Chapter 1

Introduction of Cortex-M4

1.1 Processor Registers

- Length of the register is 32 bits.
- General-purpose registers: R0 to R12.
  - Low registers: R0 to R7, can be accessed by all instructions.
  - High registers: R8 to R12, can only be accessed by certain instructions.
- Stack pointer (SP): R13, records the current position of the stack.
- Link register (LR): R14, used to store the return address for function calls.
- Program counter (PC): R15, records the address of the current instruction code.
- Special-purpose registers:
  - xPSR: Program status register, contains flags for the current state of the processor.
    - APSR: Application program status register, contains flags for the application.
      - N: Negative flag, set if the result of the last operation was negative.
      - Z: Zero flag, set if the result of the last operation was zero.
      - C: Carry flag, set if there was a carry out from the last operation.
      - V: Overflow flag, set if there was an overflow in the last operation.
    - IPSR: Interrupt program status register, contains the current interrupt number.
      - ISR (Interrupt Service Routine): The address of the current interrupt handler.
    - EPSR: Execution program status register, contains the current execution state.
      - ICT/IT: Indicates whether the processor is in an interrupt or thread mode.

- T: Thumb state, indicates whether the processor is executing in Thumb mode.

1.2 Memory Map

- Core (0.5 GB): 0x00000000 to 0x1FFFFFFF
- SRAM (0.5 GB): 0x20000000 to 0x3FFFFFFF
- Peripherals (0.5 GB): 0x40000000 to 0x5FFFFFFF
- External RAM (1 GB): 0x60000000 to 0x9FFFFFFF
- External Device (1 GB): 0xA0000000 to 0xDFFFFFFF
- Private Peripheral Bus (PPB) - External: 0xE0000000 to 0xE003FFFF
- Private Peripheral Bus (PPB) - Internal: 0xE0040000 to 0xE00FFFFF
- System: 0xE0100000 to 0xFFFFFFFF (PPB+System=0.5 GB)

1.3 Bit-Band Operation

- Bit-band operation: Allow a single load/store operation to access a single bit in the memory.
- Bit-band alias address:
  - SRAM bit-band alias: 0x22000000 to 0x23FFFFFF
  - Peripheral bit-band alias: 0x42000000 to 0x43FFFFFF
- Calculation of bit-band alias address:
  - Bit-band alias address = Base address + (Byte offset $\times$ 32) + (Bit number $\times$ 4)
- Benefits:
  - Faster bit operations
  - Fewer instructions
  - Atomic operation, avoid hazards

Chapter 2

Assembly Language

2.1 Assembly Language Basics

- Format:`label mnemonic operand1, operand2, operand3 ; comment`
  - Label: Identifier for the instruction, used for branching.
  - Mnemonic: The operation to be performed (e.g., MOV, ADD, SUB).
  - Operands: The data to be operated on, can be registers, immediate values, or memory addresses.
  - Comment: Optional, starts with a semicolon, used for documentation.
- Suffixes:
  - S - Suffix for Update APSR (e.g., ADDS, SUBS).
  - B - Suffix for byte operations (e.g., LDRB, STRB).
  - H - Suffix for halfword operations (e.g., LDRH, STRH).
  - D - Suffix for doubleword operations (e.g., LDRD, STRD).
  - SB - Suffix for signed byte operations (e.g., LDRSB).
  - SH - Suffix for signed halfword operations (e.g., LDRSH).
  - Conditional execution: Instructions can be executed conditionally based on the flags in the xPSR register.

Condition	Meaning	Menemonic
EQ	Equal	`BEQ`
NE	Not equal	`BNE`
CS	Carry set (unsigned higher or same)	`BCS`
CC	Carry clear (unsigned lower)	`BCC`
MI	Minus (negative)	`BMI`
PL	Plus (positive or zero)	`BPL`
VS	Overflow set	`BVS`
VC	Overflow clear	`BVC`
HI	Unsigned higher (not lower or same)	`BHI`
LS	Unsigned lower or same	`BLS`
GE	Signed greater than or equal	`BGE`
LT	Signed less than	`BLT`
GT	Signed greater than	`BGT`
LE	Signed less than or equal	`BLE`

Table 2.1: Conditional Execution Mnemonics

2.2 Mnemonic Instructions

2.2.1 Memory access instructions

Load and Store Single Register

Mnemonic	Description	Bit	Syntax
`LDR`	Load register from memory	32	`LDR Rn, [address]`
`LDRB`	Load byte from memory	8	`LDRB Rn, [address]`
`LDRSB`	Load signed byte from memory	8	`LDRSB Rn, [address]`
`LDRH`	Load halfword from memory	16	`LDRH Rn, [address]`
`LDRSH`	Load signed halfword from memory	16	`LDRSH Rn, [address]`
`LDRD`	Load doubleword from memory	64	`LDRD Rn, Rm, [address]`
`STR`	Store register to memory	32	`STR Rn, [address]`
`STRB`	Store byte to memory	8	`STRB Rn, [address]`
`STRH`	Store halfword to memory	16	`STRH Rn, [address]`
`STRD`	Store doubleword to memory	64	`STRD Rn, Rm, [address]`

Table 2.2: Memory Access Instructions

- Address offset:

- Immediate offset:
  - Pre-indexed: `LDR R0, [R1, #4]` (Load from address in R1 + 4)
  - Pre-indexed with update: `LDR R0, [R1, #4]!` (Load from address in R1 + 4 and update R1 to R1 + 4)
  - Post-indexed: `LDR R0, [R1], #4` (Load from address in R1 and then update R1 to R1 + 4)
- Register offset:
  - Pre-indexed: `LDR R0, [R1, R2]` (Load from address in R1 + R2)
  - Pre-indexed with update: `LDR R0, [R1, R2]!` (Load from address in R1 + R2 and update R1 to R1 + R2)
  - Post-indexed: `LDR R0, [R1], R2` (Load from address in R1 and then update R1 to R1 + R2)
  - Register offset: `LDR R0, [R1, R2, LSL #2]` (Load from address in [R1 + (R2 shifted left by 2 bits)])

Load and Store Multiple Registers

Mnemonic	Description	Syntax
`LDMIA`	Register increase 4 after load	`LDMIA Rn!, [addresses]`
`LDMDB`	Register decrease 4 before load	`LDMDB Rn!, [addresses]`
`STRIA`	Register increase 4 after store	`STMIA Rn!, [addresses]`
`STRDB`	Register decrease 4 before store	`STMDB Rn!, [addresses]`
`PUSH`	Push registers onto the stack	`PUSH R0, R1, ...`
`POP`	Pop registers from the stack	`POP R0, R1, ...`

Table 2.3: Load and Store Multiple Registers Instructions

2.2.2 Data Movement

Mnemonic	Description	Bit	Syntax
`ADR`	Load address of label into register	-	`ADR Rn, label`
`MOV(S)`	Move value to register, update APSR if S is present	-	`MOVS Rn, Rm`
`MVN(S)`	Move NOT value to register, update APSR if S is present	-	`MVNS Rn, Rm`
`MOVW`	Move 16-bit immediate value to register	16	`MOVW Rn, #imm`
`MOVT`	Move 16-bit immediate value to upper half of register	16	`MOVT Rn, #imm`

Table 2.4: Data Movement Instructions

- Rm can be a register or an immediate value or a label.

2.2.3 Arithmetic Operations

Mnemonic	Description	Syntax
ADD(S)	Add two registers, update APSR if S is present	ADDS R0, Rn, Rm
ADC(S)	Add with carry, update APSR if S is present	ADCS R0, Rn, Rm
SUB(S)	Subtract two registers, update APSR if S is present	SUBS R0, Rn, Rm
SBC(S)	Subtract with carry, update APSR if S is present	SBCS R0, Rn, Rm
RSB(S)	Reverse subtract, update APSR if S is present	RSBS R0, Rn, Rm
MUL(S)	Multiply two registers, update APSR if S is present	MULS R0, Rn, Rm
MLA	Multiply and accumulate	MLA R0, Rn, Rm, Ra
MLS	Multiply and subtract	MLS R0, Rn, Rm, Ra
SMMUL(R)	MUL take upper 32 bits, add 2^{32} if R is present	SMMUL R0, Rn, Rm
SMMLA(R)	MLA take upper 32 bits, add 2^{32} if R is present	SMMLA R0, Rn, Rm, Ra
SMMLS(R)	MLS take upper 32 bits, add 2^{32} if R is present	SMMLS R0, Rn, Rm, Ra
UMULL	Unsigned multiply long, result in R0:R1	UMULL R0, R1, Rn, Rm
UMLAL	Unsigned multiply accumulate long, result in R0:R1	UMLAL R0, R1, Rn, Rm
SMULL	Signed multiply long, result in R0:R1	SMULL R0, R1, Rn, Rm
SMLAL	Signed multiply accumulate long, result in R0:R1	SMLAL R0, R1, Rn, Rm
SDIV	Signed divide, result in R0	SDIV R0, Rn, Rm
UDIV	Unsigned divide, result in R0	UDIV R0, Rn, Rm
SSAT	Saturates a signed value to the signed range	SSAT R0, n, Rm
USAT	Saturates a signed value to the unsigned range	SSAT R0, n, Rm
QADD	ADD take extreme value when result exceeds range	QADD R0, Rn, Rm
QSUB	SUB take extreme value when result exceeds range	QSUB R0, Rn, Rm
SXTB	Signed extend byte to word	SXTB R0, Rm
UXTB	Unsigned extend byte to word	UXTB R0, Rm
SXTH	Signed extend half word to word	SXTH R0, Rm
UXTH	Unsigned extend half word to word	UXTH R0, Rm
ASR(S)	Arithmetic Shift Right	ASRS R0, Rn, Rm
LSL(S)	Logical Shift Left	LSLS R0, Rn, Rm
LSR(S)	Logical Shift Right	LSRS R0, Rn, Rm
ROR(S)	Rotate Right	RORS R0, Rn, Rm
RRX(S)	Rotate Right with Extend	RRXS R0, Rn
	All Rm can be a register or an immediate value	

Table 2.5: Arithmetic Operations Instructions

- `ADDS R0, R1, R2`: R0 = R1 + R2, APSR updated, C = 1 if there is a carry, C = 0 if no carry.

- `SUBS R0, R1, R2`: R0 = R1 - R2, APSR updated, C = 1 if R1 >= R2, C = 0 if R1 < R2.

- `RSBS R0, R1, R2`: R0 = R2 - R1, APSR updated.

- `MLA R0, R1, R2, R3`: R0 = R3 + R1 × R2.

- `MLS R0, R1, R2, R3`: R0 = R3 − R1 × R2.

- `SMMXX`: Result take upper 32 bits (Result is 64 bits)

- `SMMXXR`: Result add 0x80000000 then take upper 32 bits (Result is 64 bits)

- Min and Max for `XSAT` and `UXXX`:

  - `SSAT`: $-2^{n-1} \leqslant x \leqslant 2^{n-1} - 1$
  - `USAT`: $0 \leqslant x \leqslant 2^{n} - 1$
  - `UADD` and `USUB`: $-2^{31} \leqslant x \leqslant 2^{31} - 1$, Q = 1 if saturates

- Shift code (Can be used as an operation instruction in Rm):

  - `LSL #3`: 0b00000001 << 3 = 0b00001000
  - `LSR #3`: 0b00001000 >> 3 = 0b00000001
  - `ASR #3`: 0b11100000 >> 3 = 0b11111000
  - `ROR #3`: 0b10100101 >> 3 = 0b10110100
  - `RRX`: 0b11010100 (C = 1) >> 1 = 0b11101010 (C = 0)

2.2.4 Logic Operations

Mnemonic	Description	Syntax
`AND(S)`	Logical AND, update APSR if S is present	`ANDS R0, Rn, Rm`
`ORR(S)`	Logical OR, update APSR if S is present	`ORRS R0, Rn, Rm`
`EOR(S)`	Logical XOR, update APSR if S is present	`EORS R0, Rn, Rm`
`BIC(S)`	Bit clear, update APSR if S is present	`BICS R0, Rn, Rm`
`ORN(S)`	Bit clear OR, update APSR if S is present	`ORNS R0, Rn, Rm`
`MVN(S)`	Move NOT value to register, update APSR if S is present	`MVNS R0, Rn`
`BFC`	Bit field clear	`BFC Rn, m, n`
`BFI`	Bit field insert	`BFI Rn, Rm, m, n`
`SBFX`	Signed bit field extract	`SBFX R0, Rn, m, n`
`UBFX`	Unsigned bit field extract	`UBFX R0, Rn, m, n`

Table 2.6: Logic Operations Instructions

- For bit field operations:
  - `BFC Rn, m, n`: Clear bits from m to m+n in Rn.
  - `BFI Rn, Rm, m, n`: Insert bits from Rm into Rn starting at bit m and ending at bit m+n.
  - `SBFX R0, Rn, m, n`: Extract signed bits from Rn starting at bit m and ending at bit m+n into R0.
  - `UBFX R0, Rn, m, n`: Extract unsigned bits from Rn starting at bit m and ending at bit m+n into R0.

2.2.5 Conditional Instructions

Mnemonic	Description	Syntax
`BL`	Jump to label (Store the return address to LR)	`BL label`
`BLX`	Jump to Rn (Store the return address to LR)	`BLX Rn`
`BX`	Jump to Rn	`BX Rn`
`B`	Jump to label	`B label`
`CMP`	Compare (Rn−Rm) and update ASPR	`CMP Rn, Rm`
`CMN`	Rn+Rm and update ASPR	`CMN Rn, Rm`
`TST`	Rn AND Rm and update ASPR	`TST Rn, Rm`
`TEQ`	Rn EOR Rm and update ASPR	`TEQ Rn, Rm`

Table 2.7: Conditional Instructions

Condition code	Meaning	Requirement
`EQ`	Equal (`==`)	Z = 1
`NE`	Not equal (`!=`)	Z = 0
`HS`	Unsigned higher or same (`>=`)	C = 1
`LO`	Unsigned lower (`<`)	C = 0
`HI`	Unsigned higher (`>`)	C = 1 && Z = 0
`LS`	Unsigned lower or same (`<=`)	C = 0 || Z = 1
`GE`	Signed greater than or equal (`>=`)	N = V
`LT`	Signed less than (`<`)	N ≠ V
`GT`	Signed greater than (`>`)	N = V && Z = 0
`LE`	Signed less than or equal (`<=`)	N ≠ V || Z = 1
`CS`	Carry set (`>=`)	C = 1
`CC`	Carry clear (`<`)	C = 0
`MI`	Minus (negative) (`<0`)	N = 1
`PL`	Plus (positive or zero) (`>=0`)	N = 0
`VS`	Overflow set (`overflow`)	V = 1
`VC`	Overflow clear (`no overflow`)	V = 0
`AL`	Always (default)	-

Table 2.8: Condition Codes for Conditional Instructions

2.3 ASM and C Interfacing

- R0 to R3: Used for passing arguments to functions, need not be saved in stack.
  - R0: First argument / result / scratch register 1
  - R1: Second argument / result / scratch register 2
  - R2: Third argument / scratch register 3
  - R3: Fourth argument / scratch register 4
- R4 to R11: Used for local variables and temporary storage, must be stored in stack if used in a function.

Chapter 3

General Purpose IO

3.1 GPIO Register Map

- GPIOx_MODER: 32-bit, configures each bit as input or output or other
- GPIOx_OTYPER: 32-bit, output type configuration (push-pull or open-drain)
- GPIOx_OSPEEDR: 32-bit, configures the maximum frequency of an output pin
- GPIOx_PUPDR: 32-bit, configures the internal pull-up or pull-down register
- GPIOx_IDR: 32-bit, the input data register
- GPIOx_ODR: 32-bit, the output data register
- GPIOx_BSRR: 32-bit, the bit set/reset register
- GPIOx_LCKR: 32-bit, the bit lock register
- GPIOx_AFRH: 32-bit, higher bits for alternate function selection register
- GPIOx_AFRL: 32-bit, lower bits for alternate function selection register
- RCC Clock enable:
  - 0: GPIOAEN
  - 1: GPIOBEN
  - 2: GPIOCEN
  - 3: GPIODEN
  - 4: GPIOEEN
  - 7: GPIOHEN

- MODER: Total 16 (0-15), each takes 2 bits
  - `00`: Input (reset state)
  - `01`: General purpose output mode
  - `10`: Alternate function mode
  - `11`: Analog mode
- OTYPER: Total 16, each takes 1 bit (reserved 16 bit)
  - `0`: Output push-pull (reset state)
  - `1`: Output open-drain
- OSPEEDR: Total 16, each takes 2 bits
  - `00`: Low speed
  - `01`: Medium speed
  - `10`: High speed
  - `11`: Very high speed
  - Slew rate $= \max\left(\frac{\Delta V}{\Delta t}\right)$
- PUPDR: Total 16, each takes 2 bits
  - `00`: No pull-up, pull-down
  - `01`: Pull-up
  - `10`: Pull-down
  - `11`: Reserved
- IDR/ODR: Total 16, each takes 1 bit

3.2 GPIO in ASM

- Boundary address:
  - RCC: `0x40023800 - 0x40023BFF`
    - RCCGPIOxEN: `0x40023830` 0-4, 7
  - GPIOA: `0x40020000 - 0x400203FF`
  - GPIOB: `0x40020400 - 0x400207FF`

  - GPIOC: `0x40020800 - 0x40020BFF`
  - GPIOD: `0x40020C00 - 0x40020FFF`
  - GPIOE: `0x40021000 - 0x400213FF`
  - GPIOH: `0x40021C00 - 0x40021FFF`

- Offset of GPIOx:
  - MODER: `0x00`
  - OTYPER: `0x04`
  - OSPEEDR: `0x08`
  - PUPDR: `0x0C`
  - IDR: `0x10`
  - ODR: `0x14`
  - BSRR: `0x18`
  - LCKR: `0x1C`
  - AFRL: `0x20`
  - AFRH: `0x24`

- Direct Addressing Example:

```
    .syntax unified                 ; Use ASM unified assembly syntax
    .cpu cortex-m4
    .section .text
    .global main
RCC_BASE        = 0x40023800        ; Base address of RCC
GPIOA_BASE      = 0x40020000
RCC_AHB1ENR     = RCC_BASE + 0x30
GPIOA_MODER     = GPIOA_BASE + 0x00
GPIOA_OSPEEDR   = GPIOA_BASE + 0x08
GPIOA_OTYPER    = GPIOA_BASE + 0x04
GPIOA_ODR       = GPIOA_BASE + 0x14

main:
        LDR R1, =RCC_AHB1ENR
        LDR R2, [R1]
        ORR R2, R2, 0x01            ; Set GPIOAEN to 1 (bit 0)
        STR R2, [R1]
```

```
        LDR R1, =GPIOA_MODER
        LDR R2, [R1]
        BIC R2, R2, #0x03 << 10     ; Clear mode bits for PA5
        ORR R2, R2, #0x01 << 10     ; Set PA5 output mode (01)
        STR R2, [R1]
```

- Immediate Offset Example

```
    .syntax unified                 ; Use ASM unified assembly syntax
    .cpu cortex-m4
    .section .text
    .global main
RCC_BASE        = 0x40023800        ; Base address of RCC
GPIOA_BASE      = 0x40020000
RCC_AHB1ENR     = 0x30
GPIOA_MODER     = 0x00
GPIOA_OSPEEDR   = 0x08
GPIOA_OTYPER    = 0x04
GPIOA_ODR       = 0x14

main:
        LDR R1, =RCC_BASE
        LDR R2, [R1, #RCC_AHB1ENR]  ;
        ORR R2, R2, 0x01            ; Set GPIOAEN to 1 (bit 0)
        STR R2, [R1, #RCC_AHB1ENR]  ;

        LDR R1, =GPIOA_BASE
        LDR R2, [R1, #GPIOA_MODER]  ;
        BIC R2, R2, #0x03 << 10     ; Clear mode bits for PA5
        ORR R2, R2, #0x01 << 10     ; Set PA5 output mode (01)
        STR R2, [R1, #GPIOA_MODER]  ;
```

Chapter 4

General Purpose Timer

4.1 Base-Time Unit

- ARR: Auto-reload register (16-bit for TIM3, TIM4; 32-bit for TIM2, TIM5)
- PSC: Programmable prescaler
- CK_INT: Internal CLK (Default is 84 MHz)
- CK_CNT: $\frac{\texttt{CK\_INT}}{\texttt{PSC}+1}$
- Frequency: $\frac{\texttt{CK\_CNT}}{(\texttt{ARR}+1)(\texttt{RCR}+1)}(\texttt{RCR}=0)$
- Counter Mode:
  - Up: Counts up from 0 to ARR, then reset to 0 when ARR is reached
  - Down: Counts down from ARR to 0, then reset to ARR when 0 is reached
  - Center Aligned mode 1
  - Center Aligned mode 2
  - Center Aligned mode 3
- Preload: Only change ARR when counter reset if enable.

4.2 Output PWM Mode

4.2.1 Standard PWM Active High

- $\texttt{OCREF} = \begin{cases} 1, & \texttt{CNT} < \texttt{CCR} \\ 0, & \texttt{CNT} \geqslant \texttt{CCR} \end{cases}$

4.2.2 Inverted PWM Active Low

- $\texttt{OCREF} = \begin{cases} 0, & \texttt{CNT} < \texttt{CCR} \\ 1, & \texttt{CNT} \geqslant \texttt{CCR} \end{cases}$

4.3 Key Configuration Register

4.3.1 Offset

- `CR1` (Control Register 1): `0x00`
- `CCER` (Capture/Compare Enable Register): `0x20`
- `CNT`: `0x24`
- `PSC`: `0x28`
- `ARR`: `0x2C`
- `CCR1`: `0x34`
- `CCR2`: `0x38`
- `CCR3`: `0x3C`
- `CCR4`: `0x40`

4.3.2 Configuration

- `CR1`:
  - Bit 0: `CEN` Counter Enable
    - `0`: Disable
    - `1`: Enable
  - Bit 4: `DIR`
    - `0`: Upcounter
    - `1`: Downcounter
  - Bit 6:5: `CMS`
    - `00`: Edge-aligned mode
    - `01`: Center Aligned mode 1
    - `10`: Center Aligned mode 2

        * `11`: Center Aligned mode 3

- `CCER`:
    - Bit $4x$, $x = 0, 1, 2, 3$: `CC`$(x+1)$`E`
        * Output
            · `0`: Off
            · `1`: On
        * Input
            · `0`: Capture Disable
            · `1`: Capture Enable
    - Bit $4x+1$:$4x+3$, $x = 0, 1, 2, 3$: `CC`$(x+1)$`P, Reserved, CC`$(x+1)$`NP`
        * Output
            · `000`: OC1 active high
            · `001`: OC1 active low
        * Input
            · `000`: Rising Edge
            · `001`: Falling Edge
            · `101`: Both Edge

Chapter 5

Analogue to Digital Conversion

- ADC CLK: $t_{clk} = \dfrac{1}{f_{clk}}$, $f_{clk} = 21$ MHz usually (Max 30 MHz)
- Sample and hold sample time: $t_s = s_s \times t_{clk}$
- Conversion time: $t_c = (s_c + 3) \times t_{clk}$
- "Total" sampling time: $t_{total} = t_s + t_c$
- Minimum sampling time: $t_{min} = 5 \times R \times C$, where R is the input resistance and C is the input capacitance.
- Quantization: $\Delta V = \dfrac{V_{ref}}{2^n}$, where n is the number of bits (usually 12 bits, resolution bit)
- Quantization error: $e = \dfrac{\Delta V}{2}$

Chapter 6

Universal Synchronous Asynchronous Receiver Transmitter

6.1 Data Packet

- Typical 8N1: 8 data bits, no parity, 1 stop bit
- Other common configurations:
  - 8E1: 8 data bits, even parity, 1 stop bit
  - 8O1: 8 data bits, odd parity, 1 stop bit
- TX is low during the start bit, high during the stop bit.
- Stop bit can be 0.5, 1, 1.5, or 2 bits long.

6.2 Oversampling

- Oversampling: To improve the accuracy of the received data by sampling multiple times per bit.
- Oversampling by 8: Samples 8 times per bit, sampled value is 4, 5, 6.
- Oversampling by 16: Samples 16 times per bit, sampled value is 8, 9, 10.
- To improve the accuracy of the received data and to mitigate the noise.

6.3 Baud Rate

- Baud rate: The number of signal changes per second.

$$\text{Baud Rate} = \frac{f_{clk}}{8 \times (2 - \text{OVER8} \times \text{USARTDIV})}$$

- f_{clk}: Clock frequency (usually PCLK2, 42 MHz).
- OVER8: 0 for oversampling by 8, 1 for oversampling by 16.
- USARTDIV: An unsigned fixed-point number that is coded on the `USART_BRR` register

$$\text{Desire: USARTDIV} = \frac{f_{clk}}{8 \times (2 - \text{OVER8} \times \text{Baud Rate})}$$

$$\text{Actual: USARTDIV} = \text{OVER8} \times 16 + \frac{\text{DIV\_Fraction}}{16}$$

- DIV_Fraction: The fractional part of the USARTDIV. Usually calculated as:

$$\text{DIV\_Fraction} = \lceil \text{USARTDIV\_Fratinal\_Part} \times 16 \rceil$$

- Error: The error in baud rate can be calculated as:

$$\text{Error} = \left| \frac{\text{Actual Baud Rate} - \text{Desired Baud Rate}}{\text{Desired Baud Rate}} \right| \times 100\%$$

- Transmission rate: $\frac{\text{Baud Rate}}{\text{Number of bits}}$.