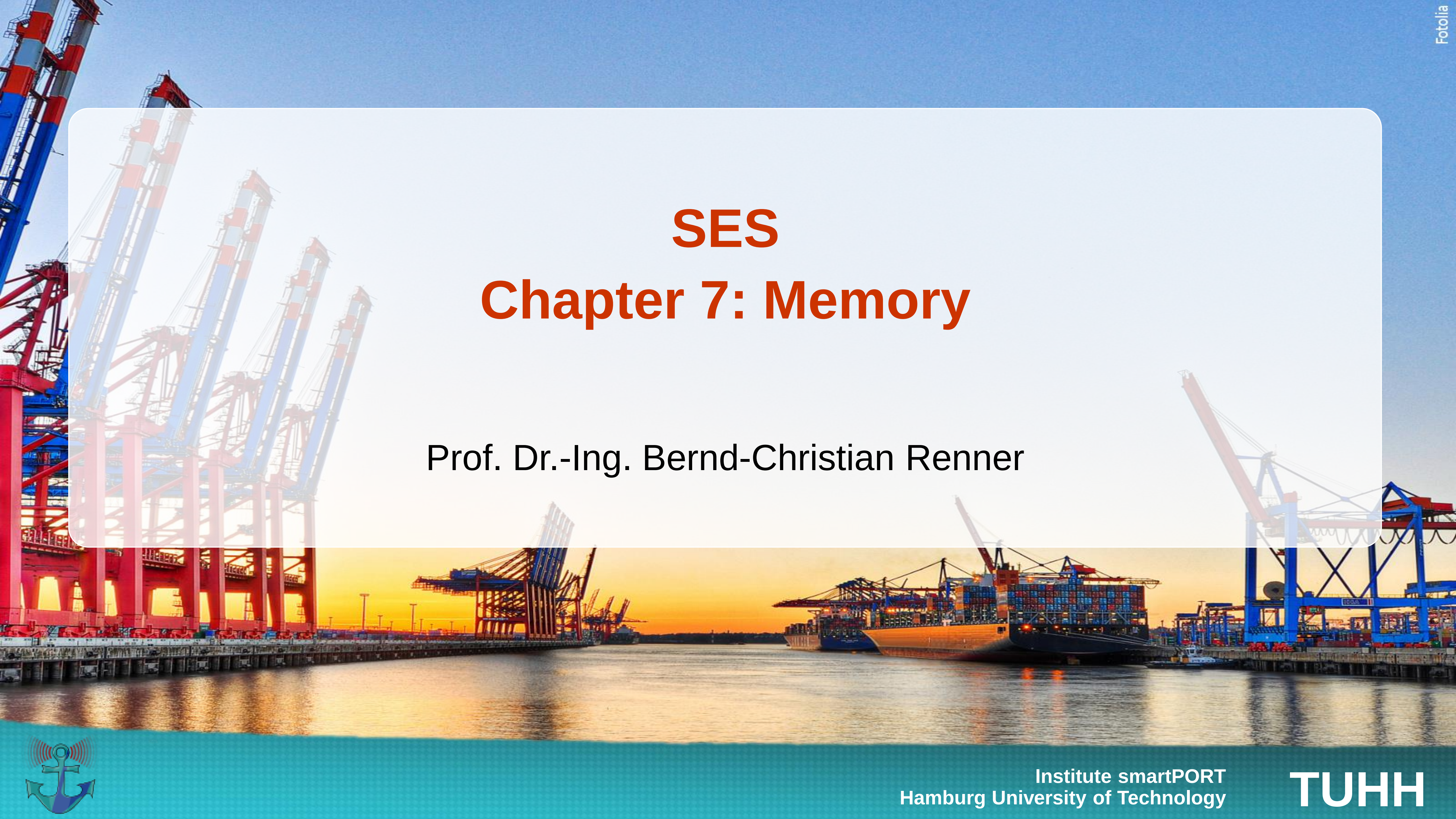


SES
Chapter 7: Memory

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology

TUHH

Contents

Introduction

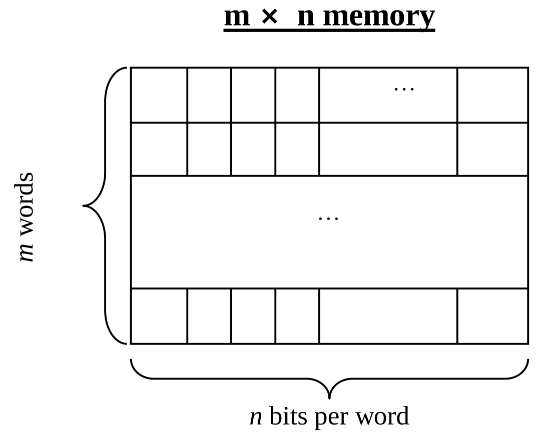
- Embedded system’s functionality aspects
  - Processing
    - processors
    - transformation of data
  - Storage
    - memory
    - retention of data
  - Communication
    - buses
    - transfer of data

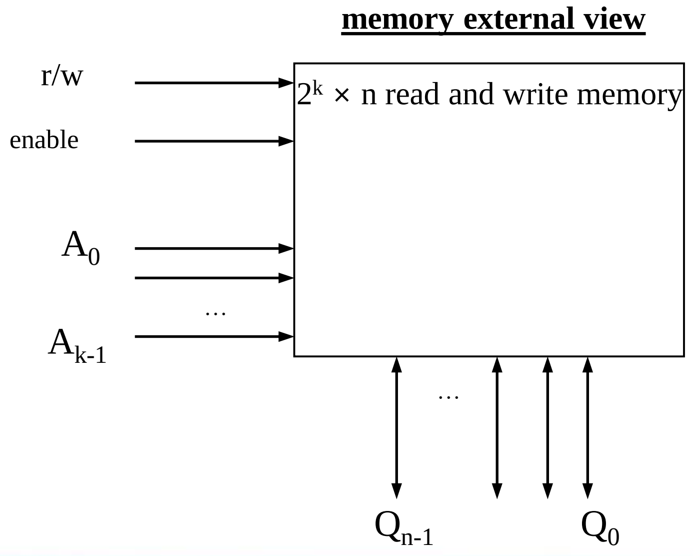
1

Basic Concepts

Overview

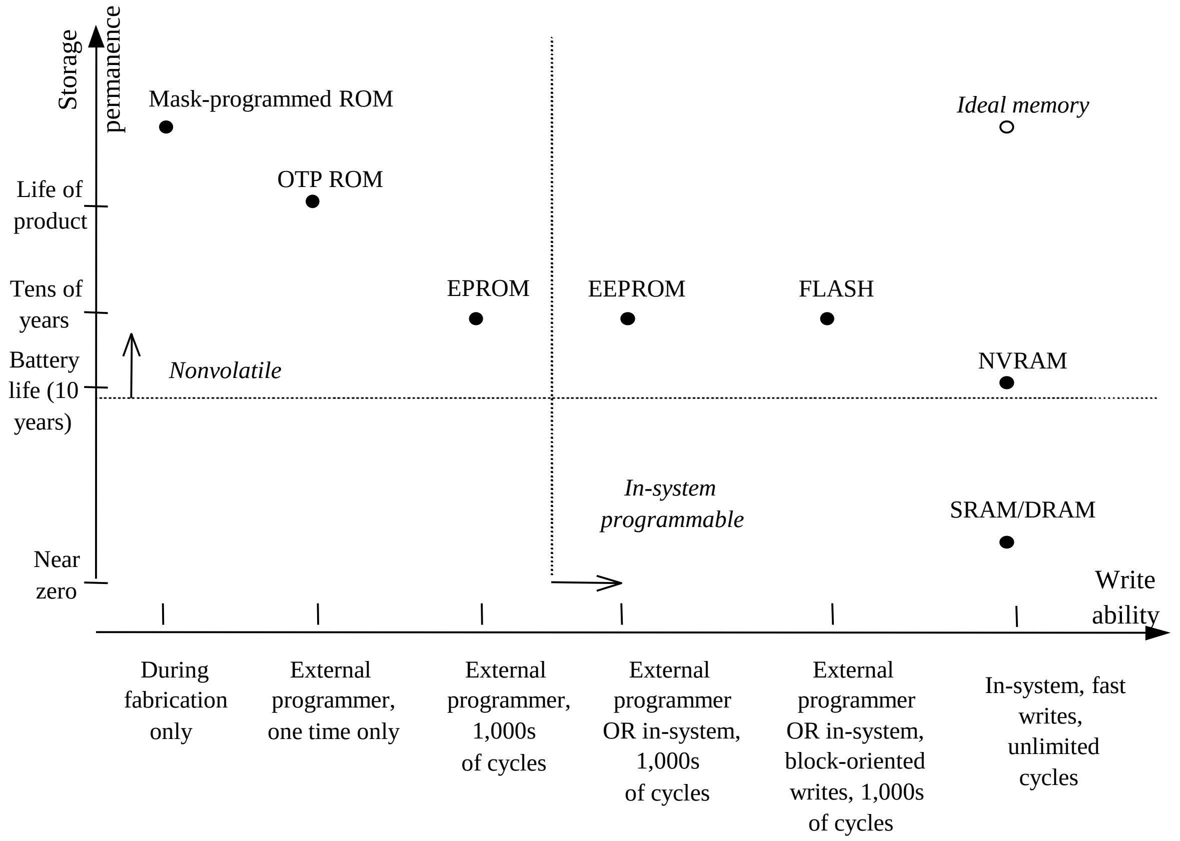
- Memory stores large number of bits
  - *m* x *n*: *m* words of *n* bits each
  - word: group of bits handled as a unit by processor
  - k = $\log_2(m)$ address input signals ($m = 2^k$ words)
  - e.g., 4,096 x 8 memory:
    - 32,768 bits
    - 12 address input signals ($2^{12} = 4096$)
    - 8 input/output data signals
- Memory access
  - r/w: selects read or write
  - enable: read or write only when asserted
  - A_i: address inputs, Q_i: data outputs
  - Multiport: multiple accesses to different locations simultaneously (requires multiple sets of address and data lines)





Write Ability/Storage Permanence

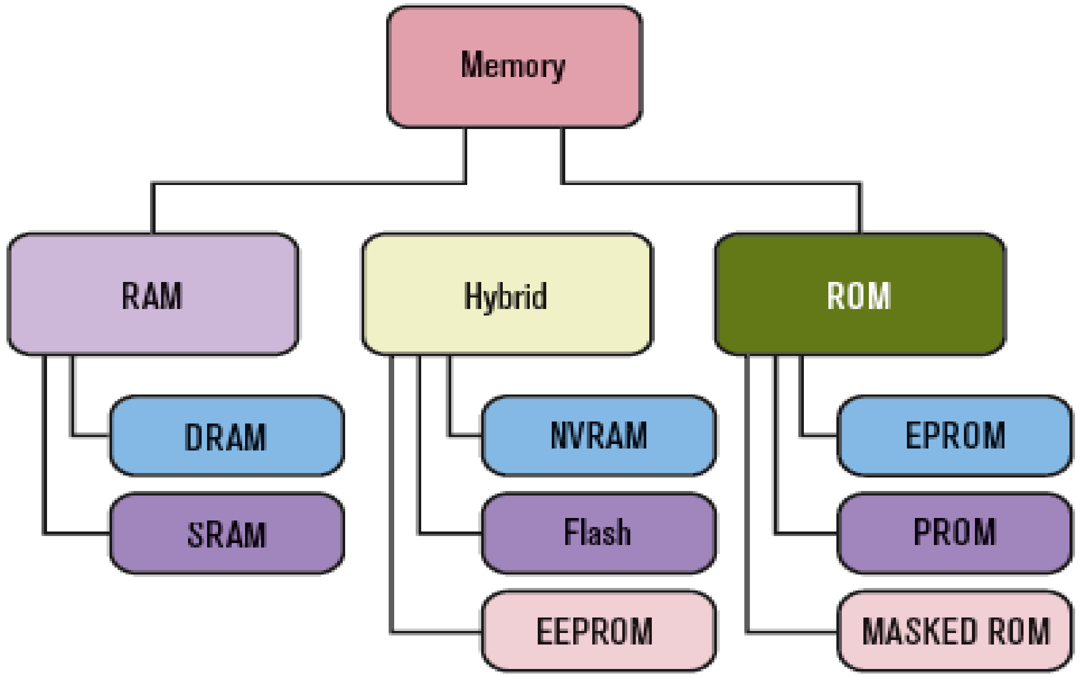
- ROM/RAM now and then
  - ROM: read only; today: bits stored without power
  - RAM: random access; today: lose stored bits without power
- Distinctions blurred
  - Advanced ROMs can be written to
    - e.g., EEPROM (Electrically Erasable Programmable ROM), Flash
  - Advanced RAMs can hold bits without power
    - e.g., NVRAM (Non Volatile Random Access Memory)
- Write ability: Manner and speed a memory can be written
- Storage permanence: ability of memory to hold stored bits after they are written



Write ability and storage permanence of memories, showing relative degrees along each axis (not to scale).

Write Ability

- Ranges of write ability
  - High end
    - processor writes to memory simply and quickly
    - e.g., RAM
  - Middle range
    - processor writes to memory, but slower
    - e.g., FLASH, EEPROM
  - Lower range
    - special equipment, “programmer”, must be used to write to memory
    - e.g., EPROM, OTP ROM
  - Low end
    - bits stored only during fabrication
    - e.g., mask-programmed ROM
- In-system programmable memory
  - Can be written to by a processor in the embedded system using the memory
  - Memories in high end and middle range are in-system programmable memory



Storage Permanence

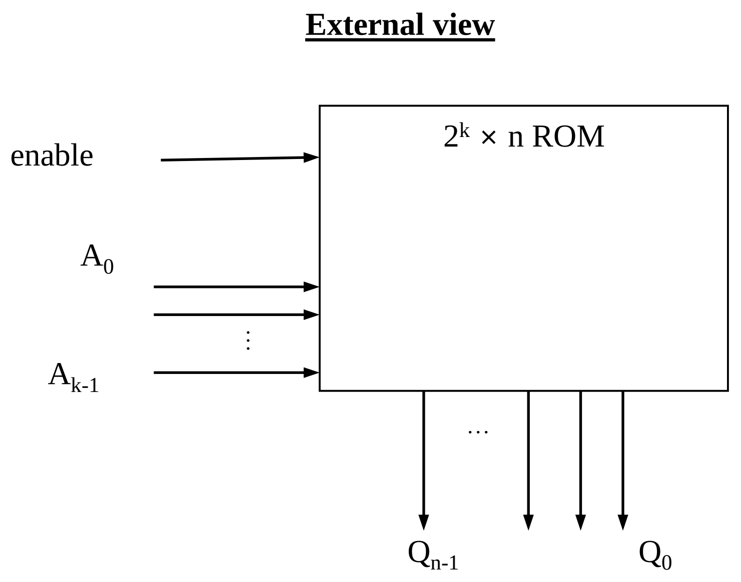
- Range of storage permanence
  - High end
    - essentially *never* loses bits
    - e.g., mask-programmed ROM
  - Middle range
    - holds bits days, months, or years after memory's power source turned off
    - e.g., Flash, NVRAM
  - Lower range
    - holds bits as long as power supplied to memory
    - e.g., SRAM (Static random access memory)
  - Low end
    - begins to lose bits almost immediately after written
    - e.g., DRAM (Dynamic Random Access Memory)
- Nonvolatile (persistent) memory
  - Holds bits after power is no longer supplied
  - High end and middle range of storage permanence

2

Common Memory Types

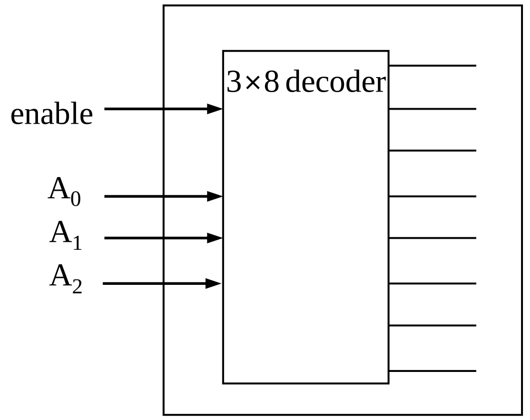
Overview

- ROM: “Read-Only” Memory
  - Today: Nonvolatile memory
  - Can be read from but not always written to, by a processor in embedded system
  - Traditionally written to (*programmed*) before inserting to embedded system
  - Access times are often slower than RAM: roughly 1.5 times that of DRAM
  - Uses
    - Store software program for general-purpose processor
      - program instructions can be one or more ROM words
    - Store constant data needed by system
      - Lookup tables



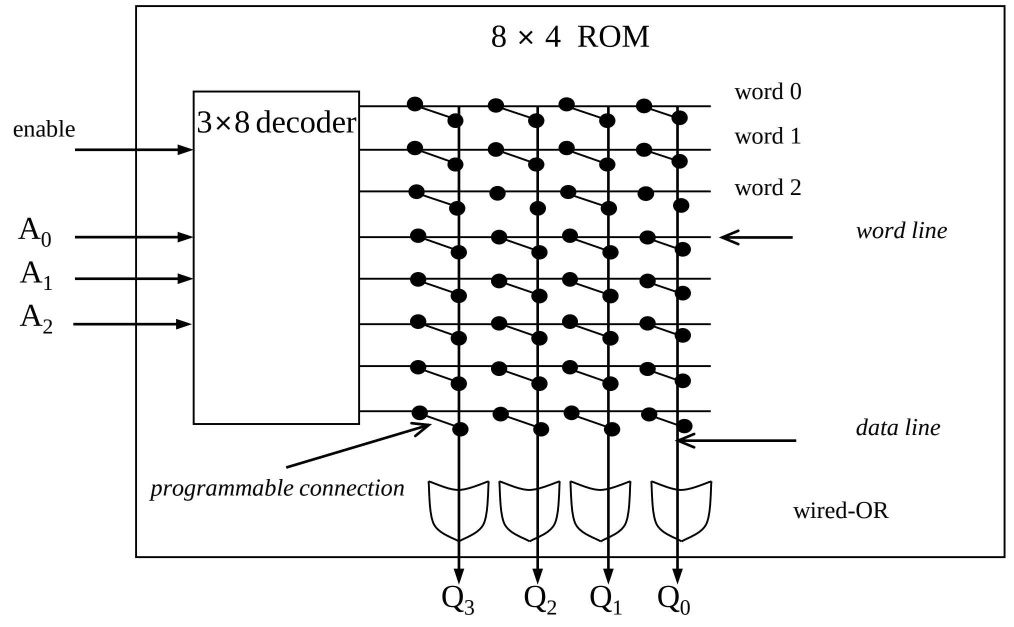
Decoder

- Decoder: Converts its binary input into a *one-hot* output
  - One-hop output: Exactly one of output lines is given a 1 at a time
- Decoder with n outputs and $\log_2$ n inputs is called a $\log_2$ n x n decoder
- Example: 3 x 8 decoder
  Input: 111 Output: line 7 is 1
- Extra input: enable
  When enable is 0: all outputs are 0
  When enable is 1: decode functions as before



Example: Internal view of a 8 x 4 ROM

- Horizontal lines = words
- Vertical lines = data
- Lines connected only at circles



- Example: Address 010
  - Decoder sets line of word 2 to 1
  - Data lines Q_3 and Q_1 are set to 1 because there is a "programmed" connection with word 2's line
  - Word 2 is not connected with data lines Q_2 and Q_0
  - Output is 1010

Mask-programmed ROM

- Connections "programmed" at fabrication
  - set of masks
- Lowest write ability
  - only once
- Highest storage permanence
  - bits never change unless damaged
- Typically used for final design of high-volume systems
  - spread out NRE cost for a low unit cost

OTP ROM: One-time programmable ROM

- Connections “programmed” after manufacture by user with special device
  - user provides file of desired contents of ROM
  - file input to machine called ROM programmer
  - each programmable connection is a fuse
  - ROM programmer blows fuses where connections should not exist
- Very low write ability
  - written only once and requires ROM programmer device
- Very high storage permanence
  - bits don’t change unless reconnected to programmer and more fuses blown
- Commonly used in final products
  - cheaper, harder to inadvertently modify

EPROM: Erasable programmable ROM

- Programmable component is a MOS transistor
  - Transistor has “floating” gate surrounded by an insulator
  - Programmed using higher voltage
  - EPROM is erased in their entirety
    - Ultraviolet light (UV) is used for erasing (takes 5 to 30 minutes)
    - EPROM package has quartz window
- Better write ability
  - can be erased and reprogrammed thousands of times
- Reading is much faster than writing
- Reduced storage permanence
  - Data lasts about 10 years but is susceptible to radiation and electric noise
- Typically used during development, limited usage in productions parts (window covered)

EEPROM: Electrically EPROM

- EEPROM: Electrically erasable programmable ROM
- Programmed and erased electronically
  - can program and erase individual words
- Better write ability
  - in-system programmable with built-in memory controller
  - writes very slow due to erasing and programming
    - “busy” pin indicates to processor EEPROM still writing
  - can be erased and programmed tens of thousands of times
- Similar storage permanence to EPROM (≈ 10 years)
  - Far more convenient than EPROMs, but more expensive

Flash Memory

- Extension of EEPROM
  - Same floating gate principle
  - Same write ability and storage permanence
  - Block operation only, typically several kilo bytes
- Fast erase
  - Large blocks of memory erased at once, rather than one word at a time as in traditional EEPROM
- Writes to single words may be slower than EEPROM
  - Entire block must be read, word updated, then entire block written back
  - EEPROM takes more die area than flash memory for same capacity
- Most microcontrollers use Flash as program memory
- Embedded systems: Storage for large data items in nonvolatile memory
- Recently flash memory is used as a replacement for hard disks
  - Solid-state drive (SSD)

RAM: “Random-access”/read-write memory

- Typically volatile memory
  - Bits are not held without power supply
  - Name *random-access* is no longer characteristic
- Internal structure more complex than ROM
  - A word consists of several memory cells, each storing 1 bit
- SRAM: Static RAM
  - Memory cell uses flip-flop to store bit
  - Holds data as long as power supplied, no refresh required
- DRAM: Dynamic RAM
  - Memory cell uses MOS transistor and capacitor to store bit
  - “Refresh” required due to capacitor leak
    - typical refresh rate 15.625 µs
  - Slower to access than SRAM, but cheaper than SRAM

RAM: “Random-access”/read-write memory

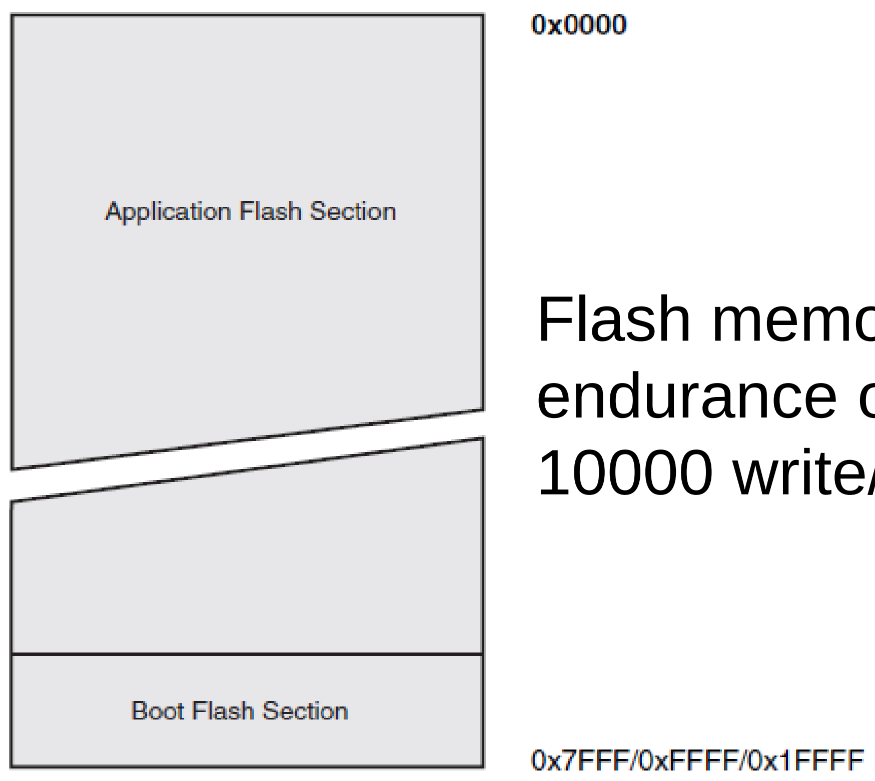
- NVRAM: Nonvolatile RAM
  - Holds data after external power removed
  - *Battery-backed* RAM
    - SRAM with own permanently connected battery
    - Write cycles as fast as read cycles (nanoseconds)
    - No limit on number of writes unlike nonvolatile ROM-based memory
  - SRAM with EEPROM or flash of same size
    - Stores complete RAM contents on EEPROM or flash before power turned off
  - New developments: FeRAM, MRAM, PRAM

Characteristics of Memory Types

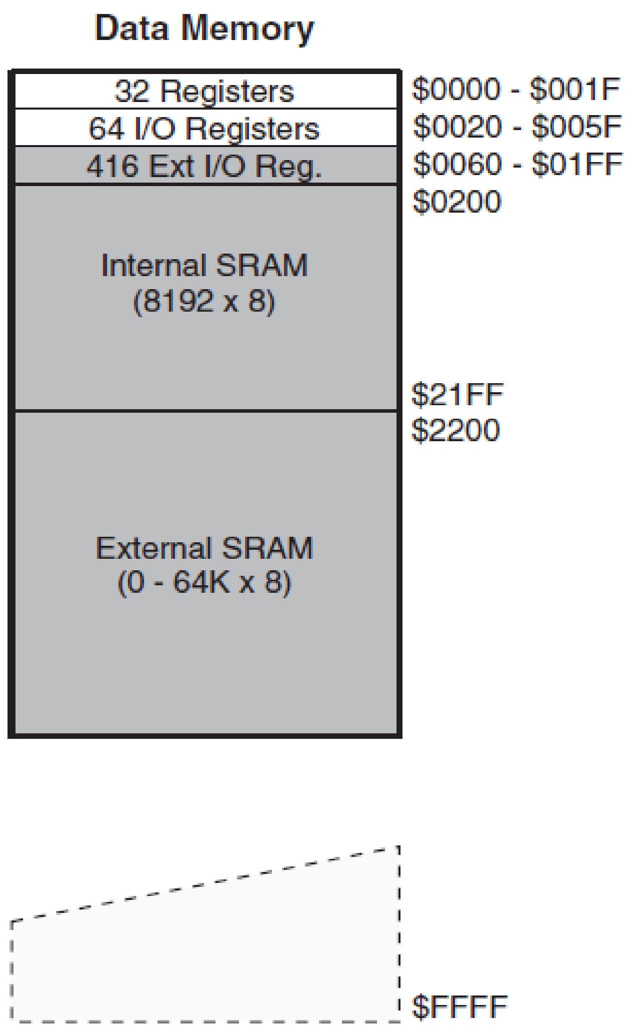
Type	*Volatile?*	*Writeable?*	*Erase Size*	*Max Erase Cycles*	*Cost (per Byte)*	*Speed*
SRAM	Yes	Yes	Byte	Unlimited	Expensive	Fast
DRAM	Yes	Yes	Byte	Unlimited	Moderate	Moderate
Masked ROM	No	No	n/a	n/a	Inexpensive	Fast
PROM	No	Once, with a device programmer	n/a	n/a	Moderate	Fast
EPROM	No	Yes, with a device programmer	Entire Chip	Limited (consult datasheet)	Moderate	Fast
EEPROM	No	Yes	Byte	Limited (consult datasheet)	Expensive	Fast to read, slow to erase/write
Flash	No	Yes	Sector	Limited (consult datasheet)	Moderate	Fast to read, slow to erase/write
NVRAM	No	Yes	Byte	Unlimited	Expensive (SRAM + battery)	Fast

Memory sizes of ATmega 128RFA1

Flash	SRAM	EEPROM
128 kB	16 kB	4 kB

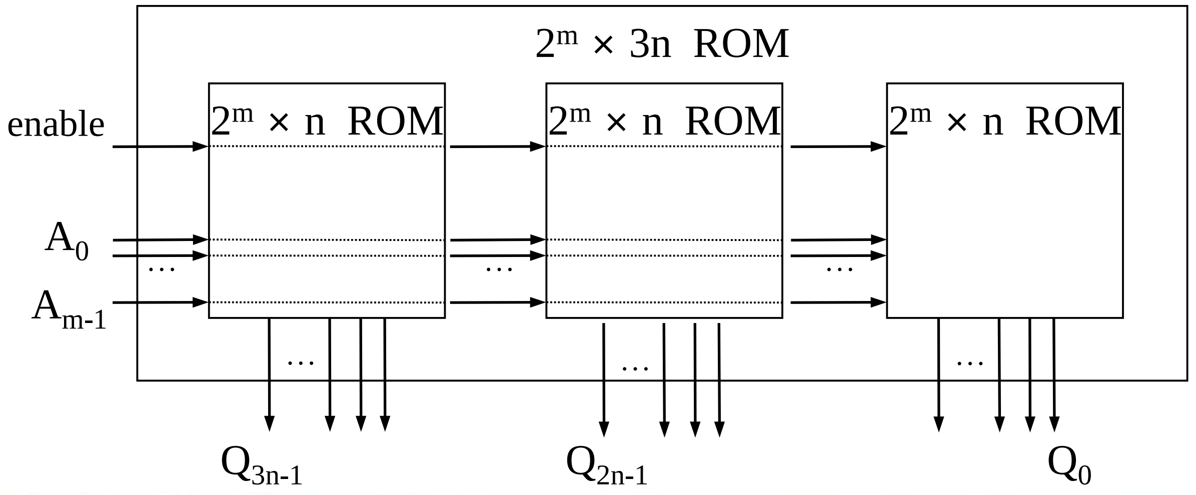


Flash memory has an endurance of at least 10000 write/erase cycles



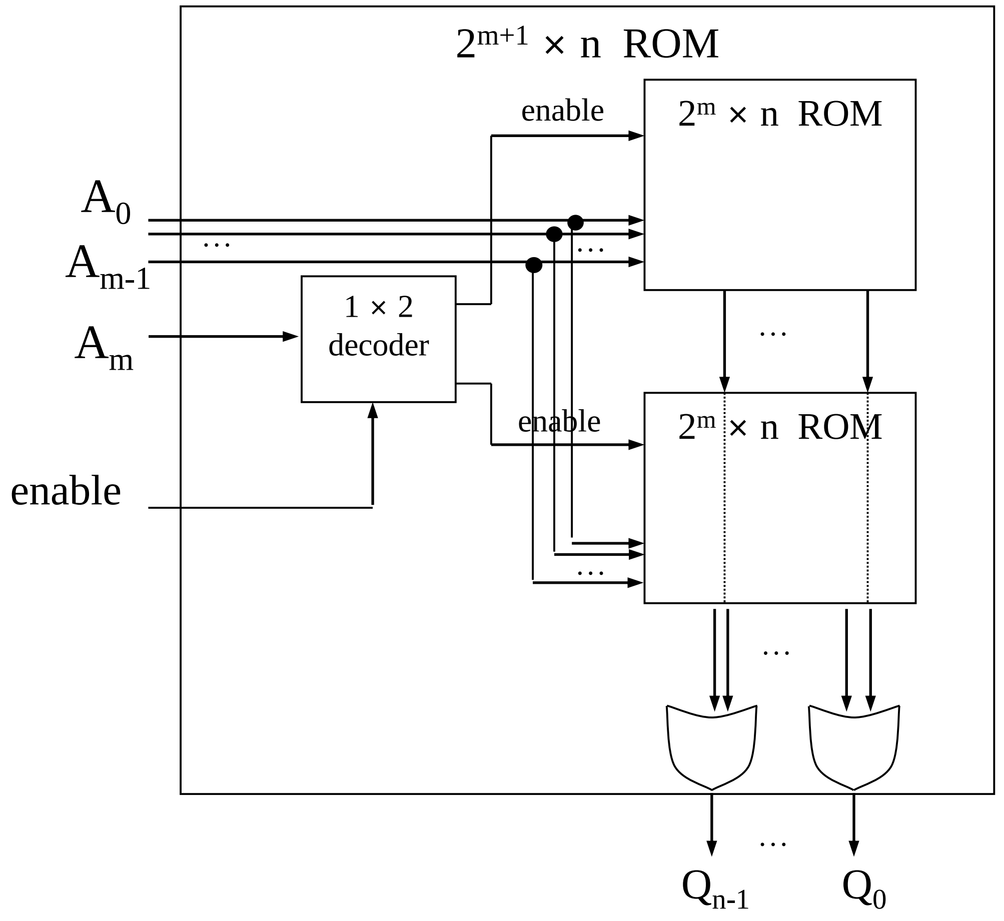
Composing memory

- Memory size needed often differs from size of readily available memories
- When available memory is larger, simply ignore unneeded high-order address bits and higher data lines
- When available memory is smaller, compose several smaller memories into one larger memory
  - Connect side-by-side to increase width of words



Composing memory

- Connect top to bottom to increase number of words
  - added high-order address line selects smaller memory containing desired word using a decoder
- Combine techniques to increase number and width of words



Memory Management Unit (MMU)

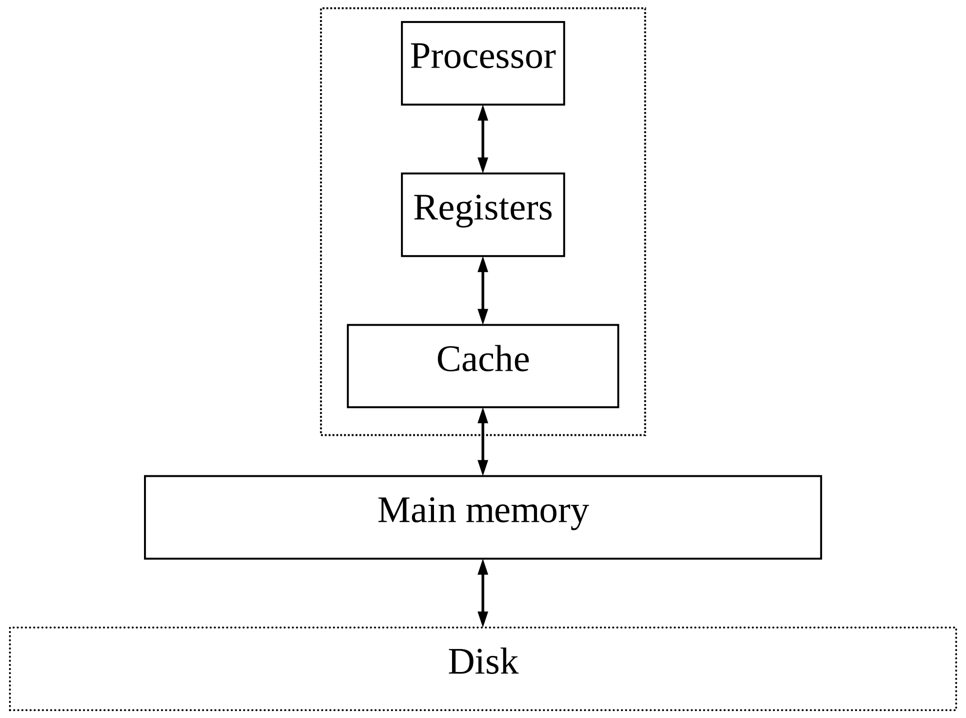
- Duties of MMU
  - Handles DRAM refresh, bus interface and arbitration
  - Takes care of memory sharing among multiple processes
  - Translates logic memory addresses from processor to physical memory addresses of DRAM (for paging)
- Modern CPUs often come with MMU built-in
  - AVR 8 Bit has no MMU
- Single-purpose processors can be used

3

Memory Hierarchy

Overview

- Main memory
  - Large, inexpensive, slow memory stores entire program and data
  - Usually DRAM
- Cache
  - Small, expensive, fast memory stores copy of likely accessed parts of larger memory
  - Often multiple levels of cache



Cache

- Usually designed with SRAM
  - Faster (1-3 cycles vs. several cycles for main memory), but more expensive than DRAM
- Usually on same chip as processor
  - Space limited, much smaller than off-chip main memory
- Cache operation
  - Request for main memory access (read or write)
  - First, check cache for copy
    - Cache hit: copy is in cache, quick access
    - Cache miss: copy not in cache, read data and possibly neighbors into cache
- Cache to RAM Ratio
  - A processor might have 512 KB of cache and 512 MB of RAM
  - Algorithms must select the 0.1% of memory that is likely to be most accessed
- Several cache design choices
  - Cache mapping, replacement policies, and write techniques
- Caches are unsuitable for real time applications

1

EEPROM Example

EEPROM Example

```
uint8_t EEPROM_read(uint16_t uAddress) {
    while (EECR & (1 << EEPE)) {
        /* Wait for completion of previous write */
    }

    /* Set up address register */
    EEAR  = uAddress;

    /* Start EEPROM read by writing EERE */
    EECR |= (1 << EERE);

    /* Return data from Data Register */
    return EEDR;
}
```

EEAR = EEprom Address Register (16 bit)
EEDR = EEprom Data Register (8bit)
EECR = EEprom Control Register

EEPROM read access takes one instruction, requested data is available immediately
When EEPROM is read, CPU is halted for four cycles before next instruction is executed

- Condition that EPPROM is ready for reading or writing: EECR & (1 << EEPE) == 0
- EEPEth bit of EECR must be checked before reading from EEPROM, because address setting and reading from EPROM isn‘t possible during write
- Writing EERE in EECR enables reading
- Value of byte at address in EEAR is in EEDR

EEPROM Example

```
void EEPROM_write(uint16_t uAddress, uint8_t uData) {
    while (EECR & (1 << EEPE)) {
        /* Wait for completion of previous write */
    }

    /* Set up address and Data Registers */
    EEAR  = uAddress;
    EEDR  = uData;

    /* Pepare for writing (Master Write Enable) */
    EECR |= (1 << EEMPE);

    /* Start EEPROM write by setting EEPE */
    EECR |= (1 << EEPE);
}
```

- Bits EEMPE and EEPE must be set in EECR to start writing to EEPROM
- EEPE is automatically written to zero (cleared) on end of writing

EEPROM Example

- EEMPE: EEPROM Master Programming Enable
  - When EEMPE is set, setting EEPE within four clock cycles will write data to the EEPROM at the selected address
  - If EEMPE is zero, setting EEPE will have no effect
  - When EEMPE has been written to one by software, hardware clears the bit to zero after four clock cycles
- EEPE: EEPROM Programming Enable
  - Write Enable Signal EEPE is write strobe to EEPROM
  - When address and data are correctly set up, EEPE bit must be written to one to write value into EEPROM
  - EEMPE bit must be written to one before a logical one is written to EEPE, otherwise no EEPROM write takes place

avr/eeprom.h

- `uint8_t eeprom_read_byte (const uint8_t *p)`
- `void eeprom_write_byte (uint8_t * p, uint8_t val)`

- And many more functions

Attribute EEMEM

- Keyword EEMEM indicates to compiler that variables are stored in EEPROM instead of SRAM address space like a normal variable
- It creates separate .eep file which has to be written to chip separately
- Example

```
#include <avr/eeprom.h>
uint8_t EEMEM nonVolatileChar;
uint16_t EEMEM nonVolatileInt;
uint8_t EEMEM nonVolatileString[10];
```

- EEPROM address space starts from location 0

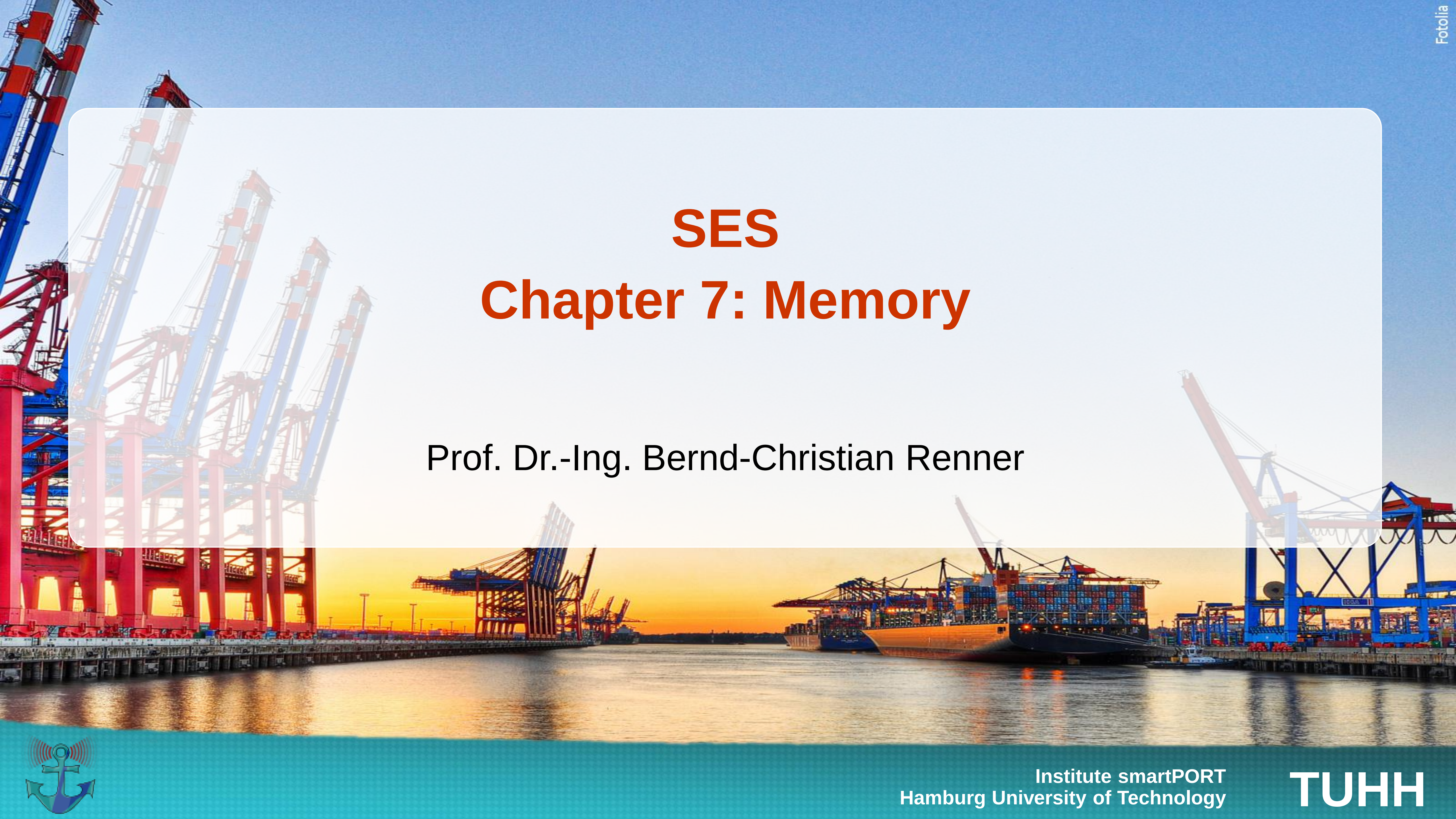
Attribute EEMEM

- To access the seperate EEPROM memory space it is required to use the eeprom read and write routines
- Example (cnt)

```
int main(void) {
  uint8_t sramChar;
  sramChar = eeprom_read_byte(&nonVolatileChar);
}
```

- To set a default EEPROM value in GCC, simply assign a value to a EEMEM variable

```
uint8_t EEMEM someVariable = 42;
```



SES
Chapter 7: Memory

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology

TUHH