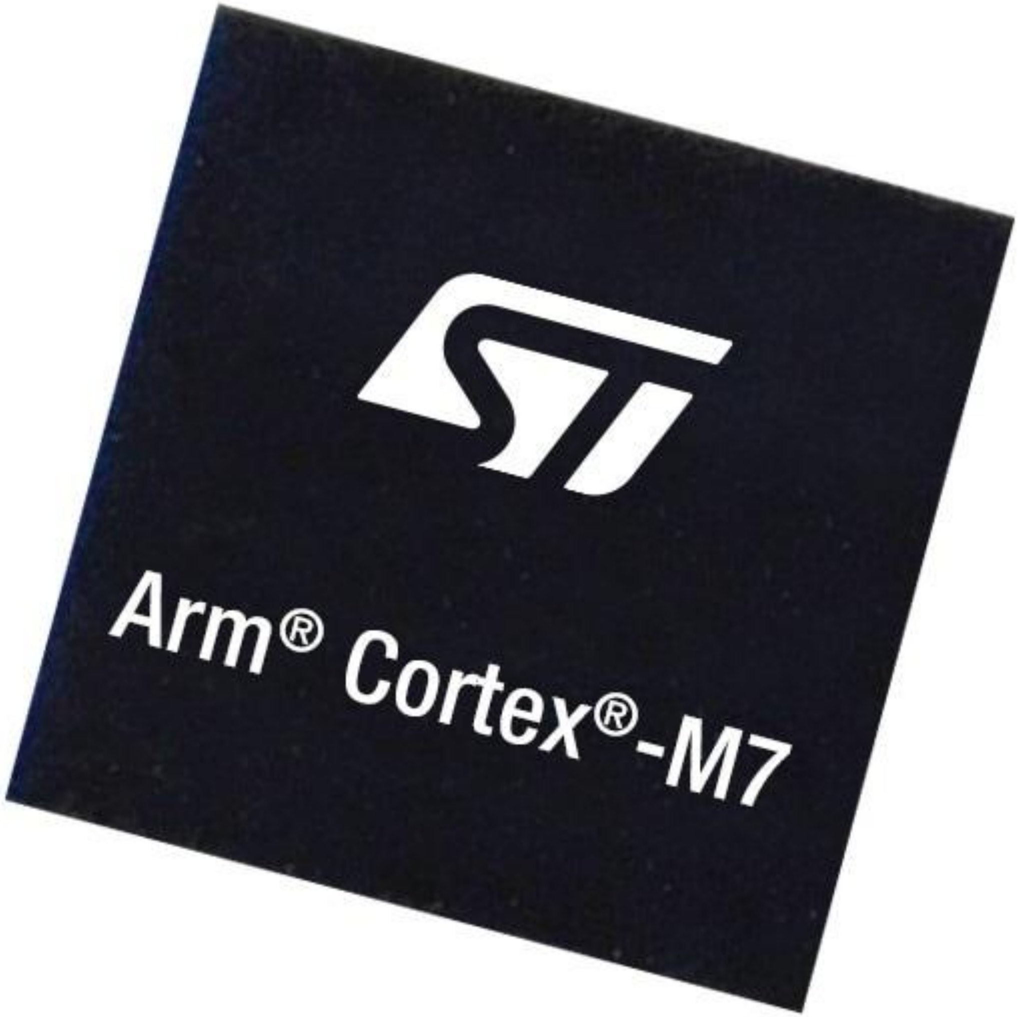
Booting and boot sequence in arm-based microcontrollers

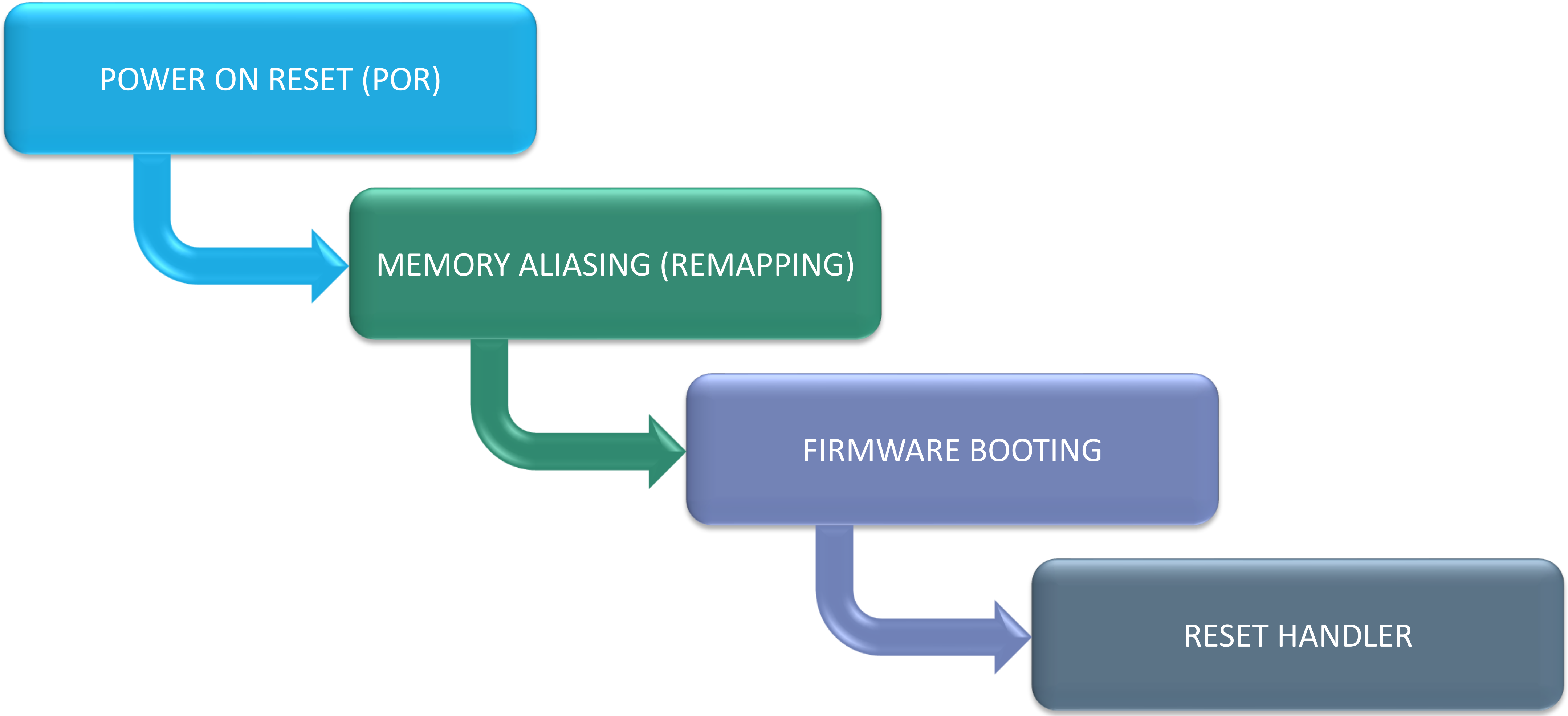
NEOCRUX
SYSTEMS

Definition of booting microcontroller

- Powering-up the microcontroller
- Checks the proper functioning of the hardware components of the microcontroller
- Initializes the system by loading the software components of the microcontroller
- Ensure the integrity of the system



Overview of boot sequence in arm-based microcontrollers



POWER ON RESET (POR)

MEMORY ALIASING (REMAPPING)

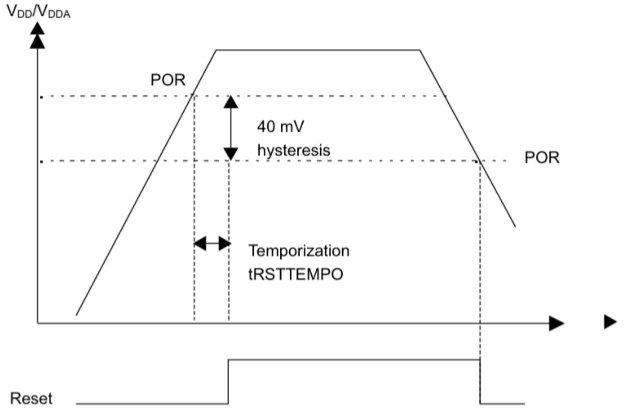
FIRMWARE BOOTING

RESET HANDLER

Power on reset (POR)

Is an integrated circuit in the microcontroller that:

- Detects stable power supply reaching the required thresholds and enter MCU to reset state
- Resets CPU, peripherals, memory controllers and clears registers, latches to default
- Waits until the clocks are configured and valid
- MCU remains in reset state until all the checks finish and everything work well

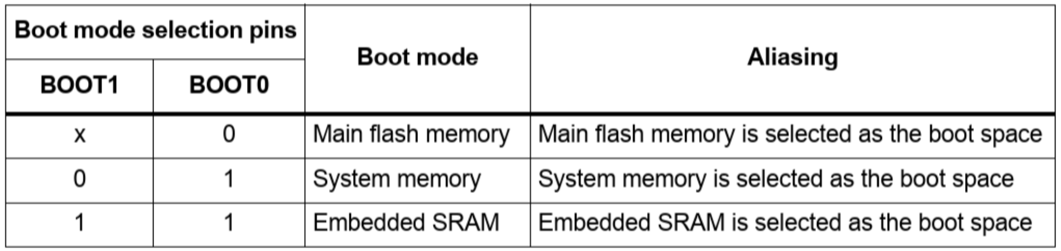


Symbol	Condition	Min	Typical	Max	Unit
V_{POR}	Falling edge	1.60	1.68	1.76	V
	Rising edge	1.64	1.72	1.80	V
$T_{RSTTEMPO}$		0.5	1.5	3.0	ms

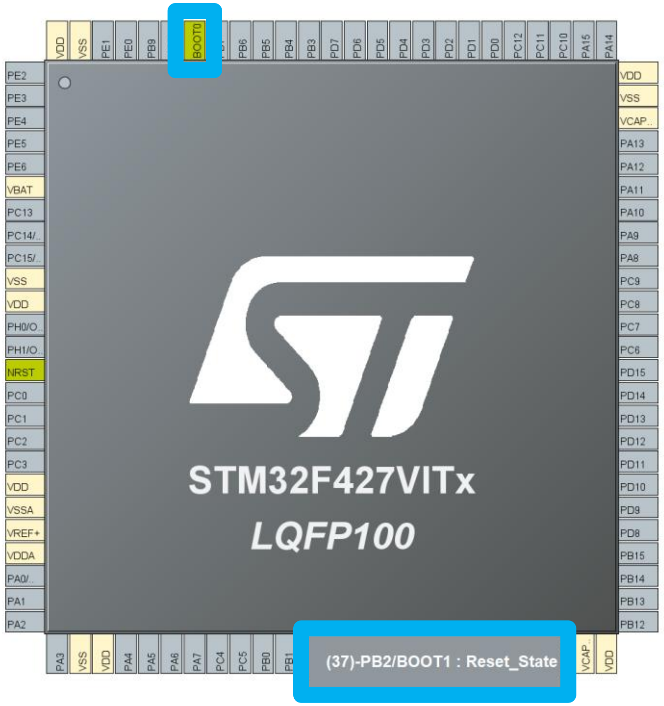
Example: STM32F427VIT6

MEMORY ALIASING (REMAPPING)

- After being powered, the processor points on address 0x00000000 (initial stack pointer)
- User can choose the boot mode (SRAM, FLASH, SYSTEM MEMORY) using pins BOOT0 and BOOT1
- Memories are mapped in other addresses, the remapper brings it to the initial stack pointer
- Helps to define the vector table and reads the correct memory content

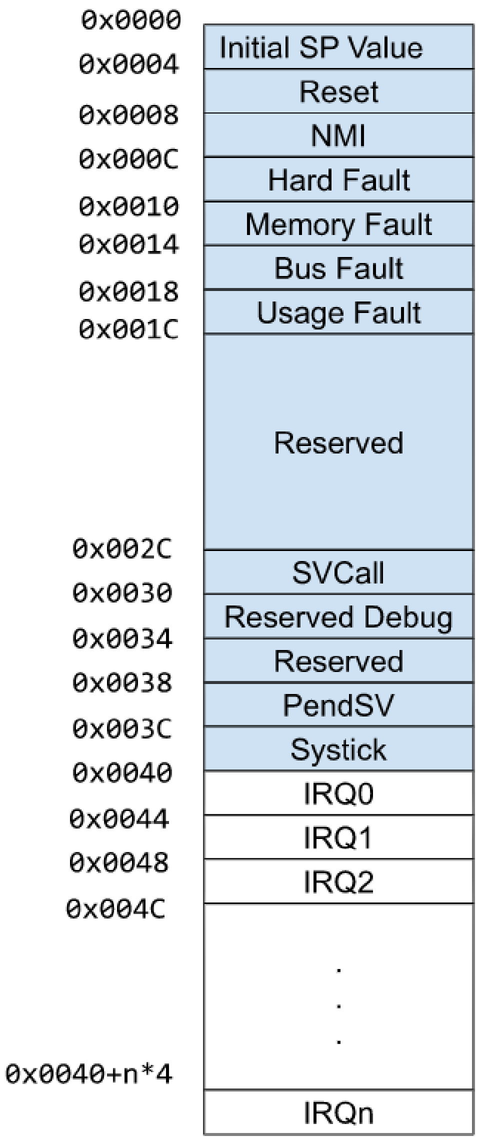


Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main flash memory	Main flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space



FIRMWARE BOOTING

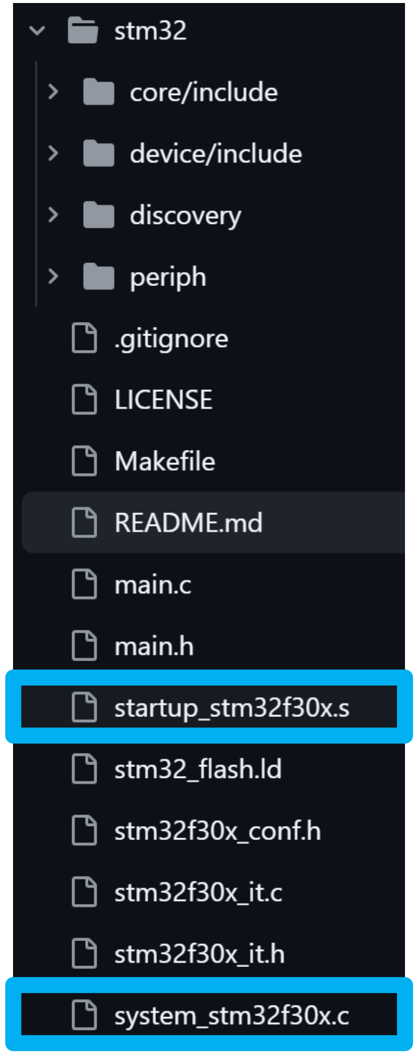
- Gives the developper flexibility for updates, fail-safe recovery and debugging.
- Decide how and where the firmware is loaded.
- Stack pointer (SP) will be loaded with content of address 0x00000000 (top of stack)
- Program counter (PC) will be loaded with content of address 0x00000004 (reset handler function)
- If boot mode is system memory, it should determines whether to load firmware from external FLASH, USB, UART or SPI.



0x0000	Initial SP Value
0x0004	Reset
0x0008	NMI
0x000C	Hard Fault
0x0010	Memory Fault
0x0014	Bus Fault
0x0018	Usage Fault
0x001C	Reserved
0x002C	SVCall
0x0030	Reserved Debug
0x0034	Reserved
0x0038	PendSV
0x003C	Systick
0x0040	IRQ0
0x0044	IRQ1
0x0048	IRQ2
0x004C	. . .
0x0040+n*4	IRQn

RESET HANDLER

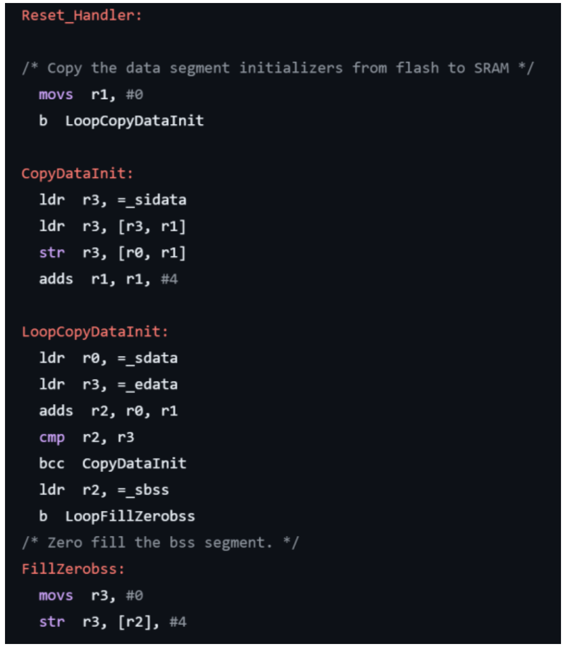
- Is the first piece of code executed in the firmware after system reset
- The function is written in startup file
- It copies memory data segment (.data) from flash to ram and fills the BSS segment with 0’s
- It calls SystemInit() function defined in system_<device>.c, which sets the system clock tree, PLLs, flash wait states, bus dividers



- stm32
  - core/include
  - device/include
  - discovery
  - periph
  - .gitignore
  - LICENSE
  - Makefile
  - README.md
  - main.c
  - main.h
  - startup_stm32f30x.s
  - stm32_flash.ld
  - stm32f30x_conf.h
  - stm32f30x_it.c
  - stm32f30x_it.h
  - system_stm32f30x.c

RESET HANDLER

- Sets the peripherals to their default state
- Initializes MMU (Memory Management Unit) if available
- Calls main() function

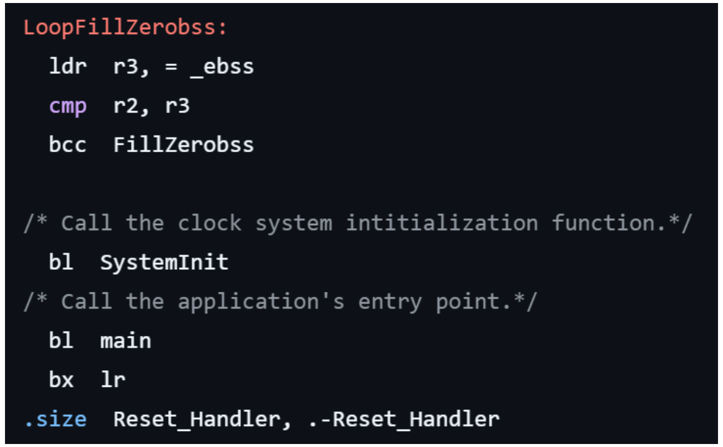


```
Reset_Handler:

/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
  ldr  r2, =_sbss
  b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
  movs  r3, #0
  str  r3, [r2], #4
```



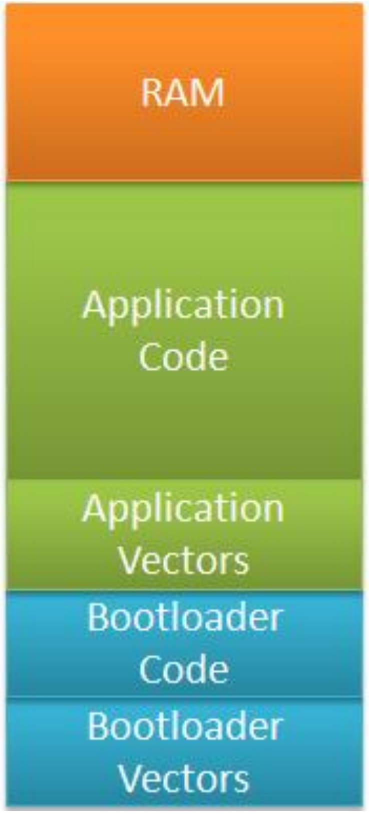
```
LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  bcc  FillZerobss

/* Call the clock system intitialization function.*/
  bl  SystemInit
/* Call the application's entry point.*/
  bl  main
  bx  lr
.size  Reset_Handler, .-Reset_Handler
```

BOOTLOADER

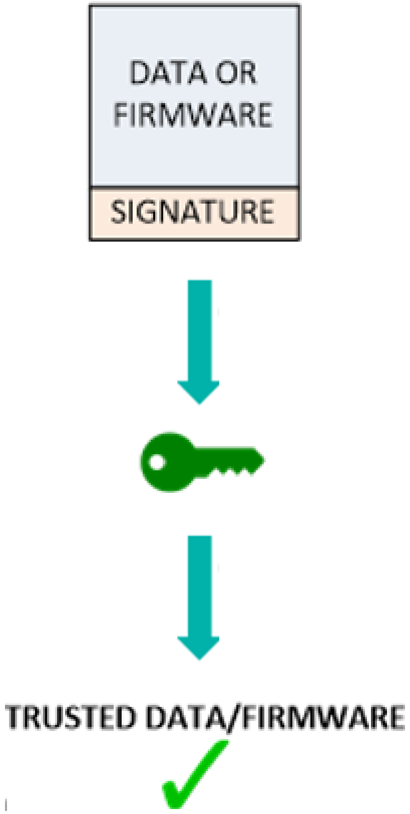
- A software that runs before the main application code
- It initializes the essential hardware and peripherals
- It updates/upgrades the firmware in embedded systems, without the need for physical access to the microcontroller
- It supports various communication protocols like USB, UART, SPI, ETHERNET or wireless communication

➢ It helps implement a security layer, verifies the integrity and authenticity of the firmware. Additionally, it is useful for fixing bugs in the firmware.



SECURE BOOT

- Is a mechanism that allows only trusted and authorized firmware to run
- It protects embedded systems from reverse engineering, firmware replacement and malware injection
- It uses digital signatures and cryptographic keys
- Establishes a chain of trust from microcontroller's power on



NEOCRUX
SYSTEMS

VISIT OUR LINKEDIN PAGE: NEOCRUX SYSTEMS