFS, so we will use the prebuilt sdcard

2018-04-18-raspbian-stretch.img then will delete every thing in boot partition (FAT32) and put Our images.



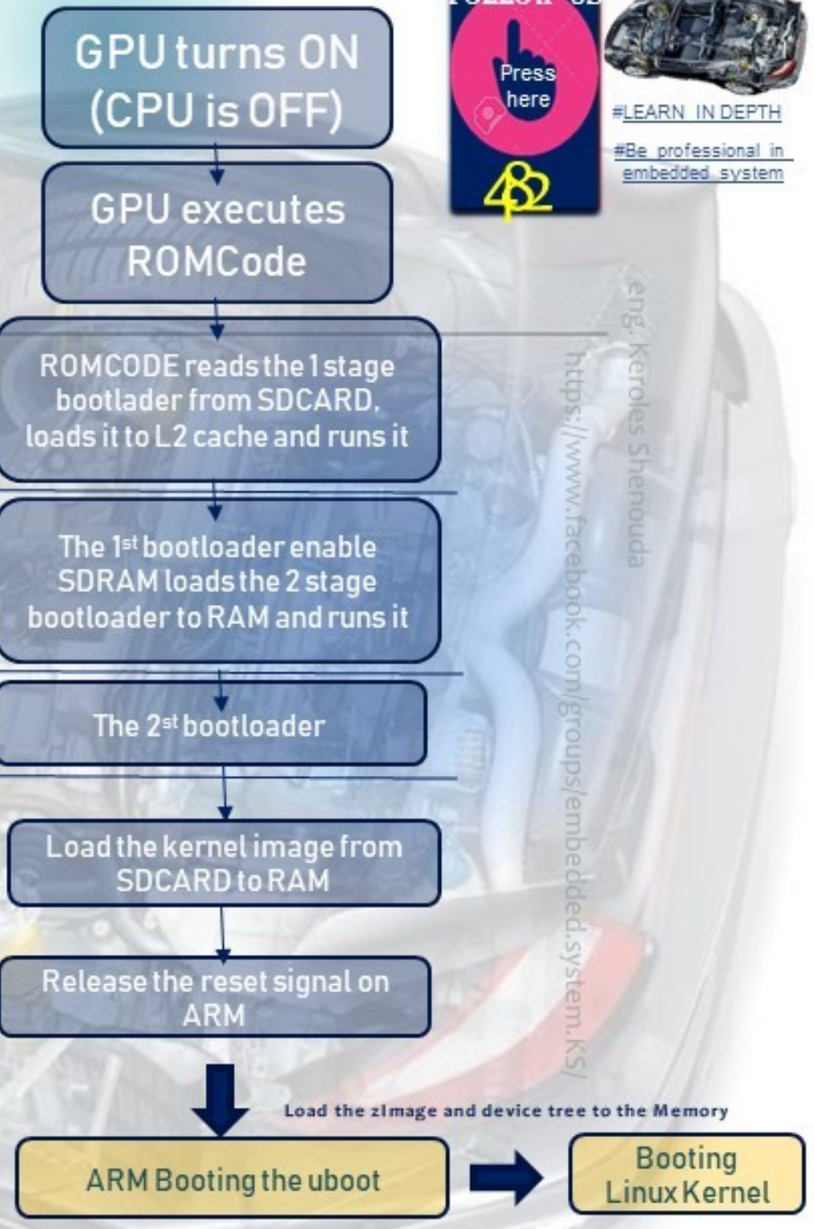
Prepare the SDCARD Cont.

- Copy the uboot, zImage (linux_kernel) and GPU binary/elf files to the FAT partition.
  - Kindly Note the uboot.bin should be renamed to **bootcode.bin**



start.elf

Kernel7.img = Uboot.bin



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

Load and boot the kernel using U-Boot

```
keroles@karas:~$ sudo picocom  -b 115200 /dev/ttyUSB0
[sudo] password for keroles:
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
```

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

Load and boot the kernel using U-Boot

```
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> printenv kernel_addr_r
kernel_addr_r=0x00080000
U-Boot> printenv
  arch board_name board_rev_scheme boot_a_script boot_prefixes boot_scripts
  bootcmd_mmc1 dhcpuboot ethaddr fdt_addr_r fdtcontroladdr initrd_high
  preboot pxefile_addr_r scan_dev_for_efi scriptaddr serial# usbethaddr ...
U-Boot> printenv fdt_addr_r
fdt_addr_r=0x02600000
U-Boot> printenv bootcmd_mmc1
bootcmd_mmc1=devnum=1; run mmc_boot
U-Boot> printenv mmc_boot
mmc_boot=if mmc dev ${devnum}; then devtype=mmc; run scan_dev_for_boot_part; fi
U-Boot> fat
  fatinfo fatload fatls fatmkdir fatrm fatsize fatwrite
U-Boot> fatls mmc 0:1
            System Volume Information/
            origin/
    52064   bootcode.bin
  2825124   start.elf
   467140   kernel7.img
    13507   bcm2836-rpi-2-b.dtb
  5249664   zImage

5 file(s), 2 dir(s)

U-Boot>
```

Kernel address

Device tree address

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

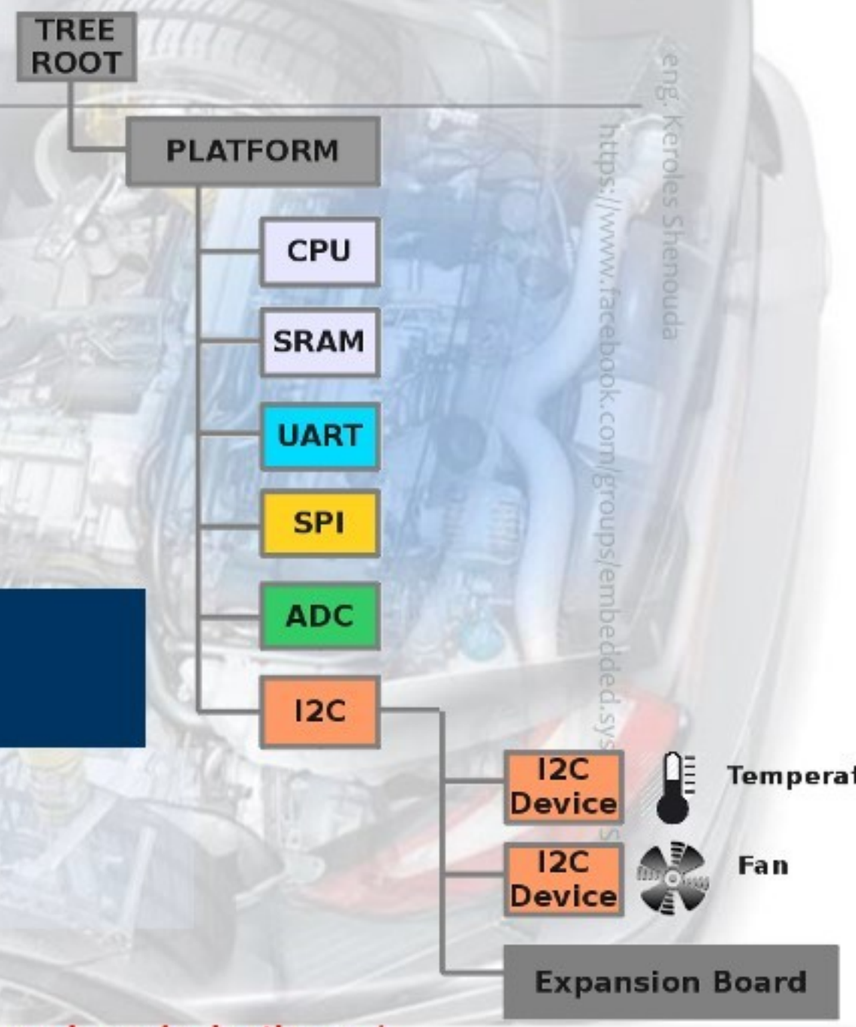
Load and boot the kernel using U-Boot

```
U-Boot> ext4ls mmc 0:2
<DIR>       4096 .
<DIR>       4096 ..
<DIR>       4096 bin
<DIR>       4096 boot
<DIR>       4096 dev
<DIR>       4096 etc
<DIR>       4096 home
<DIR>       4096 lib
<DIR>       4096 lost+found
<DIR>       4096 media
<DIR>       4096 mnt
<DIR>       4096 opt
<DIR>       4096 proc
<DIR>       4096 root
<DIR>       4096 run
<DIR>       4096 sbin
<DIR>       4096 srv
<DIR>       4096 sys
<DIR>       4096 tmp
<DIR>       4096 usr
<DIR>       4096 var
U-Boot> sete
  setenv setexpr
U-Boot> setenv bootargs 'rw earlyprintk loglevel=8 root=/dev/mmcblk02 console=ttyAMA0 dhcp'
U-Boot> fatload mmc 0:1 $kernel_addr_r zImage
5249664 bytes read in 364 ms (13.8 MiB/s)
U-Boot> fatload mmc 0:1 $fdt_addr_r bcm2836-rpi-2-b.dtb
13507 bytes read in 10 ms (1.3 MiB/s)
U-Boot> bootz $kernel_addr_r - $fdt_addr_r
```

- bootargs
- Linux kernel
- Device tree

```
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 5.4.0-rc8+ (embedded_system_ks@embedded-KS) (gcc version 6.3.0 20170406 (Ubuntu/Linaro 6.3.0-
12ubuntu2)) #6 Fri Feb 21 23:06:51 EET 2020
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c53c7d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] OF: fdt: Machine model: Raspberry Pi 2 Model B
[    0.000000] Memory policy: Data cache writeback
[    0.000000] cma: Reserved 32 MiB at 0x06000000
[    0.000000] On node 0 totalpages: 32768
[    0.000000]   Normal zone: 256 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 32768 pages, LIFO batch:7
```

Device trees



Device tree

- Is Open Firmware specification, IEEE-1275-1994
  - Description of hardware (contains no code)
- The kernel no longer contains the description of the hardware,
  - it is located in a separate binary: the device tree DTB
- The **kernel** needs to know **details about hardware**
  - to decide which **drivers** to initialize
  - to configure device parameters such as **register addresses** and **IRQ**
- The **bootloader** loads two binaries: **the kernel image** and **the DTB**
  - Kernel image remains uImage or zImage
  - DTB located in arch/arm/boot/dts, one per board
- U-Boot command:
  - boot[mz] <kernel img addr> - <dtb addr>

Device Tree

- Is Tree of nodes, containing attributes, for example:
  - ***.dts** source files are in arch/<ARCH>/boot/dts
  - compiled to ***.dtb**, using dtc
- dtb file is loaded into memory by the bootloader
- The U-boot bootz command takes three arguments
  - bootz <kernel> <ramdisk> <dt binary>
    - If there is no ramdisk (which is common)
      - bootz <kernel> - <dt binary>
- The dtb is also known as a **Device Tree Blob**, or a **Flattened Device Tree** (fdt)

```
/dts-v1/;
/{
    model = "TI AM335x BeagleBone";
    compatible = ti,am335x-bone, "ti,am33xx";
    #address-cells = <1>;
    #size-cells = <1>;
    memory@0x80000000 {
        device_type = "memory";
        reg = <0x80000000 0x20000000>; /* 512 MB */
    };
   [...]
};
```

booting with a Device Tree

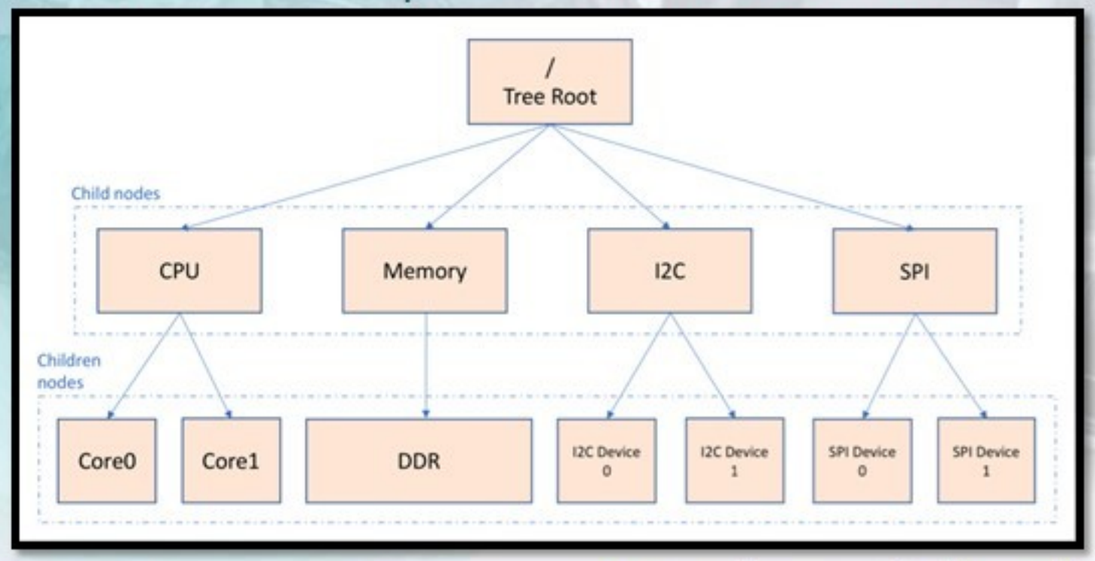
- Some bootloaders have no specific support for the Device Tree
- To solve this issue you can apped device tree to the zImage by CONFIG_ARM_APPENDED_DTB.
  - It tells the kernel to look for a DTB right after the kernel image.
    - cat arch/arm/boot/zImage arch/arm/boot/dts/myboard.dtb > my-zImage
    - mkimage … -d my-zImage my-uImage

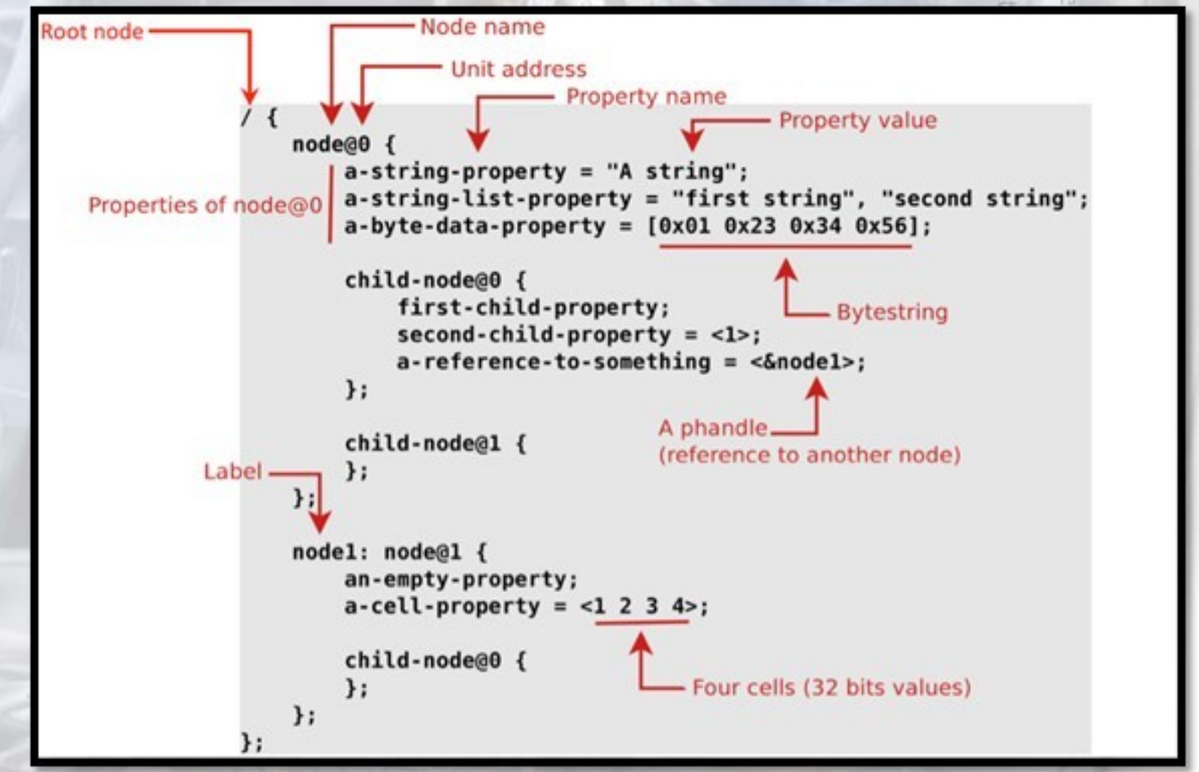
```
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> Boot options
                                Boot options
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highl
hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                -*- Flattened Device Tree support
                [*]   Support for the traditional ATAGS boot data passing
                [ ]     Provide old way to pass kernel parameters
                (0) Compressed ROM boot loader base address
                (0) Compressed ROM boot loader BSS address
                [ ] Use appended device tree blob to zImage (EXPERIMENTAL)
                ()  Default kernel command string
                [*] Kexec system call (EXPERIMENTAL)
                [*]   Export atags in procfs
                [*] Build kdump crash kernel (EXPERIMENTAL)
                -*- Auto calculation of the decompressed kernel image address
                [ ] UEFI runtime support
```

```
.config - Linux/arm 5.4.0-rc8 Kernel Configuration
> Boot options
                Use appended device tree blob to zImage (EXPERIMENTAL)
CONFIG_ARM_APPENDED_DTB:

With this option, the boot code will look for a device tree binary
(DTB) appended to zImage
(e.g. cat zImage <filename>.dtb > zImage_w_dtb).

This is meant as a backward compatibility convenience for those
systems with a bootloader that can't be upgraded to accommodate
the documented boot protocol using a device tree.

Beware that there is very little in terms of protection against
this option being confused by leftover garbage in memory that might
look like a DTB header after a reboot if no actual DTB is appended
to zImage.  Do not leave this option active in a production kernel
if you don't intend to always append a DTB.  Proper passing of the
location into r2 of a bootloader provided DTB is always preferable
to this option.

Symbol: ARM_APPENDED_DTB [=n]
Type  : bool
Prompt: Use appended device tree blob to zImage (EXPERIMENTAL)
  Location:
    -> Boot options
  Defined at arch/arm/Kconfig:1777
  Depends on: OF [=y]
  Selected by [n]:
  - ARCH_GEMINI [=n] && ARCH_MULTI_V4 [=n]
  - MACH_IXP4XX_OF [=n] && ARCH_IXP4XX [=n]
```

Device tree Basic structure

- Represents hardware as a hierarchy
- Starts at a root node, named "/"
- Nodes may contain child nodes
  - Each node contains name = value pairs
- Must contain version: /dts-v1/;
- Comments are C style: /* this is a comment */





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

Exploring the DT on the target

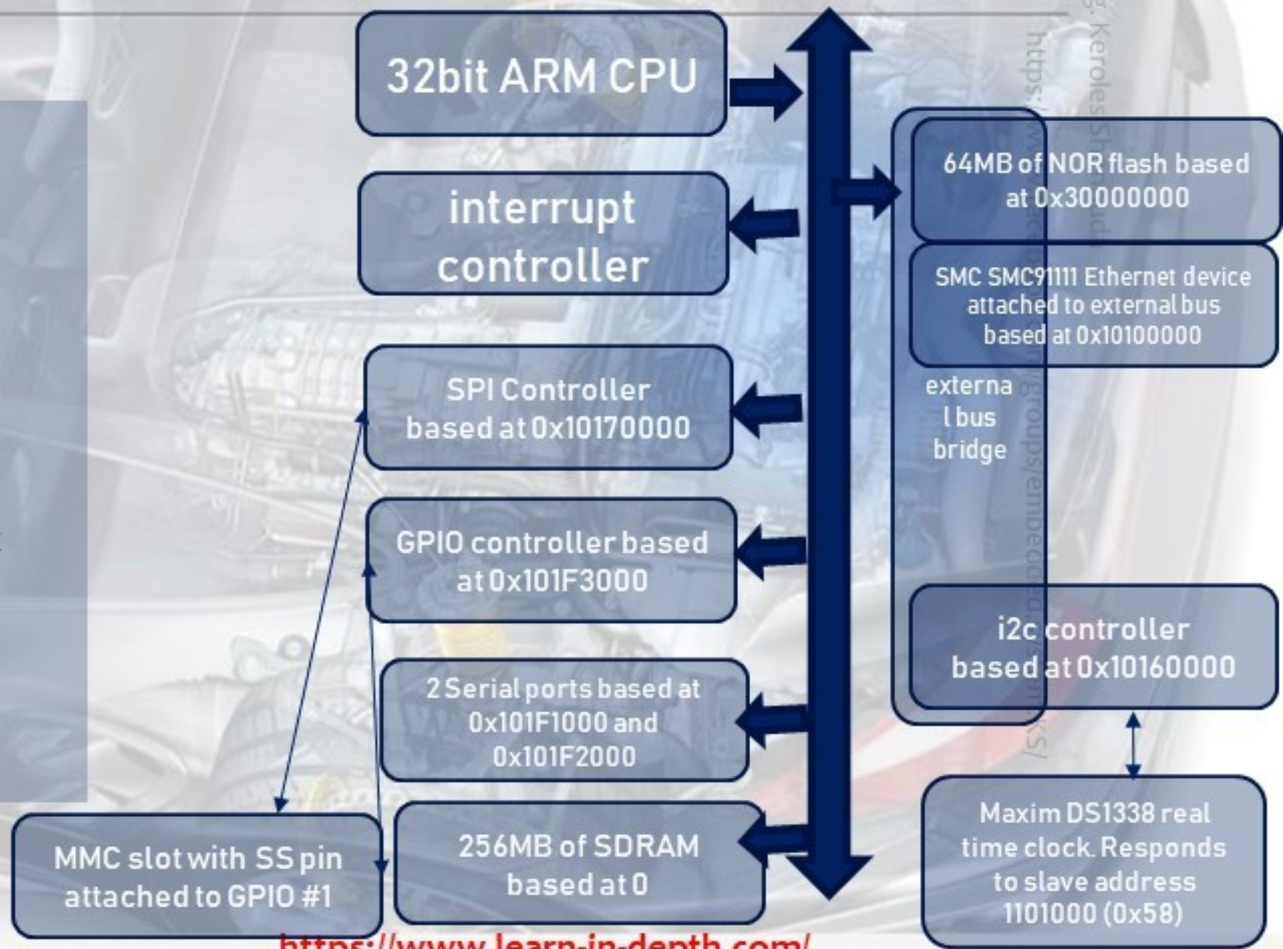
- ls /sys/firmware/devicetree/base, there is a directory/le representation of the Device Tree contents
- If dtc is available on the target, possible to "unpack" the Device Tree using:
  - dtc -I fs /sys/firmware/devicetree/base

```
[root@vexpress ]# ls -l /sys/firmware/devicetree/base/
total 0
-r--r--r--    1 0        0                4 Feb 22 01:52 #address-cells
-r--r--r--    1 0        0                4 Feb 22 01:52 #size-cells
drwxr-xr-x    2 0        0                0 Feb 22 01:52 aliases
-r--r--r--    1 0        0                4 Feb 22 01:52 arm,hbi
-r--r--r--    1 0        0                4 Feb 22 01:52 arm,vexpress,site
drwxr-xr-x    2 0        0                0 Feb 22 01:52 cache-controller@1e00a000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 chosen
drwxr-xr-x    3 0        0                0 Feb 22 01:52 clcd@10020000
-r--r--r--    1 0        0               34 Feb 22 01:52 compatible
drwxr-xr-x    6 0        0                0 Feb 22 01:52 cpus
drwxr-xr-x   15 0        0                0 Feb 22 01:52 dcc
drwxr-xr-x    2 0        0                0 Feb 22 01:52 hsb@e0000000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 interrupt-controller@1e001000
-r--r--r--    1 0        0                4 Feb 22 01:52 interrupt-parent
drwxr-xr-x    2 0        0                0 Feb 22 01:52 memory-controller@100e0000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 memory-controller@100e1000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 memory@60000000
-r--r--r--    1 0        0                8 Feb 22 01:52 model
-r--r--r--    1 0        0                1 Feb 22 01:52 name
drwxr-xr-x    2 0        0                0 Feb 22 01:52 pmu
drwxr-xr-x    3 0        0                0 Feb 22 01:52 reserved-memory
drwxr-xr-x    2 0        0                0 Feb 22 01:52 scu@1e000000
drwxr-xr-x    3 0        0                0 Feb 22 01:48 smb@4000000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 timer@100e4000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 timer@1e000600
drwxr-xr-x    2 0        0                0 Feb 22 01:52 watchdog@100e5000
drwxr-xr-x    2 0        0                0 Feb 22 01:52 watchdog@1e000620
[root@vexpress ]#
```


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

Let us try to write a device tree from scratch to Sample Machine

- One 32bit ARM CPU
- processor local bus attached to memory mapped serial port, spi bus controller, i2c controller, interrupt controller, and external bus bridge
- 256MB of SDRAM based at 0
- 2 Serial ports based at 0x101F1000 and 0x101F2000
- GPIO controller based at 0x101F3000
- SPI controller based at 0x10170000 with following devices
  - MMC slot with SS pin attached to GPIO #1
- External bus bridge with following devices
  - SMC SMC91111 Ethernet device attached to external bus based at 0x10100000
  - i2c controller based at 0x10160000 with following devices
    - Maxim DS1338 real time clock. Responds to slave address 1101000 (0x58)
  - 64MB of NOR flash based at 0x30000000

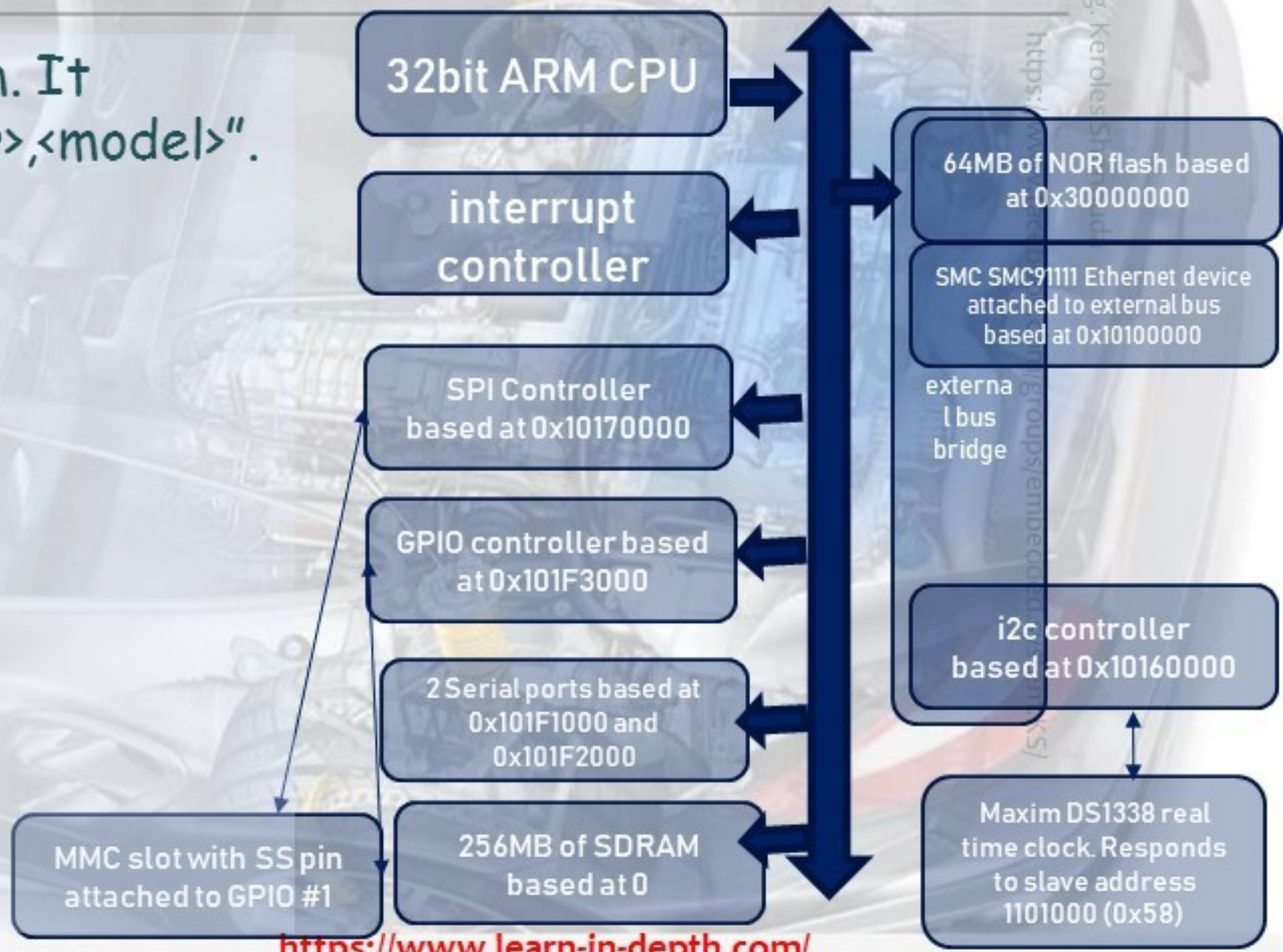


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Let us try to write a device tree from scratch to Sample Machine Cont.

- **compatible** specifies the name of the system. It contains a string in the form "<manufacturer>,<model>".

```
/dts-v1/;

/ {
compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
};
```

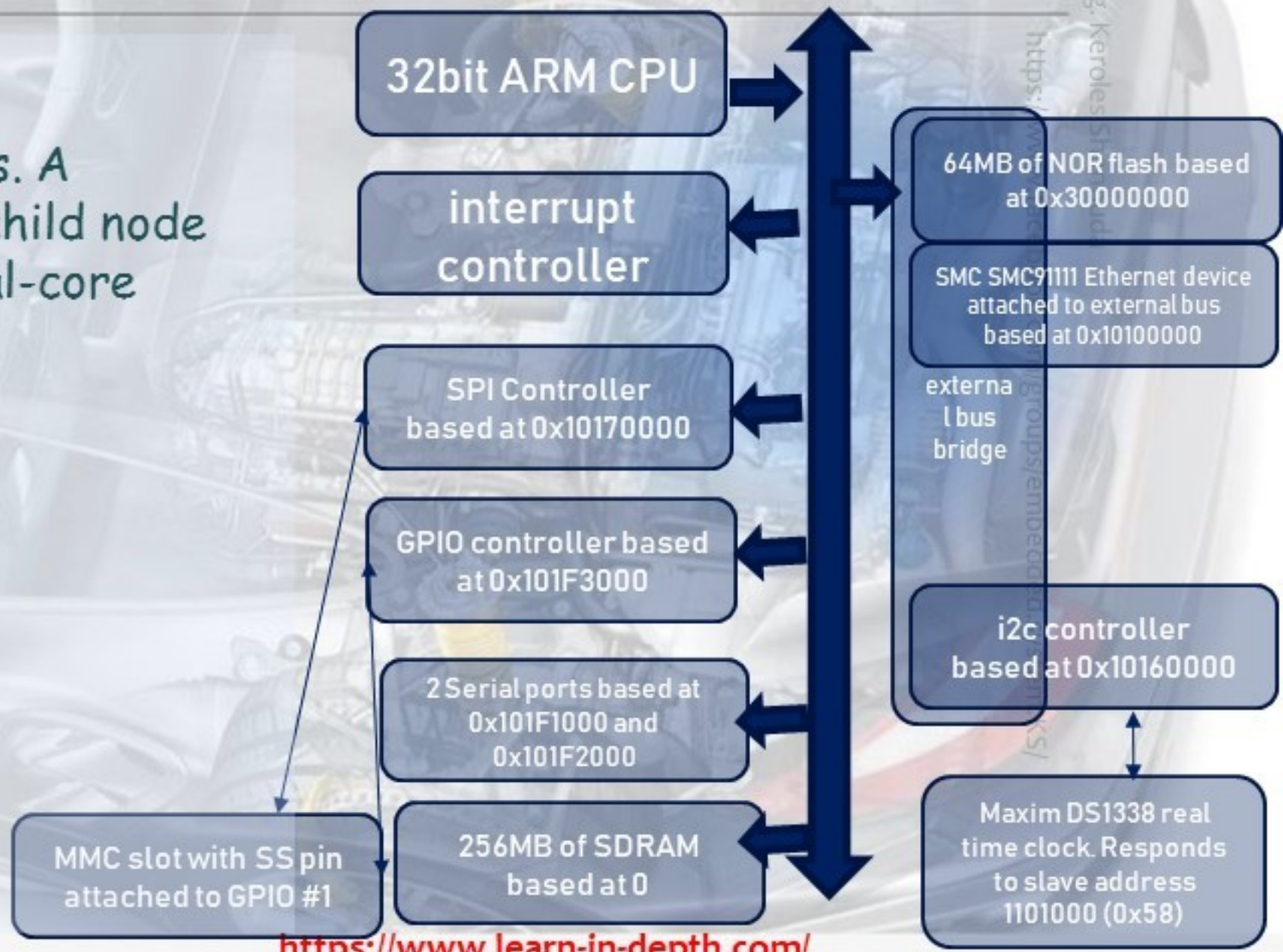


Let us try to write a device tree from scratch to Sample Machine Cont.


```
[root@vexpress ]# cat  /sys/firmware/devicetree/base/cpus/cpu@0/compatible
arm,cortex-a9[root@vexpress ]#
```

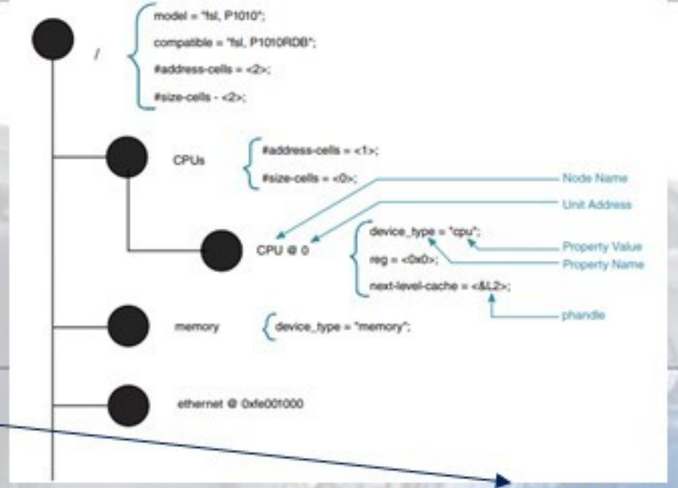
- CPUs
  - Next step is to describe for each of the CPUs. A container node named "cpus" is added with a child node for each CPU. In this case the system is a dual-core Cortex A9 system from ARM.

```
/dts-v1/;

/ {
compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
 cpus {
     cpu@0 {
        compatible = "arm,cortex-a9";
     };
     cpu@1 {
        compatible = "arm,cortex-a9";
     };
  };
};
```



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

Syntax



Aliases

The aliases node is an index of other nodes. The properties of the node are paths within the device tree, not phandles

```
aliases {
 etherneto = &eneto;
 ethernet1 = &enet1;
 ethernet2 = &enet2;
 serialo = &serialo;
 serial1 = &serial1;
 pcio = &pcio;
};
```

Phandle

A phandle (pointer handle) is a 32-bit value associated with a node that is used to uniquely identify that node so that the node can be reference from a property in another node.

```
name@address {
<key>= <&label>;
};
```

```
label: name@adresss { }
```

<&label> is converted to the phandle for the labeled node by the DTC

Properties

A node may contain multiple properties arranged in name = value format. The **name** consists of a **string**, while **value** can be an **array of strings, bytes, numbers, or phandles, or a mixture** of types.

For example, value can be:

- compatible = "fsl,mpc8610-msi", "fsl,mpic-msi";
- reg = <0 0 0x8000000>;
- interrupt-parent = <&mpic>;

Node names

- The node name is a label used to identify the node
- Child nodes must be uniquely named, but can alternatively be addressed by a "unit name,"
  - for example, i2c@3000, i2c@4000, and so on

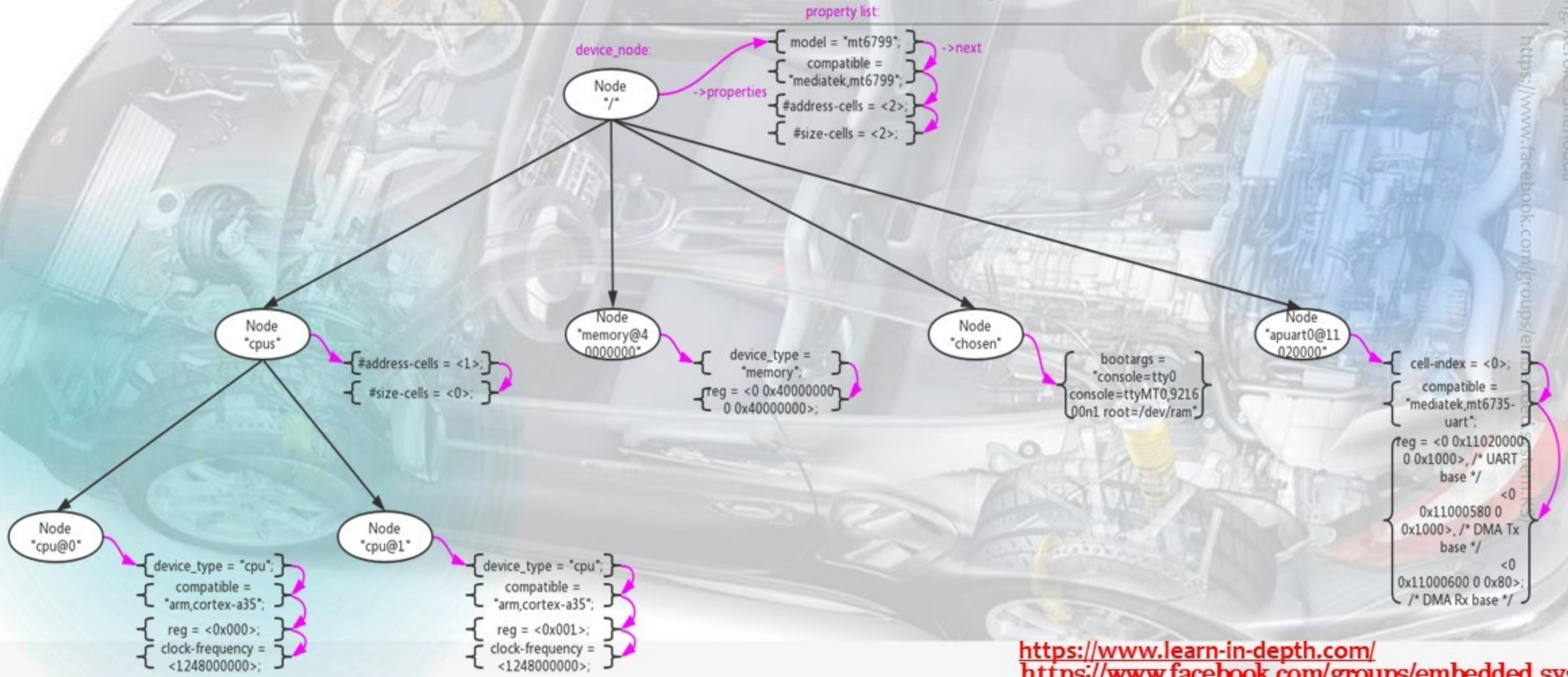
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Phandle example

- dtc creates a phandle from a label when it sees a reference from another node
- Decompiling the dtb shows the actual code (next slide)

```
{
    interrupt-controller@48200000 {
        compatible = "ti,am33xx-intc";
        interrupt-controller;
        #interrupt-cells = <0x1>;
        reg = <0x48200000 0x1000>;
        linux,phandle = <0x1>;
        phandle = <0x1>;                <-------- declaration
        };

    serial@44e09000 {
        compatible = "ti,omap3-uart";
        ti,hwmods = "uart1";
        clock-frequency = <0x2dc6c00>;
        reg = <0x44e09000 0x2000>;
        interrupts = <0x48>;
        status = "okay";
        interrupt-parent = <0x1>;       <------- reference
    };
```

Device tree Nodes example



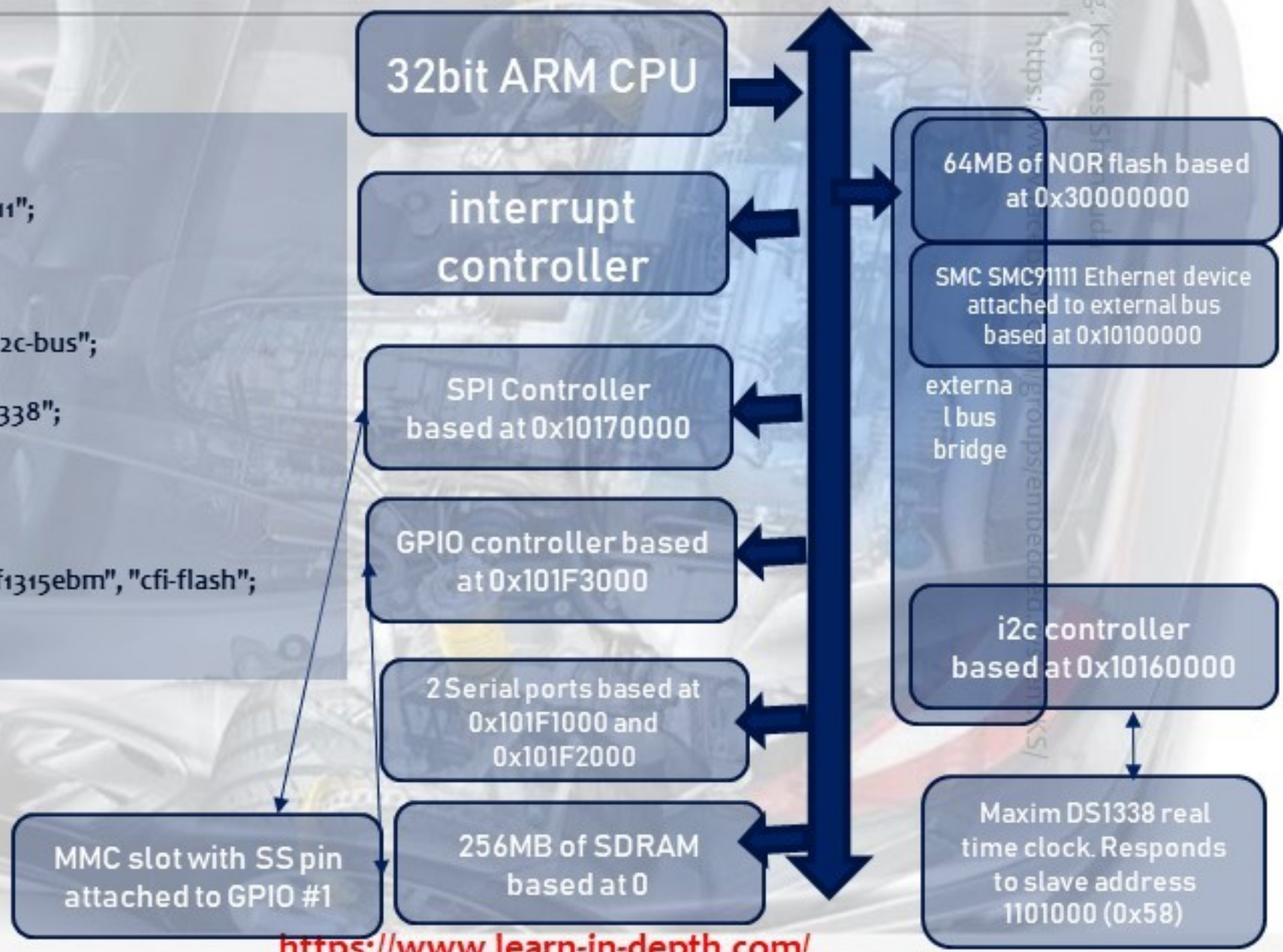
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Let us try to write a device tree from scratch to Sample Machine Cont.

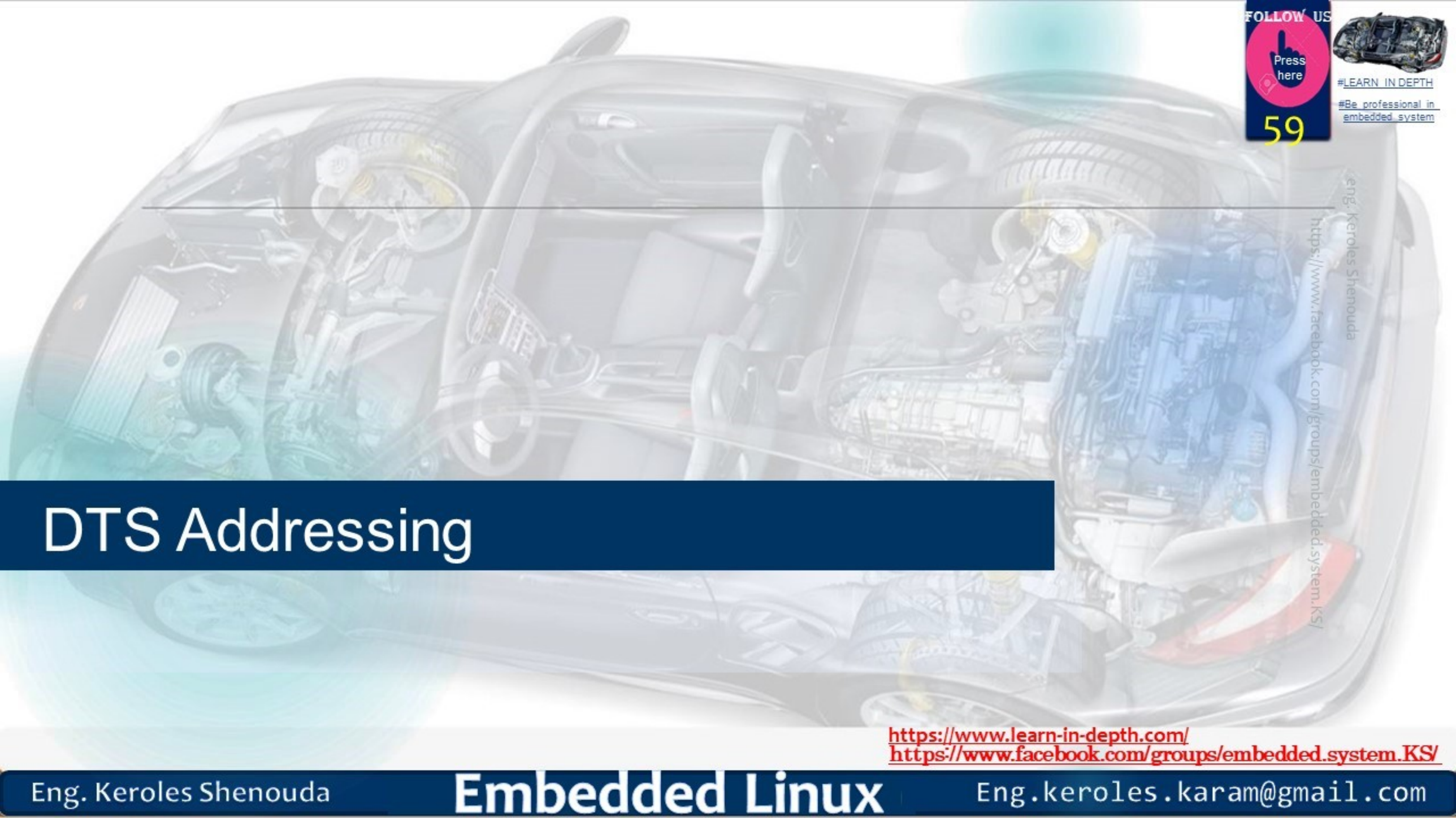
- Devices
  - Every device in the system is represented by a device tree node..

```
/dts-v1/;

/ {
compatible = "arm,vexpress,v2p-ca9", "arm,vexpress";
 cpus {
     cpu@0 {
       compatible = "arm,cortex-a9";
     };
     cpu@1 {
       compatible = "arm,cortex-a9";
     };
  };
 serial@101F0000 {
     compatible = "arm,pl011";
  };

  serial@101F2000 {
     compatible = "arm,pl011";
  };

  gpio@101F3000 {
     compatible = "arm,pl061";
  };

  interrupt-controller@10140000 {
     compatible = "arm,pl190";
  };

  spi@10115000 {
     compatible = "arm,pl022";
  };
};
```

```
external-bus {
    ethernet@0,0 {
      compatible = "smc,smc91c111";
    };

    i2c@1,0 {
      compatible = "acme,a1234-i2c-bus";
      rtc@58 {
        compatible = "maxim,ds1338";
      };
    };

    flash@2,0 {
      compatible = "samsung,k8f1315ebm", "cfi-flash";
    };
  };
```



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



DTS Addressing

DTS Addressing

#LEARN IN DEPTH

#Be professional in embedded system

Ranges (Address Translation)

memory mapped address the device tree must specify how to translate addresses from one domain to another. The ranges property is used for this purpose.

- Offset 0 from chip select 0 is mapped to address range 0x10100000..0x101effff
- Offset 0 from chip select 1 is mapped to address range 0x10160000..0x1016ffff
- Offset 0 from chip select 2 is mapped to address range 0x30000000..0x3affffff

```
...
external-bus {
    #address-cells = <2>
    #size-cells = <1>;
    ranges = <0 0  0x10100000   0x10000     // Chipselect 1, Ethernet
              1 0  0x10160000   0x10000     // Chipselect 2, i2c controller
              2 0  0x30000000   0x1000000>; // Chipselect 3, NOR Flash

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
            reg = <58>;
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "cfi-flash";
        reg = <2 0 0x4000000>;
    };
};
```

Memory Mapped Devices

#size-cells is used to state how large the length field is in each child reg tuple.

In the following example,

- each address value is 1 cell (32 bits), and each length value is also 1 cell, which is typical on 32 bit systems.
- 64 bit machines may use a value of 2 for #address-cells and #size-cells to get 64 bit addressing in the device tree.

```
/ {
    #address-cells = <1>;
    #size-cells = <1>;

    ...

    serial@101f0000 {
        compatible = "arm,pl011";
        reg = <0x101f0000 0x1000 >;
    };
```

ADDR Size

Some devices live on a bus with a different addressing scheme for devices attached to the external bus with the chip select number encoded into the address.

```
external-bus {
    #address-cells = <2>;
    #size-cells = <1>;

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "c
        reg = <2 0 0x4000000>;
```

each reg entry contains 3 cells;
the chipselect number, the offset, and the length.

CPU addressing

The CPU nodes represent the simplest case when talking about addressing. Each CPU is assigned a single unique ID, and there is no size associated with CPU ids.

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        compatible = "arm,cortex-a9";
        reg = <0>;
    };
    cpu@1 {
        compatible = "arm,cortex-a9";
        reg = <1>;
    };
};
```

Non Memory Mapped Devices

Other devices are not memory mapped on the processor bus. They can have address ranges, but they are not directly accessible by the CPU.
To take the example of i2c devices

```
#address-cells = <1>;
#size-cells = <0>;
reg = <1 0 0x1000>;
rtc@58 {
    compatible = "maxim,ds1338";
    reg = <58>;
};
```

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

CPU addressing

- The CPU nodes represent the simplest case when talking about addressing.
- Each CPU is assigned a single unique ID, and there is no size associated with CPU ids.

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        compatible = "arm,cortex-a9";
        reg = <0>;
    };
    cpu@1 {
        compatible = "arm,cortex-a9";
        reg = <1>;
    };
};
```

Memory Mapped Devices

- **#size-cells** is used to state how large the length field is in each child reg tuple.
- In the following example,
  - each **address value** is 1 cell (32 bits), and each **length value** is also 1 cell, which is typical on 32 bit systems.
  - 64 bit machines may use a value of 2 for #address-cells and #size-cells to get 64 bit addressing in the device tree.

```
/ {
    #address-cells = <1>;
    #size-cells = <1>;

    ...

    serial@101f0000 {
        compatible = "arm,pl011";
        reg = <0x101f0000 0x1000 >;
    };
```

ADDR Size

Memory Mapped Devices Cont.

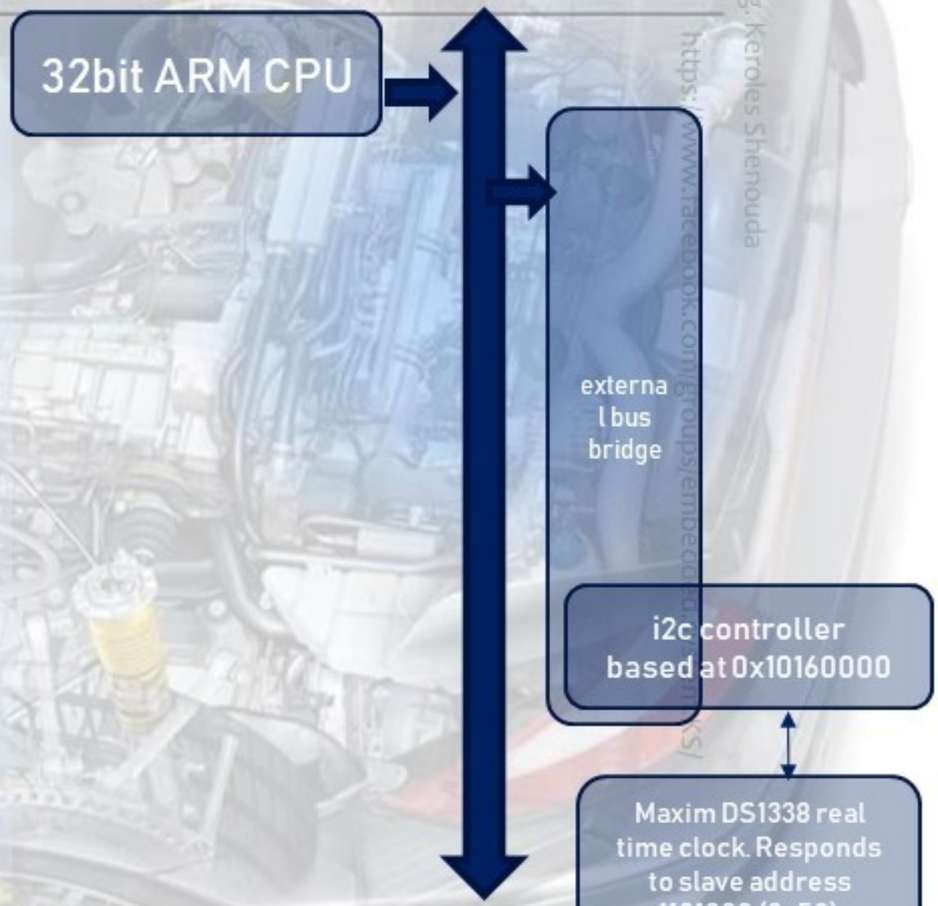
- Some devices live on a bus with a different addressing scheme
  - for devices attached to the external bus with the chip select number encoded into the address.
- In this example
  - each reg entry contains 3 cells;
  - the **chipselect number**, the **offset**, and the **length**.

```
external-bus {
    #address-cells = <2>;
    #size-cells = <1>;

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "c
        reg = <2 0 0x4000000>;
```

Non Memory Mapped Devices

- Other devices are not memory mapped on the processor bus. They can have address ranges, but they are not directly accessible by the CPU.
- To take the example of i2c devices

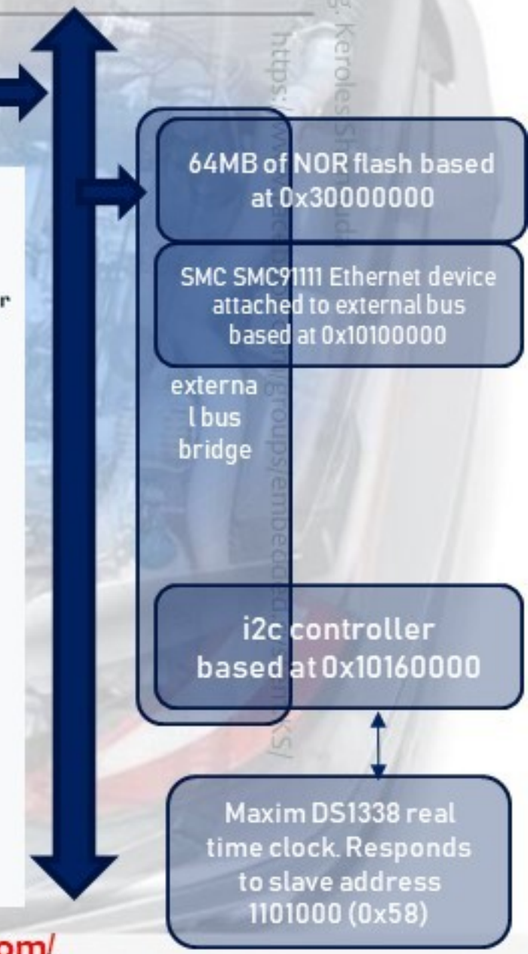
```
#address-cells = <1>;
#size-cells = <0>;
reg = <1 0 0x1000>;
rtc@58 {
    compatible = "maxim,ds1338";
    reg = <58>;
};
```



Ranges (Address Translation)

- memory mapped address the device tree must specify how to translate addresses from one domain to another.
- The ranges property is used for this purpose.
  - Offset 0 from chip select 0 is mapped to address range 0x10100000..0x1010ffff
  - Offset 0 from chip select 1 is mapped to address range 0x10160000..0x1016ffff
  - Offset 0 from chip select 2 is mapped to address range 0x30000000..0x30ffffff

```
...
external-bus {
    #address-cells = <2>
    #size-cells = <1>;
    ranges = <0 0  0x10100000   0x10000     // Chipselect 1, Ethernet
              1 0  0x10160000   0x10000     // Chipselect 2, i2c controller
              2 0  0x30000000   0x1000000>; // Chipselect 3, NOR Flash

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
            reg = <58>;
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "cfi-flash";
        reg = <2 0 0x4000000>;
    };
```



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

DTS Addressing

#LEARN IN DEPTH

#Be professional in embedded system

Ranges (Address Translation)

memory mapped address the device tree must specify how to translate addresses from one domain to another. The ranges property is used for this purpose.

- Offset 0 from chip select 0 is mapped to address range 0x10100000..0x1010ffff
- Offset 0 from chip select 1 is mapped to address range 0x10160000..0x1016ffff
- Offset 0 from chip select 2 is mapped to address range 0x30000000..0x3fffffff

```
...
external-bus {
    #address-cells = <2>
    #size-cells = <1>;
    ranges = <0 0  0x10100000   0x10000     // Chipselect 1, Ethernet
              1 0  0x10160000   0x10000     // Chipselect 2, i2c controller
              2 0  0x30000000   0x1000000>; // Chipselect 3, NOR Flash

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
            reg = <58>;
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "cfi-flash";
        reg = <2 0 0x4000000>;
    };
};
```

Memory Mapped Devices

#size-cells is used to state how large the length field is in each child reg tuple.

In the following example,

- each address value is 1 cell (32 bits), and each length value is also 1 cell, which is typical on 32 bit systems.
- 64 bit machines may use a value of 2 for #address-cells and #size-cells to get 64 bit addressing in the device tree.

```
/ {
    #address-cells = <1>;
    #size-cells = <1>;

    ...

    serial@101f0000 {
        compatible = "arm,pl011";
        reg = <0x101f0000 0x1000 >;
    };
```

ADDR Size

Some devices live on a bus with a different addressing scheme for devices attached to the external bus with the chip select number encoded into the address.

```
external-bus {
    #address-cells = <2>;
    #size-cells = <1>;

    ethernet@0,0 {
        compatible = "smc,smc91c111";
        reg = <0 0 0x1000>;
    };

    i2c@1,0 {
        compatible = "acme,a1234-i2c-bus";
        reg = <1 0 0x1000>;
        rtc@58 {
            compatible = "maxim,ds1338";
        };
    };

    flash@2,0 {
        compatible = "samsung,k8f1315ebm", "c
        reg = <2 0 0x4000000>;
```

each reg entry contains 3 cells; the chipselect number, the offset, and the length.

CPU addressing

The CPU nodes represent the simplest case when talking about addressing. Each CPU is assigned a single unique ID, and there is no size associated with CPU ids.

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        compatible = "arm,cortex-a9";
        reg = <0>;
    };
    cpu@1 {
        compatible = "arm,cortex-a9";
        reg = <1>;
    };
};
```

Non Memory Mapped Devices

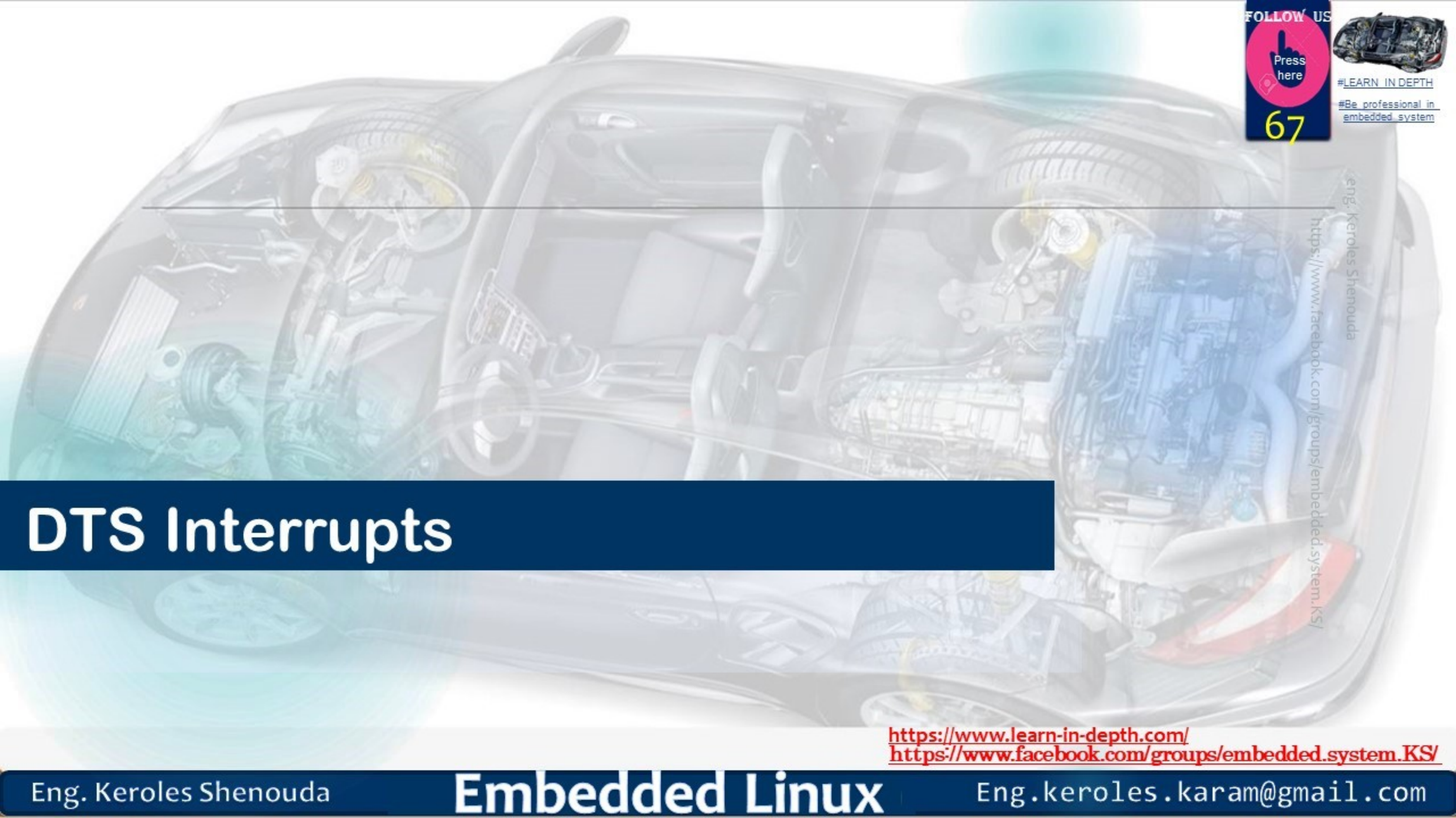
Other devices are not memory mapped on the processor bus. They can have address ranges, but they are not directly accessible by the CPU.

To take the example of i2c devices

```
#address-cells = <1>;
#size-cells = <0>;
reg = <1 0 0x1000>;
rtc@58 {
    compatible = "maxim,ds1338";
    reg = <58>;
};
```

https://www.learn-in-depth.com/

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



DTS Interrupts

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

DTS Interrupts

#LEARN IN DEPTH

#Be professional in embedded system

interrupts

A property of a **device node** containing a list of interrupt specifiers, one for each interrupt output signal on the device.

```
compatible = "acme,coyotes-revenge";
#address-cells = <1>;
#size-cells = <1>;
interrupt-parent = <&intc>;
```

interrupt-parent

A property of a device node containing a **phandle** to the **interrupt controller** that it is attached to. Nodes that do not have an interrupt-parent property can also inherit the property from their parent node.

```
gpio@101f3000 {
    compatible = "arm,pl061";
    reg = <0x101f3000 0x1000
           0x101f4000 0x0010>;
    interrupts = < 3 0 >;
};

intc: interrupt-controller@10140000 {
    compatible = "arm,pl190";
    reg = <0x10140000 0x1000 >;
    interrupt-controller;
    #interrupt-cells = <2>;
};

spi@10115000 {
    compatible = "arm,pl022";
    reg = <0x10115000 0x1000 >;
    interrupts = < 4 0 >;
};
```

#interrupt-cells

This is a property of the **interrupt controller node**. It states how many cells are in an interrupt specifier for this interrupt controller (Similar to #address-cells and #size-cells).

interrupt-controller

An empty property declaring a node as a device that receives interrupt signals

```
intc: interrupt-controller@10140000 {
    compatible = "arm,pl190";
    reg = <0x10140000 0x1000 >;
    interrupt-controller;
    #interrupt-cells = <2>;
};
```

Some things to notice:

- The machine has **a single interrupt controller**, interrupt-controller@10140000.
- The label 'intc:' has been added to the interrupt controller node, and the label was used to assign a phandle to the interrupt-parent property in the root node. This interrupt-parent value becomes the default for the system because all child nodes inherit it unless it is explicitly overridden.
- Each device uses an interrupt property to specify a different interrupt input line.
- **#interrupt-cells is 2**, so each interrupt specifier has 2 cells. This example uses the common pattern of using **the first cell to encode the interrupt line number**, and **the second cell to encode flags such as active high vs. active low,** or **edge vs. level sensitive**. For any given interrupt controller, refer to the controller's binding documentation to learn how the specifier is encoded.

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

Another example

e9v3qdl-sabresd.dtsi

```
gpio-keys {
    compatible = "gpio-keys";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpio_keys>;

    home {
        label = "vol-";
        gpios = <&gpio4 8 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_VOLUMEDOWN>;
    };

    enter {
        label = "vol+";
        gpios = <&gpio4 9 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_VOLUMEUP>;
    };

};
```

/drivers/input/keyboard/Gpio_keys.c

```
static const struct of_device_id gpio_keys_of_match[] = {
    { .compatible = "gpio-keys", },
    { },
};
MODULE_DEVICE_TABLE(of, gpio_keys_of_match);

static struct platform_driver gpio_keys_device_driver = {
    .probe      = gpio_keys_probe,
    .remove     = gpio_keys_remove,
    .driver     = {
        .name   = "gpio-keys",
        .pm = &gpio_keys_pm_ops,
        .of_match_table = of_match_ptr(gpio_keys_of_match),
    }
};
```

Device Tree inclusion

- can be split in several files, including each other.
- **.dtsi** files are included files, while .dts files are final Device Trees
- Typically, **.dtsi** will contain definition of **SoC-level** information The
- **.dts** le contains **the board-level** information.
- Inclusion using the DT operator /include/, or since a few kernel releases, the DTS go through the C preprocessor, so
  - #include is recommended.

Definition of the AM33xx SoC

```
/ {
     compatible = "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
               compatible = "ti,omap3-uart";
               reg = <0x44e09000 0x2000>;
               interrupts = <72>;
               status = "disabled";
          };
     };
};
```

am33xx.dtsi

+

Definition of the BeagleBone board

```
#include "am33xx.dtsi"

/ {
     compatible = "ti,am335x-bone", "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
               pinctrl-names = "default";
               pinctrl-0 = <&uart0_pins>;
               status = "okay";
          };
     };
};
```

am335x-bone.dts

=

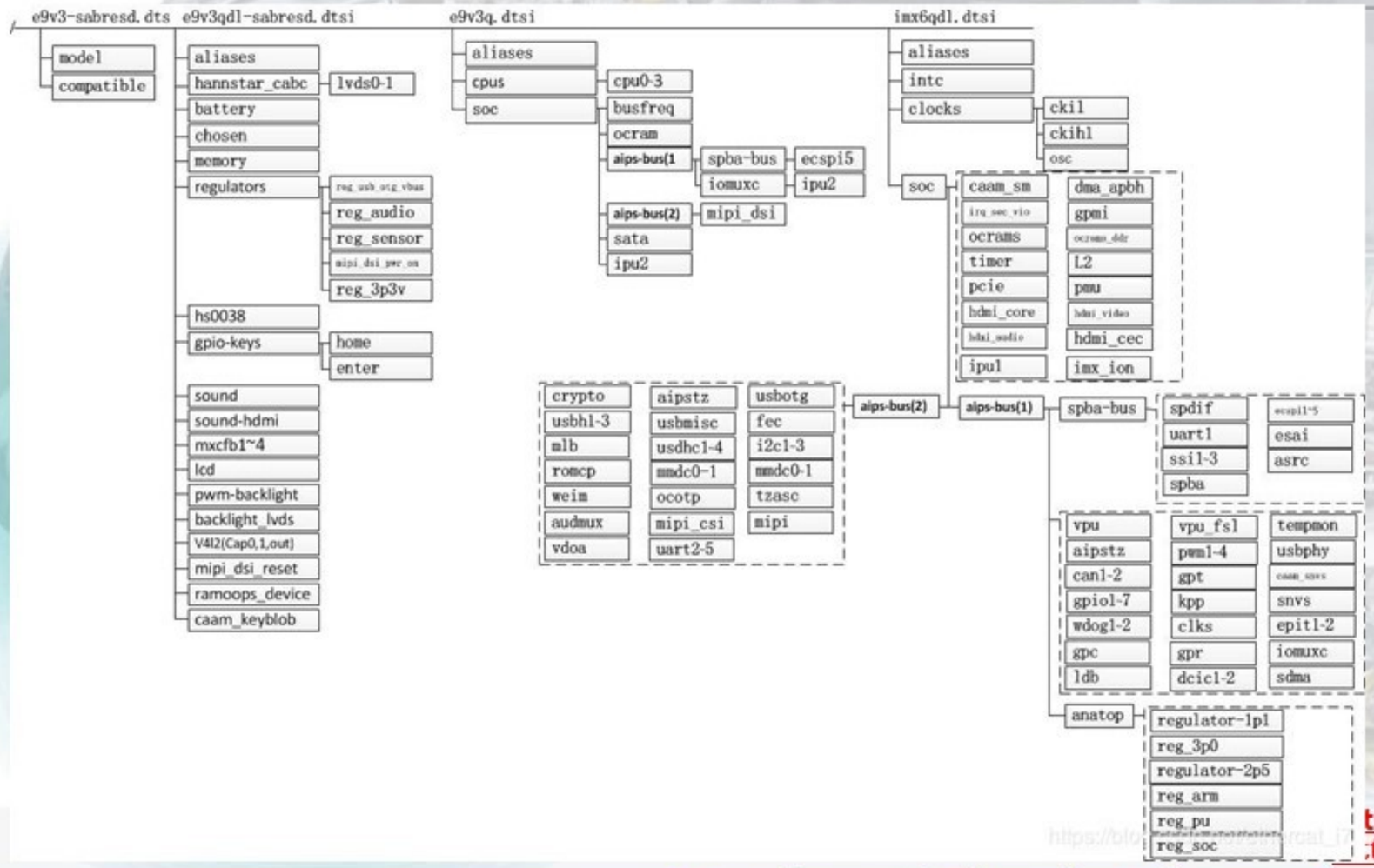
Compiled DTB

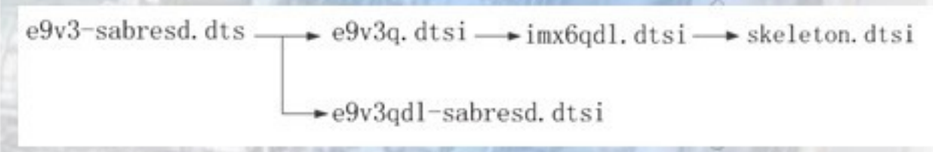
```
/ {
     compatible = "ti,am335x-bone", "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
               compatible = "ti,omap3-uart";
               reg = <0x44e09000 0x2000>;
               interrupts = <72>;
               pinctrl-names = "default";
               pinctrl-0 = <&uart0_pins>;
               status = "okay";
          };
     };
};
```

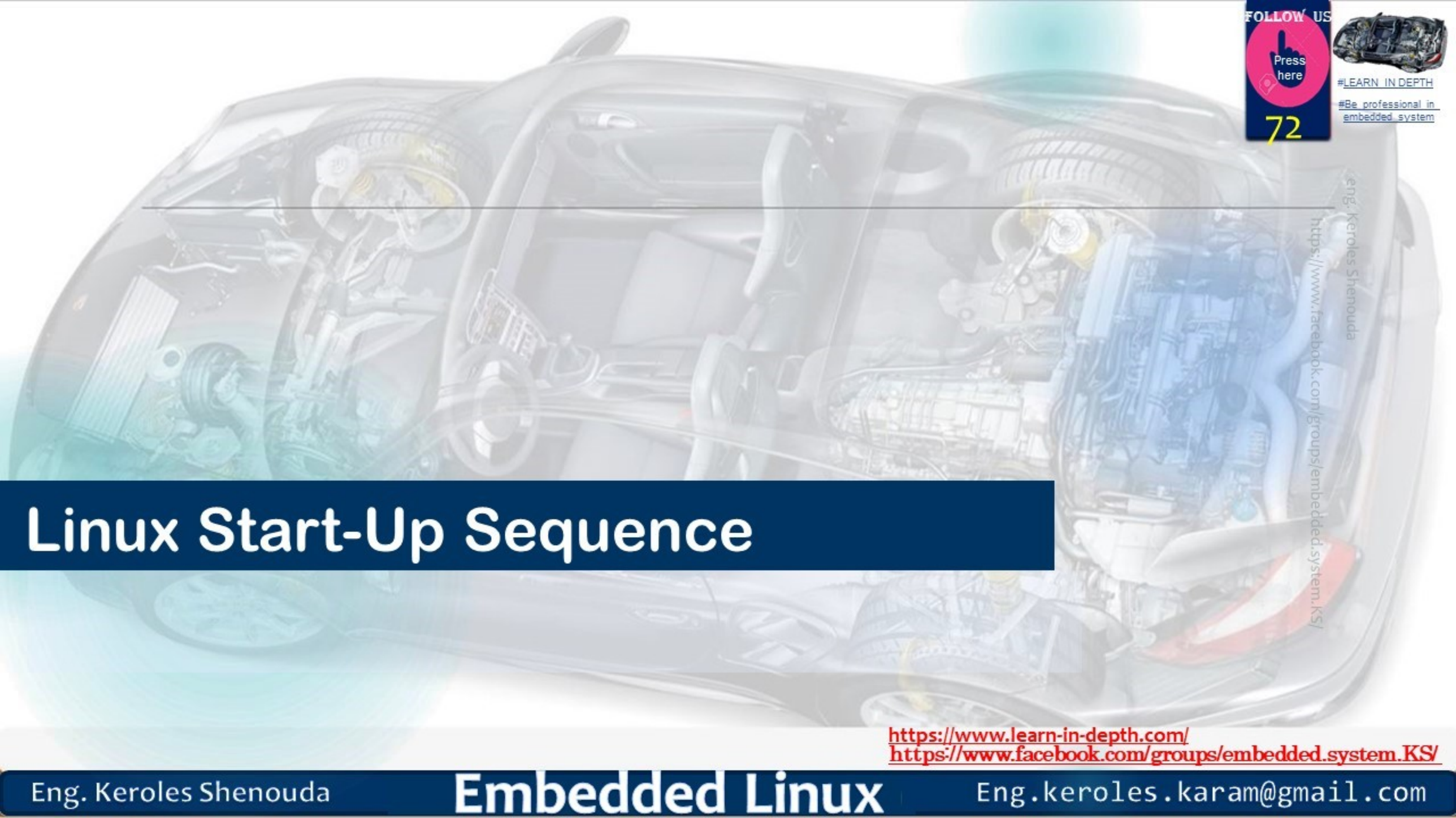
am335x-bone.dtb

Note: the real DTB is in binary format. Here we show the text equivalent of the DTB contents;

e9v3-sabresd.dts example







Linux Start-Up Sequence

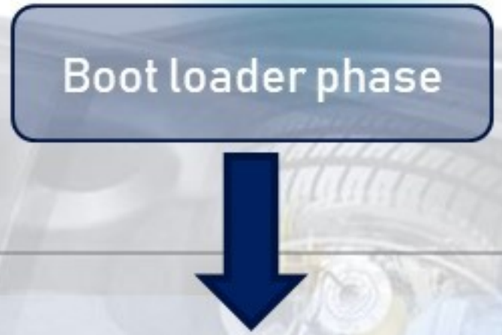
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux Start-Up Sequence

- Boot loader phase:
  - Typically this stage does the hardware initialization and testing, loads the kernel image, and transfers control to the Linux kernel.
- Kernel initialization phase:
  - This stage does the platform-specific initialization, **brings up the kernel subsystems, turns on multitasking, mounts the root file system, and jumps to user space**
- **User- space initialization phase**:
  - Typically this phase brings up **the services, does network initialization**, and then **issues a log-in prompt**.

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

Boot Loader Phase



- **Hardware Initialization This typically includes:**
  - Configuring the CPU speed
  - Memory initialization, such as setting up the registers, clearing the memory, and determining the size of the onboard memory
  - Turning on the caches
  - Setting up the serial port for the boot console
  - Once the above steps are completed successfully, the next step is loading the Linux kernel.
- Downloading Kernel Image and Initial Ram Disk
  - The boot loader needs to **locate the kernel image**, which may be on the **system flash** or **network**.
  - In case the image is compressed (which often is the case), the image needs to be **decompressed**.
  - Also if an **initial ram disk is present**, the boot loader needs to load the image of the initial ram disk to the memory.

Boot loader phase

Boot Loader Phase (Cont.)

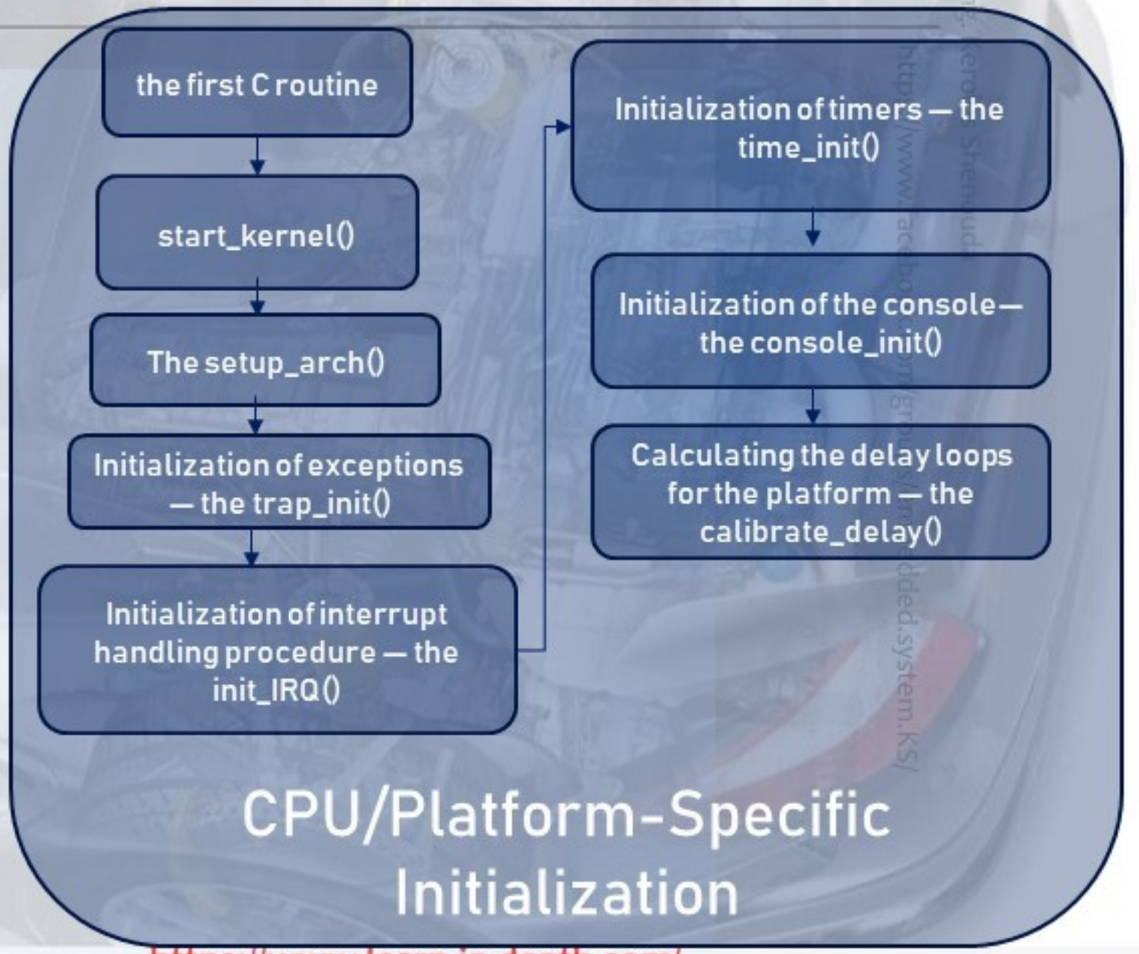
- Setting Up Arguments
  - Argument passing is a very powerful option supported by the Linux kernel. Linux provides a generic way to pass arguments to the kernel across all platforms
  - To set up kernel argument used next string:
    - **setenv bootargs 'console=ttyS0,115200n8 root=/dev/mmcblk0p2 rootfstype=ext2 rw'**
      Here you can see next arguments:
      - 1) console kernel settings 115200n8 and ttyS0 (can be ttyS2 in case console on ext screen)
      - 2) Root fs mount path to SD card /dev/mmcblk0p2, filetype ext2 (can be ext3) privileges read and write
- Jumping to Kernel Entry Point
  - The kernel entry point is decided by the linker script when building the kernel

Kernel Start-Up

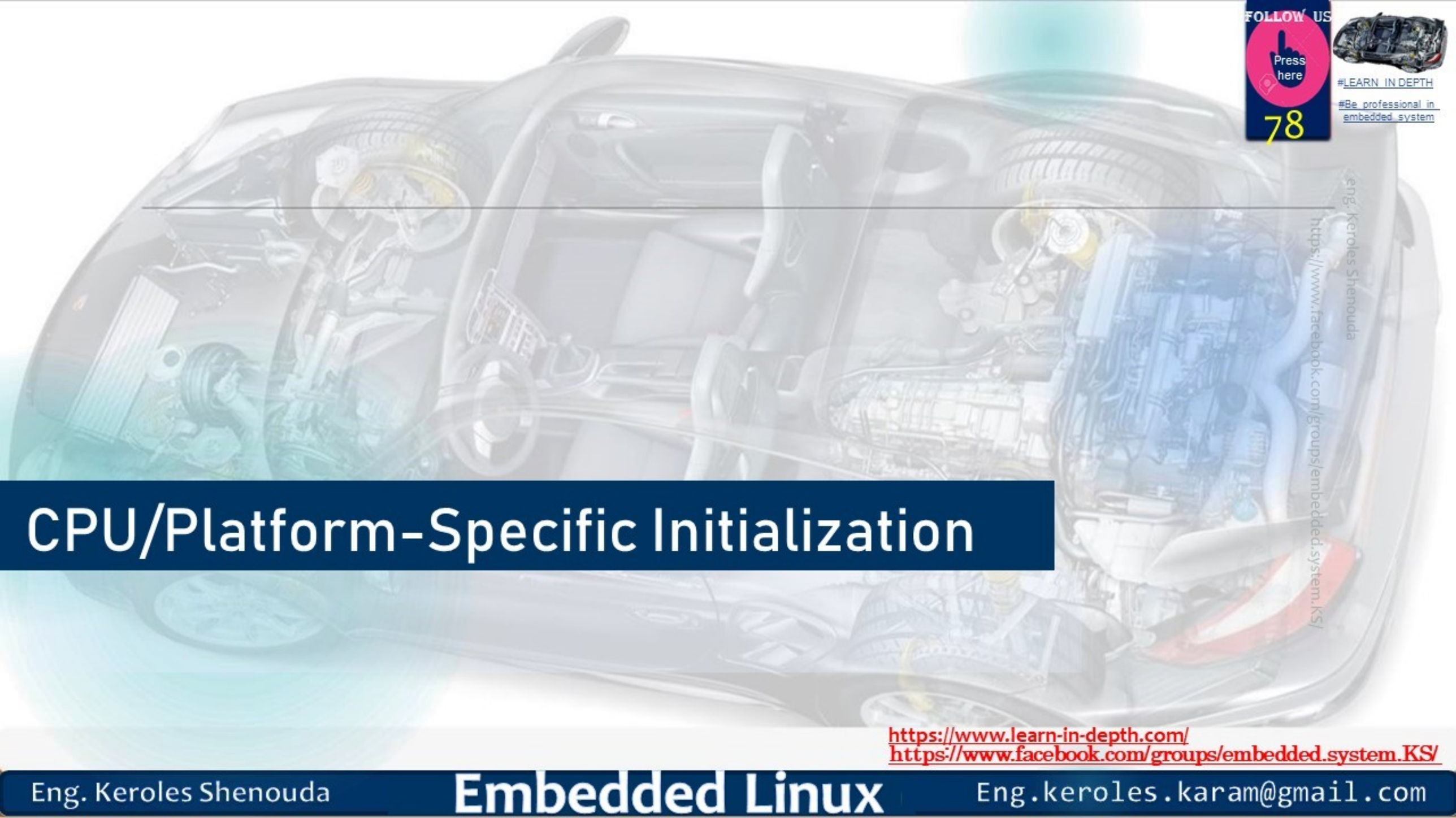
- The kernel start-up can be split into the following phases.
  - CPU/Platform-Specific Initialization
  - Subsystem Initialization
  - Driver Initialization
  - Mounting Root File System
  - Moving to User Space

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

CPU/Platform-Specific Initialization



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

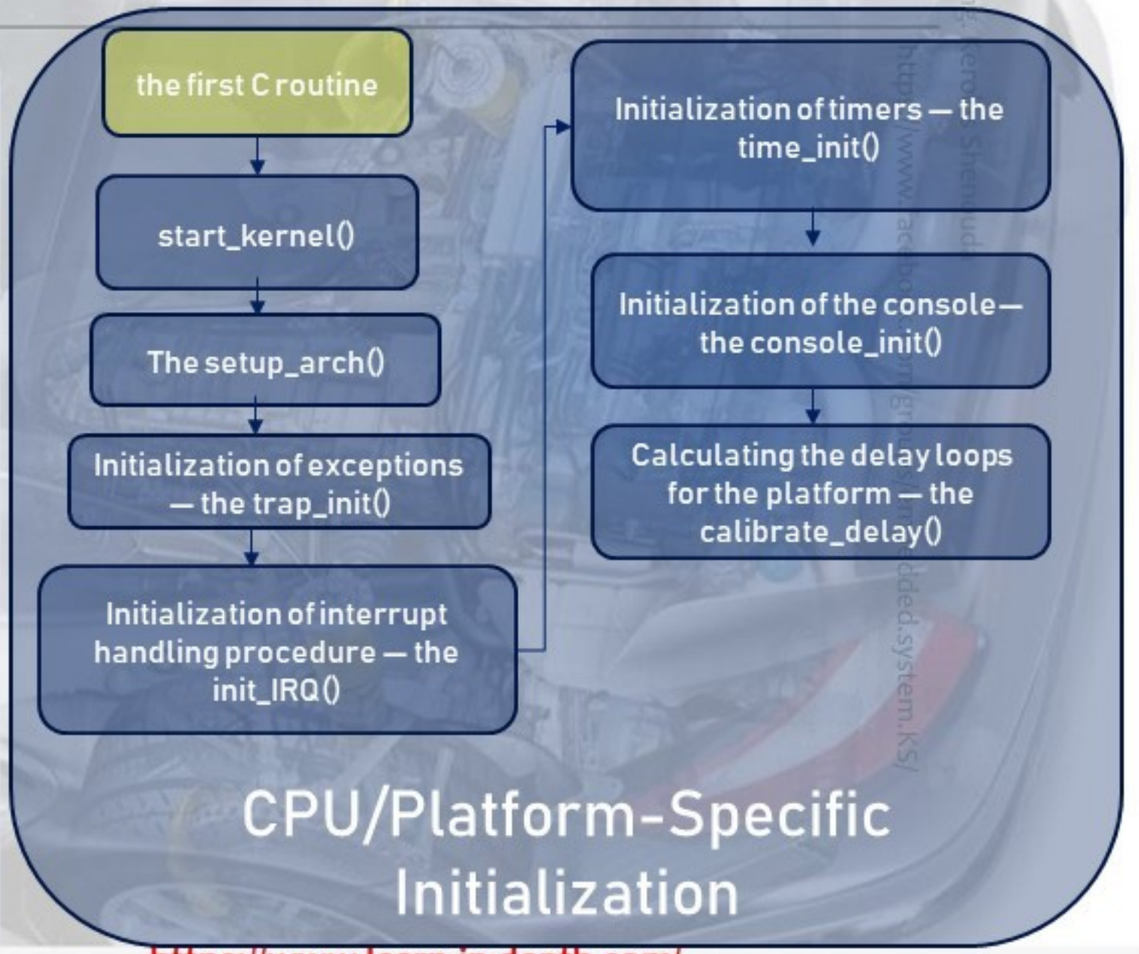


CPU/Platform-Specific Initialization

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

CPU/Platform-Specific Initialization

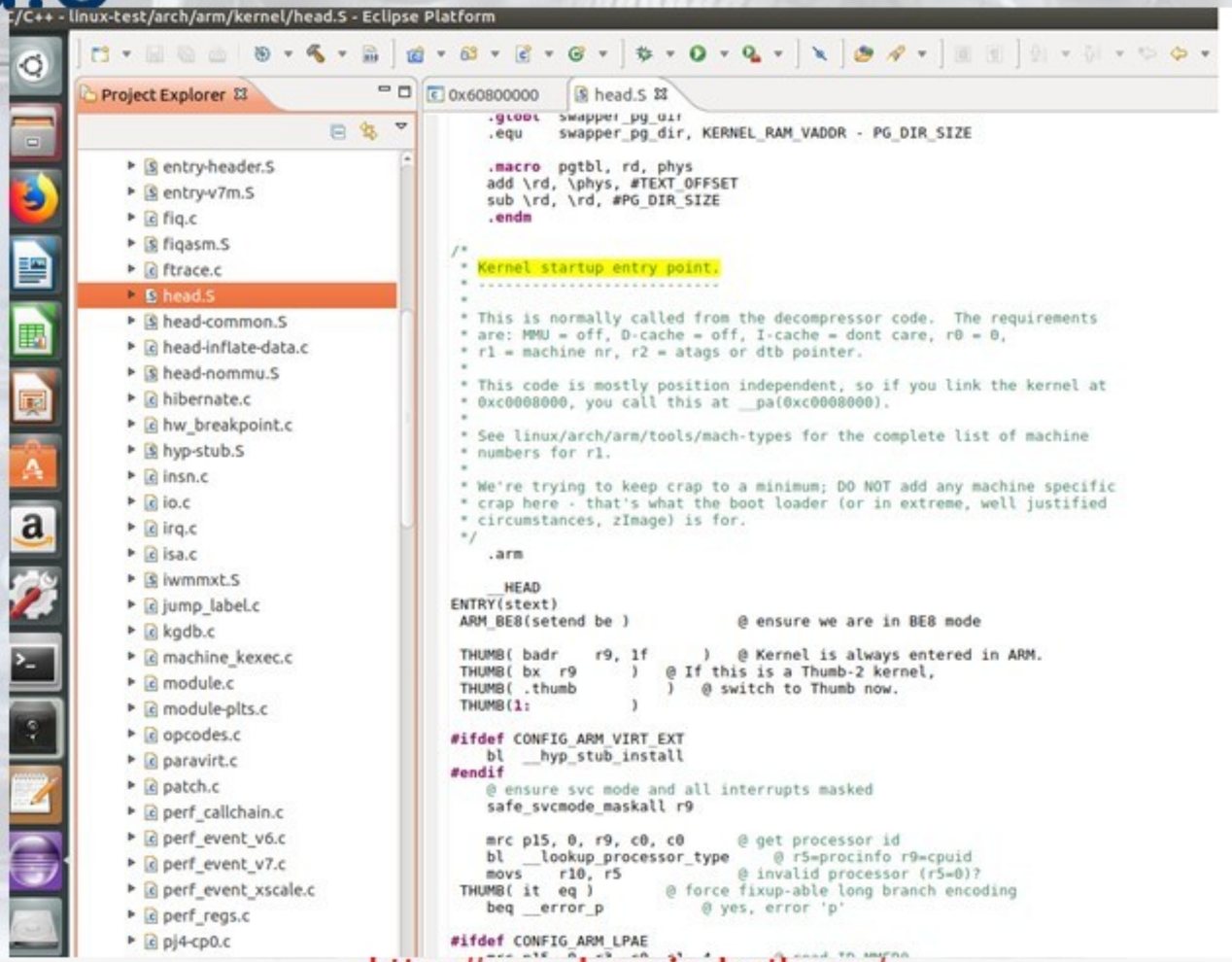
- The kernel entry point is an assembly language routine; the name of this entry point varies (stext on ARM, kernel_entry on MIPS, etc.). Look at the linker script to know the entry point for your platform. This function normally resides in the arch/kernel/head.S file
- In our vexpress A9 system will be
  - arch/arm/kernel/head.S



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

the first C routine: arch/arm/kernel/head.S

- This function does the following:
  - On machines that do not have the MMU turned on, this turns on the MMU.
  - Do cache initialization.
  - Set up the BSS by zeroing it out
  - Set up the stack so that the first C routine can be invoked. The first C routine is the **start_kernel()** function in init/main.c
    - This function is a jumbo function that does a lot of things until it terminates in an idle task (the first task in the system having a process id of 0)



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

start_kernel()
init/main.c

- The kernel's main() function

```
start_kernel(){:
boot_cpu_init();
setup_arch(&command_line);
page_alloc_init();
pr_notice("Kernel command line: );
mm_init();
sched_init();
init_IRQ();
init_timers(); timekeeping_init();
console_init();
rest_init();
}
```

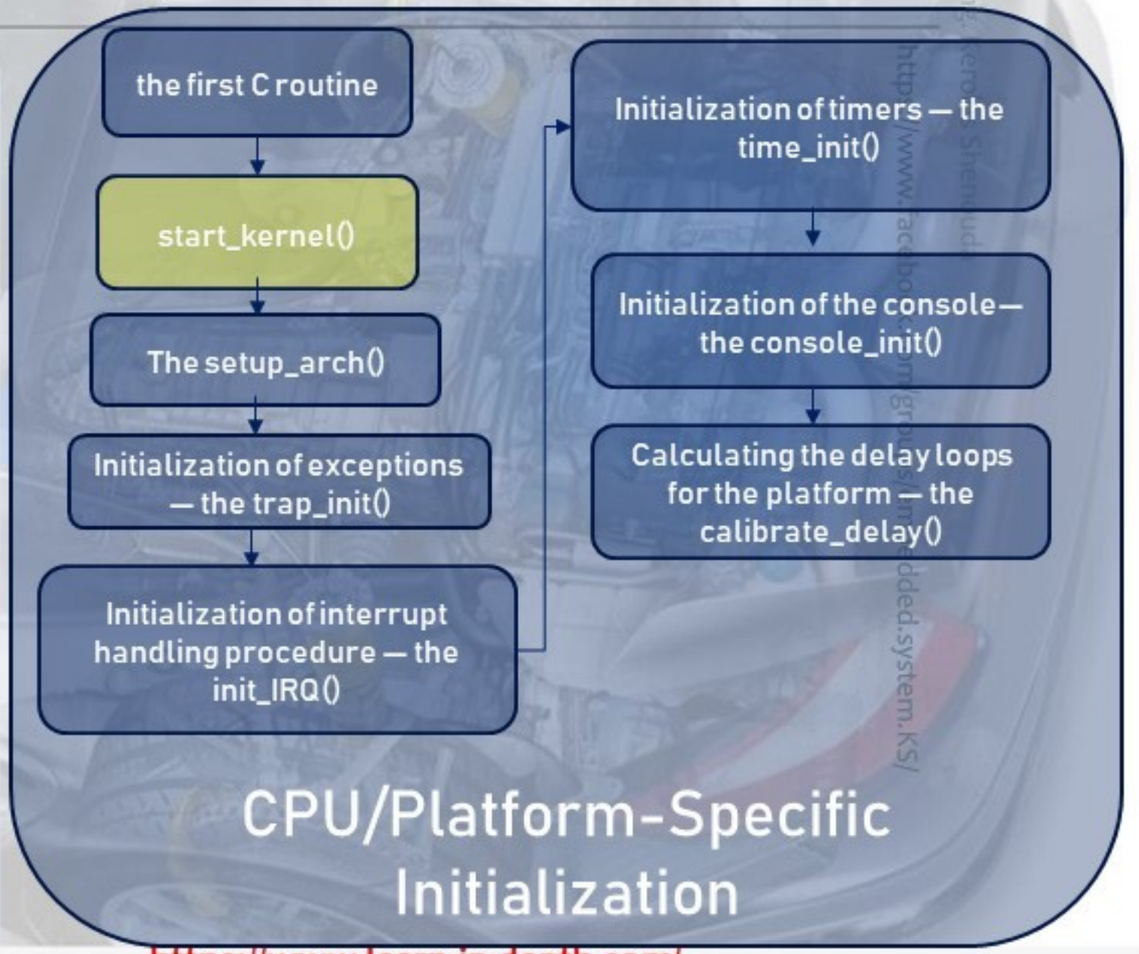
"Activate the first processor

process the device-tree

setup page tables and start virtual memory

All timestamps before are [0.000000]

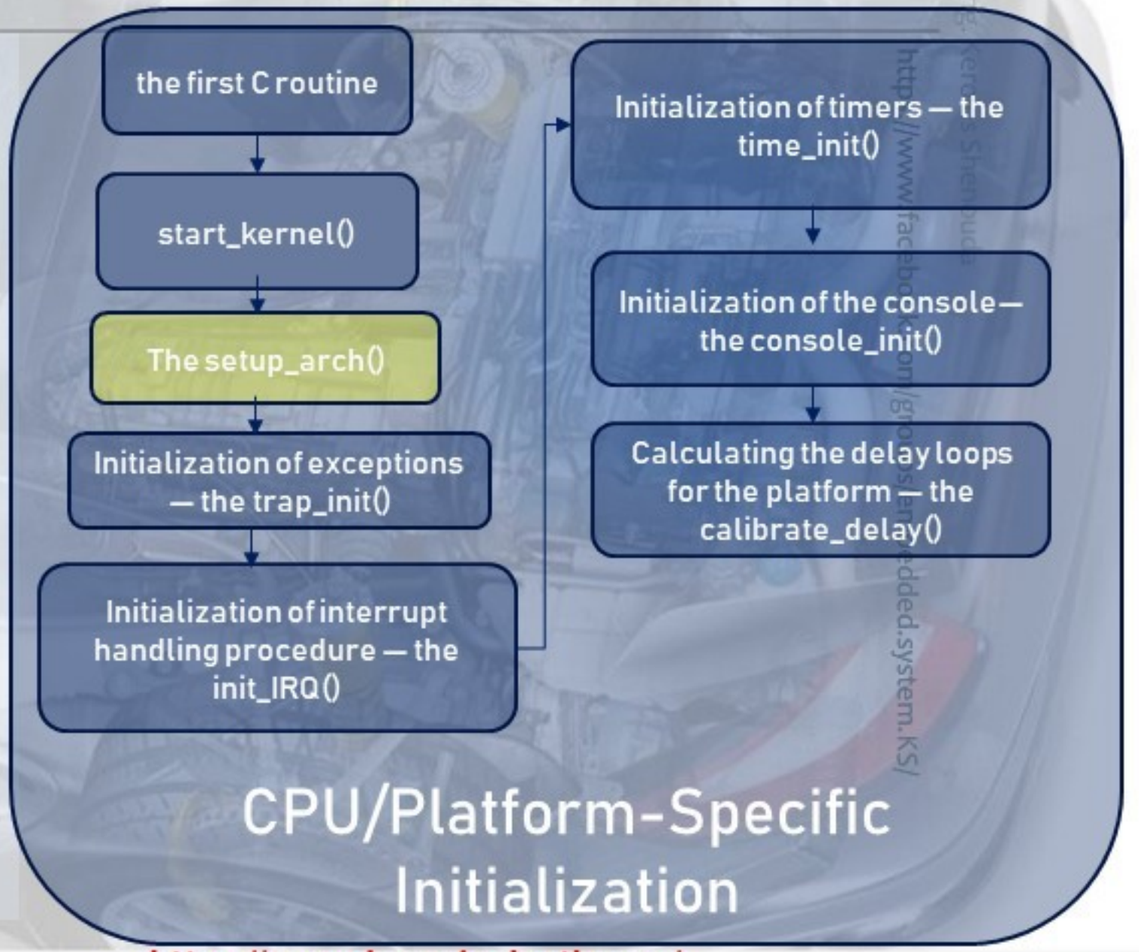
start userspace



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

The setup_arch() function

- This function does the platform- and CPUspecific initialization so that the rest of the initialization can be invoked safely
- It is responsible on:
  - Recognizing the processor.
  - Recognizing the board.
  - Analysis of command-line parameters passed to the kernel.
  - Identifying the ram disk.
  - Calling the bootmem functions.
  - Calling the paging initialization function



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

#LEARN IN DEPTH

#Be professional in embedded system

setup_arch() function

linux-test/arch/arm/kernel/setup.c - Eclipse Platform

Debug

- linux-test Debug [C/C++ Attach to Application]
  - vmlinux
    - Thread #1 1 (CPU#0 [running]) (Suspended : Breakpoint)
      - setup_arch() at setup.c:1,077 0x80a03384
      - start_kernel() at main.c:597 0x80a00ab0
      - 0x0
  - arm-none-eabi-gdb (7.11.90.20160917)

0x0 | head.S | main.c | setup.c

```
    } else
        pr_info("CPU: All CPU(s) started in SVC mode.\n");
#endif
}

void __init setup_arch(char **cmdline_p)
{
    const struct machine_desc *mdesc;

    setup_processor();
    mdesc = setup_machine_fdt(__atags_pointer);
    if (!mdesc)
        mdesc = setup_machine_tags(__atags_pointer, __machine_arch_type);
    if (!mdesc) {
        early_print("\nError: invalid dtb and unrecognized/unsupported machine ID\n");
        early_print("  r1=0x%08x, r2=0x%08x\n", __machine_arch_type,
                    __atags_pointer);
        if (__atags_pointer)
            early_print("  r2[]=%*ph\n", 16,
                        phys_to_virt(__atags_pointer));
        dump_machine_table();
    }
```

embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_Labs/L

embedded_syste... | embedded_syste... | embedded_syste... | embedded_syste... | embedded_syste...

```
ERROR: can't get kernel image!
=> setenv bootargs  'rw earlyprintk loglevel=8 root=/dev/mmcblk0p2  console=ttyAMA0 dh
=> fatload mmc 0:1 0x60000000 zImage
4594264 bytes read in 3945 ms (1.1 MiB/s)
=> fatload mmc  0:1 0x63000000 vexpress-v2p-ca9.dtb
44143 bytes read in 31 ms (445.3 KiB/s)
=> bootz 0x60000000 - 0x63000000
Kernel image @ 0x60000000 [ 0x000000 - 0x461a58 ]
## Flattened Device Tree blob at 63000000
   Booting using the fdt blob at 0x63000000
   Loading Device Tree to 7fe80000, end 7fe8673e ... OK

Starting kernel ...
```


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

#LEARN IN DEPTH

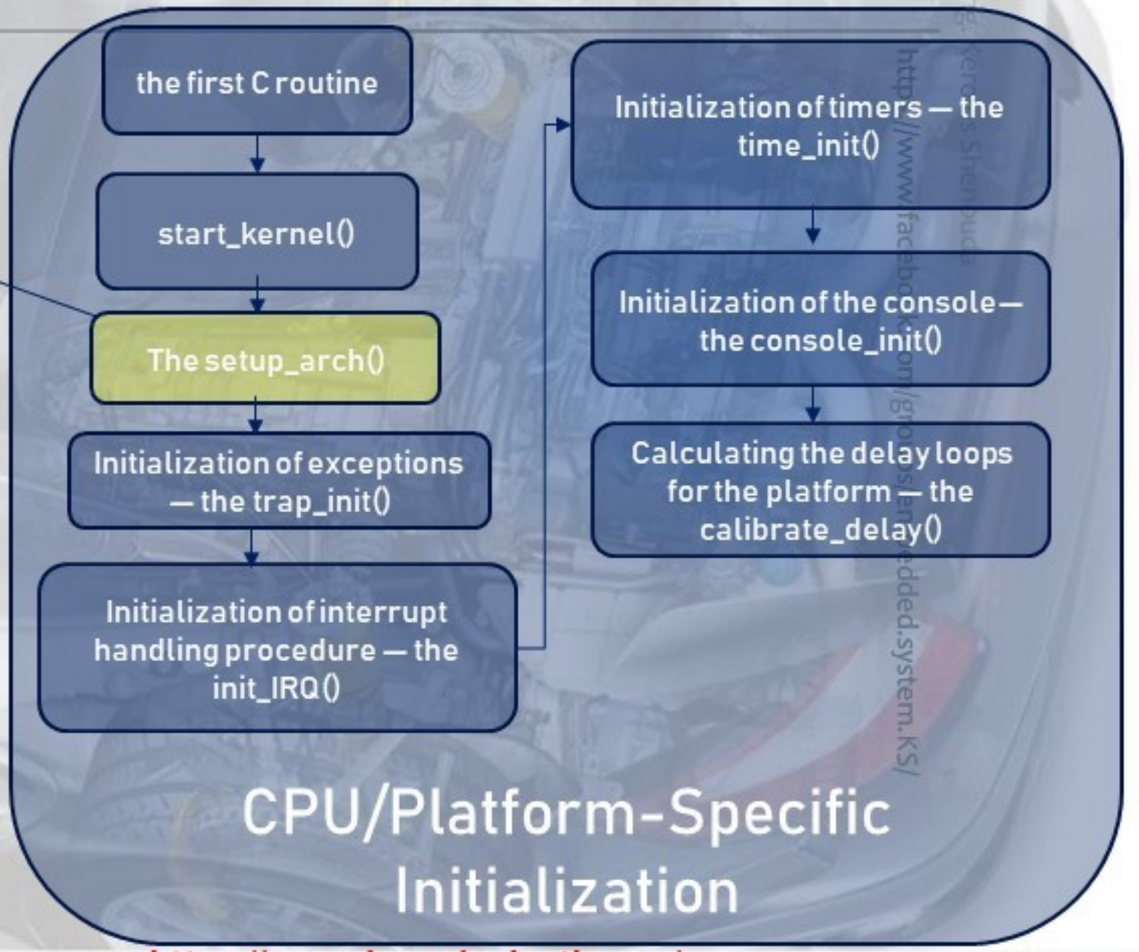
#Be professional in embedded system

The setup_arch() function

unflatten_device_tree()

__unflatten_device_tree() function and converts it **into a tree structure of node_node**.

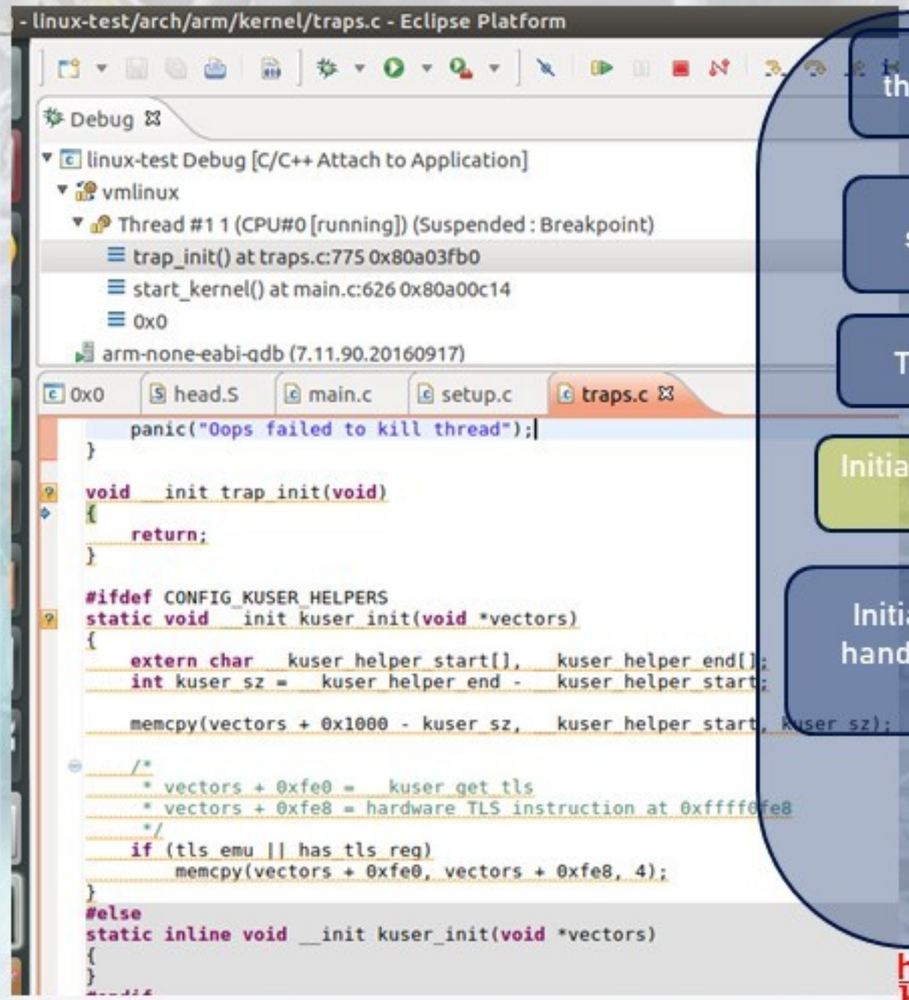
```
static void __unflatten_device_tree()
{
  ...
        /* First pass, scan for size */
        start = 0;
        size = (unsigned long)unflatten_dt_node(blob, NULL, &start, NULL, NULL, 0, true);
        size = ALIGN(size, 4);
  ...
        /* Second pass, do actual unflattening */
        start = 0;
        unflatten_dt_node(blob, mem, &start, NULL, mynodes, 0, false);
  ...
}
```

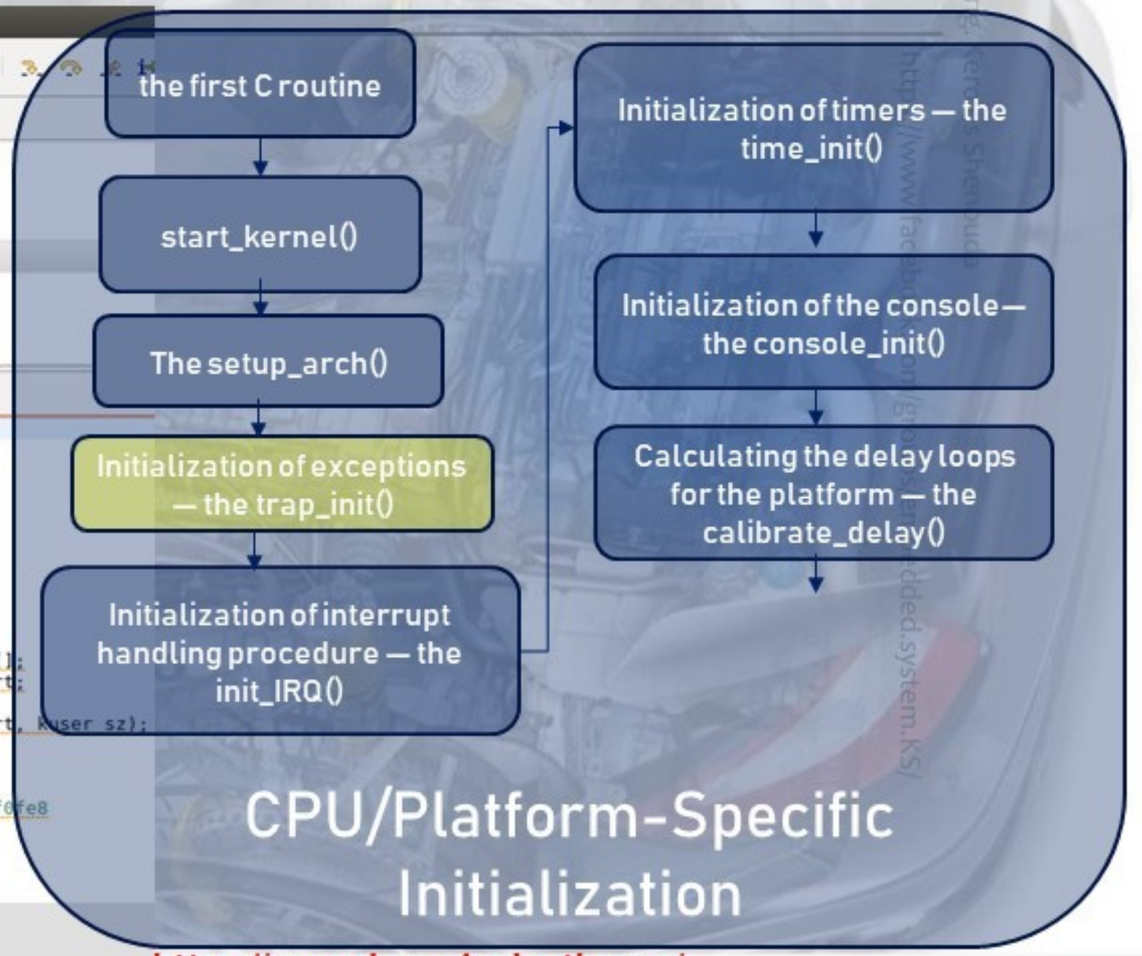


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Initialization of exceptions — the trap_init() function

- sets the kernel-specified exception handlers. Prior to this if an exception happens, the outcome is platform-specific. (For example, on some platforms the boot loader-specified exception handlers get invoked.)

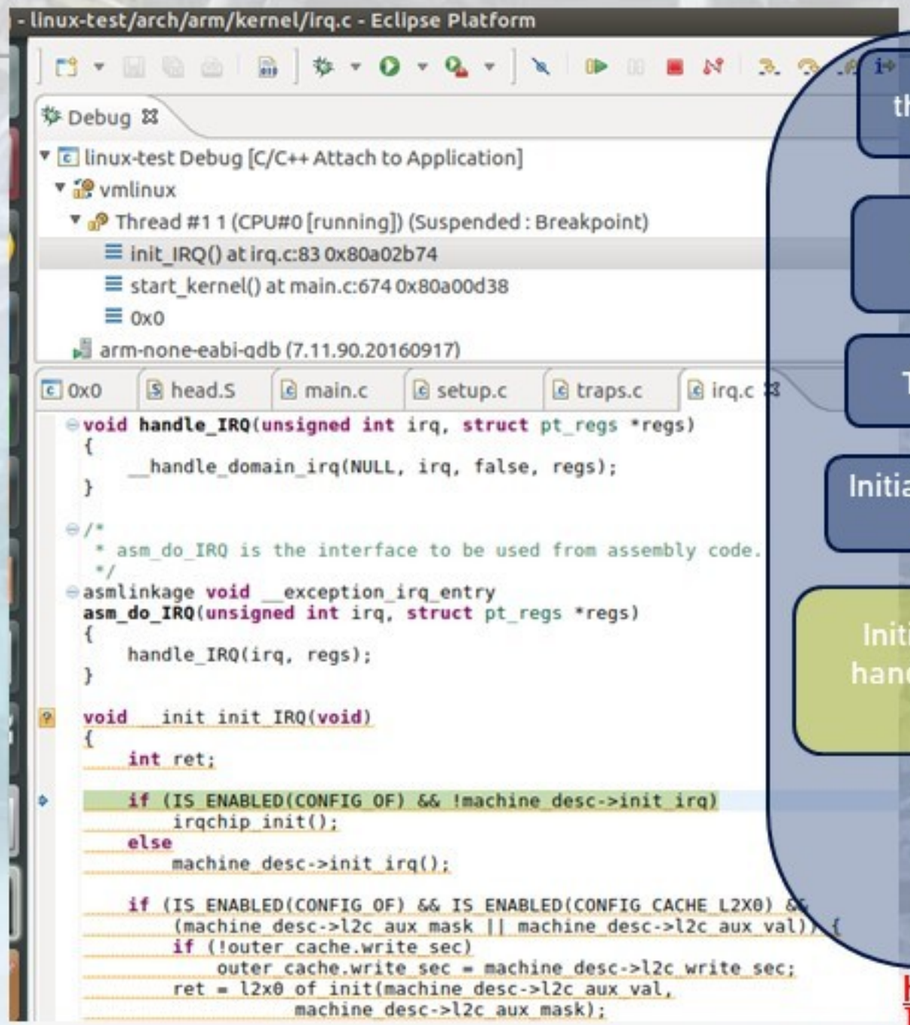


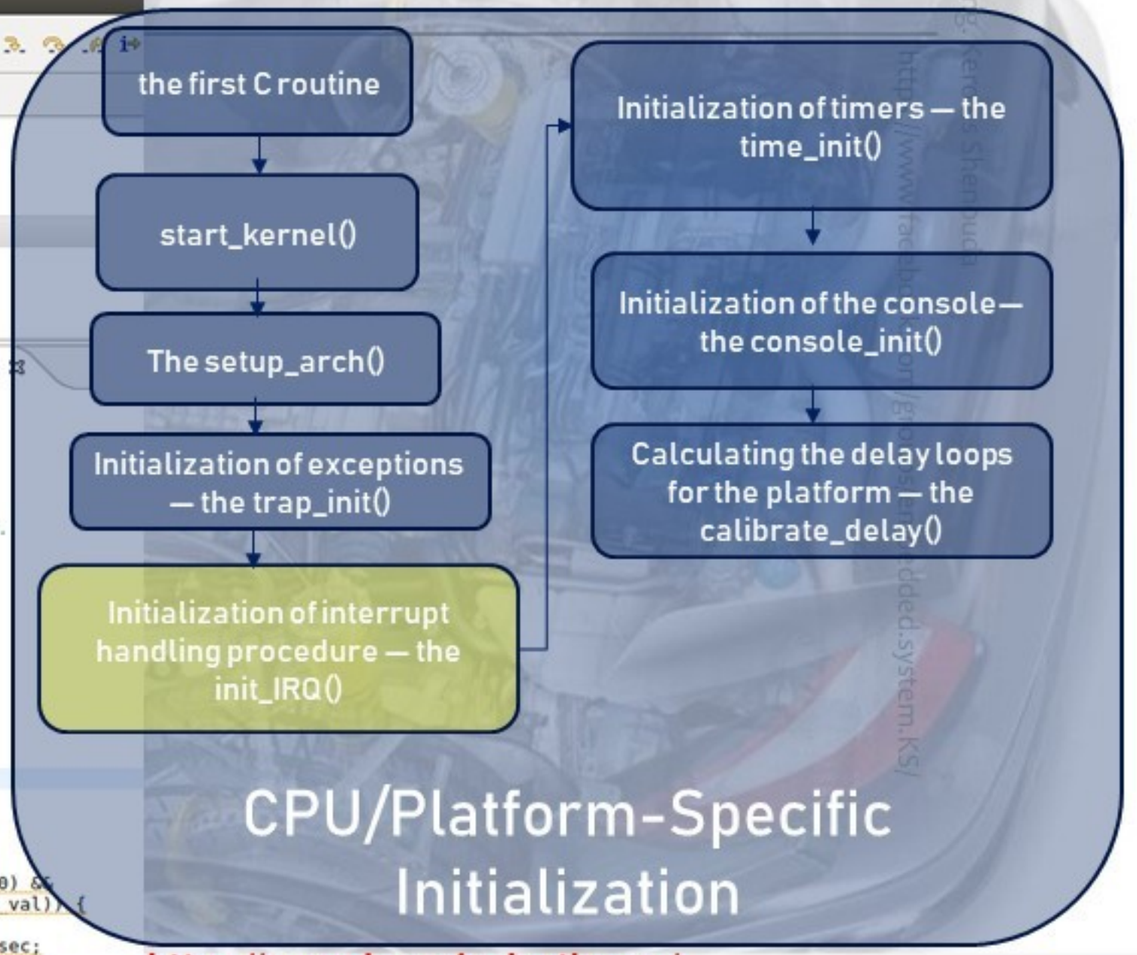


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Initialization of interrupt handling procedure — the init_IRQ()

- This function **initializes the interrupt controller** and the interrupt descriptors (these are data structures that are used by the BSP to route interrupts; more of this in the next chapter).
- Note that interrupts are not enabled at this point; this is the responsibility of the individual, drivers owning the interrupt lines to enable them during their initialization which is called later. (For example, the timer initialization would make sure that the timer interrupt line is enabled.)

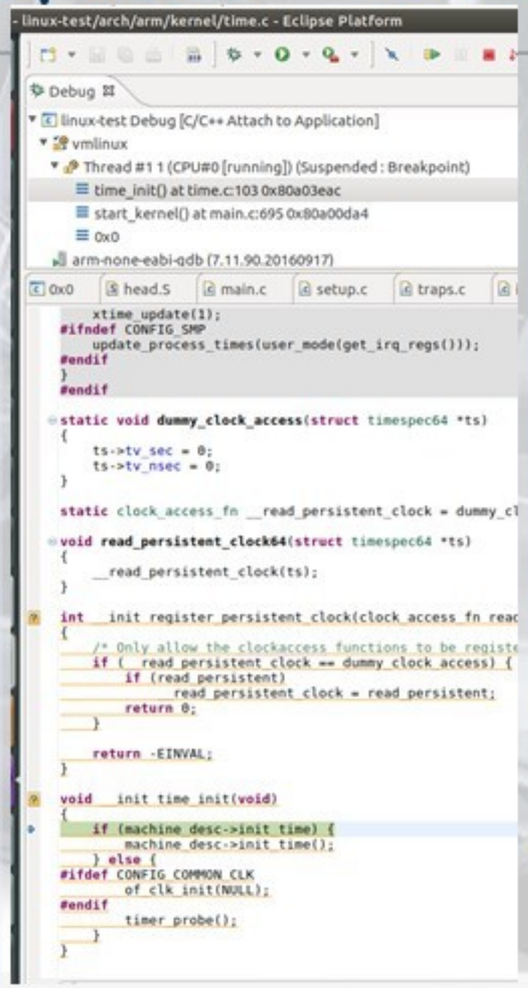


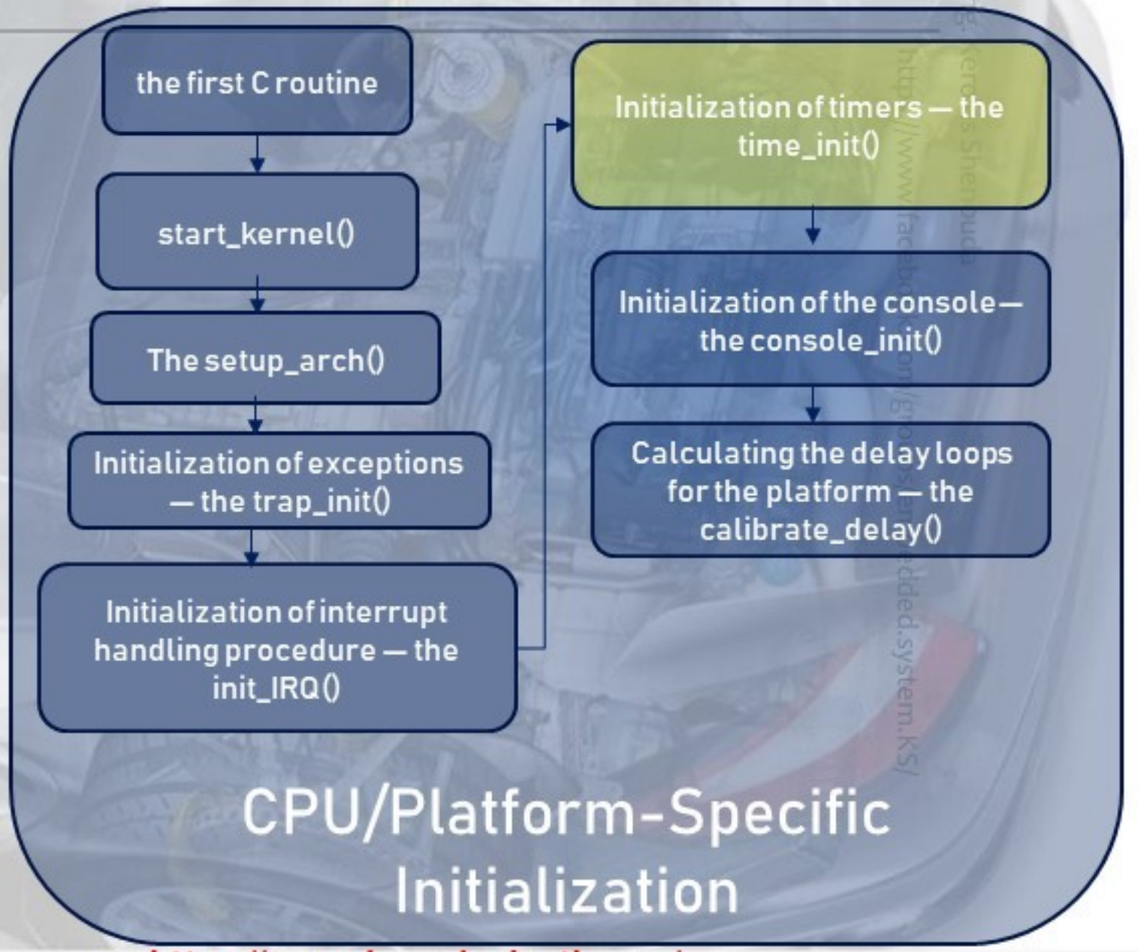


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Initialization of exceptions — the trap_init() function

- This function initializes the timer tick hardware so that the system starts producing the periodic tick, which is the system heartbeat

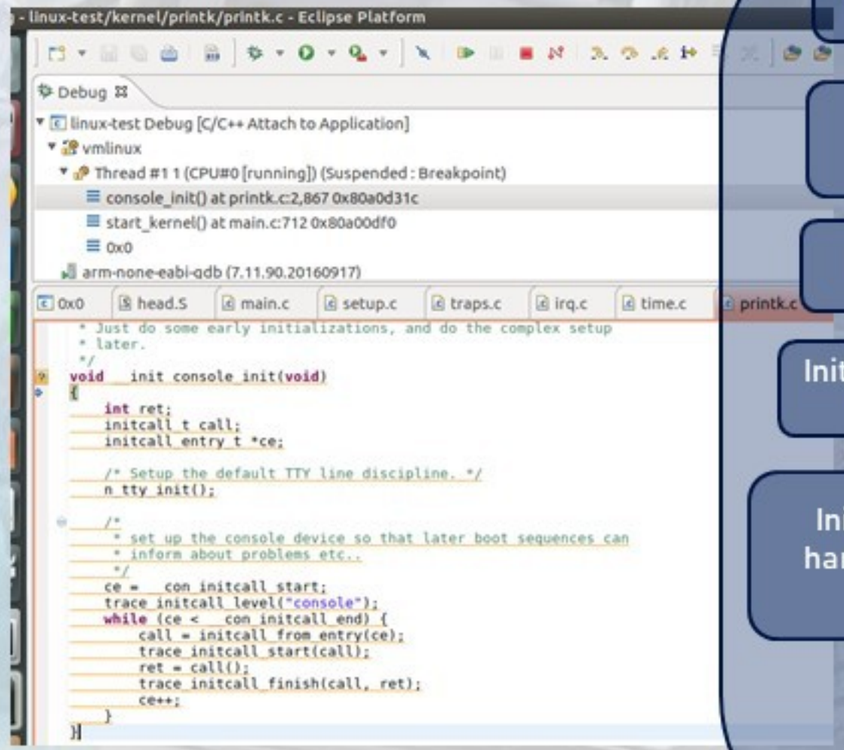


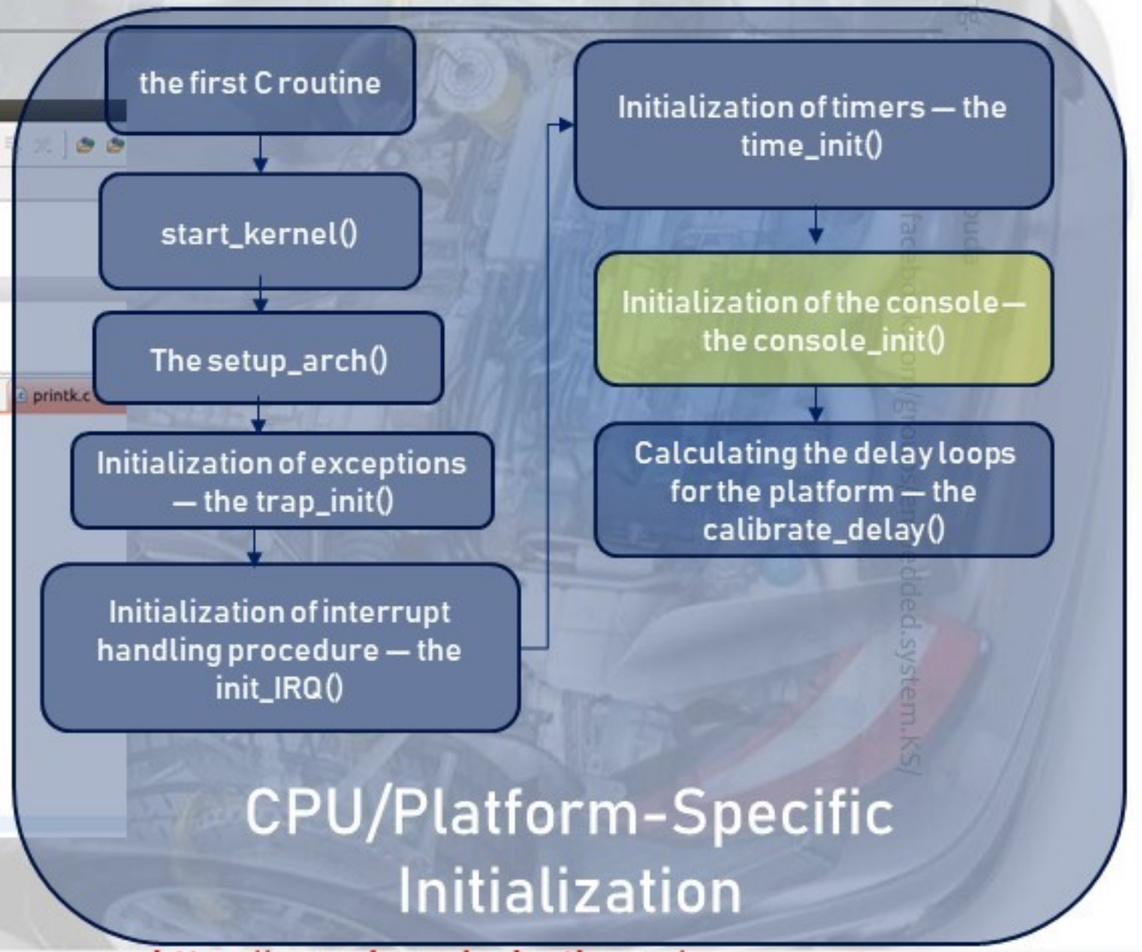


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Initialization of the console—the console_init() function

- This function does the initialization of the serial device as a console.
- Once the console is up, all the start-up messages appear on the screen.
- To print a message from the kernel, **the printk() function has to be used.**
- (printk() is a very powerful function as it can be called from anywhere, even from interrupt handlers.)

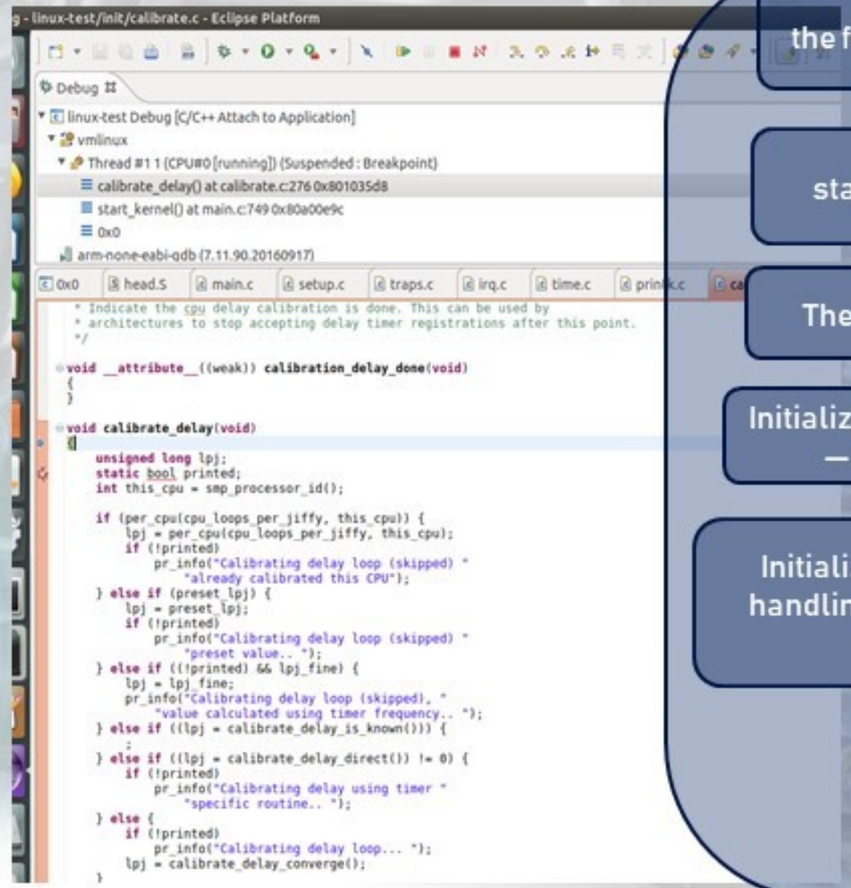


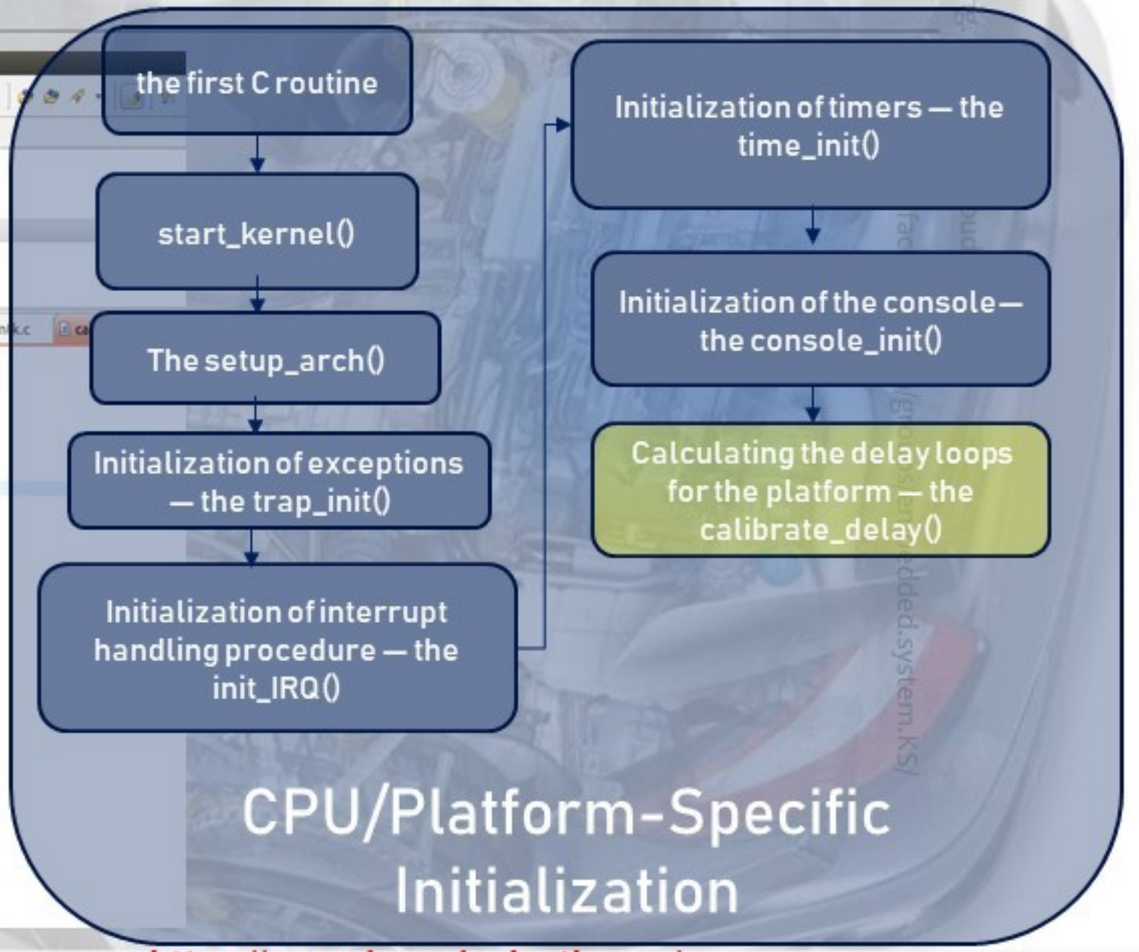


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

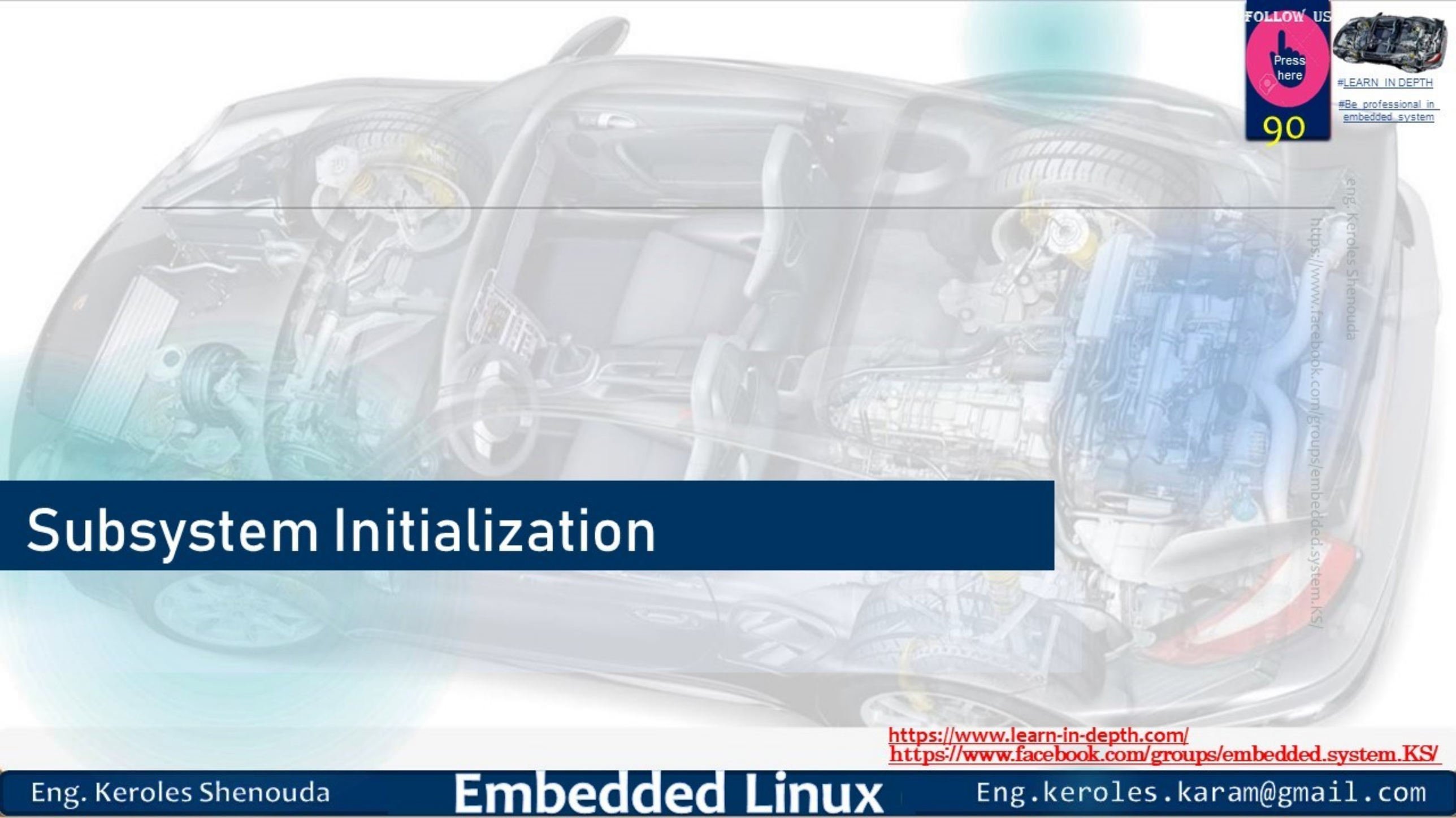
Calculating the delay loops for the platform — the calibrate_delay()

- This function is used to implement microdelays within the kernel using the udelay() function.
- The udelay() function spins for a few cycles for the microseconds specified as the argument.
- For udelay to work, the number of clock cycles per microsecond needs to be known by the kernel. This is exactly done by this function;
- Note that the working of this depends on the timer interrupt.





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



Subsystem Initialization

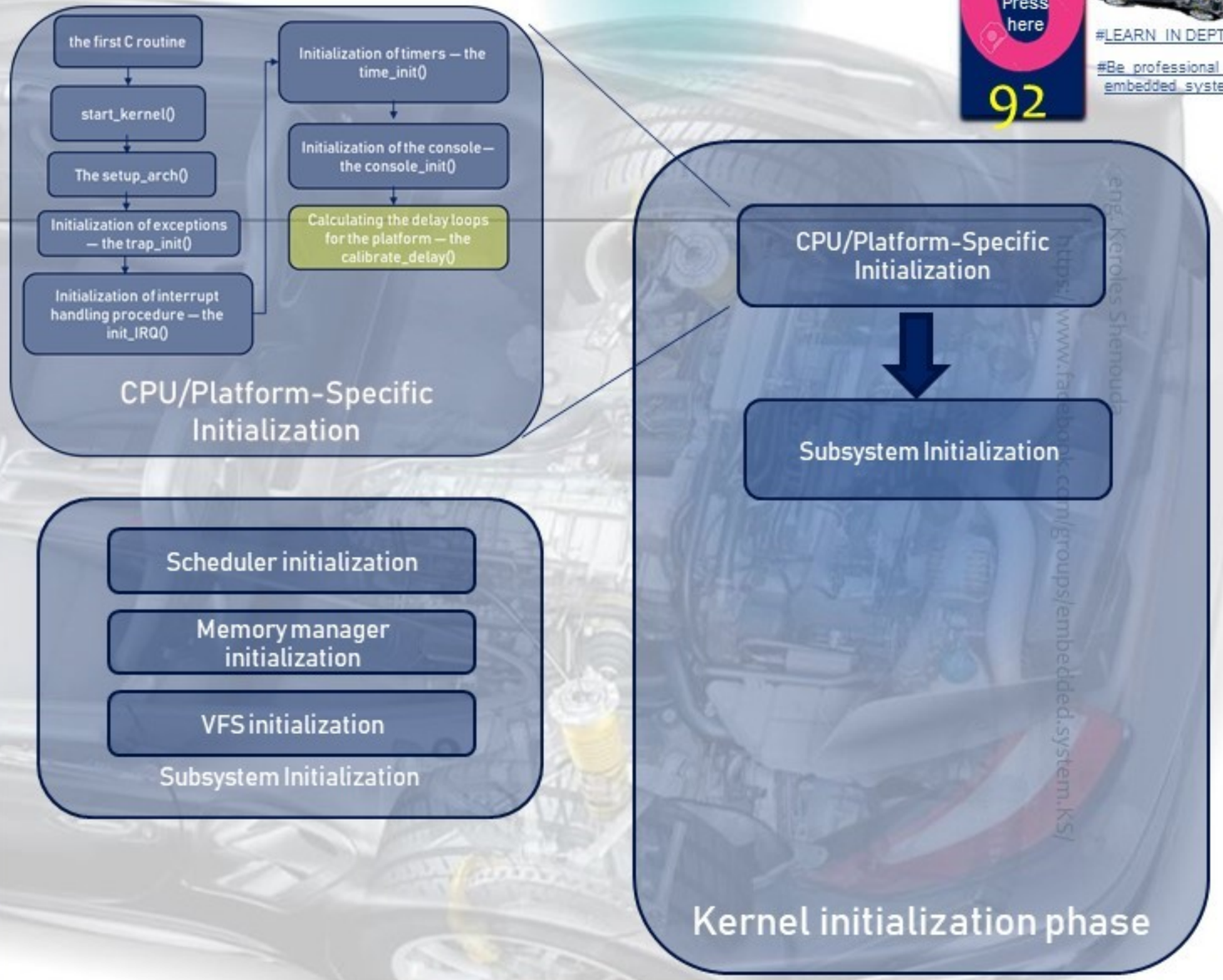
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Subsystem Initialization

- This includes
  - Scheduler initialization
  - Memory manager initialization
  - VFS initialization
- Note that most of the subsystem initialization is done in the **start_kernel()** function.
  - At the end of this function, the kernel creates another process, the **init** process, to do the rest of the initialization (driver initialization, initcalls, mounting the root file system, and jumping to user space) and the current process becomes the idle process with process id of 0.

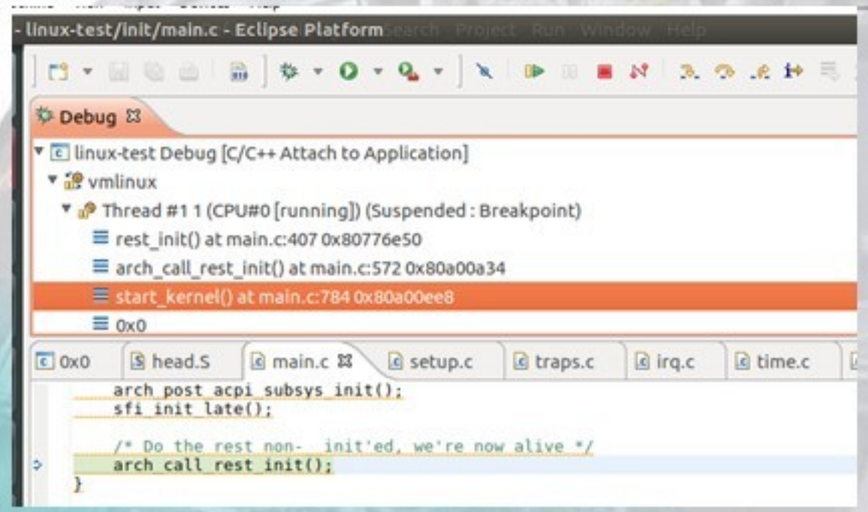
Subsystem Initialization

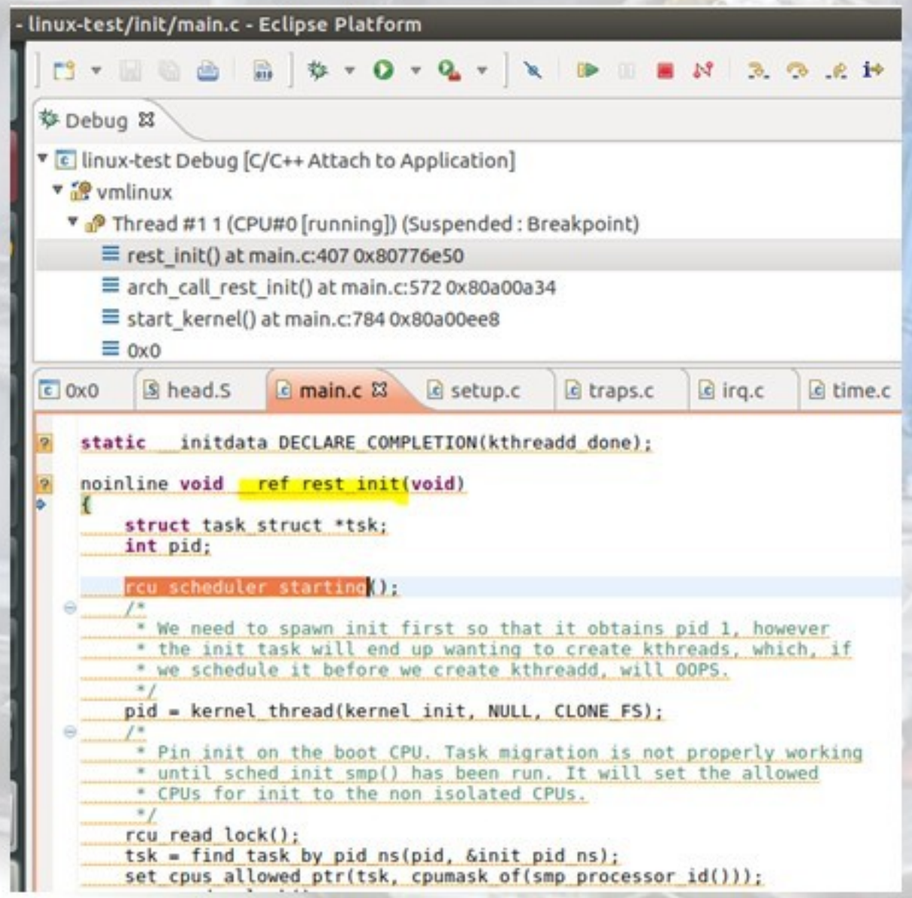
- that most of the subsystem initialization is done in the **start_kernel**() function.
- At the end of this function, the kernel creates another process, the **init process**, to do the rest of the initialization
  - (driver initialization,
  - initcalls,
  - mounting the root file system,
  - and jumping to user space)
- and the current process becomes the idle process with process id of 0



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

Subsystem Initialization Cont.





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

#LEARN IN DEPTH

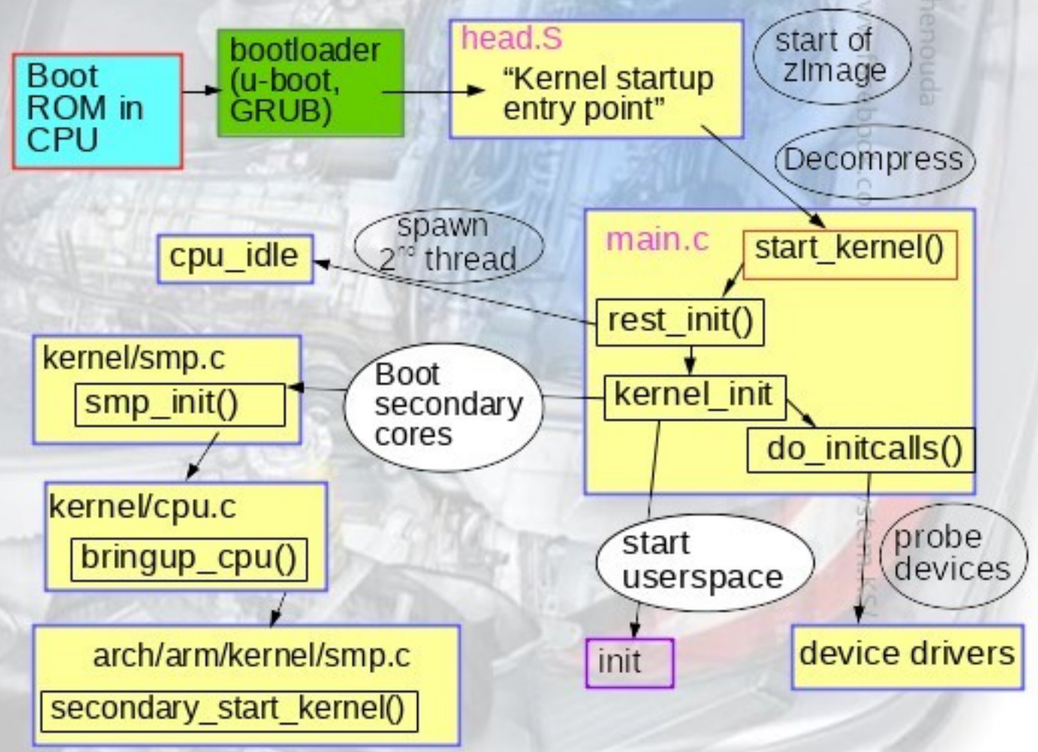
#Be professional in embedded system

Driver Initialization

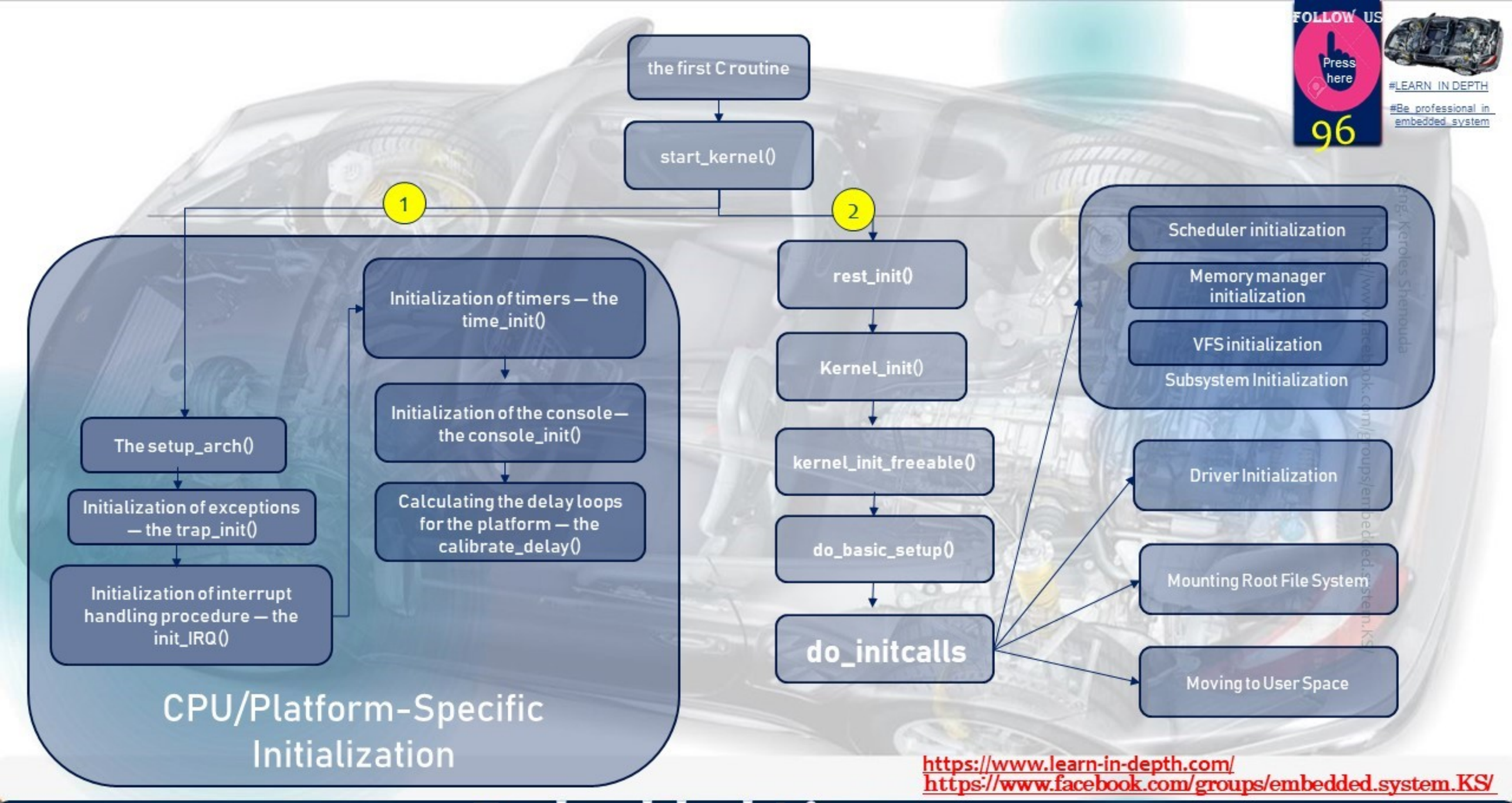
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

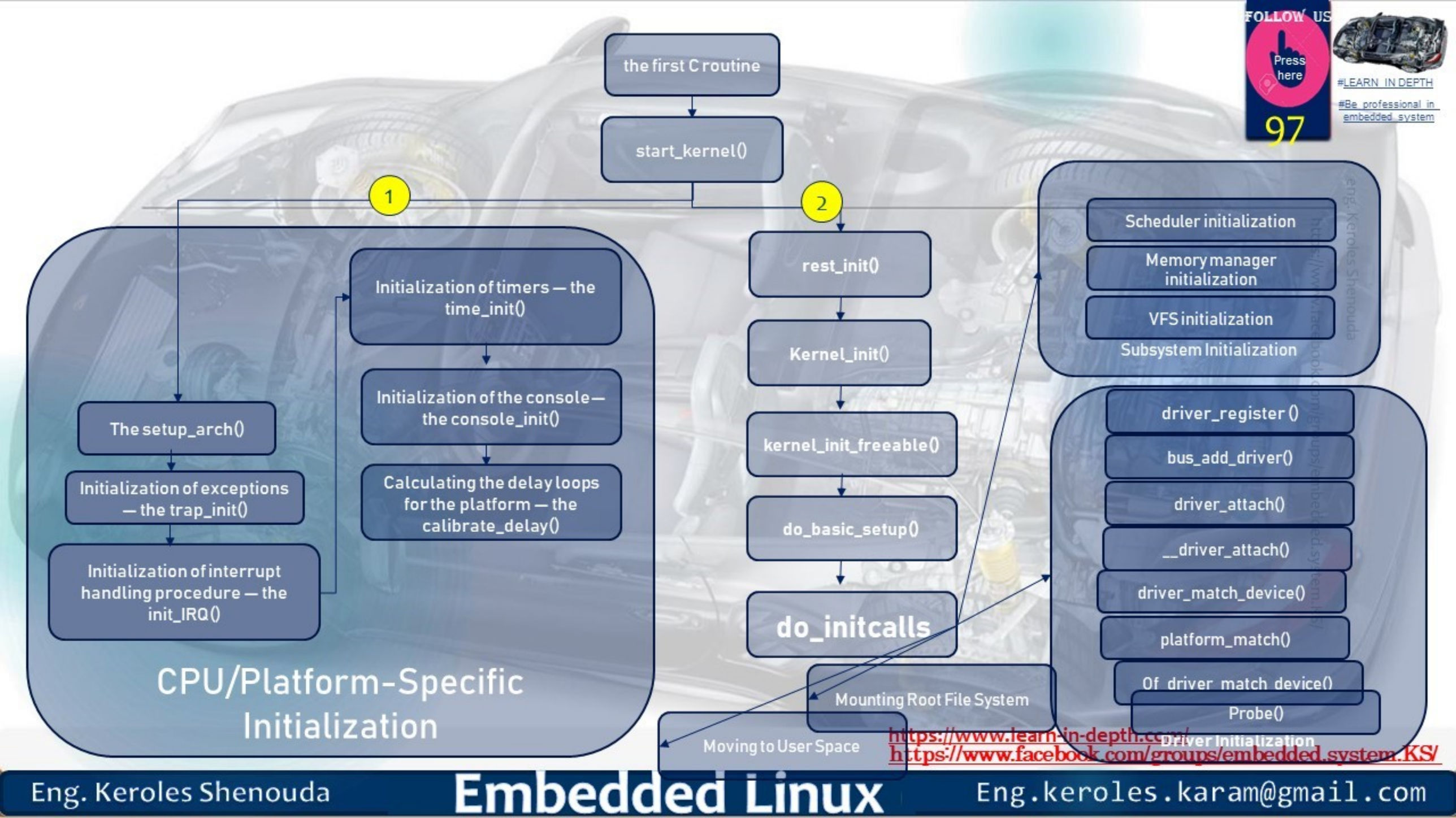
Driver Initialization

- The driver initialization is done after the process and memory management is up.
- It gets done in the context of the **init** process.



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


FOLLOW US
Press here
96
#LEARN IN DEPTH
#Be professional in embedded system
the first C routine
start_kernel()
1
2
The setup_arch()
Initialization of exceptions — the trap_init()
Initialization of interrupt handling procedure — the init_IRQ()
Initialization of timers — the time_init()
Initialization of the console— the console_init()
Calculating the delay loops for the platform — the calibrate_delay()
CPU/Platform-Specific Initialization
rest_init()
Kernel_init()
kernel_init_freeable()
do_basic_setup()
do_initcalls
Scheduler initialization
Memory manager initialization
VFS initialization
Subsystem Initialization
Driver Initialization
Mounting Root File System
Moving to User Space
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/
Eng. Keroles Shenouda
Embedded Linux
Eng.keroles.karam@gmail.com


the first C routine
start_kernel()
1
2
The setup_arch()
Initialization of exceptions — the trap_init()
Initialization of interrupt handling procedure — the init_IRQ()
Initialization of timers — the time_init()
Initialization of the console— the console_init()
Calculating the delay loops for the platform — the calibrate_delay()
CPU/Platform-Specific Initialization
rest_init()
Kernel_init()
kernel_init_freeable()
do_basic_setup()
do_initcalls
Scheduler initialization
Memory manager initialization
VFS initialization
Subsystem Initialization
driver_register ()
bus_add_driver()
driver_attach()
__driver_attach()
driver_match_device()
platform_match()
Of_driver_match_device()
Probe()
Driver Initialization
Mounting Root File System
Moving to User Space
FOLLOW US
Press here
97
#LEARN_IN DEPTH
#Be_professional_in_embedded_system
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/
Eng. Keroles Shenouda
Embedded Linux
Eng.keroles.karam@gmail.com

matching equipment

- Track driver_register() function, driver_register() --> bus_add_driver() --> driver_attach() --> __driver_attach
- Look for the matching device in driver_match_device().
- If the match is successful, execute the **driven probe** function.
- driver_match_device() will eventually call the platform's matching function platform_match():

```
static int __driver_attach(struct device *dev, void *data)
{
        struct device_driver *drv = data;
        if (!driver_match_device(drv, dev))
                return 0;

        if (dev->parent)        /* Needed for USB */
                device_lock(dev->parent);
        device_lock(dev);
        if (!dev->driver)
                driver_probe_device(drv, dev);
        device_unlock(dev);
        if (dev->parent)
                device_unlock(dev->parent);
        return 0;
}
```

matching equipment

- As you can see from the code, platform_match() will match in a variety of ways:
- Of_driver_match_device will be compared with the compatible attribute in the device according to the compatible attribute in the driver of_match_table.
- Second, call acpi_driver_match_device() to match.
- If the first two methods do not match, the final comparison between the device and the driver's name string is the same

```
static int platform_match(struct device *dev, struct device_driver *drv)
{
        struct platform_device *pdev = to_platform_device(dev);
        struct platform_driver *pdrv = to_platform_driver(drv);

        /* When driver_override is set, only bind to the matching
driver */
        if (pdev->driver_override)
                return !strcmp(pdev->driver_override, drv-
>name);

        /* Attempt an OF style match first */
        if (of_driver_match_device(dev, drv))
                return 1;

        /* Then try ACPI style match */
        if (acpi_driver_match_device(dev, drv))
                return 1;

        /* Then try to match against the id table */
        if (pdrv->id_table)
                return platform_match_id(pdrv->id_table,
pdev) != NULL;

        /* fall-back to driver name match */
        return (strcmp(pdev->name, drv->name) == 0);
}
```

Then probe the driver

```
};

static int sii902x_probe(struct i2c_client *client,
                         const struct i2c_device_id *id)
{
    struct device *dev = &client->dev;
    unsigned int status = 0;
    struct sii902x *sii902x;
    u8 chipid[4];
    int ret;

    ret = i2c_check_functionality(client->adapter,
                                  I2C_FUNC_SMBUS_BYTE_DATA);
    if (!ret) {
        dev_err(dev, "I2C adapter not suitable\n");
        return -EIO;
    }
```

#LEARN IN DEPTH

#Be professional in embedded system

Driver Initialization

Bring up symmetric multiprocessing (SMP

- linux-test/init/main.c - Eclipse Platform

Debug

linux-test Debug [C/C++ Attach to Application]
vmlinux
Thread #1 1 (CPU#0 [running]) (Suspended : Breakpoint)
kernel_init() at main.c:1,106 0x80776f00
ret_from_fork() at entry-common.S:155 0x801010e8
arm-none-eabi-gdb (7.11.90.20160917)

0x0 | head.S | main.c | setup.c | traps.c | irq.c | time.

```
    free_initmem_default(POISON_FREE_INITMEM);
}

static int __ref kernel_init(void *unused)
{
    int ret;

    kernel_init_freeable();
    /* need to finish all async __init code before freeing the memory */
    async_synchronize_full();
    ftrace_free_init_mem();
    free_initmem();
    mark_readonly();

    /*
     * Kernel mappings are now finalized - update the userspace page-tabl
     * to finalize PTI.
     */
    pti_finalize();

    system_state = SYSTEM_RUNNING;
    numa_default_policy();

    rcu_end_inkernel_boot();

    if (ramdisk_execute_command) {
        ret = run_init_process(ramdisk_execute_command);
        if (!ret)
            return 0;
        pr_err("Failed to execute %s (error %d)\n",
               ramdisk_execute_command, ret);
    }

    /*
```

- linux-test/kernel/smp.c - Eclipse Platform

Debug

linux-test Debug [C/C++ Attach to Application]
vmlinux
Thread #1 1 (CPU#0 [running]) (Suspended : Breakpoint)
smp_init() at smp.c:579 0x80a10b2c
kernel_init_freeable() at main.c:1,184 0x80a01028
kernel_init() at main.c:1,109 0x80776f08
ret_from_fork() at entry-common.S:155 0x801010e8

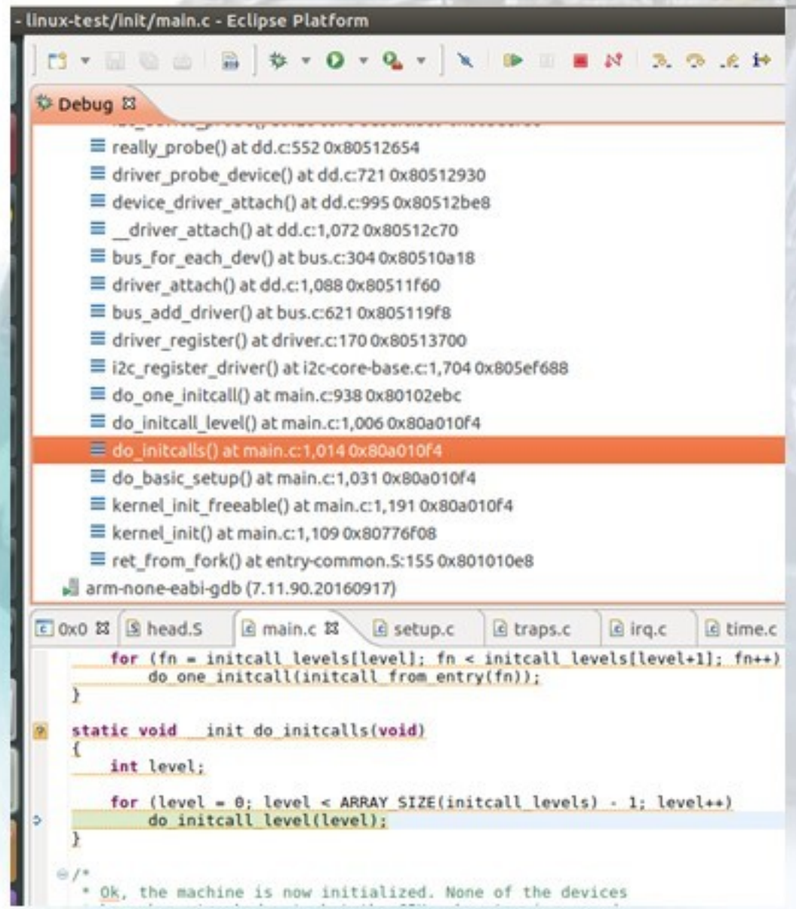
0x0 | head.S | main.c | setup.c | traps.c | irq.c | time.c

```
/* An arch may set nr_cpu_ids earlier if needed, so this would b
void __init setup_nr_cpu_ids(void)
{
    nr_cpu_ids = find_last_bit(cpumask_bits(cpu_possible_mask),N
}

/* Called by boot processor to activate the rest. */
void __init smp_init(void)
{
    int num_nodes, num_cpus;
    unsigned int cpu;

    idle_threads_init();
    cpuhp_threads_init();

    pr_info("Bringing up secondary CPUs ...\n");

    /* FIXME: This should be done in userspace --RR */
    for_each_present_cpu(cpu) {
        if (num_online_cpus() >= setup_max_cpus)
            break;
        if (!cpu_online(cpu))
            cpu_up(cpu);
    }
```

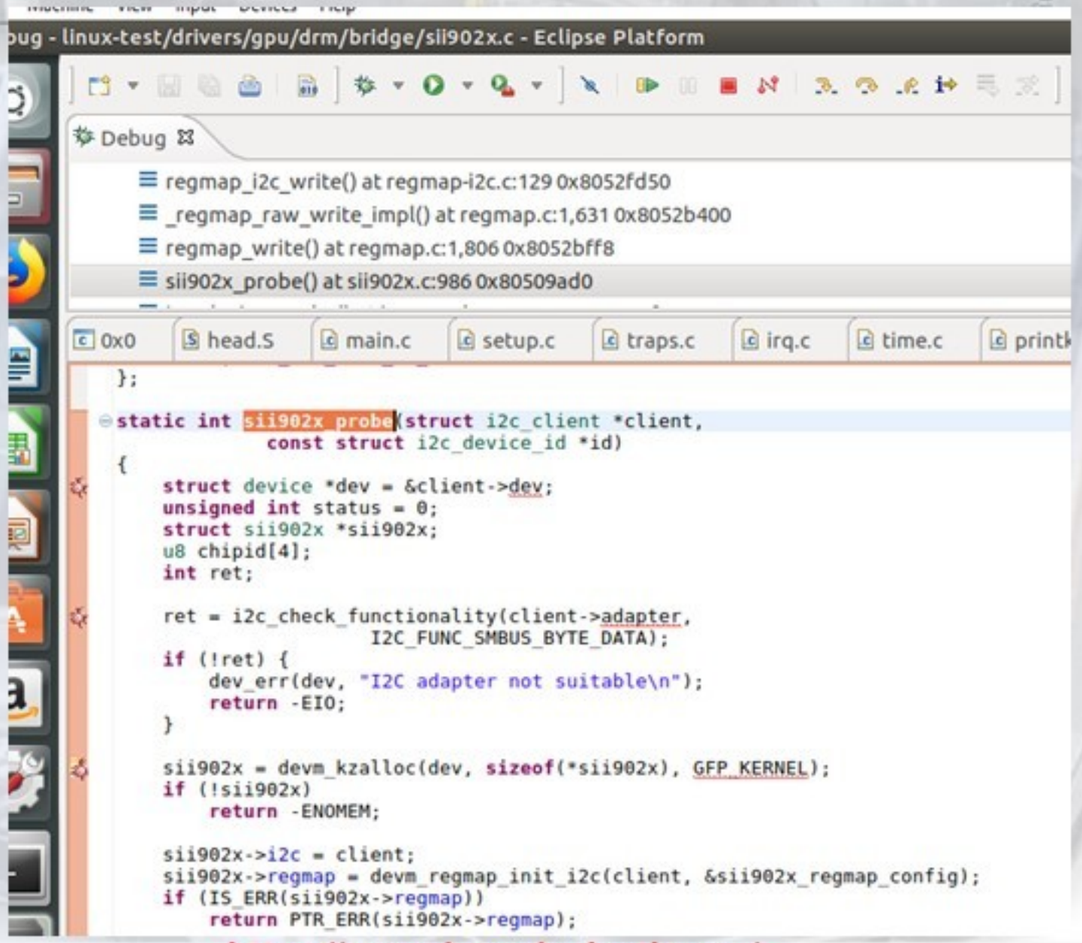
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_KS_labs/LABS/Linux_Labs/lab2

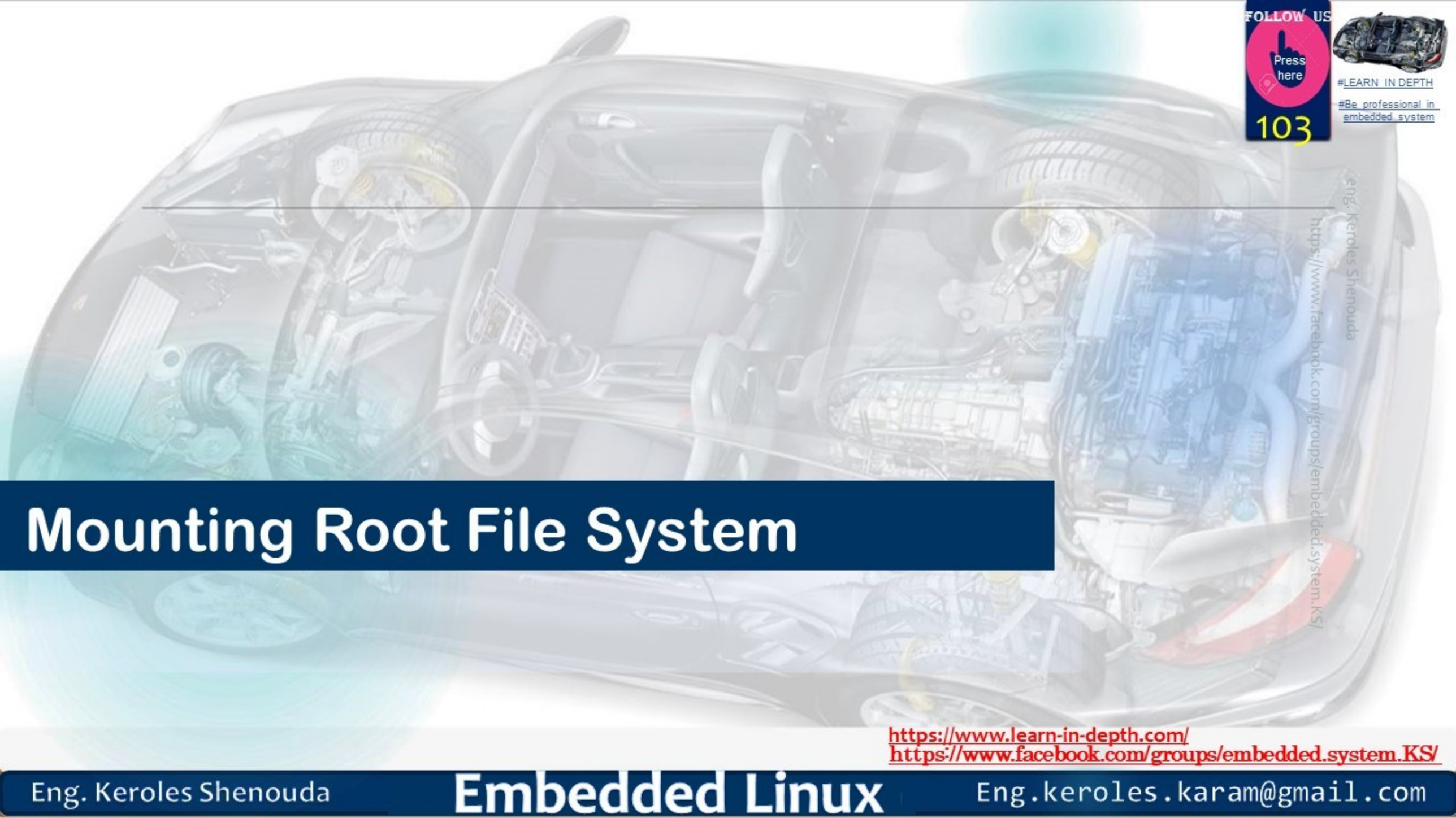
```
Calibrating local timer... 90.32MHz.
Calibrating delay loop... 559.10 BogoMIPS (lpj=2795520)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
CPU: Testing write buffer coherency: ok
CPU0: Spectre v2: using BPIALL workaround
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
Setting up static identity map for 0x60100000 - 0x60100060
rcu: Hierarchical SRCU implementation.
smp: Bringing up secondary CPUs ...
smp: Brought up 1 node, 1 CPU
SMP: Total of 1 processors activated (559.10 BogoMIPS).
CPU: All CPU(s) started in SVC mode.
devtmpfs: initialized
```

Driver Initialization Cont.



do_initcalls() will probe all the embedded linux device drivers





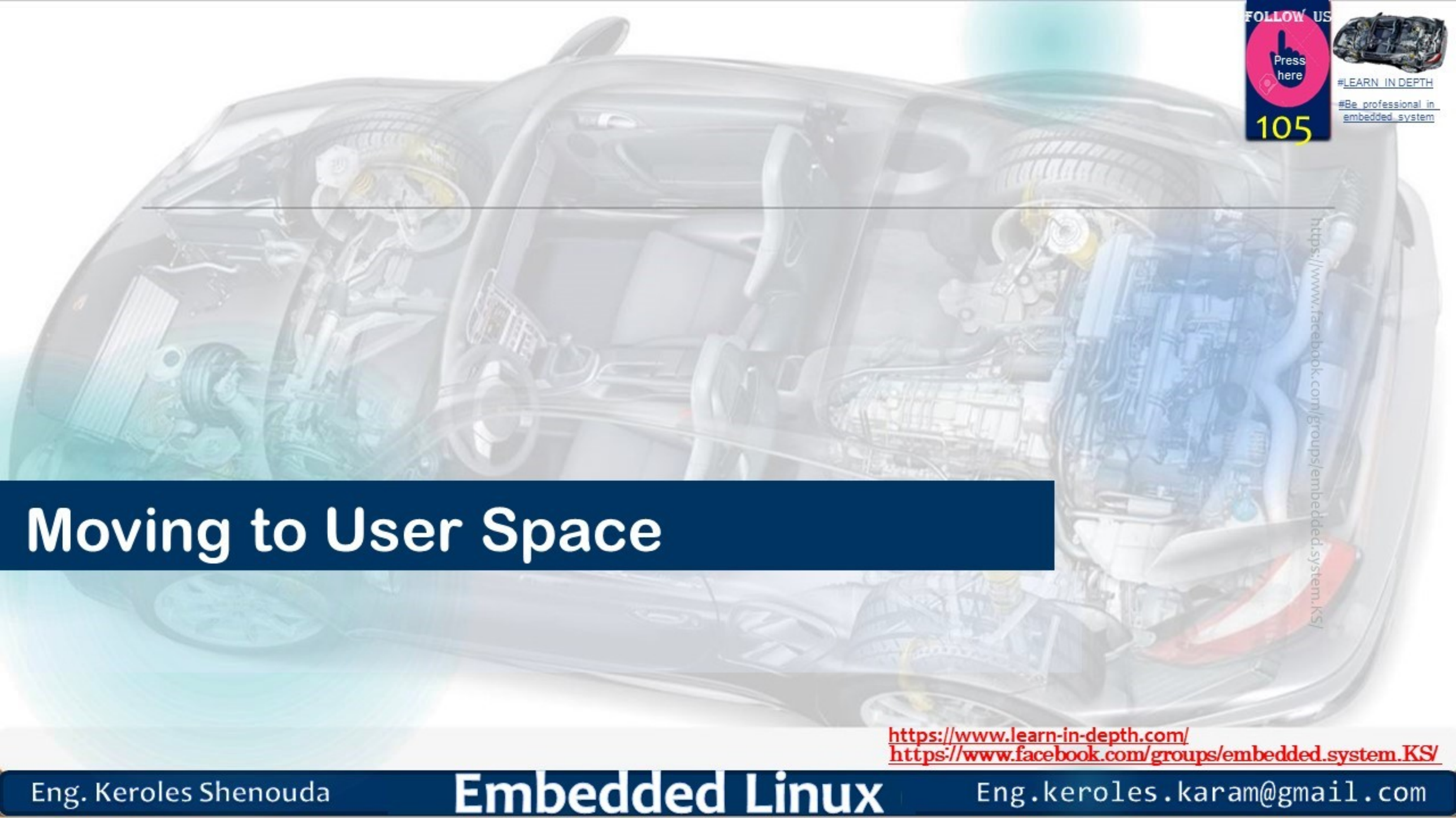
Mounting Root File System

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

Mounting Root File System

- Recall that the root file system is the master file system using which other file systems can be mounted.
- Its mounting marks an **important** process in the booting stage as the kernel can start its transition **to user space**
- The location of the RFS can be passed as **a command line argument** from **the boot loader** using the **boot loader tag** "root=".
- There are three kinds of root file systems that are normally used on embedded systems:
  - The initial ram disk
  - Network-based file system using NFS
  - Flash-based file system
  - SDCARD
- If the root file system is not mounted, the kernel will stall execution and enter the panic mode after logging the complaint on the console:
  - **Unable to mount root fs on device**

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

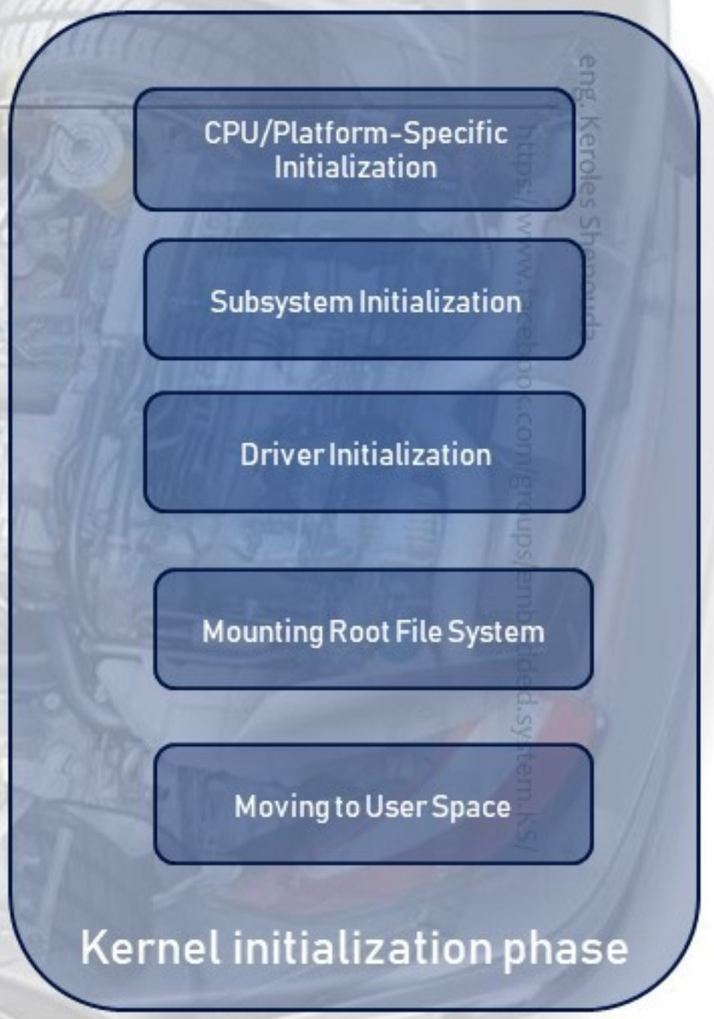


Moving to User Space

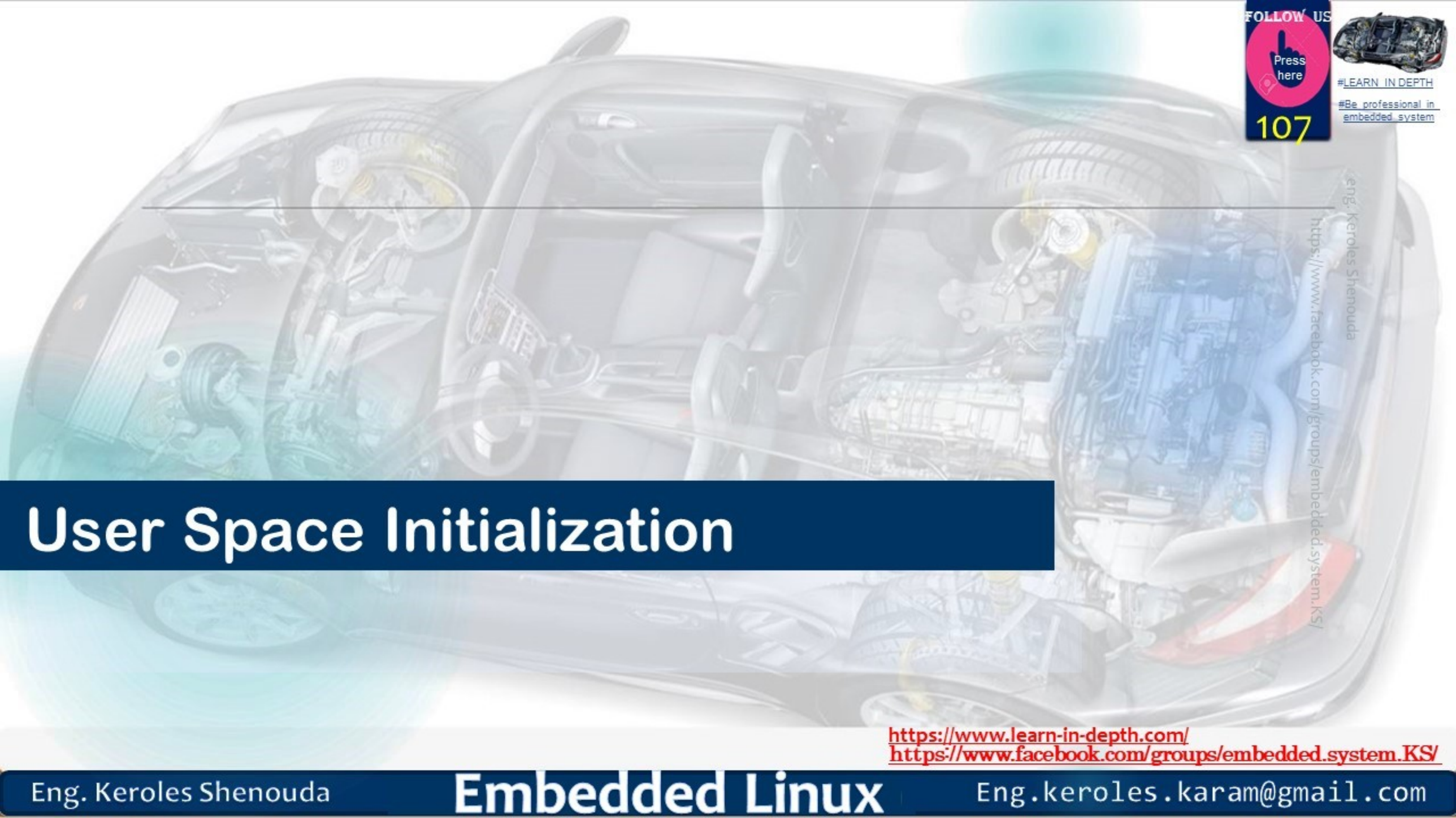
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Moving to User Space

- The kernel that is executing in the context of the init process jumps to the user space by overlaying itself (using execve) with the executable image of a special program also referred to as **init**.
- This executable normally resides in the root file system in the directory **/sbin**.
- Note that the user can specify the **init** program using a command line argument to the kernel.
- However, if the kernel is unable to load either the user-specified **init** program or the default one,
  - it enters the panic state after logging the complaint:
    - No init found. Try passing init= option to the kernel.



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



User Space Initialization

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

User Space Initialization

- The responsibility of the kernel ends with the transition to the init process. What the init process does and how it starts the services is dependent on the distribution.
- The /sbin/init Process and /etc/inittab
  - It **can never be killed**. Linux offers a signal called SIGKILL that can terminate execution of any process but it cannot kill the init process
- The **init process can be configured** on any system using the ***inittab*** file, which typically resides in the ***/etc directory***.
- init reads the inittab file and does the actions accordingly in a sequential manner.
- init also decides the system state known as run level.
  - A run level is a number that is passed as an argument to init
- The **process** that needs to be executed during system start-up. This is typically the file **/etc/rc.d/rc.sysinit** file.

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

The rc.sysinit File

- This file does the system initialization before the **services** are started.
- Typically this file does the following on an embedded system
  - Mount special file systems such as proc, ramfs
  - Create directories and links if necessary
  - Set the hostname for the system
  - Set up networking configuration on the system

Starting Services

- the script **/etc/rc.d/rc** is responsible for starting the services.
- A service is defined as a facility to control a system process.
- Using services, a process can be stopped, restarted, and its status can be queried.
- The services are normally organized into directories based on the run levels;
- depending on what run level is chosen the services are stopped or started.
- After performing the above steps, the **init** starts **a log-in program** on a TTY or runs a window manager on the graphics display (depending on the run level).

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/
- Detailed Linux device tree
- Device Tree for DummIES
- Device Tree Usage
- 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