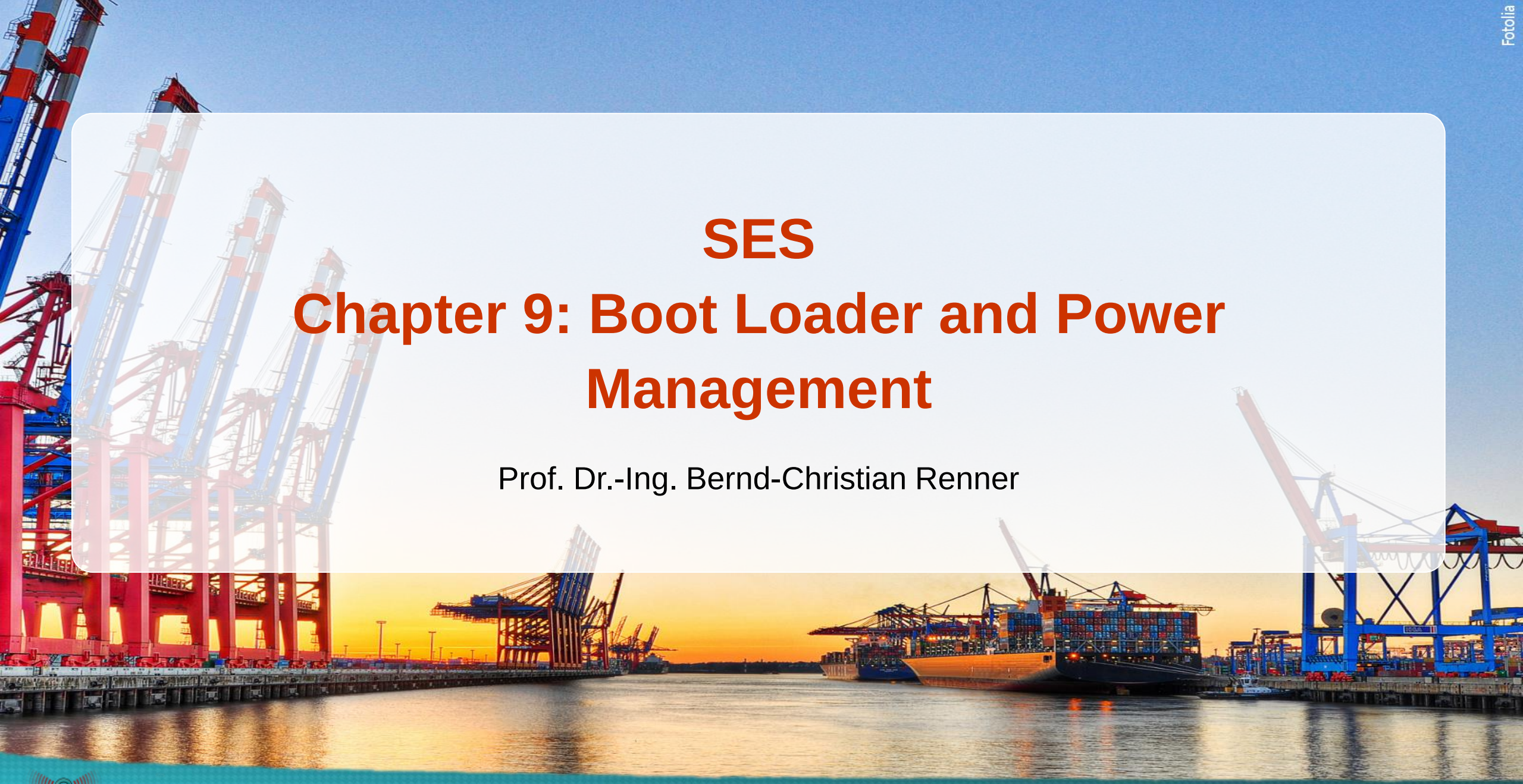


SES
Chapter 9: Boot Loader and Power Management

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology
TUHH

Contents

1

Boot Loader

Boot Loader

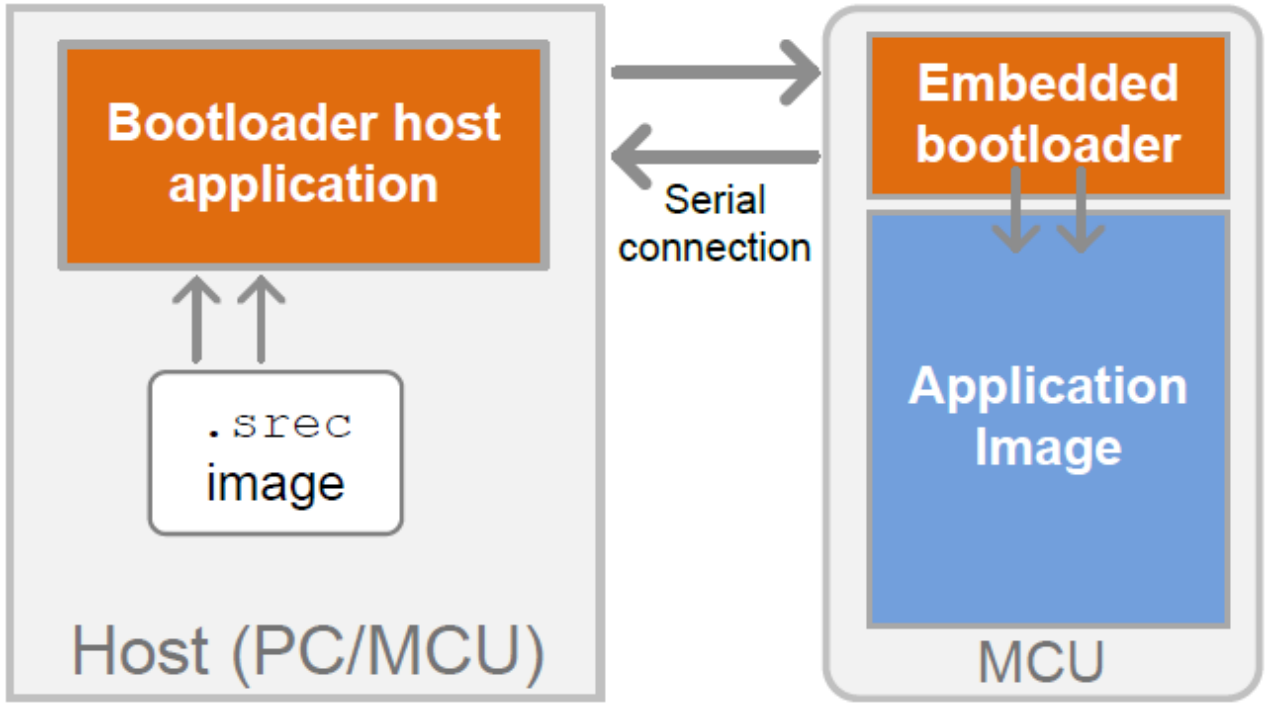
- Application whose primary purpose is to allow a system's software to be updated without using specialized hardware such as a JTAG programmer
- Boot Loader
  - resides in protected program memory (ROM)
  - first software to run after power up or reset (depending on configuration)
  - manages system's images
  - can communicate over a variety of protocols such as USART, CAN, I2C, Ethernet, USB, ….

Boot Loader: Behavioral Models

1. Boot loading process is completely automated and self-contained within system
   - Example: SD Card boot-loader
     - boot loader automatically detects new firmware and manages its own flashing process
2. Boot loader initializes into an idle state and awaits instructions from an outside source
   - Source is often PC-based software application that commands boot-loader into states necessary to flash a new image
   - Example: AVRDude

Serial Boot Loader

- Serial boot loader allows loading of firmware images to an MCU over serial connection

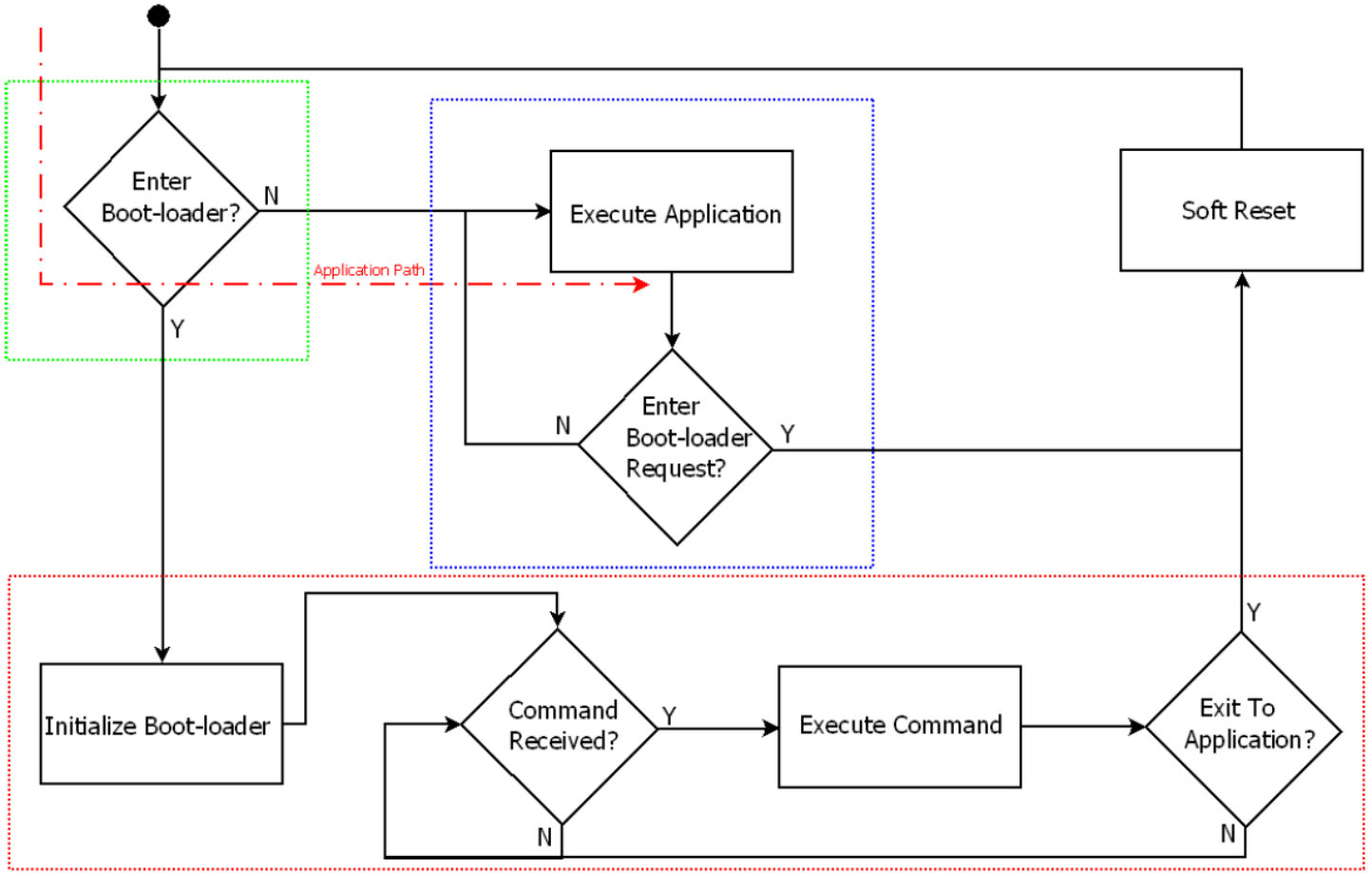


Serial Boot Loader

- Sequence of operations
  - Open image flashing tool, e.g. on PC
  - Start boot loader
  - Erase flash
  - Send binary file information to boot loader
    - Different formats
  - Quit boot-loader and enter application
- To receive a program, boot loader must interface with
  - a storage system (e.g. Flash) and
  - a serial communication method within system (e.g. SPI)

Serial Boot loader

- Master-slave principle
  - PC is master, boot loader is slave
  - Example protocols: Device Firmware Upgrade (DFU) standard, AVRProg



Boot Loader

- Branching code (green) makes decision as to whether boot loader or application is executed
- Entering boot loader is
  - initiated by a trigger such as a command received via USART, SPI interface, or button press
  - Boot Reset Fuse can be programmed so that Reset Vector is pointing to Boot Flash start address
- Boot loader code (red) performs boot loading functions
  - Boot loader usually begins initializing peripherals required in order to perform all of its functions
  - This allows boot loader to communicate with outside world and accept commands to perform flashing instructions

Reset

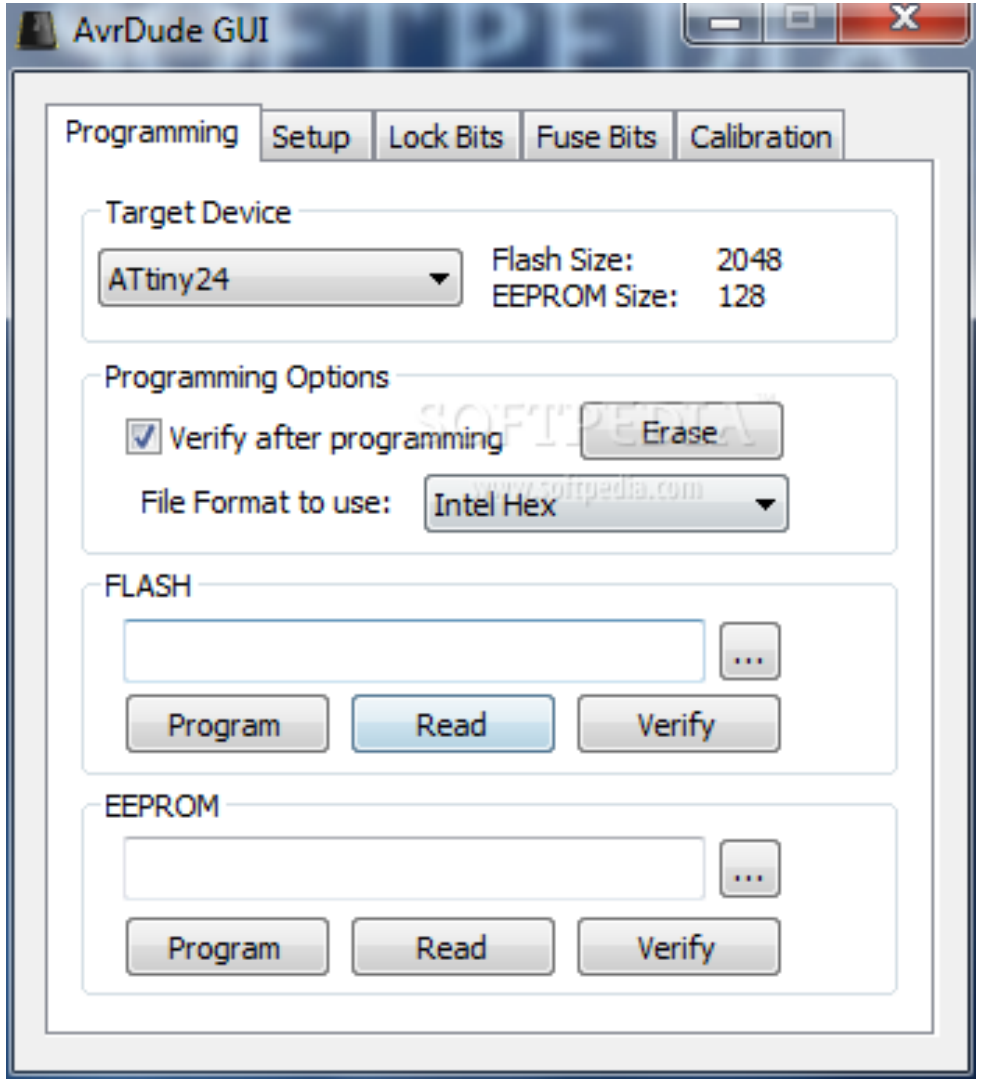
- Sources for reset
  - Power-on Reset
    - Supply voltage VCC is above power-on reset threshold
  - External Reset
    - RESET-Pin has low-level for at least minimal duration of pulse (500 ns)
  - Watchdog Reset
    - Watchdog-Timer has overflow
- Reason of reset can be read out from bits 0 – 3 of MCUCSR, (MCU Control & Status Register)

AVR Boot Loader

- Boot loader called
  - from application code using a jump or
  - on RESET (by programming the BOOTRST Fuse)
    - When BOOTRST Fuse is programmed, CPU starts execution in boot loader section on Reset, instead of starting at address 0
    - BOOTRST/BOOTSZ Fuse can be changed using Serial or Parallel Programming
- Boot loader resides in ISP (In-System Programmable) Flash memory
- Boot loader downloads application program into ISP Flash memory

AVR Downloader/Uploader: AVRDude

- PC program for downloading and uploading the on-chip memories of Atmel's AVR microcontrollers
  - It can program Flash and EEPROM,
  - and where supported by serial programming protocol, it can program fuse and lock bits

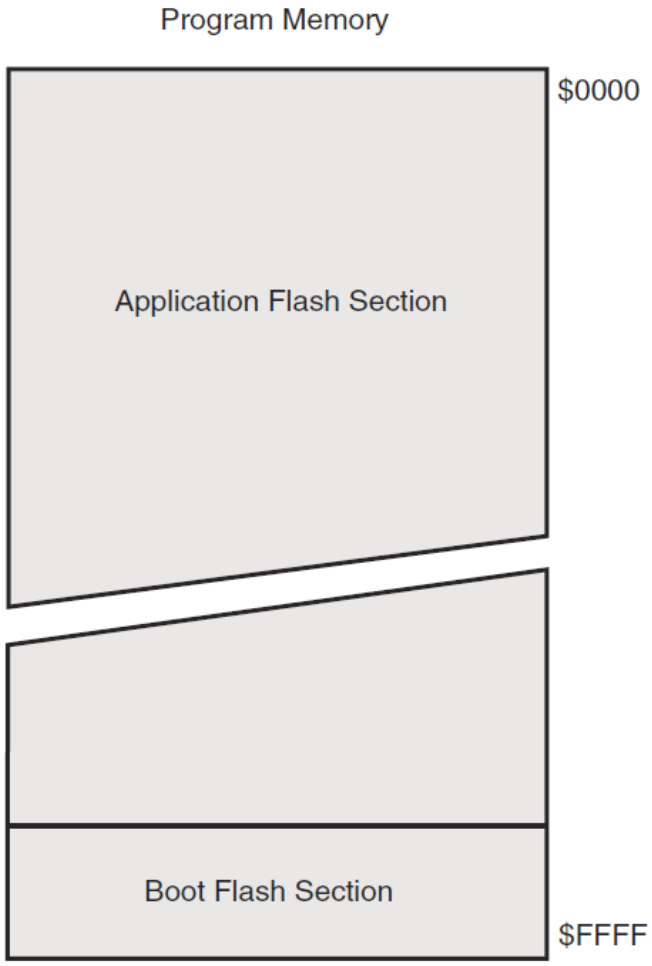


AVR: Image Transfer

- On-chip ISP Flash allows program memory to be reprogrammed in-system through
  - an SPI serial interface,
  - by a conventional nonvolatile memory programmer,
  - or by an on-chip boot loader running on AVR core
- Software in boot loader section continues to run while application Flash section is updated, providing true *Read-While-Write* operation
- Boot loader section has capability to write into entire Flash, including boot loader memory itself
  - i.e. boot loader can even modify itself

AVR: Organization of Flash Memory

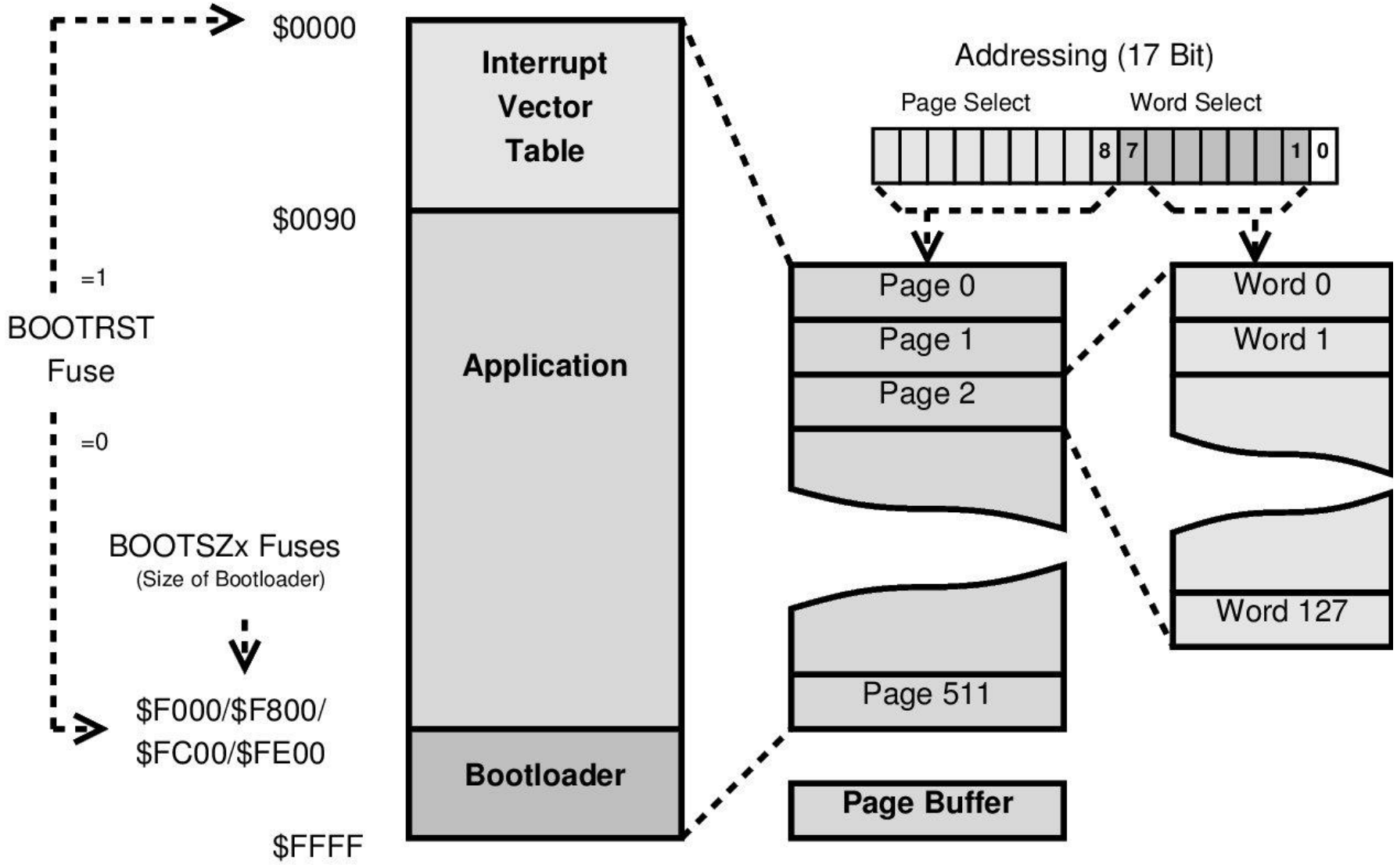
- Logically organized in two main sections
  - Application section (code for application)
  - Boot section
- Boot section can also be used for ordinary application code
- Size of sections is configured by BOOTSZ Fuses (512, 1024, 2048, or 4096 words)
- Flash addresses refer to words (2 Bytes)
- Flash memory is divided into pages containing 32, 64, or 128 words each
- Example (SES board):
  128 KB Flash, page size of 128 words, 512 pages



AVR: Organization of Flash Memory

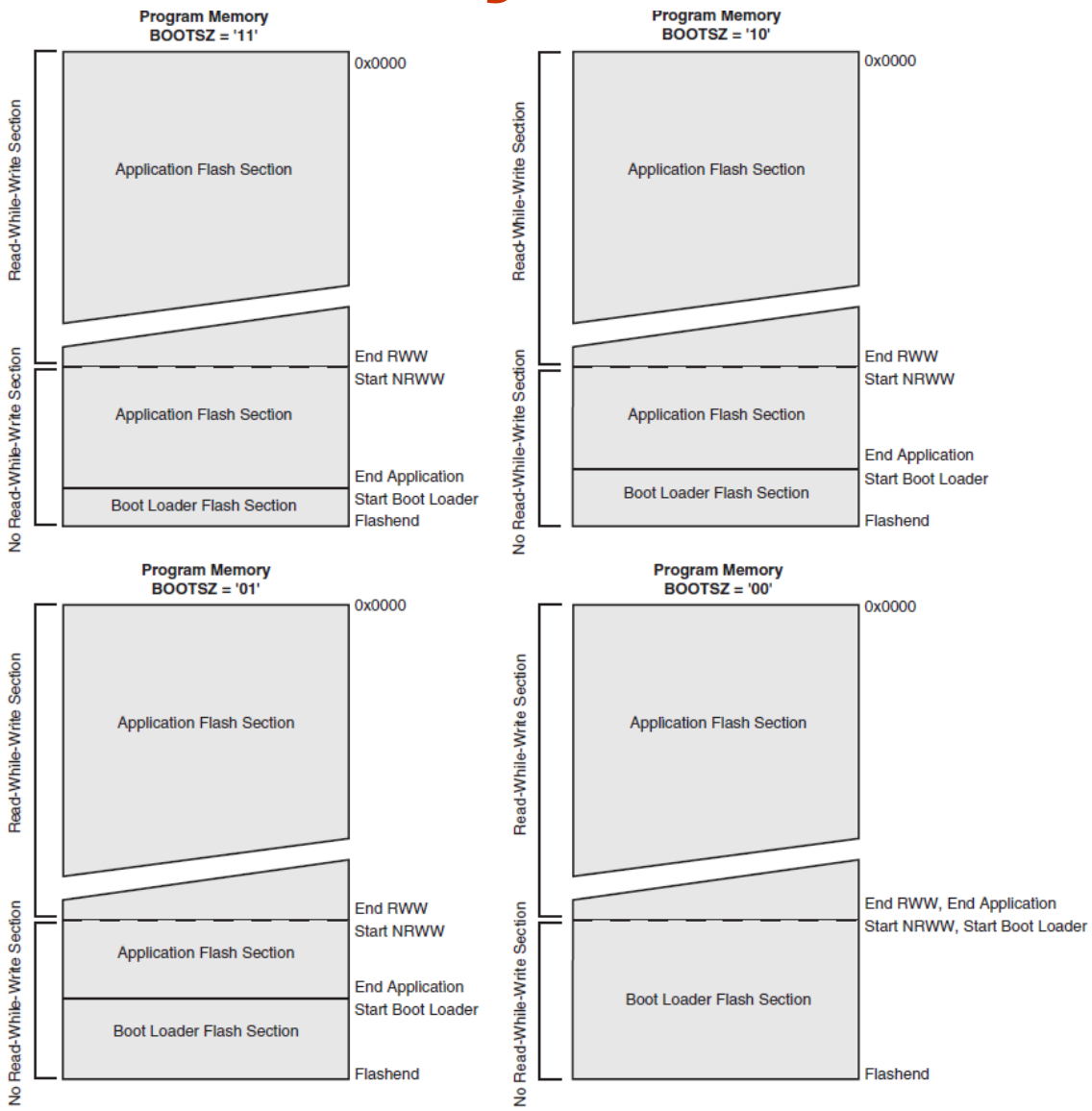
- Flash is 128 kB, thus byte addressing requires 17 bits
  - Writing: word addressing (originating in width of instruction), i.e. 2 bytes
  - Reading: byte addressing (e.g. for reading data)
- AVR lib uses 17 bit addresses (stored in 4 bytes), for write operations the last bit is ignored
  - 7 Bits to address a word in a page (128 words per page)
    - bit number 0 only used for reading bytes (not words), but only words can be written
  - 9 bits to address a page (512 pages)
  - Reading from Flash memory is per
    - byte or (17 bit addresses, far addresses)
    - word (16 bit addresses, near addresses)
- Flash memory updates are done page by page
- Before writing new data to a page, page must be erased

AVR: Organization of Flash Memory



AVR: Organization of Flash Memory

- In addition, Flash is physically divided into two fixed-size sections
  - Read-While-Write (RWW) section
  - No-Read-While-Write (NRWW) section
- When erasing or writing a page located inside the
  - RWW section, the NRWW section can be read during the operation
  - NRWW section, the CPU is halted during the entire operation
- Boot loader section is always contained in NRWW section



AVR: RWW- and NRWW Sections

- NRWW section is accessible while updating RWW section, but it is impossible to access RWW section when it's being updated
- Functionality makes it possible to continue execution of critical code while updating the RWW section

- All self-programming operations are performed using SPM (Store Program Memory) instructions
- SPM instructions
  - write into application Flash memory
  - are disabled when executed from application section

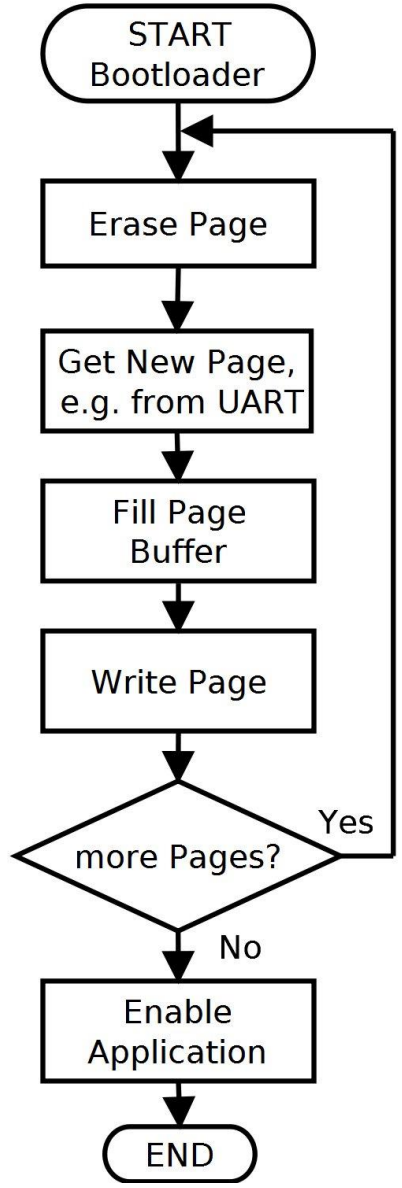
Boot Loader

- New data is first written to Page Buffer
  - Separate (not part of SRAM memory) write-only buffer holding one temporary page
  - Page Buffer is
    - filled word by word
    - copied to Flash memory in one operation
- Boot loader can use Interrupts
  - In this case interrupt vector table must reside at beginning of boot loader section
  - Early in code for your bootloader (before using interrupts), add

```
MCUCR = (1<<IVCE);
MCUCR = (1<<IVSEL);
```

AVR: Programing a Boot Loader

```
#include <avr/boot.h>

// Erase the flash page that contains address
// (byte address, i.e. 17 bits)
boot_page_erase(address);

// writes a word (2 bytes) into the page buffer
boot_page_fill(address, data_word);

// writes page buffer to addressed page (address)
boot_page_write(address);

// enables RWW section (needed for reading)
boot_rww_enable();

//  jump to application address 0 (inline assembler)
__asm__ __volatile__ ("jmp 0" ::);
```



Usage of Boot API

```
void boot_program_page(uint32_t page, uint8_t * buf) {
  uint16_t i;
  uint8_t sreg = SREG;
  cli();                                    // Disable interrupts

  boot_page_erase(page);
  boot_spm_busy_wait();                     // Wait until memory is erased

  for (i = 0; i < SPM_PAGESIZE; i += 2) {
    uint16_t w = *buf++;                    // Set up little-endian word
    w += (*buf++) << 8;
    boot_page_fill(page + i, w);
  }

  boot_page_write(page);                    // Copy buffer to flash page
  boot_spm_busy_wait();                     // Wait until memory is written

  // Re-enable RWW-section to jump back to application after bootloading
  boot_rww_enable();
  SREG = sreg;
}
```

Reading from Flash

```
#include <avr/pgmspace.h>
// reads a byte from Flash with a 32-bit address
pgm_read_byte_far(address)

// reads a byte from Flash with a 16-bit address
pgm_read_byte_near(address)

// reads a word from Flash with a 32-bit address
pgm_read_word_far(address)

// reads a dword from Flash with a 32-bit address
pgm_read_dword_far(address)
```

- `address` is 4 Bytes, `far` addresses 17 bit, `near` addresses 16 Bit
- With `near` addresses only the lower 64 Kbytes can be addressed
- Reading can be used for verification

AVR: Boot Loader Lock Bits

- If no boot loader capability is needed, entire Flash is available for application code
- Boot loader has two separate sets of Boot Lock bits which can be set independently
- Protection levels
  - Protect entire Flash from a software update by MCU
  - Protect only boot loader Flash section from a software update by MCU
  - Protect only Application Flash section from a software update by MCU
  - Allow software update in entire Flash

2

Power Management

Power Management

- Sleep modes enable an application to shut down unused modules in MCU, thereby saving power
- In battery-powered (e.g. all handheld products) applications, battery life is a key factor of design
- Two options for power saving
  1. Limit MCU clock rate
     - Has big influence on MCU's power consumption
     - Relationship between frequency and power is linear - double frequency yields double current consumption
     - Some MCUs allow to control the MCU's core clock
       - Core clock can operate at 1MHz for some instructions and then, one instruction later; it can slow to 32 kHz
  2. Enabling and disabling peripherals

AVR Power Management

- AVR provides various sleep modes allowing to tailor power consumption to application's requirements
- Six sleep modes:
  - idle mode
  - ADC noise reduction mode
  - power-down mode
  - power-save mode
  - standby mode
  - extended standby mode

AVR Power Management

- Most common sleep modes
  - Idle Mode
    - Stops CPU, leaves peripherals running (UART, ADC, …)
    - MCU will continue program execution immediately after wake up
  - Power-down mode
    - Deepest sleep mode
    - All oscillators are stopped, only external interrupts and watchdog continue operating
  - Power-save mode
    - Same as power-down mode except that timer crystal oscillator continues to operate, timer can wake-up MCU
  - RAM and registers are always kept!

AVR sleep modes (ATmega1281)

Sleep Mode	Usable modules	Power consumption 3.3 V/1 MHz	Wake-up source	Wakeup time
Active	all	1.2 mA	-	-
Idle	all	0.3 mA	all	6 cycles
ADC Noise Reduction	ADC, timer 2, ext. interrupts, TWI, Watchdog	~ 0.3 mA	ADC, timer 2, ext. level interrupt, TWI address match, brown out detector reset, external reset, watchdog reset	6 cycles
Power Save	Timer 2, ext. interrupts, TWI, watchdog	10 µA	Timer 2, ext. level interrupt, TWI address match, brown out detector reset, external reset, watchdog reset	28,4ms
Stand By	ext. interrupts TWI, watchdog	35 µA	Ext. level interrupt, TWI address match, brown out detector reset, external reset, watchdog reset	6 cycles
Power Down	ext. interrupts TWI, watchdog	0.3 µA	Ext. level interrupt, TWI address match, brown out detector reset, external reset, watchdog reset, PCINT	28,4ms

AVR Power Management

- Steps required to use the sleep modes
  - Select desired sleep mode by selecting SM bits in SMCR
  - Enable interrupts that will wake up MCU
  - Set sleep enabled bit SE in SMCR
  - Execute SLEEP instruction

SMCR (Sleep Mode Control Register)

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

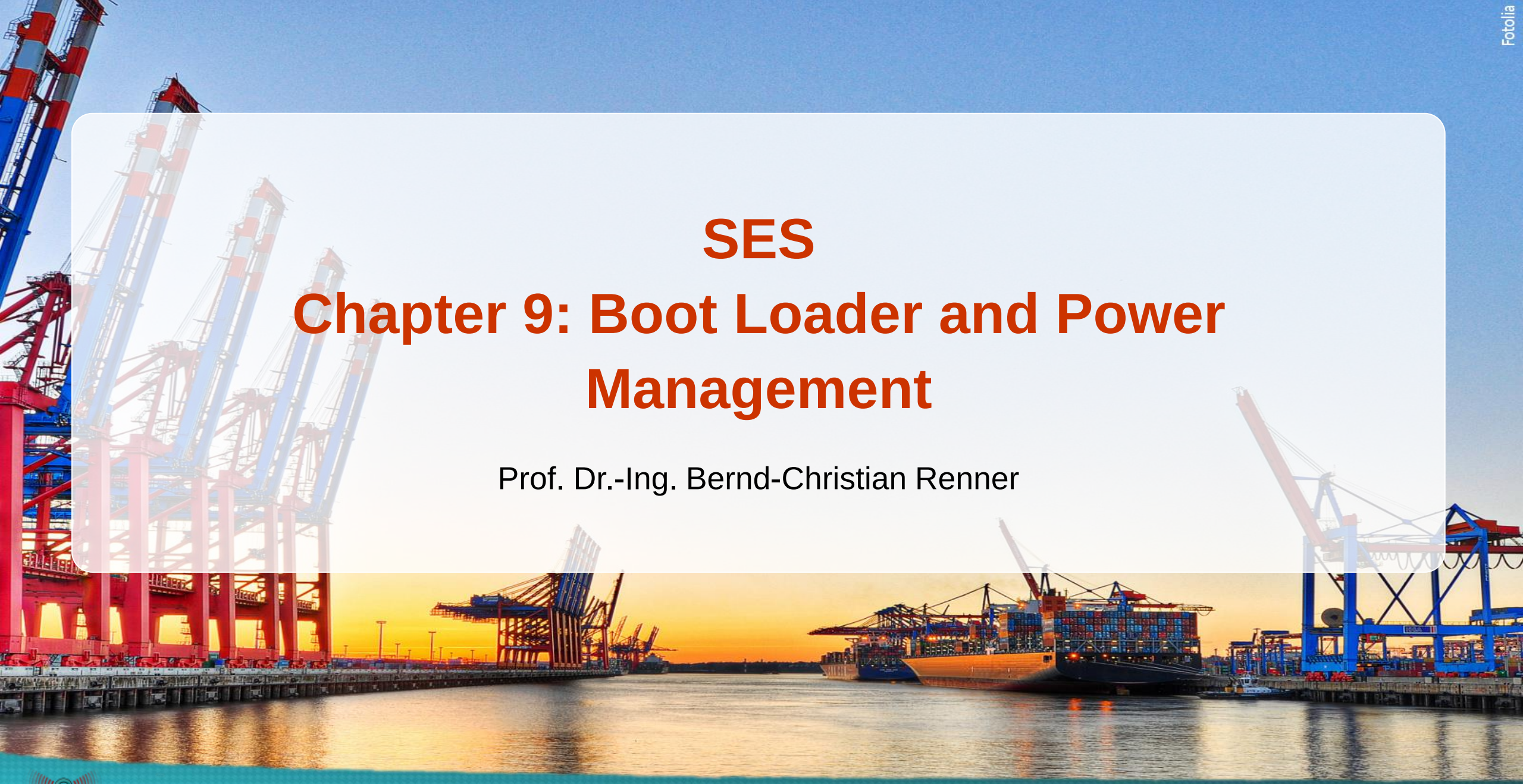
- SM2, SM1, SM0 bits select which sleep mode will be activated
- An enabled interrupt wakes up MCU
- MCU is then halted for four cycles in addition to start-up time and
  - executes the interrupt routine, and
  - resumes execution from instruction following SLEEP
- Contents of register file and SRAM are unaltered when device wakes up from sleep

AVR Sleep Mode Issues

- Sleep modes should
  - be used as much as possible
  - be selected so only required device's functions are operating
- Modules that may need special consideration when trying to achieve the lowest possible power consumption
  - ADC: To save power, ADC should be disabled before entering any sleep mode (if not needed)
  - Analog Comparator: When entering ADC Noise Reduction mode, it should be disabled. In other sleep modes, it is automatically disabled
  - Watchdog Timer: Turn off if not needed. If enabled, it is enabled in all sleep modes
  - Port Pins: When entering a sleep mode, all port pins should be configured to use minimum power
    - port pins in input hi-Z mode, i.e. DDRxn and PORTxn 0

Summary

- Boot loader
  - Tool required to initialize system in preparation for execution of operating system or primary program that runs on system
- Power management
  - AVR provides various sleep modes allowing the user to tailor power consumption to application’s requirements



SES
Chapter 9: Boot Loader and Power Management

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology
TUHH