Computertechnik 2

Jil Zerndt
FS 2025

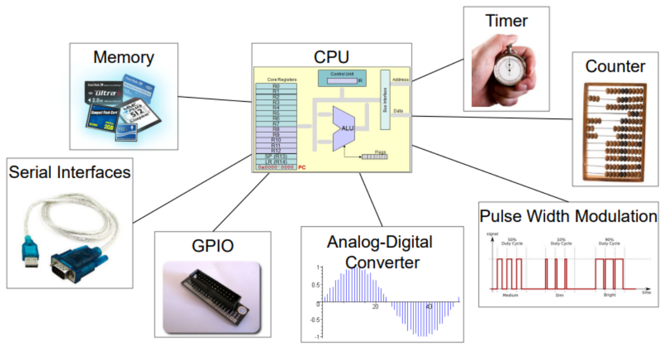
Microcontroller

Embedded Systems

- low cost (usb sticks, consumer electronics)
- low power (sensor networks, mobile devices)
- small size (smart cards, wearables)
- real time (anti-lock brakes, process control)
- reliability (medical devices, automotive)
- extreme environment (space, automotive)

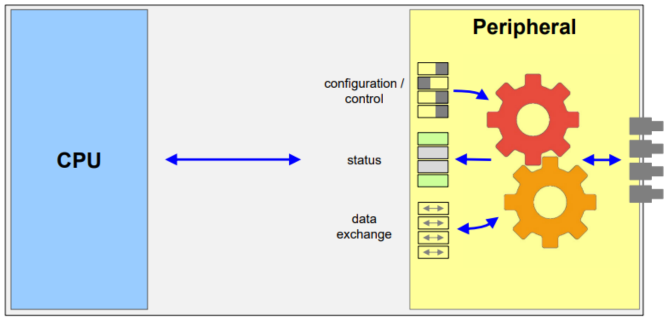
Single Chip Solution

⇒ CPU with integrated memory and peripherals



Peripherals

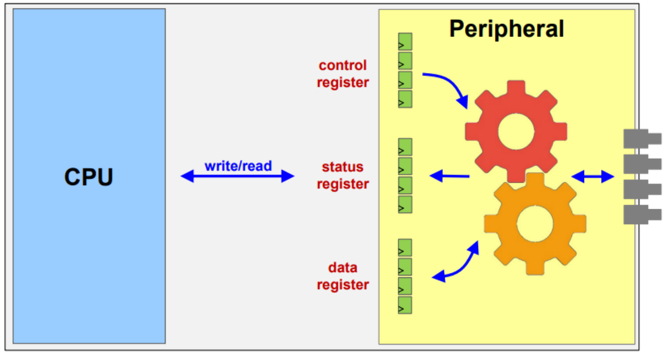
- configurable hardware blocks of a microcontroller
- accepts a specific task from the CPU, executes task and returns result (status, e.g. task completion, error)
- oftentimes interfaces to the outside world
  many (not all) interact with external MCU pins (grey things very right side of image)
- examples: GPIO, UART, SPI, ADC...



Peripheral Registers

the CPU controls and monitors Peripherals through registers

- Registers are arrays of flip-flops (storage elements with two states, i.e. 0 or 1)
- Each flip-flop stores one bit of information
- CPU writes to and reads from registers



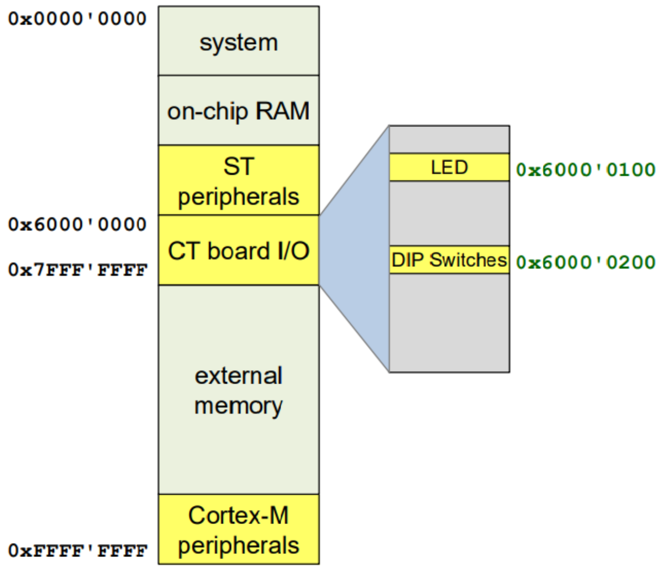
A Peripheral typically has multiple registers that can be categorized as:

- **Control Registers**
  enable CPU to configure the peripheral
- **Status Registers**
  enable CPU to monitor the peripheral
- **Data Registers**
  enable CPU to exchange data with the peripheral

Memory-mapped Peripheral Registers

Distinction between LEDs and DIP switches on the CT-Board:

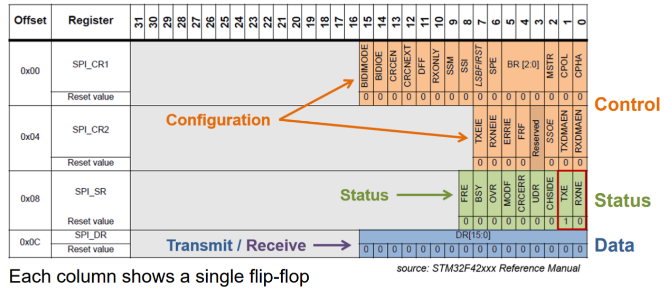
- Control register → controls states of LEDs
- Status register → monitors states of DIP switches



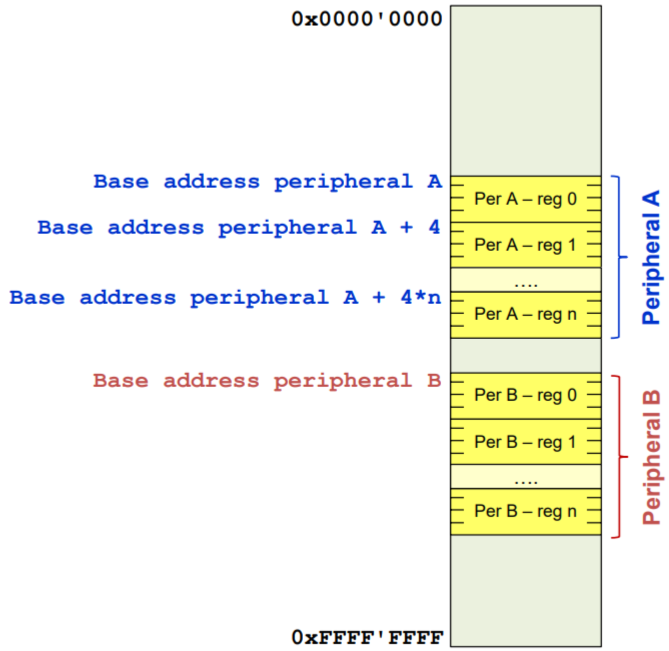
CPU access to individual peripheral registers

- ARM & STM map the peripheral registers into the memory address range
- Reference Manual shows the defined addresses

Example SPI (Serial Peripheral Interface)



Each column shows a single flip-flop



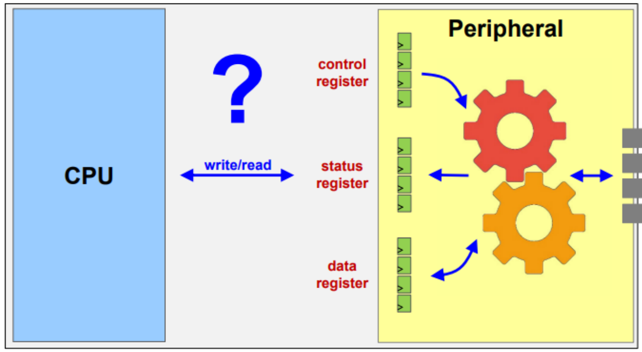
Accessing control registers in C

dereference pointer
cast

```
#define LED31_0_REG     (*((volatile uint32_t *)(0x60000100)))

#define BUTTON_REG      (*((volatile uint32_t *)(0x60000210)))

// Write LED register to 0xBBCC'DDEE
LED31_0_REG = 0xBBCCDDEE;

// Read button register to aux_var
aux_var = BUTTON_REG;
```

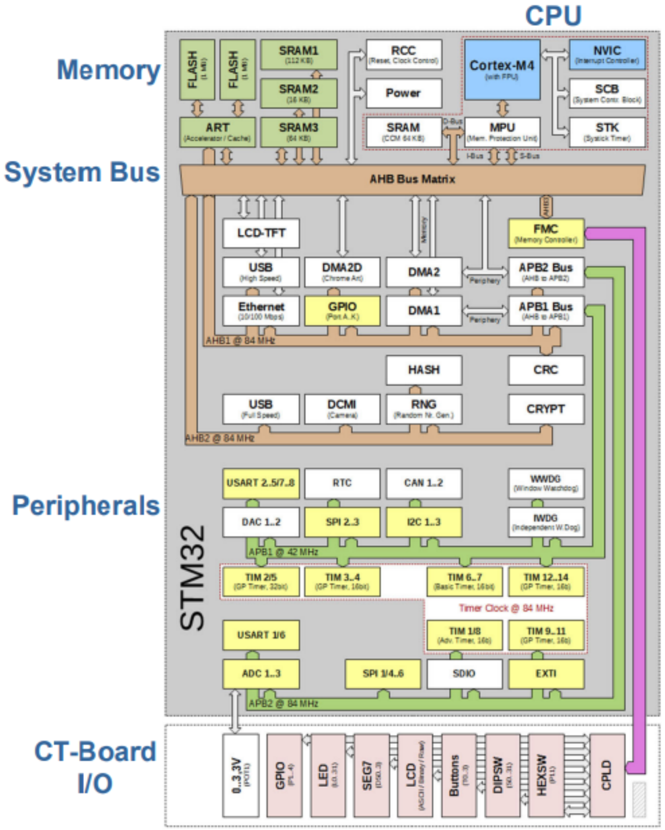
CPU read/write to peripheral registers How does the CPU write to and read from peripheral registers?

- CPU reads/writes to peripheral registers
- CPU uses memory-mapped I/O to access peripheral registers
- CPU uses load/store instructions to access peripheral registers



⇒ System Buses

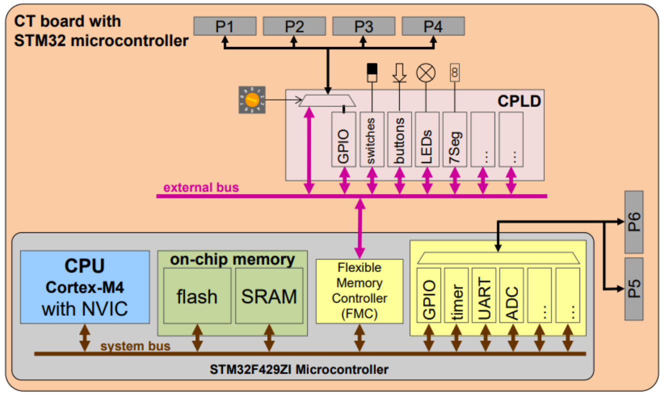
STM32 Microcontroller with CPU, on-chip memory, and peripherals interconnected through the system bus(es)



A distributed system with parallel (simultaneous) processing of data in many peripherals. All under the supervision of the CPU.

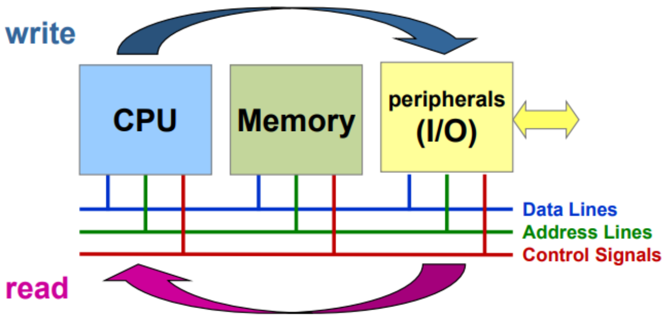
Note: ARM calls their system buses AHB (ARM High-performance Bus) and APB (ARM Peripheral Bus). On complex chips, it is state-of-the-art to partition the system bus into multiple interconnected buses.

CT Board with STM32 Microcontroller and Buses



System Bus

- Interconnects CPU with memory and peripherals
- CPU acts as master: initiating and controlling all transfers
- Peripherals and memory act as slaves: responding to requests from the CPU
- System bus is a shared resource

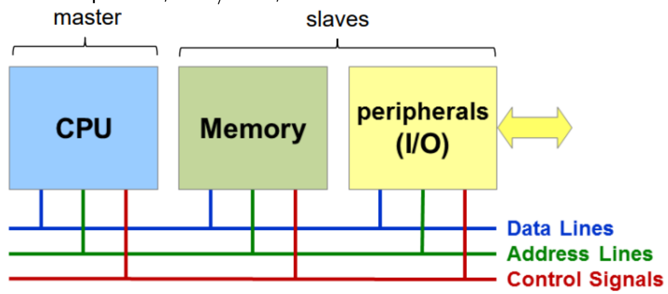


Bus Specification

- **Protocol and operations**
- **Signals**
  - Number of Signals
  - Signal descriptions
- **Timing**
  - Frequency
  - Setup and hold times
- **Electrical properties** (not in exam)
  - Drive strength
  - Load
- **Mechanical requirements** (not in exam)

Signal Groups

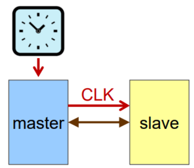
- **Data lines**
  - Bidirectional (read/write)
  - Number of lines → data bus width (8, 16, 32, 64 parallel lines of data)
  - Example: Cortex-M has 32 address lines → 4GB address space → $0x00000000$ to $0xFFFFFFFF$
- **Address lines**
  - Unidirectional: from Master to slaves
  - Number of lines → size of address space
- **Control signals**
  - Control read/write direction
  - Provide timing information
  - Chip select, read/write, etc.



Bus Timing Options

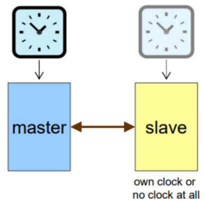
Synchronous

- Master and slaves use a common clock
  Often a dedicated clock signal from master to slave, but clock can also be encoded in a data signal
- Clock edges control bus transfer on both sides
- Used by most on-chip buses
- Off-chip: DDR and synchronous RAM



Asynchronous

- Slaves have no access to clock of the master
- Control signals carry timing information to allow synchronization
- Widely used for low data-rate off-chip memories → parallel flash memories and asynchronous RAM



Multiple devices driving the same data line

What if one device drives a logic 1 (Vcc) and another device drives a logic 0 (Gnd)?
$\Rightarrow$ Electrical short circuit! $rightarrow$ bus contention (SStreitigkeit")

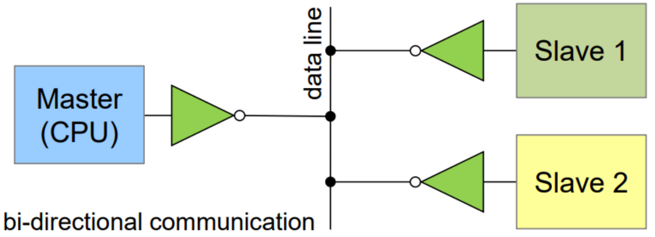


Figure only shows output paths, input paths are not shown.

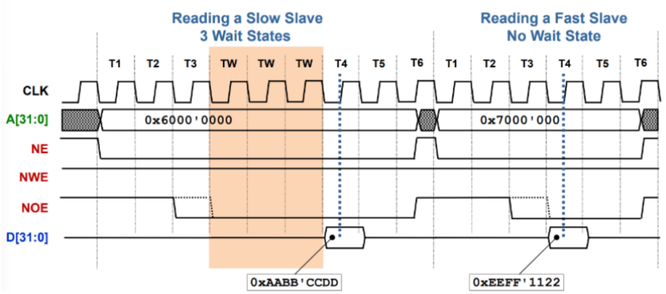
CPU defines who drives the data bus at which moment in time:

- `write` CPU drives bus $rightarrow$ all slave drivers disconnected
- `read` CPU releases bus $rightarrow$ one slave drives bus (selected through values on address lines, other slave drivers disconnected)

Electrically disconnecting a driver is called **tri-state** or **high-impedance** (Hi-Z) state. (switch)
textbfCHECK IF CORRECT

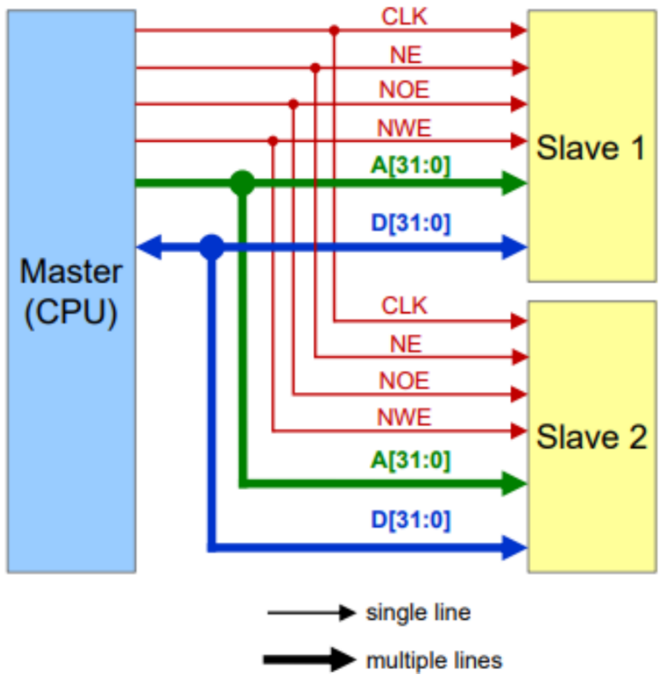
Slow Slaves

Wait states are inserted to slow down the CPU to match the speed of the slowest peripheral (depending on the address of an access)



Block Diagram

- Address lines [31:0]
- Data lines [31:0]
- Control signals
  - CLK $\rightarrow$ clock
  - NE $\rightarrow$ Not enable
  - NWE $\rightarrow$ Not write enable
  - NOE $\rightarrow$ Not output enable



Timing Diagram

- `write` D[:] to A[:] $\rightarrow$ NE, NWE $= 0$
- `read` D[:] from A[:] $\rightarrow$ NE, NOE $= 0$

Address Decoding

- CPU uses address lines to select a peripheral
- Each peripheral has a unique address range
- Address decoding logic generates a chip select signal for each peripheral