Computertechnik 2

Jil Zerndt
FS 2025

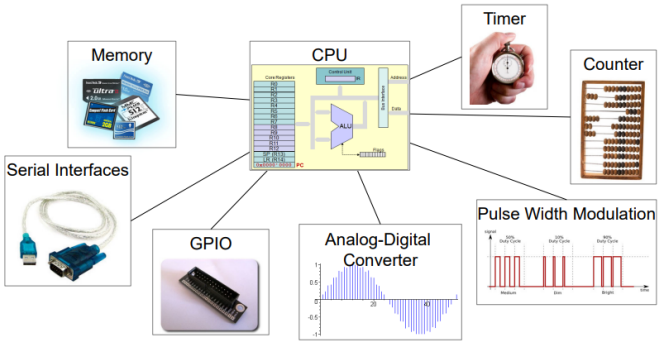
Microcontroller

- Microcontroller
- System Bus
- Digital Logic Basics
- Synchronous Bus
- Control and Status Registers
- Address Decoding
- Slow Slaves (Peripherals)
- Bus Hierarchies
- Accessing Control Registers in C
- Conclusions

Embedded Systems

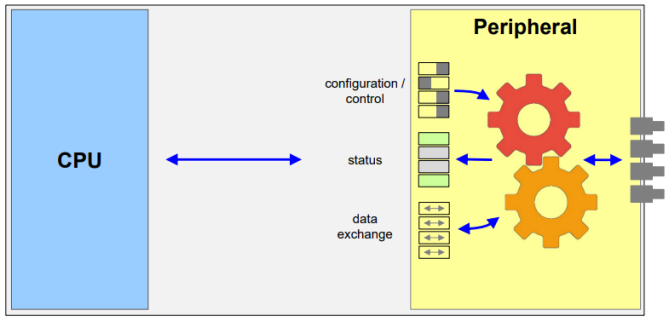
- low cost (usb sticks, consumer electronics)
- low power (sensor networks, mobile devices)
- small size (smart cards, wearables)
- real time (anti-lock brakes, process control)
- reliability (medical devices, automotive)
- extreme environment (space, automotive)

Single Chip Solution ⇒ CPU with integrated memory and peripherals



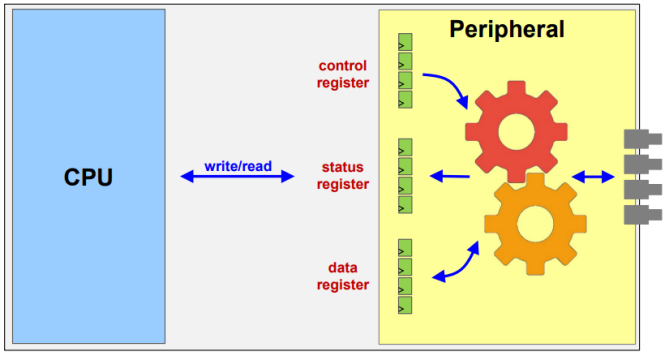
Peripherals

- 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

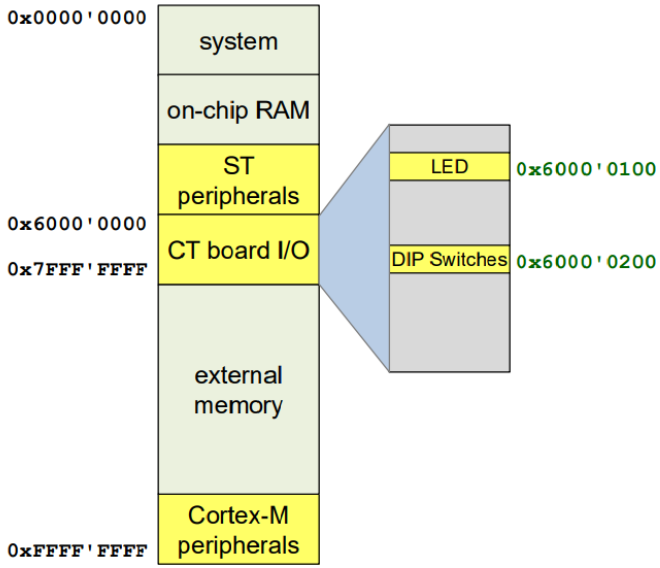


A Peripheral typically has multiple registers that can be categorized as:

- **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:

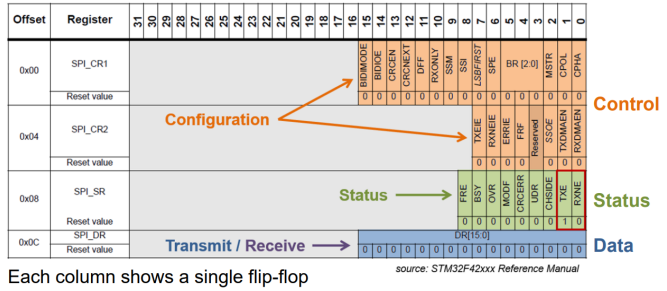
- Control register → controls states of LEDs
- Status register → monitors states of DIP switches



CPU access to individual peripheral registers

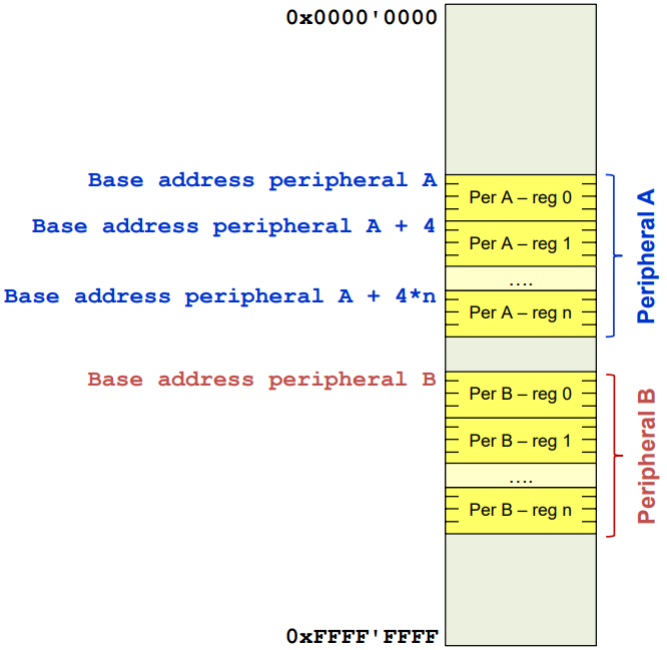
- ARM & STM map the peripheral registers into the memory address range
- Reference Manual shows the defined addresses

Example SPI (Serial Peripheral Interface)



source: STM32F42xxx Reference Manual

Each column shows a single flip-flop

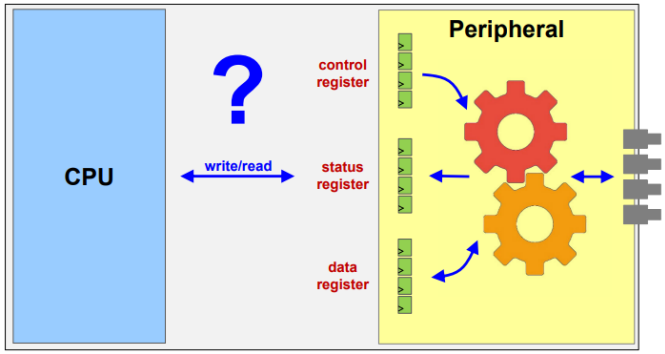


Accessing control registers in C

```
                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;
```

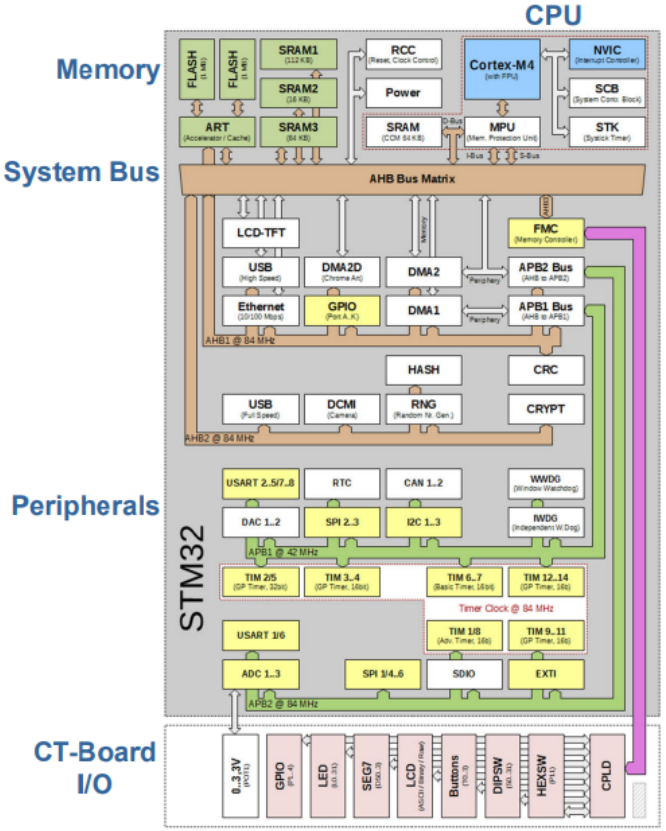
CPU read/write to peripheral registers How does the CPU write to and read from peripheral registers?

- 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

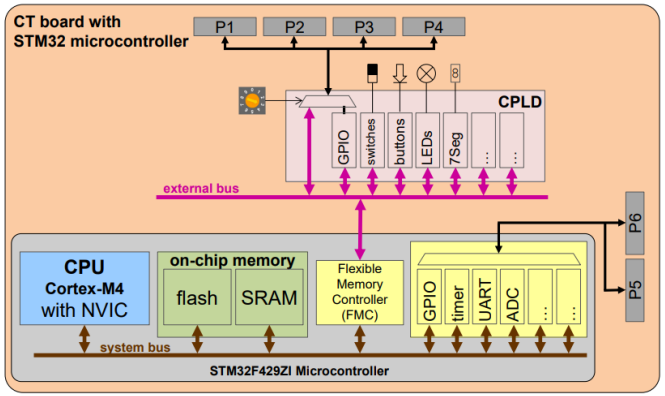
STM32 Microcontroller with CPU, on-chip memory, and peripherals interconnected through the system bus(es)



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

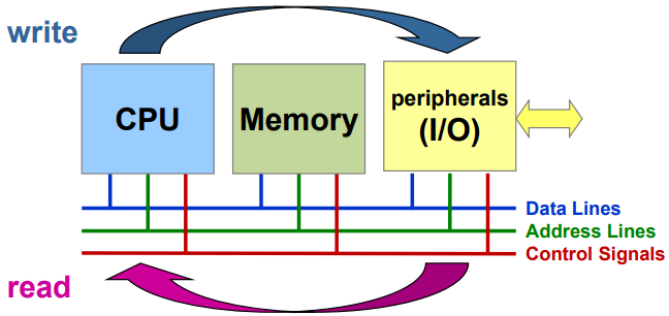
Note: ARM calls their system buses AHB (ARM High-performance Bus) and APB (ARM Peripheral Bus). On complex chips, it is state-of-the-art to partition the system bus into multiple interconnected buses.

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

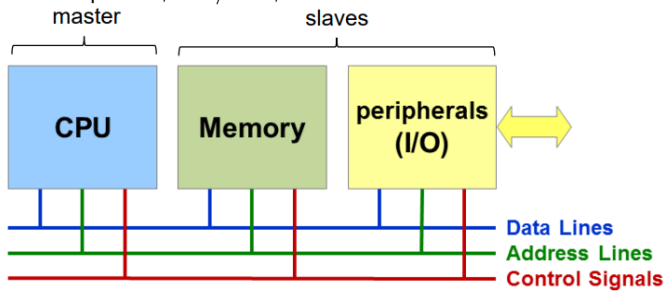


Bus Specification

- **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.



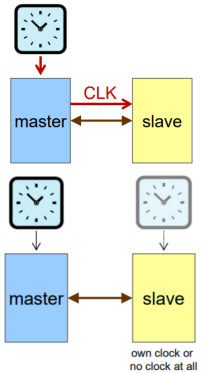
Bus Timing Options

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)?
⇒ Electrical short circuit! $rightarrow$ bus contention (SStreitigkeit")

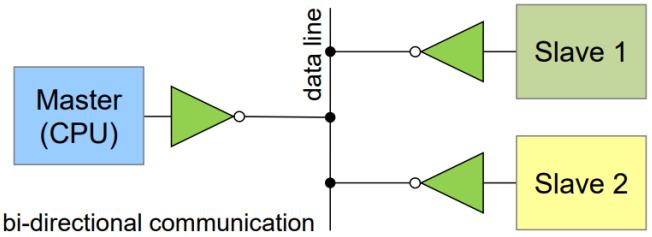


Figure only shows output paths, input paths are not shown.

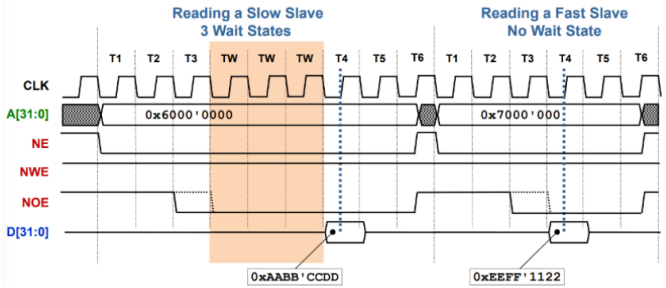
CPU defines who drives the data bus at which moment in time:

- `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

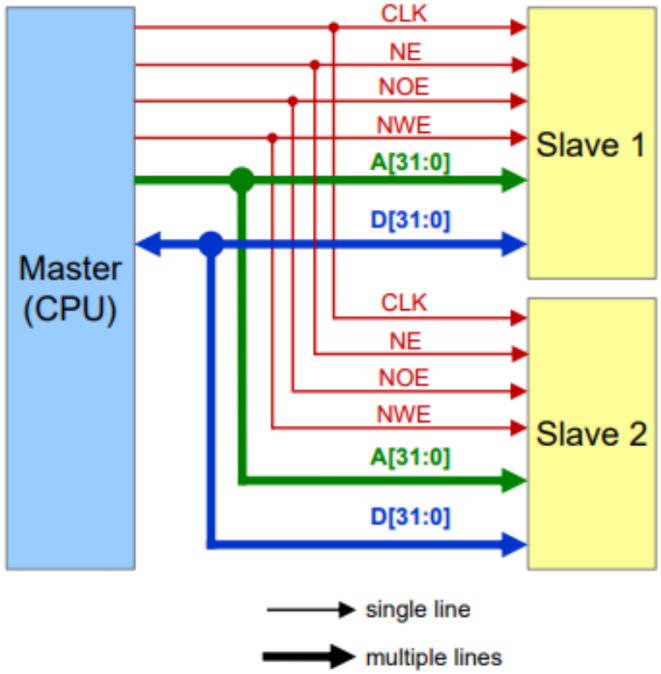
Slow Slaves

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 → clock
  - NE → Not enable
  - NWE → Not write enable
  - NOE → Not output enable



Control Bits

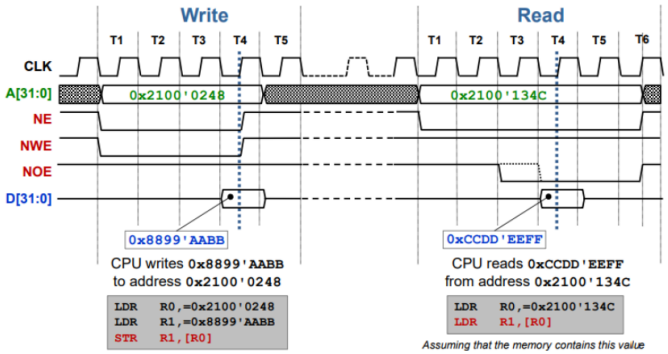
- Allow CPU to configure Slaves
- CPU writes to register bit to configure Slave
- Slave uses output of register bit to configure itself
- Example: SPI Slave Select (SS) bit
- Usually read/write access to control bits

Status Bits

- Allow CPU to monitor Slaves
- CPU reads register bit to monitor Slave
- Slave uses input of register bit to monitor itself (Slave writes to register bit)
- Example: SPI Busy bit
- Usually read-only access to status bits

Timing Diagram

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



Bus Access Size

is determined by the NBL (0-3) (No Byte Line) signals

- NBL = 1 → Byte used for Read/Write
- NBL = 00
- NBL[0:3] = 0011 → Read Halfword
- NBL[0:3] = 1111 → Read Word

CHECK IF CORRECT

Address Decoding

Interpretation of address line values. See whether bus access targets a particular address or address range.

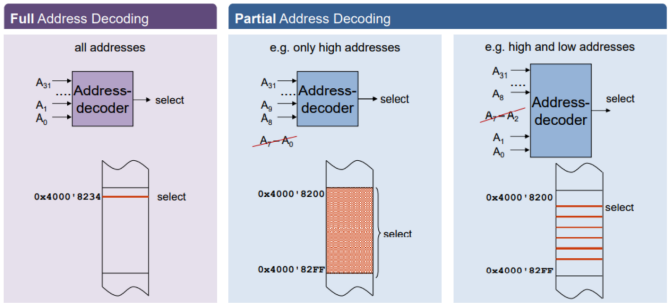
- CPU uses address lines to select a peripheral
- Each peripheral has a unique address range
- Address decoding logic generates a chip select signal for each peripheral

Full Address Decoding

- All address lines are decoded
- A control register can be accessed at exactly one location
- 1:1 mapping: A unique address maps to a single hardware register

Partial Address Decoding

- Only a subset of address lines are decoded
- A control register can be accessed at multiple locations
- 1:n mapping: Multiple addresses map to the same hardware register
- Map a hardware register to several addresses



GPIO

General Purpose Input Output

Register Address = Base address + Offset
- Offset is given for each register in reference Manual
- Base address is defined in memory map (reference manual)

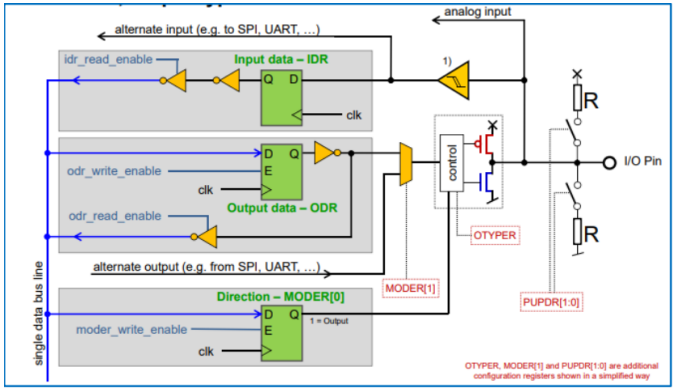
GPIO

Situation:
- Microcontroller as a general purpose device
- Many functional blocks included

Problem:
- Limited number of pins
- Many functions to be implemented (many functional blocks included)

Solution:
- Many (all) pins are configurable as GPIO
- Select the needed I/O pins and functions
- «pin sharing»
- Output multiplexer needs to be configured

Structure



Configuration Registers

- **GPIOx_MODER** (Mode Register)
- **GPIOx_OTYPER** (Output Type Register)
- **GPIOx_OSPEEDR** (Output Speed Register)
- **GPIOx_PUPDR** (Pull-up/Pull-down Register)

Data Operations

- Input: Read register **GPIOx_IDR** (Input Data Register)
- Output: Write register **GPIOx_ODR** (Output Data Register) or **GPIOx_BSRR** (Bit Set Reset Register)

Setting and Clearing Bits GPIOx_BSRR

- 0-15: Set Bits → 1: Set, 0: No Change
- 16-31: Reset/Clear Bits → 1: Reset, 0: No Change
- Ensures atomic access in software (no interruption possible)

Configuration

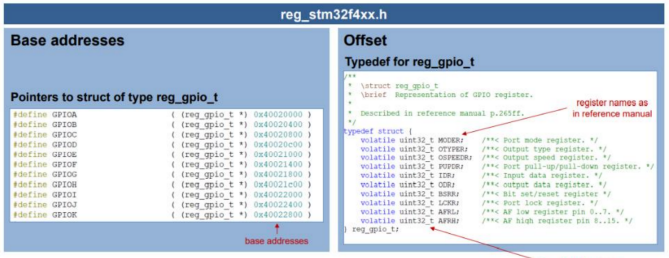
- **Mode:** Input, Output, Alternate Function, Analog
- **Type:** Push-Pull, Open Drain
- **Speed:** Low, Medium, High, Very High
- **Pull-Up/Pull-Down:** No, Pull-Up, Pull-Down, Reserved

Hardware Abstraction Layer (HAL)

```
#define ADDR (* ((volatile uintXX_t *) (0x40020000)))
```

```
#define GPIOA_MODER     (*((volatile uint32_t *)(0x40020000)))
```

Accessing a register:
- each GPIO port has the same 10 registers
- there are 11 GPIO ports → GPIOA to GPIOI




```
GPIOA->MODER = 0x55555555;    // all output
```