Embedded System
Southern University of Science and Technology
Mengxuan Wu
12212006

Final Review

Mengxuan Wu

1 Introduction

1.1 MPU, MCU, SoC

- **Microprocessor Unit (MPU)**: A MPU is a single processor that can be used for general-purpose computing. All data processing and control logics are implemented in the MPU. However, it requires additional components to function as a complete computer system, such as memory, input/output devices, and other components.
- **Microcontroller Unit (MCU)**: A MCU typically has a single processor as its core. It includes memory, input/output devices, and other basic peripherals on a single chip. It is typically used in embedded systems, where a complete computer system is not required.
- **System on Chip (SoC)**: A SoC can have single or multiple processors, memory, input/output devices, and other peripherals on a single chip. It may integrate more powerful blocks than MCU, such as a GPU, DSP, or FPGA. It is capable of running complex operating systems and applications.

1.2 Embedded System

An embedded system is typically designed to handle a particular task only. "Embedded systems are information processing systems that are embedded into an enclosing product" Thus, compared to general-purpose computers, embedded systems are more specialized and optimized for specific tasks, and includes only the core and necessary components.

1.3 ARM Based Microcontroller

- **ARM Architecture**: ARM (Advanced RISC Machine) is a family of Reduced Instruction Set Computing (RISC) architectures for computer processors.
- **ARM Cortex-A Series**: The ARM Cortex-A series is a family of application processors designed for high-performance applications.
-
- **ARM Cortex-R Series**: The ARM Cortex-R series is a family of real-time processors designed for real-time applications.

- **ARM Cortex-M Series**: The ARM Cortex-M series is a family of microcontroller cores designed for embedded applications. The Cortex-M cores are optimized for low power consumption and high performance. The Cortex-M series includes the Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M3, Cortex-M4, Cortex-M7, and Cortex-M23/M33.

Companies like STMicroelectronics, NXP, and Texas Instruments that produce ARM-based microcontrollers, needs to pay "Architectural License Fee". Specifically, they pay the fee for the right to use the ARM ISA. However, the peripherals and other components are designed by the companies themselves.

1.4 Minimum System

A minimal system refers to a system that includes the core components necessary for the microcontroller to function. The core components include the microcontroller, power supply, clock source, and reset circuit.

2 STM32 GPIO

2.1 Numbering System

For 32-bit microcontrollers (like ARM), a word is 32 bits or 4 bytes, and a half-word is 16 bits or 2 bytes.

Some common prefixes for binary numbers are:

- **K**: Kilo, $2^{10} = 1024$
- **M**: Mega, $2^{20} = 1,048,576$
- **G**: Giga, $2^{30} = 1,073,741,824$
- **T**: Tera, $2^{40} = 1,099,511,627,776$
- **P**: Peta, $2^{50} = 1,125,899,906,842,624$
- **E**: Exa, $2^{60} = 1,152,921,504,606,846,976$
- **Z**: Zetta, $2^{70} = 1,180,591,620,717,411,303,424$

For small numbers, another set of prefixes are used:

- **m**: milli, $10^{-3} = 0.001$
- μ: micro, $10^{-6} = 0.000001$
- **n**: nano, $10^{-9} = 0.000000001$
- **p**: pico, $10^{-12} = 0.000000000001$
- **f**: femto, $10^{-15} = 0.000000000000001$

To convert a number with $n+m$ digits in base r to decimal, we can use the following formula:

$$\begin{aligned} D &= \overline{d_{n-1}\cdots d_1 d_0 . d_{-1} d_{-2} \cdots d_{-m}} \\ &= d_{n-1}r^{n-1} + \cdots + d_1 r^1 + d_0 r^0 + d_{-1}r^{-1} + \cdots + d_{-m}r^{-m} \\ &= \sum_{i=-m}^{n-1} d_i r^i \end{aligned}$$

To convert a decimal number to base r, we can use the following algorithm:

	Quotient	Remainder	Coefficient
$13 \div 2$	6	1	$a_0 = 1$
$6 \div 2$	3	0	$a_1 = 0$
$3 \div 2$	1	1	$a_2 = 1$
$1 \div 2$	0	1	$a_3 = 1$

Integer Part

	Integer	Fraction	Coefficient
0.375×2	0	0.75	$a_{-1} = 0$
0.75×2	1	0.5	$a_{-2} = 1$
0.5×2	1	0	$a_{-3} = 1$

Fractional Part

To be noticed, the coefficients are read in **different order**. For the integer part, we read the coefficients from bottom to top. For the fractional part, we read the coefficients from top to bottom.

1's complement is the bitwise negation of a binary number. 2's complement is the 1's complement plus 1. For negative numbers, modern computers use 2's complement to represent them, since arithmetic operations are easier to implement.

2.2 Computer Architecture

- **Von Neumann Architecture**: In the Von Neumann architecture, the CPU can execute instructions and read/write data from/to the same memory. The CPU and memory are connected by a single bus. The program and data are stored in the same memory.
- **Harvard Architecture**: In the Harvard architecture, the CPU has separate memory for instructions and data. The CPU can fetch instructions and read/write data simultaneously. The program and data are stored in separate memories.

Cortex-M microcontrollers use different architectures. For example, the Cortex-M0 uses the Von Neumann architecture, while the Cortex-M3 uses the Harvard architecture.

2.2.1 ALU

The Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU can perform operations such as addition, subtraction, multiplication, division, AND, OR, XOR, and NOT. It also performs shift and rotate operations.

NZCV flags are used to store the results of arithmetic operations. The NZCV flags are:

- **N**: Negative flag, set if the result is negative. N = 1 if the most significant bit is 1.

- **Z**: Zero flag, set if the result is zero. Z = 1 if all bits are 0.
- **C**: Carry flag, set if there is a carry out of the most significant bit.
  - For unsigned addition, C = 1 if there is a carry out of the most significant bit.
  - For unsigned subtraction, C = 1 if there is no borrow.
  - For shift and rotate operations, C = the last bit shifted out.
- **V**: Overflow flag, set if there is an overflow. If adding the same sign numbers results in a different sign number, V = 1. Non-arithmetical operations (like shift and rotate) do not affect the V flag.

For most arithmetic operations, the NZCV flags are only set if the command has a suffix of "S" (e.g. ADDS, SUBS, MULS, etc.). For example, the command "ADD R0, R1, R2" does not set the NZCV flags, while the command "ADDS R0, R1, R2" sets the NZCV flags.

Some instructions like "CMP" always set the NZCV flags, since the result is not stored in any register.

Another important detail is about shift. Arithmetic shift is a shift operation that preserves the sign bit. Logical shift is a shift operation that fills the empty bits with 0.

2.2.2 Memory

The memory in ARM is byte-addressable, which means each address points to a byte. For 32-bit microcontrollers, the maximum addressable memory is 2^{32} bytes or 4 GB.

There are two endianess: big-endian and little-endian. In big-endian, the most significant byte is stored at the smallest address. In little-endian, the least significant byte is stored at the smallest address. ARM uses little-endian. ARM chooses little-endian.

2.2.3 CPU

CPU has a series of registers, including:

- **Low Registers**: R0 to R7 are low registers. They can be accessed by any instruction.
- **High Registers**: R8 to R12 are high registers. They can only be accessed by some instructions.
- **General Purpose Registers**: R0 to R12 are also called general purpose registers.
- **Stack Pointer (SP)**: SP is used to point to the top of the stack. In Cortex-M3, there are two stack pointers: MSP (Main Stack Pointer) and PSP (Process Stack Pointer). MSP is used in privileged mode, while PSP is used for application code.
- **Link Register (LR)**: LR is used to store the return address of a subroutine.
- **Program Counter (PC)**: PC is used to store the address of the next instruction to be executed.
- **Program Status Register (PSR)**: PSR is used to store the status of the CPU. It includes the NZCV flags, the interrupt status, and the execution mode.

2.3 GPIO

STM32 microcontrollers use memory-mapped I/O to control the GPIO pins. The GPIO registers are located in the memory space. The pins are named in Groups as port A, port B, port C, etc. Each port has 16 pins, named from 0 to 15.

By default, the pins are set as floating input. To program the pins, the following steps are needed:

1. Enable the clock for the GPIO port. `RCC->APB2ENR` is used.
2. Configure the mode of the pin. CRL and CRH registers are used. Each of the 16 pins has 4 bits to configure, where CNF and MODE are 2 bits each (MODE on the lower bits).
3. (Optional) Configure pull up or pull down for input pins.

Pull up means the pin is connected to VCC through a resistor, and the pin is high when the button is not pressed. Pull down means the pin is connected to GND through a resistor, and the pin is low when the button is not pressed.

3 C Programming

3.1 Macro

A macro is a fragment of code that has been given a name. Whenever the name is used, it is replaced by the contents of the macro. Macros are defined using the `#define` directive.

Macros serve similarly as an inline function. Thus, it does not require saving the return address and other overheads. However, if switching overhead is shorter than the execution time of the function, it is better to use a function.

3.2 Struct and Union

A struct is a user-defined data type that groups related data together. A union is a user-defined data type that allows storing different data types in the same memory location.

To be aware that memory alignment is important. A word must be aligned to a multiple of 4 bytes, a half-word must be aligned to a multiple of 2 bytes, and a byte can be stored anywhere. However, the members of a struct are stored in memory in the order they are declared. Thus, the size of a struct may be larger than the sum of the sizes of its members.

3.3 Storage Class

There are four storage classes in C: auto, register, static, and extern.

- **auto**: Any variable declared inside a function without a storage class is automatically an auto variable, and they are only accessible within the function.
- **register**: The register storage class is used to define local variables that should be stored in a register. It is used to optimize the access time of variables.

- **static**: The static storage class is used to declare static variables that are preserved between function calls. Static variables are initialized only once.
- **extern**: The extern storage class is used to declare variables that are defined in another file.

3.4 Volatile

The volatile keyword is used to tell the compiler that a variable may change at any time without any action being taken by the code (by outside sources). The volatile keyword is used to prevent the compiler from optimizing the code.

4 ARM Assembly

4.1 Immediate Addressing

Immediate addressing is used to load a constant value into a register. The constant value is specified in the instruction itself. For example, the instruction `MOV R0, #0x1234` loads the value 0x1234 into register R0.

Although 12 bits are used to store the immediate value, only 8 bits are used for the value. The remaining 4 bits are used to specify the rotation. The final value is calculated the 8-bit value right rotated by 2 times the rotation value.

4.2 Indirect Addressing

Indirect addressing is used to load a value from memory into a register. The address of the memory location is stored in a register. For example, the instruction `LDR R0, [R1]` loads the value from the memory location pointed to by R1 into R0.

Base plus offset addressing is a special case of indirect addressing. The address of the memory location is calculated by adding an offset to the base address stored in a register. For example, the instruction `LDR R0, [R1, #4]` loads the value from the memory location pointed to by R1 plus 4 into R0.

Automatic update addressing is a special case of base plus offset addressing. The offset is added to the base address, and the base address is updated to the new address. In pre-indexed addressing, the offset is added to the base address before the value is loaded, written as `LDR R0, [R1, #4]!`. In post-indexed addressing, the offset is added to the base address after the value is loaded, written as `LDR R0, [R1], #4`.

5 Interrupt

5.1 Subroutine

A subroutine is a sequence of instructions that performs a specific task. Subroutines are used to break down a large program into smaller, more manageable parts. Subroutines are also called functions or procedures.

In ARM assembly, a subroutine is called using the `BL` instruction. The `BL` instruction stores the current PC value adding 4 to the link register (LR) and jumps to the subroutine. The `BX LR` instruction is used to return from the subroutine.

If the subroutine is called from another subroutine, context switching is needed. Command `PUSH {R4, LR}` is used to save the context, and command `POP {R4, LR}` is used to restore the context. ARM uses full descending stack to store context, which means the stack grows from high memory to low memory. The stack pointer points to the last used memory location.

5.2 Interrupt

An interrupt is a signal to the processor that an event has occurred that requires immediate attention. Interrupts can be generated by hardware devices or software. Interrupts are used to handle time-critical events, such as I/O operations, real-time processing, and error handling.

In ARM assembly, the following steps happen when an interrupt occurs:

1. The processor saves 8 registers (R0 to R3, R12, LR, PC, and PSR) to the stack.
2. Performs interrupt service routine (ISR) by loading the address of the ISR into the PC. The addresses of the ISR are stored in the vector table.
3. Restores the 8 registers from the stack.

Nested vectored interrupt controller (NVIC) is used to manage interrupts. The NVIC is a peripheral that receives interrupt requests from different sources and decides which interrupt to service first. For each interrupt, the NVIC has a preempt-priority level, a sub-priority level, and a natural priority level. The principle of priority is:

1. The smaller the number, the higher the priority.
2. Higher preempt-priority level interrupts can preempt lower preempt-priority level interrupts.
3. When two interrupts have the same preempt-priority level, the sub-priority level is used to determine which interrupt to service first. However, they cannot preempt each other.
4. When two interrupts have the same preempt-priority level and sub-priority level, the natural priority level is used to determine which interrupt to service first.

The priority group register (AIRCR) is used to configure the number of bits used for preempt-priority and sub-priority levels. The priority register (IPR) is used to set the preempt-priority and sub-priority levels for each interrupt. By default, 2 bits are used for preempt-priority and 2 bits are used for sub-priority.

6 UART

6.1 UART Protocol

UART stands for Universal Asynchronous Receiver/Transmitter. It is a serial communication protocol that is used to transfer data between two devices. UART is asynchronous, which means that the data is transmitted without a clock signal. UART is also full-duplex, which means that data can be transmitted and received at the same time, by using two separate data lines.

A UART data frame consists of:

- **Start Bit**: The start bit is used to indicate the beginning of the data frame. It is always low (0). Since the line is high (1) when idle, the start bit is used to notify the receiver that data is coming.
- **Data Bits**: The data bits are the actual data being transmitted. The number of data bits can be 5, 6, 7, or 8 bits. The LSB is transmitted first.
- **Parity Bit**: The parity bit is used for error checking. It can be odd, even, or none.
- **Stop Bit**: The stop bit is used to indicate the end of the data frame. It is always high (1).

The baud rate is the rate at which data is transmitted over a UART connection. It is measured in bits per second (bps). The transmission rate is the number of data bits transmitted per second.

6.2 Problems in UART

6.2.1 Synchronization

Without a clock signal, the receiver needs to know when to sample the data. The receiver uses the start bit to synchronize with the transmitter. The receiver samples the data bits in the middle of the bit period.

The final stop bit is sampled after 9.5 bit periods. Thus, the maximum error is 0.5 bit periods, or 5%. Considering both the transmitter and receiver, the maximum error is 2.5%.

6.2.2 Glitch

A glitch is a short-lived signal that can occur when the signal changes. Glitches can cause errors in the received data. To prevent glitches, the receiver over-samples the data. The receiver samples the data at least 16 times per bit period, and the central 3 samples are used for voting.

6.2.3 Error Detection

Parity bit is used for error detection. Even parity means the number of 1s in the data bits and the parity bit is even. Odd parity means the number of 1s in the data bits and the parity bit is odd. If the parity is incorrect, an error is detected.

6.3 STM32 UART

In STM32 microcontrollers, the `USART_BRR` register is used to set the baud rate. The lower 4 bits are used to set the fraction part, and the upper 12 bits are used to set the integer part. The baud rate is calculated as:

$$\text{USARTDIV} = \text{Integer part} + \text{Fraction part}/16$$

$$\text{Baud rate} = \frac{\text{Clock frequency}}{16 \times \text{USARTDIV}}$$

7 I2C

I2C stands for Inter-Integrated Circuit. It is a serial communication protocol that is used to transfer data between two devices. I2C is a multi-master, multi-slave protocol, which means that multiple devices can communicate on the same bus. I2C is synchronous, which means that the data is transmitted with a clock signal. I2C is half-duplex, which means that data can be transmitted and received, but not at the same time.

By default, the I2C bus is in an idle state, with both the SDA and SCL lines high. To start a communication, the master device sends a start condition by pulling the SDA line low while the SCL line is high. The master then sends the 7-bit address of the slave device it wants to communicate with, followed by a read/write bit. The slave device with the matching address responds with an acknowledge bit. The master then sends or receives data from the slave device. Each byte of data is followed by an acknowledge bit. The communication is ended with a stop condition, where the SDA line is pulled high while the SCL line is high.

I2C sends MSB first. For read/write bit, 0 means master write and 1 means master read. For acknowledge bit, 0 means acknowledge and 1 means no acknowledge. When master writes, the slave sends ACK or NAK to tell the master if the data is received correctly. When master reads, the master sends a NAK to tell the slave to stop sending data.

Master can repeatedly start a communication, i.e. sending a start condition (slave address) without sending a stop condition for previous communication. This is called repeated start condition. It is used if the master wants to read and write data from/to the same slave device. Without sending a stop condition, the master can hold the bus and continue the communication.

To solve interference, the master always listens to the bus. If the master sends a 1 but reads a 0, it means there is interference. The master should stop the communication to give the bus to other devices.

8 SPI

SPI stands for Serial Peripheral Interface. It is a serial communication protocol that is used to transfer data between one master device and one or more slave devices. SPI is a full-duplex protocol, which means that data can be transmitted and received at the same time. SPI is synchronous, which means that the data is transmitted with a clock signal.

In SPI, there is one master device and one or more slave devices. The master device generates the clock signal and controls the data transfer. The master device selects the slave device it wants to communicate with by pulling the slave select (SS) line low. The master device sends data to the slave device on the MOSI (Master Out Slave In) line, and the slave device sends data to the master device on the MISO (Master In Slave Out) line.

The combination of the clock polarity (CPOL) and clock phase (CPHA) determines the clock signal. CPOL determines the idle state of the clock signal. CPHA is 0 if the data is sampled on the first edge of the clock signal, and 1 if the data is sampled on the second edge of the clock signal.

9 Timer

9.1 Clock Tree

The clock tree is used to generate different clock signals for different peripherals. In STM32 microcontrollers, LSI and LSE are used for independent watchdog and RTC, HSI and HSE are used for the system clock. PLL is used to increase the frequency of the system clock. The peripheral clock is derived from the system clock by dividing the system clock.

External clocks are generated oscillators that are connected to the microcontroller. Internal clocks are generated by RC oscillators that are integrated into the microcontroller.

9.2 Timer in MCU

For each timer, there are three values: the prescaler, the auto-reload value, and the counter value. The prescaler is used to divide the clock frequency. The auto-reload value is used to set the period of the timer. The counter value is used to store the current value of the timer. They are stored in the `TIMx_PSC`, `TIMx_ARR`, and `TIMx_CNT` registers.

The `TIMx_CR1` register is used to configure the timer. The `CEN` bit is used to enable the timer. The `OPM` bit is used to enable one-pulse mode (the timer stops after one cycle). The `CMS` and `DIR` bits are used to configure the counting mode.

The `TIMx_SR` register is used to check the status of the timer. The `UIF` bit is used to check if the update event has occurred. This bit will be set when the counter overflows in up-counting mode or underflows in down-counting mode. The behavior in up and down counting mode is different. The underflow event happens when the counter reaches 1 when counting down and the overflow event happens when the counter reaches the auto-reload value minus 1 when counting up. And both of them will set the `UIF` bit.

9.3 Output Compare & PWM

The `OCnM` register is used to configure the output compare mode. An interrupt can be generated when the counter value matches the value in the `CCR` register and the `CCnIF` bit is set.

The PWM mode is used to generate a PWM signal. The duty cycle of the PWM signal is determined by the value in the `CCR` register. The PWM signal is generated by comparing the counter value with the value in the `CCR` register. In PWM mode 1, the output is high when the counter value is less than the `CCR` value and low otherwise. PWM mode 2 is the opposite.

9.4 Input Capture

The input capture mode is used to capture the time interval between two events. The timer can capture rising or falling edges of an external signal. The counter value is stored in the `CCR` register and an interrupt is generated when the capture event occurs.

9.5 Systick Timer

The Systick timer is a 24-bit timer that is used to generate periodic interrupts. It can only count down and generates a Systick interrupt when the counter reaches 0. When

writing to its value register, the counter is reset to 0 without generating an interrupt.

10 File System

10.1 Storage

There are two types of storage: volatile storage and non-volatile storage. Volatile storage is temporary and is lost when the power is turned off. Non-volatile storage is permanent and retains data when the power is turned off.

Volatile storage includes SRAM and DRAM. SRAM is faster and more expensive than DRAM. DRAM is slower and cheaper than SRAM. Non-volatile storage includes Disk, ROM, EPROM, EEPROM, and Flash.

MCU uses Flash memory and EEPROM to store data. EEPROM probides byte-level read and write. Flash memory provides sector-level read and write, however sector erase is needed before writing.

10.2 File System

FAT (File Allocation Table) is a file system that is used to organize and manage files on a storage device. FAT is a simple file system that is supported by most operating systems. FAT uses a file allocation table to keep track of the files on the storage device.

With FAT, the storage device is divided into clusters. Each cluster is a fixed-size block of data. A root directory is used to store the file's initial cluster number, file name, and file attributes. The file allocation table is used to store the cluster number and the next cluster number of each cluster.